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 [ ]:

```
# 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 [==============================] - 3s 3ms/step - loss: 1.5238 - accuracy:
0.5107 - val_loss: 0.4271 - val_accuracy: 0.8898
Epoch 2/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.4119 - accuracy:
0.8882 - val_loss: 0.3367 - val_accuracy: 0.9054
Epoch 3/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.3105 - accuracy:
0.9123 - val_loss: 0.2963 - val_accuracy: 0.9090
Epoch 4/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.2753 - accuracy:
0.9229 - val_loss: 0.2547 - val_accuracy: 0.9326
Epoch 5/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.2377 - accuracy:
0.9358 - val_loss: 0.2409 - val_accuracy: 0.9322
Epoch 6/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.2136 - accuracy:
0.9384 - val_loss: 0.2301 - val_accuracy: 0.9356
Epoch 7/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1865 - accuracy:
0.9457 - val_loss: 0.2131 - val_accuracy: 0.9432
Epoch 8/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1547 - accuracy:
0.9573 - val_loss: 0.2076 - val_accuracy: 0.9438
Epoch 9/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1453 - accuracy:
0.9594 - val_loss: 0.2000 - val_accuracy: 0.9452
Epoch 10/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1298 - accuracy:
0.9646 - val_loss: 0.2036 - val_accuracy: 0.9446
Epoch 11/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1236 - accuracy:
0.9651 - val_loss: 0.1991 - val_accuracy: 0.9392
Epoch 12/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.1061 - accuracy:
0.9720 - val_loss: 0.1865 - val_accuracy: 0.9474
Epoch 13/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0971 - accuracy:
0.9740 - val_loss: 0.1685 - val_accuracy: 0.9534
Epoch 14/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0926 - accuracy:
0.9735 - val_loss: 0.1695 - val_accuracy: 0.9552
Epoch 15/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0827 - accuracy:
0.9808 - val_loss: 0.1931 - val_accuracy: 0.9470
Epoch 16/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0779 - accuracy:
0.9786 - val_loss: 0.1724 - val_accuracy: 0.9524
```

```
Epoch 17/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0699 - accuracy:
0.9805 - val_loss: 0.1643 - val_accuracy: 0.9578
Epoch 18/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0556 - accuracy:
0.9857 - val_loss: 0.1645 - val_accuracy: 0.9572
Epoch 19/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0527 - accuracy:
0.9883 - val_loss: 0.1605 - val_accuracy: 0.9562
Epoch 20/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0474 - accuracy:
0.9884 - val_loss: 0.1616 - val_accuracy: 0.9574
Epoch 21/3000
1000/1000 [==============================] - 2s 2ms/step - loss: 0.0437 - accuracy:
0.9907 - val_loss: 0.1684 - val_accuracy: 0.9554
Epoch 22/3000
 168/1000 [====>.........................] - ETA: 1s - loss: 0.0317 - accuracy: 0.99
70
```

In [ ]:

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

In [ ]:

```
# 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 [ ]:

```
# 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]))
```

In [ ]: