SMARCLE 2021 winter study Team 4

14장 베스트 모델(Best Model) 만들기

: 와인의 종류 예측하기 실험
(Red Wine or White Wine ?)

17 송원진, 17 신도현, 18 권수지, 20 이유빈

SMARCLE

Contents

SMARCLE

01 데이터 확인과 실행

SMARCLE

01

```
df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1)
```

```
print(df.head(5))
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
964	8.5	0.47	0.27	1.9	0.058	18	38	0.99518	3.16	0.85	11.1	6	1
664	12.1	0.4	0.52	2	0.092	15	54	1	3.03	0.66	10.2	5	1
1692	6.9	0.21	0.33	1.8	0.034	48	136	0.9899	3.25	0.41	12.6	7	0
5801	6.7	0.24	0.31	2.3	0.044	37	113	0.99013	3.29	0.46	12.9	6	0
2207	6.1	0.28	0.25	17.75	0.044	48	161	0.9993	3.34	0.48	9.5	5	0

01

```
print(df.info( ))
```

Data	columns (total 13 columns):		
0	6497	non-null	float64
1	6497	non-null	float64
2	6497	non-null	float64
3	6497	non-null	float64
4	6497	non-null	float64
5	6497	non-null	float64
6	6497	non-null	float64
7	6497	non-null	float64
8	6497	non-null	float64
9	6497	non-null	float64
10	6497	non-null	float64
11	6497	non-null	int64
12	6497	non-null	int64
dtypes: float64(11), int64(2)			
memory usage: 710.6 KB			

01

UCI 머신러닝 저장소

0	주석산 농도	7	밀도
1	아세트산 농도	8	pH
2	구연산 농도	9	황산칼륨 농도
3	잔류 당분 농도	10	알코올 도수
4	염화나트륨 농도	11	와인의 맛(0~10등급)
5	유리 아황산 농도	12	class (1: 레드와인, 0: 화이트와인)
6	총 아황산 농도		

SMARCLE

```
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

01

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping

import pandas as pd
import numpy
import tensorflow as tf
import matplotlib.pyplot as plt
```

01

```
# seed 값 설정
seed = 0
numpy.random.seed(seed)
tf.random.set_seed(3)

# 데이터 입력
df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=1)
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]
```

SMARCLE

01

```
# 모델 설정
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'))

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

```
# 모델 실행
model.fit(X, Y, epochs=200, batch_size=200)

# 결과 출력
print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```

```
Epoch 198/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.0496
- accuracy: 0.9858
Epoch 199/200
6497/6497 [==============================] - 0s 7us/step - loss: 0.0499
- accuracy: 0.9848
Epoch 200/200
6497/6497 [==============================] - 0s 6us/step - loss: 0.0493
- accuracy: 0.9865
6497/6497 [==============================] - 0s 18us/step

 Accuracy: 0.9858
```

02

모델 업데이트

02

preview) 모델의 저장 / 재사용 (p.172)

```
from keras.models import load_model
```

model.save('모델이름.h5') : 모델의 저장

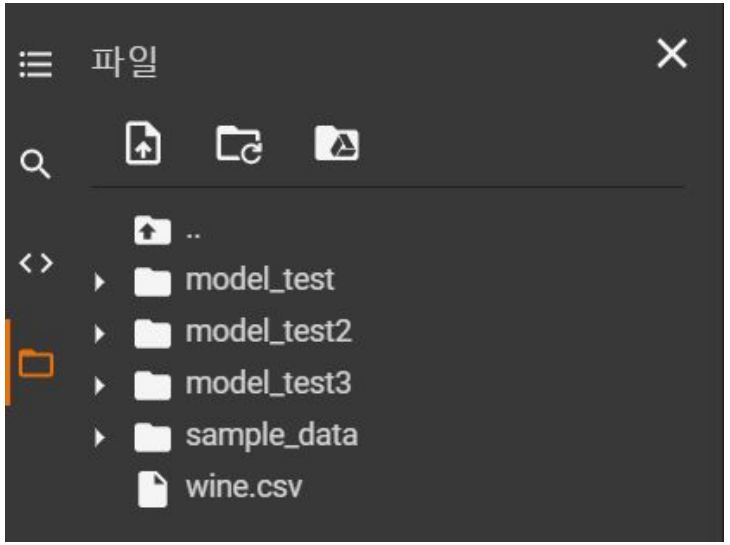
model = load_model('모델이름.h5')
: 모델 불러오기 (재사용)

SMARCLE

02

모델명에 에포크 + 모델의 정확도를 기록하기

1. 모델이 저장될 폴더를 지정
   (해당 폴더가 있는지 확인 -> 있으면 해당 폴더에 저장
   없으면 새 폴더 만들기)



2. hdf5 확장자로 저장
   (* 파일이름: 에포크 횟수 - 테스트셋 오차값. hdf5)

ex] 100번째 에포크 실행 결과 오차값 0.0612 :
파일명) 100-0.0612.hdf5

SMARCLE

```
import os

MODEL_DIR = './model/'                          #모델이 저장될 폴더를 지정
    if not os.path.exists(MODEL_DIR):           #만약 위 폴더가 없으면
        os.mkdir(MODEL_DIR)                     #mkdir(폴더명) : 새 폴더 생성

modelpath=''./model/{epoch:02d}-{val_loss:.4f}.hdf5''
                    5-0.0612.hdf5 -> 05-0.0612.hdf5
```

loss : 학습셋에 대한 오차값
val_loss : 테스트셋에 대한 오차값

02

ModelCheckpoint() : 우리의 시간은 소중해

```
from keras.callbacks import ModelCheckpoint
```

#케라스 콜백함수

```
checkpointer = ModelCheckpoint(_,_,_)
```

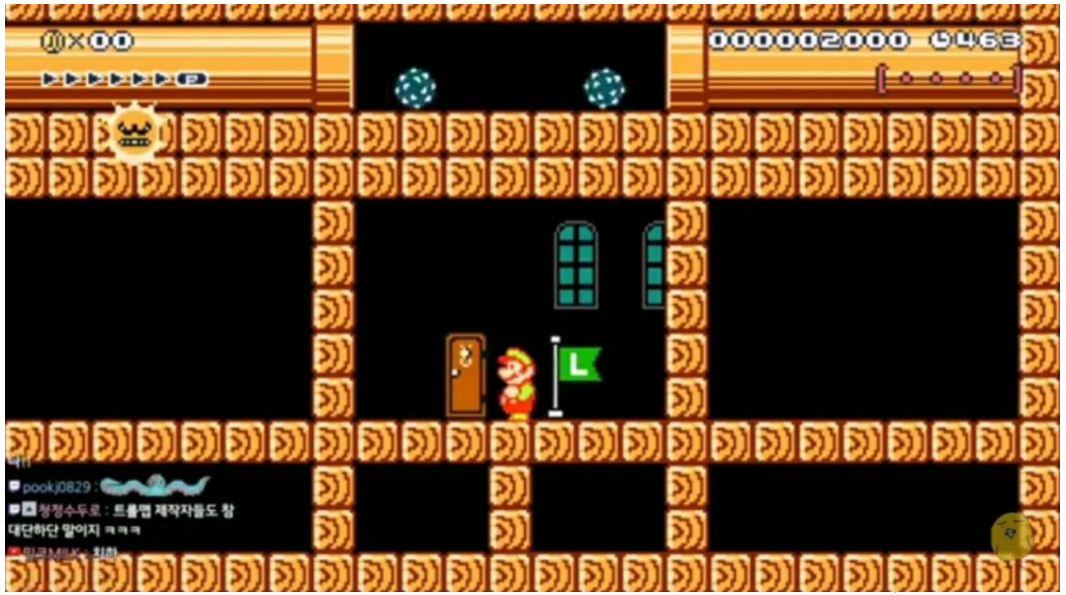
M.C를 쓰는 이유?

적합한 score를 도출 시 가중치 중간 저장,

memory overflow, crash가 발생 시

> 다시 weight 불러와 학습 진행 가능

>> time save



SMARCLE

02

ModelCheckpoint() :

```
from keras.callbacks import ModelCheckpoint
```

#케라스 콜백함수

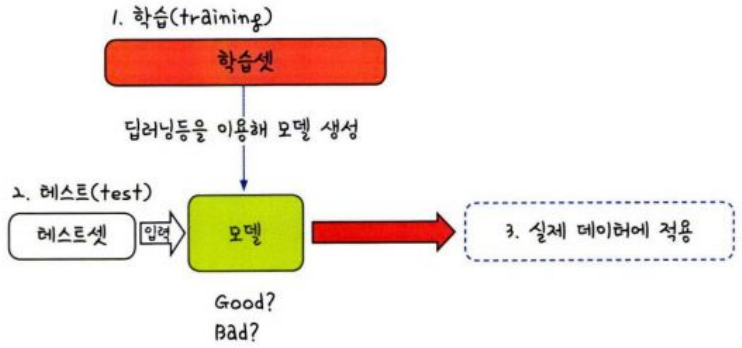
```
checkpointer =
ModelCheckpoint(filepath=modelpath,monitor='val_loss', verbose=1)
```

filepath : 모델이 저장될 곳을 ____로 지정 (model)

monitor : 'val_loss' 값이 개선되었을 때 호출

verbose : 1 (해당 함수의 진행상황을 출력 O)

0 (X)



SMARCLE

02

ModelCheckpoint() :

```
from keras.callbacks import ModelCheckpoint        #케라스 콜백함수

checkpointer = ModelCheckpoint(filepath=modelpath,monitor='val_loss', verbose=1, save_best_only=True)
```

- save_best_only=True:
  앞서 저장한 모델보다 좋아진 결과일 때만! 모델을 폴더에 저장

SMARCLE

정리

- callback : 특정 조건에서 자동으로 실행되는 함수

- 에포크가 진행되면서, 모든 값이 저장되는 것이 X
  테스트 오차를 실행한 결괏값이 향상되었을 때만 저장됨!

03

그래프로 확인하기

```
df = df_pre.sample(frac=0.15)

history = model.fit(X, Y, validation_split=0.33 , epochs=3500,
batch_size=500)
```

에포크를 얼마나 지정할 지 결정해야 한다.

```
import matplotlib.pyplot as plt

y_vloss=history.history['val_loss']
y_acc=history.history['accuracy']
```

03

```
x_len = numpy.arange(len(y_acc))
plt.plot(x_len, y_vloss, 'o', c ='red' , markersize=3)
plt.plot(x_len, y_acc, 'o', c = 'blue', markersize=3)
```

```
y_vloss = hist.history['val_loss']
y_acc = hist.history['accuracy']

print(y_acc)
x_len = np.arange(len(y_acc))
print(x_len)
plt.grid(True)
plt.plot(x_len,y_vloss, "ro")
plt.plot(x_len,y_acc,"bo")
```

```
[0.6928224925994873, 0.6928224925994873, 0.6928224925994873, 0.6888589859008789, 0.42967095971107483, 0.622089684009552, 0.6934770345687866, 0.6966722946166992, 0.7157976031303406, 0.7396575212478
[   0    1    2 ... 1997 1998 1999]
[<matplotlib.lines.Line2D at 0x7f682ae19b38>]
```

SMARCLE

```
[6]  import pandas as pd
     import numpy
     import os
     import matplotlib.pyplot as plt
     import tensorflow as tf
```

```
[7]  from keras.models import Sequential
     from keras.layers import Dense
     from keras.callbacks import ModelCheckpoint
```

```
[8]  # seed 값 설정
     numpy.random.seed(3)
     tf.random.set_seed(3)
```

```
[9]  #데이터 입력
     df_pre = pd.read_csv('wine.csv', header=None)
     df = df_pre.sample(frac=0.15)

     dataset = df.values
     X = dataset[:,0:12]
     Y = dataset[:,12]
```

SMARCLE

```
[10] # 모델의 설정
     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'))
```

```
[11] # 모델 컴파일
     model.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

```
[12] # 모델 저장 폴더 설정
     MODEL_DIR = './model/'
     if not os.path.exists(MODEL_DIR):
        os.mkdir(MODEL_DIR)
```

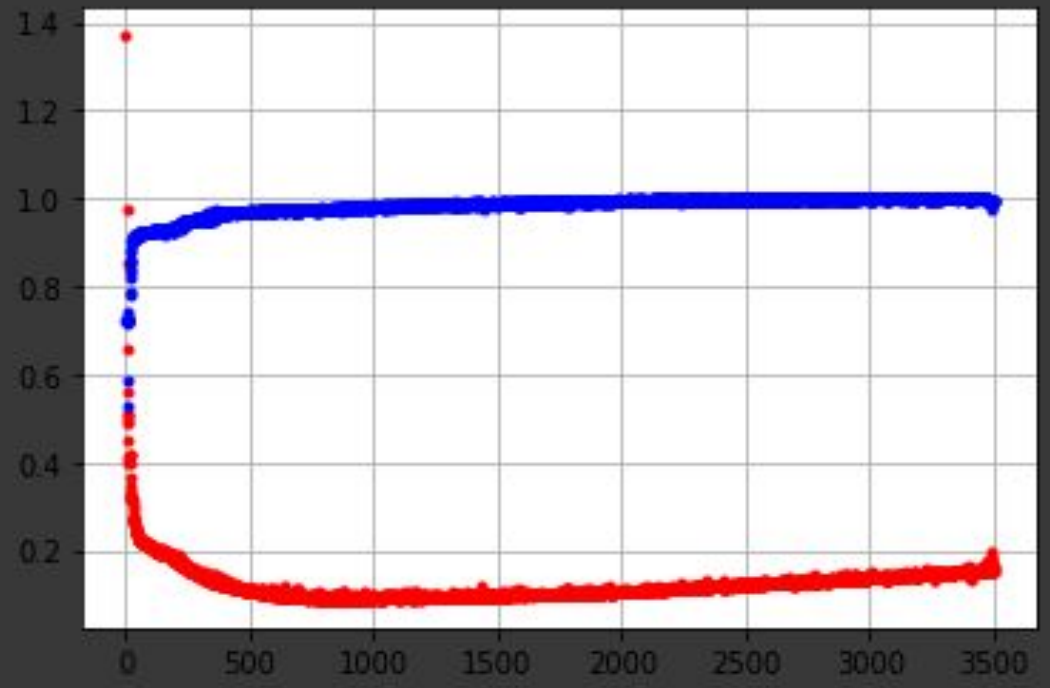
```
[13] # 모델 저장 조건 설정
     modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"
     checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1, save_best_only=True)
```

```python
# 모델 실행 및 저장
history = model.fit(X, Y, validation_split=0.33, epochs=3500, batch_size=500)

# y_vloss에 테스트셋으로 실험 결과의 오차 값을 저장
y_vloss=history.history['val_loss']

# y_acc 에 학습 셋으로 측정한 정확도의 값을 저장
y_acc=history.history['accuracy']
```

```
스트리밍 출력 내용이 길어서 마지막 5000줄이 삭제되었습니다.
Epoch 1001/3500
2/2 [==============================] - 0s 58ms/step - loss: 0.0460 - accuracy: 0.9804 - val_loss: 0.0928 - val_accuracy: 0.9814
Epoch 1002/3500
2/2 [==============================] - 0s 49ms/step - loss: 0.0459 - accuracy: 0.9834 - val_loss: 0.0914 - val_accuracy: 0.9783
Epoch 1003/3500
2/2 [==============================] - 0s 57ms/step - loss: 0.0473 - accuracy: 0.9834 - val_loss: 0.0915 - val_accuracy: 0.9752
Epoch 1004/3500
2/2 [==============================] - 0s 50ms/step - loss: 0.0460 - accuracy: 0.9811 - val_loss: 0.0922 - val_accuracy: 0.9783
Epoch 1005/3500
2/2 [==============================] - 0s 57ms/step - loss: 0.0461 - accuracy: 0.9828 - val_loss: 0.0906 - val_accuracy: 0.9814
Epoch 1006/3500
2/2 [==============================] - 0s 47ms/step - loss: 0.0452 - accuracy: 0.9817 - val_loss: 0.0902 - val_accuracy: 0.9814
Epoch 1007/3500
2/2 [==============================] - 0s 51ms/step - loss: 0.0452 - accuracy: 0.9848 - val_loss: 0.0912 - val_accuracy: 0.9814
Epoch 1008/3500
2/2 [==============================] - 0s 48ms/step - loss: 0.0452 - accuracy: 0.9834 - val_loss: 0.0929 - val_accuracy: 0.9783
Epoch 1009/3500
2/2 [==============================] - 0s 62ms/step - loss: 0.0476 - accuracy: 0.9794 - val_loss: 0.0923 - val_accuracy: 0.9783
Epoch 1010/3500
2/2 [==============================] - 0s 49ms/step - loss: 0.0459 - accuracy: 0.9841 - val_loss: 0.0913 - val_accuracy: 0.9720
Epoch 1011/3500
2/2 [==============================] - 0s 53ms/step - loss: 0.0480 - accuracy: 0.9828 - val_loss: 0.0928 - val_accuracy: 0.9783
Epoch 1012/3500
2/2 [==============================] - 0s 44ms/step - loss: 0.0476 - accuracy: 0.9794 - val_loss: 0.0964 - val_accuracy: 0.9814
Epoch 1013/3500
2/2 [==============================] - 0s 51ms/step - loss: 0.0478 - accuracy: 0.9801 - val_loss: 0.0939 - val_accuracy: 0.9783
Epoch 1014/3500
2/2 [==============================] - 0s 143ms/step - loss: 0.0487 - accuracy: 0.9787 - val_loss: 0.0922 - val_accuracy: 0.9752
Epoch 1015/3500
2/2 [==============================] - 0s 52ms/step - loss: 0.0460 - accuracy: 0.9828 - val_loss: 0.0920 - val_accuracy: 0.9752
Epoch 1016/3500
2/2 [==============================] - 0s 50ms/step - loss: 0.0458 - accuracy: 0.9841 - val_loss: 0.0935 - val_accuracy: 0.9783
Epoch 1017/3500
2/2 [==============================] - 0s 56ms/step - loss: 0.0461 - accuracy: 0.9845 - val_loss: 0.0968 - val_accuracy: 0.9752
Epoch 1018/3500
2/2 [==============================] - 0s 50ms/step - loss: 0.0485 - accuracy: 0.9828 - val_loss: 0.0929 - val_accuracy: 0.9752
Epoch 1019/3500
2/2 [==============================] - 0s 58ms/step - loss: 0.0474 - accuracy: 0.9828 - val_loss: 0.0918 - val_accuracy: 0.9752
Epoch 1020/3500
2/2 [==============================] - 0s 51ms/step - loss: 0.0469 - accuracy: 0.9828 - val_loss: 0.0922 - val_accuracy: 0.9783
```

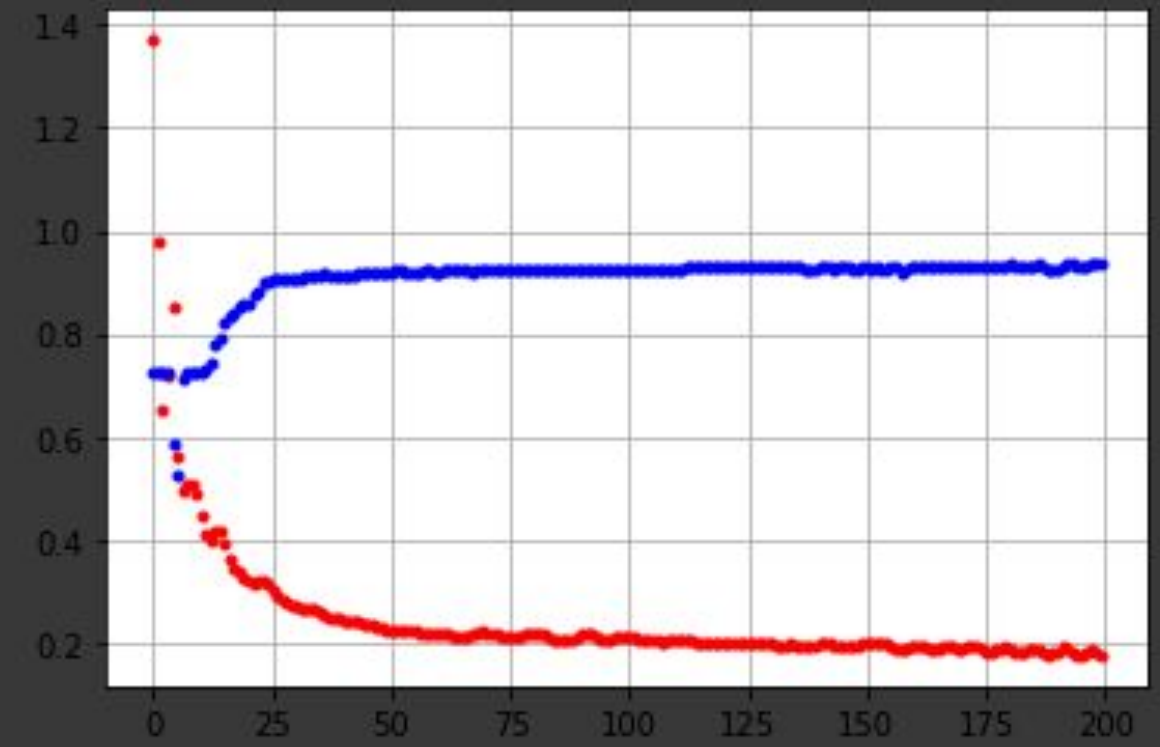
```
Epoch 3480/3500
2/2 [==============================] - 0s 60ms/step - loss: 0.0276 - accuracy: 0.9916 - val_loss: 0.1570 - val_accuracy: 0.9752
Epoch 3481/3500
2/2 [==============================] - 0s 55ms/step - loss: 0.0133 - accuracy: 0.9966 - val_loss: 0.1664 - val_accuracy: 0.9720
Epoch 3482/3500
2/2 [==============================] - 0s 84ms/step - loss: 0.0082 - accuracy: 0.9966 - val_loss: 0.1566 - val_accuracy: 0.9814
Epoch 3483/3500
2/2 [==============================] - 0s 64ms/step - loss: 0.0064 - accuracy: 0.9973 - val_loss: 0.1629 - val_accuracy: 0.9814
Epoch 3484/3500
2/2 [==============================] - 0s 54ms/step - loss: 0.0091 - accuracy: 0.9963 - val_loss: 0.1564 - val_accuracy: 0.9783
Epoch 3485/3500
2/2 [==============================] - 0s 56ms/step - loss: 0.0088 - accuracy: 0.9963 - val_loss: 0.1565 - val_accuracy: 0.9814
Epoch 3486/3500
2/2 [==============================] - 0s 51ms/step - loss: 0.0148 - accuracy: 0.9973 - val_loss: 0.1822 - val_accuracy: 0.9720
Epoch 3487/3500
2/2 [==============================] - 0s 65ms/step - loss: 0.0415 - accuracy: 0.9777 - val_loss: 0.1508 - val_accuracy: 0.9752
Epoch 3488/3500
2/2 [==============================] - 0s 52ms/step - loss: 0.0072 - accuracy: 0.9973 - val_loss: 0.1971 - val_accuracy: 0.9658
Epoch 3489/3500
2/2 [==============================] - 0s 55ms/step - loss: 0.0325 - accuracy: 0.9848 - val_loss: 0.1659 - val_accuracy: 0.9814
Epoch 3490/3500
2/2 [==============================] - 0s 56ms/step - loss: 0.0232 - accuracy: 0.9922 - val_loss: 0.1824 - val_accuracy: 0.9752
Epoch 3491/3500
2/2 [==============================] - 0s 63ms/step - loss: 0.0312 - accuracy: 0.9889 - val_loss: 0.1548 - val_accuracy: 0.9752
Epoch 3492/3500
2/2 [==============================] - 0s 74ms/step - loss: 0.0171 - accuracy: 0.9925 - val_loss: 0.1574 - val_accuracy: 0.9752
Epoch 3493/3500
2/2 [==============================] - 0s 56ms/step - loss: 0.0162 - accuracy: 0.9946 - val_loss: 0.1727 - val_accuracy: 0.9720
Epoch 3494/3500
2/2 [==============================] - 0s 55ms/step - loss: 0.0348 - accuracy: 0.9821 - val_loss: 0.1502 - val_accuracy: 0.9783
Epoch 3495/3500
2/2 [==============================] - 0s 55ms/step - loss: 0.0085 - accuracy: 0.9973 - val_loss: 0.1784 - val_accuracy: 0.9720
Epoch 3496/3500
2/2 [==============================] - 0s 54ms/step - loss: 0.0149 - accuracy: 0.9922 - val_loss: 0.1564 - val_accuracy: 0.9814
Epoch 3497/3500
2/2 [==============================] - 0s 55ms/step - loss: 0.0085 - accuracy: 0.9973 - val_loss: 0.1644 - val_accuracy: 0.9814
Epoch 3498/3500
2/2 [==============================] - 0s 55ms/step - loss: 0.0126 - accuracy: 0.9949 - val_loss: 0.1500 - val_accuracy: 0.9752
Epoch 3499/3500
2/2 [==============================] - 0s 64ms/step - loss: 0.0062 - accuracy: 0.9973 - val_loss: 0.1592 - val_accuracy: 0.9752
Epoch 3500/3500
2/2 [==============================] - 0s 53ms/step - loss: 0.0138 - accuracy: 0.9966 - val_loss: 0.1492 - val_accuracy: 0.9783
```

```
# x값을 지정하고 정확도를 파란색으로, 오차를 빨간색으로 표시
x_len = numpy.arange(len(y_acc))
plt.plot(x_len, y_vloss, "o", c="red", markersize=3)
plt.plot(x_len, y_acc, "o", c="blue", markersize=3)
plt.grid(True)
plt.show()
```

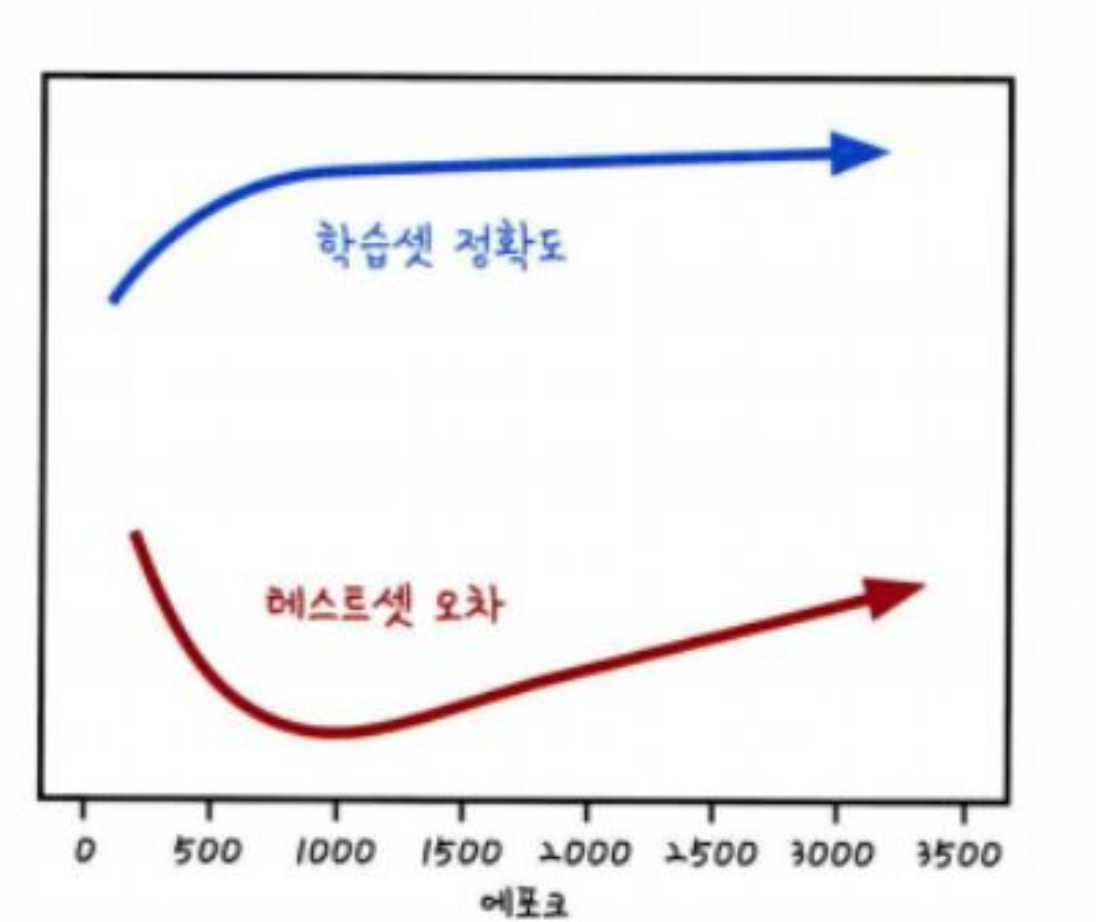


에포크 = 200일때

```
# x값을 지정하고 정확도를 파란색으로, 오차를 빨간색으로 표시
x_len = numpy.arange(len(y_acc))
plt.plot(x_len, y_vloss, "o", c="red", markersize=3)
plt.plot(x_len, y_acc, "o", c="blue", markersize=3)
plt.grid(True)
plt.show()
```



단순화시킨 그래프 결과



학습이 진행될 수록

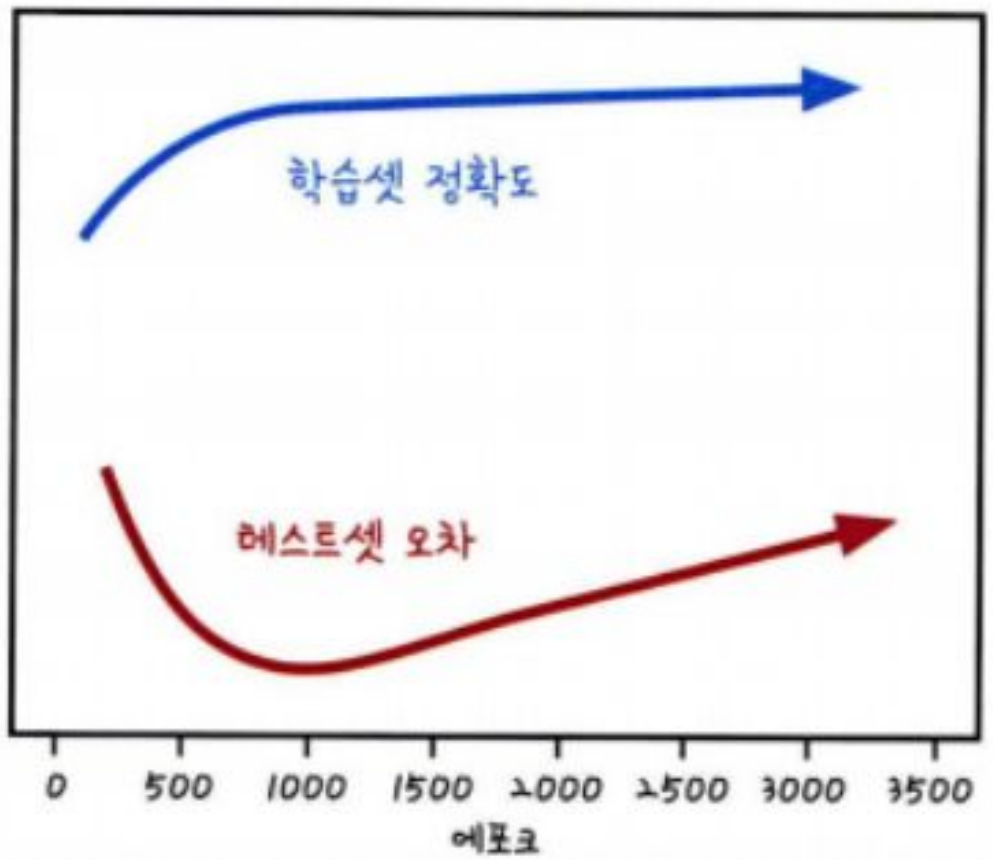
학습셋의 정확도는 오르지만

테스트셋에서는 과적합이 발생

04

학습의 자동 중단

보완 : EarlyStopping()



: 학습이 진행되어도 테스트셋 오차가 줄지 않으면 학습을 멈추게 하는 함수

- from keras.callbacks import EarlyStopping

04

EarlyStopping() 함수에

+ 모니터할 값 #여기선 val_loss
+ 테스트 오차가 좋아지지 않아도 몇 번까지 기다릴지(patience) 값

→ early_stopping_callback 변수에 저장

- early_stopping_callback = EarlyStopping(monitor='val_loss', patience=100)

→ 에포크 횟수(epoch), 배치 크기(batch_size) 설정

→ 앞서 입력한 early_stopping_callback 값 호출(callbacks)

- model.fit(X,Y, validation_split=0.33, epochs=2000, batch_size=500, callbacks=[early_stopping_callback])

04

정리

```
.....
model.compil(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

+

- early_stopping_callback= EarlyStopping(monitor='val_loss',patience=100) #학습 자동 중단
- model.fit(X,Y, validation_split=0.2, epochs=2000, batch_size=500, callbacks=[early_stopping_callback]) #모델 실행

+ print(''\n Accuracy: %.4f'' % (model.evaluate(X,Y)[1])) #출력

SMARCLE

04

코드로 살펴보기

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping

import pandas as pd
import numpy
import tensorflow as tf

# seed 값 설정
numpy.random.seed(3)
tf.random.set_seed(3)
```

```
df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=0.15)

dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]

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

04

코드로 살펴보기

```
# 학습 자동 중단 설정
early_stopping_callback = EarlyStopping(monitor='val_loss',
patience=100)

# 모델 실행
model.fit(X, Y, validation_split=0.2, epochs=2000, batch_
size=500, callbacks=[early_stopping_callback])

# 결과 출력
print("\n Accuracy: %.4f" % (model.evaluate(X, Y)[1]))
```

04

코드로 살펴보기

(중략)

```
Epoch 775/2000
780/780 [==============================] - 0s - loss: 0.0186 - acc:
0.9962 - val_loss: 0.0662 - val_acc: 0.9795
Epoch 776/2000
780/780 [==============================] - 0s - loss: 0.0193 - acc:
0.9936 - val_loss: 0.0743 - val_acc: 0.9795
Epoch 777/2000
780/780 [==============================] - 0s - loss: 0.0198 - acc:
0.9936 - val_loss: 0.0660 - val_acc: 0.9795
Epoch 778/2000
780/780 [==============================] - 0s - loss: 0.0236 - acc:
0.9936 - val_loss: 0.0699 - val_acc: 0.9846
 32/975 [.....] - ETA: 0s
 Accuracy: 0.9918
```

04

```
from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping

import pandas as pd
import numpy
import os
import tensorflow as tf

# seed 값 설정
numpy.random.seed(3)
tf.random.set_seed(3)

df_pre = pd.read_csv('../dataset/wine.csv', header=None)
df = df_pre.sample(frac=0.15)
dataset = df.values
X = dataset[:,0:12]
Y = dataset[:,12]

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.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# 모델 저장 폴더 만들기
MODEL_DIR = './model/'
if not os.path.exists(MODEL_DIR):
    os.mkdir(MODEL_DIR)

modelpath="./model/{epoch:02d}-{val_loss:.4f}.hdf5"

# 모델 업데이트 및 저장
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_
loss', verbose=1, save_best_only=True)

# 학습 자동 중단 설정
early_stopping_callback = EarlyStopping(monitor='val_loss',
patience=100)

model.fit(X, Y, validation_split=0.2, epochs=3500, batch_size=500,
verbose=0, callbacks=[early_stopping_callback,checkpointer])
```

(중략)

Epoch 00680: val_loss did not improve

Epoch 00681: val_loss did not improve

Epoch 00682: val_loss improved from 0.05415 to 0.05286, saving model to ./model/682-0.0529.hdf5

Epoch 00683: val_loss did not improve

Epoch 00684: val_loss did not improve

Epoch 00685: val_loss did not improve

(중략)

Epoch 00781: val_loss did not improve

Epoch 00782: val_loss did not improve

Epoch 00783: val_loss did not improve

Thank you (❁´◡`❁)

QnA

SMARCLE