10장 | 자연어 처리를 위한 임베딩

WEEK	15주차

10.1 임베딩

임베딩(embedding)은 사람이 사용하는 언어(자연어)를 컴퓨터가 이해할 수 있는 언어(숫자) 형태인 벡터(vector)로 변환한 결과 혹은 일련의 과정을 의미함

임베딩 역할

- 단어 및 문장 간 관련성 계산
- 의미적 혹은 문법적 정보의 함축(e.g. 왕-여왕, 교사-학생)

임베딩 방법 : 희소 표현 기반 임베딩, 횟수 기반 임베딩, 예측 기반 임베딩, 횟수/예측 기반 임베딩

10.1.1 희소 표현 기반 임베딩

희소 표현(sparse representation)은 대부분의 값이 0으로 채워져 있는 경우로, 대표적으로 **원-핫 인코딩**이 있음

원-핫 인코딩(one-hot encoding)

주어진 텍스트를 숫자(벡터)로 변환해 주는 것
단어 N개를 각각 N차원의 벡터로 표현하는 방식으로, 단어가 포함되어 있는 위치에 1을 넣고 나머지에는 0 값을 채움

원-핫 인코딩의 단점

1. 수학적 의미에서 원-핫 벡터들은 하나의 요소만 1 값을 갖고 나머지는 모두 0인 희소 벡터(sparse vector)를 가짐. 이때 두 단어에 대한 벡터의 내적을 구해 보면 0 값을 갖게 되므로 직교(orthogonal)를 이룸. 즉, 단어끼리의 관계성(유의어, 반의어) 없이 서로 독립적인 관계가 됨
2. '차원의 저주(curse of dimensionality)' 문제 발생 : 하나의 단어를 표현하는 데 말뭉치(corpus)에 있는 수만큼 차원이 존재하기 때문에 복잡해짐

원-핫 인코딩에 대한 대안으로, 신경망에 기반하여 단어를 벡터로 바꾸는 방법론들 주목 (e.g. 워드투벡터(Word2Vec), 글로브(GloVe), 패스트텍스트(FastText) 등)

10.1.2 횟수 기반 임베딩

횟수 기반은 단어가 출현한 빈도를 고려하여 임베딩하는 방법으로, 대표적으로 카운터 벡터와 TF-IDF가 있음

카운터벡터(counter vector)

문서 집합에서 단어를 토큰으로 생성하고 각 단어의 출현 빈도수를 이용하여 인코딩해서 벡터를 만드는 방법
즉, 토크나이징과 벡터화가 동시에 가능한 방법

사이킷런의 `CountVectorizer()`를 사용하여 코드로 구현 가능

1. 문서를 토큰 리스트로 변환
2. 각 문서에서 토큰의 출현 빈도를 셈
3. 각 문서를 인코딩하고 벡터로 변환

TF-IDF(Term Frequency-Inverse Document Frequency)

정보 검색론(Information Retrieval, IR)에서 가중치를 구할 때 사용되는 알고리즘

TF(Term Frequency)(단어 빈도)는 문제 내에서 특정 단어가 출현한 빈도를 의미함

$tf_{t,d}$는 특정 문서 d에서 특정 단어 t의 등장 횟수를 의미함

$$tf_{t,d} = \begin{cases} 1 + \log count(t,d) & count(t,d) > 0\text{일 때} \\ 0 & \text{그 외} \end{cases}$$

$$tf_{t,d} = \log(count(t,d) + 1)$$

(t(term): 단어, d(document): 문서 한 개)

IDF(Inverse Document Frequency)(역문서 빈도)를 이해하려면 **DF(Document Frequency)(문서 빈도)**에 대한 개념부터 이해해야 함
DF는 한 단어가 전체 문서에서 얼마나 공통적으로 많이 등장하는지 나타내는 값으로, 특정 단어가 나타난 문서 개수로 이해하면 됨

$$df_t = \text{특정 단어 } t\text{가 포함된 문서 개수}$$

특정 단어 t가 모든 문서에 등장하는 **일반적인 단어**(e.g. a, the)라면, TF-IDF 가중치를 낮추어 줄 필요가 있음
따라서 DF 값이 클수록 TF-IDF의 가중치 값을 낮추기 위해 DF 값에 역수를 취하는데, 이 값이 IDF임
역수를 취하면 전체 문서 개수가 많아질수록 IDF 값도 커지므로 IDF는 로그(log)를 취해야 함

$$idf_t = \log(\frac{N}{df_t}) = \log(\frac{\text{전체 문서 개수}}{\text{특정 단어 } t\text{가 포함된 문서 개수}})$$

△ 전체 문서에 특정 단어가 발생하는 빈도가 0이라면 분모가 0이 되는 상황이 발생
이를 방지하고자 다음과 같이 분모에 1을 더해 주는 것을
스무딩(smoothing)이라고 함

$$idf_t = \log(\frac{N}{1+df_t}) = \log(\frac{\text{전체 문서 개수}}{1+\text{특정 단어 } t\text{가 포함된 문서 개수}})$$

TF-IDF 사용되는 상황

- 키워드 검색을 기반으로 하는 검색 엔진
- 중요 키워드 분석
- 검색 엔진에서 검색 결과의 순위 결정

TF-IDF 값은 특정 문서 내에서 단어의 출현 빈도가 높거나 전체 문서에서 특정 단어가 포함된 문서가 적을수록 TF-IDF 값이 높음
따라서 이 값을 사용하여 문서에 나타나는 흔한 단어(e.g. a, the)들을 걸러 내거나 특정 단어에 대한 중요도를 찾을 수 있음

10.1.3 예측 기반 임베딩

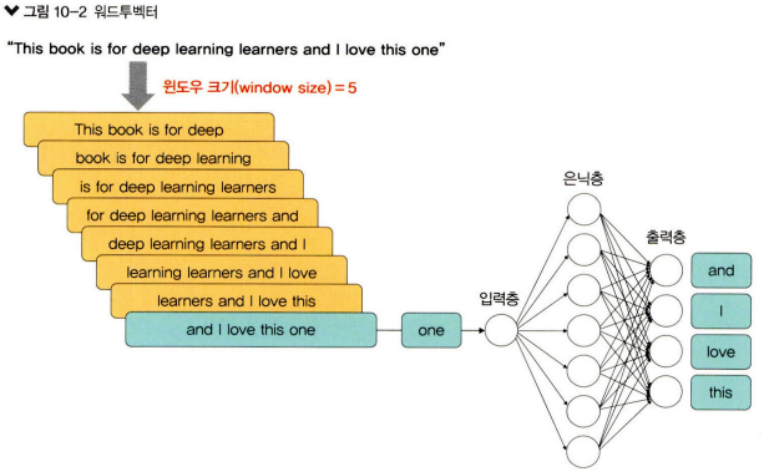
예측 기반 임베딩은 신경망 구조 혹은 모델을 이용하여 특정 문맥에서 어떤 단어가 나올지 예측하면서 단어를 벡터로 만드는 방식으로, 대표적으로 워드투벡터가 있음

워드투벡터(Word2Vec)

신경망 알고리즘으로, 주어진 텍스트에서 텍스트의 각 단어마다 하나씩 일련의 벡터를 출력함

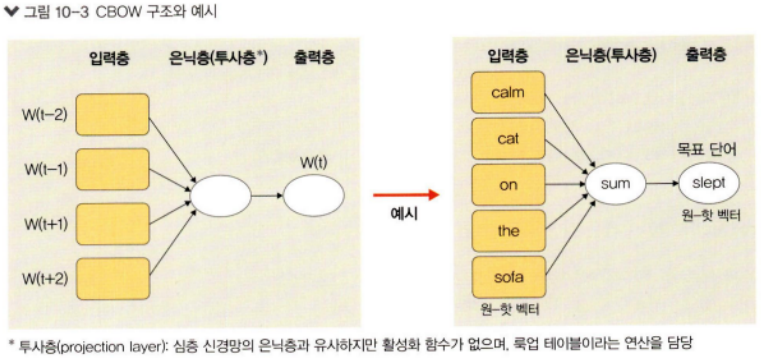
워드투벡터의 출력 벡터가 2차원 그래프에 표시될 때, 의미론적으로 유사한 단어의 벡터는 서로 가깝게 표현됨
'서로 가깝다'는 의미는 코사인 유사도를 이용하여 단어 간의 거리를 측정한 결과로 나타나는 관계성을 의미함
즉, 워드투벡터를 이용하면 특정 단어의 동의어를 찾을 수 있음

워드투벡터 수행 과정 : 일정한 크기의 윈도우(window)로 분할된 텍스트를 신경망 입력으로 사용함. 이때 모든 분할된 텍스트는 한 쌍의 대상 단어와 컨텍스트로 네트워크에 공급됨. 또한, 네트워크의 은닉층에는 각 단어에 대한 가중치가 포함되어 있음



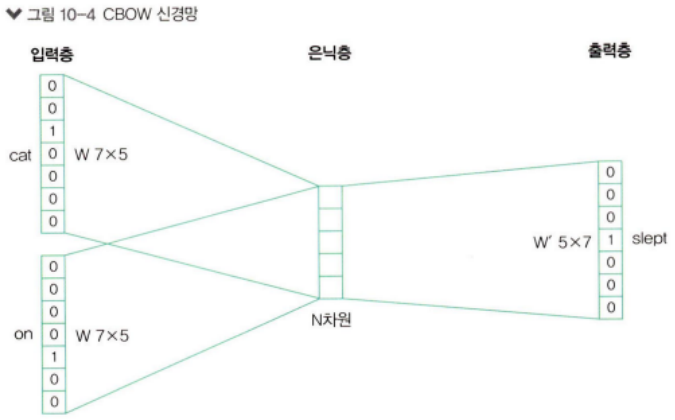
CBOW(Continuous Bag Of Words)

단어를 여러 개 나열한 후 이와 관련된 단어를 추정하는 방식
즉, 문장에서 등장하는
n개의 단어 열에서 다음에 등장할 단어를 예측함



CBOW는 위와 같은 신경망 구조를 가짐
여기에서 각 문맥 단어를 은닉층으로 투사하는 가중치 행렬은 모든 단어에서 공통으로 사용됨

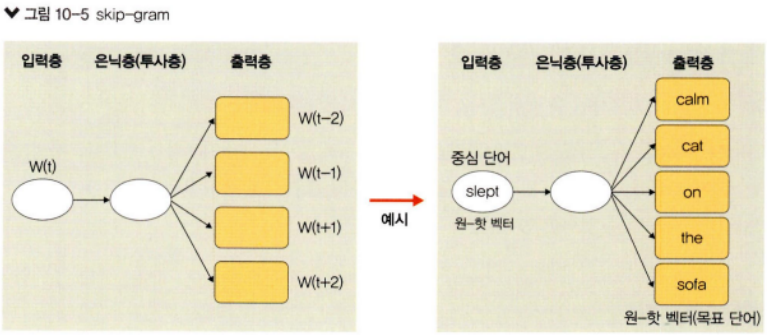
CBOW 신경망에서 크기가 N인 은닉층을 가지고 있을 때, 은닉층 크기 N은 입력 텍스트를 임베딩한 벡터 크기
다음으로 입력층과 은닉층 사이의 가중치
W는 VxN 행렬이며, 은닉층에서 출력층 사이의 가중치 W'는 NxV 행렬이고, V는 단어 집합의 크기를 의미



skip-gram

CBOW 방식과 반대로 특정한 단어에서 문맥이 될 수 있는 단어를 예측함
즉, skip-gram은 중심 단어에서 주변 단어를 예측하는 방식을 사용

보통 입력 단어 주변의 단어 k개를 문맥으로 보고 예측 모형을 만드는데, 이 k 값을 윈도우 크기라고 함



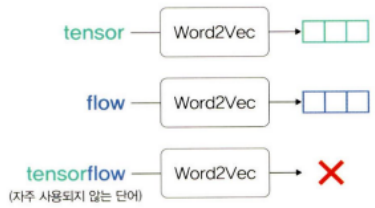
CBOW와 skip-gram 중 어떤 알고리즘이 더 좋다고 결론을 내리기보다는 분석하고자 하는 데이터 성격, 분석에 대한 접근 방법 및 도출하고자 하는 결론 등을 종합적으로 고려하여 필요한 라이브러리를 사용할 수 있어야 함

패스트텍스트(FastText)

워드투벡터의 단점을 보완하고자 페이스북에서 개발한 임베딩 알고리즘

기존 워드투벡터의 워드 임베딩 방식은 분산 표현(distributed representation)을 이용하여 단어의 분산 분포가 유사한 단어들에 비슷한 벡터 값을 할당하여 표현함
→ 사전에 없는 단어에 대해서는 벡터 값을 얻을 수 없음
→ 자주 사용되지 않는 단어에 대해서는 학습이 불안정함

▼ 그림 10-6 워드투벡터 단점



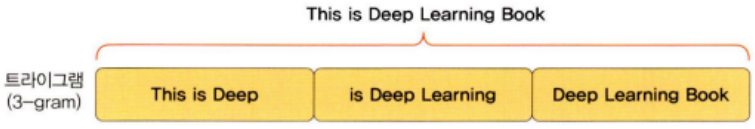
패스트텍스트는 이러한 단점들을 보완하려고 개발된 단어 표현(word representation) 방법을 사용함

패스트텍스트는 노이즈에 강하며, 새로운 단어에 대해서는 형태적 유사성을 고려한 벡터값을 얻기 때문에 자연어 처리 분야에서 많이 사용되는 알고리즘

사전에 없는 단어에 벡터 값을 부여하는 방법

패스트텍스트는 주어진 문서의 각 단어의 n-그램(n-gram)으로 표현함
이때 n의 설정에 따라 단어의 분리 수준이 결정됨

▼ 그림 10-7 트라이그램



n값에 따른 단어의 분리(부분단어(subword))는 다음 표와 같음

▼ 표 10-1 n 값에 따른 단어의 분리

문장	n 값	단어의 분리
This is Deep Learning Book	1	〈This, is, Deep, Learning, Book〉
	2	〈This is, is Deep, Deep Learning, Learning Book〉
	3	〈This is Deep, is Deep Learning, Deep Learning Book〉

패스트텍스트는 인공 신경망을 이용하여 학습이 완료된 후 데이터셋의 모든 단어를 각 n-그램에 대해 임베딩함
따라서 사전에 없는 단어가 등장한다면 n-그램으로 분리된 부분 단어와 유사도를 계산하여 의미를 유추할 수 있음

Note | n-그램

n-그램(n-gram)은 n개의 어절/음절을 연쇄적으로 분류하여 그 빈도를 따짐
n이 1일 때를 유니그램(unigram)이라 하고, n이 2일 때를 바이그램(bigram)이라 하고, n이 3일 때를 트라이그램(trigram)이라고 함

자주 사용되지 않는 단어에 학습 안정성을 확보하는 방법

워드투벡터는 단어의 출현 빈도가 적으면 임베딩의 정확도가 낮은 단점

△ 패스트텍스트는 등장 빈도수가 적더라도, n-그램으로 임베딩하기 때문에 참고할 수 있는 경우의 수가 많음 → 상대적으로 자주 사용되지 않는 단어에서도 정확도가 높음

10.1.4 횟수/예측 기반 임베딩

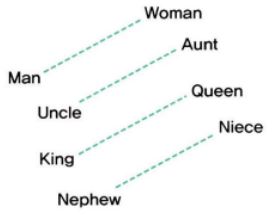
앞서 살펴본 횟수 기반과 예측 기반의 단점을 보완하기 위한 임베딩 기법에는 대표적으로 글로브가 있음

글로브(GloVe, Global Vectors for Word Representation)

글로브는 횟수 기반의 LSA(Latent Semantic Analysis)(잠재 의미 분석)와 예측 기반의 워드투벡터 단점을 보완하기 위한 모델

이름에서 예측할 수 있듯이 단어에 대한 글로벌 동시 발생 확률(global co-occurence statistics) 정보를 포함하는 단어 임베딩 방법임
즉, 단어에 대한 통계 정보와 skip-gram을 합친 방식이라고 할 수 있음

❤ 그림 10-8 글로브를 이용한 단어 간 관련성 예시

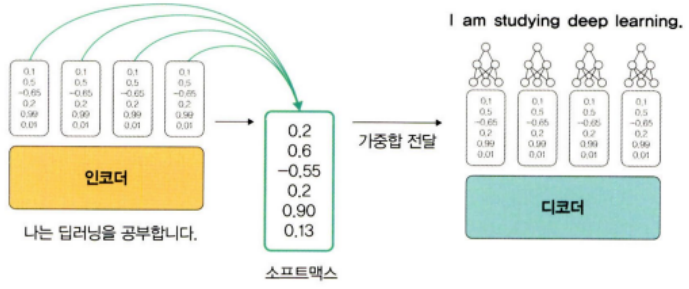


10.2 트랜스포머 어텐션

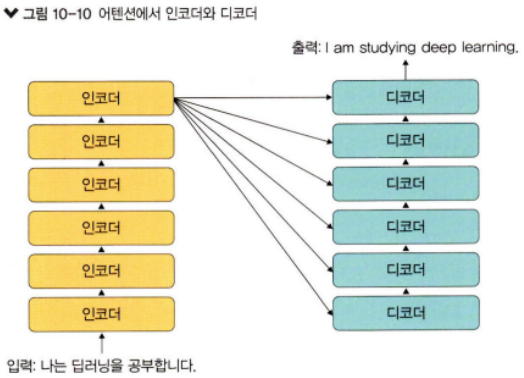
어텐션(attention)은 주로 언어 번역에서 사용되기 때문에 인코더와 디코더 네트워크를 사용함
즉, 입력에 대한 벡터 변환을 인코더(encoder)에서 처리하고 모든 벡터를 디코더로 보냄
모든 벡터를 전달하는 이유 : 시간이 흐를수록 초기 정보를 잃어버리는 기울기 소멸 문제를 해결하기 위해서
△ 모든 벡터가 전달되기 때문에 행렬 크기가 굉장히 커지는 단점
→ 해결하기 위해 소프트맥스 함수를 사용하여
가중합을 구하고 그 값을 디코더에 전달

가중합만 전달되어도 정보 多 → 디코더 부담
⇒ 디코더는 은닉 상태에 대해 중점적으로 집중해서 보아야 할 벡터를 소프트맥스 함수로 점수를 매긴 후 각각을 은닉 상태의 벡터들과 곱함. 그리고 이 은닉 상태를 모두 더해서 하나의 값으로 만듦

❤ 그림 10-9 어텐션



트랜스포머(transformer)는 어텐션을 극대화하는 방법
어텐션에서 다룬 인코더와 디코더에는 네트워크가 하나씩 있었음 △ 트랜스포머는 인코더와 디코더를 여러 개 중첩시킨 구조
이때 각각의 인코더와 디코더를 블록(block)이라고 함

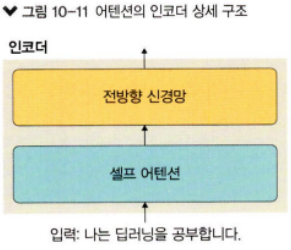


인코더 블록 구조

하나의 인코더는 셀프 어텐션(self-attention)과 전방향 신경망(feed forward neural network)으로 구성

인코더에서는 단어를 벡터로 임베딩하며, 이를 셀프 어텐션과 전방향 신경망으로 전달함

이때 셀프 어텐션은 문장에서 각 단어끼리 얼마나 관계가 있는지를 계산해서 반영함
즉, 셀프 어텐션으로 문장 안에서 단어 간 관계를 파악할 수 있음
셀프 어텐션에서 파악된 단어 간 관계는 전방향 신경망으로 전달됨

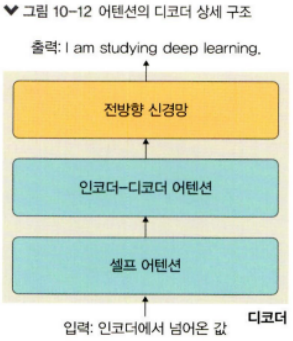


디코더 블록 구조

디코더는 총 세 개 층 가짐

인코더에서 넘어온 벡터가 처음으로 만나는 것이 셀프 어텐션 층
즉, 인코더와 동일하다고 이해하면 됨

셀프 어텐션 층을 지나면 인코더-디코더 어텐션(encoder-decoder attention) 층이 있음
인코더가 처리한 정보를 받아 어텐션 메커니즘 수행하고, 마지막으로 전방향 신경망으로 데이터가 전달됨



어텐션 메커니즘 이해 - 수식을 통해

어텐션 스코어 : 현재 디코더의 시점 i에서 단어를 예측하기 위해, 인코더의 모든 은닉 상태 값(h_j)이 디코더의 현 시점의 은닉 상태(s_i)와 얼마나 관련이 있는지(유사한지)를 판단하는 값

인코더의 모든 은닉 상태의 값(
h_j)과 디코더에서의 이전 시점의 은닉 상태(s_{i-1})값을 이용하여 구할 수 있음

$$e_{ij} = a(s_{i-1}, h_j)$$

어텐션 스코어 값을 소프트맥스 함수에 적용하여 확률로 변환
계산된 0~1 사이의 값들이 특정 시점(time step)에 대한 가중치, 즉 시간의 가중치가 되며 다음과 같은 수식을 이용함

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

시간의 가중치(a_{ij})와 은닉 상태(h_j)의 가중합을 계산하면 하나의 벡터가 계산되는데, 이것이 컨텍스트 벡터(context vector)임

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j$$

마지막으로 디코더의 은닉 상태를 구해야 함

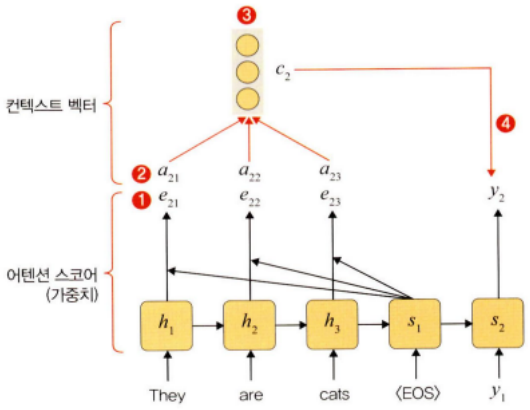
디코더의 은닉 상태를 구하기 위해서는 컨텍스트 벡터와 디코더 이전 시점의 은닉 상태와 출력이 필요함

다음과 같이 어텐션이 적용된 인코더-디코더의 수식에서는 컨텍스트 벡터(c_i)가 계속 변하고 있는 반면에 어텐션이 적용되지 않은 인코더-디코더의 수식에서는 컨텍스트 벡터(c)가 고정되어 있음

(어텐션 적용) 이전 시점의 은닉 상태 값을 구하는 수식: $s_i = f(s_{i-1}, y_{i-1}, c_i)$

(어텐션 미적용) 이전 시점의 은닉 상태 값을 구하는 수식: $s_i = f(s_{i-1}, y_{i-1}, c)$

그림 10-13 어텐션 메커니즘 예시

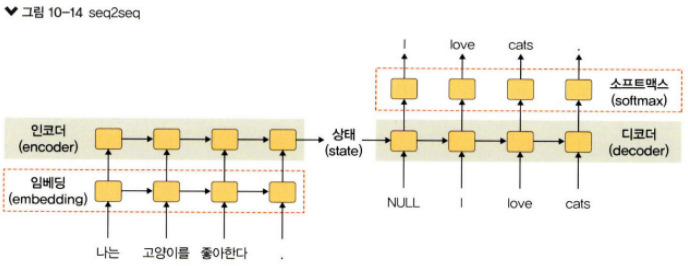


10.2.1 seq2seq

seq2seq(sequence to sequence)는 입력 시퀀스에 대한 출력 시퀀스를 만들기 위한 모델

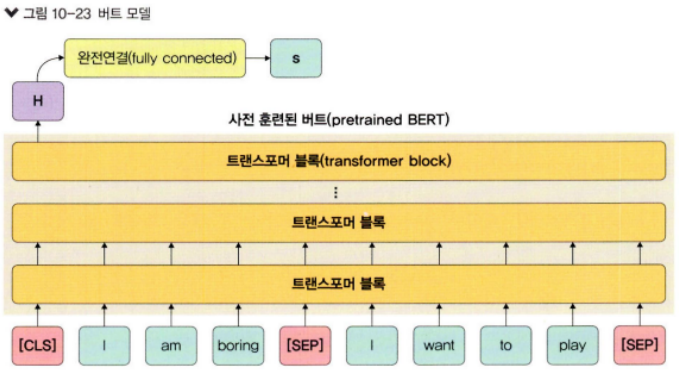
품사 판별과 같은 시퀀스 레이블링과는 차이가 있음
시퀀스 레이블링이란 입력 단어가
$x_1, x_2, ..., x_n$ 이라면 출력은 $y_1, y_2, ..., y_n$이 되는 형태
즉, 입력과 출력에 대한 문자열(sequence)이 같음

△ seq2seq는 품사 판별보다는 번역에 초점을 둔 모델
번역은 입력 시퀀스의
$x_{1:n}$과 의미가 동일한 출력 시퀀스 $y_{1:m}$을 만드는 것
→
x_i, y_i 간의 관계는 중요하지 않음 + 각 시퀀스 길이도 서로 다를 수 있음



10.2.2 버트(BERT)

구글이 공개한 인공지능 언어 모델 BERT는 기존의 단방향 자연어 처리 모델들의 단점을 보완한 양방향 자연어 처리 모델
검색 문장의 단어를 입력된 순서대로 하나씩 처리하는 것이 아니라, 트랜스포머를 이용하여 구현되었으며 방대한 양의 텍스트 데이터로 사전 훈련된 언어 모델



버트의 기본 구조는 트랜스포머라는 인코더를 쌓아 올린 구조로, 주로 문장 예측을 할 때 사용함
튜닝을 통해 최고의 성능을 낸 기존 사례들을 참고해서 사전 학습된 버트 위에 분류를 위한 신경망을 한층 추가하는 방식을 사용함
즉, 버트는 트랜스포머와 사전 학습을 사용하여 성능을 향상시킨 모델

버트의 학습 절차

1. 문장을 버트의 입력 형식에 맞게 변환. 이때 문장의 시작은 [CLS], 문장의 끝은 [SEP]으로 표시함
2. 한 문장의 단어들에 대해 토큰화(tokenization)를 진행함
3. 마지막으로 각 토큰들에 대해 고유의 아이디를 부여함. 토큰이 존재하지 않는 자리는 0으로 채움

버트 모델은 전이 학습을 기반으로 한다고 했는데, 이때 전이는 인코더-디코더로 된 모델
기존 인코더-디코더 모델들과 다르게 CNN, RNN을 이용하지 않고 어텐션 개념을 도입했음
즉, 버트에서 전이 학습은 인코더-디코더 중
인코더만 사용하는 모델

버트는 두 가지 버전이 있음
BERT-base(L=12, H=768, A=12)와 BERT-large(L=24, H=1024, A=16)
L : 전이 블록 숫자, H : 은닉층 크기, A : 전이 블록에서 사용되는 어텐션 블록 숫자

즉, L, H, A가 크다는 것은 블록을 많이 쌓았고, 표현하는 은닉층이 크며 어텐션 개수를 많이 사용했다는 의미