

딥러닝을 활용한 자연어 처리(3)

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

09

APPLICATION

ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn like humans.

The term may also be applied to any machine that exhibits human-like traits such as learning and problem-solving.

Artificial intelligence (AI) refers to the simulation of human

Notice

Artificial intelligence (AI) refers to the simulation of human

Artificial intelligence (AI) refers to the simulation of human intelligence in machines that are programmed to think and learn like humans.

이 교육과정은 교육부 ‘성인학습자 역량 강화 교육콘텐츠 개발’ 사업의 일환으로써
교육부로부터 예산을 지원 받아 고려사이버대학교가 개발하여 운영하고 있습니다.
제공하는 강좌 및 학습에 따르는 모든 산출물의 저작권은 교육부, 한국교육학술정보원,
한국원격대학협의회와 고려사이버대학교가 공동 소유하고 있습니다.

THINKING

생각해보기

- ✓ **Convolution**을 활용한 텍스트 분류는
어떻게 할까요?



학습목표

Artificial Intelligence (AI) refers
to the simulation of human

GOALS

Artificial Intelligence has
risen to the position of
human intelligence in various
fields and industries. It has
not only helped humans to work
better, but also to work
smarter.

The technology will be applied
to various fields such as
healthcare, education, and
entertainment, and it will
continue to evolve and
improve.

- 1 **Convolution** 개요와 **Convolution**에 대한 작동 방식에 대해 이해하고 설명할 수 있다.
- 2 고차원에서의 **Convolution**을 이해하고 설명할 수 있다.
- 3 **1D Conv**, **2D Conv**에 대해 이해하고 설명할 수 있다.
- 4 **Max-pooling**와 **Dense Layer**에 대해 설명할 수 있다.
- 5 자연어 처리에서 어떻게 **1D Conv**을 사용하는지 파악할 수 있다.
- 6 **IMDB 데이터셋**을 활용한 자연어 처리 실습을 할 수 있다.



- 1 CNN 개요
- 2 자연어 처리에서의 CNN
- 3 실습

"The more things change, the more they stay the same."
In any machine learning problem,
there are many things that remain the same,
and many that change, and
this is the key to success.

Artificial intelligence (AI) refers
to the simulation of human
intelligence by machines,
which can be used to perform
tasks that would otherwise
require human intelligence.

CONTENTS

학습내용

Artificial intelligence (AI) refers
to the simulation of human

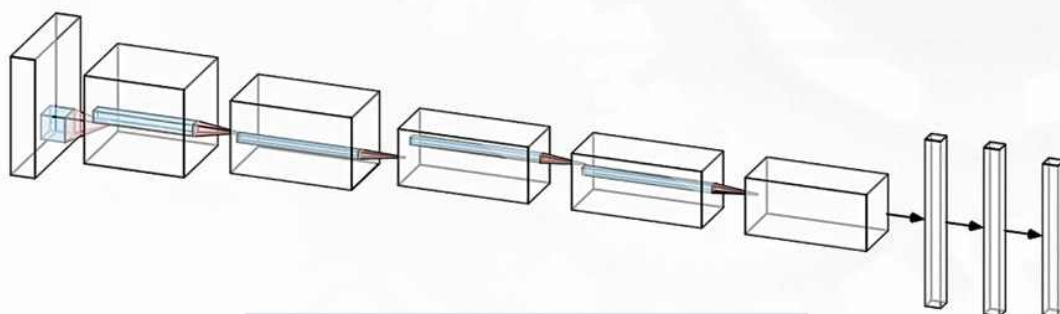
CNN 개요



01 Convolution 개요

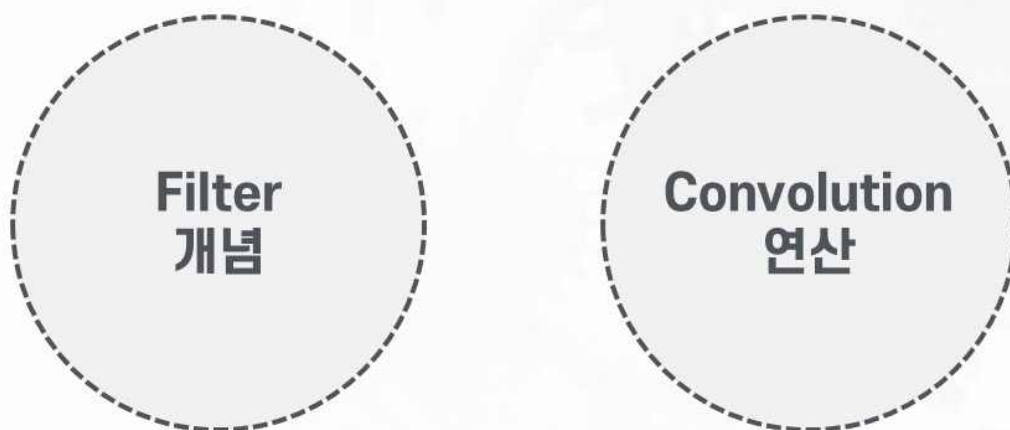
Convolution Neural Network(이하 CNN)은

“동물의 시각 신경망과 유사한 점이 많음”

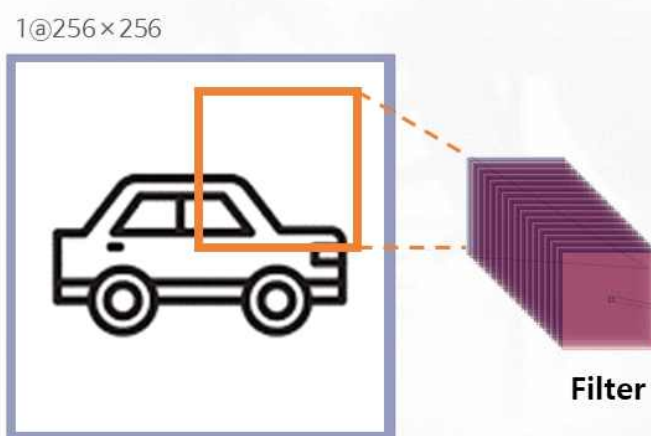


Convolutional Neural Network

CNN을 구성하는 가장 중요한 요소

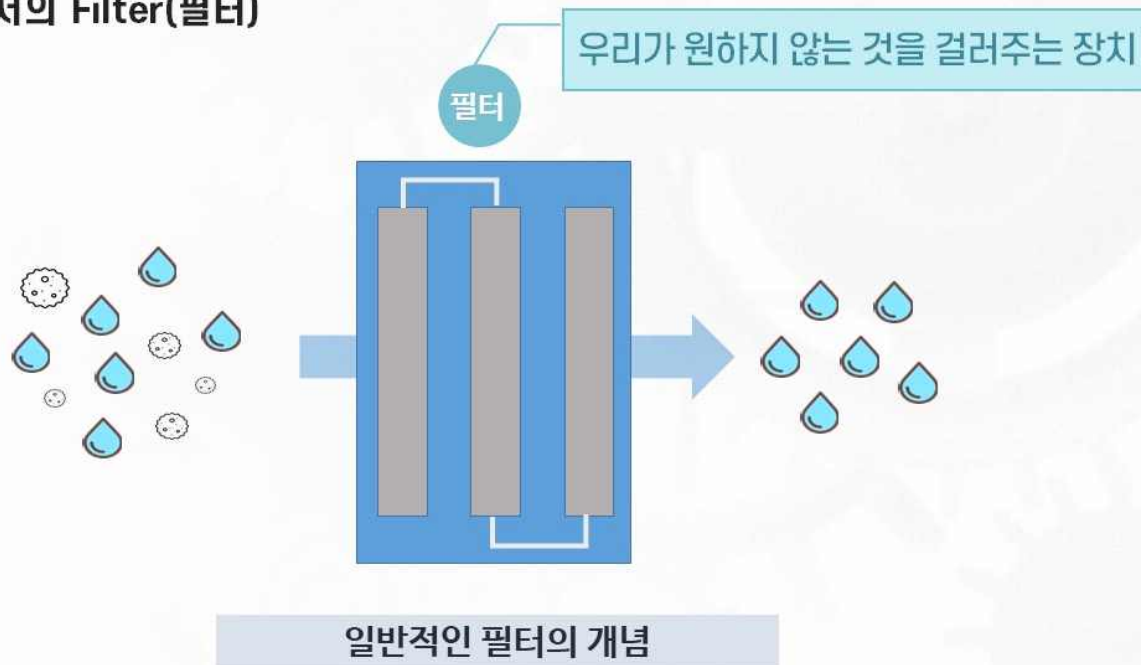


CNN을 구성하는 가장 중요한 요소



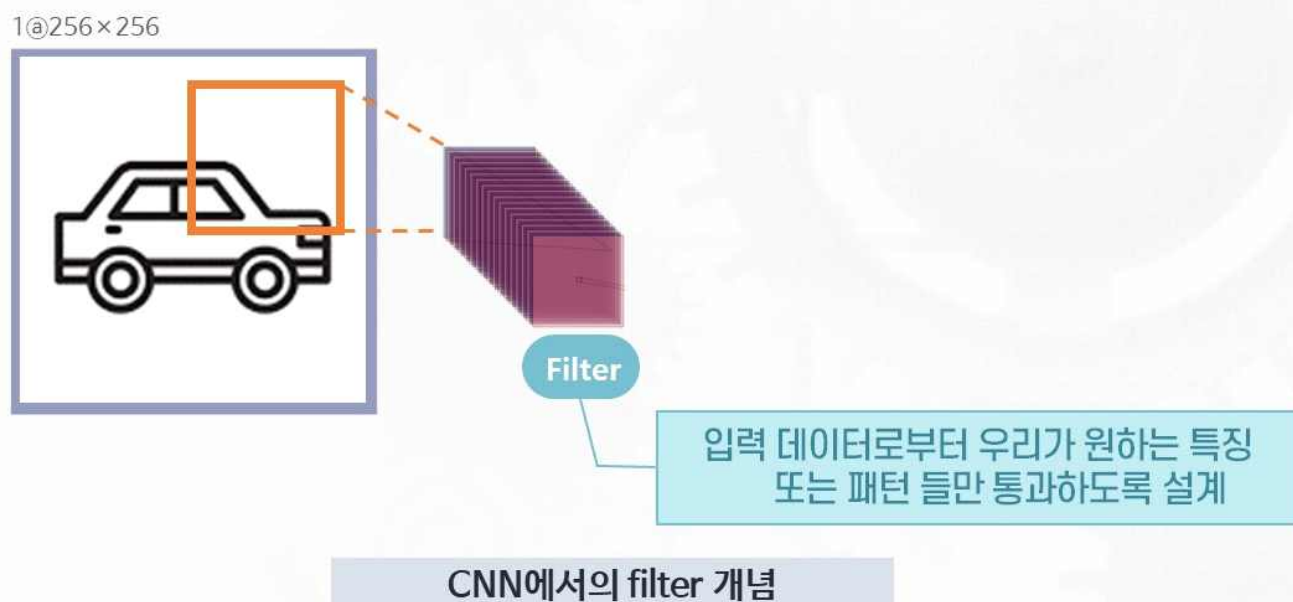
Filter와 Convolution 연산

CNN에서의 Filter(필터)



출처: 퍼블릭에이아이(www.publicai.co.kr)

CNN에서의 Filter(필터)



출처: 퍼블릭에이아이(www.publicai.co.kr)

CNN에서의 Convolution 연산(합성곱 연산)

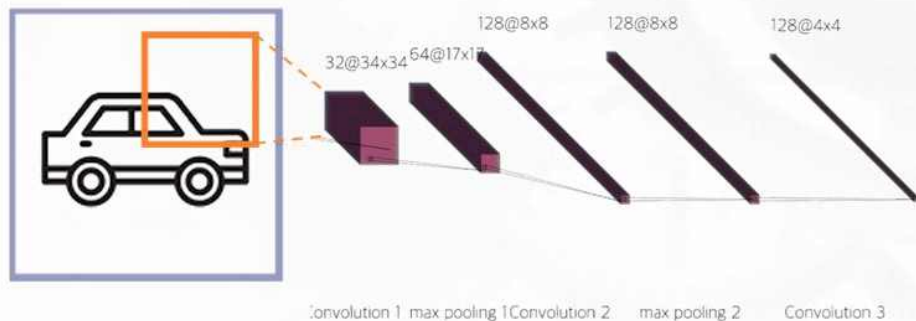
- 합성곱 연산과 필터의 역할



출처: 퍼블릭에이아이(www.publicai.co.kr)

인공신경망과 동물이 사물을 인식하는 방법

- 실제로 CNN 인공신경망의 경우 동물이 사물을 인식하는 방법과 같이 겹겹이 층으로 쌓여 동작함

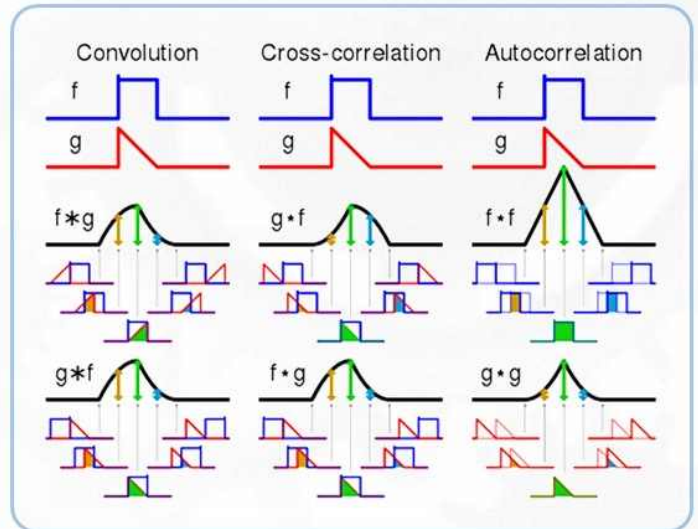


Convolutional Neural Network

출처: 퍼블릭에이아이(www.publicai.co.kr)

CNN에서의 Convolution 연산(합성곱 연산)

- 합성곱(合成-, convolution, 콘벌루션)
: 하나의 함수와 또 다른 함수를 반전
이동한 값을 곱한 다음, 구간에 대해
적분하여 새로운 함수를 구하는 수학
연산자



출처: 퍼블릭에이아이 (www.publicai.co.kr)

CNN에서의 Convolution 연산(합성곱 연산)

$$\sum \left(\begin{array}{|c|c|c|} \hline \text{input} & & \\ \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \otimes \begin{array}{|c|c|} \hline \text{filter} & \\ \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & \\ \hline & \\ \hline \end{array}$$

Window : 입력 데이터 내 filter와 FMA 연산이 되는 부분

FMA : Fused Multiply Add (단일 곱셈 누산기)

① Element-wise multiply

입력 특징 맵

1	1	0
---	---	---

1	0	1
---	---	---

1x1 1x0 0x1

② Summation

$$1 \times 1 + 1 \times 0 + 0 \times 1 = 1$$

출처: 퍼블릭에이아이 (www.publicai.co.kr)

CNN에서의 Convolution 연산(합성곱 연산)

$$\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$$

$$\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$$

$$\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & 0 \\ \hline \end{array}$$

$$\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & 18 \\ \hline \end{array}$$

출처: 퍼블릭에이아이(www.publicai.co.kr)

- Window가 움직이는 크기도 지정할 수 있음
- 왼쪽 그림에는 Stride 가 1,1로 지정되어 있어 오른쪽으로 한 칸 그리고 아래쪽으로 한 칸 이동하지만 Stride을 2, 2로 지정하면 2칸씩 건너 뛸 수도 있음

CNN에서의 Convolution 연산(합성곱 연산)

- Stride 가 2,2 인 Convolution Layer

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array}$$

출처: 퍼블릭에이아이(www.publicai.co.kr)

- Window가 움직이는 크기도 지정할 수 있음
- 왼쪽 그림에는 Stride 가 1,1로 지정되어 있어 오른쪽으로 한 칸 그리고 아래쪽으로 한 칸 이동하지만 Stride을 2, 2로 지정하면 2칸씩 건너 뛸 수도 있음

CNN에서의 Convolution 연산(합성곱 연산)

- 이미지의 특징을 추출하는 방법

$$\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & \\ \hline & \\ \hline \end{array}$$

// //

출처: 퍼블릭에이아이 (www.publicai.co.kr)

→ Filter와 window와 유사하면 출력값은 크게 되고 Filter와 window가 유사하지 않으면 상대적으로 적은 값을 가지게 됨

CNN에서의 Convolution 연산(합성곱 연산)

- 이동한 Window내의 특징과 필터의 특징

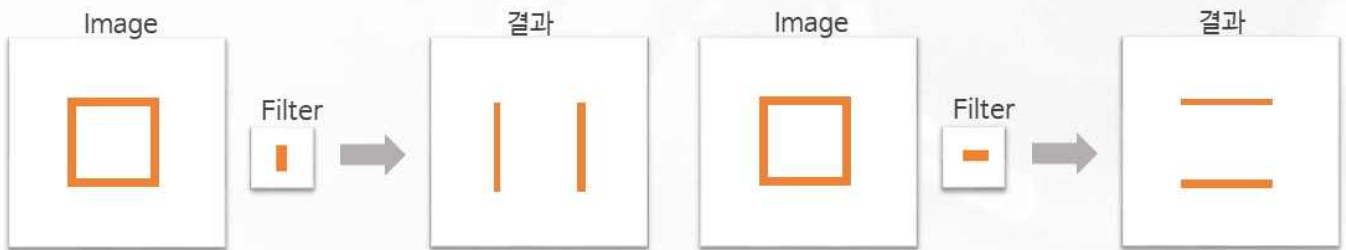
$$\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline & \\ \hline \end{array}$$

/ /

출처: 퍼블릭에이아이 (www.publicai.co.kr)

CNN에서의 Convolution 연산(합성곱 연산)

- 합성곱 연산과 필터의 역할



출처: 퍼블릭에이아이 (www.publicai.co.kr)

CNN에서의 Convolution 연산(합성곱 연산)

$$\begin{aligned}
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & \\ \hline & \\ \hline \end{array} \\
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline & \\ \hline \end{array} \\
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & \\ \hline \end{array} \\
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & 18 \\ \hline \end{array}
 \end{aligned}$$

출처: 퍼블릭에이아이 (www.publicai.co.kr)

Tensorflow로 따라 해보기

```

1 import tensorflow as tf
2 import numpy as np
3 from tensorflow.keras.layers import Conv2D, Input, Dense
4 from tensorflow.keras.models import Model

1 # 입력 벡터를 설정합니다.
2 input_array = np.array(
3     [[9, 0, 9],
4      [0, 9, 0],
5      [9, 0, 9]]
6 )
7 input_array = input_array.reshape([1,3,3,1])
8
9 # convolution 에 사용할 filter 입니다.
10 filter_ = np.array([1, 0, 0, 1]).reshape([2, 2, 1, 1])

1 input_tensor = Input(shape=[3,3,1])
2 layer = Conv2D(filters=1, kernel_size=2, use_bias=False)(input_tensor)
3
4 model = Model(input_tensor, layer)
5 model.layers[1].set_weights([filter_])
6

1 model.predict(input_array)

array([[[[18.],
         [ 0.]],
        [[ 0.],
         [18.]]]], dtype=float32)

```

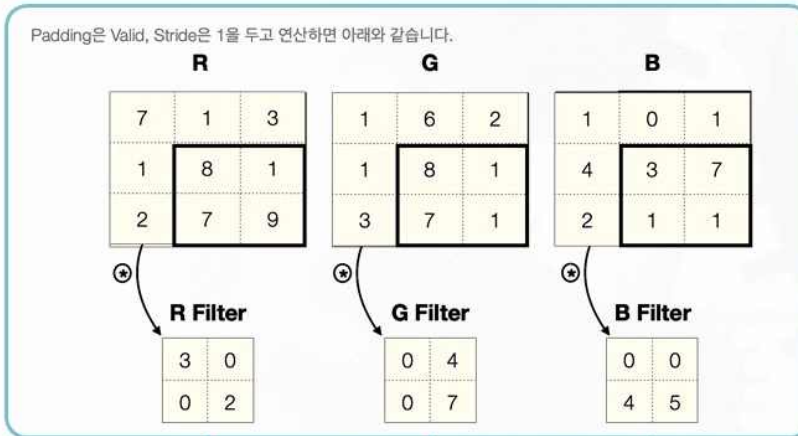
03 고차원에서의 Convolution

$$\begin{aligned}
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & \\ \hline & \\ \hline \end{array} \\
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline & \\ \hline \end{array} \\
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & \\ \hline \end{array} \\
 &\Sigma \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline 0 & 18 \\ \hline \end{array}
 \end{aligned}$$

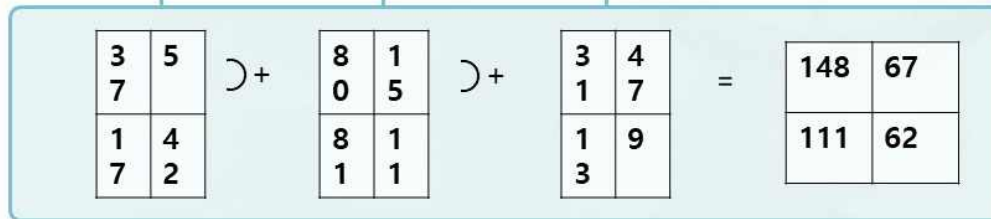
→ 2차원 입력 영상 Convolution에 이어서
3차원 이상의 입력 데이터에 대해서도
Convolution을 어떻게 수행해야 하는지
학습함

03 고차원에서의 Convolution

Artificial Intelligence (AI) refers to the simulation of human intelligence in the computers.



→ 고차원에서의 Convolution도
Element-wise multiply 후에
모든 값이 더해짐



출처: 퍼블릭에이아이(www.publicai.co.kr)

03 고차원에서의 Convolution

Artificial Intelligence (AI) refers to the simulation of human intelligence in the computers.

Tensoflow Convolution Function

```
[59] 1 input_array = np.array(
2     [[7, 1, 1],
3      [1, 0, 2],
4      [3, 2, 1]],
5
6     [[1, 1, 4],
7      [8, 8, 3],
8      [1, 1, 7]],
9
10    [[2, 3, 2],
11     [7, 7, 1],
12     [9, 1, 1]])
13 )
14 input_array.shape
15
16 # 텐서플로에 입력 가능하도록 앞에 차원 하나를 추가합니다.
17 input_array = input_array.reshape([1,3,3,3])
18
19 # 필터도 3차원으로 만듭니다.
20 filter_ = np.array(
21     [[3, 0, 0],
22      [0, 4, 0],
23      [0, 0, 4],
24      [2, 7, 5]]
25 )
26 filter_ = filter_.reshape([2,2,3,1])

[61] 1 input_tensor = Input(shape=[3,3,3])
2     layer = Conv2D(filters=1, kernel_size=2, use_bias=False)(input_tensor)
3
4     model = Model(input_tensor, layer)
5     model.layers[1].set_weights([filter_])

1 model.predict(input_array)

WARNING:tensorflow:8 out of the last 9 calls to <function Model.make_predict_
array([[[[124.],
        [ 67.]],
        [[111.],
        [ 62.]]]], dtype=float32)
```

1D Convolution 연산 vs 2D Convolution 연산

- 1D Convolution 과 2D Convolution의 차이

2D Convolution

$$\sum \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline & \\ \hline \end{array}$$

1D Convolution

$$\sum \left(\begin{array}{|c|c|c|c|c|c|c|c|} \hline 9 & 0 & 0 & 9 & 0 & 9 & 9 & 9 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|c|} \hline 1 & 0 & 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|c|c|c|c|c|} \hline 18 & 0 & 9 & 18 & & & \\ \hline \end{array}$$

출처: 퍼블릭에이아이(www.publicai.co.kr)

1D Convolution 연산 vs 2D Convolution 연산

2D Convolution

$$\sum \left(\begin{array}{|c|c|c|} \hline 9 & 0 & 9 \\ \hline 0 & 9 & 0 \\ \hline 9 & 0 & 9 \\ \hline \end{array} \odot \begin{array}{|c|c|} \hline 1 & 0 \\ \hline 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|} \hline 18 & 0 \\ \hline & \\ \hline \end{array}$$

1D Convolution

$$\sum \left(\begin{array}{|c|c|c|c|c|c|c|c|} \hline 9 & 0 & 9 & 0 & 9 & 0 & 9 & 0 & 9 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|c|} \hline 1 & 0 & 0 & 1 \\ \hline \end{array} \right) = \begin{array}{|c|c|c|c|c|c|c|} \hline 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ \hline \end{array}$$

→ 1D, 2D Convolution의 원리는 동일하지만 차이점은
Filter와 Filter가 적용될 Window가 1D 와 2D라는 것임

→ 2D Convolution 은 2D 특성을 추출하고

출처: 퍼블릭에이아이(www.publicai.co.kr)

1D Convolution은 1D 특성을 추출함

Tensorflow 1D Convolution 코드

1D Convolution Layer

```
[67] 1 input_array = np.array(
2     [9,0,9,0,9,0,9,0,9]
3 )
4 input_array = input_array.reshape([1,9,1])
5
6 filter_ = np.array(
7     [1, 0, 0, 1]
8 )
9
10 filter_ = filter_.reshape([4,1,1])
11
```

```
[69] 1 input_tensor = Input(shape=[9, 1])
2     layer = Conv1D(filters=1, kernel_size=4, use_bias=False)(input_tensor)
3
4     model = Model(input_tensor, layer)
5     model.layers[1].set_weights([filter_])
```

```
[70] 1 model.predict(input_array)
```

WARNING:tensorflow:9 out of the last 10 calls to <function Model.make_predict_func
array([[9.],
 [9.],
 [9.],
 [9.],
 [9.],
 [9.]], dtype=float32)

Pooling 연산

1	3	2	1
2	9	1	1
1	3	2	3
2	6	1	2



9	2
6	3

- Convolution을 수행 시 입력데이터의 크기가 거의 줄어 들지 않음
- 이로 인해 연산량이 증가하게 되고 학습 속도가 느려지게 됨
- 연산량을 줄이고 학습 속도를 빠르게 하기 위해 Pooling을 수행함

Pooling 연산

1	3	2	1
2	9	1	1
1	3	2	3
2	6	1	2



9	2
6	3

- Max Pooling은 Window 내 가장 큰 값을 반환함
- 일반적으로 Window를 겹치지 않게 함으로서 입력 데이터를 반으로 줄임

Pooling 연산

Max Pooling

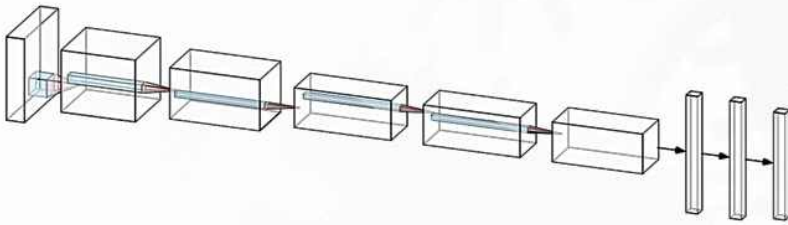
```
[76] 1 input_array = np.array(
      2     [[1, 3, 2, 1],
      3     [2, 9, 1, 1],
      4     [1, 3, 2, 3],
      5     [2, 6, 1, 2]]
      6 )
      7
      8 input_array = input_array.reshape(1,4,4,1)
```

```
[80] 1 input_tensor = Input(shape=[4, 4, 1])
      2 layer = MaxPool2D()(input_tensor)
      3 model = Model(input_tensor, layer)
      4 model.predict(input_array)
      5
```

```
WARNING:tensorflow:10 out of the last 11 calls to <fu
array([[[[9.],
        [2.]],

        [[6.],
        [3.]]]], dtype=float32)
```


Fully Connected Layer



출처: 퍼블릭에이아이(www.publicai.co.kr)

- 입력 데이터가 Convolution layer을
지난 다음에는 Fully Connected
Layer(DenseLayer)을
지나게 함으로서
최종적으로 우리가 원하는 형태로
변환함

Dense Layer

```

1 input_tensor = Input(shape=[16, 16, 1])
2
3 # Convolution
4 layer = Conv2D(filters=1, kernel_size=2, use_bias=False)(input_tensor)
5 layer = Conv2D(filters=1, kernel_size=2, use_bias=False)(layer)
6 layer = MaxPool2D()(layer)
7
8 # Convolution
9 layer = Conv2D(filters=1, kernel_size=2, use_bias=False)(input_tensor)
10 layer = Conv2D(filters=1, kernel_size=2, use_bias=False)(layer)
11 layer = MaxPool2D()(layer)
12
13
14 # Flatten
15 layer = Flatten()(layer)
16
17 # Dense Layer
18 layer = Dense(units=128)(layer)
19 layer = Dense(units=128)(layer)
20 layer = Dense(units=1)(layer)
21
22 # Model
23 model = Model(input_tensor, layer)

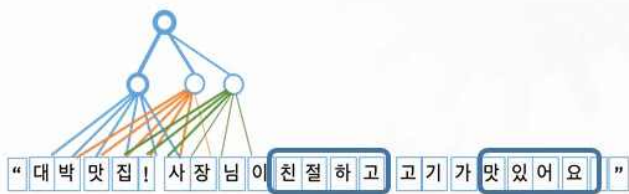
```

02

자연어 처리에서의 CNN



01 자연어 처리에서의 1D Convolution Neural Network



특정 패턴이 해당 문장이
긍정적인걸 파악 할 수 있게 해줌

- 일반적으로 자연어는 1D Sequence의 데이터임
- 추가적인 전가 필처리요하지만 일반적으로 우리는 자연어에서 목적 함수에 영향을 미치는 패턴을 1D Convolution을 활용해 찾을 수 있음

01 자연어 처리에서의 1D Convolution Neural Network

Artificial Intelligence (AI) starts
in the simulation of human

“ 대 박 맛 집 ! 사 장 님 이 친 절 하 고 고 기 가 맛 있 어 요 ! ”

↓ embedding

• [[-0.031105, -0.43216, 0.30075, ..., -0.54213, 0.84074, 0.17665]
대 [-0.63628, 1.2969, -1.278, ..., -0.71431, -0.34964, -0.033868],
박 [-0.63258, 2.2969, -3.278, ..., -4.71431, -1.34964, -2.033868],
! [0.20712, 0.45525, -0.59307, ..., -0.14499, 0.5076, -0.75084]]

↓ AI 모델

긍정 / 부정

- 자연어를 Embedding 한 후 적절한 AI 모형에 넣어 분류 또는 회귀 모델을 만들 수 있음
- 해당 과정에서는 Conv1D를 활용해 긍정과 부정의 리뷰를 분류 해보도록 함

출처: 퍼블릭에이아이(www.publicai.co.kr)

02 IMDB 데이터셋을 활용한 자연어 처리 실습

Artificial Intelligence (AI) starts
in the simulation of human

IMDB Dataset 설명 및 프로젝트 설명

IMDB Dataset은

“ 약 25,000 개 영어 단어로 구성되어
있는 영어 리뷰 데이터 셋임 ”

IMDB Dataset 설명 및 프로젝트 설명

IMDB 감정 분석 프로젝트는

DB 데이터와 Convolution Neural Network을 활용해

리뷰가 긍정(Positive)인지 부정(Negative) 인지 파악

IMDB 감정 분류 데이터셋 다운로드

- Keras 을 활용한 IMDB 데이터셋 다운로드

```
from tensorflow.keras.datasets import imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data()
```

Signatures:
 imdb.load_data(
 path="imdb.npz",
 num_words=None,
 skip_top=0,
 maxlen=None,
 seed=111,
 start_char=1,
 oov_char=2,
 index_from=3,
 **kwargs,
)
Docstrings:
 Loads the IMDB dataset.
Arguments:
 path: where to cache the data (relative to ~/.keras/dataset).
 num_words: max number of words to include. Words are ranked
 by how often they occur (in the training set) and only
 the most frequent words are kept
 skip_top: skip the top N most frequently occurring words
 (which may not be informative).
Returns:
 Tuple of Numpy arrays: (x_train, y_train), (x_test, y_test).
Raises:
 ValueError: in case 'maxlen' is so low
 that no input sequence could be kept.
 Note that the 'out of vocabulary' character is only used for
 words that were present in the training set but are not included
 because they're not making the 'num_words' cut here.
 Words that were not seen in the training set but are in the test set
 have simply been skipped.
 File: ~/anaconda3/envs/python3.6/site-packages/tensorflow_core/python/keras/datasets/imdb.py
 Type: function

➡ Keras에서 제공해주는 Dataset은
일반적으로 (x_train, y_train), (x_test, y_test)
이 형태로 제공됨

IMDB 감정 분류 데이터셋 다운로드

IMDB movie review sentiment
classification dataset

Load_data function

```
tf.keras.datasets.imdb.load_data(
    path="imdb.npz",
    num_words=None,
    skip_top=0,
    maxlen=None,
    seed=113,
    start_char=1,
    oov_char=2,
    index_from=3,
    **kwargs
)
```

Loads the IMDB dataset.

This is a dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.

Arguments

- path**: where to cache the data (relative to `~/keras/datasets`).
- num_words**: integer or None. Words are ranked by how often they occur (in the training set) and only the `num_words` most frequent words are kept. Any less frequent word will appear as `oov_char` value in the sequence data. If None, all words are kept. Defaults to None, so all words are kept.
- skip_top**: skip the top N most frequently occurring words (which may not be informative). These words will appear as `oov_char` value in the dataset. Defaults to 0, so no words are skipped.
- maxlen**: int or None. Maximum sequence length. Any longer sequence will be truncated. Defaults to None, which means no truncation.
- seed**: int. Seed for reproducible data shuffling.
- start_char**: int. The start of a sequence will be marked with this character. Defaults to 1 because 0 is usually the padding character.
- oov_char**: int. The out-of-vocabulary character. Words that were cut out because of the `num_words` or `skip_top` limits will be replaced with this character.
- index_from**: int. Index actual words with this index and higher.
- **kwargs**: Used for backwards compatibility.

```
x_train = [
    [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023, 1024, 1025, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035, 1036, 1037, 1038, 1039, 1040, 1041, 1042, 1043, 1044, 1045, 1046, 1047, 1048, 1049, 1050, 1051, 1052, 1053, 1054, 1055, 1056, 1057, 1058, 1059, 1060, 1061, 1062, 1063, 1064, 1065, 1066, 1067, 1068, 1069, 1070, 1071, 1072, 1073, 1074, 1075, 1076, 1077, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1085, 1086, 1087, 1088, 1089, 1090, 1091, 1092, 1093, 1094, 1095, 1096, 1097, 1098, 1099, 1100, 1101, 1102, 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111, 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120, 1121, 1122, 1123, 1124, 1125, 1126, 1127, 1128, 1129, 1130, 1131, 1132, 1133, 1134, 1135, 1136, 1137, 1138, 1139, 1140, 1141, 1142, 1143, 1144, 1145, 1146, 1147, 1148, 1149, 1150, 1151, 1152, 1153, 1154, 1155, 1156, 1157, 1158, 1159, 1160, 1161, 1162, 1163, 1164, 1165, 1166, 1167, 1168, 1169, 1170, 1171, 1172, 1173, 1174, 1175, 1176, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194, 1195, 1196, 1197, 1198, 1199, 1200, 1201, 1202, 1203, 1204, 1205, 1206, 1207, 1208, 1209, 1210, 1211, 1212, 1213, 1214, 1215, 1216, 1217, 1218, 1219, 1220, 1221, 1222, 1223, 1224, 1225, 1226, 1227, 1228, 1229, 1230, 1231, 1232, 1233, 1234, 1235, 1236, 1237, 1238, 1239, 1240, 1241, 1242, 1243, 1244, 1245, 1246, 1247, 1248, 1249, 1250, 1251, 1252, 1253, 1254, 1255, 1256, 1257, 1258, 1259, 1260, 1261, 1262, 1263, 1264, 1265, 1266, 1267, 1268, 1269, 1270, 1271, 1272, 1273, 1274, 1275, 1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283, 1284, 1285, 1286, 1287, 1288, 1289, 1290, 1291, 1292, 1293, 1294, 1295, 1296, 1297, 1298, 1299, 1300, 1301, 1302, 1303, 1304, 1305, 1306, 1307, 1308, 1309, 1310, 1311, 1312, 1313, 1314, 1315, 1316, 1317, 1318, 1319, 1320, 1321, 1322, 1323, 1324, 1325, 1326, 1327, 1328, 1329, 1330, 1331, 1332, 1333, 1334, 1335, 1336, 1337, 1338, 1339, 1340, 1341, 1342, 1343, 1344, 1345, 1346, 1347, 1348, 1349, 1350, 1351, 1352, 1353, 1354, 1355, 1356, 1357, 1358, 1359, 1360, 1361, 1362, 1363, 1364, 1365, 1366, 1367, 1368, 1369, 1370, 1371, 1372, 1373, 1374, 1375, 1376, 1377, 1378, 1379, 1380, 1381, 1382, 1383, 1384, 1385, 1386, 1387, 1388, 1389, 1390, 1391, 1392, 1393, 1394, 1395, 1396, 1397, 1398, 1399, 1400, 1401, 1402, 1403, 1404, 1405, 1406, 1407, 1408, 1409, 1410, 1411, 1412, 1413, 1414, 1415, 1416, 1417, 1418, 1419, 1420, 1421, 1422, 1423, 1424, 1425, 1426, 1427, 1428, 1429, 1430, 1431, 1432, 1433, 1434, 1435, 1436, 1437, 1438, 1439, 1440, 1441, 1442, 1443, 1444, 1445, 1446, 1447, 1448, 1449, 1450, 1451, 1452, 1453, 1454, 1455, 1456, 1457, 1458, 1459, 1460, 1461, 1462, 1463, 1464, 1465, 1466, 1467, 1468, 1469, 1470, 1471, 1472, 1473, 1474, 1475, 1476, 1477, 1478, 1479, 1480, 1481, 1482, 1483, 1484, 1485, 1486, 1487, 1488, 1489, 1490, 1491, 1492, 1493, 1494, 1495, 1496, 1497, 1498, 1499, 1500, 1501, 1502, 1503, 1504, 1505, 1506, 1507, 1508, 1509, 1510, 1511, 1512, 1513, 1514, 1515, 1516, 1517, 1518, 1519, 1520, 1521, 1522, 1523, 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538, 1539, 1540, 1541, 1542, 1543, 1544, 1545, 1546, 1547, 1548, 1549, 1550, 1551, 1552, 1553, 1554, 1555, 1556, 1557, 1558, 1559, 1560, 1561, 1562, 1563, 1564, 1565, 1566, 1567, 1568, 1569, 1570, 1571, 1572, 1573, 1574, 1575, 1576, 1577, 1578, 1579, 1580, 1581, 1582, 1583, 1584, 1585, 1586, 1587, 1588, 1589, 1590, 1591, 1592, 1593, 1594, 1595, 1596, 1597, 1598, 1599, 1600, 1601, 1602, 1603, 1604, 1605, 1606, 1607, 1608, 1609, 1610, 1611, 1612, 1613, 1614, 1615, 1616, 1617, 1618, 1619, 1620, 1621, 1622, 1623, 1624, 1625, 1626, 1627, 1628, 1629, 1630, 1631, 1632, 1633, 1634, 1635, 1636, 1637, 1638, 1639, 1640, 1641, 1642, 1643, 1644, 1645, 1646, 1647, 1648, 1649, 1650, 1651, 1652, 1653, 1654, 1655, 1656, 1657, 1658, 1659, 1660, 1661, 1662, 1663, 1664, 1665, 1666, 1667, 1668, 1669, 1670, 1671, 1672, 1673, 1674, 1675, 1676, 1677, 1678, 1679, 1680, 1681, 1682, 1683, 1684, 1685, 1686, 1687, 1688, 1689, 1690, 1691, 1692, 1693, 1694, 1695, 1696, 1697, 1698, 1699, 1700, 1701, 1702, 1703, 1704, 1705, 1706, 1707, 1708, 1709, 1710, 1711, 1712, 1713, 1714, 1715, 1716, 1717, 1718, 1719, 1720, 1721, 1722, 1723, 1724, 1725, 1726, 1727, 1728, 1729, 1730, 1731, 1732, 1733, 1734, 1735, 1736, 1737, 1738, 1739, 1740, 1741, 1742, 1743, 1744, 1745, 1746, 1747, 1748, 1749, 1750, 1751, 1752, 1753, 1754, 1755, 1756, 1757, 1758, 1759, 1760, 1761, 1762, 1763, 1764, 1765, 1766, 1767, 1768, 1769, 1770, 1771, 1772, 1773, 1774, 1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1785, 1786, 1787, 1788, 1789, 1790, 1791, 1792, 1793, 1794, 1795, 1796, 1797, 1798, 1799, 1800, 1801, 1802, 1803, 1804, 1805, 1806, 1807, 1808, 1809, 1810, 1811, 1812, 1813, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1840, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849, 1850, 1851, 1852, 1853, 1854, 1855, 1856, 1857, 1858, 1859, 1860, 1861, 1862, 1863, 1864, 1865, 1866, 1867, 1868, 1869, 1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1879, 1880, 1881, 1882, 1883, 1884, 1885, 1886, 1887, 1888, 1889, 1890, 1891, 1892, 1893, 1894, 1895, 1896, 1897, 1898, 1899, 1900, 1901, 1902, 1903, 1904, 1905, 1906, 1907, 1908, 1909, 1910, 1911, 1912, 1913, 1914, 1915, 1916, 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933, 1934, 1935, 1936, 1937, 1938, 1939, 1940, 1941, 1942, 1943, 1944, 1945, 1946, 1947, 1948, 1949, 1950, 1951, 1952, 1953, 1954, 1955, 1956, 1957, 1958, 1959, 1960, 1961, 1962, 1963, 1964, 1965, 1966, 1967, 1968, 1969, 1970, 1971, 1972, 1973, 1974, 1975, 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 207
```

Data EDA

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
# 문장의 길이를 300 으로 맞춥니다. 문장이 길면 자르고 문장이 짧으면 padding 을 수행합니다.
x_train_padded = pad_sequences(x_train, maxlen=300, padding='post')
x_test_padded = pad_sequences(x_test, maxlen=300, padding='post')
```

- 문장을 제한된 길이로 늘리거나 줄이는 작업을 수행함
Padding = 'post' 임으로

Maxlen = 5, padding='post'



1 : !
2 : world
3 : hello
4 : apple

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

- 입력 벡터의 크기를 지정함
- 입력 벡터의 크기는 이전 코드의 pad_sequence 함수의 Maxlen 길이와 같음

```
# 문장의 길이를 300 으로 맞춥니다. 문장이 길면 자르고 문장이 짧으면 padding 을 수행합니다.
x_train_padded = pad_sequences(x_train, maxlen=300, padding='post')
x_test_padded = pad_sequences(x_test, maxlen=300, padding='post')
```

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

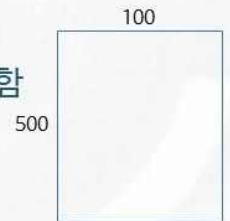
# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

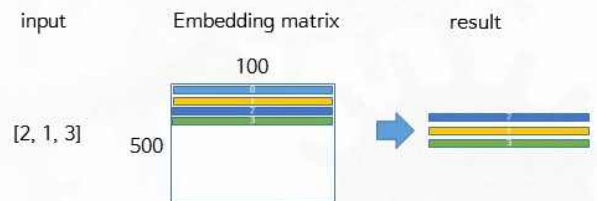
# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

- 크기가 500, 100인 Embedding matrix를 생성함



- 입력값이 들어오면 해당 index에 맞는 Vector를 출력함



Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

→ Convolution layer을 생성함

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

Max pooling layer을 통해 출력 크기를
반으로 줄임

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

- Convolution layer을 생성함
- 입력 크기가 반으로 준 만큼
필터의 개수를 2배 더 증가시킴
- 그리고 2개의 convolution
layer을 통과한 후에는 다시
출력의 크기를 반으로 줄임

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

- Convolution layer을 생성함
- 입력 크기가 반으로 준 만큼 필터의 개수를 2배 더 증가시킴
- 그리고 2개의 convolution layer을 통과한 후에는 다시 출력의 크기를 반으로 줄임

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

- Convolution layer 출력 값을 한 줄로 피는 작업을 수행함

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

- Activation function이 sigmoid인 Dense Layer를 추가함
- Dense Layer를 통과한 Layer은 0, 1의 사이의 확률 값을 가짐
- 우리는 0.5 이상이면 긍정, 0.5 이하면 부정으로 예측함

Model

```
# Model
K.clear_session()

# input define
inputs = Input(shape=(300))

# weight shape (3000, 500)
layer = Embedding(input_dim=500, output_dim=100)(inputs)

# block 1
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=64, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 2
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=128, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

# block 3
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = Conv1D(filters=256, kernel_size=5, activation='relu', kernel_initializer='he_normal')(layer)
layer = MaxPool1D()(layer)

layer = Flatten()(layer)
layer = Dense(1, activation='sigmoid')(layer)
model = Model(inputs, layer)
```

Input, output을 묶어 Keras Model로 만들

Training

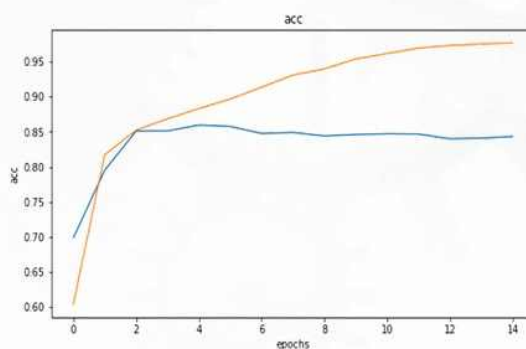
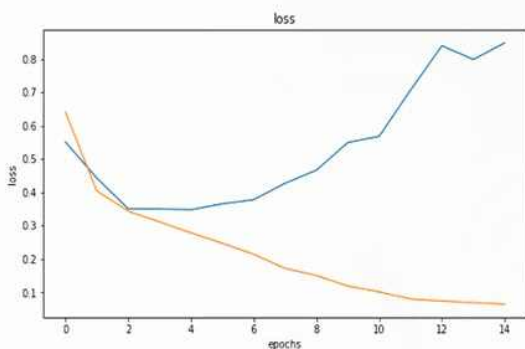
```
model.compile('adam', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(x=padded_train_x, y=y_train, validation_data=(padded_test_x, y_test), epochs=15)
```

- Optimizer는 adam으로, loss function은 binary cross entropy로 평가 방법은 accuracy로 함
- 학습 방법을 지정했다면 이제 학습함

```
# result visualization
fig, axes = plt.subplots(1,2, figsize=(20,5))
axes = np.array(axes).ravel()
axes[0].plot(history.history["val_loss"])
axes[0].plot(history.history["loss"])
axes[0].set_title('loss')
axes[0].set_xlabel('epochs')
axes[0].set_ylabel('loss')

axes[1].plot(history.history["val_acc"])
axes[1].plot(history.history["acc"])
axes[1].set_title('acc')
axes[1].set_xlabel('epochs')
axes[1].set_ylabel('acc')
```

```
Text(0, 0.5, 'acc')
```



Artificial intelligence (AI) refers
to the simulation of human

APPLICATION

03

실습



Artificial intelligence (AI) refers
to the simulation of human

SUMMARY

학습정리

- ◆ Convolution 개요
- ◆ Convolution에 대한 작동 방식
- ◆ 고차원에서의 Convolution
- ◆ 1D Conv, 2D Conv
- ◆ Max-pooling와 Dense Layer
- ◆ 자연어 처리에서의 1D Conv 사용
- ◆ IMDB 데이터셋을 활용한 자연어 처리 실습

확장하기

1. CNN에서의 **Filter**는 어떤 역할을 할까요?
2. CNN에서의 **Convolution 연산**은 어떤 원리로 동작할까요?
3. **Pooling 연산**을 진행하는 이유는 무엇일까요?
4. 자연어처리에서 **Conv1D layer**를 사용하는 이유는 무엇일까요?
5. **Keras**에서 제공하는 데이터셋은 어떤 형태로 존재할까요?



참고 문헌

REFERENCE

◆ 참고 사이트

- 용어들에 대한 정의 : <https://ko.wikipedia.org/wiki>.
- Stanford CS224n : <http://web.stanford.edu/class/cs224n/>
- 퍼블릭에이아이 (www.publicai.co.kr)