자연어처리 과제#4

1. 자료 8-B의 MNIST 숫자 인식 프로그램을 수행하여 자료의 30쪽에 있는 결과와 유사하게 10번까지의 epoch 결과를 제출하라. 정확도가 30쪽에 있는 숫자와 다르게 나타나는 이유를 설명하라.

```
# Loading the MNIST dataset
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

from keras import models
from keras import layers

# Network architecture
network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
# The compilation step
network.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# Preparing image data
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

# Preparing the labels
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
# Fit the model to training data
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
# 중간 레이어 단자수 512개의 실행 결과
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.8755 - loss: 0.4356
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 4s 9ms/step - accuracy: 0.9657 - loss: 0.1143
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9788 - loss: 0.0717
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9862 - loss: 0.0475
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 10ms/step - accuracy: 0.9889 - loss:
0.0368
```

딥러닝 모델의 가중치는 학습 시작 시 랜덤하게 초기화되는데, 초기 가중치가 어떻게 설정되느냐에 따라 학습 과정이 다르게 진행될 수 있으며, 이는 학습 후 정확도에도 영향을 미쳐 실행 시 마다 정확도의 차이가 발생할 수 있다.
또한, 데이터는 매 epoch마다 무작위로 셔플링되는데, 셔플링 과정에서 데이터가 학습되는 순서가 바뀌면 모델이 학습하는 방식에도 영향을 미친다. 이로 인해 학습 결과가 달라지고, 정확도의 차이가 발생할 수 있다.

---

2. 위의 MNIST 프로그램에서 중간 레이어의 단자수를 256과 1024로 수정하여 1번에서 구한 결과와 비교하라. 전체적인 수행시간이 어떻게 달라지는지 비교하라.

```
# Loading the MNIST dataset
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

from keras import models
from keras import layers

# Network architecture
network = models.Sequential()
                    # 중간 레이어의 단자수를 256으로 수정
network.add(layers.Dense(256, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
# The compilation step
network.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# Preparing image data
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

# Preparing the labels
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
# Fit the model to training data
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
# 중간 레이어 단자수 256개의 실행 결과
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.8706 - loss: 0.4740
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9576 - loss: 0.1452
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9728 - loss: 0.0940
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - accuracy: 0.9797 - loss: 0.0681
Epoch 5/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 7ms/step - accuracy: 0.9848 - loss: 0.0503
```

1번에서는 중간 레이어 단자수를 512개로 설정했다. 중간 레이어 단자수를 256개로 줄인 뒤 실행한 결과는 1번에 비해 수행시간이 더 빠른 것을 볼 수 있었다. 1번에서는 epoch 당 3~5초가 소모된 반면, 중간레이어 단자수를 256개로 줄이니 epoch 당 약 2~3초가 소모되었다.

```
# Loading the MNIST dataset
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

from keras import models
from keras import layers

# Network architecture
network = models.Sequential()
                    # 중간 레이어의 단자수를 1024로 수정
network.add(layers.Dense(1024, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
# The compilation step
network.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

# Preparing image data
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255

# Preparing the labels
from keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
# Fit the model to training data
network.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
# 중간 레이어 단자수 1024개의 실행 결과
Epoch 1/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 14ms/step - accuracy: 0.8736 - loss:
0.4243
Epoch 2/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 17ms/step - accuracy: 0.9704 - loss:
0.0984
Epoch 3/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 10s 17ms/step - accuracy: 0.9817 - loss:
0.0610
Epoch 4/5
469/469 ━━━━━━━━━━━━━━━━━━━━ 9s 14ms/step - accuracy: 0.9878 - loss:
0.0397
Epoch 5/5
```

```
469/469 ━━━━━━━━━━━━━━━━━━━━ 8s 16ms/step - accuracy: 0.9908 - loss:
0.0315
```

1번에서는 중간 레이어 단자수를 512개로 설정했다. 중간 레이어 단자수를 1024개로 늘린 뒤 실행한 결과는 1번에 비해 수행 시간이 더 느린 것을 볼 수 있었다. 1번에서는 epoch당 3~5초가 소모된 반면, 중간레이어 단자수를 1024개로 늘리니 epoch 당 약 8~10초가 소모되었다.

---

3. 자료 8-B의 IMDB 영화평을 분류하는 사례를 epoch을 6번 진행하여 수행하라. 정확도와 오차가 어떻게 나타나는지 결과를 제출하라.

```
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
num_words=10000)

import numpy as np
# 입력 텍스트 vectorization
def vectorize_sequences(sequences, dimension=10000):
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1.
  return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')

from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=
['accuracy'])
                          # epochs를 6으로 수정
model.fit(x_train, y_train, epochs=6, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.7343 - loss: 0.5831
Epoch 2/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.8883 - loss: 0.3228
Epoch 3/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 23ms/step - accuracy: 0.9171 - loss: 0.2368
Epoch 4/6
```

```
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9301 - loss: 0.1941
Epoch 5/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9404 - loss: 0.1704
Epoch 6/6
49/49 ━━━━━━━━━━━━━━━━━━━━ 1s 22ms/step - accuracy: 0.9486 - loss: 0.1514
782/782 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8812 - loss: 0.3022
```

자료 8-B의 코드에서 epochs를 6으로 수정하고 실행하여 정확도와 오차 결과를 출력했다.
epcoch를 4에서 6으로 수정했더니 정확도가 줄어드는것을 볼 수 있었다.
이런 현상은 모델이 훈련집합을 여러번 반복 학습하면서 훈련집합에 과잉적합(Overfitting)된 탓에 일어난 것이라고 예상된다.