

인공지능 프로그래밍 경진대회



텐서플로우를 이용한 영화 리뷰 텍스트 이진 분류 인공지능

컴퓨터정보공학과 PE반

팀명 : 한수와 아이들

20190747 기호정

20192635 김한수

20161887 염경훈

목차

1. 개요

- 1) 팀명과 인원 구성
- 2) 딥러닝 목적

2. 회귀와 분류

3. 데이터 저장과 전처리

- 1) 데이터 읽어 오기
- 2) 데이터 수, 속성 수 등 소개
- 3) 훈련과 테스트 데이터 나누기
- 4) 정규화 등의 전처리

4. 딥러닝 모델

- 1) 모델 종류, 입력, 중간(은닉), 출력 층, 패러미터 수
- 2) 옵티마이저, 손실함수

5. 데이터와 훈련과정, 예측 결과의 시각화

- 1) 모델 훈련
- 2) 모델 평가
- 3) 정확도와 손실 그래프 그리기

6. Callback 함수 사용

1. 개요

본 모델은 텐서플로우에서 제공하는 영화 리뷰 텍스트 이진 분류 예시를 학습하며 작성하였음을 밝힙니다.

1) 팀명과 인원구성

팀명 : 한수와 아이들

인원 구성 : 총 3명

기호정 20190747

김한수 20192635 ✓ 팀장

염경훈 20161887

2) 딥러닝 목적

이 프로젝트는 영화 리뷰 텍스트의 연관성을 모델에 학습시켜 리뷰가 긍정적인지 부정적인지 판별하는 것이 목표입니다.

2. 회귀와 분류

이 프로젝트는 영화 리뷰(review) 텍스트를 긍정(positive) 또는 부정(negative)으로 분류합니다.

즉, 이진 분류입니다.

3. 데이터 저장과 전처리

1) 데이터 읽어 오기

IMDB 데이터셋 다운로드

먼저, 아래 코드를 입력하여 IMDB 데이터셋을 다운로드합니다.

```
[ ] 1 imdb = keras.datasets.imdb
    2
    3 (train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17465344/17464789 [=====] - 0s 0us/step
```

매개변수 num_words=10000은 훈련 데이터에서 가장 많이 등장하는 상위 10,000개의 단어를 선택합니다. 데이터 크기를 적당하게 유지하기 위해 드물게 등장하는 단어는 제외합니다.

2) 데이터 수, 속성 수 등 소개

데이터셋으로 인터넷 영화 데이터베이스(Internet Movie Database)에서 수집한 50,000개의 영화 리뷰 텍스트를 담은 IMDB 데이터셋을 사용했습니다. 이 데이터셋은 텐서플로에서 제공하며, **25,000개 리뷰는 훈련용으로, 25,000개는 테스트용으로 나뉘어져 있습니다.** 그리고 훈련 세트와 테스트 세트의 긍정적인 리뷰와 부정적인 리뷰의 개수가 동일하도록 균형이 잡혀있습니다.

리뷰 데이터(단어의 시퀀스(sequence))는 미리 전처리되어 정수 시퀀스로 변환되어 있습니다. 각 정수는 어휘 사전에 있는 특정 단어를 의미합니다.

위에서 말했듯 데이터셋의 샘플은 전처리된 정수 배열입니다.

이 정수는 영화 리뷰에 나오는 단어를 나타냅니다. 레이블(label)은 정수 0 또는 1입니다. 0은 부정적인 리뷰, 1은 긍정적인 리뷰입니다.

첫 번째 리뷰를 확인해보면 다음과 같습니다.

리뷰 텍스트는 어휘 사전의 특정 단어를 나타내는 정수로 변환되어 있습니다. 첫 번째 리뷰를 확인해 보겠습니다:

```
[ ] 1 print(train_data[0])

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9
```

위와 같이 데이터가 정수로 되어있으면 어떤 텍스트로 되어있는지 감이 안 오기 때문에 정수를 단어로 변환해보겠습니다.

정수를 단어로 다시 변환하기

정수를 다시 정수를 다시 텍스트로 변환하는 방법이 있다면 유용할 것입니다. 여기에서는 정수와 문자열을 매핑한 딕셔너리(dictionary) 객체에 질의하는 헬퍼(helper) 함수를 만들겠습니다:

```
[25] 1 # 단어와 정수 인덱스를 매핑한 딕셔너리
      2 word_index = imdb.get_word_index()
      3
      4 # 처음 몇 개 인덱스는 사전에 정의되어 있습니다
      5 word_index = {k:(v+3) for k,v in word_index.items()}
      6 word_index["<PAD>"] = 0
      7 word_index["<START>"] = 1
      8 word_index["<UNK>"] = 2 # unknown
      9 word_index["<UNUSED>"] = 3
      10
      11 reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
      12
      13 def decode_review(text):
      14     return ' '.join([reverse_word_index.get(i, '?') for i in text])
```

이제 `decode_review` 함수를 사용해 첫 번째 리뷰 텍스트를 출력할 수 있습니다:

위에서 만든 헬퍼 함수를 이용해 첫 번째 리뷰 텍스트를 출력해보면 아래와 같은 결과가 나옵니다.

```
[26] 1 decode_review(train_data[0])

'<START> this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert <UNK> is an amazing actor and now the same being director <UNK> father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for <UNK> and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also <UNK> to the two little boy's that played the <UNK> of norman and paul they were just brilliant children are often left out of the <UNK> list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what...'
```

3) 훈련과 테스트 데이터 나누기

훈련과 테스트 데이터 나누기

우리는 이전에 데이터를 영화리뷰 데이터를 다운로드할 때 훈련 샘플과 테스트 샘플이 각각 25000개로 나누어져있다고 했었습니다. 아래 코드로 한 번 확인해보겠습니다.

```
[27] 1 print("훈련 샘플: {}, 레이블: {}".format(len(train_data), len(train_labels)))
      2 print("테스트 샘플: {}, 레이블: {}".format(len(test_data), len(test_labels)))
```

훈련 샘플: 25000, 레이블: 25000
테스트 샘플: 25000, 레이블: 25000

4) 정규화 등의 전처리

현재 아래코드로 확인한 것과 같이 영화 리뷰들의 길이가 모두 다릅니다. 신경망의 입력은 길이가 같아야하므로 전처리 과정이 필요합니다.

```
[ ]      1 len(train_data[0]), len(train_data[1])

(218, 189)
```

정규화 등의 전처리

이 프로젝트에서는, 정수 배열의 길이가 모두 같도록 패딩(padding)을 추가해 `max_length * num_reviews` 크기의 정수 텐서를 만듭니다. 이런 형태의 텐서를 다룰 수 있는 임베딩(embedding) 층을 신경망의 첫 번째 층으로 사용합니다.

영화 리뷰의 길이가 같아야 하므로 [pad_sequences](#) 함수를 사용해 길이를 맞추겠습니다:

```
[ ]      1 train_data = keras.preprocessing.sequence.pad_sequences(train_data,
2      |                                     value=word_index["<PAD>"],
3      |                                     padding='post',
4      |                                     maxlen=256)
5
6      6 test_data = keras.preprocessing.sequence.pad_sequences(test_data,
7      |                                                         value=word_index["<PAD>"],
8      |                                                         padding='post',
9      |                                                         maxlen=256)
```

이후 샘플의 길이를 확인해보면 길이가 동일해진 것을 확인할 수 있었습니다.

샘플의 길이를 확인해 보겠습니다:

```
[29]      1 len(train_data[0]), len(train_data[1])

(256, 256)
```

실제로 패딩이 된 데이터를 아래 코드를 통해 확인할 수 있습니다.

첫 번째 리뷰 데이터를 확인해보면 0으로 패딩된 것을 알 수 있습니다.

```
[30] 1 print(train_data[0])
```

```
[ 1 14 22 16 43 530 973 1622 1385 65 458 4468 66 3941
 4 173 36 256 5 25 100 43 838 112 50 670 2 9
35 480 284 5 150 4 172 112 167 2 336 385 39 4
172 4536 1111 17 546 38 13 447 4 192 50 16 6 147
2025 19 14 22 4 1920 4613 469 4 22 71 87 12 16
43 530 38 76 15 13 1247 4 22 17 515 17 12 16
626 18 2 5 62 386 12 8 316 8 106 5 4 2223
5244 16 480 66 3785 33 4 130 12 16 38 619 5 25
124 51 36 135 48 25 1415 33 6 22 12 215 28 77
52 5 14 407 16 82 2 8 4 107 117 5952 15 256
4 2 7 3766 5 723 36 71 43 530 476 26 400 317
46 7 4 2 1029 13 104 88 4 381 15 297 98 32
2071 56 26 141 6 194 7486 18 4 226 22 21 134 476
26 480 5 144 30 5535 18 51 36 28 224 92 25 104
4 226 65 16 38 1334 88 12 16 283 5 16 4472 113
103 32 15 16 5345 19 178 32 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0]
```

4. 딥러닝 모델

1) 모델 종류, 입력, 중간(은닉), 출력 층, 패러미터 수

모델은 ANN방식을 사용하고 중간에 은닉층이 하나 존재합니다.

모델 층을 쌓아가는 코드와 설명은 아래와 같습니다.

```
[ ] 1 # 입력 크기는 영화 리뷰 데이터셋에 적용된 어휘 사전의 크기입니다(10,000개의 단어)
    2 vocab_size = 10000
    3
    4 model = keras.Sequential()
    5 model.add(keras.layers.Embedding(vocab_size, 16, input_shape=(None,)))
    6 model.add(keras.layers.GlobalAveragePooling1D())
    7 model.add(keras.layers.Dense(16, activation='relu'))
    8 model.add(keras.layers.Dense(1, activation='sigmoid'))
    9
    10 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 16)	160000
global_average_pooling1d (GlobalAveragePooling1D)	(None, 16)	0
dense (Dense)	(None, 16)	272
dense_1 (Dense)	(None, 1)	17
Total params: 160,289		
Trainable params: 160,289		
Non-trainable params: 0		

1. 첫 번째 층은 **Embedding** 층입니다. 이 층은 정수로 인코딩된 단어를 입력 받고 각 단어 인덱스에 해당하는 임베딩 벡터를 찾습니다. 이 벡터는 모델이 훈련되면서 학습됩니다. 이 벡터는 출력 배열에 새로운 차원으로 추가됩니다. 최종 차원은 **(batch, sequence, embedding)** 이 됩니다.embedding 관련 문서 <http://doc.mindscale.kr/km/unstructured/11.html>
[+]임베딩 층을 사용했을 때 장점

- 텍스트 전처리에 대해 신경 쓸 필요가 없습니다.
- 전이 학습의 장점을 이용합니다.
- 임베딩은 고정 크기이기 때문에 처리 과정이 단순해집니다.

2. 그다음 **GlobalAveragePooling1D** 층은 **sequence** 차원에 대해 평균을 계산하여 각 샘플에 대해 고정된 길이의 출력 벡터를 반환합니다. 이는 가변 길이 입력을 다루는 가장 간단한 방법입니다.
3. 이 고정 길이의 출력 벡터는 16개의 은닉 유닛을 가진 완전 연결(fully-connected) 층(**Dense**)을 거칩니다.
4. 마지막 층은 하나의 출력 노드(node)를 가진 완전 연결 층입니다. **sigmoid** 활성화 함수를 사용하여 0과 1 사이의 실수를 출력합니다. 이 값은 확률 또는 신뢰도를 나타냅니다.

니다.

2) 옵티마이저, 손실함수

손실 함수와 옵티마이저

여기서는 이진 분류 모델이고 확률을 출력하므로(출력층의 유닛이 하나이고 sigmoid 활성화 함수를 사용합니다), `binary_crossentropy` 손실 함수를 사용하겠습니다.

강의 시간에 배운 `mean_squared_error(MSE)`를 선택할 수도 있지만, 일반적으로 `binary_crossentropy`가 확률을 다루는데 적합하다고 합니다. 이 함수는 확률 분포 간의 거리를 측정합니다. 여기에서는 정답인 타겟 분포와 예측 분포 사이의 거리입니다.

이제 모델이 사용할 옵티마이저와 손실 함수를 설정해 보겠습니다:

```
[ ] 1 model.compile(optimizer='adam',  
2               loss='binary_crossentropy',  
3               metrics=['accuracy'])
```

5. 데이터와 훈련과정, 예측 결과의 시각화

데이터와 훈련과정, 예측 결과의 시각화

모델을 훈련할 때 모델이 만난 적 없는 데이터에서 정확도를 확인하는 것이 좋다고 합니다. 여기서는 원본 훈련 데이터에서 10,000개의 샘플을 떼어내어 *검증 세트(validation set)*를 만듭니다.

(왜 테스트 세트를 사용하지 않는가?: 훈련 데이터만을 사용하여 모델을 개발하고 튜닝하는 것이 목표. 그다음 테스트 세트를 사용해서 딱 한 번만 정확도를 평가)

```
[49] 1 x_val = train_data[:10000]  
2     partial_x_train = train_data[10000:]  
3  
4     y_val = train_labels[:10000]  
5     partial_y_train = train_labels[10000:]
```

1) 모델 훈련

모델 훈련

배치사이즈를 512로 설정하고 40번훈련합니다.훈련하는 동안 10,000개의 검증 세트에서 모델의 손실과 정확도를 모니터링합니다:

```
[50] 1 history = model.fit(partial_x_train,  
2                       partial_y_train,  
3                       epochs=40,  
4                       batch_size=512,  
5                       validation_data=(x_val, y_val),  
6                       verbose=1)
```

```
Epoch 34/40
30/30 [=====] - 1s 17ms/step - loss: 0.1327 - accuracy: 0.9609 - val_loss: 0.2895 - val_accuracy: 0
Epoch 35/40
30/30 [=====] - 1s 18ms/step - loss: 0.1284 - accuracy: 0.9626 - val_loss: 0.2906 - val_accuracy: 0
Epoch 36/40
30/30 [=====] - 1s 17ms/step - loss: 0.1235 - accuracy: 0.9649 - val_loss: 0.2938 - val_accuracy: 0
Epoch 37/40
30/30 [=====] - 1s 17ms/step - loss: 0.1198 - accuracy: 0.9658 - val_loss: 0.2946 - val_accuracy: 0
Epoch 38/40
30/30 [=====] - 1s 17ms/step - loss: 0.1149 - accuracy: 0.9677 - val_loss: 0.2965 - val_accuracy: 0
Epoch 39/40
30/30 [=====] - 1s 17ms/step - loss: 0.1106 - accuracy: 0.9693 - val_loss: 0.2995 - val_accuracy: 0
Epoch 40/40
30/30 [=====] - 0s 16ms/step - loss: 0.1068 - accuracy: 0.9709 - val_loss: 0.3003 - val_accuracy: 0
```

2) 모델 평가

모델 평가

손실과 정확도를 출력하여 모델의 성능을 확인합니다.

```
[51] 1 results = model.evaluate(test_data, test_labels, verbose=2)
      2
      3 print(results)
```

```
782/782 - 1s - loss: 0.3190 - accuracy: 0.8740
[0.3189658522605896, 0.8740000128746033]
```

여기서는 약 87% 정도의 정확도를 달성했습니다.

고급 방법을 사용한 모델은 95%에 가까운 정확도를 얻을 수 있다고 합니다..

3) 정확도와 손실 그래프 그리기

정확도와 손실 그래프 그리기

위에서 반환한 history 객체에는 훈련하는 동안 일어난 모든 정보가 담긴 딕셔너리(dictionary)가 들어 있습니다. 이를 확인해보면 아래와 같습니다.

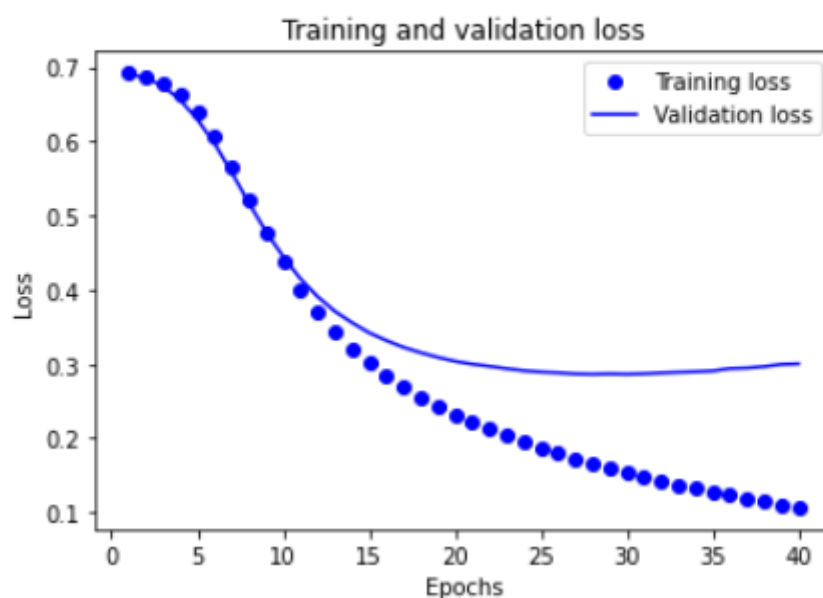
```
[52] 1 history_dict = history.history
      2 history_dict.keys()

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

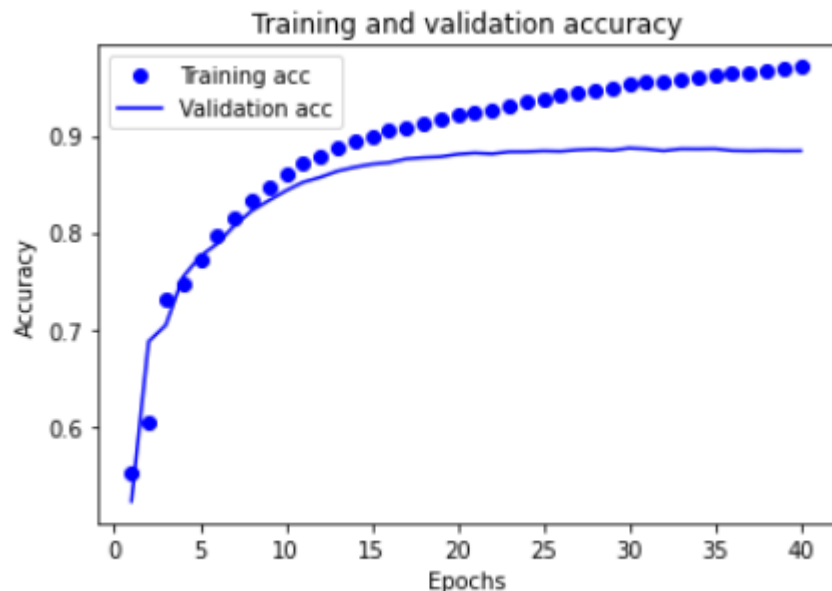
네 개의 항목이 있습니다. 훈련과 검증 단계에서 모니터링하는 지표들입니다. 훈련 손실과 검증 손실을 그래프로 그려 보고, 훈련 정확도와 검증 정확도도 그래프로 그려서 비교해 보겠습니다.

아래 코드를 통해 각각 검증셋, 테스트셋에 대하여 정확도와 손실값을 그래프로 그려볼 수 있었습니다.

```
[53] 1 import matplotlib.pyplot as plt
      2
      3 acc = history_dict['accuracy']
      4 val_acc = history_dict['val_accuracy']
      5 loss = history_dict['loss']
      6 val_loss = history_dict['val_loss']
      7
      8 epochs = range(1, len(acc) + 1)
      9
     10 # 훈련 손실 "bo"는 "파란색 점"입니다
     11 plt.plot(epochs, loss, 'bo', label='Training loss')
     12 # 검증 손실 "b"는 "파란 실선"입니다
     13 plt.plot(epochs, val_loss, 'b', label='Validation loss')
     14 plt.title('Training and validation loss')
     15 plt.xlabel('Epochs')
     16 plt.ylabel('Loss')
     17 plt.legend()
     18
     19 plt.show()
```



```
[54] 1 plt.clf()    # 그림을 초기화합니다
      2
      3 # 훈련 정확도
      4 plt.plot(epochs, acc, 'bo', label='Training acc')
      5 # 검증 정확도
      6 plt.plot(epochs, val_acc, 'b', label='Validation acc')
      7 plt.title('Training and validation accuracy')
      8 plt.xlabel('Epochs')
      9 plt.ylabel('Accuracy')
     10 plt.legend()
     11
     12 plt.show()
```



위 그래프에서 점선은 훈련 손실과 훈련 정확도를, 실선은 검증 손실과 검증 정확도를 나타냅니다.

훈련 손실이 에포크마다 감소하고 훈련 정확도가 증가하는 것을 확인할 수 있었습니다. 이 흐름을 통해 경사 하강법 최적화가 적용되고 있다는 것을 알 수 있었습니다.

반면에 검증 손실과 검증 정확도에서는 **20번째 에포크 이후가 최적점인 것**으로 볼 수 있습니다. 이는 **과대적합이 발생한 것**으로 보입니다.

그래서 이 **과대적합을 막기 위해서 콜백 함수를 사용해**봤습니다.

6. Callback 함수 사용

아래 코드를 통해 모델을 수정합니다.

keras에서 제공하는 **EarlyStopping** callback함수를 사용해 **val_loss(검증 손실)값이 2번 연속으로 증가하면 훈련을 중지하도록** 했습니다.

```
[61] 1 history = model.fit(partial_x_train,
2                             partial_y_train,
3                             epochs=40,
4                             batch_size=512,
5                             validation_data=(x_val, y_val),
6                             verbose=1,
7                             callbacks=[tf.keras.callbacks.EarlyStopping(patience=2, monitor='val_loss')])
```

```
Epoch 21/40
30/30 [=====] - 1s 18ms/step - loss: 0.2052 - accuracy: 0.9292 - val_loss: 0.2925 - val_accuracy: 0.8
Epoch 22/40
30/30 [=====] - 1s 18ms/step - loss: 0.1962 - accuracy: 0.9318 - val_loss: 0.2917 - val_accuracy: 0.8
Epoch 23/40
30/30 [=====] - 1s 17ms/step - loss: 0.1880 - accuracy: 0.9365 - val_loss: 0.2880 - val_accuracy: 0.8
Epoch 24/40
30/30 [=====] - 1s 17ms/step - loss: 0.1796 - accuracy: 0.9405 - val_loss: 0.2873 - val_accuracy: 0.8
Epoch 25/40
30/30 [=====] - 0s 17ms/step - loss: 0.1723 - accuracy: 0.9437 - val_loss: 0.2856 - val_accuracy: 0.8
Epoch 26/40
30/30 [=====] - 1s 17ms/step - loss: 0.1652 - accuracy: 0.9472 - val_loss: 0.2866 - val_accuracy: 0.8
Epoch 27/40
30/30 [=====] - 0s 16ms/step - loss: 0.1588 - accuracy: 0.9491 - val_loss: 0.2848 - val_accuracy: 0.8
Epoch 28/40
30/30 [=====] - 1s 18ms/step - loss: 0.1522 - accuracy: 0.9525 - val_loss: 0.2852 - val_accuracy: 0.8
Epoch 29/40
30/30 [=====] - 1s 17ms/step - loss: 0.1463 - accuracy: 0.9549 - val_loss: 0.2867 - val_accuracy: 0.8
```

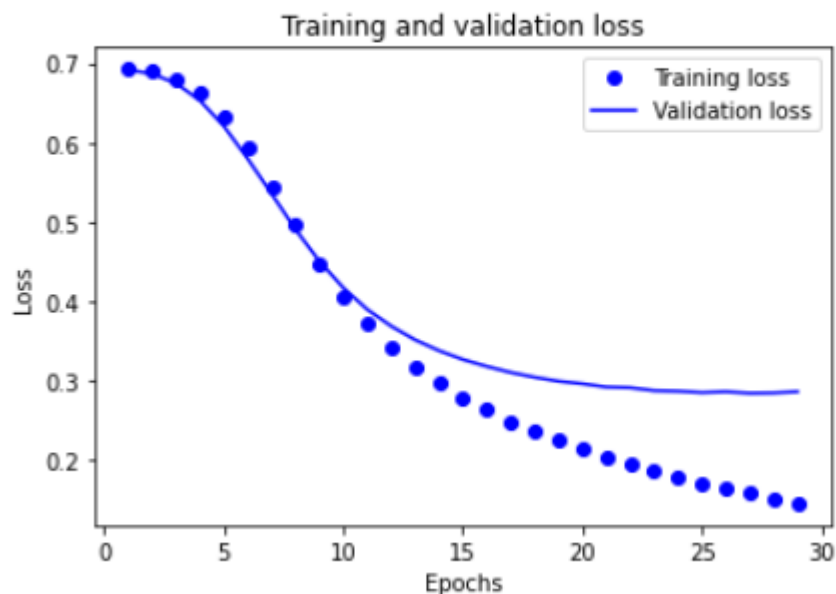
callback 함수를 추가함으로써 **29번째 에포크에서 학습을 종료하는 것을** 확인할 수 있었습니다.

이후 손실값과 정확도를 그래프로 확인해보았습니다.

```

1 import matplotlib.pyplot as plt
2
3 acc = history_dict['accuracy']
4 val_acc = history_dict['val_accuracy']
5 loss = history_dict['loss']
6 val_loss = history_dict['val_loss']
7
8 epochs = range(1, len(acc) + 1)
9
10 # 훈련 손실 "bo"는 "파란색 점"입니다
11 plt.plot(epochs, loss, 'bo', label='Training loss')
12 # 검증 손실 "b"는 "파란 실선"입니다
13 plt.plot(epochs, val_loss, 'b', label='Validation loss')
14 plt.title('Training and validation loss')
15 plt.xlabel('Epochs')
16 plt.ylabel('Loss')
17 plt.legend()
18
19 plt.show()

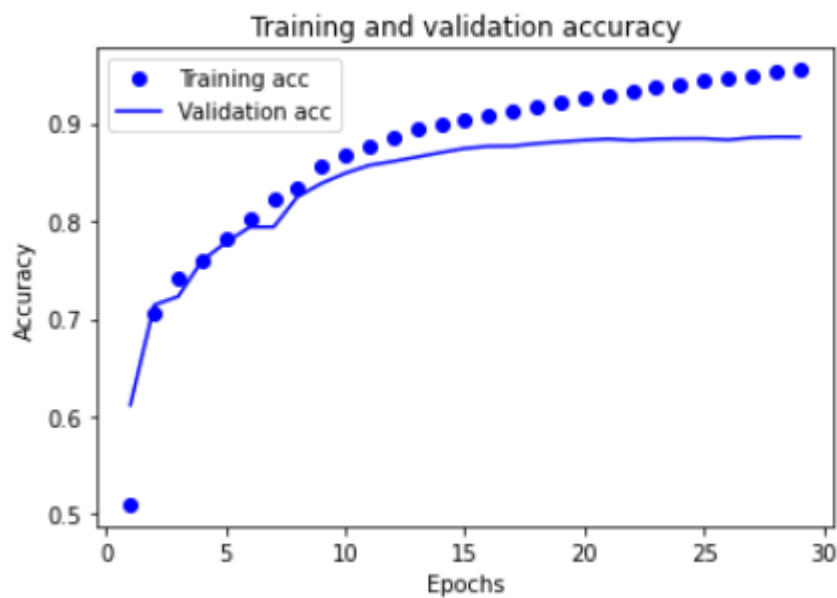
```



```

1 plt.clf() # 그림을 초기화합니다
2
3 # 훈련 정확도
4 plt.plot(epochs, acc, 'bo', label='Training acc')
5 # 검증 정확도
6 plt.plot(epochs, val_acc, 'b', label='Validation acc')
7 plt.title('Training and validation accuracy')
8 plt.xlabel('Epochs')
9 plt.ylabel('Accuracy')
10 plt.legend()
11
12 plt.show()

```



이전과 큰 변화는 없었지만 **정확성과 손실 수치를 확인해보니 차이를 확인할 수** 있었습니다.

```
[51] 1 results = model.evaluate(test_data, test_labels, verbose=2)
      2
      3 print(results)
```

```
782/782 - 1s - loss: 0.3190 - accuracy: 0.8740
[0.3189658522605896, 0.8740000128746033]
```

callback 함수 적용 전

```
[65] 1 results = model.evaluate(test_data, test_labels, verbose=2)
      2
      3 print(results)
```

```
782/782 - 1s - loss: 0.3033 - accuracy: 0.8759
[0.30331748723983765, 0.8759199976921082]
```

성능이 확연하게 변화하진 않았지만 손실이 줄고 정확성이 향상됨을 확인할 수 있었습니다.

callback 함수 적용 후