10장. 자연어 처리를 위한 임베딩

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 의미합니다.

임베딩 역할은 다음과 같습니다.

- 단어 및 문장 간 관련성 계산
- 의미적 혹은 문법적 정보의 함축(**예** 왕-여왕, 교사-학생)

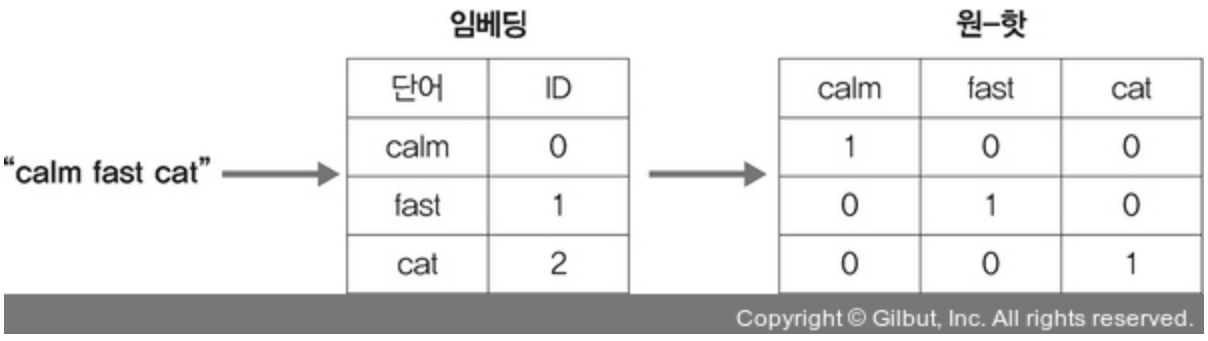
임베딩 방법에 따라 희소 표현 기반 임베딩, 횟수 기반 임베딩, 예측 기반 임베딩, 횟수/예측 기반 임베딩이 있습니다.

10.1.1 희소 표현 기반 임베딩

희소 표현(sparse representation)은 대부분의 값이 0으로 채워져 있는 경우로, 대표적으로 원-핫 인코딩이 있습니다.

원-핫 인코딩

원-핫 인코딩(one-hot encoding)이란 주어진 텍스트를 숫자(벡터)로 변환해 주는 것입니다. 다시 말해 단어 N개를 각각 N차원의 벡터로 표현하는 방식으로, 단어가 포함되어 있는 위치에 1을 넣고 나머지에는 0 값을 채웁니다. 예를 들어 딕셔너리에 [calm, fast, cat] 같은 값이 있다면 fast를 표현하는 벡터는 [0, 1, 0]이 됩니다.



▲ **그림 10-1** 원-핫 인코딩

하지만 원-핫 인코딩에는 치명적인 단점이 있습니다.

첫째, 수학적인 의미에서 원-핫 벡터들은 하나의 요소만 1 값을 갖고 나머지는 모두 0인 희소 벡터(sparse vector)를 갖습니다. 이때 두 단어에 대한 벡터의 내적(inner product)을 구해 보면 0 값을 갖게 되므로 직교(orthogonal)를 이룹니다. 즉, 단어끼리 관계성(유의어, 반의어) 없이 서로 독립적(independent)인 관계가 됩니다.

둘째, '차원의 저주(curse of dimensionality)' 문제가 발생합니다. 하나의 단어를 표현하는 데 말뭉치(corpus)에 있는 수만큼 차원이 존재하기 때문에 복잡해집니다. 예를 들어 단어 10만 개를 포함한 데이터셋에 원-핫 인코딩 배열을 구성한다면 그 차원 개수는 10만 개에 이르게 됩니다.

원-핫 인코딩에 대한 대안으로 신경망에 기반하여 단어를 벡터로 바꾸는 방법론들이 주목을 받고 있습니다. 예를 들어 워드투벡터(Word2Vec), 글로브(GloVe), 패스트텍스트(FastText) 등이 대표적인 방법론입니다.

10.1.2 횟수 기반 임베딩

횟수 기반은 단어가 출현한 빈도를 고려하여 임베딩하는 방법입니다. 대표적으로 카운터 벡터와 TF-IDF가 있습니다.

카운터 벡터

카운터 벡터(counter vector)는 문서 집합에서 단어를 토큰으로 생성하고 각 단어의 출현 빈도수를 이용하여 인코딩해서 벡터를 만드는 방법입니다. 즉, 토크나이징과 벡터화가 동시에 가능한 방법입니다.

카운터 벡터는 사이킷런의 CountVectorizer()를 사용하여 코드로 구현할 수 있습니다. CountVectorizer()는 다음 작업이 가능합니다.

1. 문서를 토큰 리스트로 변환합니다.

2. 각 문서에서 토큰의 출현 빈도를 셉니다.

3. 각 문서를 인코딩하고 벡터로 변환합니다.

TF-IDF

TF-IDF(Term Frequency-Inverse Document Frequency)는 정보 검색론(Information Retrieval, IR)에서 가중치를 구할 때 사용되는 알고리즘입니다.

TF(Term Frequency)(단어 빈도)는 문서 내에서 특정 단어가 출현한 빈도를 의미합니다. 예를 들어 TF에 딥러닝과 신문기사라는 단어가 포함되어 있다고 가정합니다. 이것은 '신문기사'에서 '딥러닝'이라는 단어가 몇 번 등장했는지 의미합니다. 즉, '신문기사'에서 '딥러닝'이라는 단어가

많이 등장한다면 이 기사는 딥러닝과 관련이 높다고 할 수 있으며, 다음 수식을 사용합니다. 이때 tft,d는 특정 문서 d에서 특정 단어 t의 등장 횟수를 의미합니다.

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

혹은

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

$$(t(\text{term}) : \text{단어}, d(\text{document}) : \text{문서 한 개})$$

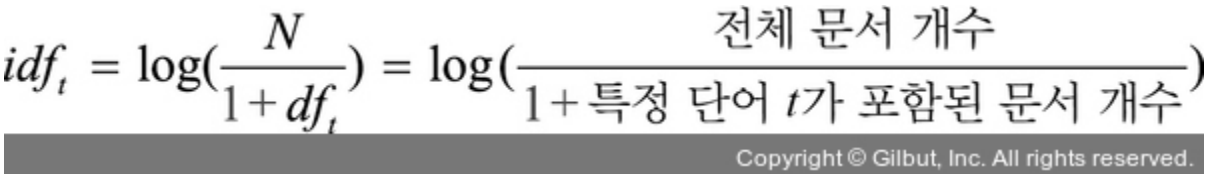
IDF(Inverse Document Frequency)(역문서 빈도)를 이해하려면 DF(Document Frequency)(문서 빈도)에 대한 개념부터 이해해야 합니다. DF는 한 단어가 전체 문서에서 얼마나 공통적으로 많이 등장하는지 나타내는 값입니다. 즉, 특정 단어가 나타난 문서 개수라고 이해하면 됩니다.

dft = 특정 단어 t가 포함된 문서 개수

특정 단어 t가 모든 문서에 등장하는 일반적인 단어(**예** a, the)라면, TF-IDF 가중치를 낮추어 줄 필요가 있습니다. 따라서 DF 값이 클수록 TF-IDF의 가중치 값을 낮추기 위해 DF 값에 역수를 취하는데, 이 값이 IDF입니다. 역수를 취하면 전체 문서 개수가 많아질수록 IDF 값도 커지므로 IDF는 로그(log)를 취해야 합니다. 이것을 수식으로 표현하면 다음과 같습니다.

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

이때 중요한 점은 전체 문서에 특정 단어가 발생하는 빈도가 0이라면 분모가 0이 되는 상황이 발생합니다. 이를 방지하고자 다음과 같이 분모에 1을 더해 주는 것을 스무딩(smoothing)이라고 합니다.



TF-IDF는 다음 상황에서 사용됩니다.

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

10.1.3 예측 기반 임베딩

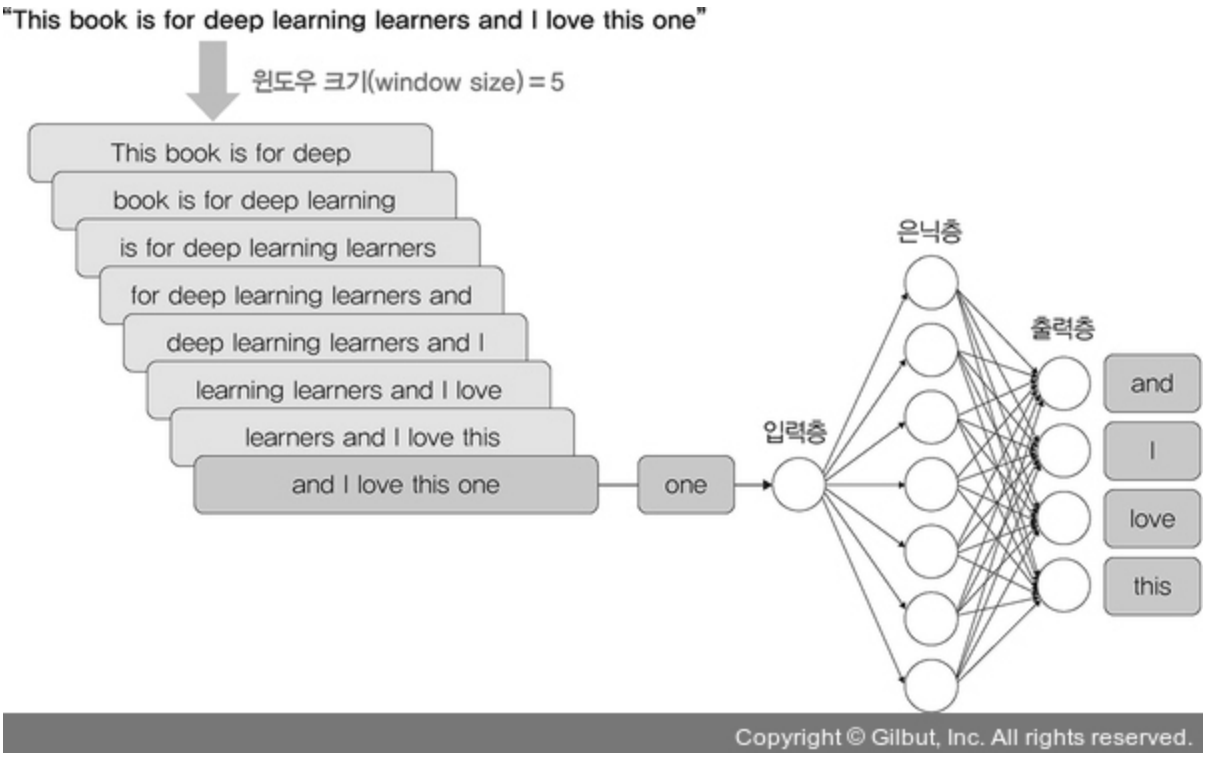
예측 기반 임베딩은 신경망 구조 혹은 모델을 이용하여 특정 문맥에서 어떤 단어가 나올지 예측하면서 단어를 벡터로 만드는 방식입니다. 대표적으로 워드투벡터가 있습니다.

워드투벡터

워드투벡터(Word2Vec)는 신경망 알고리즘으로, 주어진 텍스트에서 텍스트의 각 단어마다 하나씩 일련의 벡터를 출력합니다.

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

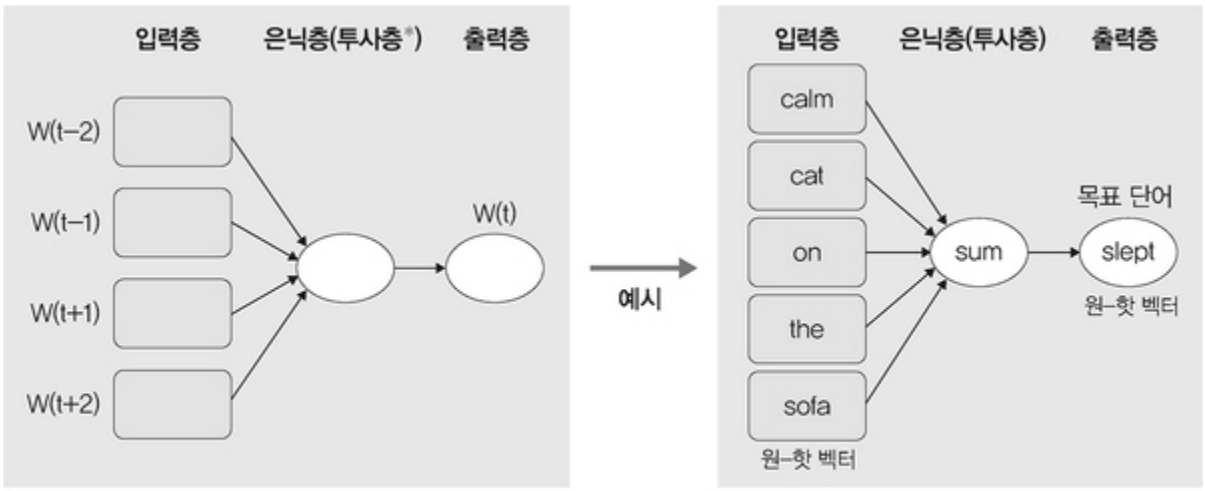
워드투벡터가 수행되는 과정은 다음 그림과 같습니다. 일정한 크기의 윈도우(window)로 분할된 텍스트를 신경망 입력으로 사용합니다. 이때 모든 분할된 텍스트는 한 쌍의 대상 단어와 컨텍스트로 네트워크에 공급됩니다. 다음 그림과 같이 대상 단어는 'one'이고 컨텍스트는 'and', 'I', 'love', 'this' 단어로 구성됩니다. 또한, 네트워크의 은닉층에는 각 단어에 대한 가중치가 포함되어 있습니다.



CBOW

CBOW(Continuous Bag Of Words)는 단어를 여러 개 나열한 후 이와 관련된 단어를 추정하는 방식입니다. 즉, 문장에서 등장하는 n개의 단어 열에서 다음에 등장할 단어를 예측합니다. 예를 들어 “calm cat slept on the sofa”라는 문장이 있을 때, “calm cat on the sofa”라는 문맥이 주어지면 “slept”를 예측하는 것이 CBOW입니다.

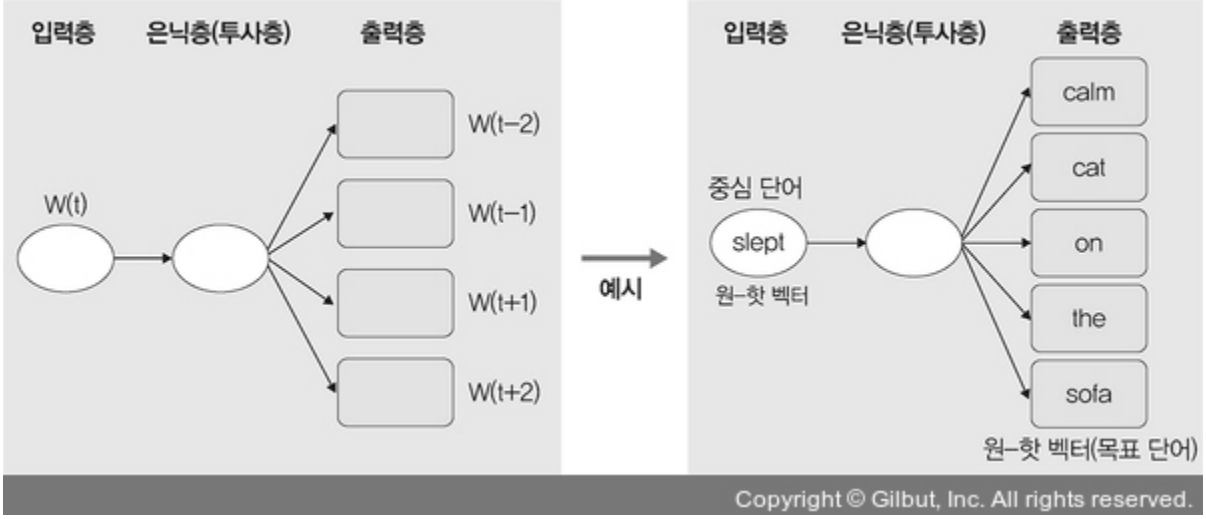
CBOW는 다음 그림과 같은 신경망 구조를 갖습니다. 여기에서 각 문맥 단어를 은닉층으로 투사하는 가중치 행렬은 모든 단어에서 공통으로 사용됩니다.



CBOW의 신경망을 좀 더 자세히 살펴보겠습니다. CBOW 신경망에서 크기가 N인 은닉층을 가지고 있을 때, 은닉층 크기 N은 입력 텍스트를 임베딩한 벡터 크기입니다. 다시 말해 다음 그림에서 은닉층 크기는 N=5이기 때문에 해당 CBOW를 수행한 후 벡터 크기는 5가 됩니다. 다음으로 입력층과 은닉층 사이의 가중치 W는 V×N 행렬이며, 은닉층에서 출력층 사이의 가중치 W′는 N×V 행렬입니다. 여기에서 V는 단어 집합의 크기를 의미합니다. 즉, 다음 그림과 같이 원-핫 벡터의 차원이 7이고, N이 5라면 가중치 W는 7×5 행렬이고, W′는 5×7 행렬이 됩니다.

skip-gram

skip-gram 방식은 CBOW 방식과 반대로 특정한 단어에서 문맥이 될 수 있는 단어를 예측합니다. 즉, skip-gram은 다음 그림과 같이 중심 단어에서 주변 단어를 예측하는 방식을 사용합니다. 예를 들어 다음 그림과 같이 중심 단어 'slept'를 이용하여 그 앞과 뒤의 단어들을 유추하는 것이 skip-gram입니다.

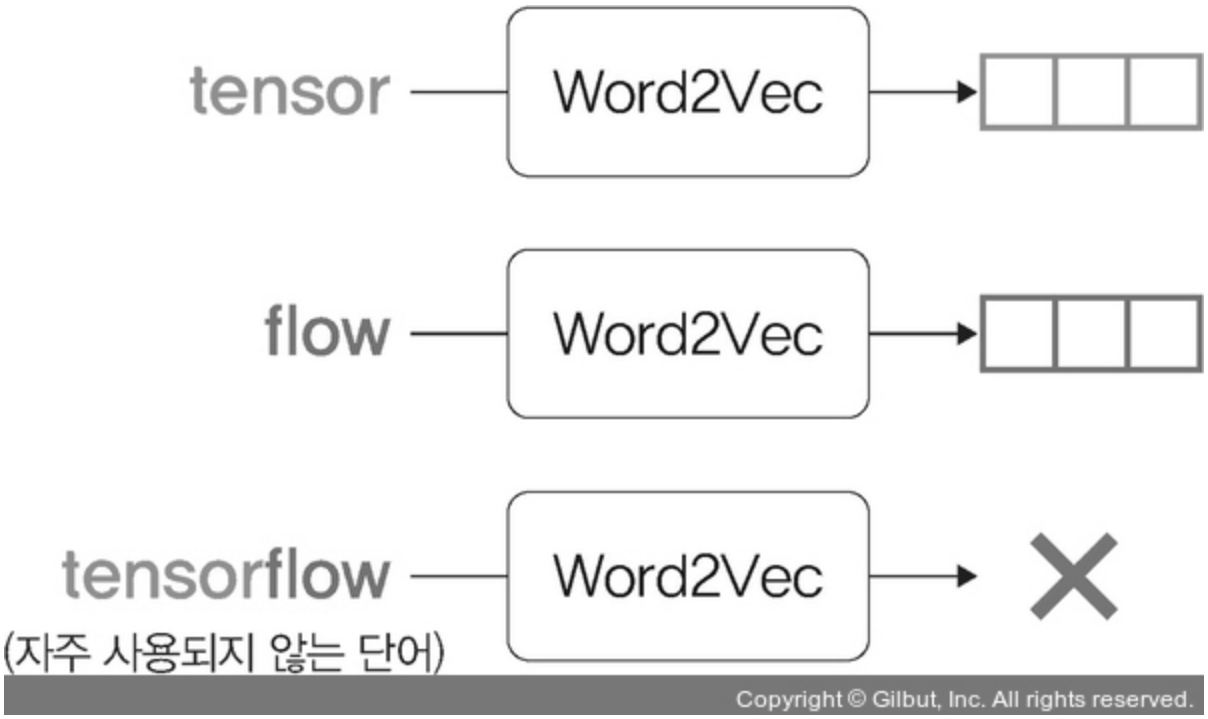


▲ **그림 10-5** skip-gram

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

패스트텍스트

패스트텍스트(FastText)는 워드투벡터의 단점을 보완하고자 페이스북에서 개발한 임베딩 알고리즘입니다. 기존 워드투벡터의 워드 임베딩 방식은 분산 표현(distributed representation)을 이용하여 단어의 분산 분포가 유사한 단어들에 비슷한 벡터 값을 할당하여 표현합니다. 따라서 워드투벡터는 사전에 없는 단어에 대해서는 벡터 값을 얻을 수 없습니다. 또한, 워드투벡터는 자주 사용되지 않는 단어에 대해서는 학습이 불안정합니다.

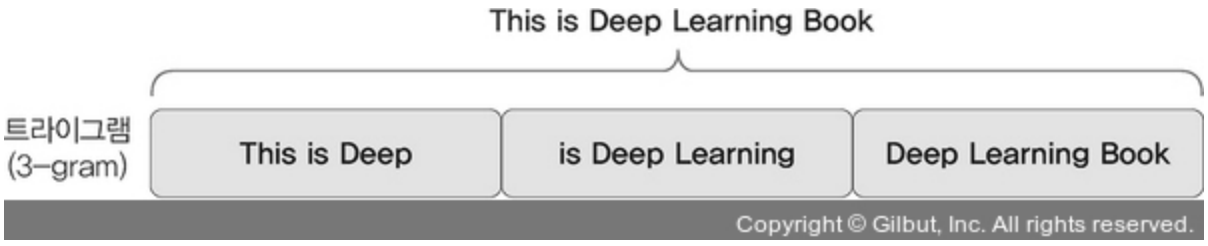


▲ **그림 10-6** 워드투벡터 단점

패스트텍스트는 이러한 단점들을 보완하려고 개발된 단어 표현(word representation) 방법을 사용합니다. 패스트텍스트는 노이즈에 강하며, 새로운 단어에 대해서는 형태적 유사성을 고려한 벡터 값을 얻기 때문에 자연어 처리 분야에서 많이 사용되는 알고리즘입니다. 패스트텍스트가 워드투벡터 단점을 극복하는 방법은 다음과 같습니다.

- 사전에 없는 단어에 벡터 값을 부여하는 방법

패스트텍스트는 주어진 문서의 각 단어를 n-그램(n-gram)으로 표현합니다. 이때 n의 설정에 따라 단어의 분리 수준이 결정됩니다. 예를 들어 n을 3으로 설정(트라이그램(trigram))하면 'This is Deep Learning Book'은 This is Deep, is Deep Learning, Deep Learning Book으로 분리한 후 임베딩합니다. 즉, 다음 그림과 같이 분리됩니다.



▲ **그림 10-7** 트라이그램

n 값에 따른 단어의 분리(부분 단어(subword))는 다음 표와 같습니다.

▼ **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 임베딩합니다. 따라서 사전에 없는 단어가 등장한다면 n-그램으로 분리된 부분 단어와 유사도를 계산하여 의미를 유추할 수 있습니다.

- 자주 사용되지 않는 단어에 학습 안정성을 확보하는 방법

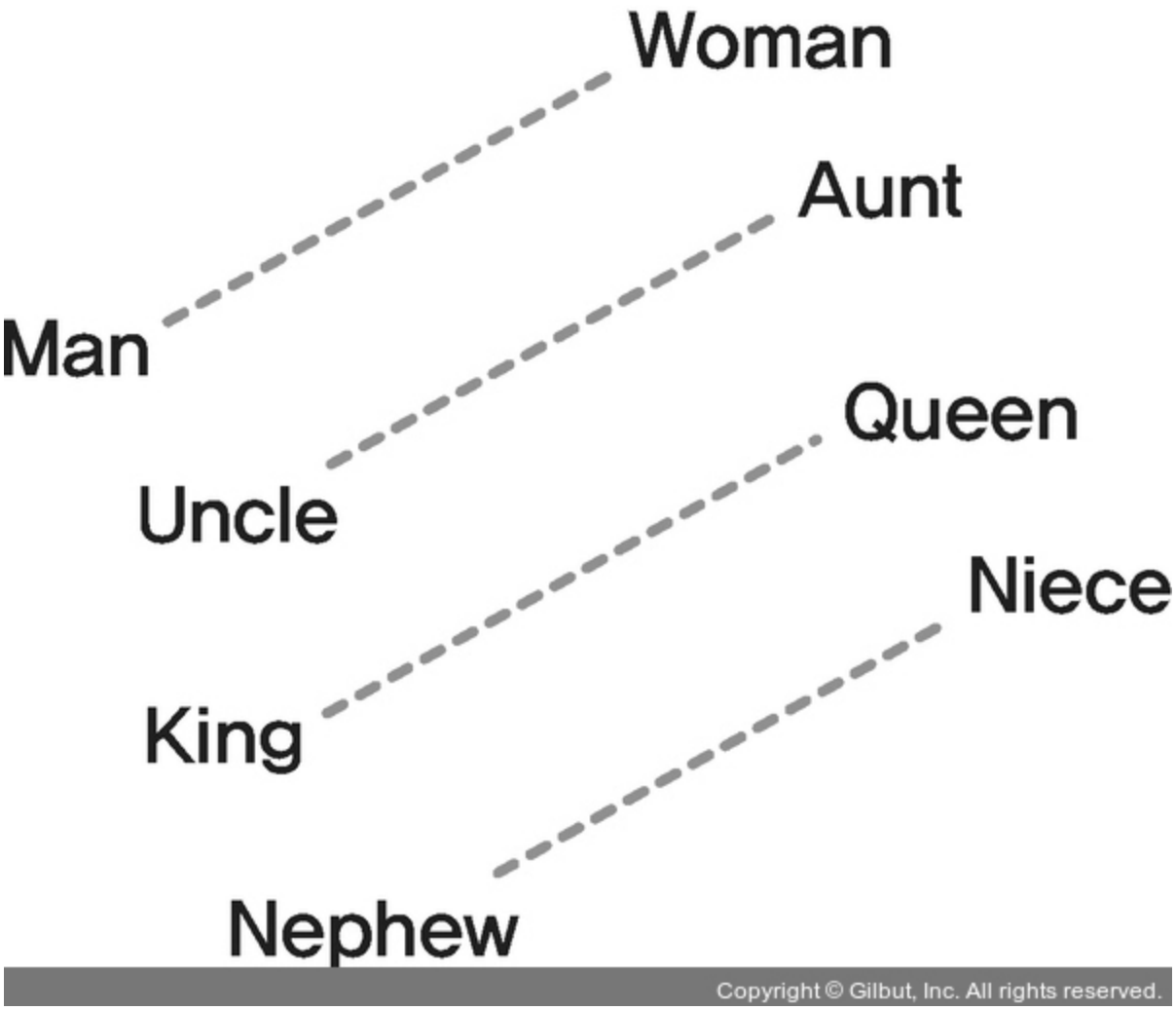
워드투벡터는 단어의 출현 빈도가 적으면 임베딩의 정확도가 낮은 단점이 있었습니다. 참고할 수 있는 경우의 수가 적기 때문에 상대적으로 정확도가 낮아 임베딩되지 않습니다. 하지만 패스트텍스트는 등장 빈도수가 적더라도, n-그램으로 임베딩하기 때문에 참고할 수 있는 경우의 수가 많습니다. 따라서 상대적으로 자주 사용되지 않는 단어에서도 정확도가 높습니다.

10.1.4 횟수/예측 기반 임베딩

앞서 살펴본 횟수 기반과 예측 기반의 단점을 보완하기 위한 임베딩 기법에는 대표적으로 **글로브**가 있습니다.

글로브

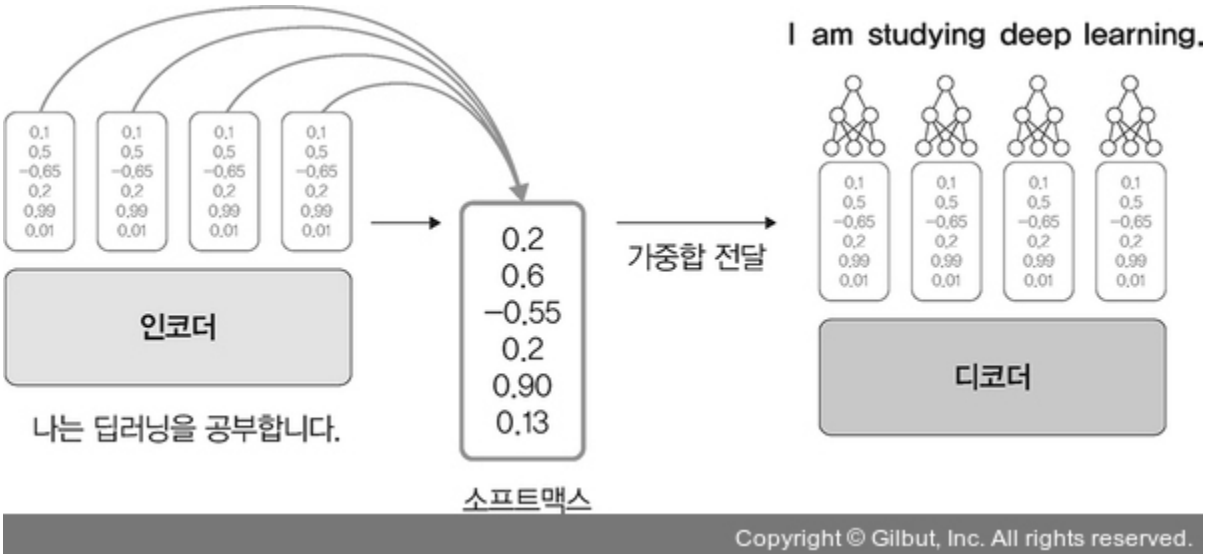
글로브(GloVe, Global Vectors for Word Representation)는 횟수 기반의 LSA(Latent Semantic Analysis)(잠재 의미 분석)와 예측 기반의 워드투벡터 단점을 보완하기 위한 모델입니다. 글로브는 그 이름에서 유추할 수 있듯이 단어에 대한 글로벌 동시 발생 확률(global co-occurrence statistics) 정보를 포함하는 단어 임베딩 방법입니다. 즉, 단어에 대한 통계 정보와 skip-gram을 합친 방식이라고 할 수 있습니다. 다시 풀어서 이야기하면 skip-gram 방법을 사용하되 통계적 기법이 추가된 것이라고 할 수 있습니다. 따라서 글로브를 사용하면 다음 그림과 같이 단어 간 관련성을 통계적 방법으로 표현해 줍니다.



10.2 트랜스포머 어텐션

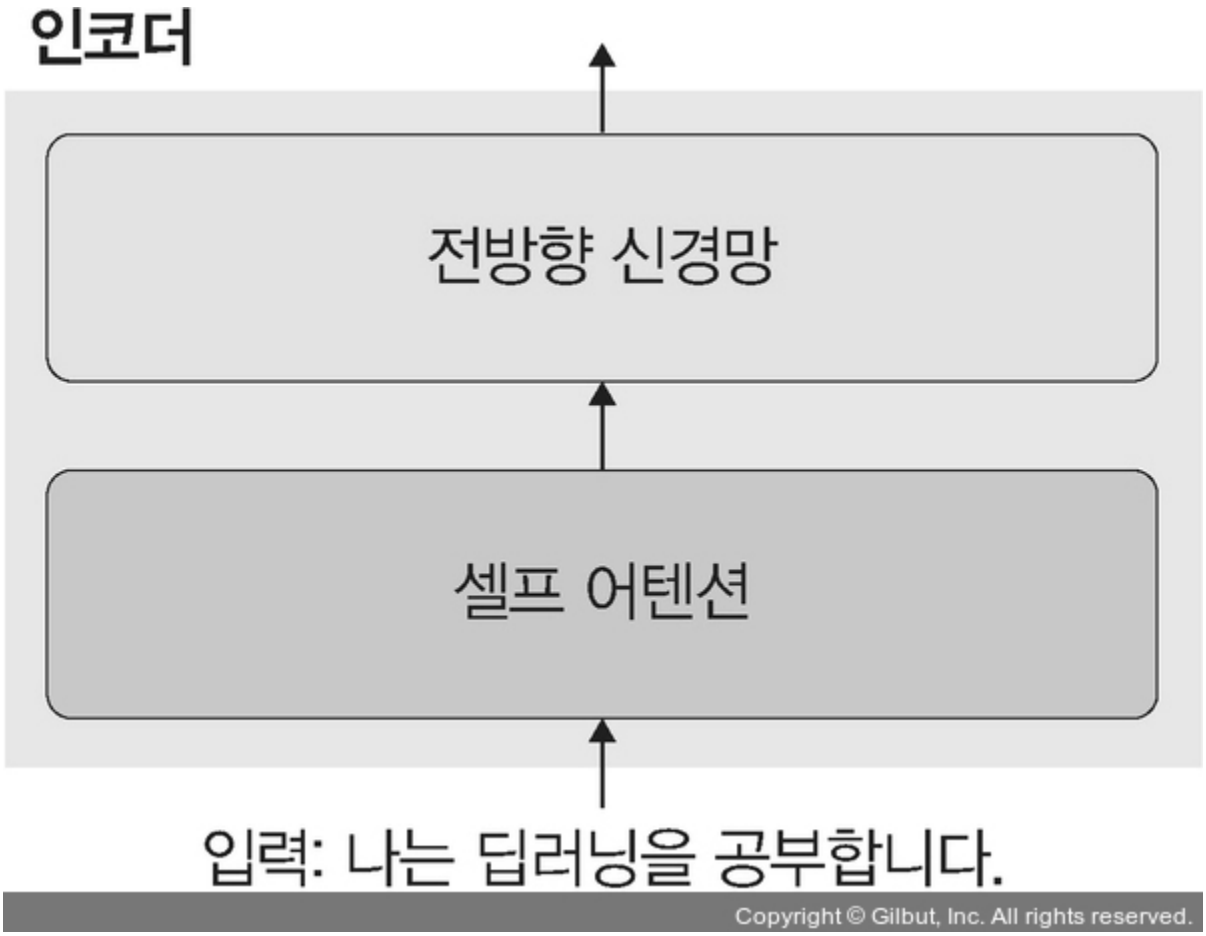
어텐션(attention)은 주로 언어 번역에서 사용되기 때문에 인코더와 디코더 네트워크를 사용합니다. 즉, 입력에 대한 벡터 변환을 인코더(encoder)에서 처리하고 모든 벡터를 디코더로 보냅니다. 이렇게 모든 벡터를 전달하는 이유는 시간이 흐를수록 초기 정보를 잃어버리는 기울기 소멸 문제를 해결하기 위해서입니다. 하지만 모든 벡터가 전달되기 때문에 행렬 크기가 굉장히 커지는 단점이 있는데, 이것을 해결하기 위해 소프트맥스 함수를 사용하여 가중합을 구하고 그 값을 디코더에 전달합니다.

가중합만 전달되었더라도 정보를 많이 전달받은 디코더는 부담일 수밖에 없습니다. 따라서 디코더는 은닉 상태에 대해 중점적으로 집중(attention)해서 보아야 할 벡터를 소프트맥스 함수로 점수를 매긴 후 각각을 은닉 상태의 벡터들과 곱합니다. 그리고 이 은닉 상태를 모두 더해서 하나의 값으로 만듭니다. 즉, 어텐션은 모든 벡터 중에서 꼭 살펴보아야 할 벡터들에 집중하겠다는 의미입니다.



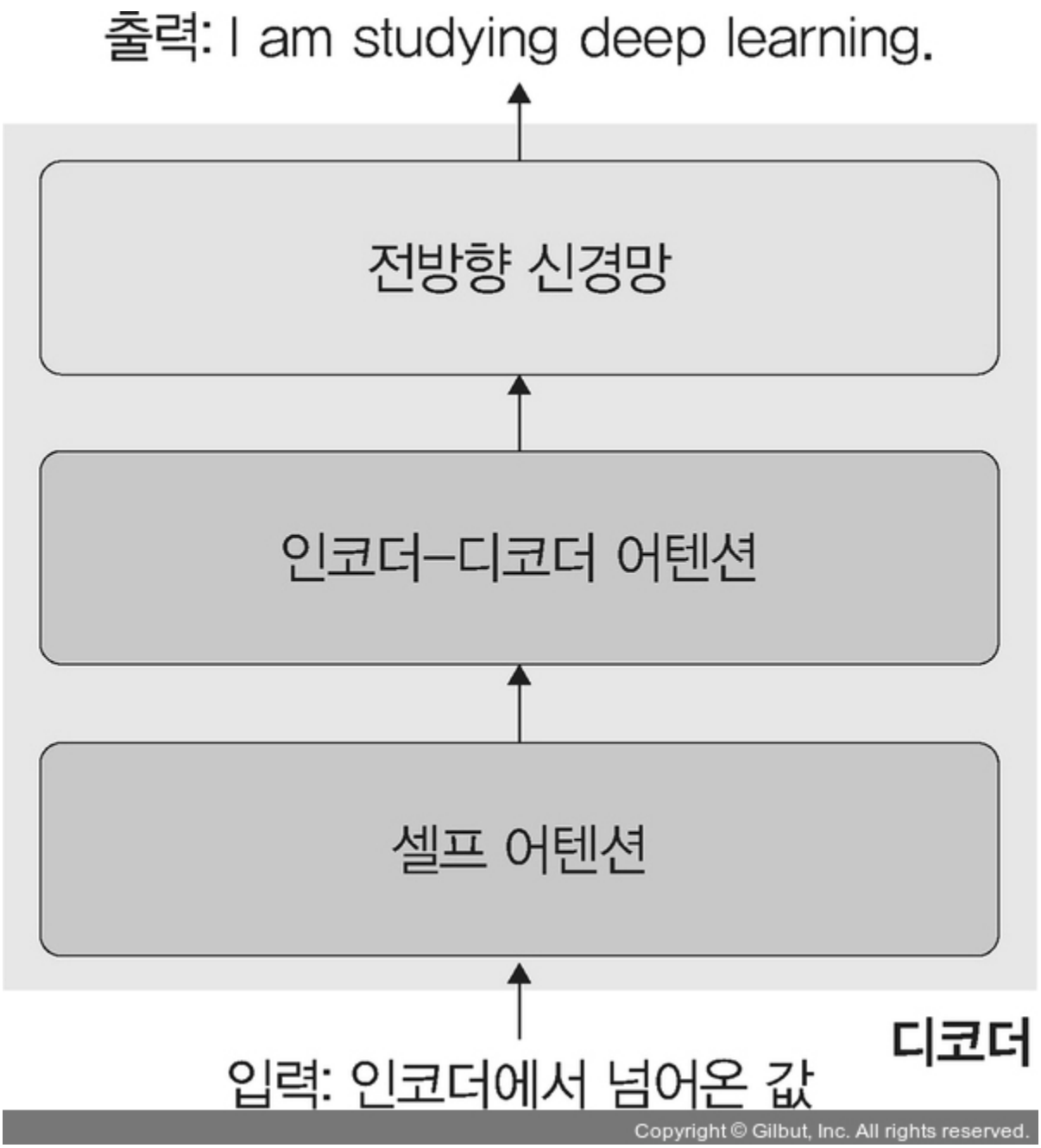
트랜스포머(transformer)는 2017년 6월에 “Attention is All You Need”(Ashish Vaswani et al.) 논문에서 발표된 것으로 어텐션을 극대화하는 방법입니다. 어텐션에서 다룬 인코더와 디코더에는 네트워크가 하나씩 있었습니다. 하지만 트랜스포머는 인코더와 디코더를 여러 개 중첩시킨 구조입니다. 이때 각각의 인코더와 디코더를 블록(block)이라고 합니다(논문에서는 인코더 블록과 디코더 블록을 여섯 개씩 중첩시킨 구조를 사용합니다).

먼저 **인코더** 블록 구조를 살펴보겠습니다. 참고로 모든 블록 구조는 동일합니다. 하나의 인코더는 셀프 어텐션(self-attention)과 전방향 신경망(feed forward neural network)으로 구성되어 있습니다. 인코더에서는 단어를 벡터로 임베딩하며, 이를 셀프 어텐션과 전방향 신경망으로 전달합니다. 이때 셀프 어텐션은 문장에서 각 단어끼리 얼마나 관계가 있는지를 계산해서 반영합니다. 즉, 셀프 어텐션으로 문장 안에서 단어 간 관계를 파악할 수 있습니다. 셀프 어텐션에서 파악된 단어 간 관계는 전방향 신경망으로 전달됩니다.



▲ **그림 10-11** 어텐션의 인코더 상세 구조

이번에는 **디코더**를 알아보겠습니다. 디코더는 층을 총 세 개 가지고 있는데, 인코더에서 넘어온 벡터가 처음으로 만나는 것이 셀프 어텐션 층입니다. 즉, 인코더와 동일하다고 이해하면 됩니다. 셀프 어텐션 층을 지나면 인코더-디코더 어텐션(encoder-decoder attention) 층이 있습니다. 인코더-디코더 어텐션 층에서는 인코더가 처리한 정보를 받아 어텐션 메커니즘을 수행하고, 마지막으로 전방향 신경망으로 데이터가 전달됩니다.



어텐션 메커니즘을 구체적으로 수식을 이용하여 알아보겠습니다.

어텐션 메커니즘을 이용하기 위해서는 가장 먼저 어텐션 스코어를 구해야 합니다.

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

어텐션 스코어란 현재 디코더의 시점 i에서 단어를 예측하기 위해, 인코더의 모든 은닉 상태 값(hj)이 디코더의 현 시점의 은닉 상태(si)와 얼마나 관련이 있는지(유사한지)를 판단하는 값입니다. 따라서 어텐션 스코어는 앞의 수식처럼 인코더의 모든 은닉 상태의 값(hj)과 디코더에서의 이전 시점의 은닉 상태(si-1) 값을 이용하여 구할 수 있습니다.

어텐션 스코어가 계산되었다면 이 값을 소프트맥스(softmax) 함수에 적용하여 확률로 변환합니다. 이렇게 계산된 0~1 사이의 값들이 특정 시점(time step)에 대한 가중치, 즉 시간의 가중치가 되며 다음과 같은 수식을 이용합니다.

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

이제 시간의 가중치(aij)와 은닉 상태(hj)의 가중합을 계산하면 하나의 벡터가 계산되는데, 이것이 컨텍스트 벡터(context vector)입니다. 수식은 다음과 같습니다.

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

마지막으로 디코더의 은닉 상태를 구해야 합니다. 디코더의 은닉 상태를 구하기 위해서는 컨텍스트 벡터와 디코더 이전 시점의 은닉 상태와 출력이 필요합니다. 다음과 같이 어텐션이 적용된 인코더-디코더의 수식에서는 컨텍스트 벡터(ci)가 계속 변하고 있는 반면에 어텐션이 적용되지 않은 인코더-디코더 수식에서는 컨텍스트 벡터(c)가 고정되어 있습니다. 이전 시점의 은닉 상태 값을 구하는 공식은 다음과 같습니다.

(어텐션 적용) 이전 시점의 은닉 상태 값을 구하는 수식:

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

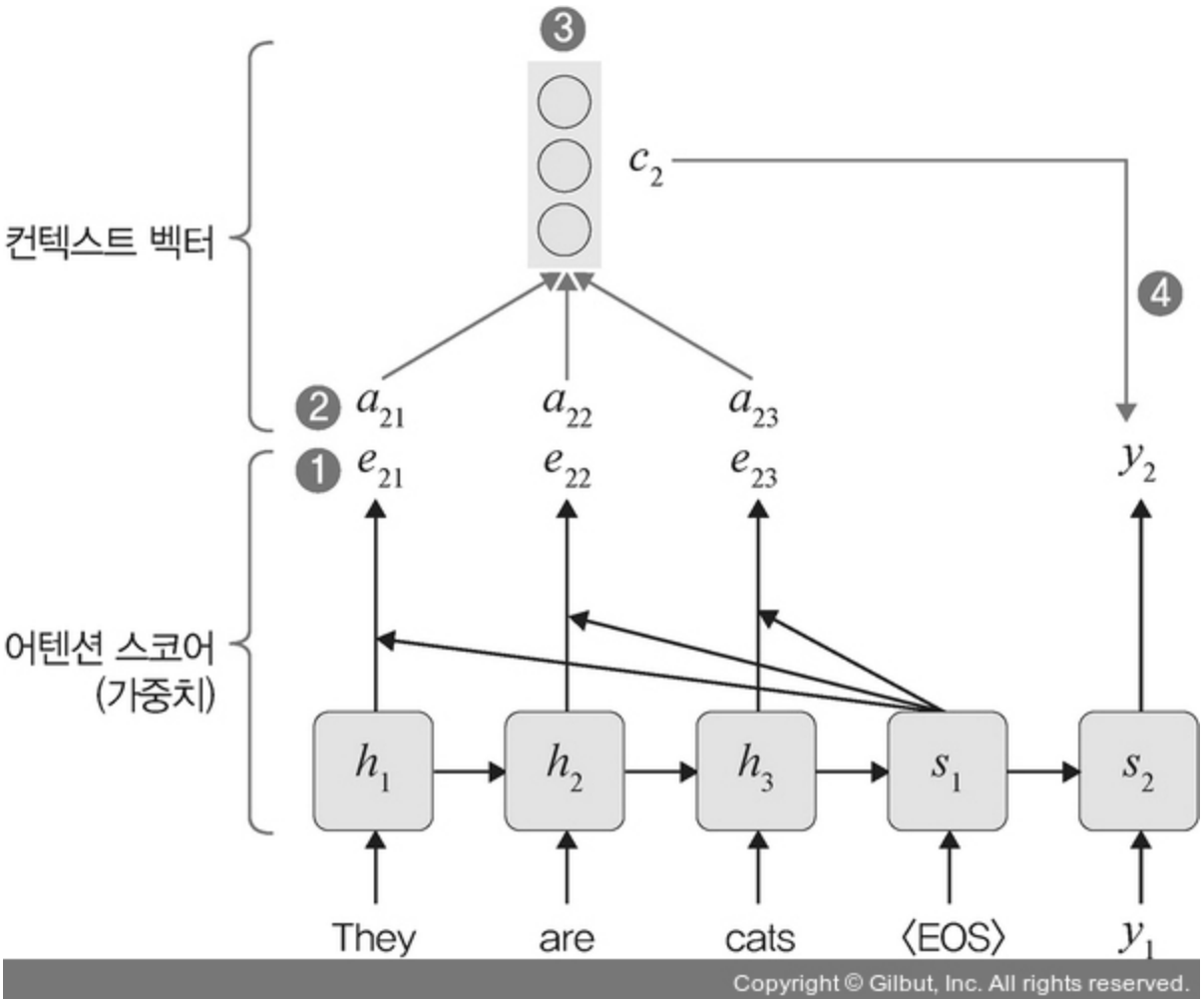
(어텐션 미적용) 이전 시점의 은닉 상태 값을 구하는 수식:

$$s_i = f(s_{i-1}, y_{i-1}, c)$$

이 모든 과정을 순전파 과정을 통해 정리하면 다음과 같습니다.

‘They are cats’라는 입력 시퀀스에 대해 현 시점의 디코더 은닉 상태인 s2와 그 출력인 y2를 계산하기 위해 먼저 s1과 모든 인코더 은닉 상태에 대한 어텐션 스코어를 계산합니다. e21, e22, e23처럼 어텐션 스코어를 구했으면 소프트맥스 함수를 적용하여 시간의 가중치(a21, a22, a23)를 구합니다. 이후 ‘시간의 가중치’와 인코더의 은닉 상태 값들을 이용하여 가중합을 계산함으로써 컨텍스트 벡터(c2)를 구합니다.

최종적으로 앞에서 구한 컨텍스트 벡터와 디코더 이전 시점의 은닉 상태와 출력을 이용하여 최종적으로 다음 디코더의 은닉 상태(y2)를 출력합니다.

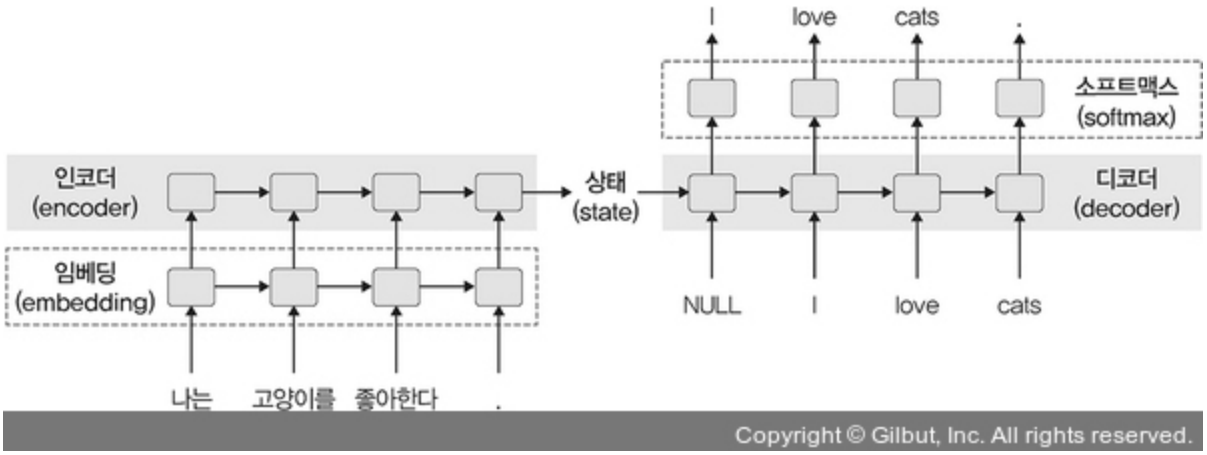


▲ **그림 10-13** 어텐션 메커니즘 예시

자세한 구현 방법은 이어서 seq2seq, 버트(BERT)에서 예제로 알아보겠습니다.

10.2.1 seq2seq

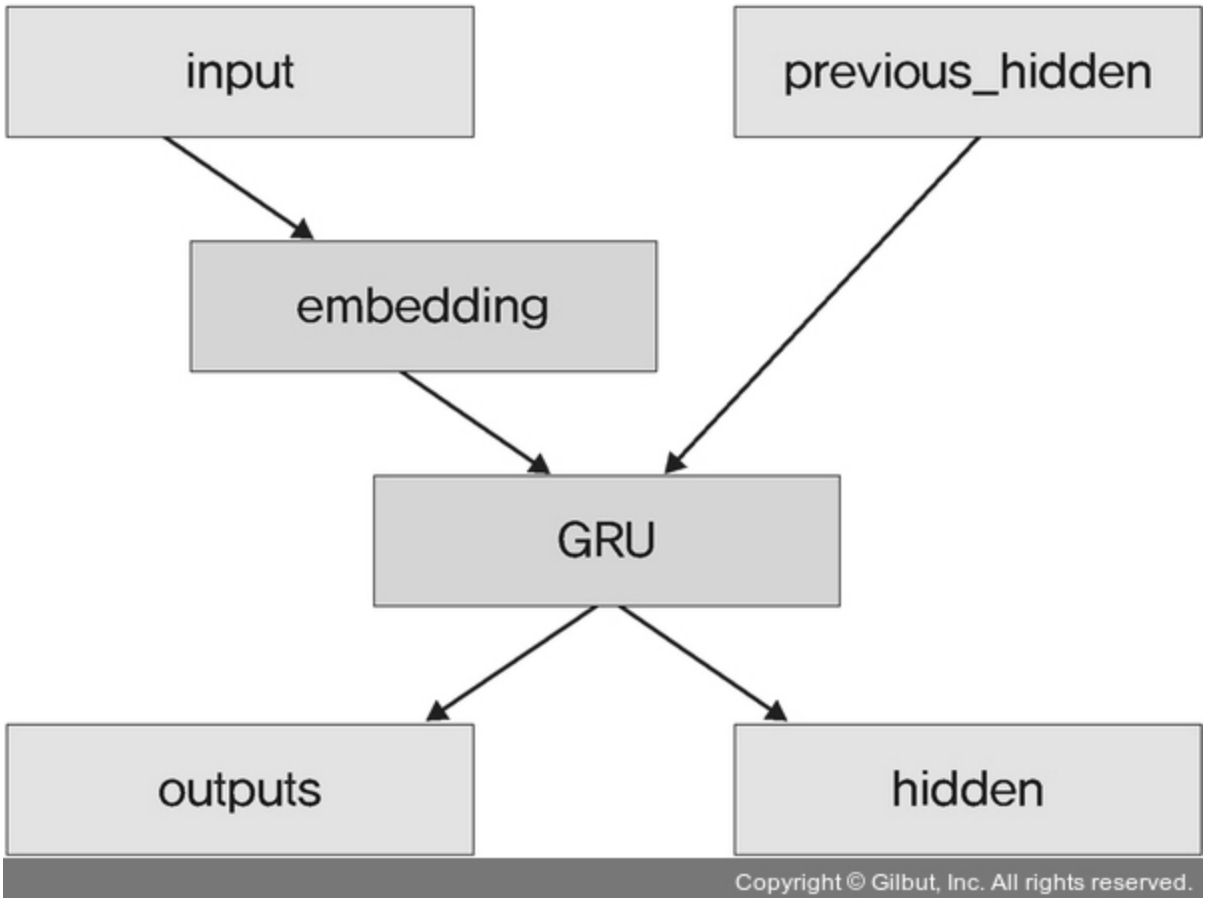
seq2seq(sequence to sequence)는 입력 시퀀스(input sequence)에 대한 출력 시퀀스(output sequence)를 만들기 위한 모델입니다. seq2seq는 품사 판별과 같은 시퀀스 레이블링(sequence labeling)과는 차이가 있습니다. 시퀀스 레이블링이란 입력 단어가 x1, x2, …, xn이라면 출력은 y1, y2, …, yn이 되는 형태입니다. 즉, 입력과 출력에 대한 문자열(sequence)이 같습니다. 하지만 seq2seq는 품사 판별보다는 번역에 초점을 둔 모델입니다. 번역은 입력 시퀀스의 x1:n과 의미가 동일한 출력 시퀀스 y1:m을 만드는 것이며, xi, yi 간의 관계는 중요하지 않습니다. 그리고 각 시퀀스 길이도 서로 다를 수 있습니다.



▲ **그림 10-14** seq2seq

파이토치에서 seq2seq 모델을 사용하기 위해서는 먼저 인코더와 디코더를 정의해야 합니다. 입력(영어) 문장이 인코더로 주입되면 디코더(프랑스어)로 번역되어 출력됩니다. 인코더와 디코더를 이용하면 문장의 번역뿐만 아니라 다음 입력을 예측하는 것도 가능합니다. 이때 각 입력 문장의 끝에는 문장의 끝을 알리는 토큰이 할당됩니다.

인코더는 입력 문장을 단어별로 순서대로 인코딩을 하게 되며, 문장의 끝을 표시하는 토큰이 붙습니다. 또한, 인코더는 임베딩 계층과 GRU 계층으로 구성됩니다. 코드 10-29와 비교하기 위해 인터넷 네트워크를 영문으로 작성했습니다.



▲ **그림 10-16** 임베딩 네트워크

임베딩 계층은 입력에 대한 임베딩 결과가 저장되어 있는 딕셔너리를 조회하는 테이블과도 같습니다. 이후 GRU 계층과 연결되는데, GRU 계층은 연속하여 들어오는 입력을 계산합니다. 또한, 이전 계층의 은닉 상태를 계산한 후 망각 게이트와 업데이트 게이트를 갱신합니다.

정확하게 번역되었는지 확인하기 위해 구글과 같은 번역 사이트를 이용해 보세요. 정확하지는 않지만 비슷하게 번역된 듯합니다.

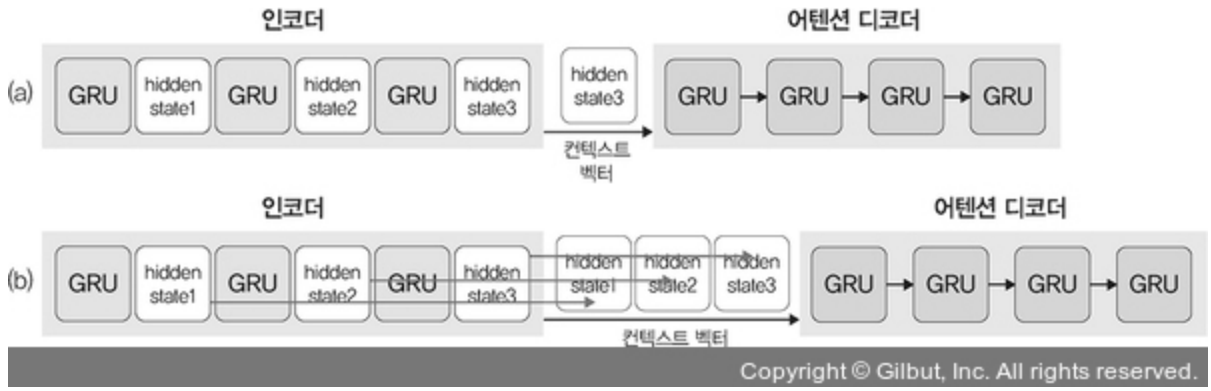
seq2seq는 인코더와 디코더 네트워크를 사용합니다. 하지만 일반적인 seq2seq 모델은 입력 문장이 긴 시퀀스일 경우 정확한 처리가 어렵습니다. 그림 10-20의 (a)와 같이 인코더에서 사용하는 RNN(LSTM, GRU)의 마지막 은닉 상태만 디코더로 전달되기 때문입니다. 즉, 다음과 같은 이유로 어텐션 메커니즘(attention mechanism)이 등장했습니다.

1. 하나의 고정된 크기의 벡터에 모든 정보를 담다 보니 정보의 손실 발생

2. RNN에서 발생할 수 있는 기울기 소멸(vanishing gradient) 문제 발생

어텐션 메커니즘은 그림 10-20의 (b)와 같이 디코딩 프로세스 동안 입력 시퀀스의 모든 숨겨진 상태를 유지하고 활용하기 때문에 정보의 손실과 기울기 소멸 문제가 발생하지 않습니다. 즉, 그

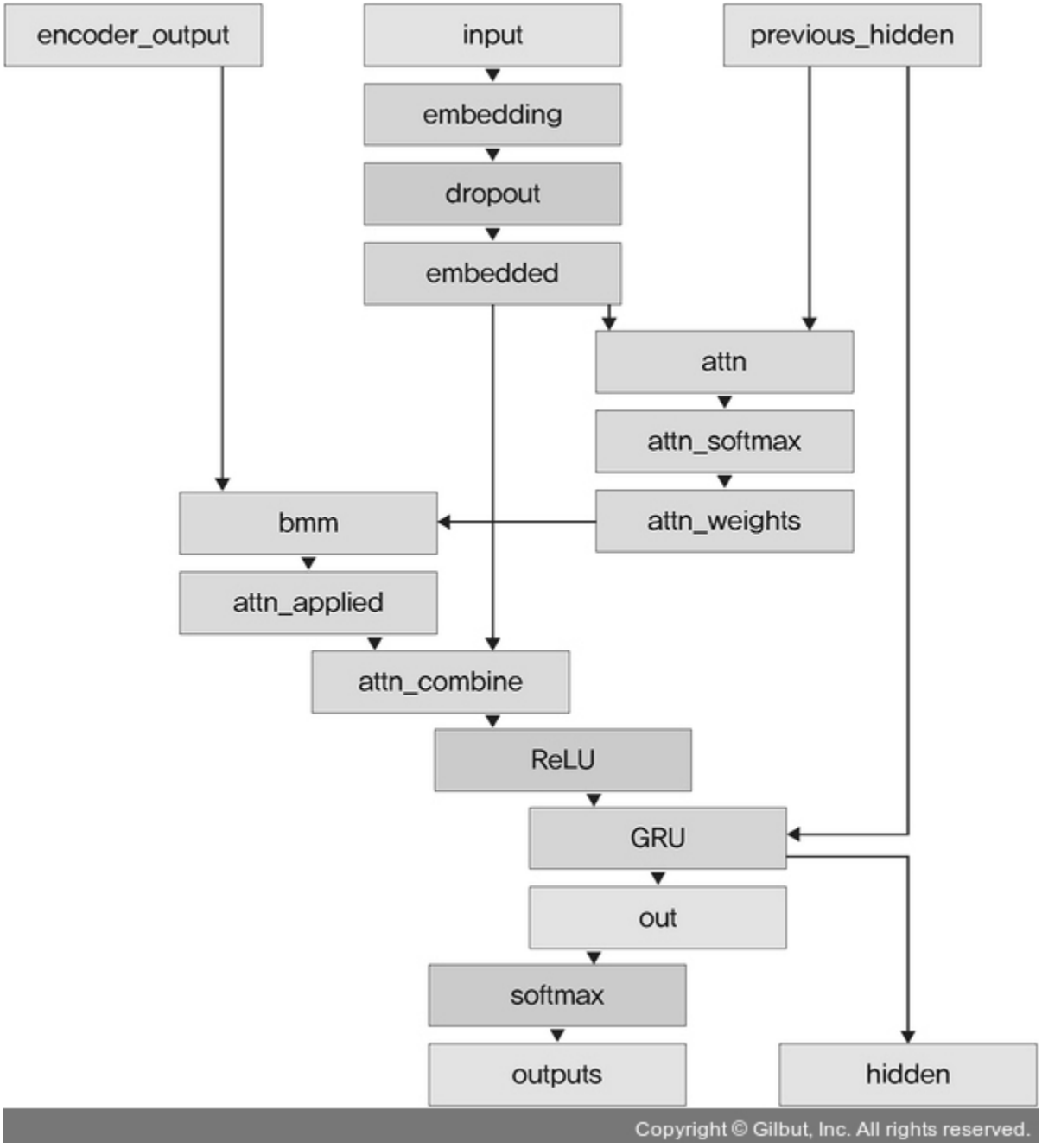
림 10-20의 (b)와 같이 컨텍스트 벡터는 인코더의 전체 은닉 상태들과 디코더의 이전 은닉 상태를 바탕으로 만들어집니다.



▲ **그림 10-20** 어텐션 디코딩

정리하면 어텐션 메커니즘이 쓰이지 않은 기존 모델과 다른 점은 디코더에서 컨텍스트 벡터가 모두 같은 것을 쓰거나 단순히 전파되는 것이 아니라 특정 시점(time step)마다 다른 컨텍스트 벡터를 사용한다는 것입니다.

다음 그림은 코드 10-37에서 사용되는 모델의 네트워크를 도식화한 것입니다. 모델과 비교를 위해 네트워크의 영문명을 그대로 사용했습니다.

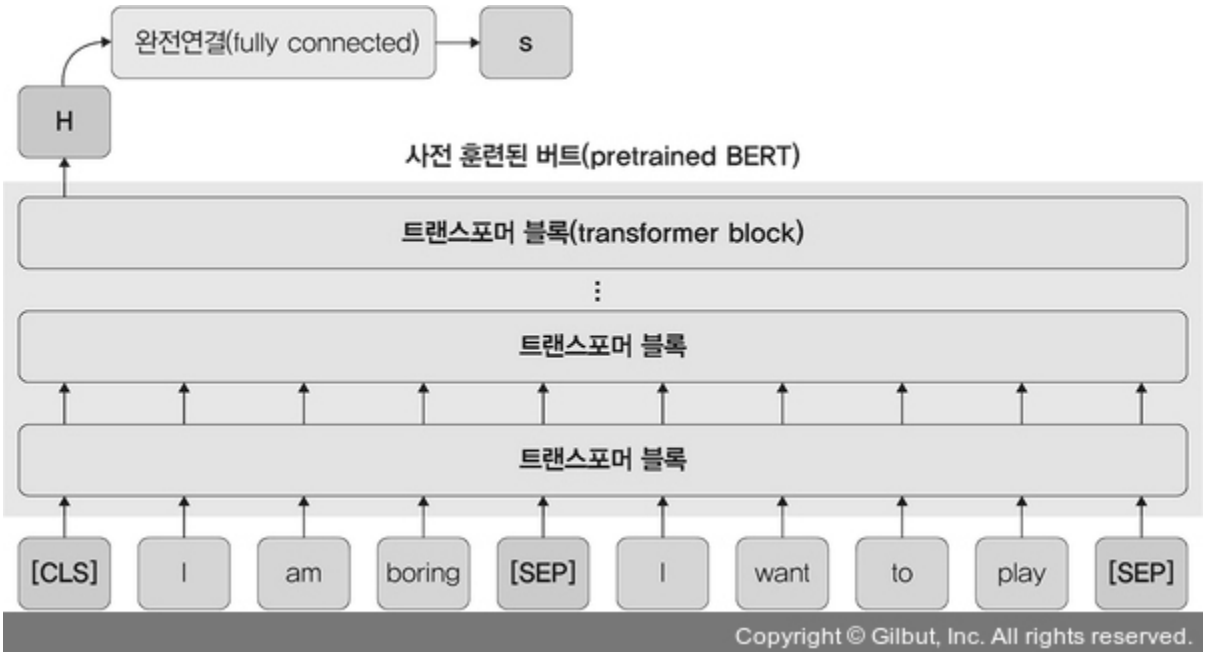


▲ **그림 10-21** 어텐션이 적용된 디코더

10.2.2 버트(BERT)

2018년 11월, 구글이 공개한 인공지능(AI) 언어 모델 BERT(Bidirectional Encoder Representations from Transformers)(이하 버트)는 기존의 단방향 자연어 처리 모델들의 단점을 보완한 양방향 자연어 처리 모델입니다. 검색 문장의 단어를 입력된 순서대로 하나씩 처리하

는 것이 아니라, 트랜스포머를 이용하여 구현되었으며 방대한 양의 텍스트 데이터로 사전 훈련된 언어 모델입니다.



▲ **그림 10-23** 버트 모델

버트의 기본 구조는 그림 10-23과 같이 트랜스포머(transformer)라는 인코더를 쌓아 올린 구조로, 주로 문장 예측(Next Sentence Prediction, NSP)을 할 때 사용합니다. 튜닝**4**을 통해 최고의 성능을 낸 기존 사례들을 참고해서 사전 학습된 버트 위에 분류를 위한 신경망을 한층 추가하는 방식을 사용합니다. 즉, 버트는 트랜스포머와 사전 학습을 사용하여 성능을 향상시킨 모델입니다.

버트의 학습 절차는 다음과 같습니다.

1. 그림 10-23과 같이 문장을 버트의 입력 형식에 맞게 변환합니다. 이때 문장의 시작은 [CLS], 문장의 끝은 [SEP]로 표시합니다.

2. 한 문장의 단어들에 대해 토큰화(tokenization)를 진행합니다. 예를 들어 '고양이'라는 단어의 경우 '고##', '#양#', '##이'로 토큰화합니다.

3. 마지막으로 각 토큰들에 대해 고유의 아이디를 부여합니다. 토큰이 존재하지 않는 자리는 0으로 채웁니다.

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

버트는 두 가지 버전이 있는데, BERT-base(L=12, H=768, A=12)와 BERT-large(L=24, H=1024, A=16)입니다. 이때 L은 전이 블록 숫자이고, H는 은닉층 크기, A는 전이 블록에서 사용되는 어텐션 블록 숫자입니다. 즉, L, H, A가 크다는 것은 블록을 많이 쌓았고, 표현하는 은닉층이 크며 어텐션 개수를 많이 사용했다는 의미입니다. BERT-base는 학습 파라미터 1.1억 개가 있고, BERT-large는 학습 파라미터 3.4억 개가 있습니다.

10.3 한국어 임베딩

지금까지 영어에 대한 임베딩을 진행했는데, 한국어에 대한 임베딩도 영어와 동일합니다. 얼마나 동일한지 코드로 직접 확인해 보겠습니다. 예를 들어 ‘10.2.2 버트’에서 사용했던 데이터셋과 사전 훈련된 버트 모델 ‘bert-base-multilingual-cased’를 사용해서 한국어 임베딩을 확인해 보겠습니다.