5.1 토큰화와 임베딩

INDEX

토큰화(Tokenization)와 임베딩(Embedding)

서울 18반 이재윤 강사

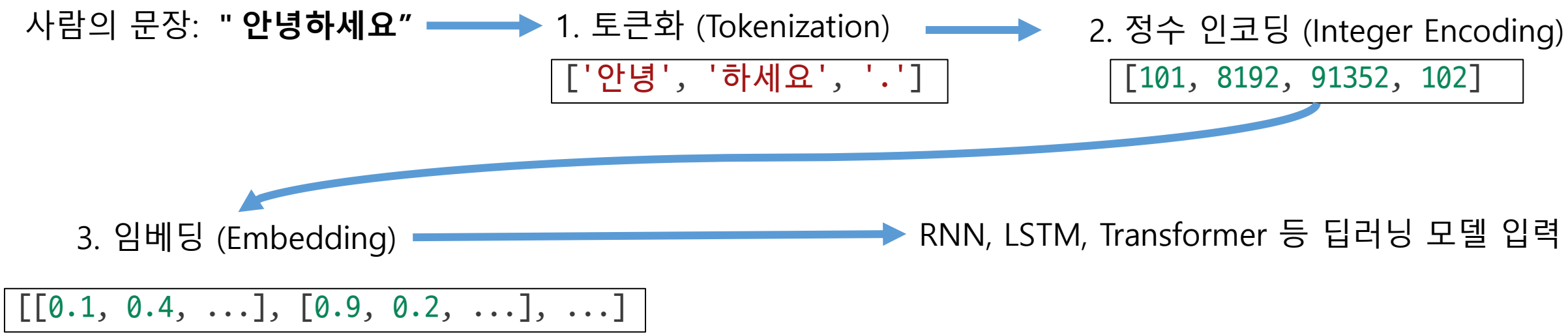
학습 목표

- **토큰화(Tokenization)**의 개념을 이해하고, 문장을 토큰으로 분리할 수 있음
- **임베딩(Embedding)**의 필요성을 이해하고, 단어를 벡터로 표현하는 이유를 설명할 수 있음
- **허깅페이스(Hugging Face)** 라이브러리를 사용하여 실제 토큰화와 임베딩을 실습할 수 있음

토큰화(Tokenization)와 임베딩(Embedding)

왜 지금 토큰화와 임베딩을 배워야 하는가?

- **NLP 모델의 학습 과정 (큰 그림)**



토크나이저 (Tokenizer)란

토크나이저(Tokenizer)란

토크나이저란?

- 문장을 **토큰(Token)**이라는 의미 있는 단위로 나누고, 각 토큰을 **정수(ID)**로 변환하는 도구.
  - **예시**:
    - 문장: "안녕하세요"
    - 토큰화: ["안녕", "하세요"]
    - 정수 인코딩: [8192, 91352]

컴퓨터가 글자를 **이해**하기 위한 첫 번째 번역 작업

토큰화 실습

실습: 허깅 페이스(Hugging Face) 를 활용한 토큰화

- 허깅페이스(Hugging Face)란?
  - 최신 NLP 딥러닝 기술을 손쉽게 사용할 수 있도록 도와주는 플랫폼이자 라이브러리

- 주요 기능
- Models: 사전 학습된 수많은 NLP 모델 공유 (오늘 쓸 `klue/bert-base`도 그 중 하나)
- Datasets: 모델 학습 및 평가를 위한 데이터셋 공유
- Transformers: 모델, 토크나이저 등을 쉽게 로드하고 사용할 수 있는 파이썬 라이브러리

개발자들이 '바퀴를 재발명'하지 않도록 도와주는 고마운 생태계

실습: 허깅 페이스(Hugging Face) 를 활용한 토큰화

1. 필요한 라이브러리 설치 및 로딩

```
# !pip install transformers
from transformers import AutoTokenizer
```

- **AutoTokenizer란?**
  - **허깅페이스(Hugging Face)의 transformers 라이브러리가 제공하는 클래스**
  - **모델 이름("klue/bert-base")만 알려주면, 해당 모델에 맞는 토크나이저를 자동으로 찾아주는 편리한 기능을 제공함**

실습: 허깅 페이스(Hugging Face) 를 활용한 토큰화

2. 사전 학습된 토크나이저 가져오기

```
# 한국어 BERT 모델의 토크나이저를 로드
tokenizer = AutoTokenizer.from_pretrained("klue/bert-base")
print(tokenizer)
```

- klue/bert-base: 한국어 처리를 위해 잘 학습된 모델
- 우리는 이 모델이 가진 단어사전과 토큰화 규칙을 그대로 사용할 것

실습: 허깅 페이스(Hugging Face) 를 활용한 토큰화

3. 분석할 문장의 정의

```
text = "안녕하세요. 이 실습은 허깅페이스 토크나이저 사용법을 익히는 좋은 예제입니다."
```

4. 토크나이저로 문장 인코딩

```
encoded_input = tokenizer(text)
print(encoded_input)
```

5. 실행 결과 확인 input_ids: 각 토큰에 해당하는 정수 ID 리스트 (가장 중요!)

```
{'input_ids': [2, 5891, 2205, 5971, 18,.....],
'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0...],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1.....
]}
```

실습: 허깅 페이스(Hugging Face) 를 활용한 토큰화

6. 인코딩 결과 확인하기

- input_ids를 다시 토큰으로 변환

```
tokens = tokenizer.convert_ids_to_tokens(encoded_input['input_ids'])
print(tokens)
```

```
['[CLS]', '안녕', '##하', '##세요', '.', '이', '실습', '##은', '허', '##깅', '##페이스', '토크',
'##나이', '##저', '사용법', '##을', '익히', '##는', '좋', '##은', '예', '##제', '##입니다', '.',
'[SEP]']
```

•[CLS], [SEP]: 문장의 시작과 끝을 알리는 스페셜 토큰

•##하세요, ##은: 서브워드(Subword). 단어가 더 작은 단위로 나뉜 것.

- **"모르는 단어" 문제(OOV)를 해결하는데 효과적임.**

임베딩(Embedding)

토크나이저(Tokenizer)란

임베딩(Embedding)이란?

- **정의**: 토큰에 부여된 **정수 ID**를, 의미를 함축한 고차원의 벡터(Vector)로 변환하는 과정
- **정수 ID**: 단순한 숫자. 단어 간의 관계 표현 불가
  - 8192(안녕)와 8193(반가워)는 숫자로는 가깝지만, 의미는 다름
- **임베딩 벡터**: 단어의 의미, 문맥, 관계성을 좌표값으로 표현
  - 의미가 비슷한 단어들은 벡터 공간에서 가까운 위치에 존재함

토크나이저(Tokenizer)란

임베딩(Embedding)이란?

- **임베딩 벡터**는 단어의 의미적 관계를 **벡터 연산**으로 표현 가능

벡터("왕") - 벡터("남자") + 벡터("여자") ≈ 벡터("여왕")

벡터("서울") - 벡터("대한민국") + 벡터("일본") ≈ 벡터("도쿄")

- 임베딩은 모델이 단어의 **유사도**와 **패턴**을 학습할 수 있게 함
- 이 벡터가 RNN, LSTM의 실제 입력값이 됨

실습: 임베딩 벡터 추출하기

1. 필요한 라이브러리 로딩

```
# PyTorch 버전의 모델을 로드
from transformers import AutoModel
```

2. 사전 학습된 모델 가져오기

```
# 토크나이저와 동일한 이름의 모델을 로드해야 짝이 맞음
model = AutoModel.from_pretrained("klue/bert-base")
print(model)
```

토크나이저(Tokenizer)란

실습: 임베딩 벡터 추출하기

3. 모델 입력 형식에 맞게 텍스트 인코딩

```
# return_tensors='pt' : 결과를 PyTorch 텐서 형태로 반환
encoded_input = tokenizer(text, return_tensors='pt')
```

4. 인코딩된 입력을 모델에 전달

```
# **를 사용하여 딕셔너리의 각 항목을 모델의 인자로 전달
output = model(**encoded_input)
```

5. 결과 확인 (마지막 은닉 상태 == 임베딩 벡터)

```
# output의 여러 결과 중 last_hidden_state가 최종 임베딩 벡터
embedding_vector = output.last_hidden_state
print(embedding_vector.shape)
```

```
torch.Size([1, 26, 768])
```

실습: 임베딩 벡터 추출하기

6. 임베딩 결과 Shape 분석

```
torch.Size([1, 26, 768])
```

- (1, 26, 768) - Batch Size
    - 한 번에 처리한 문장의 개수. (우리는 1개 문장을 넣었음)
- (1, 26, 768) - Sequence Length
- 문장을 토큰화 했을 때의 토큰 개수
    - [CLS], ..., [SEP] 까지 총 26개의 토큰
    - (1, 26, 768) - Hidden Size / Embedding Dimension
    - 하나의 토큰을 표현하는데 사용된 숫자의 개수 (벡터의 차원)
    - 즉, 26개의 모든 토큰이 각각 768개의 숫자로 이루어진 벡터로 변환되었음

다음 시간에
만나요!