Classification : 2 Class

1.환경준비

(1) 라이브러리 로딩

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

from sklearn.model_selection import train_test_split
from sklearn.metrics import *
from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential
from keras.layers import Dense
from keras.backend import clear_session
from keras.optimizers import Adam
```

- 학습곡선 함수

```
def dl_history_plot(history):
    plt.figure(figsize=(10,6))
    plt.plot(history['loss'], label='train_err', marker = '.')
    plt.plot(history['val_loss'], label='val_err', marker = '.')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid()
    plt.show()
```

(2) 데이터로딩

```
path = "https://raw.githubusercontent.com/DA4BAM/dataset/master/titanic.3.csv"
data = pd.read_csv(path)
data.drop(['Age_scale1', 'AgeGroup', 'SibSp','Parch' ], axis = 1, inplace = True)
data.head()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Family
0	0	3	male	22.0	7.2500	S	2
1	1	1	female	38.0	71.2833	C	2
2	1	3	female	26.0	7.9250	S	1
3	1	1	female	35.0	53.1000	S	2
4	0	3	male	35.0	8.0500	S	1

Next steps: Generate code with data View recommended plots

2.데이터 준비

Sex, Age, Fare 만 이용하여 Survived 를 예측하는 모델을 만들어 봅시다.

(1) 데이터 준비

```
target = 'Survived'
features = ['Sex', 'Age', 'Fare']
x = data.loc[:, features]
y = data.loc[:, target]
```

(2) 가변수화

```
x = pd.get_dummies(x, columns = ['Sex'], drop_first = True)
x.head()
```

	Age	Fare	Sex_male
0	22.0	7.2500	True
1	38.0	71.2833	False
2	26.0	7.9250	False
3	35.0	53.1000	False
4	35.0	8.0500	True

Next steps: Generate code with x View recommended plots

(3) 데이터분할

```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=.3, random_state = 20)
```

(4) Scaling

```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
```

3.딥러닝1: 3개 feature

(1) 모델설계

```
nfeatures = x_train.shape[1]
nfeatures
```

```
3
```

```
# 메모리 정리
clear_session()

# Sequential 모델 만들기
model = Sequential( Dense( 1 , input_shape = (nfeatures ,), activation= 'sigmoid') )

# 모델요약
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 4

=================================================================
Total params: 4 (16.00 Byte)
Trainable params: 4 (16.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

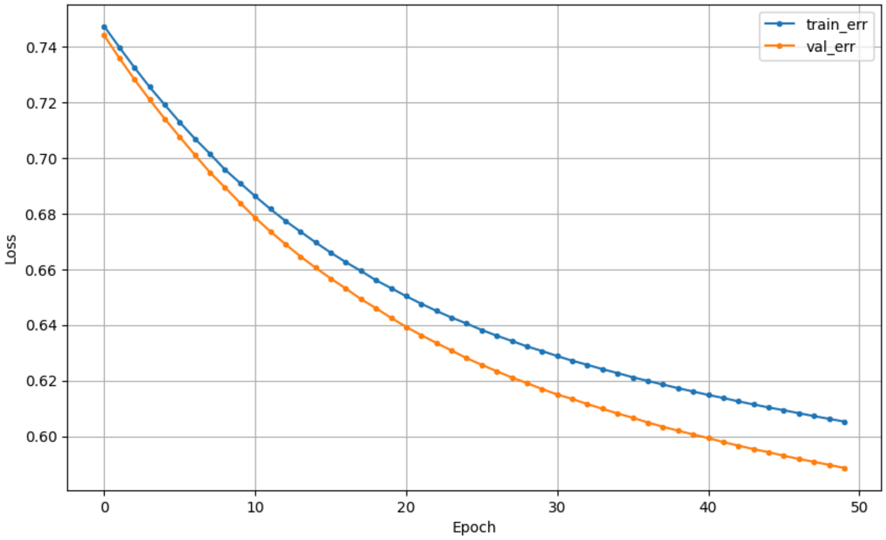
(2) compile + 학습

```
model.compile(optimizer = Adam(lr=0.01), loss = 'binary_crossentropy')

history = model.fit(x_train, y_train,
                    epochs = 50, validation_split=0.2).history
```

```
Epoch 22/50
16/16 [==============================] - 0s 8ms/step - loss: 0.6476 - val_loss: 0.6363
Epoch 23/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6450 - val_loss: 0.6335
Epoch 24/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6426 - val_loss: 0.6308
Epoch 25/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6405 - val_loss: 0.6280
Epoch 26/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6382 - val_loss: 0.6257
Epoch 27/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6361 - val_loss: 0.6233
Epoch 28/50
16/16 [==============================] - 0s 11ms/step - loss: 0.6342 - val_loss: 0.6211
Epoch 29/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6322 - val_loss: 0.6191
Epoch 30/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6306 - val_loss: 0.6169
Epoch 31/50
16/16 [==============================] - 0s 8ms/step - loss: 0.6288 - val_loss: 0.6150
Epoch 32/50
16/16 [==============================] - 0s 11ms/step - loss: 0.6271 - val_loss: 0.6134
Epoch 33/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6256 - val_loss: 0.6115
Epoch 34/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6241 - val_loss: 0.6098
Epoch 35/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6227 - val_loss: 0.6081
Epoch 36/50
16/16 [==============================] - 0s 8ms/step - loss: 0.6212 - val_loss: 0.6066
Epoch 37/50
16/16 [==============================] - 0s 5ms/step - loss: 0.6199 - val_loss: 0.6049
Epoch 38/50
16/16 [==============================] - 0s 13ms/step - loss: 0.6185 - val_loss: 0.6034
Epoch 39/50
16/16 [==============================] - 0s 15ms/step - loss: 0.6173 - val_loss: 0.6020
Epoch 40/50
16/16 [==============================] - 0s 8ms/step - loss: 0.6160 - val_loss: 0.6006
Epoch 41/50
16/16 [==============================] - 0s 8ms/step - loss: 0.6148 - val_loss: 0.5993
Epoch 42/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6137 - val_loss: 0.5979
Epoch 43/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6125 - val_loss: 0.5966
Epoch 44/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6114 - val_loss: 0.5953
Epoch 45/50
16/16 [==============================] - 0s 8ms/step - loss: 0.6103 - val_loss: 0.5942
Epoch 46/50
16/16 [==============================] - 0s 7ms/step - loss: 0.6093 - val_loss: 0.5930
Epoch 47/50
16/16 [==============================] - 0s 7ms/step - loss: 0.6082 - val_loss: 0.5918
Epoch 48/50
16/16 [==============================] - 0s 10ms/step - loss: 0.6072 - val_loss: 0.5908
Epoch 49/50
16/16 [==============================] - 0s 5ms/step - loss: 0.6062 - val_loss: 0.5897
Epoch 50/50
16/16 [==============================] - 0s 9ms/step - loss: 0.6052 - val_loss: 0.5886
```

- 학습결과 그래프

```
dl_history_plot(history)
```



(3) 예측 및 검증

```
pred = model.predict(x_val)

# activation이 sigmoid --> 0 ~ 1 사이의 확률값.
# 그러므로 cut-off value(보통 0.5)를 기준으로 잘라서 0과 1로 만들어 준다.
pred = np.where(pred >= .5, 1, 0)
```

```
    9/9 [==============================] - 0s 3ms/step
```

```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred))
```

```
    [[166   4]
     [ 68  30]]
                  precision    recall  f1-score   support

               0       0.71      0.98      0.82       170
               1       0.88      0.31      0.45        98

        accuracy                           0.73       268
       macro avg       0.80      0.64      0.64       268
    weighted avg       0.77      0.73      0.69       268
```

4.딥러닝2 : 전체 feature

- 이제 전체 데이터를 가지고 모델링을 시도해 보겠습니다.

(1) 데이터 전처리

- 데이터 준비

```
target = 'Survived'
x = data.drop(target, axis = 1)
y = data.loc[:, target]
```

- 가변수화

```
cat_cols = ['Pclass','Sex', 'Embarked']
x = pd.get_dummies(x, columns = cat_cols, drop_first = True)
```

- 데이터분할

```
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=.3, random_state = 20)
```

- 스케일링

```
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
```

(2) 모델링

- 모델 설계

```
n = x_train.shape[1]
n
```

```
8
```

```
# 메모리 정리
clear_session()

# Sequential 모델
model = Sequential( Dense( 1, input_shape = (n, ), activation = 'sigmoid'))

# 모델요약
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 1)                 9         
                                                                 
=================================================================
Total params: 9 (36.00 Byte)
Trainable params: 9 (36.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

- compile + 학습

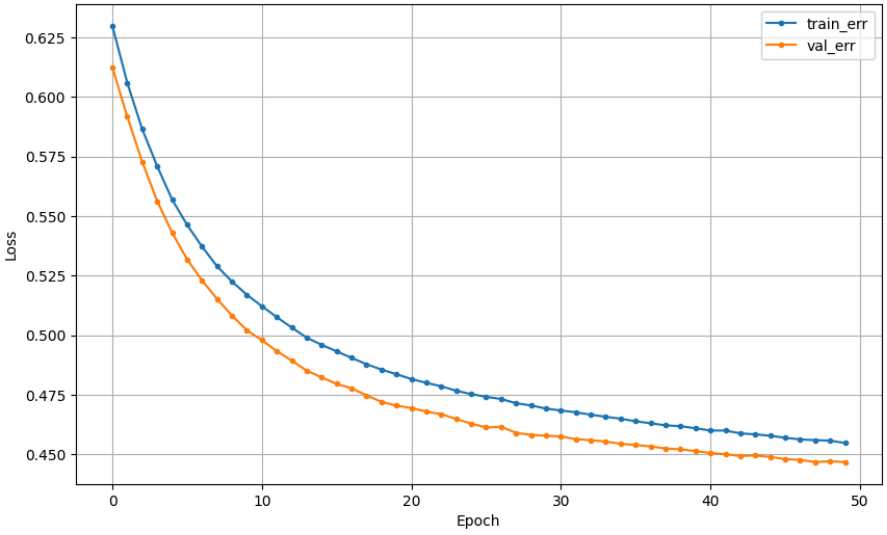
```
model.compile(optimizer = Adam(learning_rate = 0.01), loss = 'binary_crossentropy')
history = model.fit(x_train, y_train, epochs = 50, validation_split=.2).history
```

```
Epoch 1/50
16/16 [==============================] - 1s 18ms/step - loss: 0.6299 - val_loss: 0.6125
Epoch 2/50
16/16 [==============================] - 0s 6ms/step - loss: 0.6059 - val_loss: 0.5917
Epoch 3/50
16/16 [==============================] - 0s 5ms/step - loss: 0.5865 - val_loss: 0.5728
Epoch 4/50
16/16 [==============================] - 0s 6ms/step - loss: 0.5709 - val_loss: 0.5563
Epoch 5/50
16/16 [==============================] - 0s 5ms/step - loss: 0.5569 - val_loss: 0.5430
Epoch 6/50
16/16 [==============================] - 0s 5ms/step - loss: 0.5463 - val_loss: 0.5319
Epoch 7/50
16/16 [==============================] - 0s 6ms/step - loss: 0.5372 - val_loss: 0.5230
Epoch 8/50
16/16 [==============================] - 0s 6ms/step - loss: 0.5290 - val_loss: 0.5153
Epoch 9/50
16/16 [==============================] - 0s 6ms/step - loss: 0.5226 - val_loss: 0.5083
Epoch 10/50
16/16 [==============================] - 0s 5ms/step - loss: 0.5170 - val_loss: 0.5021
Epoch 11/50
16/16 [==============================] - 0s 5ms/step - loss: 0.5122 - val_loss: 0.4979
Epoch 12/50
```

```
16/16 [==============================] - 0s 6ms/step - loss: 0.5076 - val_loss: 0.4934
Epoch 13/50
16/16 [==============================] - 0s 5ms/step - loss: 0.5032 - val_loss: 0.4893
Epoch 14/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4990 - val_loss: 0.4852
Epoch 15/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4960 - val_loss: 0.4823
Epoch 16/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4933 - val_loss: 0.4796
Epoch 17/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4905 - val_loss: 0.4778
Epoch 18/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4879 - val_loss: 0.4748
Epoch 19/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4856 - val_loss: 0.4721
Epoch 20/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4837 - val_loss: 0.4705
Epoch 21/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4816 - val_loss: 0.4695
Epoch 22/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4801 - val_loss: 0.4680
Epoch 23/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4786 - val_loss: 0.4669
Epoch 24/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4767 - val_loss: 0.4648
Epoch 25/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4753 - val_loss: 0.4630
Epoch 26/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4742 - val_loss: 0.4613
Epoch 27/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4733 - val_loss: 0.4616
Epoch 28/50
16/16 [==============================] - 0s 3ms/step - loss: 0.4715 - val_loss: 0.4591
Epoch 29/50
```

- 학습결과 그래프

```
dl_history_plot(history)
```



- 예측 및 검증

```
pred2 = model.predict(x_val)
pred2 = np.where( pred2 >= .5 , 1, 0)
print(classification_report(y_val, pred2))
```

```
9/9 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       0.82      0.83      0.82       170
           1       0.70      0.68      0.69        98

    accuracy                           0.78       268
   macro avg       0.76      0.76      0.76       268
weighted avg       0.78      0.78      0.78       268
```

5.딥러닝3 : hidden layer

- 이제 레이어를 추가해 보겠습니다.

(1) 모델 설계

```
n = x_train.shape[1]
n
```

```
8
```

```
# 메모리 정리
clear_session()

# Sequential 모델
model3 = Sequential([ Dense( 4, input_shape = (n ,), activation = 'relu'),
                      Dense( 1, activation = 'sigmoid')])

# 모델요약
model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 4)                 36        
                                                                 
 dense_1 (Dense)             (None, 1)                 5         
                                                                 
=================================================================
Total params: 41 (164.00 Byte)
Trainable params: 41 (164.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

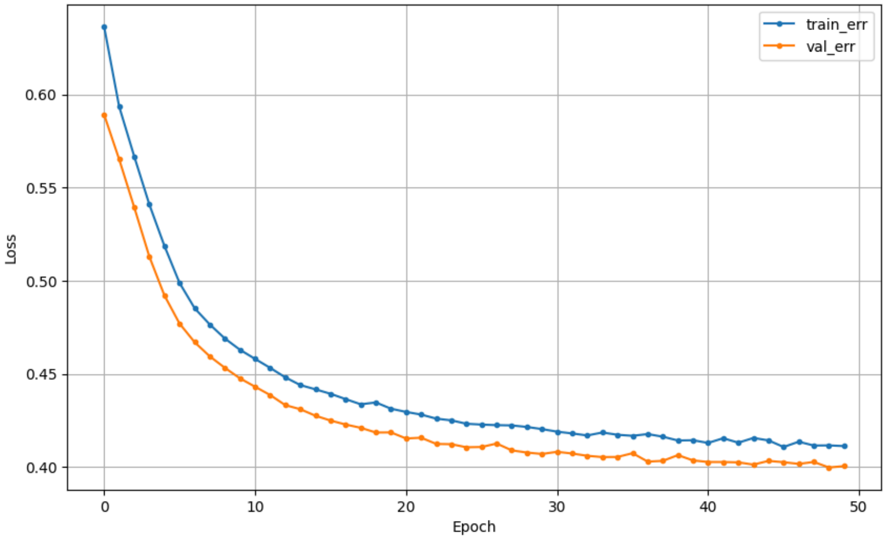
(2) compile + 학습

```
model3.compile( optimizer=Adam(learning_rate= 0.01), loss ='binary_crossentropy')
hist = model3.fit(x_train, y_train, epochs = 50, validation_split=.2 ).history
```

```
    Epoch 34/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4185 - val_loss: 0.4053
    Epoch 35/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4172 - val_loss: 0.4053
    Epoch 36/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4167 - val_loss: 0.4074
    Epoch 37/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4176 - val_loss: 0.4028
    Epoch 38/50
    16/16 [==============================] - 0s 3ms/step - loss: 0.4162 - val_loss: 0.4032
    Epoch 39/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4142 - val_loss: 0.4063
    Epoch 40/50
    16/16 [==============================] - 0s 3ms/step - loss: 0.4143 - val_loss: 0.4035
    Epoch 41/50
    16/16 [==============================] - 0s 5ms/step - loss: 0.4129 - val_loss: 0.4026
    Epoch 42/50
    16/16 [==============================] - 0s 3ms/step - loss: 0.4154 - val_loss: 0.4026
    Epoch 43/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4130 - val_loss: 0.4024
    Epoch 44/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4155 - val_loss: 0.4012
    Epoch 45/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4142 - val_loss: 0.4032
    Epoch 46/50
    16/16 [==============================] - 0s 3ms/step - loss: 0.4107 - val_loss: 0.4025
    Epoch 47/50
    16/16 [==============================] - 0s 3ms/step - loss: 0.4135 - val_loss: 0.4016
    Epoch 48/50
    16/16 [==============================] - 0s 4ms/step - loss: 0.4115 - val_loss: 0.4027
    Epoch 49/50
    16/16 [==============================] - 0s 5ms/step - loss: 0.4115 - val_loss: 0.3997
    Epoch 50/50
    16/16 [==============================] - 0s 5ms/step - loss: 0.4112 - val_loss: 0.4004
```

- 학습결과 그래프

```
dl_history_plot(hist)
```



- 예측 및 검증

```
pred3 = model3.predict(x_val)
pred3 = np.where(pred3 >= 0.5, 1, 0)
```

```
    9/9 [==============================] - 0s 2ms/step
```

```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred3))
```

```
[[166   4]
 [ 68  30]]
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       170
           1       0.76      0.59      0.67        98

    accuracy                           0.78       268
   macro avg       0.78      0.74      0.75       268
weighted avg       0.78      0.78      0.78       268
```

(3) 실습1

- 다음의 summary를 보고 모델을 설계하시오.

Layer (type)	Output Shape	옵션
dense (Dense)	(None, 16)	node, input_shape, activation = 'relu'
dense_1 (Dense)	(None, 1)	node, activation = 'sigmoid'

```
n
```

```
8
```

```
clear_session()
model1 = Sequential([Dense(16, input_shape=(n, ), activation='relu'),
                     Dense(1, activation='sigmoid')])
model1.summary()
```

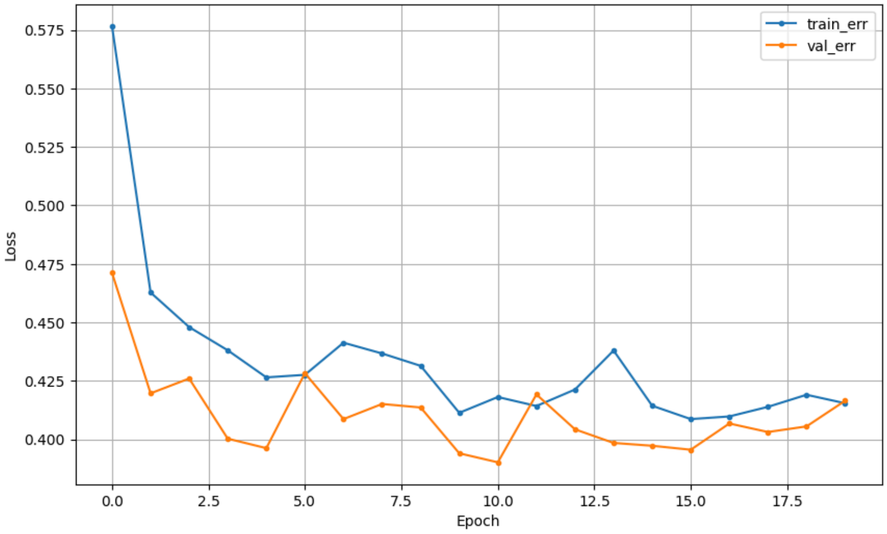
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 16)                144

 dense_1 (Dense)             (None, 1)                 17

=================================================================
Total params: 161 (644.00 Byte)
Trainable params: 161 (644.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model1.compile(optimizer=Adam(0.1), loss='binary_crossentropy')
hist = model1.fit(x_train, y_train, epochs=20, validation_split=0.2).history
```

```
Epoch 1/20
16/16 [==============================] - 1s 22ms/step - loss: 0.5767 - val_loss: 0.4712
Epoch 2/20
16/16 [==============================] - 0s 7ms/step - loss: 0.4628 - val_loss: 0.4197
Epoch 3/20
16/16 [==============================] - 0s 8ms/step - loss: 0.4480 - val_loss: 0.4260
Epoch 4/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4381 - val_loss: 0.4003
Epoch 5/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4265 - val_loss: 0.3962
Epoch 6/20
16/16 [==============================] - 0s 5ms/step - loss: 0.4276 - val_loss: 0.4283
Epoch 7/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4413 - val_loss: 0.4087
Epoch 8/20
16/16 [==============================] - 0s 7ms/step - loss: 0.4367 - val_loss: 0.4151
Epoch 9/20
16/16 [==============================] - 0s 7ms/step - loss: 0.4314 - val_loss: 0.4136
Epoch 10/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4113 - val_loss: 0.3940
Epoch 11/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4181 - val_loss: 0.3902
Epoch 12/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4142 - val_loss: 0.4192
Epoch 13/20
16/16 [==============================] - 0s 6ms/step - loss: 0.4213 - val_loss: 0.4043
Epoch 14/20
16/16 [==============================] - 0s 7ms/step - loss: 0.4380 - val_loss: 0.3985
Epoch 15/20
16/16 [==============================] - 0s 5ms/step - loss: 0.4144 - val_loss: 0.3973
Epoch 16/20
```

```
    16/16 [==============================] - 0s 5ms/step - loss: 0.4087 - val_loss: 0.3956
    Epoch 17/20
    16/16 [==============================] - 0s 6ms/step - loss: 0.4098 - val_loss: 0.4068
    Epoch 18/20
    16/16 [==============================] - 0s 7ms/step - loss: 0.4139 - val_loss: 0.4031
    Epoch 19/20
    16/16 [==============================] - 0s 6ms/step - loss: 0.4191 - val_loss: 0.4055
    Epoch 20/20
    16/16 [==============================] - 0s 7ms/step - loss: 0.4154 - val_loss: 0.4165
```

```
dl_history_plot(hist)
```



```
pred = model1.predict(x_val)
pred = np.where(pred >= 0.5, 1, 0)
```

```
    9/9 [==============================] - 0s 3ms/step
```

```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred))
```

```
    [[148  22]
     [ 35  63]]
                  precision    recall  f1-score   support

               0       0.81      0.87      0.84       170
               1       0.74      0.64      0.69        98

        accuracy                           0.79       268
       macro avg       0.77      0.76      0.76       268
    weighted avg       0.78      0.79      0.78       268
```

(4) 실습2

- 다음의 summary를 보고 모델을 설계하시오.

Layer (type)	Output Shape	옵션
dense (Dense)	(None, 16)	node, input_shape, activation = 'relu'
dense_1 (Dense)	(None, 8)	node, activation = 'relu'
dense_2 (Dense)	(None, 1)	node, activation = 'sigmoid'

```
clear_session()
model2 = Sequential([Dense(16, input_shape=(n, ), activation='relu'),
                     Dense(8, activation='relu'),
                     Dense(1, activation='sigmoid')])
model2.summary()
```

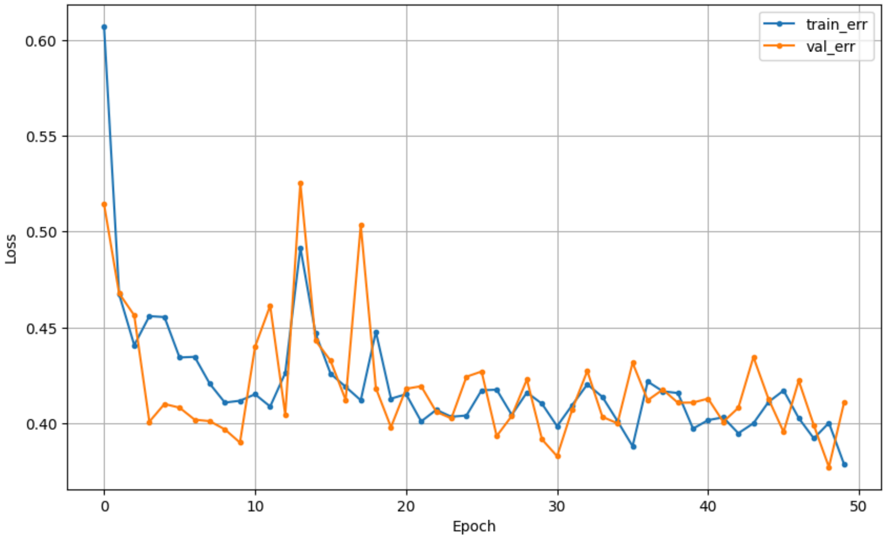
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 16)                144       
                                                                 
 dense_1 (Dense)             (None, 8)                 136       
                                                                 
 dense_2 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 289 (1.13 KB)
Trainable params: 289 (1.13 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model2.compile(optimizer=Adam(0.1), loss='binary_crossentropy')
hist = model2.fit(x_train, y_train, epochs = 50, validation_split=0.2).history
```

```
Epoch 1/50
16/16 [==============================] - 1s 20ms/step - loss: 0.6072 - val_loss: 0.5143
Epoch 2/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4673 - val_loss: 0.4678
Epoch 3/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4408 - val_loss: 0.4563
Epoch 4/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4559 - val_loss: 0.4008
Epoch 5/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4555 - val_loss: 0.4101
Epoch 6/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4344 - val_loss: 0.4082
Epoch 7/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4347 - val_loss: 0.4019
Epoch 8/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4207 - val_loss: 0.4011
Epoch 9/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4108 - val_loss: 0.3968
Epoch 10/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4118 - val_loss: 0.3900
Epoch 11/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4152 - val_loss: 0.4400
Epoch 12/50
16/16 [==============================] - 0s 8ms/step - loss: 0.4088 - val_loss: 0.4613
Epoch 13/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4262 - val_loss: 0.4046
Epoch 14/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4917 - val_loss: 0.5253
Epoch 15/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4470 - val_loss: 0.4433
Epoch 16/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4259 - val_loss: 0.4327
Epoch 17/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4190 - val_loss: 0.4122
Epoch 18/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4120 - val_loss: 0.5035
Epoch 19/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4478 - val_loss: 0.4183
Epoch 20/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4129 - val_loss: 0.3980
Epoch 21/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4152 - val_loss: 0.4180
Epoch 22/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4010 - val_loss: 0.4194
Epoch 23/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4071 - val_loss: 0.4060
Epoch 24/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4034 - val_loss: 0.4026
Epoch 25/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4041 - val_loss: 0.4243
Epoch 26/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4172 - val_loss: 0.4270
Epoch 27/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4175 - val_loss: 0.3934
Epoch 28/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4044 - val_loss: 0.4039
Epoch 29/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4161 - val_loss: 0.4229
```

```
pred = model2.predict(x_val)
pred = np.where(pred >= 0.5, 1, 0)

    9/9 [==============================] - 0s 5ms/step
```

```
dl_history_plot(hist)
```



```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred))

    [[145  25]
     [ 36  62]]
                  precision    recall  f1-score   support

               0       0.80      0.85      0.83       170
               1       0.71      0.63      0.67        98

        accuracy                           0.77       268
       macro avg       0.76      0.74      0.75       268
    weighted avg       0.77      0.77      0.77       268
```

(5) 실습3

- 이번에는 여러분이 원하는 대로 설계하고, 학습해 봅시다.

```
n

    8
```

```
clear_session()
model3 = Sequential([Dense(16, input_shape=(n, ), activation='relu'),
                     Dense(8, activation='relu'),
                     Dense(4, activation='relu'),
                     Dense(1, activation='sigmoid')])
model3.summary()

    Model: "sequential"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     dense (Dense)               (None, 16)                144

     dense_1 (Dense)             (None, 8)                 136
```

```
 dense_2 (Dense)             (None, 4)                 36        
                                                                 
 dense_3 (Dense)             (None, 1)                 5         
                                                                 
=================================================================
Total params: 321 (1.25 KB)
Trainable params: 321 (1.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

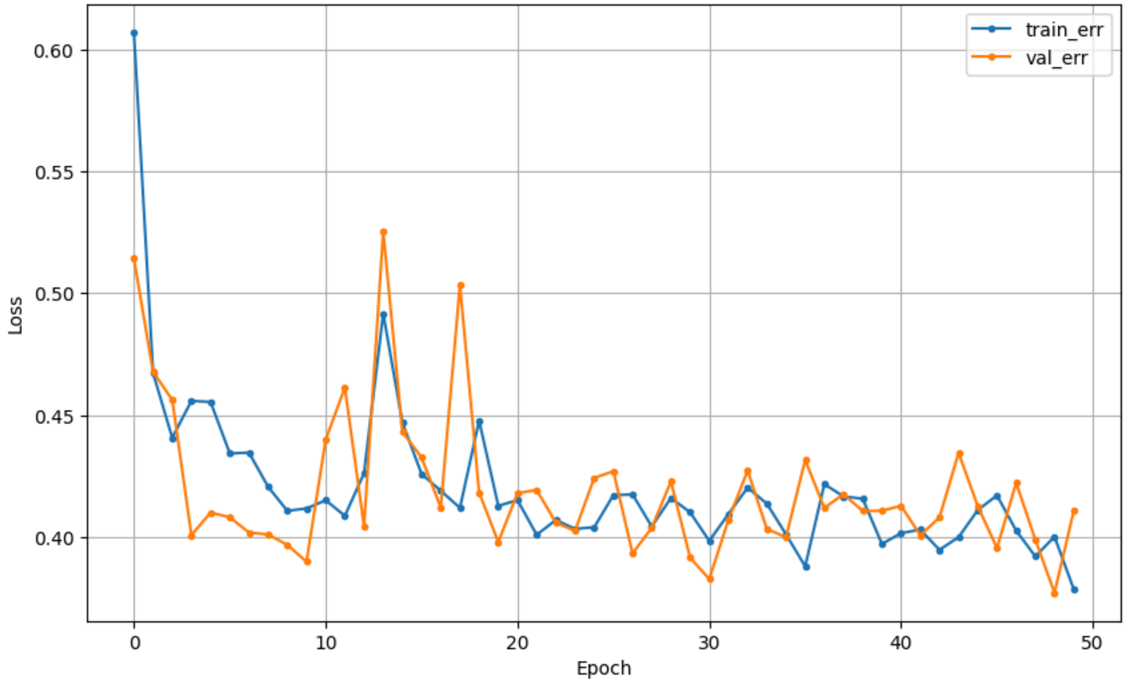
```
model3.compile(optimizer=Adam(0.05), loss='binary_crossentropy')
model3.fit(x_train, y_train, epochs=50, validation_split=0.2)
```

```
16/16 [==============================] - 0s 5ms/step - loss: 0.4234 - val_loss: 0.4136
Epoch 23/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4035 - val_loss: 0.4077
Epoch 24/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4415 - val_loss: 0.4152
Epoch 25/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4284 - val_loss: 0.4034
Epoch 26/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4175 - val_loss: 0.4398
Epoch 27/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4244 - val_loss: 0.4082
Epoch 28/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4038 - val_loss: 0.4070
Epoch 29/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4149 - val_loss: 0.3999
Epoch 30/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4061 - val_loss: 0.4216
Epoch 31/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4101 - val_loss: 0.4003
Epoch 32/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4019 - val_loss: 0.4084
Epoch 33/50
16/16 [==============================] - 0s 6ms/step - loss: 0.3984 - val_loss: 0.3996
Epoch 34/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4072 - val_loss: 0.4024
Epoch 35/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4039 - val_loss: 0.4122
Epoch 36/50
16/16 [==============================] - 0s 7ms/step - loss: 0.4058 - val_loss: 0.4099
Epoch 37/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4086 - val_loss: 0.4088
Epoch 38/50
16/16 [==============================] - 0s 6ms/step - loss: 0.4135 - val_loss: 0.4033
Epoch 39/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4106 - val_loss: 0.4122
Epoch 40/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4028 - val_loss: 0.4014
Epoch 41/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4106 - val_loss: 0.4360
Epoch 42/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4009 - val_loss: 0.4055
Epoch 43/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4006 - val_loss: 0.4124
Epoch 44/50
16/16 [==============================] - 0s 5ms/step - loss: 0.3990 - val_loss: 0.4296
Epoch 45/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4274 - val_loss: 0.5295
Epoch 46/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4381 - val_loss: 0.4725
Epoch 47/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4469 - val_loss: 0.4274
Epoch 48/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4387 - val_loss: 0.4088
Epoch 49/50
16/16 [==============================] - 0s 5ms/step - loss: 0.4088 - val_loss: 0.4246
Epoch 50/50
16/16 [==============================] - 0s 4ms/step - loss: 0.4117 - val_loss: 0.4311
<keras.src.callbacks.History at 0x7f0397b29ab0>
```

```
pred = model3.predict(x_val)
pred = np.where(pred >= 0.45, 1 , 0)
```

```
9/9 [==============================] - 0s 2ms/step
```

```
dl_history_plot(hist)
```



```
print(confusion_matrix(y_val, pred))
print(classification_report(y_val, pred))
```

```
[[136  34]
 [ 25  73]]
              precision    recall  f1-score   support

           0       0.84      0.80      0.82       170
           1       0.68      0.74      0.71        98

    accuracy                           0.78       268
   macro avg       0.76      0.77      0.77       268
weighted avg       0.79      0.78      0.78       268
```

코딩을 시작하거나 AI로 코드를 생성하세요.