인공지능 실습 Report

Classifying movie reviews:
a binary classification example 추가 실험

201632230 컴퓨터공학과 이다은

2020. 05. 10

1. 실습문제 정의

layer의 수, layer의 node 수, 다양한 활성화 함수, 다양한 loss 함수, num_words 개수 변경, epochs와 batch_size 변경 등을 통해 model.summary() 추가 결과를 확인한다.

2. 문제해결 방법 제시

① 2개의 hidden layers를 1개 혹은 3개의 hidden layers로 변경하여 검증과 테스트 정도에 어느정도 영향을 미치는지 알아본다.

② 16개의 layer nodes를 32개 혹은 64개의 layer nodes로 변경하여 검증과 테스트 정도에 어느정도 영향을 미치는지 알아본다.

③ relu 활성화 함수 대신 tanh 활성화 함수를 사용해본다.

④ binary_crossentropy loss 함수 대신 mse loss 함수를 사용해본다.

⑤ num_words = 10000을 num_words = 5000으로 변경해본다.

⑥ epochs = 4, batch_size = 512를 epochs = 10, batch_size = 100으로 변경해본다.

각 방법에서 model.summary()로 model이 어떻게 구성되었는지 확인해본다.

3. 문서화된 소스 코드

① 2개의 hidden layers를 1개의 hidden layer로 변경한 후, 모델 훈련과 검증을 진행했다.

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))     # 1개의 hidden layer 사용
model.add(layers.Dense(1, activation='sigmoid'))
```

```
# 훈련 검증(validation)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer = 'rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 174us/step - loss: 0.4898 - acc: 0.7968 - val_loss: 0.3755 - val_acc: 0.8721
Epoch 2/20
15000/15000 [==============================] - 3s 168us/step - loss: 0.3056 - acc: 0.9054 - val_loss: 0.3101 - val_acc: 0.8864
Epoch 3/20
15000/15000 [==============================] - 2s 166us/step - loss: 0.2368 - acc: 0.9234 - val_loss: 0.2850 - val_acc: 0.8908
Epoch 4/20
15000/15000 [==============================] - 2s 165us/step - loss: 0.1917 - acc: 0.9427 - val_loss: 0.2751 - val_acc: 0.8906
Epoch 5/20
15000/15000 [==============================] - 2s 166us/step - loss: 0.1639 - acc: 0.9505 - val_loss: 0.2741 - val_acc: 0.8903
Epoch 6/20
15000/15000 [==============================] - 3s 169us/step - loss: 0.1397 - acc: 0.9585 - val_loss: 0.2854 - val_acc: 0.8875
```

다음으로 모델 재훈련 후, 모델 테스트를 하였다.

```
# 모델 재훈련

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

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

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
25000/25000 [==============================] - 3s 116us/step - loss: 0.4308 - accuracy: 0.8336
Epoch 2/4
25000/25000 [==============================] - 3s 108us/step - loss: 0.2678 - accuracy: 0.9090
Epoch 3/4
25000/25000 [==============================] - 3s 106us/step - loss: 0.2142 - accuracy: 0.9271
Epoch 4/4
25000/25000 [==============================] - 3s 106us/step - loss: 0.1826 - accuracy: 0.9375
25000/25000 [==============================] - 3s 129us/step
```

마지막으로 model.summary()을 이용하여 model에 대해 살펴봤다.

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_22 (Dense)             (None, 16)                160016
_________________________________________________________________
dense_23 (Dense)             (None, 1)                 17
=================================================================
Total params: 160,033
Trainable params: 160,033
Non-trainable params: 0
```

model이 1개의 hidden layer, 1개의 output layer로 구성되어, 2개의 hidden layers를 사용했을 때와는 다른 양상을 보인다. 반면, 각 layer의 nodes 수와 파라미터의 개수는 이전과 동일하다.

② 16개의 layer nodes를 64개의 layer nodes로 변경한 후, 모델 훈련과 검증을 진행했다.

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))      # layer nodes를 16에서 64로 변경
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
# 훈련 검증(validation)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer = 'rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 195us/step - loss: 0.4859 - acc: 0.7791 - val_loss: 0.3255 - val_acc: 0.8816
Epoch 2/20
15000/15000 [==============================] - 3s 189us/step - loss: 0.2621 - acc: 0.9039 - val_loss: 0.2865 - val_acc: 0.8844
Epoch 3/20
15000/15000 [==============================] - 3s 186us/step - loss: 0.1934 - acc: 0.9293 - val_loss: 0.2789 - val_acc: 0.8890
Epoch 4/20
15000/15000 [==============================] - 3s 192us/step - loss: 0.1419 - acc: 0.9484 - val_loss: 0.3001 - val_acc: 0.8855
Epoch 5/20
15000/15000 [==============================] - 3s 191us/step - loss: 0.1167 - acc: 0.9596 - val_loss: 0.3359 - val_acc: 0.8740
Epoch 6/20
15000/15000 [==============================] - 3s 187us/step - loss: 0.0855 - acc: 0.9728 - val_loss: 0.3371 - val_acc: 0.8844
```

다음으로 모델 재훈련 후, 모델 테스트를 하였다.

```
# 모델 재훈련

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

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

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
25000/25000 [==============================] - 3s 139us/step - loss: 0.4205 - accuracy: 0.8119
Epoch 2/4
25000/25000 [==============================] - 3s 132us/step - loss: 0.2374 - accuracy: 0.9096
Epoch 3/4
25000/25000 [==============================] - 3s 126us/step - loss: 0.1842 - accuracy: 0.9309
Epoch 4/4
25000/25000 [==============================] - 3s 127us/step - loss: 0.1440 - accuracy: 0.9469
25000/25000 [==============================] - 4s 146us/step
```

마지막으로 model.summary()을 이용하여 model에 대해 살펴봤다.

```
model.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_33 (Dense)             (None, 64)                640064
_________________________________________________________________
dense_34 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_35 (Dense)             (None, 1)                 65
=================================================================
Total params: 644,289
Trainable params: 644,289
Non-trainable params: 0
```

model의 hidden layers가 각 64개의 nodes 수로 구성되어, 16개의 nodes 수를 사용했을 때와는 다른 양상을 보인다. 또한, 각 layer의 파라미터 개수도 이전보다 크게 증가하였다. 한편, hidden layers와 output layer의 개수는 이전과 동일하다.

③ relu 활성화 함수 대신 tanh 활성화 함수를 사용했다.

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='tanh', input_shape=(10000,)))    # relu 활성화 함수 대신 tanh 활성화 함수 사용
model.add(layers.Dense(16, activation='tanh'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
# 훈련 검증(validation)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer = 'rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 180us/step - loss: 0.4817 - acc: 0.7955 - val_loss: 0.3984 - val_acc: 0.8345
Epoch 2/20
15000/15000 [==============================] - 3s 168us/step - loss: 0.2774 - acc: 0.9061 - val_loss: 0.2920 - val_acc: 0.8839
Epoch 3/20
15000/15000 [==============================] - 2s 165us/step - loss: 0.1940 - acc: 0.9363 - val_loss: 0.3201 - val_acc: 0.8688
Epoch 4/20
15000/15000 [==============================] - 2s 164us/step - loss: 0.1500 - acc: 0.9481 - val_loss: 0.2860 - val_acc: 0.8817
Epoch 5/20
15000/15000 [==============================] - 2s 166us/step - loss: 0.1109 - acc: 0.9639 - val_loss: 0.3197 - val_acc: 0.8786
Epoch 6/20
```

다음으로 모델 재훈련 후, 모델 테스트를 하였다.

```
# 모델 재훈련

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

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

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
25000/25000 [==============================] - 3s 110us/step - loss: 0.4371 - accuracy: 0.8252
Epoch 2/4
25000/25000 [==============================] - 3s 110us/step - loss: 0.2449 - accuracy: 0.9142
Epoch 3/4
25000/25000 [==============================] - 3s 123us/step - loss: 0.1815 - accuracy: 0.9338
Epoch 4/4
25000/25000 [==============================] - 3s 115us/step - loss: 0.1509 - accuracy: 0.9463
25000/25000 [==============================] - 3s 119us/step
```

마지막으로 model.summary()을 이용하여 model에 대해 살펴봤다.

```
model.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_39 (Dense)             (None, 16)                160016
_________________________________________________________________
dense_40 (Dense)             (None, 16)                272
_________________________________________________________________
dense_41 (Dense)             (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
```

model의 relu 활성화 함수가 tanh 활성화 함수로 변경되었기에 model의 layers, nodes 개수와 파라미터의 개수에는 변화가 없다. 모두 이전과 동일한 값을 가지고 있다.

④ binary_crossentropy loss 함수 대신 mse loss 함수를 사용했다.

```
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'))
```

```
# 훈련 검증(validation)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer = 'rmsprop',
              loss='mse',       # binary_crossentropy loss 함수 대신 mse loss 함수 사용
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 187us/step - loss: 0.0437 - acc: 0.9447 - val_loss: 0.0418 - val_acc: 0.9479
Epoch 2/20
15000/15000 [==============================] - 3s 173us/step - loss: 0.0332 - acc: 0.9613 - val_loss: 0.0475 - val_acc: 0.9396
Epoch 3/20
15000/15000 [==============================] - 3s 169us/step - loss: 0.0273 - acc: 0.9717 - val_loss: 0.0526 - val_acc: 0.9309
Epoch 4/20
15000/15000 [==============================] - 3s 175us/step - loss: 0.0232 - acc: 0.9763 - val_loss: 0.0540 - val_acc: 0.9272
Epoch 5/20
15000/15000 [==============================] - 3s 174us/step - loss: 0.0192 - acc: 0.9825 - val_loss: 0.0598 - val_acc: 0.9218
Epoch 6/20
15000/15000 [==============================] - 3s 181us/step - loss: 0.0165 - acc: 0.9850 - val_loss: 0.0612 - val_acc: 0.9204
```

다음으로 모델 재훈련 후, 모델 테스트를 하였다.

```
# 모델 재훈련

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='mse',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
25000/25000 [==============================] - 3s 122us/step - loss: 0.1471 - accuracy: 0.8230
Epoch 2/4
25000/25000 [==============================] - 3s 111us/step - loss: 0.0771 - accuracy: 0.9113
Epoch 3/4
25000/25000 [==============================] - 3s 112us/step - loss: 0.0580 - accuracy: 0.9317
Epoch 4/4
25000/25000 [==============================] - 3s 111us/step - loss: 0.0475 - accuracy: 0.9449
25000/25000 [==============================] - 3s 129us/step
```

마지막으로 model.summary()을 이용하여 model에 대해 살펴봤다.

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 16)                160016
_________________________________________________________________
dense_8 (Dense)              (None, 16)                272
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
```

model의 binary_crossentropy loss 함수가 mse loss로 변경되었기에 model의 layers, nodes 개수와 파라미터의 개수에는 변화가 없다. 모두 이전과 동일한 값을 가지고 있다.

⑤ num_words = 10000을 num_words = 5000으로 변경하였다.

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words = 5000)      # num_words = 10000에서 5000으로 변경
```

```
max([max(sequence) for sequence in train_data])          # 단어 사전의 최대 index도 9999에서 4999로 변경됨
```

```
4999
```

```
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'))
```

```
# 훈련 검증(validation)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer = 'rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 214us/step - loss: 0.5057 - acc: 0.7976 - val_loss: 0.3954 - val_acc: 0.8535
Epoch 2/20
15000/15000 [==============================] - 3s 171us/step - loss: 0.3203 - acc: 0.8913 - val_loss: 0.3086 - val_acc: 0.8843
Epoch 3/20
15000/15000 [==============================] - 3s 170us/step - loss: 0.2513 - acc: 0.9117 - val_loss: 0.2979 - val_acc: 0.8789
Epoch 4/20
15000/15000 [==============================] - 3s 176us/step - loss: 0.2155 - acc: 0.9218 - val_loss: 0.2937 - val_acc: 0.8801
Epoch 5/20
15000/15000 [==============================] - 3s 177us/step - loss: 0.1883 - acc: 0.9311 - val_loss: 0.2855 - val_acc: 0.8837
Epoch 6/20
15000/15000 [==============================] - 3s 173us/step - loss: 0.1697 - acc: 0.9392 - val_loss: 0.2954 - val_acc: 0.8813
```

다음으로 모델 재훈련 후, 모델 테스트를 하였다.

```
# 모델 재훈련

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'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/4
25000/25000 [==============================] - 3s 116us/step - loss: 0.4560 - accuracy: 0.8178
Epoch 2/4
25000/25000 [==============================] - 3s 114us/step - loss: 0.2793 - accuracy: 0.8986
Epoch 3/4
25000/25000 [==============================] - 3s 117us/step - loss: 0.2325 - accuracy: 0.9132
Epoch 4/4
25000/25000 [==============================] - 3s 119us/step - loss: 0.2084 - accuracy: 0.9210
25000/25000 [==============================] - 3s 124us/step
```

마지막으로 model.summary()을 이용하여 model에 대해 살펴봤다.

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_19 (Dense)             (None, 16)                160016
_________________________________________________________________
dense_20 (Dense)             (None, 16)                272
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
```

IMDB 데이터의 num_words 개수가 10000개에서 5000개로 변경되었기에 model의 layers, nodes 개수와 파라미터 개수에는 변화가 없다. 모두 이전과 동일한 값을 가진다.

⑥ 모델 훈련과 검증은 epochs = 20, batch_size = 512를 유지한 채 진행했다.

```
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'))
```

```
# 훈련 검증(validation)

x_val = x_train[:10000]
partial_x_train = x_train[10000:]
y_val = y_train[:10000]
partial_y_train = y_train[10000:]
```

```
model.compile(optimizer = 'rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs = 20,
                    batch_size = 512,
                    validation_data=(x_val, y_val))
```

```
Train on 15000 samples, validate on 10000 samples
Epoch 1/20
15000/15000 [==============================] - 3s 194us/step - loss: 0.0334 - acc: 0.9889 - val_loss: 0.0401 - val_acc: 0.9901
Epoch 2/20
15000/15000 [==============================] - 3s 170us/step - loss: 0.0199 - acc: 0.9950 - val_loss: 0.0446 - val_acc: 0.9876
Epoch 3/20
15000/15000 [==============================] - 3s 178us/step - loss: 0.0142 - acc: 0.9970 - val_loss: 0.0534 - val_acc: 0.9855
Epoch 4/20
15000/15000 [==============================] - 3s 175us/step - loss: 0.0095 - acc: 0.9980 - val_loss: 0.0589 - val_acc: 0.9834
Epoch 5/20
15000/15000 [==============================] - 3s 180us/step - loss: 0.0062 - acc: 0.9987 - val_loss: 0.0783 - val_acc: 0.9785
Epoch 6/20
15000/15000 [==============================] - 3s 173us/step - loss: 0.0040 - acc: 0.9991 - val_loss: 0.0931 - val_acc: 0.9757
```

모델 재훈련은 epochs = 4, batch_size = 512를 epochs = 10, batch_size = 100으로 변경한 뒤 진행했다. 모델 재훈련 후에는 모델 테스트를 하였다.

```
# 모델 재훈련

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'])

model.fit(x_train, y_train, epochs=10, batch_size=100)            # epochs = 40에서 10으로, batch_size = 512에서 100으로 변경
results = model.evaluate(x_test, y_test)
```

```
Epoch 1/10
25000/25000 [==============================] - 3s 127us/step - loss: 0.3415 - accuracy: 0.8640
Epoch 2/10
25000/25000 [==============================] - 3s 125us/step - loss: 0.2069 - accuracy: 0.9208
Epoch 3/10
25000/25000 [==============================] - 3s 124us/step - loss: 0.1678 - accuracy: 0.9355
Epoch 4/10
25000/25000 [==============================] - 3s 127us/step - loss: 0.1417 - accuracy: 0.9486
Epoch 5/10
25000/25000 [==============================] - 3s 126us/step - loss: 0.1177 - accuracy: 0.9573
Epoch 6/10
25000/25000 [==============================] - 3s 127us/step - loss: 0.0977 - accuracy: 0.9650
Epoch 7/10
25000/25000 [==============================] - 3s 125us/step - loss: 0.0784 - accuracy: 0.9732
Epoch 8/10
25000/25000 [==============================] - 3s 126us/step - loss: 0.0621 - accuracy: 0.9790
Epoch 9/10
25000/25000 [==============================] - 3s 124us/step - loss: 0.0468 - accuracy: 0.9850
Epoch 10/10
25000/25000 [==============================] - 3s 123us/step - loss: 0.0356 - accuracy: 0.9894
25000/25000 [==============================] - 3s 125us/step
```

마지막으로 model.summary()을 이용하여 model에 대해 살펴봤다.

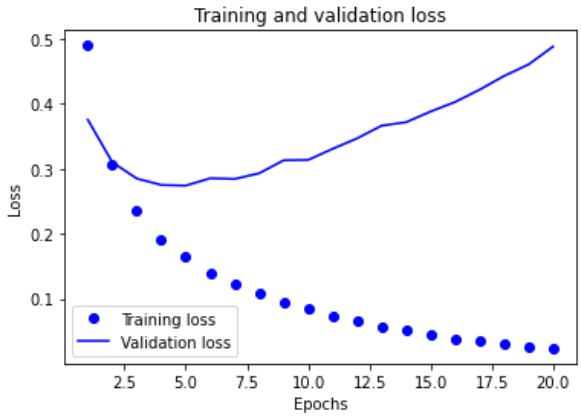
```
model.summary()
```

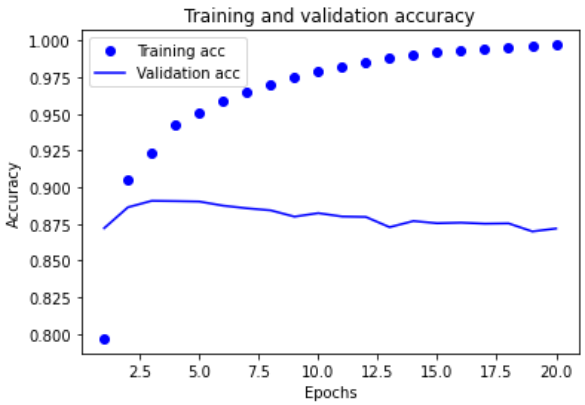
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_34 (Dense)             (None, 16)                160016
_________________________________________________________________
dense_35 (Dense)             (None, 16)                272
_________________________________________________________________
dense_36 (Dense)             (None, 1)                 17
=================================================================
Total params: 160,305
Trainable params: 160,305
Non-trainable params: 0
```

model 훈련의 epochs 횟수가 20회에서 10회로, batch_size 개수가 512개에서 100개로 변경되었기에 model의 layers, nodes 개수와 파라미터 개수에는 변화가 없다. 모두 이전과 동일한 값을 가지고 있다.

4. 실행 결과

① 검증 정도 확인을 위해 matplotlib 라이브러리를 이용하여 훈련과 검증의 loss, accuracy를 그래프로 나타냈다.





테스트 정도 확인을 위해 evaluate()와 predict()를 이용하여 테스트 loss, accuracy와 새로운 데이터 예측 결과를 살펴보았다.

```
results
```

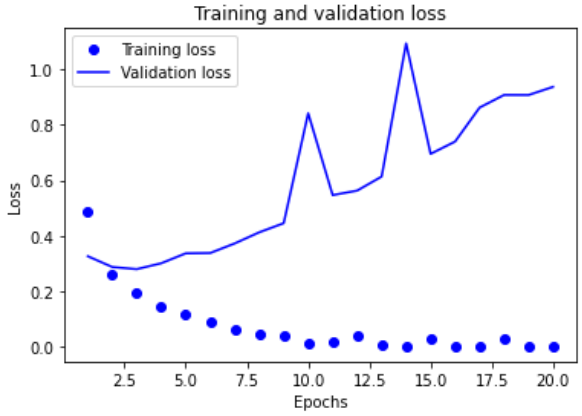
```
[0.2852894259548187, 0.8845199942588806]
```

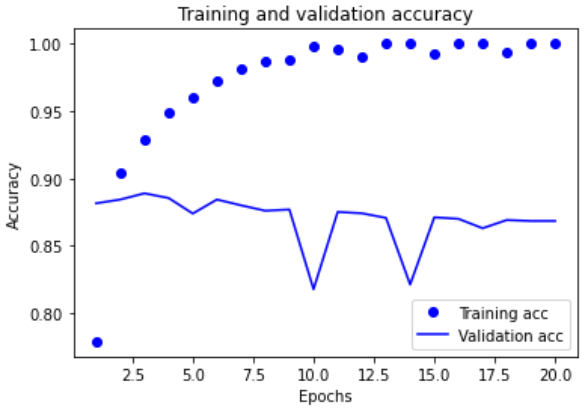
```
# 새로운 데이터 예측

model.predict(x_test)
```

```
array([[0.21720901],
       [0.9993044 ],
       [0.85337913],
       ...,
       [0.12502757],
       [0.08821565],
       [0.45390597]], dtype=float32)
```

② 검증 정도 확인을 위해 matplotlib 라이브러리를 이용하여 훈련과 검증의 loss, accuracy를 그래프로 나타냈다.





테스트 정도 확인을 위해 evaluate()와 predict()를 이용하여 테스트 loss, accuracy와 새로운 데이터 예측 결과를 살펴보았다.

```
results
```

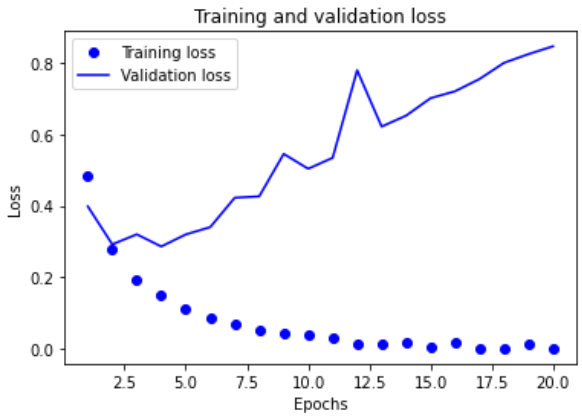
```
[0.3248141668653488, 0.8800399899482727]
```

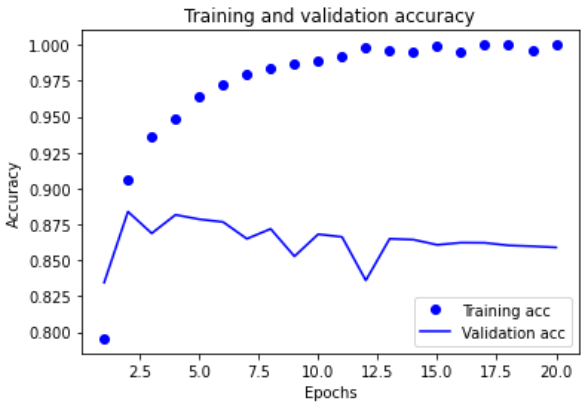
```
# 새로운 데이터 예측

model.predict(x_test)
```

```
array([[0.10874286],
       [0.99989486],
       [0.9341779 ],
       ...,
       [0.09210148],
       [0.05681401],
       [0.6903446 ]], dtype=float32)
```

③ 검증 정도 확인을 위해 matplotlib 라이브러리를 이용하여 훈련과 검증의 loss, accuracy를 그래프로 나타냈다.





테스트 정도 확인을 위해 evaluate()와 predict()를 이용하여 테스트 loss, accuracy와 새로운 데이터 예측 결과를 살펴보았다.

```
results
```

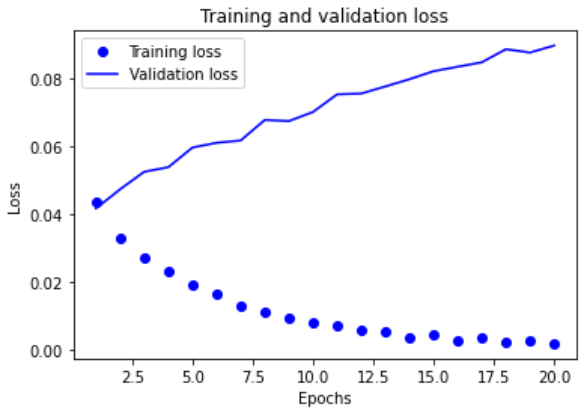
```
[0.3410276456308365, 0.8704400062561035]
```

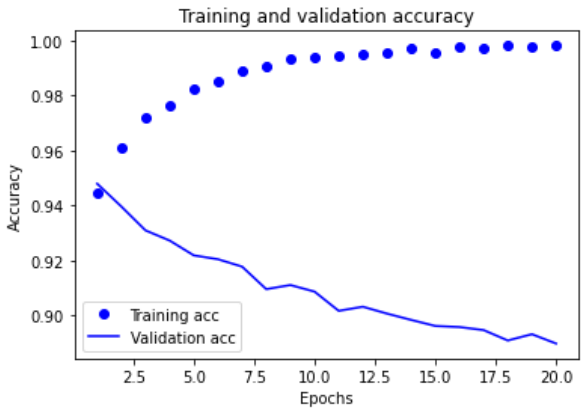
```
# 새로운 데이터 예측

model.predict(x_test)
```

```
array([[0.04260817],
       [0.99700916],
       [0.59970343],
       ...,
       [0.03949949],
       [0.02693883],
       [0.41552594]], dtype=float32)
```

④ 검증 정도 확인을 위해 matplotlib 라이브러리를 이용하여 훈련과 검증의 loss, accuracy를 그래프로 나타냈다.





테스트 정도 확인을 위해 evaluate()와 predict()를 이용하여 테스트 loss, accuracy와 새로운 데이터 예측 결과를 살펴보았다.

```
results
```

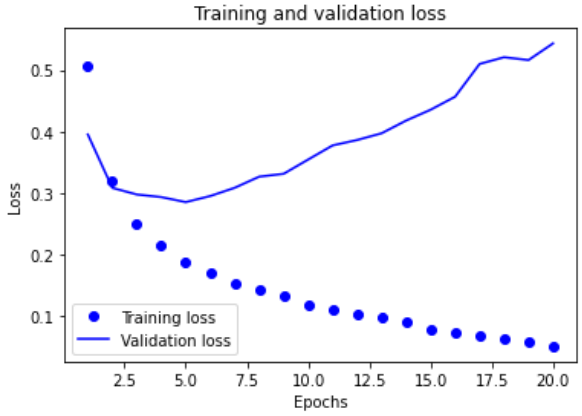
```
[0.09192302491903305, 0.8736400008201599]
```

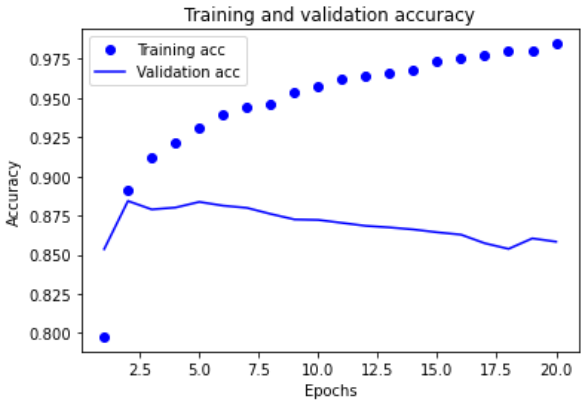
```
# 새로운 데이터 예측

model.predict(x_test)
```

```
array([[0.19204655],
       [0.99989367],
       [0.9634111 ],
       ...,
       [0.23680055],
       [0.18306726],
       [0.7873375 ]], dtype=float32)
```

⑤ 검증 정도 확인을 위해 matplotlib 라이브러리를 이용하여 훈련과 검증의 loss, accuracy를 그래프로 나타냈다.





테스트 정도 확인을 위해 evaluate()와 predict()를 이용하여 테스트 loss, accuracy와 새로운 데이터 예측 결과를 살펴보았다.

```
results
```

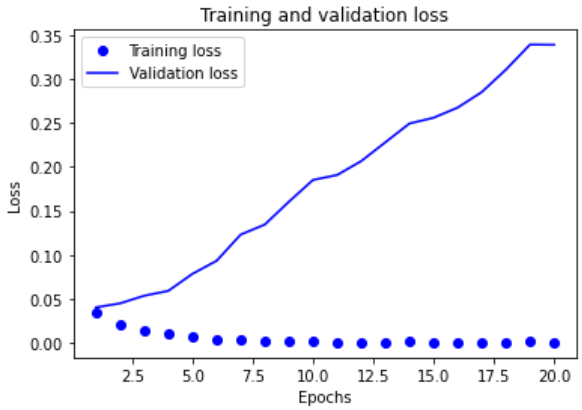
```
[0.2875643102645874, 0.8829200267791748]
```

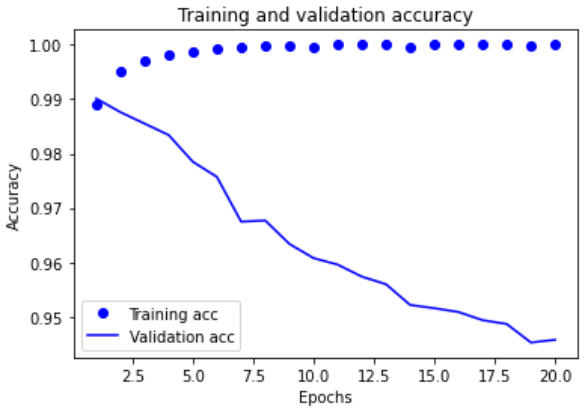
```
# 새로운 데이터 예측

model.predict(x_test)
```

```
array([[0.1665585 ],
       [0.99985516],
       [0.7191841 ],
       ...,
       [0.08833894],
       [0.08086276],
       [0.54920524]], dtype=float32)
```

⑥ 검증 정도 확인을 위해 matplotlib 라이브러리를 이용하여 훈련과 검증의 loss, accuracy를 그래프로 나타냈다.





테스트 정도 확인을 위해 evaluate()와 predict()를 이용하여 테스트 loss, accuracy와 새로운 데이터 예측 결과를 살펴보았다.

```
results
```

```
[0.7714953752541542, 0.8546800017356873]
```

```
# 새로운 데이터 예측

model.predict(x_test)
```

```
array([[0.24075255],
       [1.        ],
       [0.99999714],
       ...,
       [0.10805956],
       [0.00545931],
       [0.9407409 ]], dtype=float32)
```

5. 실행 결과에 대한 설명 및 검토

① 훈련 loss와 accuracy는 2개의 hidden layers를 사용했을 때와 비슷한 양상을 나타냈다. 한편, 검증 loss와 accuracy는 2개의 layers를 사용했을 때보다 그래프 기울기의 변화 정도가 완만했다. 이런 경우, 그래프가 역전되는 순간이 뚜렷이 드러나지 않아 훈련에 적절한 epochs가 어느 정도인지 찾기 어렵다. 한편, model 재훈련시 이전 model 재훈련의 epochs를 그대로 유지한 이유 때문인지 테스트 loss와 accuracy, 새 데이터 예측에는 변화가 적었다. 오히려 2개의 hidden layers를 사용했을 때보다 loss는 약간 늘어났고, accuracy는 약간 줄어들었다.

② 훈련 loss와 accuracy는 16개의 layer nodes를 사용했을 때보다 그래프 기울기에 미세한 변화가 나타났다. 이는 epochs가 늘어날수록 더 잘 드러났다. 한편, 검증 loss와 accuracy는 16개의 layer nodes를 사용했을 때보다 그래프 기울기가 훨씬 급변하는 모습을 보였다. 이는 epochs의 중간에서 확연했다. 이런 경우, 그래프가 역전되는 순간이 뚜렷해도 급변의 순간에 loss는 증가, 감소를 반복하고 accuracy는 감소, 증가를 반복하기에 훈련에 적절한 epochs가 어느 정도인지 알기 어렵다. 한편, model 재훈련시 이전 model 재훈련의 epochs를 그대로 유지한 이유 때문인지 테스트 loss와 accuracy, 새 데이터 예측에는 변화가 거의 없었다. 이전과 유사한 테스트 결과를 볼 수 있었다.

③ 훈련 loss와 accuracy는 relu 활성화 함수를 사용했을 때보다 그래프 기울기에 미세한 변화가 나타났다. 이는 epochs가 늘어날수록 잘 드러났다. 한편, 검증 loss와 accuracy는 relu 활성화 함수를 사용했을 때보다 그래프 기울기가 약간 급변하는 모습을 보였다. 이는 epochs의 처음과 중간에서 확연했다. 이런 경우, 그래프가 역전하는 순간이 epochs의 처음에 뚜렷하여 훈련에 적절한 epochs를 결정하기 용이하다. 한편, model 재훈련시 이전 model 재훈련의 epochs를 그대로 유지했지만 테스트 loss와 accuracy, 새로운 데이터 예측에 작은 변화가 있었다. 이전보다 테스트 loss와 accuracy는 각각 증가하고 감소했다. 하지만, 이전보다 새로운 데이터 예측 결과는 좋아졌다.

④ 훈련 loss와 accuracy는 binary_crossentropy loss 함수를 사용했을 때보다 각각 처음부터 낮고 높았다. binary_crossentropy loss 함수 사용시 첫번째 epoch에서 대략 50%의 loss와 77%의 accuracy를 보였으나, mse loss 함수에서는 첫번째 epoch에서 약 4%의 loss와 94%의 accuracy가 산출됐다. 검증 loss와 accuracy에서도 훈련 데이터와 비슷한 결과가 나타났다. 하지만, 검증 loss와 accuracy는 첫번째 epoch 이후 마지막 epoch까지 지속적으로 증가하고 감소하는 모습이 드러났다. 한편, 테스트 loss는 약 9%의 낮은 수치를 보인 반면, accuracy는 87%의 그리 높지 않은 수치를 띄었다. 오히려 binary_crossentropy loss 함수를 사용했을 때의 accuracy가 더 높았다. 이는 model이 훈련 데이터에 overfitting되었다고 볼 수 있으며, 새로운 데이터에 대한 정확한 예측을 하기 어렵다고 해석할 수 있다.

⑤ 훈련 loss와 accuracy는 num_words가 10000개일 때와 비슷한 양상을 보였다. 한편, 검증 loss와 accuracy는 num_words가 10000개일 때보다 변화되는 모습이 달랐다. 검증 loss는 두번째 epoch 이후 감소하였고, accuracy는 2번째 epoch 이후 감소하다가 3번째 epoch에서 다시 증가하였다. 그러나, 7번째 epoch부터 다시 점차 감소하게 되었다. 이는 모델 훈련에 적합한 epochs를 결정하기 어렵게 만들고 테스트 결과에도 영향을 미친다. 한편, model 재훈련의 epochs를 이전 model 재훈련의 epochs와 동일하게 설정한 이유 때문인지 테스트 loss와 accuracy, 새 데이터 예측에는 변화가 적었다. 테스트 loss는 약 1%의 차이가 있었고, accuracy는 차이가 거의 없었다.

⑥ 훈련과 검증은 epochs, batch_size 값의 변경없이 동일하게 이루어졌다. 따라서, 4번째 epoch 이후에 loss는 증가, accuracy는 감소하는 모습이 나타났다. 한편, model 재훈련의 epochs는 4에서 10으로, batch_size는 512에서 100으로 변경돼 테스트 결과도 달라졌다. 특히, 테스트 loss의 수치가 급증하였다. 값 변경 전에는 약 29%였으나, 값 변경 후에는 약 77%까지 늘어났다. 테스트 accuracy의 수치는 값 변화 전과 후가 비슷하게 나타났다. 이런 경우, model 재훈련의 epochs 조절에 실패하여 model의 overfitting을 해결하지 못했다고 볼 수 있다. 결국, model은 새 데이터에 대한 정확한 예측을 하기 어렵게 된다.

6. 실습을 통한 이해 및 개선 방안

① 2개의 hidden layers에서 1개의 hidden layer로 개수를 감소하면 검증 loss와 accuracy의 변화가 완만하게 된다. 이는 훈련에 적절한 epochs 결정을 어렵게 만든다. epochs는 테스트 loss와 accuracy, 새 데이터 예측 결과에도 영향을 미친다. 결국, 더 나은 테스트 결과를 위해 기존 hidden layers 개수를 유지하거나 epochs 값을 변경할 필요가 있다.

② 16개의 layer nodes에서 64개의 layer nodes로 개수를 증가하면 검증 loss와 accuracy의 변화가 가파르게 된다. 이는 훈련에 적절한 epochs의 결정을 어렵게 만든다. epochs는 테스트 결과에 영향을 미치기에 더 나은 결과를 위하여 기존 layer nodes 수를 유지하거나 epochs의

값을 변경해야 한다.

③ relu 활성화 함수 대신 tanh 활성화 함수를 사용하면 검증 loss와 accuracy의 변화가 가파르게 된다. 하지만, 이는 처음부터 뚜렷한 양상을 보이기에 적당한 수준의 epochs의 값 결정에 도움을 준다. 따라서, relu 활성화 함수가 아닌 tanh 활성화 함수를 이용하여도 괜찮은 테스트 결과를 얻을 수 있다.

④ binary_crossentropy loss 함수 대신 mse loss 함수를 사용하면 검증 loss와 accuracy는 첫번째 epoch부터 낮고 높은 모습을 보인다. 훈련 데이터와 검증 데이터 사이의 격차가 크기 때문에 model이 훈련 데이터에 overfitting 되었다고 해석 가능하다. 결국, 새로운 데이터에 대한 예측율을 높이기 위해 기존 binary_crossentropy loss 함수를 유지하거나 epochs 값을 변경할 필요가 있다.

⑤ num_words의 개수를 10000개가 아닌 5000개를 사용하면 검증 loss와 accuracy는 매 epoch마다 다르게 변화하는 모습을 보인다. 검증 loss는 accuracy에 비해 변화가 적지만, accuracy는 2번째~7번째 epochs와 18번째~20번째 epochs에서 증가와 감소를 반복한다. 이는 훈련에 적절한 epochs 선택을 힘들게 만든다. epochs는 테스트에도 영향을 주기에 새로운 데이터에 대한 예측율을 높이기 위해 기존 단어사전의 개수를 유지하거나 검증 loss의 변화를 기준으로 epochs의 값을 변경해야 된다.

⑥ 훈련과 검증의 epochs, batch_size는 유지되어 각각의 loss, accuracy에는 변화가 없다. 한편, 테스트의 epochs는 4에서 10으로, batch_size는 512에서 100으로 변경되어 테스트 결과가 다르게 나오는 걸 알 수 있다. 재훈련 loss와 accuracy는 매 epoch마다 감소하고 증가한다. 하지만, 최종 테스트 loss는 약 77%의 높은 수치를 보인다. 테스트 accuracy는 이전과 거의 동일한 약 85%의 수치를 가진다. 결국, 테스트 loss가 높기 때문에 새로운 데이터 예측 결과를 신뢰하기 어려워진다. 따라서, 정확한 새 데이터 예측을 위해 훈련과 검증에서 산출되는 결과로 적정 수준의 epochs를 결정해야 한다. 이를 바탕으로 model 재훈련 후 테스트를 진행하면 신뢰도 높은 데이터 예측 결과를 얻을 수 있다.

7. 결론

hidden layers의 수, layer nodes의 수, 활성화 함수, loss 함수, num_words의 개수, epochs와 batch_size의 범위는 모두 model 훈련, 검증, 테스트에 영향에 끼치는 요소들이다.

2개의 hidden layers가 1개의 hidden layer로 변경되면 훈련 loss와 accuracy에는 큰 변화가 없었지만, 검증 loss와 accuracy는 그래프 기울기가 완만해지는 변화가 있었다. 한편, 테스트 loss와 accuracy, 새 데이터 예측 결과에는 변화가 적었다. 오히려 테스트 loss는 약간 늘었고,

accuracy는 약간 줄었다.

16개의 layer nodes가 64개의 layer nodes로 변경되면 훈련 loss와 accuracy에 미세한 변화가 나타났는데, 이는 epochs 값이 커질수록 더 뚜렷했다. 검증 loss와 accuracy에는 급변하는 변화가 있었는데, 이는 epochs 값 중간에서 확연했다. 한편, 테스트 loss와 accuracy, 새로운 데이터 예측 결과에는 큰 변화가 없었고, 이전과 비슷한 결과를 나타냈다.

relu 활성화 함수 대신 tanh 활성화 함수를 사용하면 훈련 loss와 accuracy에 조금의 변화가 보였고, 이는 epochs 값이 커질수록 더 드러났다. 검증 loss와 accuracy는 그래프 기울기가 약간 급변하는 모습이 나타났다. 이는 epochs 값의 처음과 중간에서 확연했다. 한편, 테스트 loss와 accuracy는 각각 증가하고 감소했고, 새 데이터 예측 결과는 이전보다 좋게 나왔다.

binary_crossentropy loss 함수 대신 mse loss 함수를 사용하면 훈련 loss와 accuracy 값이 각각 처음부터 낮고 높게 나타났다. 검증 데이터에서도 유사한 결과가 보였다. 그러나, loss는 첫번째 epoch 이후 마지막 epoch까지 지속적으로 증가하고, accuracy는 첫번째 epoch에서 마지막 epoch까지 지속적으로 감소했다. 한편, 테스트 loss는 낮았으나 accuracy는 이전보다 오히려 적은 수치를 드러냈다. 결국, model이 훈련 데이터에 overfitting된 모습을 초래했다.

num_words가 10000개에서 5000개로 변경되면 훈련 데이터는 이전과 비슷한 양상을 보였고, 검증 데이터는 다른 양상을 나타냈다. 검증 loss는 2번째 epoch 이후 감소했지만, accuracy는 2번째 epoch 이후 감소하다가 3번째 epoch 이후 증가하고, 7번째 epoch부터 다시 감소했다. 한편, 테스트 loss와 accuracy, 새 데이터 예측 결과에는 별다른 차이가 없었다.

model 재훈련의 epochs를 4에서 10으로, batch_size를 512에서 100으로 변경하면 테스트 결과에 변화가 생겼다. 특히, 테스트 loss가 급증하여 새 데이터 예측 결과를 신뢰하기 어렵게 만들었다. 테스트 accuracy는 이전과 비슷했다. 결국, model 재훈련의 epochs 범위의 조절에 실패하여 기존의 overfitting 문제가 지속되었다.

결과적으로, 요소들의 범위 혹은 파라미터 등을 다양하게 변경하는 시도를 통해 경험을 쌓고, 가장 최적의 값을 찾아 적용해야 한다. 이로써 더 수준 높은 테스트 결과를 얻어낼 수 있다.