모델의 성능 향상시키기

1. 데이터의 확인과 검증셋

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
import pandas as pd

# 와인 데이터를 불러옵니다.
df = pd.read_csv('./data/wine.csv', header=None)

# 데이터를 미리 보겠습니다.
df
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	1
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	1
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	1
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	0
6493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	0
6494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	0
6495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	0
6496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	0

6497 rows × 13 columns

```
# 와인의 속성을 X로 와인의 분류를 y로 저장합니다.
X = df.iloc[:,0:12]
y = df.iloc[:,12]
```

```
# 학습셋과 테스트셋으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

# 모델 구조를 설정합니다.
model = Sequential()
model.add(Dense(30,  input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 모델을 실행합니다.
history=model.fit(X_train, y_train, epochs=50, batch_size=500, validation_split=0.25) # 0.8 x 0.25
```

```
8/8 [==============================] - 0s 6ms/step - loss: 0.2143 - accuracy: 0.9305 - val_loss: 0.2125 - val_accuracy: 0.9323
Epoch 25/50
8/8 [==============================] - 0s 7ms/step - loss: 0.2122 - accuracy: 0.9307 - val_loss: 0.2106 - val_accuracy: 0.9308
Epoch 26/50
8/8 [==============================] - 0s 10ms/step - loss: 0.2106 - accuracy: 0.9315 - val_loss: 0.2091 - val_accuracy: 0.9323
Epoch 27/50
8/8 [==============================] - 0s 7ms/step - loss: 0.2090 - accuracy: 0.9310 - val_loss: 0.2078 - val_accuracy: 0.9323
Epoch 28/50
8/8 [==============================] - 0s 9ms/step - loss: 0.2085 - accuracy: 0.9307 - val_loss: 0.2065 - val_accuracy: 0.9323
Epoch 29/50
8/8 [==============================] - 0s 7ms/step - loss: 0.2067 - accuracy: 0.9312 - val_loss: 0.2054 - val_accuracy: 0.9323
Epoch 30/50
8/8 [==============================] - 0s 8ms/step - loss: 0.2051 - accuracy: 0.9317 - val_loss: 0.2046 - val_accuracy: 0.9315
Epoch 31/50
8/8 [==============================] - 0s 9ms/step - loss: 0.2043 - accuracy: 0.9317 - val_loss: 0.2036 - val_accuracy: 0.9323
Epoch 32/50
8/8 [==============================] - 0s 9ms/step - loss: 0.2034 - accuracy: 0.9317 - val_loss: 0.2027 - val_accuracy: 0.9323
Epoch 33/50
8/8 [==============================] - 0s 6ms/step - loss: 0.2024 - accuracy: 0.9323 - val_loss: 0.2018 - val_accuracy: 0.9331
Epoch 34/50
8/8 [==============================] - 0s 9ms/step - loss: 0.2014 - accuracy: 0.9325 - val_loss: 0.2010 - val_accuracy: 0.9331
Epoch 35/50
8/8 [==============================] - 0s 7ms/step - loss: 0.2013 - accuracy: 0.9325 - val_loss: 0.2003 - val_accuracy: 0.9331
Epoch 36/50
8/8 [==============================] - 0s 8ms/step - loss: 0.2011 - accuracy: 0.9323 - val_loss: 0.2001 - val_accuracy: 0.9346
Epoch 37/50
8/8 [==============================] - 0s 6ms/step - loss: 0.1996 - accuracy: 0.9328 - val_loss: 0.1992 - val_accuracy: 0.9323
Epoch 38/50
8/8 [==============================] - 0s 7ms/step - loss: 0.1995 - accuracy: 0.9317 - val_loss: 0.1986 - val_accuracy: 0.9346
Epoch 39/50
8/8 [==============================] - 0s 7ms/step - loss: 0.1983 - accuracy: 0.9325 - val_loss: 0.1973 - val_accuracy: 0.9346
Epoch 40/50
8/8 [==============================] - 0s 9ms/step - loss: 0.1977 - accuracy: 0.9333 - val_loss: 0.1975 - val_accuracy: 0.9346
Epoch 41/50
8/8 [==============================] - 0s 8ms/step - loss: 0.1975 - accuracy: 0.9330 - val_loss: 0.1970 - val_accuracy: 0.9346
Epoch 42/50
8/8 [==============================] - 0s 9ms/step - loss: 0.1970 - accuracy: 0.9328 - val_loss: 0.1963 - val_accuracy: 0.9346
Epoch 43/50
8/8 [==============================] - 0s 9ms/step - loss: 0.1970 - accuracy: 0.9338 - val_loss: 0.1964 - val_accuracy: 0.9346
Epoch 44/50
8/8 [==============================] - 0s 9ms/step - loss: 0.1956 - accuracy: 0.9335 - val_loss: 0.1953 - val_accuracy: 0.9346
Epoch 45/50
8/8 [==============================] - 0s 8ms/step - loss: 0.1950 - accuracy: 0.9333 - val_loss: 0.1952 - val_accuracy: 0.9346
Epoch 46/50
8/8 [==============================] - 0s 10ms/step - loss: 0.1948 - accuracy: 0.9325 - val_loss: 0.1948 - val_accuracy: 0.9346
Epoch 47/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1947 - accuracy: 0.9330 - val_loss: 0.1940 - val_accuracy: 0.9354
Epoch 48/50
8/8 [==============================] - 0s 6ms/step - loss: 0.1945 - accuracy: 0.9343 - val_loss: 0.1940 - val_accuracy: 0.9354
Epoch 49/50
8/8 [==============================] - 0s 8ms/step - loss: 0.1932 - accuracy: 0.9343 - val_loss: 0.1928 - val_accuracy: 0.9354
Epoch 50/50
8/8 [==============================] - 0s 5ms/step - loss: 0.1926 - accuracy: 0.9343 - val_loss: 0.1924 - val_accuracy: 0.9354
```

```python
# 테스트 결과를 출력합니다.
score=model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

```
41/41 [==============================] - 0s 1ms/step - loss: 0.1692 - accuracy: 0.9462
Test accuracy: 0.9461538195610046
```

2. 모델 업데이트하기

기본 코드 불러오기

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split

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

```
# 와인 데이터를 불러옵니다.
df = pd.read_csv('./data/wine.csv', header=None)

# 와인의 속성을 X로 와인의 분류를 y로 저장합니다.
X = df.iloc[:,0:12]
y = df.iloc[:,12]

# 학습셋과 테스트셋으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

# 모델 구조를 설정합니다.
model = Sequential()
model.add(Dense(30,  input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 30)                390

 dense_5 (Dense)             (None, 12)                372

 dense_6 (Dense)             (None, 8)                 104

 dense_7 (Dense)             (None, 1)                 9

=================================================================
Total params: 875
Trainable params: 875
Non-trainable params: 0
_________________________________________________________________
```

모델의 저장 설정 및 실행

```
# 모델 저장의 조건을 설정합니다.
modelpath="./data/model/all/{epoch:02d}-{val_accuracy:.4f}.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, verbose=1)

# 모델을 실행합니다.
history=model.fit(X_train, y_train, epochs=50, batch_size=500, validation_split=0.25, verbose=0, ca
```

```
Epoch 33: saving model to ./data/model/all/33-0.9300.hdf5

Epoch 34: saving model to ./data/model/all/34-0.9292.hdf5

Epoch 35: saving model to ./data/model/all/35-0.9300.hdf5

Epoch 36: saving model to ./data/model/all/36-0.9315.hdf5

Epoch 37: saving model to ./data/model/all/37-0.9315.hdf5

Epoch 38: saving model to ./data/model/all/38-0.9323.hdf5

Epoch 39: saving model to ./data/model/all/39-0.9315.hdf5

Epoch 40: saving model to ./data/model/all/40-0.9338.hdf5

Epoch 41: saving model to ./data/model/all/41-0.9346.hdf5

Epoch 42: saving model to ./data/model/all/42-0.9323.hdf5

Epoch 43: saving model to ./data/model/all/43-0.9377.hdf5

Epoch 44: saving model to ./data/model/all/44-0.9338.hdf5

Epoch 45: saving model to ./data/model/all/45-0.9377.hdf5

Epoch 46: saving model to ./data/model/all/46-0.9377.hdf5

Epoch 47: saving model to ./data/model/all/47-0.9377.hdf5

Epoch 48: saving model to ./data/model/all/48-0.9377.hdf5

Epoch 49: saving model to ./data/model/all/49-0.9377.hdf5

Epoch 50: saving model to ./data/model/all/50-0.9362.hdf5
```

```
# 테스트 결과를 출력합니다.
score=model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

```
41/41 [==============================] - 0s 1ms/step - loss: 0.1638 - accuracy: 0.9462
Test accuracy: 0.9461538195610046
```

3. 그래프로 과적합 확인하기

```
# 그래프 확인을 위한 긴 학습
history=model.fit(X_train, y_train, epochs=2000, batch_size=500, validation_split=0.25)
```

```
Epoch 1988/2000
8/8 [==============================] - 0s 12ms/step - loss: 0.0222 - accuracy: 0.9936 - val_loss: 0.0648 - val_accuracy: 0.9838
Epoch 1989/2000
8/8 [==============================] - 0s 10ms/step - loss: 0.0226 - accuracy: 0.9928 - val_loss: 0.0671 - val_accuracy: 0.9823
Epoch 1990/2000
8/8 [==============================] - 0s 11ms/step - loss: 0.0218 - accuracy: 0.9933 - val_loss: 0.0643 - val_accuracy: 0.9846
Epoch 1991/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0198 - accuracy: 0.9944 - val_loss: 0.0611 - val_accuracy: 0.9854
Epoch 1992/2000
8/8 [==============================] - 0s 10ms/step - loss: 0.0203 - accuracy: 0.9938 - val_loss: 0.0627 - val_accuracy: 0.9854
Epoch 1993/2000
8/8 [==============================] - 0s 11ms/step - loss: 0.0202 - accuracy: 0.9938 - val_loss: 0.0666 - val_accuracy: 0.9823
Epoch 1994/2000
8/8 [==============================] - 0s 12ms/step - loss: 0.0214 - accuracy: 0.9931 - val_loss: 0.0633 - val_accuracy: 0.9862
Epoch 1995/2000
8/8 [==============================] - 0s 11ms/step - loss: 0.0210 - accuracy: 0.9941 - val_loss: 0.0602 - val_accuracy: 0.9846
Epoch 1996/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0205 - accuracy: 0.9936 - val_loss: 0.0602 - val_accuracy: 0.9838
Epoch 1997/2000
8/8 [==============================] - 0s 11ms/step - loss: 0.0195 - accuracy: 0.9949 - val_loss: 0.0645 - val_accuracy: 0.9846
Epoch 1998/2000
8/8 [==============================] - 0s 14ms/step - loss: 0.0200 - accuracy: 0.9946 - val_loss: 0.0661 - val_accuracy: 0.9838
Epoch 1999/2000
8/8 [==============================] - 0s 12ms/step - loss: 0.0207 - accuracy: 0.9931 - val_loss: 0.0634 - val_accuracy: 0.9854
Epoch 2000/2000
8/8 [==============================] - 0s 13ms/step - loss: 0.0243 - accuracy: 0.9913 - val_loss: 0.0688 - val_accuracy: 0.9808
```

```
# history에 저장된 학습 결과를 확인해 보겠습니다.
hist_df=pd.DataFrame(history.history)
hist_df
```

	loss	accuracy	val_loss	val_accuracy
0	0.162397	0.944316	0.169251	0.938462
1	0.161059	0.943033	0.167919	0.939231
2	0.158131	0.945086	0.166268	0.940000
3	0.155845	0.945086	0.164352	0.940000
4	0.153318	0.944829	0.162351	0.940769
...	...	...	...	...
1995	0.020486	0.993585	0.060234	0.983846
1996	0.019504	0.994868	0.064474	0.984615
1997	0.020011	0.994611	0.066095	0.983846
1998	0.020744	0.993072	0.063443	0.985385
1999	0.024349	0.991275	0.068789	0.980769

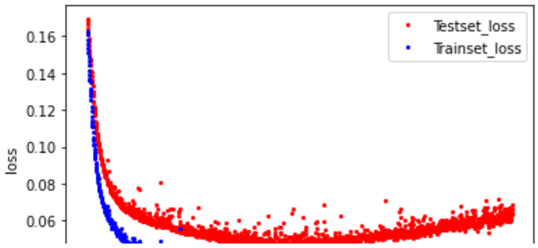
2000 rows × 4 columns

```
# y_vloss에 테스트셋(여기서는 검증셋)의 오차를 저장합니다.
y_vloss=hist_df['val_loss']

# y_loss에 학습셋의 오차를 저장합니다.
y_loss=hist_df['loss']

#x 값을 지정하고 테스트셋(검증셋)의 오차를 빨간색으로, 학습셋의 오차를 파란색으로 표시합니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, "o", c="red", markersize=2, label='Testset_loss')
plt.plot(x_len, y_loss, "o", c="blue", markersize=2, label='Trainset_loss')

plt.legend(loc='upper right')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



4. 학습의 자동 중단

기본 코드 불러오기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import os
import pandas as pd

# 와인 데이터를 불러옵니다.
df = pd.read_csv('./data/wine.csv', header=None)

# 와인의 속성을 X로 와인의 분류를 y로 저장합니다.
X = df.iloc[:,0:12]
y = df.iloc[:,12]

# 학습셋과 테스트셋으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)

# 모델 구조를 설정합니다.
model = Sequential()
model.add(Dense(30,  input_dim=12, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()

# 모델을 컴파일합니다.
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 30)                390

 dense_9 (Dense)             (None, 12)                372

 dense_10 (Dense)            (None, 8)                 104

 dense_11 (Dense)            (None, 1)                 9

=================================================================
Total params: 875
Trainable params: 875
Non-trainable params: 0
_________________________________________________________________
```

학습의 자동 중단 및 최적화 모델 저장

```
# 학습이 언제 자동 중단될지를 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=20)
```

```
# 최적화 모델이 저장될 폴더와 모델의 이름을 정합니다.
modelpath="./data/model/bestmodel.hdf5"

# 최적화 모델을 업데이트하고 저장합니다.
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=0, save_best_only=Tr

# 모델을 실행합니다.
history=model.fit(X_train, y_train, epochs=2000, batch_size=500, validation_split=0.25, verbose=1,
```

```
Epoch 342/2000
8/8 [==============================] - 0s 15ms/step - loss: 0.0400 - accuracy: 0.9902 - val_loss: 0.0708 - val_accuracy: 0.9846
Epoch 343/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0421 - accuracy: 0.9900 - val_loss: 0.0698 - val_accuracy: 0.9854
Epoch 344/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0400 - accuracy: 0.9897 - val_loss: 0.0697 - val_accuracy: 0.9846
Epoch 345/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0418 - accuracy: 0.9900 - val_loss: 0.0711 - val_accuracy: 0.9854
Epoch 346/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0424 - accuracy: 0.9897 - val_loss: 0.0752 - val_accuracy: 0.9838
Epoch 347/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0446 - accuracy: 0.9879 - val_loss: 0.0739 - val_accuracy: 0.9846
Epoch 348/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0406 - accuracy: 0.9905 - val_loss: 0.0705 - val_accuracy: 0.9846
Epoch 349/2000
8/8 [==============================] - 0s 10ms/step - loss: 0.0401 - accuracy: 0.9897 - val_loss: 0.0700 - val_accuracy: 0.9846
Epoch 350/2000
8/8 [==============================] - 0s 12ms/step - loss: 0.0413 - accuracy: 0.9890 - val_loss: 0.0691 - val_accuracy: 0.9854
Epoch 351/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0396 - accuracy: 0.9908 - val_loss: 0.0745 - val_accuracy: 0.9838
Epoch 352/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0402 - accuracy: 0.9900 - val_loss: 0.0761 - val_accuracy: 0.9823
Epoch 353/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0419 - accuracy: 0.9885 - val_loss: 0.0772 - val_accuracy: 0.9815
Epoch 354/2000
8/8 [==============================] - 0s 11ms/step - loss: 0.0420 - accuracy: 0.9892 - val_loss: 0.0765 - val_accuracy: 0.9815
Epoch 355/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0393 - accuracy: 0.9902 - val_loss: 0.0699 - val_accuracy: 0.9854
Epoch 356/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0390 - accuracy: 0.9900 - val_loss: 0.0732 - val_accuracy: 0.9838
Epoch 357/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0399 - accuracy: 0.9895 - val_loss: 0.0763 - val_accuracy: 0.9823
Epoch 358/2000
8/8 [==============================] - 0s 8ms/step - loss: 0.0395 - accuracy: 0.9905 - val_loss: 0.0705 - val_accuracy: 0.9846
Epoch 359/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0394 - accuracy: 0.9905 - val_loss: 0.0744 - val_accuracy: 0.9831
Epoch 360/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0395 - accuracy: 0.9902 - val_loss: 0.0701 - val_accuracy: 0.9846
Epoch 361/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0389 - accuracy: 0.9902 - val_loss: 0.0728 - val_accuracy: 0.9831
Epoch 362/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0402 - accuracy: 0.9900 - val_loss: 0.0697 - val_accuracy: 0.9838
Epoch 363/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0400 - accuracy: 0.9895 - val_loss: 0.0701 - val_accuracy: 0.9854
Epoch 364/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0396 - accuracy: 0.9908 - val_loss: 0.0701 - val_accuracy: 0.9846
Epoch 365/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0397 - accuracy: 0.9897 - val_loss: 0.0721 - val_accuracy: 0.9846
Epoch 366/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0412 - accuracy: 0.9895 - val_loss: 0.0719 - val_accuracy: 0.9846
Epoch 367/2000
8/8 [==============================] - 0s 9ms/step - loss: 0.0442 - accuracy: 0.9877 - val_loss: 0.0712 - val_accuracy: 0.9846
Epoch 368/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0470 - accuracy: 0.9877 - val_loss: 0.0711 - val_accuracy: 0.9831
Epoch 369/2000
8/8 [==============================] - 0s 6ms/step - loss: 0.0416 - accuracy: 0.9887 - val_loss: 0.0772 - val_accuracy: 0.9815
Epoch 370/2000
8/8 [==============================] - 0s 7ms/step - loss: 0.0408 - accuracy: 0.9890 - val_loss: 0.0771 - val_accuracy: 0.9815
```

```
# 테스트 결과를 출력합니다.
score=model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

```
41/41 [==============================] - 0s 1ms/step - loss: 0.0572 - accuracy: 0.9908
Test accuracy: 0.9907692074775696
```

0초 오전 8:59에 완료됨