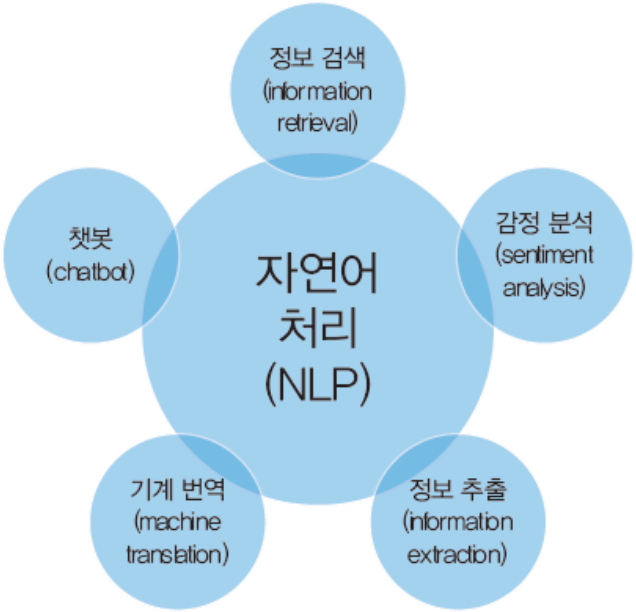
12장 자연어 처리

자연어 처리란?

- 컴퓨터가 자연어를 이해할 수 있다면, 인간과 자연스럽게 대화하는 컴퓨터가 가능할 것이다.
- 자연어 처리의 응용 분야 중 하나가 챗봇이다.

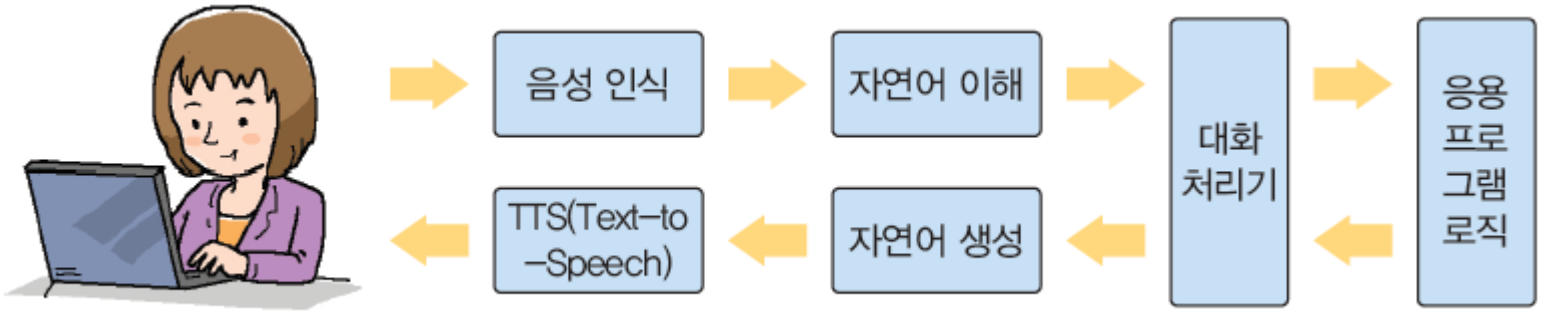
자연어 처리의 역사

- 자연어 처리는 1950년대부터 시작되어, 그동안 많은 연구가 진행되었다.
- 그동안은 단어 간의 통계적인 유사성에 바탕을 둔 방법이 많이 사용되었다.
- 최근에는 딥러닝을 이용한 방법이 많은 인기를 얻고 있다. 자연어 처리는 주로 순환 신경망(RNN)을 많이 이용한다.



음성 인식과 자연어 처리

- 자연어 처리는 텍스트 형태로 자연어를 입력받아서 처리한다.
- 음성 인식(speech recognition)은 음성에서 출발하여서, 컴퓨터를 이용하여 음성을 텍스트로 변환하는 방법론과 기술을 개발하는 분야이다



자연어 처리 라이브러리

- NLTK

```
C> pip install nltk

>>> import nltk
>>> nltk.download()
```

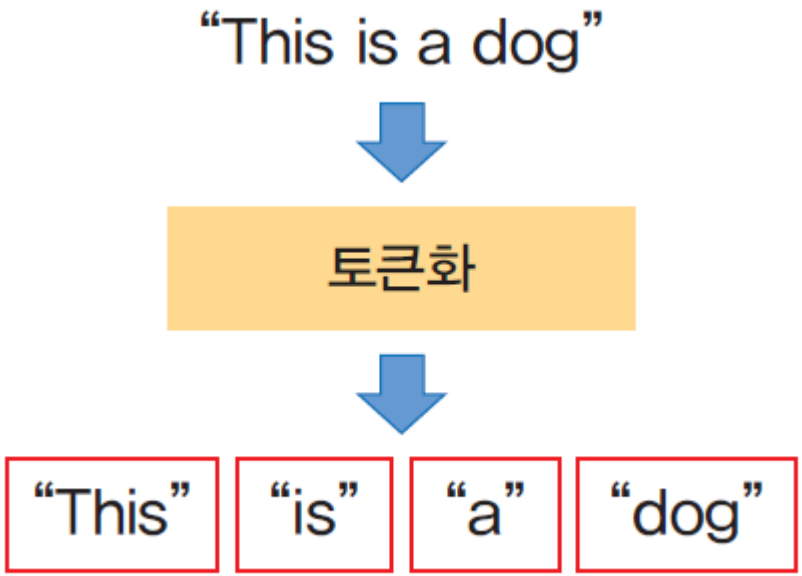
- 케라스의 자연어 처리 함수

텍스트 전처리

- 자연어 처리의 첫 단계는 텍스트 전처리이다. 텍스트를 받아서 토큰으로 분리하는 작업, 각종구두점을 삭제하는 것 등이 전처리에 속한다.

토큰화

- 말뭉치에 포함된 텍스트를 꺼내서 토큰(token)이라 불리는 단위로 나누는 작업을 토큰화(tokenization)라고 한다.



소문자로 변환하기

- 영문에서는 대문자를 소문자로 통합하는 것도, 중요한 전처리 과정이다.
- 대문자로 된 단어를소문자로 변경하면 단어의 개수를 줄일 수 있다. 또 검색할 때, “dog”와 “Dog”가 들어간 문서들을 모두 찾을 수 있다.
- 하지만 무조건 대문자를 소문자로 만들어도 안 된다. 미국을 나타내는 “US”와 우리는 의미하는 “us”는 구분되어야 한다.

구두점 제거하기(정제)

- 만약 토큰화 결과에 구두점이 포함되어 있다면 삭제하는 것이 좋다. 일반적으로 구두점들은 자연어 처리에 도움이 되지 않기 때문이다.

```
from nltk.tokenize import word_tokenize

tokens = word_tokenize("Hello World!, This is a dog.")

# 문자나 숫자인 경우에만 단어를 리스트에 추가한다.
words = [word for word in tokens if word.isalpha()]
print(words)
```

```
['Hello', 'World', 'This', 'is', 'a', 'dog']
```

- 자연어 처리를 하기 전에 말뭉치에서 자연어 분석에 도움에 되지 않는 토큰들을 제거하는 작업이 필요하다.
- 불용어(stopword)는 문장에 많이 등장하지만 큰 의미가 없는 단어들이다

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')

from nltk.corpus import stopwords
print(stopwords.words('english')[:20])
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his']
```

NLTK를 이용한 전처리

- NLTK 라이브러리에는 토큰화를 시킬 수 있는 함수 word_tokenize() 가 포함되어 있다

```
import nltk
nltk.download('punkt') # ①

from nltk.tokenize import word_tokenize # ②

text = "This is a dog." # ③
print(word_tokenize(text))
```

```
['This', 'is', 'a', 'dog', '.']
```

NLTK를 이용한 전처리

```
from nltk.tokenize import sent_tokenize # ①

text = "This is a house. This is a dog."
print(sent_tokenize(text)) # ②
```

```
['This is a house.', 'This is a dog.']
```

Keras를 이용한 전처리

- 공백으로 단어를 분할한다. (split =“ ”)
- 구두점을 필터링한다. (filters = ‘!”# $ % & () * +,-. / :; <=>? @ [\\] ^ _`{|} ~ \ t \ n’)
- 텍스트를 소문자로 변환한다. (lower = True)

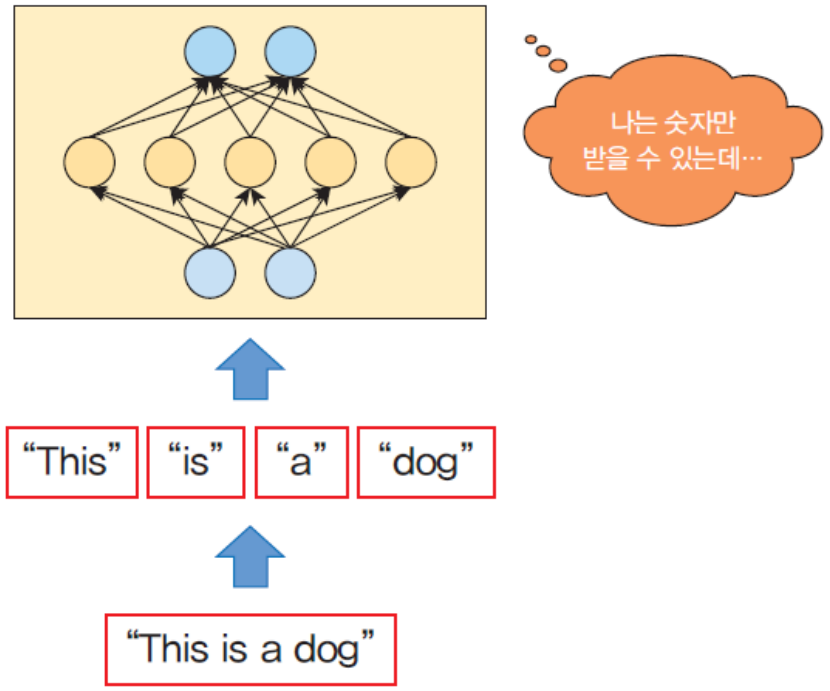
```
from tensorflow.keras.preprocessing.text import *

print(text_to_word_sequence("This is a dog."))
```

```
['this', 'is', 'a', 'dog']
```

단어의 표현

- 심층 신경망이 텍스트(언어)를 처리하려면 신경망이 소화할 수 있는 방식으로, 단어를 신경망에 제공해야 한다.

- 정수 인코딩
- 원-핫 인코딩
- 워드 임베딩



정수 인코딩

- 우리는 고유한 숫자를 사용하여 각 단어를 인코딩할 수 있다.
- 예를 들어서 말뭉치에 단어가 1,000개가 있다면, 각 단어에 1번부터 1,000번까지의 번호(정수)를 매긴다.

"The quick brown fox jumped over the brown dog"

the	quick	brown	fox	jumped	over	the	brown	dog
1	5	12	9	7	2	1	12	25

- 일반적으로는 단어를 빈도순으로 정렬한 후에, 빈도가 높은 단어부터, 번호를 차례대로 부여한다.

원-핫 인코딩(one-hot encoding)

- “원-핫(one-hot)”이라는 의미는, 이진 벡터 중에서 하나만 1이고 나머지는 모두 0이라는 것을 의미한다

케라스에서 원-핫 인코딩 만들기

- NLTK 라이브러리에는 토큰화를 시킬 수 있는 함수 word_tokenize() 가 포함되어 있다

```
import numpy as np
from keras.utils import to_categorical

# 우리가 변환하고 싶은 텍스트
text = ["cat", "dog", "cat", "bird"]

# 단어 집합
total_pets = ["cat", "dog", "turtle", "fish", "bird"]
print("text=", text)
```

케라스에서 원-핫 인코딩 만들기

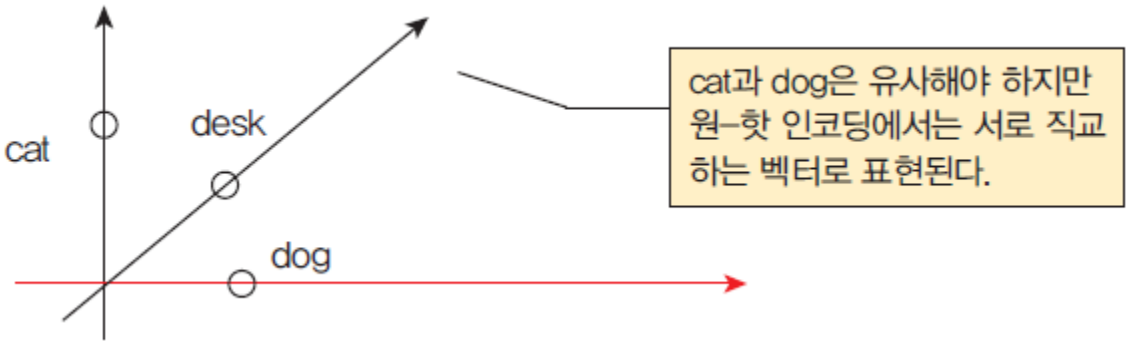
```
# 변환에 사용되는 딕셔너리를 만든다.
mapping = {}
for x in range(len(total_pets)):
  mapping[total_pets[x]] = x   #“cat"->0, "dog"->1, ...
print(mapping)

# 단어들을 순차적인 정수 인덱스로 만든다.
for x in range(len(text)):
  text[x] = mapping[text[x]]

print("text=", text)

# 순차적인 정수 인덱스를 원-핫 인코딩으로 만든다.
one_hot_encode = to_categorical(text)
print("text=", one_hot_encode)
```

실행 결과

```
text= ['cat', 'dog', 'cat', 'bird']
{'cat': 0, 'dog': 1, 'turtle': 2, 'fish': 3, 'bird': 4}
text= [0, 1, 0, 4]
text= [[1. 0. 0. 0. 0.]
       [0. 1. 0. 0. 0.]
       [1. 0. 0. 0. 0.]
       [0. 0. 0. 0. 1.]]
```

원-핫 인코딩의 약점

- 원-핫 인코딩의 최대 약점은 무엇일까? 비효율성이다. 원-핫 인코딩된 벡터는 희소하다. 즉 벡터의 대부분이 0이다. 예를 들어서 단어 집합에 10,000개의 단어가 있다고 가정하자. 각 단어를 원-핫 인코딩하면, 99.99%가 0인 벡터가 10,000개가 만들어진다
- 각 벡터들은 단어들 간의 유사도를 표현하지 못한다.



워드 임베딩

- 워드 임베딩(word embedding)은 하나의 단어를 밀집 벡터(dense vector)로 표현하는 방법이다.

- 단어를 표현하는 벡터들은 일반적으로 단어의 개수보다는 무척 차원이 작다. 이들 벡터들은 효율적이고 조밀하다.

- 이들 임베딩 벡터들은 학습을 통하여 훈련 데이터에서 자동으로 생성하는 것이 일반적이다. 신경망이 주로 사용된다

워드 임베딩

the →	1.3	−0.2	3.3	2.3
cat →	0.5	2.6	−0.8	0.4
sat →	0.4	3.1	−0.9	0.7
on →	2.1	0.2	0.1	0.4
...	...			

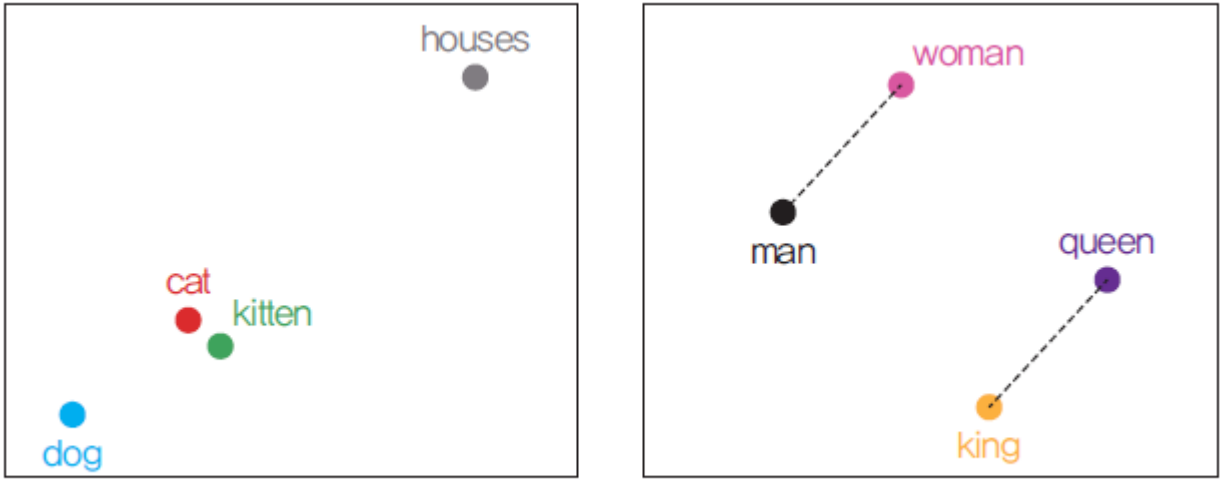


그림 12-1 워드 임베딩의 결과를 차원 축소해서 그린 것

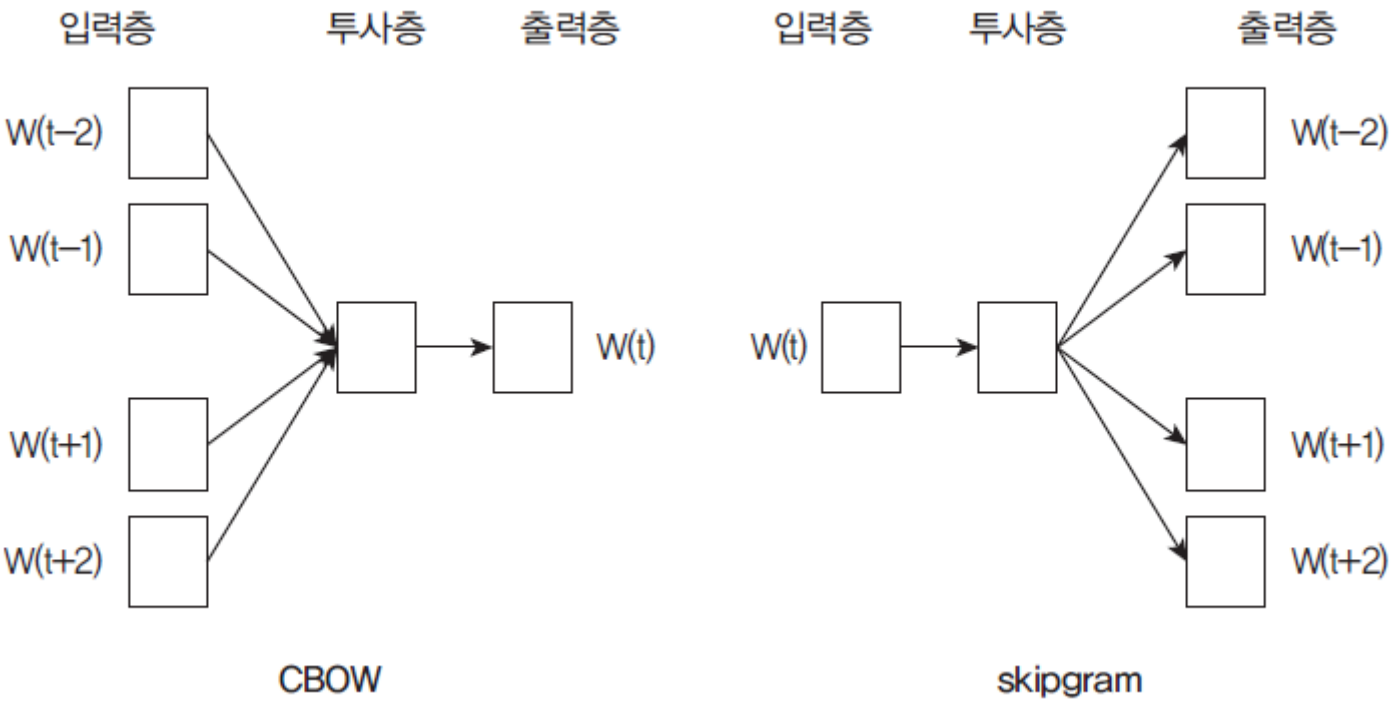
Word2vec

- Word2Vec은 단어 임베딩의 한 가지 방법이다. Word2Vec은 아래 다이어그램에 표시된 것처럼 텍스트 말뭉치를 입력으로 받아들이고 각 단어에 대한 벡터 표현을 출력하는 알고리즘이다.

- CBOW(Continuous Bag of Words) 모델
- skipgram 모델

CBOW vs skipgram

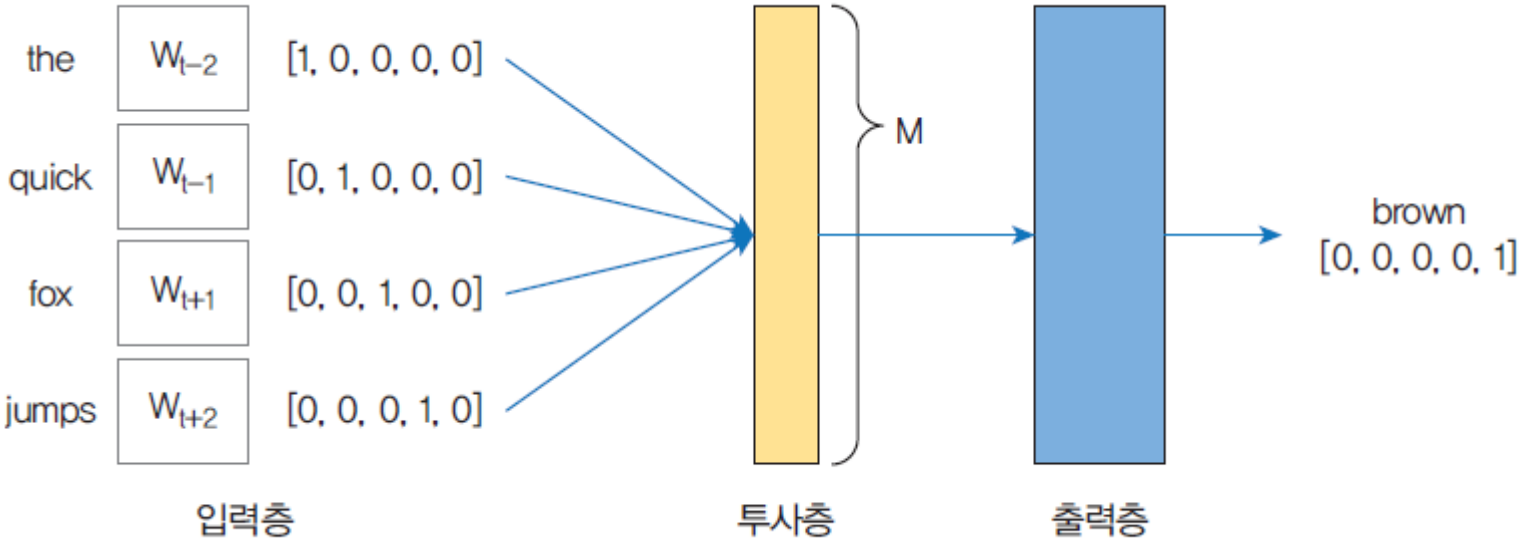
- CBOW는 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측하는 방법
- skipgram은 중간에 있는 단어로 주변 단어들을 예측하는 방법



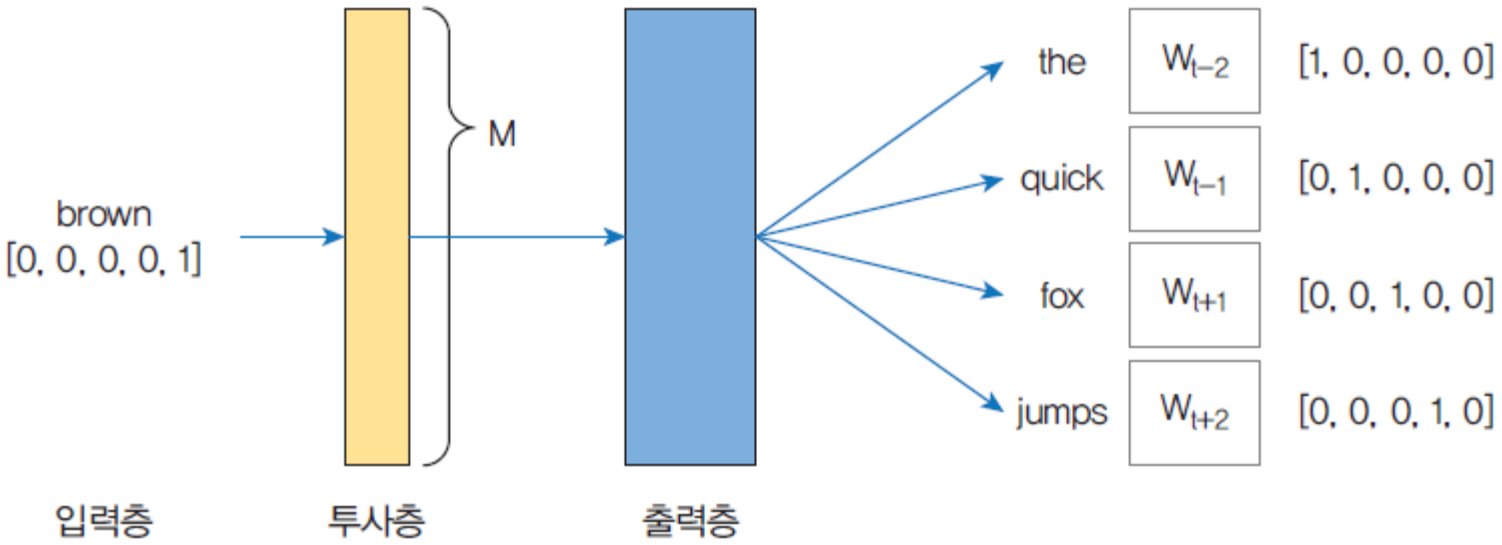
CBOW


소스 텍스트
훈련 샘플들
The quick brown fox jumps over the lazy dog.
(the, quick)
(the, brown)
The quick brown fox jumps over the lazy dog.
(quick, the)
(quick, brown)
(quick, fox)
The quick brown fox jumps over the lazy dog.
(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)
The quick brown fox jumps over the lazy dog.
(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

CBOW


the
W_{t-2}
[1, 0, 0, 0, 0]
quick
W_{t-1}
[0, 1, 0, 0, 0]
fox
W_{t+1}
[0, 0, 1, 0, 0]
jumps
W_{t+2}
[0, 0, 0, 1, 0]
M
brown
[0, 0, 0, 0, 1]
입력층
투사층
출력층

skipgram

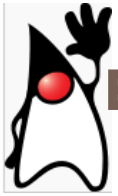

M
brown
[0, 0, 0, 0, 1]
the
W_{t-2}
[1, 0, 0, 0, 0]
quick
W_{t-1}
[0, 1, 0, 0, 0]
fox
W_{t+1}
[0, 0, 1, 0, 0]
jumps
W_{t+2}
[0, 0, 0, 1, 0]
입력층
투사층
출력층

케라스에서의 자연어 처리

- 전처리와 토큰화

```
from tensorflow.keras.preprocessing.text import Tokenizer

t = Tokenizer()
text = """Deep learning is part of a broader family of machine learning methods
          based on artificial neural networks with representation learning."""

t.fit_on_texts([text])
print("단어집합 : ", t.word_index)
```

```
단어집합 : {'learning': 1, 'of': 2, 'deep': 3, 'is': 4, 'part': 5, 'a': 6, 'broader': 7,
'family': 8, 'machine': 9, 'methods': 10, 'based': 11, 'on': 12, 'artificial': 13,
'neural': 14, 'networks': 15, 'with': 16, 'representation': 17}
```



텍스트의 정수 인코딩

```
seq = t.texts_to_sequences([text])[0]
print(text,"->", seq)
```

Deep learning is part of a broader family of machine learning methods based on artificial neural networks with representation learning. -> [3, 1, 4, 5, 2, 6, 7, 8, 2, 9, 1, 10, 11, 12, 13, 14, 15, 16, 17, 1]

샘플의 패딩

- 샘플의 길이가 다를 수 있다.
- 특히 텍스트를 문장 단위로 분리하여 신경망을 훈련시키고자 할 때 많이 발생한다.
- 보통 숫자 0을 넣어서, 길이가 다른 샘플들의 길이를 맞추게 되는데 이것을 패딩(padding)이라고 한다. 케라스에서는 pad_sequence()를 지원한다.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

X = pad_sequences([[7, 8, 9], [1, 2, 3, 4, 5], [7]], maxlen=3, padding='pre')
print(X)
```

```
[[7 8 9]
[3 4 5]
[0 0 7]]
```

pad_sequences()의 매개 변수

- pad_sequences(sequences, maxlen=None, padding='pre',
  truncating='pre', value=0.0)
  - sequences = 패딩이 수행되는 시퀀스 데이터
  - maxlen = 샘플의 최대 길이
  - padding = ‘pre’이면 앞에 0을 채우고 ‘post’이면 뒤에 0을 채운다.

케라스 Embedding 레이어

- 케라스는 텍스트 데이터를 처리하는 신경망에 사용할 수 있는 Embedding 레이어를 제공한다.

- Embedding 레이어의 입력 데이터는 정수 인코딩되어서, 각 단어가 고유한 정수로 표현되어야 한다.

케라스 Embedding 레이어

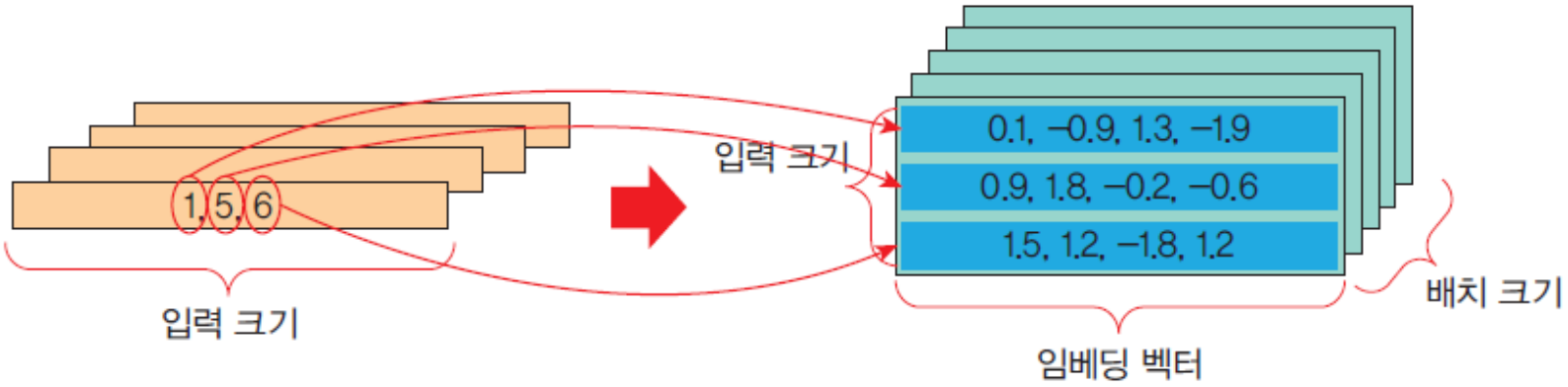
- e = Embedding(input_dim, output_dim, input_length=100)
  - input_dim: 이것은 텍스트 데이터의 어휘 크기이다. 예를 들어 데이터가 0~9 사이의 값으로 정수 인코딩된 경우 어휘의 크기는 10단어가 된다.
  - output_dim: 이것은 단어가 표현되는 벡터 공간의 크기이다. 각 단어에 대해 이 레이어의 출력 벡터 크기를 정의한다. 예를 들어 32 또는 100 이상일 수 있다.
  - input_length: 입력 시퀀스의 길이이다. 예를 들어 모든 입력 문서가 100개의 단어로 구성되어 있으면 100이 된다.
  - 입력 형태: 2D 텐서 (batch_size, sequence_length)의 형태이다. 정수 인코딩 형태이어야 한다. 즉 정수의 시퀀스이어야 한다.
  - 출력 형태: 3D 텐서 (batch_size, sequence_length, output_dim)의 형태이다.

Embedding 레이어 예제

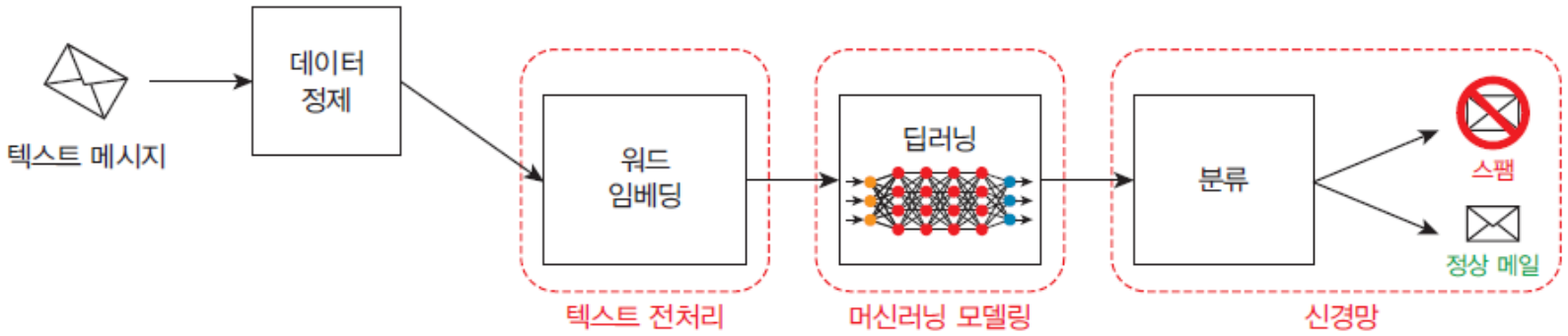
```
import numpy as np
from tensorflow.keras.layers import Embedding
from tensorflow.keras.models import Sequential

# 입력 형태: (batch_size, input_length)=(32, 3)
# 출력 형태: (None, 3, 4)
model = Sequential()
model.add(Embedding(100, 4, input_length=3))

input_array = np.random.randint(100, size=(32, 3))
model.compile('rmsprop', 'mse')
output_array = model.predict(input_array)
print(output_array.shape)
```

```
(32, 3, 4)
```

Embedding 레이어 예제



예제: 스팸 메일 분류하기



예제: 스팸 메일 분류하기

```
import numpy as np
from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences

docs = [    'additional income',
                        'best price',
                        'big bucks',
                        'cash bonus',
                        'earn extra cash',
                        'spring savings certificate',
                        'valero gas marketing',
                        'all domestic employees',
                        'nominations for oct',
                        'confirmation from spinner']
```

예제: 스팸 메일 분류하기

```
labels = np.array([1,1,1,1,1,0,0,0,0,0])

vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
print(encoded_docs)
```

```
[[30, 24], [30, 29], [1, 9], [49, 46], [29, 47, 49], [23, 39, 47], [14, 20, 31], [17,
22, 3], [42, 25, 37], [41, 5, 9]]
```

예제: 스팸 메일 분류하기

```
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

```
[[30 24 0 0]
[30 29 0 0]
[ 1 9 0 0]
[49 46 0 0]
[29 47 49 0]
[23 39 47 0]
[14 20 31 0]
[17 22 3 0]
[42 25 37 0]
[41 5 9 0]]
```

예제: 스팸 메일 분류하기

```
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

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

model.fit(padded_docs, labels, epochs=50, verbose=0)

loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('정확도=', accuracy
```

```
정확도= 1.0
```

예제: 스팸 메일 분류하기

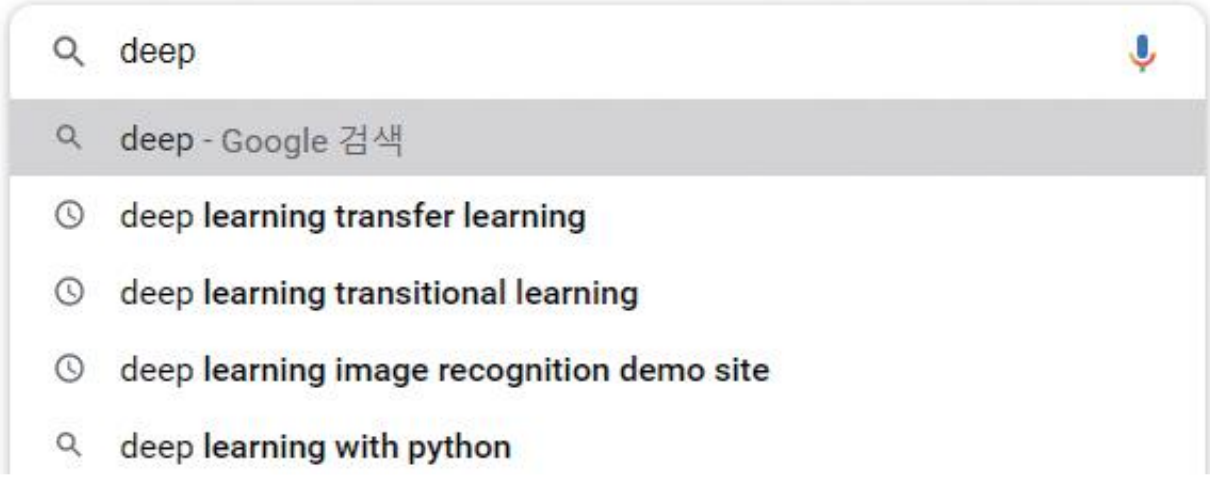
```
test_doc = ['big income']
encoded_docs = [one_hot(d, vocab_size) for d in test_doc]
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')

print(model.predict(padded_docs))
```


```
test_doc = ['big income']
encoded_docs = [one_hot(d, vocab_size) for d in test_doc]
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')

print(model.predict(padded_docs))
```

```
[[0.5746514]]
```

예제: 다음 단어 예측하기

- 자연어 처리에서 많이 등장하는 문제 중 하나는, 이전 단어를 고려하여 다음 단어를 예측하는 것이다. 우리는 구글이나 네이버에서 검색할 때 이것을 자주 본다.



예제: 다음 단어 예측하기

- 이번 절에서는 노래 가사에서 짧은 단어 시퀀스를 추출하고 이것으로 학습을 시켜서 어떻게 단어가 예측되는지 살펴보자. 일단 가장 간단한 모델부터 시작하자.

X	y
deep	learning
python	download
learning	python
stock	investing
...	...

예제: 다음 단어 예측하기

```
import numpy as np
from tensorflow.keras.layers import Embedding, Flatten, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

text_data="""Soft as the voice of an angel\n
Breathing a lesson unhead\n
Hope with a gentle persuasion\n
Whispers her comforting word\n
Wait till the darkness is over\n
Wait till the tempest is done\n
Hope for sunshine tomorrow\n
After the shower
"""
```

예제: 다음 단어 예측하기

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text_data])
encoded = tokenizer.texts_to_sequences([text_data])[0]
print(encoded)
```

```
[7, 8, 1, 9, 10, 11, 12, 13, 2, 14, 15, 3, 16, 2, 17, 18, 19, 20, 21, 22, 4, 5, 1,
23, 6, 24, 4, 5, 1, 25, 6, 26, 3, 27, 28, 29, 30, 1, 31]
```

예제: 다음 단어 예측하기

```
print(tokenizer.word_index)
vocab_size = len(tokenizer.word_index) + 1
print('어휘 크기: %d' % vocab_size)
```

```
{'the': 1, 'a': 2, 'hope': 3, 'wait': 4, 'till': 5, 'is': 6, 'soft': 7, 'as':
8, 'voice': 9, 'of': 10, 'an': 11, 'angel': 12, 'breathing': 13, 'lesson': 14,
'unhead': 15, 'with': 16, 'gentle': 17, 'persuasion': 18, 'whispers': 19, 'her': 20,
'comforting': 21, 'word': 22, 'darkness': 23, 'over': 24, 'tempest': 25, 'done':
26, 'for': 27, 'sunshine': 28, 'tomorrow': 29, 'after': 30, 'shower': 31}
어휘 크기: 32
```

예제: 다음 단어 예측하기

```
sequences = list()
for i in range(1, len(encoded)):
        sequence = encoded[i-1:i+1]
        sequences.append(sequence)
print(sequences)
print('총 시퀀스 개수: %d' % len(sequences))
```

```
[[7, 8], [8, 1], [1, 9], [9, 10], [10, 11], [11, 12], [12, 13], [13, 2], [2, 14], [14,
15], [15, 3], [3, 16], [16, 2], [2, 17], [17, 18], [18, 19], [19, 20], [20, 21], [21,
22], [22, 4], [4, 5], [5, 1], [1, 23], [23, 6], [6, 24], [24, 4], [4, 5], [5, 1], [1,
25], [25, 6], [6, 26], [26, 3], [3, 27], [27, 28], [28, 29], [29, 30], [30, 1], [1,
31]]
총 시퀀스 개수: 38
```

예제: 다음 단어 예측하기

```
sequences = np.array(sequences)
X, y = sequences[:,0],sequences[:,1]
print("X=", X)
print("y=", y)
```

```
X= [ 7 8 1 9 10 11 12 13 2 14 15 3 16 2 17 18 19 20 21 22 4 5 1 23
6 24 4 5 1 25 6 26 3 27 28 29 30 1]
y= [ 8 1 9 10 11 12 13 2 14 15 3 16 2 17 18 19 20 21 22 4 5 1 23 6
24 4 5 1 25 6 26 3 27 28 29 30 1 31]
```

신경망 모델 정의

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN, LSTM

model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=1))
model.add(LSTM(50))
model.add(Dense(vocab_size, activation='softmax'))
```

컴파일 및 학습

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

model.fit(X, y, epochs=500, verbose=2)
```

```
...
2/2 - 0s - loss: 0.3056 - accuracy: 0.8421
Epoch 500/500
2/2 - 0s - loss: 0.3053 - accuracy: 0.8421
```

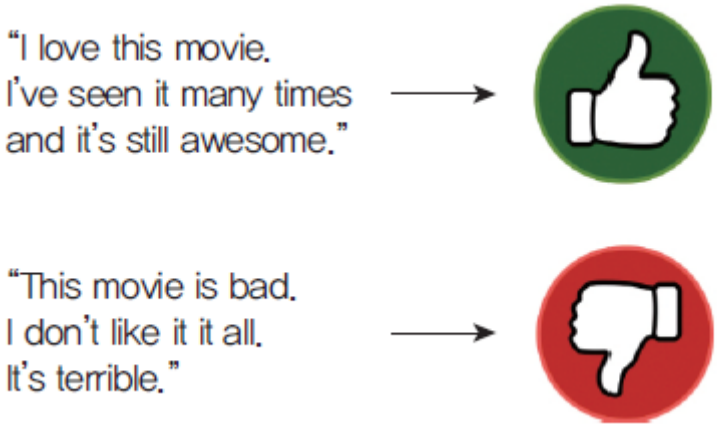
테스트

```
# 테스트 단어를 정수 인코딩한다.
test_text = 'Wait'
encoded = tokenizer.texts_to_sequences([test_text])[0]
encoded = np.array(encoded)

# 신경망의 예측값을 출력해본다.
onehot_output = model.predict(encoded)
print('onehot_output=', onehot_output)

# 가장 높은 출력을 내는 유닛을 찾는다.
output = np.argmax(onehot_output)
print('output=', output)

# 출력층의 유닛 번호를 단어로 바꾼다.
print(test_text, "=>", end=" ")
for word, index in tokenizer.word_index.items():
        if index == output:
                print(word)
```

onehot_output= [[5.6587060e-06 4.0273936e-03 6.2348531e-03 8.6066285e-03
1.5018744e-05
9.7867030e-01 6.0064325e-05 5.6844797e-06 1.7885073e-05 8.7188082e-06
3.1823198e-05 2.8105886e-04 3.3933269e-05 1.3699831e-06 3.2279258e-06
7.1233828e-08 2.3057231e-05 3.4311252e-06 1.0358917e-05 7.9929187e-08
1.4642384e-04 1.2234473e-06 1.0129405e-04 1.5086286e-05 8.9766763e-06
8.6069404e-06 9.3199278e-06 3.4068940e-05 2.0460826e-08 1.5851222e-03
3.8524475e-05 1.0519097e-05]]
output= [5]
Wait => till

예제: 영화 리뷰 감성 판별하기

- IMDB 사이트는 영화에 대한 리뷰가 올려져 있는 사이트이다. 2011년에 스탠포드 대학교에서는 이 리뷰 데이터를 사용하여 영화 리뷰가 긍정적인지 부정적인지를 학습하는 논문을 발표하였다.
- 데이터를 imdb.load_data() 함수를 통해 바로 다운로드 할 수 있다.



라이브러리 포함

```
import numpy as np

import tensorflow as tf
from tensorflow import keras

import matplotlib.pyplot as plt
```

데이터를 다운로드한다.

```
imdb = keras.datasets.imdb
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=10000)

print(x_train[0])
```

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, ...
,32]
```

리뷰 복원

```
# 단어 ->정수 인덱스 딕셔너리
word_to_index = imdb.get_word_index()

# 처음 몇 개의 인덱스는 특수 용도로 사용된다.
word_to_index = {k:(v+3) for k,v in word_to_index.items()}
word_to_index["<PAD>"] = 0                  # 문장을 채우는 기호
word_to_index["<START>"] = 1                # 시작을 표시
word_to_index["<UNK>"] = 2                  # 알려지지 않은 토큰
word_to_index["<UNUSED>"] = 3

index_to_word = dict([(value, key) for (key, value) in word_to_index.items()])

print(' '.join([index_to_word[index] for index in x_train[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 redford's is an amazing actor and ... the whole story was so lovely because it was true and was someone's life after all that was shared with us all

전처리

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *

x_train = pad_sequences(x_train, maxlen=100)
x_test = pad_sequences(x_test, maxlen=100)

vocab_size = 10000
```

신경망 구축

```
model = Sequential()
model.add(Embedding(vocab_size, 64,
                    input_length=100))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

신경망 구축

```
Model: "sequential_3"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
embedding_3 (Embedding) (None, 100, 64) 640000
_________________________________________________________________
flatten_2 (Flatten) (None, 6400) 0
_________________________________________________________________
dense_3 (Dense) (None, 64) 409664
_________________________________________________________________
dropout (Dropout) (None, 64) 0
_________________________________________________________________
dense_4 (Dense) (None, 1) 65
=================================================================
Total params: 1,049,729
Trainable params: 1,049,729
Non-trainable params: 0
_________________________________________________________________
```

모델 컴파일

```
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])
history = model.fit(x_train, y_train,
          batch_size=64, epochs=20, verbose=1,
          validation_data=(x_test, y_test))
```

```
...
Epoch 18/20
391/391 [==============================] - 2s 5ms/step - loss: 0.0545 - accuracy:
0.9882 - val_loss: 0.8295 - val_accuracy: 0.8112
Epoch 19/20
391/391 [==============================] - 2s 5ms/step - loss: 0.0527 - accuracy:
0.9890 - val_loss: 0.8738 - val_accuracy: 0.8092
Epoch 20/20
391/391 [==============================] - 2s 6ms/step - loss: 0.0417 - accuracy:
0.9924 - val_loss: 0.9395 - val_accuracy: 0.8077
```

모델 평가

```
results = model.evaluate(x_test, y_test, verbose=2)
print(results)
```

```
782/782 - 1s - loss: 0.9395 - accuracy: 0.8077
[0.93949955701828, 0.8076800107955933]
```

우리가 작성한 리뷰로 테스트해보자.

review = "What can I say about this movie that was already said? It is my favorite time travel sci-fi, adventure epic comedy in the 80's and I love this movie to death! When I saw this movie I was thrown out by its theme. An excellent sci-fi, adventure epic, I LOVE the 80s. It's simple the greatest time travel movie ever happened in the history of world cinema. I love this movie to death, I love, LOVE, love it!"

우리가 작성한 리뷰로 테스트해보자.

```
import re
review = re.sub("[^0-9a-zA-Z ]", "", review).lower()

review_encoding = []
# 리뷰의 각 단어 대하여 반복한다.
for w in review.split():
                        index = word_to_index.get(w, 2)         # 딕셔너리에 없으면 2 반
환
                        if index <= 10000:                  # 단어의 개수는 10000이하
                                review_encoding.append(index)
                        else:
                                review_encoding.append(word_to_index["UNK"])

# 2차원 리스트로 전달하여야 한다.
test_input = pad_sequences([review_encoding], maxlen = 100)
value = model.predict(test_input) # 예측
if(value > 0.5):
        print("긍정적인 리뷰입니다.")
else:
        print("부정적인 리뷰입니다.")
```

실행 결과

긍정적인 리뷰입니다.