

17.

딥러닝을 이용한 자연어 처리

목차

- 1 텍스트의 토큰화**
- 2 단어의 원-핫 인코딩과 임베딩**
- 3 텍스트를 읽고 긍정/부정 예측하기**

텍스트의 토큰화

자연어처리

음성이나 텍스트를
컴퓨터가 인식하고 처리



랩 해줘

시간이 흘러도 변하지않는것
리듬 위의 빅스비
보이는게 다가 아냐 말을하면 알겠지
다시한번 말해줘요 빅스비
내가 랩을 할테니까 노래를 불러줘요
가슴깊이 새겨둬요 빅스비
낮 놓고 기억자도 모른대도 상관없어
다시 한번 말해줘요 빅스비

여러분의 평가가 Bixby의 성장에 도움이 됩니다.



**딤러닝
이전**

딤러닝

현재

언어의 규칙 ≠ 컴퓨터의 규칙 → 여러문제 발생

[언어의 중의성]

**“ 차를 마시러 공원에 가는 차 안에서
나는 그녀에게 차였다.”**

**딤러닝
이전**

딤러닝

현재

“ 언어의 규칙 ≠ 컴퓨터의 규칙 → 여러문제 발생 ”


[다양한 표현]



**곰돌이 모자를 쓴 고양이가
메롱하고 있다.**

**메롱냥이가 곰돌이 모자를
쓴다.**

**수염이 하얗고 눈이
반짝인다.**



**딥러닝
이전**

딥러닝

현재

1. 대용량의 자연어 데이터

2. 지속적인 학습

TOKEN

텍스트를 단어, 문장, 형태소별로
작게 나눈 하나의 단위

TOKENIZ ATION

입력된 텍스트를 잘게 나누는 과정

텍스트의 토큰화 - '단어'

text_to_word_sequence() : 문장을 단어 단위로 나눌 수 있는 함수

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

```
text = "해보지 않으면 해낼 수 없다"  
result = text_to_word_sequence(text)  
print(result)
```

```
['해보지', '않으면', '해낼', '수', '없다']
```

해보지 않으면 해낼 수 없다



해보지 / 않으면 / 해낼 / 수 / 없다

텍스트의 토큰화 - 'Bag-of-Word'

Tokenizer()

: 단어의 빈도 수를 계산하는 함수

```
from keras.preprocessing.text import Tokenizer

docs = ['먼저 텍스트의 각 단어를 나누어 토큰화 합니다.',
        '텍스트의 단어로 토큰화 해야 딥러닝에서 인식됩니다.',
        '토큰화 한 결과는 딥러닝에서 사용할 수 있습니다.'],

token = Tokenizer() # 토큰화 함수 지정
token.fit_on_texts(docs) # 토큰화 함수에 문장 적용
```

- 먼저 텍스트의 각 단어를 나누어 토큰화 합니다.
- 텍스트의 단어로 토큰화 해야 딥러닝에서 인식됩니다.
- 토큰화 한 결과는 딥러닝에서 사용할 수 있습니다.

텍스트의 토큰화 - 'Bag-of-Word'

총 문장의 개수 계산

각 단어의 인덱스 값

word_counts

**document
_counts**

word_docs

word_index

단어의 빈도 수 계산

단어가 위치한 문장

텍스트의 토큰화 - 'Bag-of-Word'



word_counts

단어의 빈도 수 계산

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

```
docs = ['먼저 텍스트의 각 단어를 나누어 토큰화 합니다.',  
        '텍스트의 단어로 토큰화 해야 딥러닝에서 인식됩니다.',  
        '토큰화 한 결과는 딥러닝에서 사용할 수 있습니다.'],
```

```
token = Tokenizer() # 토큰화 함수 지정  
token.fit_on_texts(docs) # 토큰화 함수에 문장 적용
```

```
# 단어의 빈도 수를 계산한 결과 출력  
print("단어 카운트 :", token.word_counts)
```

```
단어 카운트 : OrderedDict([('먼저', 1), ('텍스트의', 2), ('각', 1), ('단어를', 1), ('나누어', 1), ('토큰화', 3), ('합니다', 1),  
                           ('단어로', 1), ('해야', 1), ('딥러닝에서', 2), ('인식됩니다', 1), ('한', 1), ('결과는', 1), ('사용할', 1), ('수', 1), ('있습니다', 1)])
```

텍스트의 토큰화 - 'Bag-of-Word'

총 문장의 개수 계산



**document
_counts**

```
# 총 문장의 개수  
print("문장 카운트 :", token.document_count)
```

문장 카운트 : 3

텍스트의 토큰화 - 'Bag-of-Word'

```
# 각 단어들이 몇 개의 문장에서 나오는 결과 출력
```

```
print("각 단어가 몇 개의 문장에 포함되어 있는가 :", token.word_docs)
```

```
각 단어가 몇 개의 문장에 포함되어 있는가 : defaultdict(<class 'int'>, {'각': 1, '텍스트의': 2,
```

```
('단어를', 1), ('나누어', 1), ('토큰화', 3), ('합니다', 1), ('단어로', 1), ('해야', 1)
```

```
('딥러닝에서', 2), ('인식됩니다', 1), ('한', 1), ('결과는', 1), ('사용할', 1), ('수', 1)
```

- 먼저 텍스트의 각 단어를 나누어 토큰화 합니다.
- 텍스트의 단어로 토큰화 해야 딥러닝에서 인식됩니다.
- 토큰화 한 결과는 딥러닝에서 사용할 수 있습니다.



word_docs

단어가 위치한 문장

텍스트의 토큰화 - 'Bag-of-Word'

```
# 각 단어에 매겨진 인덱스 값 출력  
print("각 단어에 매겨진 인덱스 값 : ", token.word_index)
```

```
각 단어에 매겨진 인덱스 값 : {'토큰화': 1, '텍스트의': 2, '딥러닝에서': 3, '먼저': 4, '각': 5, '단어를': 6,
```

```
'나누어': 7, '합니다': 8, '단어로': 9, '해야': 10, '인식됩니다': 11, '한': 12, '결과는': 13, '사용할': 14,
```

```
'사용할': 14, '수': 15, '있습니다': 16}
```

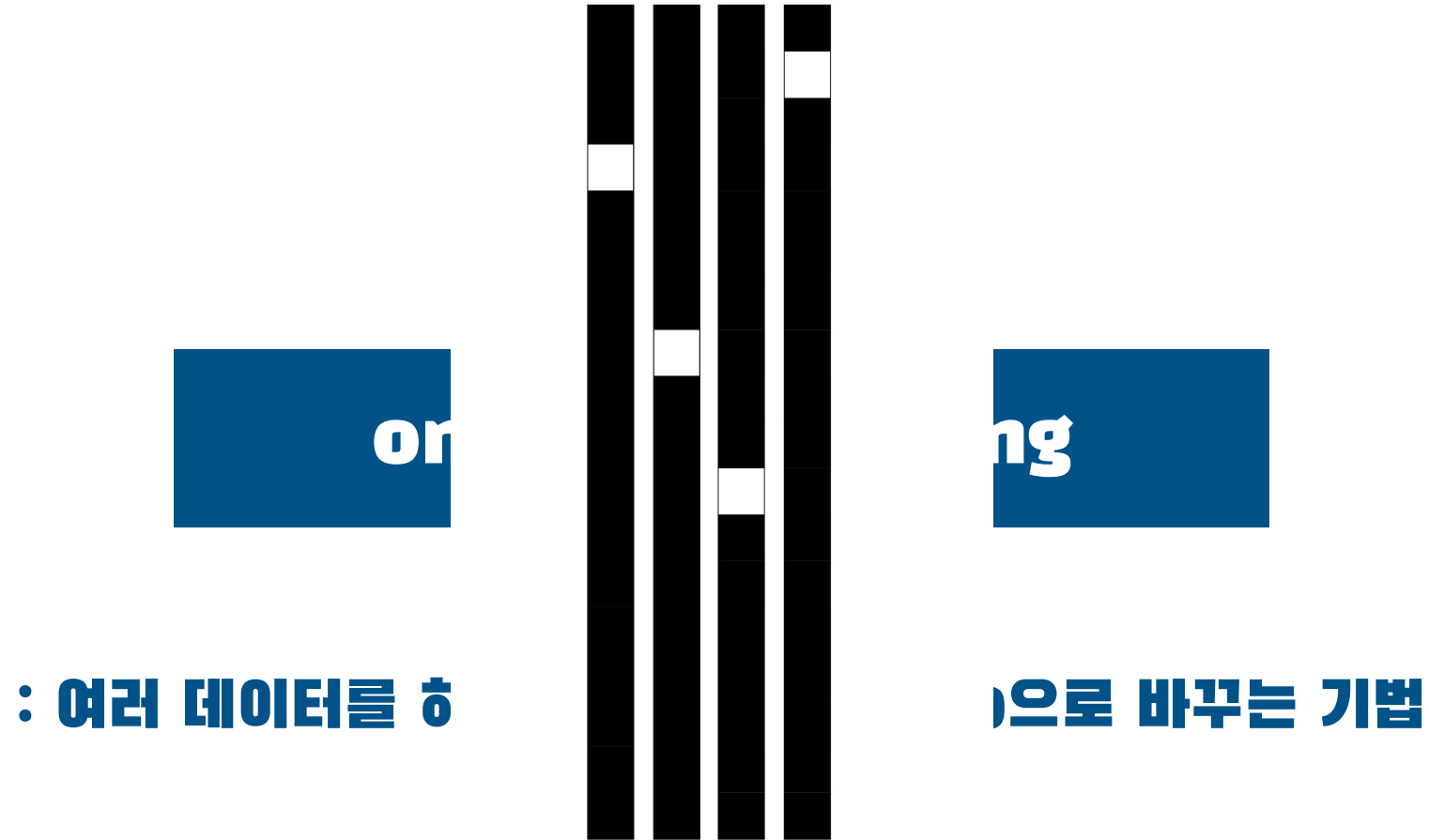
각 단어의 인덱스 값



word_index

단어의 원-핫 임베딩과 임베딩

단어의 원-핫 인코딩



One-hot word vectors:

- Sparse
- High-dimensional
- Hard-coded

단어의 원-핫 인코딩

Tokenize

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

text = "오랫동안 꿈꾸는 이는 그 꿈을 닮아간다"

token = Tokenizer()
token.fit_on_texts([text])    #토큰화하기 위해 .fit_on_texts()에 입력

print(token.word_index)

{'오랫동안': 1, '꿈꾸는': 2, '이는': 3, '그': 4, '꿈을': 5, '닮아간다': 6}
```

word_index 형태

단어의 원-핫 인코딩

text_to_sequences()

```
# 2. 토큰의 인덱스로만 채워진 새로운 배열 생성
```

```
x = token.texts_to_sequences([text])
```

```
print(x)
```

```
[[1, 2, 3, 4, 5, 6]]
```

sequence 형태

단어의 원-핫 인코딩

to_categorical()

3. 원-핫 인코딩 함수 to_categorical() 사용

```
from keras.utils import to_categorical
```

#인덱스 수에 하나를 추가해서 원-핫 인코딩 배열 생성

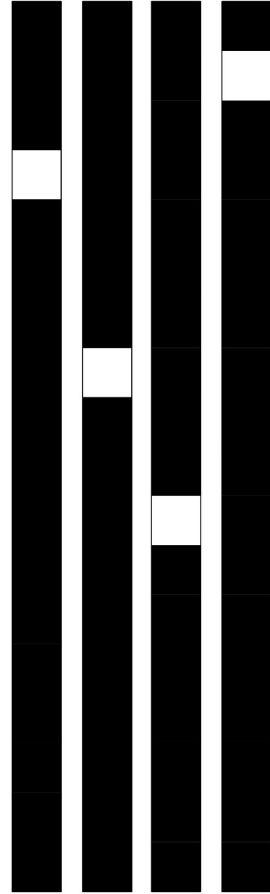
word_size = len(token.word_index) + 1 #맨 앞 주소는 인덱스이므로 0으로 비워두어야 하기 때문

x = to_categorical(x, num_classes=word_size) #x를 입력값으로 하고 분류의 개수(데이터 개수)를 word_size로 한다

```
print(x)
```

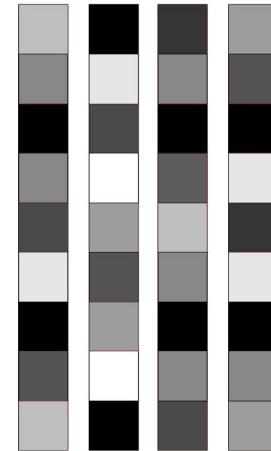
```
[[ 0  1.  0.  0.  0.  0.  0.]  
 [ 0  0.  1.  0.  0.  0.  0.]  
 [ 0  0.  0.  1.  0.  0.  0.]  
 [ 0  0.  0.  0.  1.  0.  0.]  
 [ 0  0.  0.  0.  0.  1.  0.]  
 [ 0  0.  0.  0.  0.  0.  1.]]
```

원-핫 인코딩 진행



One-hot word vectors:

- Sparse
- High-dimensional
- Hard-coded



Word embeddings:

- Dense
- Lower-dimensional
- Learned from data

임베딩

Embedding()

```
[18] from keras.models import Sequential
      from keras.layers import Embedding

      model = Sequential()
      model.add(Embedding(16, 4)) ##입력될 총 단어의 수는 16개, 임베딩 후 출력되는 벡터 크기는 4.
```

Embedding(samples, sequence_length, input_length) 함수를 통해 임베딩 공간을 생성한 모습이다.

임베딩 공간에서는 단어 간의 유사도를 계산하여 원-핫 인코딩의 각 벡터를 새로운 수치로 변환한다. 해당 과정에서는 오차 역전파가 사용된다. 함수의 매개변수는 사용할 단어의 개수(samples), 한번에 사용할 단어의 개수(input_length)로 설정한다.

임베딩 진행

실습문제.

텍스트를 읽고 긍정 부정 예측하기

배열로 변환, 데이터 길이 맞추기

```
x = token.texts_to_sequences(docs)
print(x)
```

- **text_to_sequence** : 텍스트를 숫자로 이루어진 배열로 바꾸기

딥러닝 모델에 입력을 하려면 학습 데이터의 길이가 동일해야함

```
padded_x = pad_sequences(x, 4)
print(padded_x)
```

- **pad_sequence** : 길이보다 짧은 부분은 숫자 0으로 채워주고 긴 데이터는 잘라서 같은 길이로 만들어줌

임베딩

자연어 처리에서 사람이 쓰는 자연어를 기계가 이해할 수 있도록 숫자형태인 vector로 바꾸는 과정 혹은 일련의 전체 과정

- 단어나 문장 각각을 벡터로 변환해 벡터 공간으로 끼워 넣는다는 의미

EX)

구분	메밀꽃 필 무렵	운수 좋은 날	사랑 손님과 어머니	삼포 가는 길
기차	0	2	10	7
막걸리	0	1	0	0
선술집	0	1	0	0

가장 간단한 형태의 임베딩 : 단어의 빈도를 기준으로 벡터로 변환하는 것

운수 좋은 날이라는 문서의 임베딩 : [2, 1, 1]

임베딩의 역할

1. 단어/ 문장 간 관련도 계산
2. 의미적/ 문법적 정보 함축
3. 전이 학습

임베딩의 종류와 성능

1. 행렬 분해
2. 예측 기반
3. 토픽 기반

단어 임베딩 (임베딩 함수)

임베딩 함수에 필요한 세가지 파라미터 : 입력, 출력, 단어 수

1. 몇 개의 단어 집합을 사용할 것인지 = 입력
2. 몇 개의 임베딩 결과를 사용할 것인지 = 출력
3. 매번 입력될 단어 수는 몇 개로 할지 = 단어 수

입력 : 총 몇개의 인덱스가 입력 되어야하는가

```
word_size = len(token.word_index) + 1
```

입력

출력 : 몇 개의 임베딩 결과를 사용할 것인지
단어 수 : 매번 입력될 단어의 수

```
model.add(Embedding(word_size, 8, input_length = 4))
```

출력

단어 수

실습 문제 (영화 리뷰 긍정 부정 예측)

```
import numpy
import tensorflow as tf
from numpy import array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Embedding
```

```
docs = ['너무 재밌네요', '최고예요', '참 잘 만든 영화예요', '추천하고 싶은 영화입니다', '한 번 더 보고싶네요',
        '글쎄요', '별로예요', '생각보다 지루하네요', '연기가 어색해요', '재미없어요']
```

```
classes = array([1,1,1,1,1,0,0,0,0,0])
```

#토큰화

```
token = Tokenizer()
token.fit_on_texts(docs)
print(token.word_index)
```

```
{'너무': 1, '재밌네요': 2, '최고예요': 3, '참': 4, '잘': 5, '만든': 6, '영화예요': 7, '추천하고': 8, '싶은': 9, '영화입니다': 10, '한': 11, '번': 12, '더': 13, '보고싶네요': 14, '글쎄요': 15, '별로예요': 16, '생각보다': 17, '지루하네요': 18, '연기가': 19, '어색해요': 20, '재미없어요': 21}
```

실습 문제 (영화 리뷰 긍정 부정 예측)

```
[ ] x = token.texts_to_sequences(docs)
    print(x)
```

```
[[1, 2], [3], [4, 5, 6, 7], [8, 9, 10], [11, 12, 13, 14], [15], [16], [17, 18], [19, 20], [21]]
```

```
[ ] #딤러닝 입력을 하려면 학습데이터의 길이가 같아야 하는데
    #이렇게 같게 맞춰주는 과정을 패딩이라고 함
```

```
padded_x = pad_sequences(x,4)
print(padded_x)
```

```
[[ 0  0  1  2]
 [ 0  0  0  3]
 [ 4  5  6  7]
 [ 0  8  9 10]
[11 12 13 14]
 [ 0  0  0 15]
 [ 0  0  0 16]
 [ 0  0 17 18]
 [ 0  0 19 20]
 [ 0  0  0 21]]
```

실습 문제 (영화 리뷰 긍정 부정 예측)

```
[ ] # 몇 개의 인덱스가 입력되는가
```

```
word_size = len(token.word_index) + 1
```

```
print(word_size)
```

22

```
[ ] # 몇 개의 임베딩 결과를 사용할 것인가 = 출력
```

```
model= Sequential()
```

```
# word_size 만큼의 입력값을 이용해서 8개의 임베딩 결과를 만듦
```

```
model.add(Embedding(word_size,8, input_length = 4))
```

```
model.add(Flatten())
```

```
model.add(Dense(1, activation='sigmoid'))
```

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

```
model.fit(padded_x, classes, epochs = 20)
```

```
print("\n Accuracy : %.4f" %(model.evaluate(padded_x, classes)[1]))
```

Epoch 10/20

1/1 [=====] - 0s 12ms/step - loss: 0.6400 - accuracy: 0.9000

Epoch 20/20

1/1 [=====] - 0s 12ms/step - loss: 0.6372 - accuracy: 0.9000

1/1 [=====] - 0s 149ms/step - loss: 0.6343 - accuracy: 0.9000

Accuracy : 0.9000

감사합니다
