Week10_예습과제_장서연

5장 - 토큰화

자연어 처리의 어려움

- 모호성
- 가변성
- 구조

→ 말뭉치를 일정한 단위인 토큰으로 만들어 위의 문제를 이해하고 구분 가능한 모델을 만듦.

말뭉치 : 뉴스 기사, 사용자리뷰, 저널 등에서 목적에 따라 구축되는 텍스트 데이터

토큰 : 말뭉치보다 더 작은 단위로, 개별 단어나 문장 부호와 같은 텍스트

토큰을 나누는 기준

- 공백 분할
- 정규 표현식 적용
- 어휘사전 적용
    - OOV(OUt of Vocab) : 없는 단어나 토큰 문제
    - 큰 어휘사전 : 차원의 저주, 학습 비용의 증대
    - 모든 토큰이나 단어를 벡터화 시 어휘사전에 등장하는 토큰 수만의 차원이 필요하고 희소데이터로 포현됨. 희소데이터는 출현빈도만 고려해 토큰들의 순서관계도 잘 표현 하지 못함.
- 머신러닝 활용

1. 단어 및 글자 토큰화

1) 단어 토큰화

- 띄어쓰기, 문장 부호, 대소문자 등의 특정 구분자를 활용해 토큰화 수행
- 품사 태깅, 개체명 인식, 기계 번역 등에 널리 사용
- 가장 일반적 / `split()` 으로 토큰화
- 한국어 접사, 문장부호, 오타 혹은 띄어쓰기 오류 등에 취약

예제 5.01 단어 토큰화

```
review = " 현실과 구분 불가능한 cg. 시각적 즐거움은 최고! 더불어 ost는 더더욱 최고!!"
tokenized = review.split()
print(tokenized)
```

```
['현실과', '구분', '불가능한', 'cg.', '시각적', '즐거움은', '최고!', '더불어', 'ost는', '더더욱', '최고!!']
```

2) 글자 토큰화

- 띄어쓰기 + 글자 단위로 문장 분할, 비교적 작은 단어사전 구축
- 언어 모델링과 같은 시퀀스 예측 작업에서 활용
- `list()` 로 토큰화 / 영어 : 알파벳, 한글 : 자음과 모음 `jamo` 라이브러리의 `h2j` 함

예제 5.02 글자 토큰화

```
review = "현실과 구분 불가능한 cg. 시각적 즐거움은 최고! 더불어 ost는 더더욱 최고!!"
tokenized = list(review)
print(tokenized)
```

```
['현', '실', '과', ' ', '구', '분', ' ', '불', '가', '능', '한', ' ', 'c', 'g', '.', ' ', '시', '각', '적', ' ', '즐', '거', '움', '은', ' ', '최', '고', '!', ' ', '더', '불', '어', ' ', 'o', 's', 't', '는'
```

- 컴퓨터가 한글을 인코딩하는 방식
  - 조합형 : 글자를 자모단위로 나눠 인코딩 후 조합. `j2hcj`
  - 완성형 : 조합된 글자 자체에 값을 불여해 인코딩. `h2j`

예제 5.03 자소단위 토큰화

```
!pip install jamo
```

```
Requirement already satisfied: jamo in /usr/local/lib/python3.12/dist-packages (0.4.1)
```

```
from jamo import h2j, j2hcj

review = "현실과 구분 불가능한 cg. 시각적 즐거움은 최고! 더불어 ost는 더더욱 최고!!"
decomposed = j2hcj(h2j(review))
tokenized = list(decomposed)
print(tokenized)
```

```
['ㅎ', 'ㅕ', 'ㄴ', 'ㅅ', 'ㅣ', 'ㄹ', 'ㄱ', 'ㅘ', ' ', 'ㄱ', 'ㅜ', 'ㅂ', 'ㅜ', 'ㄴ', ' ', 'ㅂ', 'ㅜ', 'ㄹ', 'ㄱ', 'ㅏ', 'ㄴ', 'ㅡ', 'ㅇ', 'ㅎ', 'ㅏ', 'ㄴ', ' ', 'c', 'g', '.', ' ', 'ㅅ', 'ㅣ', 'ㄱ', 'ㅏ', '
```

- 개별 토큰은 아무 의미가 없으므로 자연어 모델이 각 토큰의 의미를 조합해 결과도출해야함.
- 토큰 조합 : 문장 생성이나 개체명 인식 등을 구현할 때 다의어나 동음이의어가 많은 도메인에서 구별하는 것이 어려울 수 있음 + 입력 시퀀스의 길이가 길어질수록 연산량이 증가

2. 형태소 토큰화

- 텍스트를 형태소 단위로 토큰화
- 언어의 문법과 구조를 고려해 단어를 분리하고 이를 의미있는 단위로 분류
- 한국어의 특징

    - 단어가 띄어쓰기로 구분되지 않음
    - 어근 + 접사 + 조사 /
- 형태소
    - 자립 형태소 : 단어의 기본 / 명사, 동사, 형용사 등
    - 의존 형태소 : 자립형태소와 자합돼 문장에서 특정 역할 수행 / 조사, 어미, 접두사, 접미사

1) 형태소 어휘 사전

- 단어가 어떤 형태소들의 조합으로 이루어져있는지에 대한 정보
- 각 형태소가 어떤 품사에 속하는 지 + 해당 품사의 뜻 등의 정보도 함께 제공
- 품사 태깅(POS Tagging) : 텍스트 데이터를 형태소 분석해 각 형태소에 해당하는 품사 태깅
  → 자연어 처리에서 문맥 고려 가능, 더욱 정확한 분석 가능

2) KoNLPy

- JDK 기반
- 다양한 형태소 분석기 지원
- Okt
    - 명사추출(`okt.nouns`) , 구문추출(`okt.phrases`) , 형태소 추출(`okt.morphs`) , 품사태깅 (`okt.pos`) 지원

예제 5.04 Okt 토큰화

```
from konlpy.tag import Okt


okt = Okt()

sentence = "무엇이든 상상할 수 있는 사람은 무엇이든 만들어 낼 수 있다."

nouns = okt.nouns(sentence)
phrases = okt.phrases(sentence)
morphs = okt.morphs(sentence)
pos = okt.pos(sentence)

print("명사 추출 :", nouns)
print("구 추출 :", phrases)
print("형태소 추출 :", morphs)
print("품사 태깅 :", pos)
```

```
명사 추출 : ['무엇', '상상', '수', '사람', '무엇', '낼', '수']
구 추출 : ['무엇', '상상', '상상할 수', '상상할 수 있는 사람', '사람']
형태소 추출 : ['무엇', '이든', '상상', '할', '수', '있는', '사람', '은', '무엇', '이든', '만들어', '낼', '수', '있다', '.']
품사 태깅 : [('무엇', 'Noun'), ('이든', 'Josa'), ('상상', 'Noun'), ('할', 'Verb'), ('수', 'Noun'), ('있는', 'Adjective'), ('사람', 'Noun'), ('은', 'Josa'), ('무엇', 'Noun'), ('이든', 'Josa'), ('만들어', 'Ve
```

- 꼬꼬마
    - 구문 추출 지원 X, 명사추출(`kkma.sentences`) 지원

예제 5.05 꼬꼬마 토큰화

```
from konlpy.tag import Kkma


kkma = Kkma()

sentence = "무엇이든 상상할 수 있는 사람은 무엇이든 만들어 낼 수 있다."

nouns = kkma.nouns(sentence)
sentences = kkma.sentences(sentence)
morphs = kkma.morphs(sentence)
pos = kkma.pos(sentence)

print("명사 추출 :", nouns)
print("문장 추출 :", sentences)
print("형태소 추출 :", morphs)
print("품사 태깅 :", pos)
```

```
명사 추출 : ['무엇', '상상', '수', '사람', '무엇']
문장 추출 : ['무엇이든 상상할 수 있는 사람은 무엇이든 만들어 낼 수 있다.']
형태소 추출 : ['무엇', '이', '든', '상상', '하', 'ㄹ', '수', '있', '는', '사람', '은', '무엇', '이', '든', '만들', '어', '내', 'ㄹ', '수', '있', '다', '.']
품사 태깅 : [('무엇', 'NNG'), ('이', 'VCP'), ('든', 'ECE'), ('상상', 'NNG'), ('하', 'XSV'), ('ㄹ', 'ETD'), ('수', 'NNB'), ('있', 'VV'), ('는', 'ETD'), ('사람', 'NNG'), ('은', 'JX'), ('무엇', 'NP'), ('이', '
```

- 동일한 메서드라도 형태소 분석기의 특징에 따라 다른 결과
  → 시스템의 목적과 환경에 맞는 적절한 형태소 분석기 사용

3) NLTK

- 자연어 처리를 위해 개발된 라이브러리
- 토큰화, 형태소 분석, 구문 분석, 개체명 인식, 감성 분석 기능 제공
- 영어를 위해 개발 됐지만 네덜란드어, 프랑스어, 독일어 등과 같은 다양한 언어의 자연어 처리를 위한 데이터와 모델 제공

Punkt 모델

- Treebank라는 대규모의 영어 말뭉치 기반 학습, 통계기반 모델

예제 5.07 영문 토큰화

```
from nltk import tokenize


sentence = "Those who can imagine anything, can create the impossible."

word_tokens = tokenize.word_tokenize(sentence)
sent_tokens = tokenize.sent_tokenize(sentence)

print(word_tokens)
print(sent_tokens)
```

```
['Those', 'who', 'can', 'imagine', 'anything', ',', 'can', 'create', 'the', 'impossible', '.']
['Those who can imagine anything, can create the impossible.']
```

- `word_tokenize` : 단어 토크나이저
  - 문장을 입력받아 공백 기준으로 단어 분리
  - 구두점 처리
  - 각각의 단어/토큰을 추출해 리스트로 반환
- `sent_tokenize` : 문장 토크나이저
  - 문장을 입력받아 ,!?등의 구두점을 기준으로 문장 분리

- 리스트로 반환

Averaged Perceptron Tagger 모델

- Treebank라는 대규모의 영어 말뭉치 기반 학습, 퍼셉트론 기반 품사태깅

예제 5.08 영문 품사 태깅

```
from nltk import tag
from nltk import tokenize


sentence = "Those who can imagine anything, can create the impossible."

word_tokens = tokenize.word_tokenize(sentence)
pos = tag.pos_tag(word_tokens)

print(pos)
```

```
[('Those', 'DT'), ('who', 'WP'), ('can', 'MD'), ('imagine', 'VB'), ('anything', 'NN'), (',', ','), ('can', 'MD'), ('create', 'VB'), ('the', 'DT'), ('impossible', 'JJ'), ('.', '.')]
```

- `pos_tag` : 품사태깅 메서드
  - Averaged Perceptron Tagger 모델은 총 35개의 품사 태깅 가

4) spaCy

- 사이썬 기반 개발된 오픈 소스 라이브러리
- 빠른 속도와 높은 정확도를 목표로 하는 ML 기반 자연어 처리 라이브러
- GPU 가속, 24개 이상의 언어로 사전학습된 모델(`en_core_web_sm`) 제공

예제 5.09 spaCy 품사태깅

```
%pip install -q spacy
!python -m spacy download en_core_web_sm
```

```
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-any.whl (12.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 MB 99.7 MB/s eta 0:00:00
✓ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
⚠ Restart to reload dependencies
If you are in a Jupyter or Colab notebook, you may need to restart Python in
order to load all the package's dependencies. You can do this by selecting the
'Restart kernel' or 'Restart runtime' option.
```

```
import spacy


nlp = spacy.load("en_core_web_sm")
sentence = "Those who can imagine anything, can create the impossible."
doc = nlp(sentence)

for token in doc:
    print(f"[{token.pos_:5} - {token.tag_:3}] : {token.text}")
```

```
[PRON  - DT ] : Those
[PRON  - WP ] : who
[AUX   - MD ] : can
[VERB  - VB ] : imagine
[PRON  - NN ] : anything
[PUNCT - ,  ] : ,
```

- 객체 지향적으로 구현돼 처리한 결과를 `doc` 에 저장

- token 객체에 저장된 속성들
  - 기본 품사속성(`pos_`)
  - 세분화 품사 속성(`tag_`)
  - 원본 텍스트 데이터(`text`)
  - 토큰 사이 공백 포함 텍스트 데이터(`text_with_ws`)
  - 벡터(`vector`)
  - 벡터 노름(`vector_norm`)

3. 하위 단어 토큰화

- 하나의 단어가 빈번하게 사용되는 하위 단어의 조합으로 나누어 토큰화
  - ex) Rein + force + ment
- 새로운 단어나 표현/더이상 사용되지 않는 단어나 표현, 신조어/외래어/고유어, OOV 등의 문제 완화
- 바이트 페어 인코딩, 워드피스, 유니그램 모델 등

1) 바이트 페어 인코딩(Byte Pair Encoding, BPE)

- 다이그램 코딩
- 텍스트 데이터에서 가장 빈번하게 등장하는 글자쌍의 조합을 찾아 부호화하는 압축 알고리즘
- 초기에는 데이터 압축을 위해 개발, 자연어처리에서 하위단어 토큰화를 위한 방법을 ㅗ 사용
- 연속된 글자쌍이 더이상 나타나지 않거나 정해진 어휘사전에 도달할 때까지 조합탐지, 부호화 반복
  - 자주 등장하는 단어 : 하나의 토큰으로 토큰화
  - 덜 등장하는 단어 : 여러 토큰의 조합으로 표현
  - 예시) ab > A로 치환 / ra > B로 치환 / AB를 C로 치환
    → CcadC로 압축

센텐스피스

- 구글, 오픈소스 하위 단어 토크나이저
- 바이트 페어 인코딩과 유사한 알고리즘 사용해 입력 데이터 토큰화, 단어 사전 생성
- 다양한 알고리즘 지원 + 하이퍼파라미터 제공을 통한 세밀한 토크나이징 기능

코포라

- 국립 국어원/AI Hub에서 제공하는 말뭉치 데이터를 쉽게 쓸 수 있게 제공하는 오픈소스 라이브러리

예제 데이터셋 : 청와대 청원 데이터

```
from Korpora import Korpora


corpus = Korpora.load("korean_petitions")
dataset = corpus.train
petition = dataset[0]

print("청원 시작일 :", petition.begin)
print("청원 종료일 :", petition.end)
print("청원 동의 수 :", petition.num_agree)
print("청원 범주 :", petition.category)
print("청원 제목 :", petition.title)
print("청원 본문 :", petition.text[:30])
```

```
Korpora 는 다른 분들이 연구 목적으로 공유해주신 말뭉치들을
손쉽게 다운로드, 사용할 수 있는 기능만을 제공합니다.

말뭉치들을 공유해 주신 분들에게 감사드리며, 각 말뭉치 별 설명과 라이센스를 공유 드립니다.
해당 말뭉치에 대해 자세히 알고 싶으신 분은 아래의 description 을 참고,
해당 말뭉치를 연구/상용의 목적으로 이용하실 때에는 아래의 라이센스를 참고해 주시기 바랍니다.

# Description
Author : Hyunjoong Kim lovit@github
Repository : https://github.com/lovit/petitions_archive
References :

청와대 국민청원 게시판의 데이터를 월별로 수집한 것입니다.
청원은 게시판에 글을 올린 뒤, 하달 간 청원이 진행됩니다.
```

- `Korpora.load` : 말뭉치 불러오기, 데이터 로드
- `corpus` 에는 청원 데이터 저장, `.train` 으로 학습 데이터 가져올 수 있음. `petition` 에 정보 포함

예제 5.11 학습 데이터세트 생성

```
from Korpora import Korpora


corpus = Korpora.load("korean_petitions")
petitions = corpus.get_all_texts()
with open("/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/datasets/corpus.txt", "w", encoding="utf-8") as f:
    for petition in petitions:
        f.write(petition + "Wn")
```

```
Korpora 는 다른 분들이 연구 목적으로 공유해주신 말뭉치들을
손쉽게 다운로드, 사용할 수 있는 기능만을 제공합니다.

말뭉치들을 공유해 주신 분들에게 감사드리며, 각 말뭉치 별 설명과 라이센스를 공유 드립니다.
해당 말뭉치에 대해 자세히 알고 싶으신 분은 아래의 description 을 참고,
해당 말뭉치를 연구/상용의 목적으로 이용하실 때에는 아래의 라이센스를 참고해 주시기 바랍니다.

# Description
Author : Hyunjoong Kim lovit@github
Repository : https://github.com/lovit/petitions_archive
References :

청와대 국민청원 게시판의 데이터를 월별로 수집한 것입니다.
청원은 게시판에 글을 올린 뒤, 한달 간 청원이 진행됩니다.
수집되는 데이터는 청원종료가 된 이후의 데이터이며, 청원 내 댓글은 수집되지 않습니다.
단 청원의 동의 개수는 수집됩니다.
자세한 내용은 위의 repository를 참고하세요.

# License
CC0 1.0 Universal (CC0 1.0) Public Domain Dedication
Details in https://creativecommons.org/publicdomain/zero/1.0/
```

- `get_all_texts` : 본문 데이터셋 한번에 로드

예제 5.12 토크나이저 모델 학습

```
from sentencepiece import SentencePieceTrainer


SentencePieceTrainer.Train(
    "--input=/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/datasets/corpus.txtW
    --model_prefix=/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/models/petition_bpeW
    --vocab_size=8000 model_type=bpe"
)
```

- SentencePieceTrainer 클래스의 Train메서드로 토크나이저 모델 학습
- 센텐스피스는 문자열 입력을 통해 인자들을 전달 받음
- 주요 하이퍼파라미터

표 5.5 SentencePieceTrainer 매개변수

매개변수	의미
input	말뭉치 텍스트 파일의 경로
model_prefix	모델 파일 이름
vocab_size	어휘 사전 크기
character_coverage	말뭉치 내에 존재하는 글자 중 토크나이저가 다룰 수 있는 글자의 비율

model_type	토크나이저 알고리즘 ■ unigram ■ bpe ■ char ■ word
max_sentence_length	최대 문장 길이
unk_id	어휘 사전에 없는 OOV를 의미하는 unk 토큰의 id (기본값: 0)
bos_id	문장이 시작되는 지점을 의미하는 bos 토큰의 id (기본값: 1)
eos_id	문장이 끝나는 지점을 의미하는 eos 토큰의 id (기본값: 2)

예제 5.13 바이트페어 인코딩 토큰화

```
from sentencepiece import SentencePieceProcessor


tokenizer = SentencePieceProcessor()
tokenizer.load("/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/models/petition_bpe.model")

sentence = "안녕하세요, 토크나이저가 잘 학습되었군요!"
sentences = ["이렇게 입력값을 리스트로 받아서", "쉽게 토크나이저를 사용할 수 있답니다"]

tokenized_sentence = tokenizer.encode_as_pieces(sentence)
tokenized_sentences = tokenizer.encode_as_pieces(sentences)
print("단일 문장 토큰화 :", tokenized_sentence)
print("여러 문장 토큰화 :", tokenized_sentences)

encoded_sentence = tokenizer.encode_as_ids(sentence)
encoded_sentences = tokenizer.encode_as_ids(sentences)
print("단일 문장 정수 인코딩 :", encoded_sentence)
print("여러 문장 정수 인코딩 :", encoded_sentences)

decode_ids = tokenizer.decode_ids(encoded_sentences)
decode_pieces = tokenizer.decode_pieces(encoded_sentences)
print("정수 인코딩에서 문장 변환 :", decode_ids)
print("하위 단어 토큰에서 문장 변환 :", decode_pieces)
```

```
단일 문장 토큰화 : ['▁안녕하세요', ',', '▁토', '크', '나', '이', '저', '가', '▁잘', '▁학', '습', '되었', '군요', '!']
여러 문장 토큰화 : [['▁이렇게', '▁입', '력', '값을', '▁리', '스트', '로', '▁받아서'], ['▁쉽게', '▁토', '크', '나', '이', '저', '를', '▁사용할', '▁수', '▁있', '답니다']]
단일 문장 정수 인코딩 : [667, 6553, 994, 6880, 6544, 6513, 6590, 6523, 161, 110, 6554, 872, 787, 6648]
여러 문장 정수 인코딩 : [[372, 182, 6677, 4433, 1772, 1613, 6527, 4162], [1681, 994, 6880, 6544, 6513, 6590, 6536, 5852, 19, 5, 2639]]
정수 인코딩에서 문장 변환 : ['이렇게 입력값을 리스트로 받아서', '쉽게 토크나이저를 사용할 수 있답니다']
하위 단어 토큰에서 문장 변환 : ['이렇게 입력값을 리스트로 받아서', '쉽게 토크나이저를 사용할 수 있답니다']
```

- `tokenizer.load` : 모델 불러오기
- `encode_ad_pieces` 문장 토큰화
- `encode_as_ids` 토큰을 정수로 인코딩 / 토큰에 매핑된 ID값

예제 5.14 어휘 사전 불러오기

```
from sentencepiece import SentencePieceProcessor


tokenizer = SentencePieceProcessor()
tokenizer.load("/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/models/petition_bpe.model")

vocab = {idx: tokenizer.id_to_piece(idx) for idx in range(tokenizer.get_piece_size())}
print(list(vocab.items())[:5])
print("vocab size :", len(vocab))
```

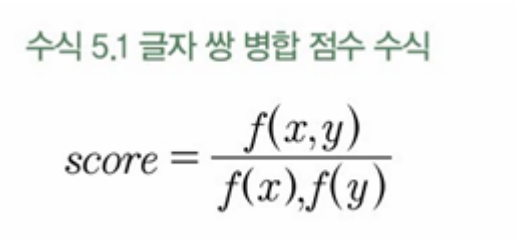
```
[(0, '<unk>'), (1, '<s>'), (2, '</s>'), (3, '니다'), (4, '▁이')]
vocab size : 8000
```

- 어휘 사전을 불러와 딕셔너리 형태로 매핑
- `get_piece_size` : 모델에서 생성된 하위 단어의 개수 반환

- `id_to_piece` 정숫값을 하위 단어로 변환
- → 하위 단어의 개수만큼 반복해 하위단어 딕혀너리 구성
  - <unk> : unknown, OOV
  - <s>, </s> : 문장의 시작지점과 종료 지점

2) 워드피스

- 빈도X, 확률 기반으로 글자쌍 병합
- 모델이 새로운 하위단어를 생성 시 이전 하위 단어와 함께 나타날 확률을 곗나해 가장 높은 확률을 가진 하위 단어 선택



- f : 빈도를 나타내는 함수
- x, y : 병합하려는 하위 단어
- $f(x, y)$: xy 글자쌍의 빈도
- $score$: xy 병합의 적절성 판단 점수

토크나이저스

- 허깅페이즈의 Tokenizers 라이브러리 사용
- 정규화, 사전 토큰화 제공

예제 5.15 워드피스 토크나이저 학습

```
from tokenizers import Tokenizer
from tokenizers.models import WordPiece
from tokenizers.normalizers import Sequence, NFD, Lowercase
from tokenizers.pre_tokenizers import Whitespace


tokenizer = Tokenizer(WordPiece())
tokenizer.normalizer = Sequence([NFD(), Lowercase()])
tokenizer.pre_tokenizer = Whitespace()

tokenizer.train(["/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/datasets/corpus.txt"])
tokenizer.save("/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/models/petition_wordpiece.json")
```

- 정규화 : 시퀀스 형식으로 인스턴스 전달
- NFD 유니코드 정규화, 소문자 변환 정규화
- 공백과 구두점을 기준으로 분리
- train으로 학습에 사용하려는 데이터세트 경로 전달
- save으로 학습결과 저장

예제 5.16 워드피스 토큰화

```
from tokenizers import Tokenizer
from tokenizers.decoders import WordPiece as WordPieceDecoder


tokenizer = Tokenizer.from_file("/content/drive/MyDrive/Euron_9thDL/pytorch_transformer/models/petition_wordpiece.json")
tokenizer.decoder = WordPieceDecoder()

sentence = "안녕하세요, 토크나이저가 잘 학습되었군요!"
sentences = ["이렇게 입력값을 리스트로 받아서", "쉽게 토크나이저를 사용할 수 있답니다"]

encoded_sentence = tokenizer.encode(sentence)
encoded_sentences = tokenizer.encode_batch(sentences)

print("인코더 형식 :", type(encoded_sentence))

print("단일 문장 토큰화 :", encoded_sentence.tokens)
print("여러 문장 토큰화 :", [enc.tokens for enc in encoded_sentences])

print("단일 문장 정수 인코딩 :", encoded_sentence.ids)
print("여러 문장 정수 인코딩 :", [enc.ids for enc in encoded_sentences])

print("정수 인코딩에서 문장 변환 :", tokenizer.decode(encoded_sentence.ids))
```

```
인코더 형식 : <class 'tokenizers.Encoding'>
단일 문장 토큰화 : ['안녕하세요', ',', '토', '##크', '##나이', '##저', '##가', '잘', '학습', '##되었', '##군요', '!']
여러 문장 토큰화 : [['이렇게', '입력', '##값을', '리스트', '##로', '받아서'], ['쉽게', '토', '##크', '##나이', '##저', '##를', '사용할', '수', '있다', '##니다']]
단일 문장 정수 인코딩 : [8760, 11, 8693, 8415, 16269, 7536, 7488, 7842, 15016, 8670, 8734, 0]
여러 문장 정수 인코딩 : [[8187, 19643, 13834, 28118, 7495, 12607], [9739, 8693, 8415, 16269, 7536, 7510, 14129, 7562, 8157, 7489]]
정수 인코딩에서 문장 변환 : 안녕하세요, 토크나이저가 잘 학습되었군요!
```

- 인코딩 메서드로 문장 토큰화, 인코딩 배치 메서드로 여러 문장 한번에 토큰화

6장 - 임베딩

텍스트 벡터화 : 텍스트 > 숫자

- 원핫 인코딩
- 빈도 벡터화

→ 둘다 벡터의 희소성이 크다는 단점이 있으므로 워드 임베딩 사용

워드 임베딩

- 단어를 고정된 길이의 실수 벡터로 표현
- 단어의 의미를 벡터 공간에서 다른 단어와의 상대적 위치로 표현해 단어간의 관계 추론
- 동적 임베딩 : 인공신경망 활용

1. 언어모델

- 입력된 문장으로 각 문장을 생성할 수 있는 확률 계산하는 모델
- 주어진 문장을 바탕으로 문맥 이해 + 문장 구성에 대한 예측 수행
- 음성인식, 자동번역, 텍스트 요약 등 다양한 자연어 처리 분야에서 활용

자기회귀 언어모델(Autoregressive Language Model)

- 입력된 문장들의 조건부 확률을 이용해 다음에 올 단어를 예측

수식 6.1 언어 모델의 조건부 확률

$$P(w_t|w_1,w_2,\cdots,w_{t-1}) = \frac{P(w_1,w_2,\cdots,w_t)}{P(w_1,w_2,\cdots,w_{t-1})}$$

수식 6.2 조건부 확률의 연쇄법칙을 적용한 언어 모델의 조건부 확률

$$P(w_t|w_1,w_2,\cdots,w_{t-1}) = P(w_1)P(w_2|w_1)\cdots P(w_t|w_1,w_2,\cdots,w_{t-1})$$

- 이전에 등장한 모든 토큰의 정보를 고려해 문맥정보를 파악 → 다음 단어 생성
- 이 과정이 반복
- 모델의 출력값이 입력값으로 사용되므로 자기회귀
- 시점별로 다음에 올 토큰 예측 → 토큰 분류 문제로 정의

통계적 언어모델(Statistical Language Model)

- 언어의 통계적 구조를 이용해 문장이나 단어의 시퀀스를 생성/분석
- 마르코프 체인을 이용해 구현
  - 빈도 기반의 조건부확률 모델
  - 이전 상태와 현재 상태간의 전이 확률을 이용해 다음 상태 예측

수식 6.3 빈도 기반의 조건부 확률

$$P(\text{만나서}|\text{안녕하세요}) = \frac{P(\text{안녕하세요 만나서})}{P(\text{안녕하세요})} = \frac{700}{1000}$$

$$P(\text{반갑습니다}|\text{안녕하세요}) = \frac{P(\text{안녕하세요 반갑습니다})}{P(\text{안녕하세요})} = \frac{100}{1000}$$

- 데이터 희소성 문제 : 관측한 적 없는 데이터를 예측하지 못하는 문제

- 기존 학습 텍스트 데이터에서 패턴 학습 → 확률 분포 생성 → 새로운 문장 생성 가능 + 다양한 종류의 텍스트 데이터 학습 가능
- 대규모 데이터 처리 효과적
- ex) GPT, BERT

GPT
- Raw Sentence: GPT는 OpenAI에서 개발한 인과적 언어 모델입니다.
- Input: GPT는 OpenAI에서 개발한
- Prediction-1: 인과적
- Prediction-2: 언어 모델입니다.

BERT
- Raw Sentence: BERT는 Google에서 개발한 마스킹된 언어 모델입니다.
- Input: Bert는 [MASK]에서 개발한 [MASK] 언어 모델입니다.
- Prediction: Google, 마스킹된

2. N-gram

- N개의 연속된 단어 시퀀스를 하나의 단위로 취급해 특정 단어 시퀀스가 등장할 확률 추정
- N=1 유니그램, 2 바이그램, 3 트라이그램, 4 이상 N-gram
- N개의 토큰을 묶어서 분석

수식 6.4 N-gram에서의 조건부 확률

$$P(w_t | w_{t-1}, w_{t-2}, \cdots, w_{t-N+1})$$

예제 6.01 N-gram

```python
import nltk


def ngrams(sentence, n):
    words = sentence.split()
    ngrams = zip(*[words[i:] for i in range(n)])
    return list(ngrams)

sentence = "안녕하세요 만나서 진심으로 반가워요"

unigram = ngrams(sentence, 1)
bigram = ngrams(sentence, 2)
trigram = ngrams(sentence, 3)

print(unigram)
print(bigram)
print(trigram)

unigram = nltk.ngrams(sentence.split(), 1)
bigram = nltk.ngrams(sentence.split(), 2)
trigram = nltk.ngrams(sentence.split(), 3)

print(list(unigram))
print(list(bigram))
print(list(trigram))
```

3. TF-IDF

- 텍스트 문서에서 특정 단어 중요도 계산
- BoW : 문서나 문장을 단어의 집합으로 표현 / 문서나 문장에 등장하는 단어의 중복을 허용해 빈도를 기록
    - 모든 단어는 동일한 가중치

표 6.2 BoW 벡터화

	I	like	this	movie	don't	famous	is
This movie is famous movie	0	0	1	2	0	1	1
I like this movie	1	1	1	1	0	0	0
I don't like this movie	1	1	1	1	1	0	0

단어 빈도

- TF : 문서 내 특정 단어의 빈도수
- 전문 용어, 관용어, 문서길이 길수록 증가

문서 빈도

- DF : 문서 집합에서 단어가 나타나는 횟수/ 단어가 몇개의 문서에서 등장하는 지 계산
- DF가 높으면 널리사용되는, 중요도가 낮은 단어로 간주 가능
- 낮으면 특정한 문맥에서만 사용되는, 중요도가 높은 단어로 간주 가능

역문서 빈도

수식 6.7 역문서 빈도

$$IDF(t,D) = \log\left(\frac{count(D)}{1+DF(t,D)}\right)$$

- IDF
- 단어 빈도 ⬇️ IDF ⬆️

TF-IDF

수식 6.8 TF-IDF

$$TF-IDF(t,d,D) = TF(t,d) \times IDF(t,d)$$

- 특정 문서내 단어가 자주 등장하지만 전체 문서 내에서는 적게 등장하면 TFIDF값 커짐
- 실습 내용은 ipynb 파일

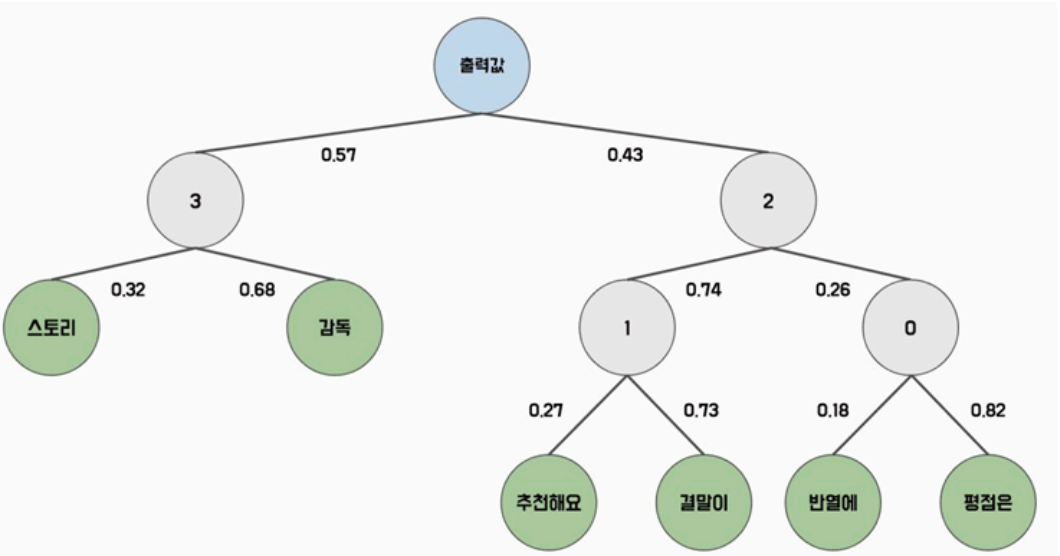


그림 6.8 계층적 소프트맥스 구조

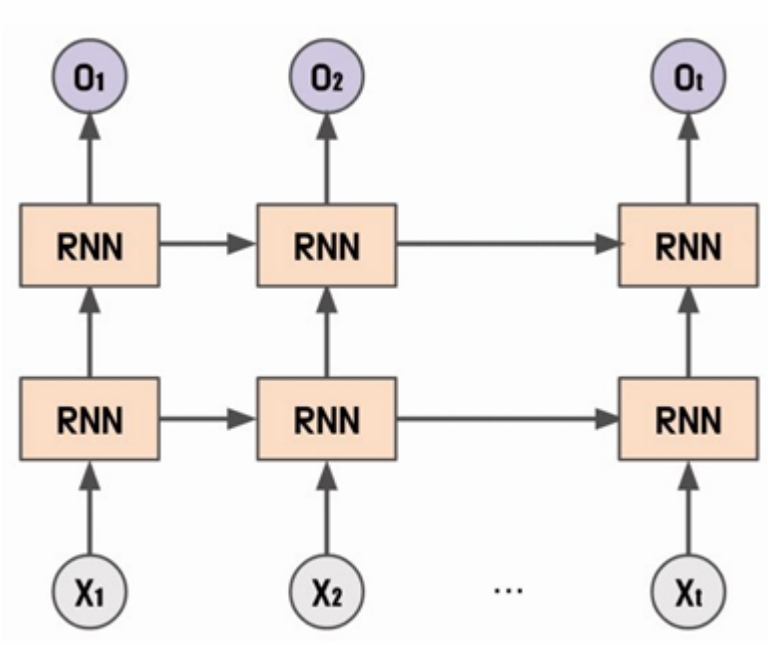


그림 6.17 다중 순환 신경망 구조