케라스 딥러닝 학습 조기 종료 함수 - EarlyStopping()

학습 내용

- MNIST 데이터 셋을 활용하여 딥러닝 모델을 구현해 본다.
- 학습 조기 종료에 대해 알아본다.

더 이상의 학습에 대한 개선이 없을때, 끝까지 학습을 시켜야 하나?

01 학습 조기 종료 시키기

- 학습의 조기 종료 함수 - EarlyStopping()
- 더 이상의 개선의 여지가 없을 때, 학습을 종료시키는 콜백함수
- fit() 함수에서는 EarlyStopping() 콜백함수가 학습 과정 중에 매번 호출됨.

02 EarlyStopping 지정 방법

```
early_stopping = EarlyStopping()
model.fit(X_train, Y_train, nb_epoch= 1000, callbacks=[early_stopping])
```

03 CallBack 함수의 사용인자

```
keras.callbacks.EarlyStopping(monitor='val_loss',
                              min_delta=0,
                              patience=0,
                              verbose=0,
                              mode='auto')
```

- monitor : 관찰 항목 (val_loss : 평가 비용 함수)
- min_delta : 개선되고 있다는 최소 변화량. 변화량이 적은 경우, 개선이 없음으로 판단.
- patience : 개선이 없다고 바로 종료하지 않고, 얼마나 기다려줄지 지정. 10번이라면 10번째 지속될 때, 학습 종료
- verbose : 얼마나 자세하게 정보를 볼지(0,1,2)
- mode : 관찰 항목에 대해 개선이 없다고 판단할 기준 지정.
  - auto : 관찰하는 이름에 따라 자동 지정
  - min : 관찰하는 있는 항목이 감소되는 것을 멈출 때 종료
  - max : 관찰하고 있는 항목이 증가되는 것을 멈출 때 종료

04 실습해 보기

In [1]:

```
from keras.utils import np_utils
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
import numpy as np
```

데이터 나누기

In [2]:

```
np.random.seed(3)

(X_train, y_train), (X_test, y_test) = mnist.load_data()

# 훈련셋과 검증셋 분리
X_val = X_train[50000:]
y_val = y_train[50000:]
X_train = X_train[:50000]
y_train = y_train[:50000]
```

In [3]:

```
X_train = X_train.reshape(50000, 784).astype('float32') / 255.0
X_val = X_val.reshape(10000, 784).astype('float32') / 255.0
X_test = X_test.reshape(10000, 784).astype('float32') / 255.0
```

In [4]:

```
# 훈련셋, 검증셋 고르기
train_rand_idxs = np.random.choice(50000, 10000)
val_rand_idxs = np.random.choice(10000, 5000)

X_train = X_train[train_rand_idxs]
y_train = y_train[train_rand_idxs]
X_val = X_val[val_rand_idxs]
y_val = y_val[val_rand_idxs]
```

In [5]:

```
# 라벨링 전환
y_train = np_utils.to_categorical(y_train)
y_val = np_utils.to_categorical(y_val)
y_test = np_utils.to_categorical(y_test)
```

In [6]:

```
print(X_train.shape, y_train.shape)
print(X_val.shape, y_val.shape)
print(X_test.shape, y_test.shape)
```

```
(10000, 784) (10000, 10)
(5000, 784) (5000, 10)
(10000, 784) (10000, 10)
```

In [7]:

```
# 2. 모델 구성하기
model = Sequential()
model.add(Dense(units=64, input_dim=28*28, activation='relu'))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=10, activation='softmax'))
```

In [8]:

```
# 3. 모델의 오차함수, 최적화 함수 설정
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

조기 종료시키기

In [9]:

```
# 4. 모델 학습시키기
from keras.callbacks import EarlyStopping
# early_stopping = EarlyStopping() # 조기종료 콜백함수 정의
early_stopping = EarlyStopping(patience = 30) # 조기종료 콜백함수 정의
hist = model.fit(X_train, y_train,
                 epochs=3000,
                 batch_size=10,
                 validation_data=(X_val, y_val),
                 callbacks=[early_stopping])
```

```
Epoch 1/3000
1000/1000 [==============================] - 4s 3ms/step - loss: 0.9599 - accuracy:
0.7374 - val_loss: 0.4121 - val_accuracy: 0.8860
Epoch 2/3000
1000/1000 [==============================] - 4s 4ms/step - loss: 0.3894 - accuracy:
0.8887 - val_loss: 0.3354 - val_accuracy: 0.9002
Epoch 3/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.3141 - accuracy:
0.9098 - val_loss: 0.2906 - val_accuracy: 0.9140
Epoch 4/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.2703 - accuracy:
0.9220 - val_loss: 0.2660 - val_accuracy: 0.9276
Epoch 5/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.2417 - accuracy:
0.9285 - val_loss: 0.2542 - val_accuracy: 0.9302
Epoch 6/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.2177 - accuracy:
0.9347 - val_loss: 0.2270 - val_accuracy: 0.9380
Epoch 7/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1958 - accuracy:
0.9434 - val_loss: 0.2326 - val_accuracy: 0.9360
Epoch 8/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1780 - accuracy:
0.9486 - val_loss: 0.2094 - val_accuracy: 0.9424
Epoch 9/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1617 - accuracy:
0.9552 - val_loss: 0.2069 - val_accuracy: 0.9438
Epoch 10/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1459 - accuracy:
0.9590 - val_loss: 0.2039 - val_accuracy: 0.9438
Epoch 11/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1336 - accuracy:
0.9635 - val_loss: 0.2207 - val_accuracy: 0.9346
Epoch 12/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1222 - accuracy:
0.9663 - val_loss: 0.1959 - val_accuracy: 0.9460
Epoch 13/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1108 - accuracy:
0.9703 - val_loss: 0.2002 - val_accuracy: 0.9432
Epoch 14/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.1013 - accuracy:
0.9744 - val_loss: 0.2133 - val_accuracy: 0.9416
Epoch 15/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0944 - accuracy:
0.9737 - val_loss: 0.1907 - val_accuracy: 0.9500
Epoch 16/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0835 - accuracy:
0.9792 - val_loss: 0.1848 - val_accuracy: 0.9486
```

```
Epoch 17/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0774 - accuracy:
0.9806 - val_loss: 0.1908 - val_accuracy: 0.9472
Epoch 18/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0712 - accuracy:
0.9824 - val_loss: 0.1795 - val_accuracy: 0.9530
Epoch 19/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0645 - accuracy:
0.9848 - val_loss: 0.1843 - val_accuracy: 0.9502
Epoch 20/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0588 - accuracy:
0.9860 - val_loss: 0.1844 - val_accuracy: 0.9494
Epoch 21/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0531 - accuracy:
0.9886 - val_loss: 0.1725 - val_accuracy: 0.9558
Epoch 22/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0483 - accuracy:
0.9897 - val_loss: 0.1817 - val_accuracy: 0.9536
Epoch 23/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0444 - accuracy:
0.9909 - val_loss: 0.1861 - val_accuracy: 0.9528
Epoch 24/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0411 - accuracy:
0.9913 - val_loss: 0.1783 - val_accuracy: 0.9550
Epoch 25/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0365 - accuracy:
0.9935 - val_loss: 0.1822 - val_accuracy: 0.9520
Epoch 26/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0332 - accuracy:
0.9943 - val_loss: 0.1755 - val_accuracy: 0.9566
Epoch 27/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0304 - accuracy:
0.9952 - val_loss: 0.1830 - val_accuracy: 0.9526
Epoch 28/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0275 - accuracy:
0.9958 - val_loss: 0.1819 - val_accuracy: 0.9556
Epoch 29/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0251 - accuracy:
0.9968 - val_loss: 0.1861 - val_accuracy: 0.9550
Epoch 30/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0225 - accuracy:
0.9971 - val_loss: 0.1860 - val_accuracy: 0.9552
Epoch 31/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0200 - accuracy:
0.9978 - val_loss: 0.1906 - val_accuracy: 0.9556
Epoch 32/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0188 - accuracy:
0.9979 - val_loss: 0.1856 - val_accuracy: 0.9564
Epoch 33/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0170 - accuracy:
0.9987 - val_loss: 0.1938 - val_accuracy: 0.9560
Epoch 34/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0156 - accuracy:
0.9990 - val_loss: 0.1940 - val_accuracy: 0.9558
Epoch 35/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0142 - accuracy:
0.9993 - val_loss: 0.1919 - val_accuracy: 0.9562
Epoch 36/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0132 - accuracy:
0.9993 - val_loss: 0.1939 - val_accuracy: 0.9570
Epoch 37/3000
```

```
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0123 - accuracy:
0.9994 - val_loss: 0.1980 - val_accuracy: 0.9544
Epoch 38/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0111 - accuracy:
0.9997 - val_loss: 0.1952 - val_accuracy: 0.9566
Epoch 39/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0104 - accuracy:
0.9999 - val_loss: 0.1973 - val_accuracy: 0.9566
Epoch 40/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0096 - accuracy:
1.0000 - val_loss: 0.1952 - val_accuracy: 0.9584
Epoch 41/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0089 - accuracy:
1.0000 - val_loss: 0.2004 - val_accuracy: 0.9558
Epoch 42/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0083 - accuracy:
1.0000 - val_loss: 0.2011 - val_accuracy: 0.9562
Epoch 43/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0079 - accuracy:
1.0000 - val_loss: 0.1986 - val_accuracy: 0.9572
Epoch 44/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0074 - accuracy:
1.0000 - val_loss: 0.2041 - val_accuracy: 0.9570
Epoch 45/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0071 - accuracy:
1.0000 - val_loss: 0.2054 - val_accuracy: 0.9570
Epoch 46/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0066 - accuracy:
1.0000 - val_loss: 0.2056 - val_accuracy: 0.9566
Epoch 47/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0063 - accuracy:
1.0000 - val_loss: 0.2090 - val_accuracy: 0.9564
Epoch 48/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0061 - accuracy:
1.0000 - val_loss: 0.2066 - val_accuracy: 0.9568
Epoch 49/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0057 - accuracy:
1.0000 - val_loss: 0.2066 - val_accuracy: 0.9572
Epoch 50/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0054 - accuracy:
1.0000 - val_loss: 0.2107 - val_accuracy: 0.9574
Epoch 51/3000
1000/1000 [==============================] - 3s 3ms/step - loss: 0.0052 - accuracy:
1.0000 - val_loss: 0.2103 - val_accuracy: 0.9574
```

In [10]:

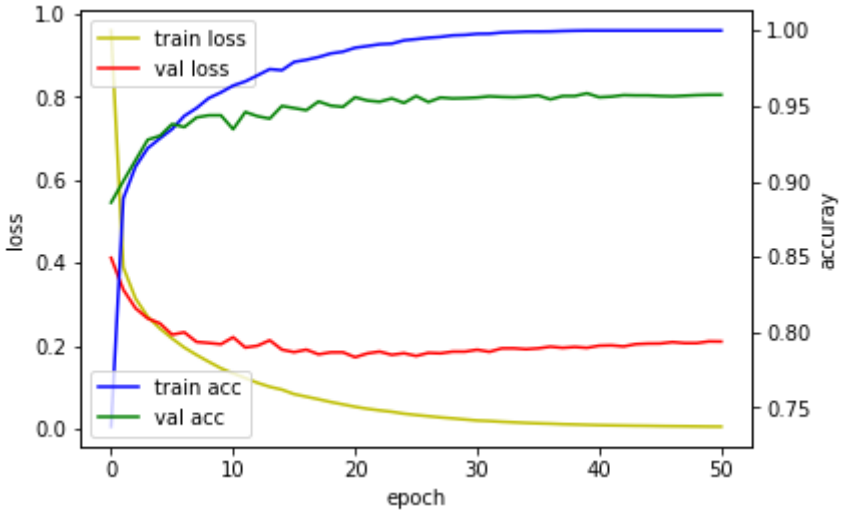
```
hist.history.keys()
```

Out[10]:

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [11]:

```
# 5. 모델 학습 과정 표시하기
%matplotlib inline
import matplotlib.pyplot as plt

fig, loss_ax = plt.subplots()

acc_ax = loss_ax.twinx()

loss_ax.plot(hist.history['loss'], 'y', label='train loss')
loss_ax.plot(hist.history['val_loss'], 'r', label='val loss')

acc_ax.plot(hist.history['accuracy'], 'b', label='train acc')
acc_ax.plot(hist.history['val_accuracy'], 'g', label='val acc')

loss_ax.set_xlabel('epoch')
loss_ax.set_ylabel('loss')
acc_ax.set_ylabel('accuray')

loss_ax.legend(loc='upper left')
acc_ax.legend(loc='lower left')

plt.show()
```



In [12]:

```
# 6. 모델 평가하기
loss_and_metrics = model.evaluate(X_test, y_test, batch_size=32)

print('')
print('loss : ' + str(loss_and_metrics[0]))
print('accuray : ' + str(loss_and_metrics[1]))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2075 - accuracy: 0.
9509

loss : 0.20746201276779175
accuray : 0.9509000182151794
```