LSTM을 이용한
로이터 뉴스 카테고리 분류하기

01 긴 내용의 텍스트를 카테고리로 분류하기

중부 지방은 대체로 맑겠으나, 남부 지방은 구름이 많겠습니다. → 날씨

올 초부터 유동성의 힘으로 주가가 일정하게 상승했습니다. → 주식

이번 선거에서는 누가 이길 것 같아? → 정치

퍼셉트론의 한계를 극복한 신경망이 다시 뜨고 있대. → 딥러닝

02 로이터 뉴스 데이터 불러오기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Embedding
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.datasets import reuters
from tensorflow.keras.callbacks import EarlyStopping

import numpy as np
import matplotlib.pyplot as plt

(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=1000, test_split=0.2)

category = np.max(y_train) + 1
print(category, '카테고리')
print(len(X_train), '학습용 뉴스 기사')
print(len(X_test), '테스트용 뉴스 기사')
print(X_train[0])
```

02 로이터 뉴스 데이터 불러오기

```
category = np.max(y_train) + 1
print(category, '카테고리')
print(len(X_train), '학습용 뉴스 기사')
print(len(X_test), '테스트용 뉴스 기사')
print(X_train[0])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/reuters.npz
2110848/2110848 [==============================] - 0s 0us/step
46 카테고리
8982 학습용 뉴스 기사
2246 테스트용 뉴스 기사
[1, 2, 2, 8, 43, 10, 447, 5, 25, 207, 270, 5, 2, 111, 16, 369, 186, 90, 67, 7, 89, 5, 19, 102, 6, 19, 124, 15, 90, 67, 84,
```

셀 실행 결과

03 데이터 전처리와 모델 학습, 결과 확인

```
X_train = sequence.pad_sequences(X_train, maxlen=100)
X_test = sequence.pad_sequences(X_test, maxlen=100)


y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


model = Sequential()
model.add(Embedding(1000, 100))
model.add(LSTM(100, activation='tanh'))
model.add(Dense(46, activation='softmax'))


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


early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)


history = model.fit(X_train, y_train, batch_size=20, epochs=200, validation_data=(X_test, y_test), callbacks=[early_stopping_callback])


print("₩n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

03 데이터 전처리와 모델 학습, 결과 확인

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)
print("₩n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))
```

```
Epoch 1/200
450/450 [==============================] - 37s 75ms/step - loss: 2.2202 - accuracy: 0.4393 - val_loss: 1.9527 - val_accuracy: 0.5076
Epoch 2/200
450/450 [==============================] - 34s 76ms/step - loss: 1.8110 - accuracy: 0.5335 - val_loss: 1.7408 - val_accuracy: 0.5574
Epoch 3/200
450/450 [==============================] - 34s 75ms/step - loss: 1.6409 - accuracy: 0.5842 - val_loss: 1.5848 - val_accuracy: 0.6077
Epoch 4/200
450/450 [==============================] - 33s 73ms/step - loss: 1.4405 - accuracy: 0.6368 - val_loss: 1.4928 - val_accuracy: 0.6322
Epoch 5/200
450/450 [==============================] - 33s 74ms/step - loss: 1.2664 - accuracy: 0.6815 - val_loss: 1.3300 - val_accuracy: 0.6665
Epoch 6/200
450/450 [==============================] - 34s 75ms/step - loss: 1.1455 - accuracy: 0.7090 - val_loss: 1.2734 - val_accuracy: 0.6754
Epoch 7/200
450/450 [==============================] - 33s 73ms/step - loss: 1.0481 - accuracy: 0.7348 - val_loss: 1.2212 - val_accuracy: 0.6963
Epoch 8/200
450/450 [==============================] - 32s 72ms/step - loss: 0.9551 - accuracy: 0.7604 - val_loss: 1.2005 - val_accuracy: 0.6999
Epoch 9/200
450/450 [==============================] - 34s 75ms/step - loss: 0.8791 - accuracy: 0.7783 - val_loss: 1.1854 - val_accuracy: 0.6986
Epoch 10/200
450/450 [==============================] - 36s 80ms/step - loss: 0.8078 - accuracy: 0.7943 - val_loss: 1.1625 - val_accuracy: 0.7164
Epoch 11/200
450/450 [==============================] - 33s 73ms/step - loss: 0.7547 - accuracy: 0.8086 - val_loss: 1.1639 - val_accuracy: 0.7173
Epoch 12/200
450/450 [==============================] - 33s 74ms/step - loss: 0.7089 - accuracy: 0.8216 - val_loss: 1.1945 - val_accuracy: 0.7110
Epoch 13/200
450/450 [==============================] - 34s 76ms/step - loss: 0.6508 - accuracy: 0.8356 - val_loss: 1.1875 - val_accuracy: 0.7257
Epoch 14/200
450/450 [==============================] - 36s 80ms/step - loss: 0.5987 - accuracy: 0.8527 - val_loss: 1.2048 - val_accuracy: 0.7275
Epoch 15/200
450/450 [==============================] - 33s 74ms/step - loss: 0.5567 - accuracy: 0.8618 - val_loss: 1.2204 - val_accuracy: 0.7257
71/71 [==============================] - 2s 27ms/step - loss: 1.2204 - accuracy: 0.7257

 Test Accuracy: 0.7257
```

셀 실행 결과

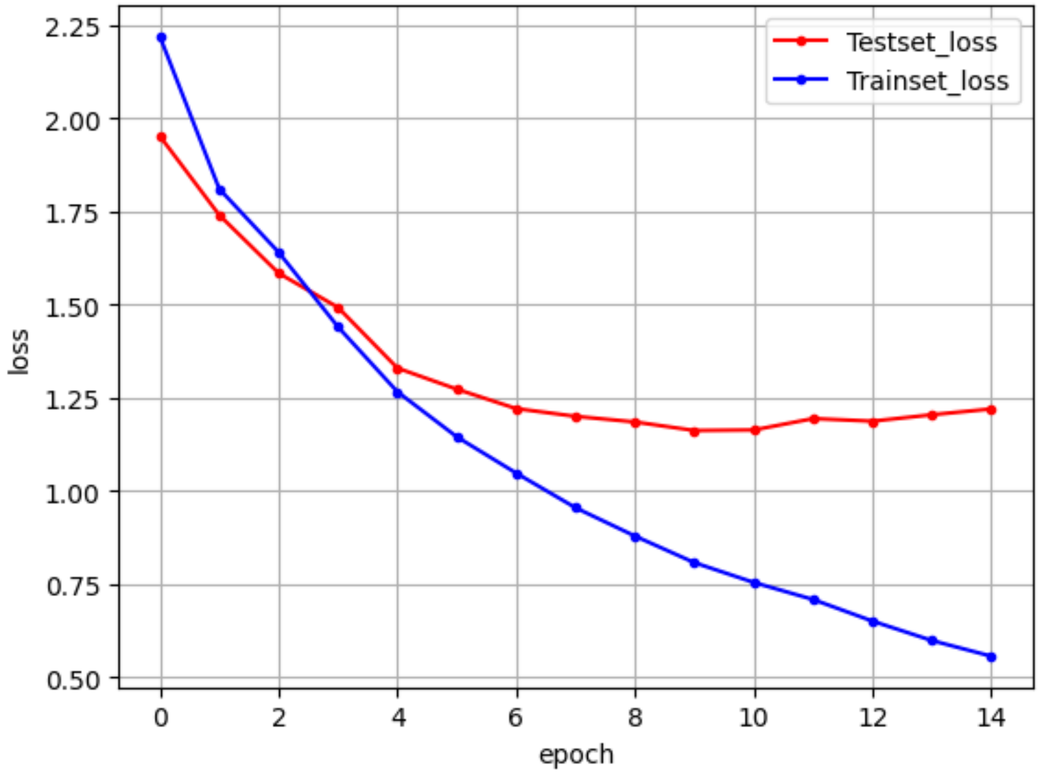
04 결과 시각화하기

```
y_vloss = history.history['val_loss']
y_loss = history.history['loss']


x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')


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

04 결과 시각화하기



셀 실행 결과