XX. NLP_Model

2020년 봄 학기

2020년 2월 29일

https://www.github.com/KU-BIG/<폴더 만들면 넣기>

이번 강의에서는 자연어 처리에 사용되는 언어 모델들에 대해 설명한다. 자연어 처리에 사용되는 언어 모델들은 단어 시퀀스(또는 문장)에 확률을 할당하는 모델이라고 볼 수 있다. 즉 가장 자연스러운 시퀀스나 문장을 찾아가는 모델이라고 볼 수 있고 보편적으로 사용되는 방법은 이전 단어들을 통해 가장 자연스러운 다음 단어를 예측하도록 하는 것이다. 언어 모델에는 통계적 방법을 활용한 모델과 인공 신경망을 활용한 모델이 존재하며 현재는 인공 신경망을 활용한 모델이 주로 활용된다. 이 장에서는 위 두 종류 모델들의 예들인 N-gram, RNN, LSTM, Seq2seq, Transformer, Attention Mechanism에 대해서 다룬다.

1. 통계적 언어 모델(Statistical Language Model)

1.1 조건부 확률

통계적 언어 모델에서 활용되는 조건부 확률을 간단히 정리해보자. 조건부 확률이란 어떤 사건 A가 발생했을 때, 사건 B가 발생할 확률을 나타낸 것으로 이를 기호로 표현하면 P(B|A)이다. 이 때, P(B|A)와 P(A), P(A,B)사이에는 다음과 같은 관계가 성립한다.

$$P(B|A) = P(A,B)/P(A) \quad 또는 \quad P(A,B) = P(A)P(B|A)$$

위의 내용을 더 많은 확률에 대해 일반화해보면 다음과 같이 표현할 수 있다.

$$P(x_1,x_2,x_3...x_n)=P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)...P(x_n|x_1...x_{n-1})$$

이제 이 내용을 문장에 적용하여 통계적 언어 모델을 알아보도록 하자.

1.2 통계적 언어 모델 방법

앞에서 말한 것처럼 언어 모델은 이전의 단어들을 가지고 다음에 올 단어를 예측하는 모델이라고 볼 수 있다. 예를 들면 'I want to play _____'라는 문장에서 앞의 주어진 단어들인 'I', 'want', 'to', 'play'를 가지고 빈칸에 들어갈 가장 적절한 말을 찾는 것이다. 이 적절한 말을 찾을 때, 통계적 언어 모델에서는 조건부 확률을 활용한다. 즉, 앞의 단어들이 주어졌다는 가정 하에 다음 단어로 나올 수 있는 가장 높은 확률을 가진 단어를 선택한다.

빈칸의 단어를 X라고 했을 때, 통계적 언어 모델에서는 가능한 모든 X에 대하여 P(I want to play X)를 계산해보고 이 중 가장 높은 확률의 X를 선택하는 것이다. 이를 조건부 확률을 통해 수식으로 풀어보면 다음과 같다.

$$P(I\ want\ to\ play\ X)= P(I)\times P(want|I)\times P(to|I\ want)\times P(play|I\ want\ to)\times P(X|I\ want\ to\ play)$$

그렇다면 오른쪽 수식에 나와있는 조건부확률은 어떤 방법을 통해 구하는 것일까? 그 답은 카운트에 기반한 확률 계산이다.

1.3 카운트 기반 확률 계산

위에서 표현한 빈칸의 단어 후보들 중 하나로 'basketball'이 있다고 가정하자. 이 때, 'basketball'에 대한 확률을 구하기 위해 필요한 P(basketball | I want to play)는 이 모델이 학습되는 말뭉치(코퍼스)에서 이 같은 경우가 몇 번 나오는지 카운팅을 통해 계산된다. 만약 말뭉치에서 I want to play가 1000번 나왔고 이 중 다음 단어로 basketball이 500번 나왔다면 P(basketball | I want to play) = 500/1000 = 0.5가 된다. 이와 같은 방식으로 오른쪽 수식의 조건부 확률을 모두 계산할 수 있고 최종적으로 확률이 가장 높은 단어를 선택해 문장을 완성시키는 것이 통계적 언어 모델의 방법이다.

카운트 기반 확률 계산은 말뭉치에서 나오는 시퀀스를 카운팅해 확률 계산을 하는 것이므로 실생활에서 사용되는 언어처럼 자연스러운 모델이 나오기 위해서는 엄청난 양의 말뭉치가 필요하다. 실생활에서는 많이 사용되지만 만약 말뭉치에 그 시퀀스가 존재하지 않는다면 이 시퀀스가 모델에서 나올 수 있는 확률은 0이 되기 때문이다. 또한 문장의 길이가 길어진다면 이와 같은 확률을 계산하기 위해서 복잡하고 많은 수의 조건부 확률을 계산해야 하는데 이 때 충분한 데이터를 관측하지 못하여 언어를 정확히 모델링하지 못하는 희소 문제(sparsity problem)또한 발생할 수 있다. 이와 같은 문제를 완화하기 위해 N-gram과 여러가지 일반화 기법이 존재하나 본 강의에서는 N-gram만을 다루기로 한다. 하지만 이와 같은 방법을 통해서도 근본적으로 문제를 해결하지는 못하기 때문에 최근 언어 모델의 트렌드는 통계적 언어 모델보다는 인공 신경망 언어 모델이 주를 이루고 있다.

1.4 N-gram 언어 모델

N-gram 언어 모델도 본질적으로는 카운트 기반 확률 계산 방식이라고 볼 수 있다. N-gram 모델이 앞의 기본적인 모델과 다른 점은 빈칸 앞의 모든 단어들을 고려하는 것이 아니라 빈칸 앞 일정 개수의 단어만을 고려하여 빈칸 단어를 유추해낸다는 점이다. N-gram에서 N이 바로 몇

개의 단어를 고려할 것인가를 나타내는 것인데 예를 들어보면 만약 3-gram 모델이 있다면 이 모델은 빈칸을 포함한 3개의 단어를 고려한다고 볼 수 있다. (위 예에서는 I want를 제외한 to play만을 고려)

이와 같은 방식을 사용하면 긴 문장 내에서도 빈칸의 단어를 도출해 낼 수 있는 장점이 있지만 여기에는 한 가지 가정이 필요하다. 만약 위의 3-gram을 사용한다고 하면 P(basketball | I want to play) ≈ P(basketball | to play)라고 볼 수 있어야한다. 이와 같은 가정을 어느 정도 만족시킬 것인지 결정하는 것이 필요하고 이는 최종적으로 우리가 N을 어떤 숫자로 정할 것인가 고려해야 하는 문제가 발생한다.

N을 선택해야 하는 문제 말고도 N-gram은 본질적으로 카운트 기반의 확률 계산이기 때문에 희소문제가 존재할 수밖에 없고 N-1개의 단어 이외에는 고려하지 않기 때문에 만약 그 앞에 빈칸의 단어와 큰 연관 있는 단어가 존재한다면 잘못된 확률 계산을 할 수 있는 위험이 있다. 이와 같은 문제들 때문에 최근에는 통계적 언어 모델보다는 인공 신경망 기반의 언어 모델이 각광받고 있다. 이제 인공 신경망 기반 언어 모델에 대해 알아보도록 하자.

2. 인공 신경망 활용 언어 모델(Neural Network Based Language Model).

2.1 인공 신경망 언어 모델의 특징

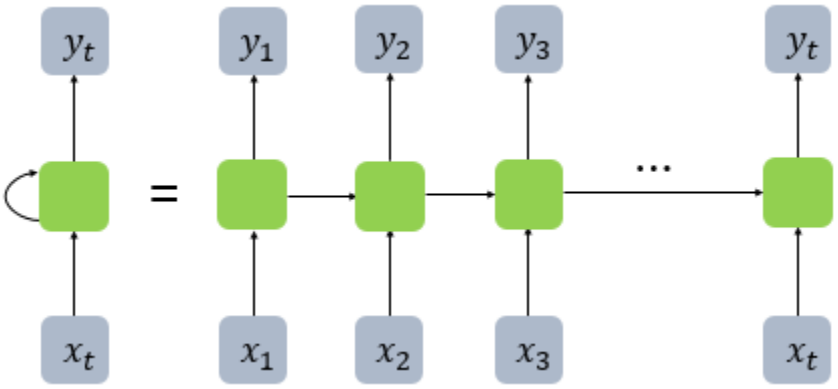
언어가 제대로 전달되기 위해서는 문장 내에서의 단어 간의 순서와 위치가 중요한 의미를 가진다. 이를 고려하기 위해 인공 신경망 활용 언어 모델에서는 인공 신경망을 time step에 따라 구성하여 단어들의 순서를 고려해주고 time step이 없는 경우(예를 들면 어텐션 메커니즘)에는 추가적인 계산 작업을 통하여 단어 별로 문장 내에서의 위치 벡터를 곱해주어 문장 내에서의

단어 위치를 고려해준다. 이제부터 인공 신경망을 활용한 언어 모델들의 일종인 RNN, LSTM, Seq2seq, 어텐션 메커니즘, 트랜스포머 등을 살펴보기로 하자.

2.2 RNN(Recurrent Neural Network)

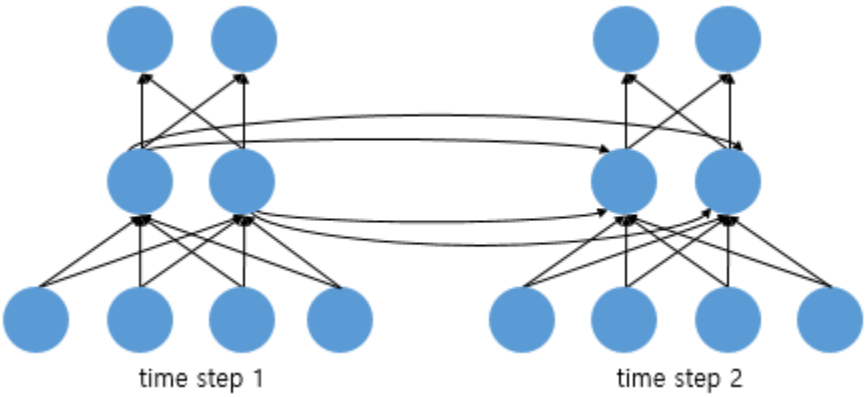
RNN은 언어 모델에 있어서 가장 기본적인 시퀀스(Sequence) 모델이라고 볼 수 있다. 시퀀스 모델이란 입력과 출력을 시퀀스 단위로 처리하는 모델인데, 언어에선 이 시퀀스가 중요한 요소 중 하나이기 때문에 언어모델에서 주로 시퀀스 모델이 활용된다. 예를 들어보면 문장 번역에서 입력은 번역하고자 하는 문장의 단어 시퀀스가 될 것이고 출력은 번역된 단어들의 시퀀스로 구성될 것인데 이 번역기 모델에서 RNN 등의 시퀀스 모델이 활용될 수 있다.

보통 처음 배우게 되는 일반적인 신경망을 살펴보면 은닉층에서 나온 값(또는 활성화 함수까지 지난 값)은 출력층 방향으로만 향했던 것을 떠올릴 수 있을 것이다. 하지만 RNN에서의 은닉층 노드의 결과값은 출력층 방향으로도 보내지고 다시 은닉층 노드의 다음 계산의 입력으로도 보내지는 특징을 가지고 있다.



출처 : https://wikidocs.net/book/2155

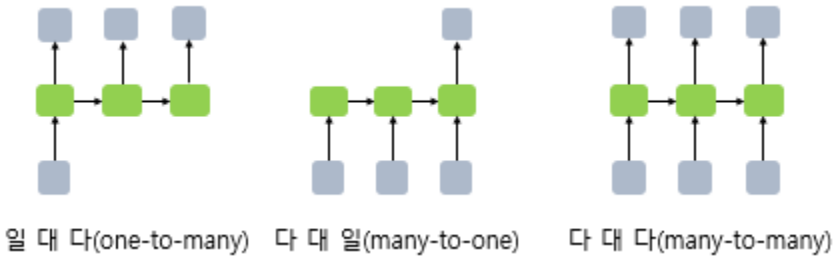
앞선 내용을 그림으로 표현하면 위의 그림과 같이 표현될 수 있다. 그림에서 x는 입력층의 입력 벡터, y는 출력층의 출력 벡터이다. 또한 가운데의 초록색 네모 박스를 은닉층(hidden layer)으로 볼 수 있고 x와 y의 아래 첨자는 각각의 시점(time step)을 나타낸다. 그림에서 알 수 있듯이 어느 한 시점 t-1에서 은닉층의 결과값은 그 시점의 출력값인 y_{t-1}로도 보내지고 다음 시점인 t 은닉층의 입력값으로도 보내지는 것을 알 수 있다. 즉, t번째 시점에서의 은닉층은 t-1번째 은닉층의 결과값과 t번째 시점의 입력 벡터를 입력값으로 받아 계산을 진행한다. 이를 통해 RNN에서 t번째 출력값은 그 전 시점에서의 값들까지 고려한 결과임을 알 수 있고 단어들 간의 순서가 중요한 언어 분야에서 활용되기 적합한 모델임을 알 수 있다. 이를 우리가 흔히 알고 있는 신경망인 피드 포워드 신경망과의 차이점을 알아보기 위해 그림을 변형시켜보면 다음과 같다.



출처 : https://wikidocs.net/book/2155

하나의 time step 별로 RNN에서는 하나의 입력층, 은닉층, 출력층이 존재하며 이들은 벡터 상태로 존재한다 (신경망에서는 뉴런이라고도 부름). 위의 그림에서 한 은닉층에서의 결과값은 그 시점의 출력층으로도 향하지만 다음 시점의 은닉층의 입력 값으로도 활용되는 것을 알 수 있고 이 점이 바로 피드 포워드 신경망과의 차이점이다.

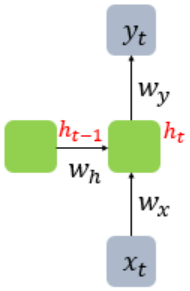
RNN은 입력과 출력의 길이를 다르게 설정할 수 있어서 다양한 분야에서 다양한 종류로 활용될 수 있다. RNN에서 나올 수 있는 다양한 유형들을 나열한 그림은 다음과 같다.



출처 : https://wikidocs.net/book/2155

각 RNN 셀에 들어갈 수 있는 값들은 문장, 단어, 형태소 등 다양하고 이 중 가장 보편적으로 이용되는 것은 단어 벡터이다. 위의 그림에서 다 대 일(many-to-one) 유형의 경우에는 하나의 출력 벡터만을 결과값으로 가지므로 실생활에서 활용되는 예들로는 스팸 메일인지 아닌지를 구분하는 스팸 메일 분류기, 제품 리뷰가 긍정인지 부정인지 판단해주는 감성 분류 등이 있다. 다 대 다(many-to-many)의 경우에는 우리가 문장을 입력하면 그 입력 문장에 따라 대답을 해주는 챗봇, 번역하고자 하는 문장을 입력하면 번역된 문장을 출력해주는 번역기 모델이 실생활에서 활용된다고 볼 수 있다.

RNN에서 은닉층과 출력층의 계산 과정을 수식으로 표현해보면 다음과 같다.



출처 : https://wikidocs.net/book/2155

위의 그림은 t 시점에서의 계산 과정을 보여주는 것이다. 이 때, h는 은닉 상태값이고 x는 입력 벡터, y는 출력 벡터라고 볼 수 있다. t 시점에서의 은닉 상태값 h_t는 이전 상태의 은닉 상태 h_{t-1}과 t 시점에서의 입력 벡터 x_t를 가지고 계산되고 이 h_t를 통해 출력벡터 y_t를 계산하게 된다. 이를 식으로 표현하면 다음과 같다.

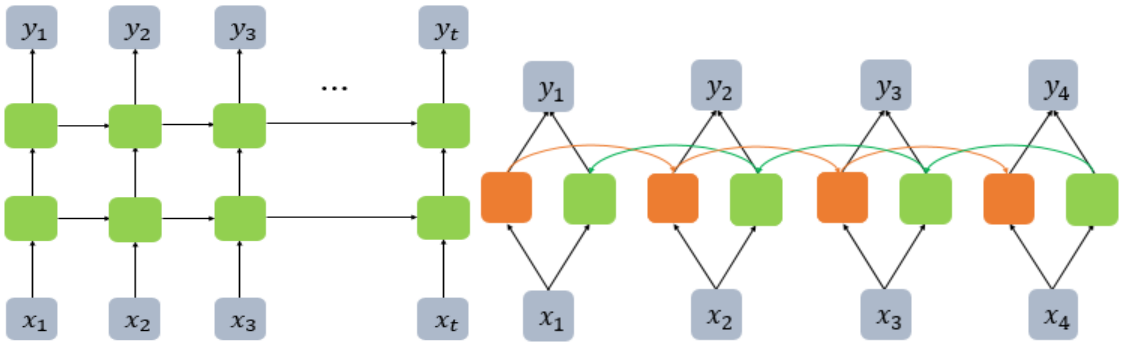
은닉층 : $h_t=\tanh(W_x x_t+W_h h_{t-1}+b)$

출력층 : $y_t=f(W_y h_t+b)$

단, f는 비선형 활성화 함수 중 하나

입력벡터와 이전 시점의 은닉벡터는 각각의 가중치 벡터 W가 곱해지고 편향 b와 더해져 하나의 벡터를 가지게 되며 이 벡터가 활성화 함수(위의 경우에서는 하이퍼볼릭탄젠트)를 지나 은닉 상태값 h_t를 가지게 된다. 출력층의 경우에는 이 은닉층의 결과값 h_t와 편향이 다시 활성화 함수 f를 지나게 되고 최종 출력 벡터를 가지게 되는 방식으로 RNN이 구성된다.

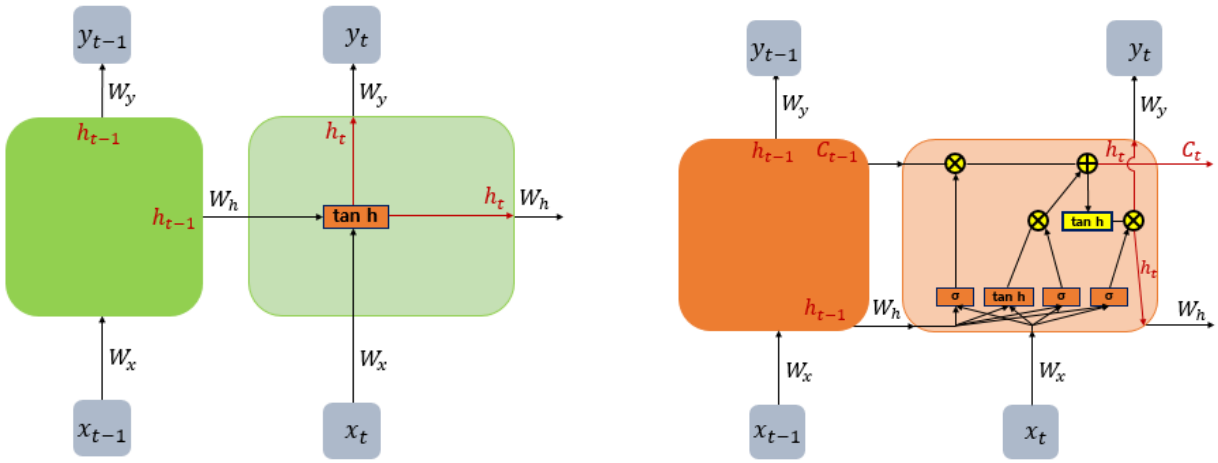
앞서 본 RNN은 기본적인 RNN 모델이라고 볼 수 있고 여기서 좀 더 변형된 RNN 모델들로는 하나의 은닉층이 아닌 다수의 은닉층을 가지고 있는 깊은 순환 신경망, 앞의 시퀀스만을 고려해 단어를 생각해내는 것이 아니라 이후 시점의 데이터까지 고려하는 양방향 RNN 등이 있다.



출처 : https://wikidocs.net/book/2155

2.3 LSTM

위에서 소개한 RNN 모델은 가장 단순한 형태의 RNN(바닐라 RNN)이었고 기존 RNN의 문제점을 보완하기 위해 다양한 변형 모델들이 나오게 되었는데 LSTM 또한 RNN의 변형 모델 중 하나이다. 이전 내용을 통해 바닐라 RNN은 출력 결과가 이전의 계산 결과에 의존한다는 것을 알 수 있었다. 하지만 time step이 길어질수록 앞의 계산 결과가 점점 희석되어 전달이 잘 되지 않는 경향이 발생하였고 이는 만약 앞 부분에 중요한 정보가 존재한다면 이를 제대로 반영하지 못한 출력값이 나타나게 되는 문제점을 야기하였다. 이를 보완하기 위해 LSTM이 등장하였다.



RNN의 모습을 표현한 그림(좌)와 LSTM의 모습을 표현한 그림(우)

출처 : https://wikidocs.net/book/2155

왼쪽 그림은 바닐라 RNN을 도식화한 것이고 오른쪽 그림은 LSTM을 도식화한 것이다. 바닐라 RNN은 단순히 이전 은닉층의 결과값과 입력벡터를 가중치 벡터와 곱하고 활성화 함수를 적용하는 방식만을 사용하였다면 LSTM은 은닉층의 메모리 셀에 입력 게이트, 망각 게이트, 출력 게이트를 추가하여 불필요한 기억을 지우고, 기억해야할 것들을 정하는 작업이 추가되었다. 이에 따라 바닐라 RNN에 비해 계산식이 조금 복잡해졌고 셀 상태(cell state)라는 새로운 값 C를

추가하였다. 셀 상태는 은닉 상태처럼 이전 시점의 셀 상태가 다음 시점의 셀 상태를 구하기 위한 입력으로 사용된다. LSTM에서 은닉 상태값과 셀 상태값을 구하기 위해서 3개의 게이트를 사용한다. 각 게이트들은 삭제 게이트, 입력 게이트, 출력 게이트라고 불리우며 이 게이트에는 공통적으로 시그모이드 함수가 존재한다. 시그모이드 함수를 거치면 0과 1사이의 값이 나오게 되는데 이 값을 통해 게이트들을 조절하고 어느 정보를 오래 가지고 있을 것인지 결정하는 작업을 진행하게 된다. 각 게이트들을 알아보기에 앞서 용어 정리를 다음과 같이 하려고 한다.

- 이하 식에서 σ는 시그모이드 함수를 의미한다.

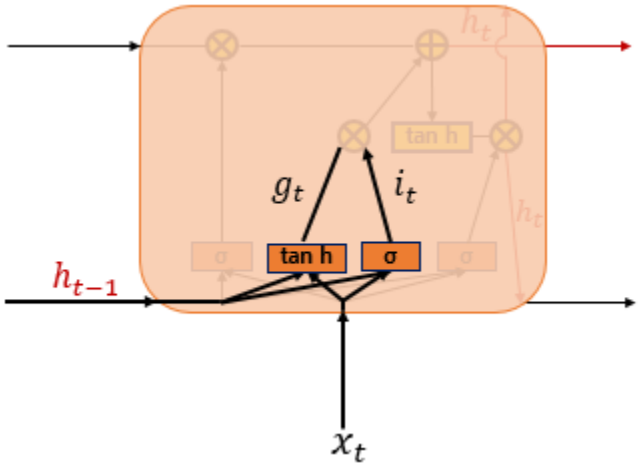
- 이하 식에서 tanh는 하이퍼볼릭탄젠트 함수를 의미한다.

- W_{xi}, W_{xg}, W_{xf}, W_{xo}는 x_t와 함께 각 게이트에서 사용되는 4개의 가중치이다.

- W_{hi}, W_{hg}, W_{hf}, W_{ho}는 h_{t-1}와 함께 각 게이트에서 사용되는 4개의 가중치이다.

- b_i, b_g, b_f, b_o는 각 게이트에서 사용되는 4개의 편향이다.

1) 입력 게이트



출처 : https://wikidocs.net/book/2155

$i_t = \sigma(W_{xi}x_t+W_{hi}h_t-1+b_i)$

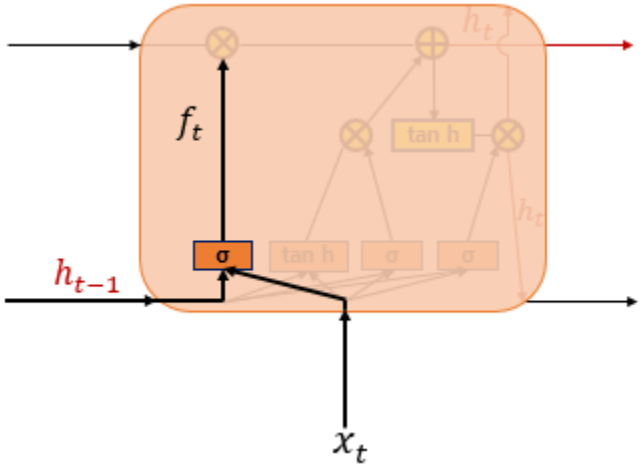
$g_t = \tanh(W_{xg}x_t+W_{hg}h_{t-1}+b_g)$

입력 게이트는 현재 정보를 기억하기 위한 게이트이다. 우선 현재 시점 t의 입력 벡터 x값과 입력 게이트로 이어지는 가중치 W_{xi}를 곱하고 t-1시점의 은닉 상태에 입력 게이트로 이어지는 가중치 W_{hi}를 곱한 뒤, 이들을 더하여 시그모이드 함수를 지나게 한다. 이를 i_t라고 하자.

그리고 현재 시점 t의 x값과 입력 게이트로 이어지는 가중치 W_{xg}를 곱한 값과 이전 시점 t-1의 은닉 상태가 입력 게이트로 이어지는 가중치 W_{hg}를 곱한 값을 더하여 하이퍼볼릭탄젠트 함수를 지나게 하는데 이를 g_t라고 한다.

시그모이드 함수를 지나 0과 1 사이의 값과 하이퍼볼릭탄젠트 함수를 지나 -1과 1사이의 값 두 개(i_t, g_t)가 나오게 된다. 이 두 개의 값을 가지고 이후 이번에 선택된 기억할 정보의 양을 정하게 된다.

2) 삭제 게이트

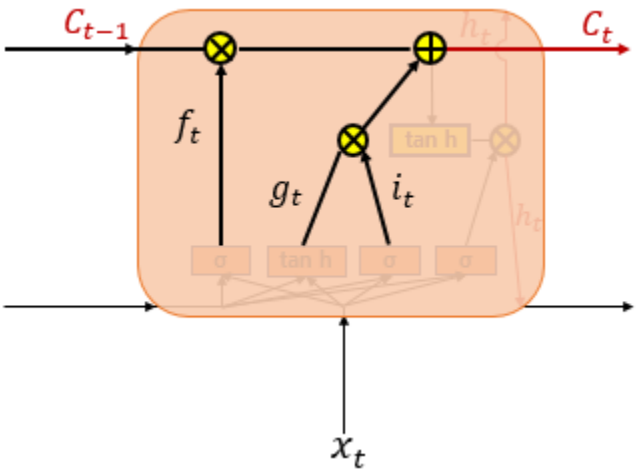


출처 : https://wikidocs.net/book/2155

$f_t = \sigma(W_{xf}x_t+W_{hf}h_{t-1}+b_f)$

삭제 게이트는 기억을 삭제하기 위한 게이트이다. 현재 시점 t의 x값과 이전 시점 t-1의 은닉 상태가 시그모이드 함수를 지나게 되면 0과 1 사이의 값이 나오게 되는데, 이 값이 곧 삭제 과정을 거친 정보의 양이라고 볼 수 있다. 0에 가까울수록 정보가 많이 삭제된 것이고 1에 가까울수록 정보를 온전히 기억한 것이다. 이후 이를 가지고 셀 상태를 구하게 된다.

3) 셀 상태(장기 상태)



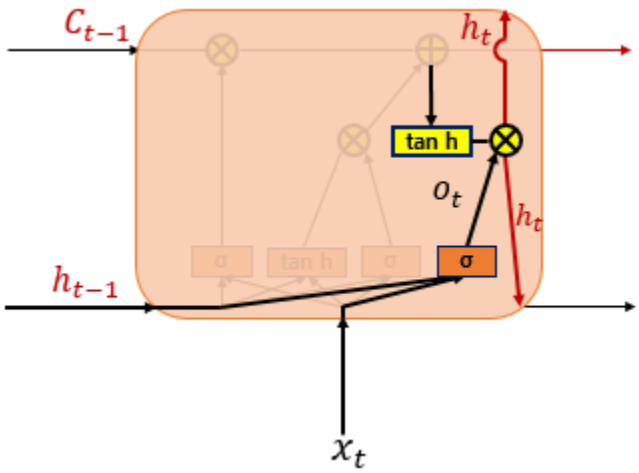
출처 : https://wikidocs.net/book/2155

$C_t = f_t \circ C_{t-1} + i_t \circ g_t$

장기 상태를 나타내는 C_t는 이전 시점의 장기 상태를 나타내는 C_t와 이번(t시점)에 계산된 기억값 g_t를 가지고 계산된다. f_t와 i_t는 앞서 말했던 것처럼 0과 1사이의 값을 가지는 벡터들인데 이들을 C_{t-1}과 g_t에 원소별 곱을 해주어 f_t와 i_t의 원소가 0에 가까운 값이면 기존 데이터를 삭제, 1에 가까운 값을 가지면 그 정보를 유지시켜주는 역할을 해준다. 즉 f_t가 1에 가까운 원소를 많이

가지게 된다면 과거 셀 대부분의 정보를 현재 셀 정보에 가지고 오고 i_t가 0인 원소를 많이 가진다면 현재 정보 대부분을 반영하지 않은 장기 상태 C를 가지게 된다.

4) 출력 게이트와 은닉 상태



출처 : https://wikidocs.net/book/2155

$o_t = \sigma(W_{xo}x_t+W_{ho}h_{t-1}+b_o)$

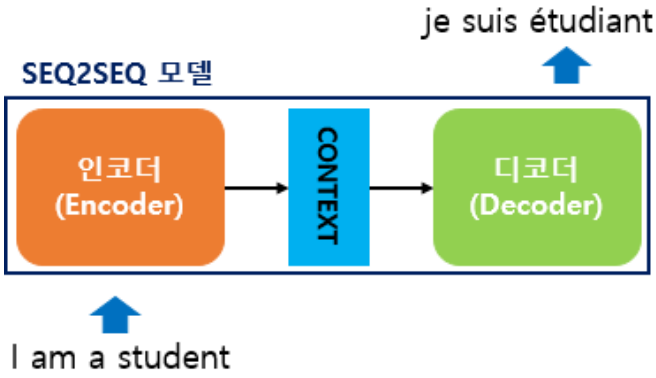
$h_t = o_t \circ \tanh(c_t)$

이 값은 앞서 보았던 바닐라 RNN에서의 은닉 상태 계산에 활성화 함수를 지난 장기 상태 C를 고려하여 출력해주는 것이다. 이를 통해 바닐라 RNN보다 과거 정보를 더욱 잘 반영할 수 있는 결과를 얻을 수 있고 이는 LSTM이 시퀀스가 긴 문장에서 큰 효과를 발휘할 수 있는 결과를 이끈다.

2.4 Seq2seq

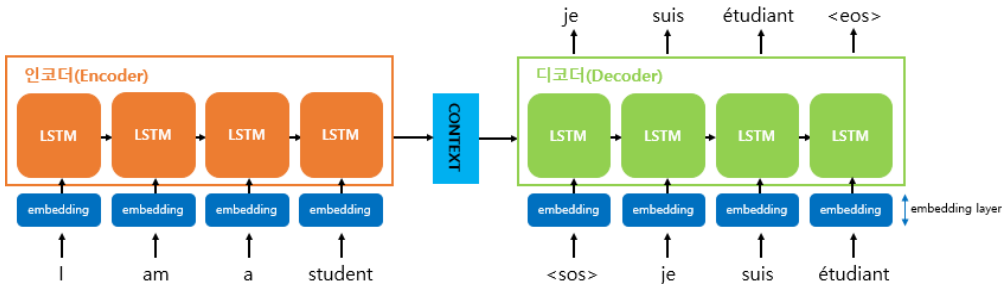
시퀀스-투-시퀀스(이하 Seq2seq)는 입력된 시퀀스로부터 다른 도메인의 시퀀스를 출력하는 분야에서 사용되는 모델이다. 예를 들어 챗봇 같은 경우 질문 도메인과 응답 도메인으로

구성하여 훈련을 진행하게 되면 질문 도메인 속 질문 시퀀스에 따라 응답 도메인 속 응답 시퀀스가 도출되므로 seq2seq 모델이 적절하다고 볼 수 있다. 이 경우 외에 번역기에도 기존 문장과 번역된 문장 각각의 도메인이 존재하므로 seq2seq 모델이 잘 활용될 수 있다.



출처 : https://wikidocs.net/book/2155

위 그림은 seq2seq 모델을 도식화한 것이다. 번역기에 사용되는 seq2seq 모델을 나타낸 것이며 입력값에 'I am a student'가 들어가면 'je suis étudiant'라는 번역된 프랑스 문장을 출력해준다. Seq2seq 모델은 두 개의 아키텍처(인코더, 디코더)로 구성된다. 인코더에서는 입력 문장의 모든 단어들을 순차적으로 입력 받은 뒤 최종적으로 전체적인 문장의 의미를 담은 하나의 벡터(컨텍스트 벡터)를 출력한다. 이후 디코더에서는 출력된 컨텍스트 벡터를 가지고 목적에 맞는 단어를 순차적으로 하나씩 출력해주고 최종적으로 우리가 원하는 출력값을 구할 수 있다.



출처 : https://wikidocs.net/book/2155

Seq2seq 구조를 좀 더 자세히 살펴보기 위해 위의 그림을 살펴보면 인코더와 디코더는 여러 개의 RNN셀들로 구성되어 있는 것을 알 수 있다. 그림에서는 바닐라 RNN보다 성능이 좋은 LSTM 모델로 셀을 구성하였다. 인코더에서 토큰화를 통해 단어로 쪼개진 단어 토큰이 각 시점에 맞게 입력값으로 들어가고 모든 입력값이 들어간 후 인코더 RNN 셀의 마지막 시점의 은닉 상태를 디코더의 첫 RNN 셀로 넘겨준다. 이를 앞에서 봤듯이 컨텍스트 벡터라고 하고 이 컨텍스트 벡터를 디코더 RNN 셀의 첫번째 은닉 상태로 사용하게 된다.

디코더에서는 초기 입력 벡터를 시작을 의미하는 심볼(위의 그림에서는 <sos>)에 해당하는 벡터로 하며 컨텍스트 벡터를 이전 시점의 은닉 상태 벡터로 보고 계산이 진행된다. 이후 계산 결과가 출력 벡터로 나오게 되면 이를 다음 시점의 입력 벡터로 보고 계산을 계속 진행한다. 이 과정을 반복하여 최종적으로 끝을 의미하는 심볼(위 그림에서는 <eos>)가 출력값으로 나오게 되면 최종적으로 출력 과정이 끝나게 된다.

이 때, 만약 seq2seq 모델을 훈련 중인 상황이라면 위의 디코더 부분에서 설명했던 이전 시점의 출력값을 현재 시점의 입력값으로 넣지 않는다. 만약 출력값이 훈련 과정 중이라 엉뚱하게 나왔다면 이후 값들도 엉뚱한 값이 나올 가능성이 높고 가중치 벡터들을 제대로 조정할 수 없기 때문이다. 이 때문에 훈련 중에서는 교사 강요라는 방식을 쓰게 된다. 교사 강요란 각 시점에서 입력값을 이전 시점의 출력값으로 가지는 것이 아니라 실제 정답 시퀀스의 t번째 단어를 입력값으로 삼아 엉뚱한 방향으로 학습되는 것을 막아주는 방법이다.

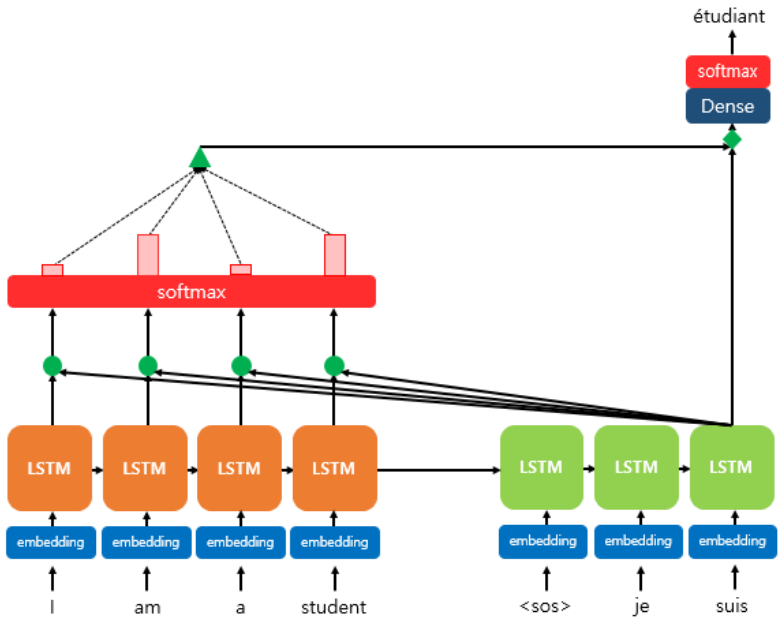
2.5 어텐션 메커니즘(Attention Mechanism)

Seq2seq 모델에는 크게 두 가지의 문제가 존재한다. 첫 번째로는 고정된 크기의 벡터(컨텍스트 벡터)에 모든 정보를 압축하려고 하니 정보 손실의 문제가 발생한다. 두 번째로는 RNN의

고질적인 문제인 기울기 손실 문제이다. 이로 인해 seq2seq는 입력 문장이 길면 품질이 떨어지는 문제점이 발생하게 되고 이를 보정하기 위해 앞으로 설명할 어텐션 기법이 등장하였다.

어텐션의 기본 아이디어는 디코더 부분에서 출력 단어를 예측하는 매 시점마다 인코더의 전체 입력 문장을 다시 한 번 참고하는 점이다. 하지만 이 때 모든 부분을 균일하게 보는 것은 아니고 해당 시점에서 예측해야 할 단어와 더 연관이 있는 부분을 더욱 유심히 보고 출력을 진행하는 방식으로 구성된다.

어텐션은 다양한 종류가 존재하나 본 강의에서는 가장 단순한 방식의 어텐션인 닷-프로덕트 어텐션을 가지고 어텐션 메커니즘을 알아보고자 한다.

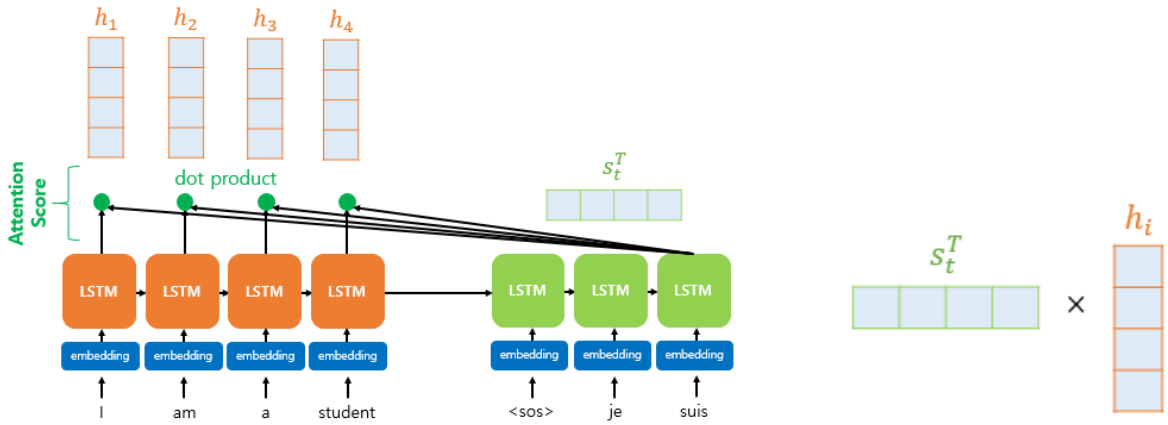


출처 : https://wikidocs.net/book/2155

위 그림은 디코더에서 세 번째 시퀀스에 해당하는 단어를 예측하는 과정을 나타낸 것이다. 위의 그림에서 주의 깊게 봐야할 부분은 인코더 부분 위에 있는 softmax 함수이다. Softmax 함수는 0과 1사이의 양의 값을 원소로 갖는 벡터를 변환시키고 원소의 총합은 1이 되도록 해준다. 그

결과 softmax 함수 위에 표현되어 있는 네모 모양의 bar들은 현재 단어를 예측하는 과정에서 도움이 많이 되는(유심히 봐야하는) 입력단어 정도를 나타낸 것이라고 볼 수 있다. 최종적으로 이 모델에서는 높은 값의 원소를 가지는 시퀀스 입력값에 대하여 더 유심히 살펴보고 출력을 진행하여 기존 seq2seq보다 더 나은 결과값을 얻을 수 있을 것이다.

Softmax 함수를 거친 결과값을 도출해 내는 과정은 다음과 같다. 우선 각 입력 시퀀스의 시점별로 어텐션 스코어를 구한다. 어텐션 스코어는 현재 시점을 t라 했을 때 t 시점에서의 디코더 은닉 상태와 인코더의 각 시점 별 은닉 상태가 얼마나 유사한지를 판단해주는 스코어이다. 즉 이 스코어가 높다면 현재 시점의 은닉 상태가 인코더의 해당 시점 은닉 상태와 비슷하므로 그 부분의 입력값을 더 주의 깊게 봐야 한다는 것을 의미한다. 닷-프로덕트 어텐션에서는 디코더의 은닉상태와 인코더의 은닉상태를 내적하여 그 값을 도출해낸다.



출처 : https://wikidocs.net/book/2155

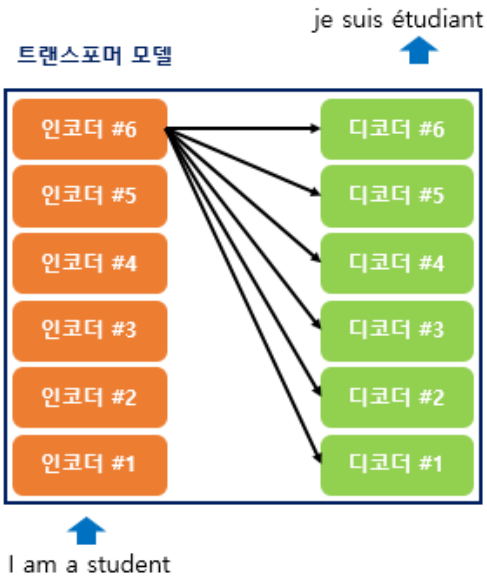
이렇게 인코더의 각 시점 별 어텐션 스코어를 구하면 하나의 벡터로 묶을 수 있을 것이고 그 벡터를 e_t라고 하자. 이 벡터를 sofrtmax함수에 넣게 되면 앞서 말한 결과값이 출력될 것이다. 이제 이 결과값을 인코더의 은닉상태와 곱한다면 최종 어텐션 값을 구할 수 있다. 이 최종

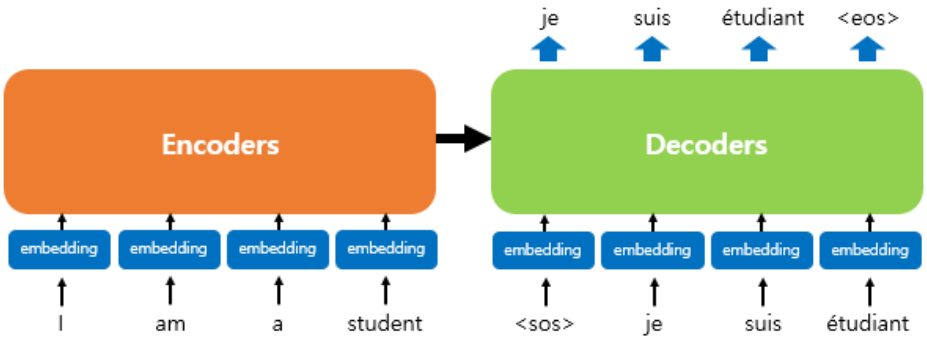
어텐션값은 디코더의 t 시점에서의 은닉상태와 결합하여 새로운 하나의 벡터로 만들어지고 이 새로운 벡터가 출력을 위한 연산의 입력으로 사용된다.

2.6 트랜스포머(Transformer)

트랜스포머는 기존의 seq2seq 구조인 인코더-디코더를 따르면서도, RNN셀은 사용하지 않고 어텐션 방법으로만 인코더-디코더 구조를 구현한 모델이다. RNN을 쓰지 않기 때문에 학습 속도가 매우 빠르다는 장점이 있으며 성능도 RNN보다 우수하다고 한다.

트랜스포머는 RNN을 이용하지 않지만 인코더-디코더 구조를 유지하고 있다. 또한 seq2seq 구조와 다른 점은 인코더와 디코더라는 단위가 한 개가 아닌 N개 존재할 수 있다는 점이다. 이전 seq2seq 구조에서는 RNN을 활용했기 때문에 t개의 time step이 존재하였다면 트랜스포머는 RNN을 이용하지 않기 때문에 t개의 time step 대신에 N개의 인코더와 디코더로 모델을 구성한다.

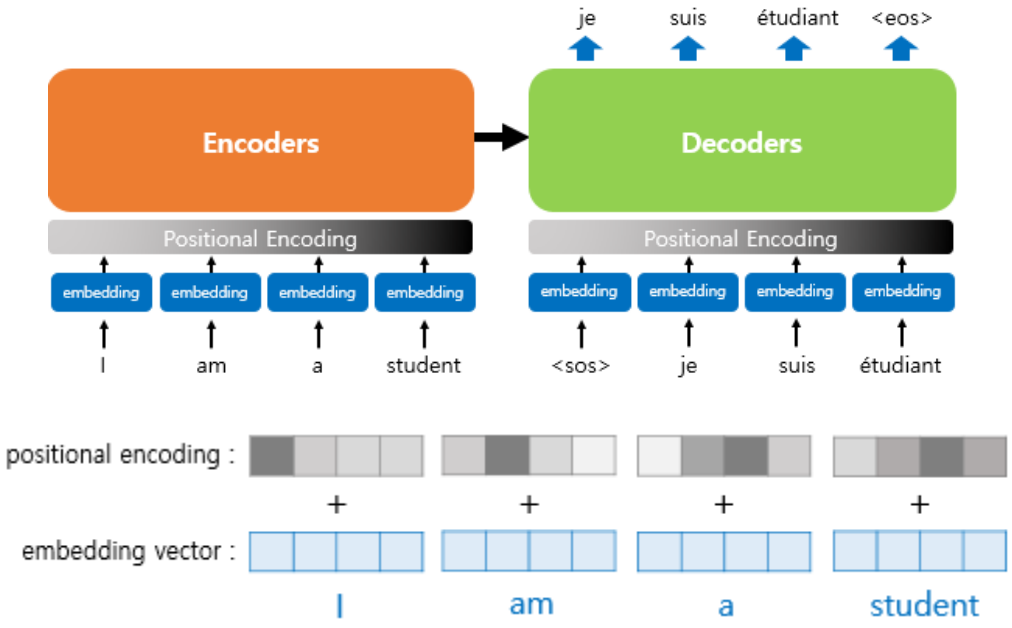




출처 : https://wikidocs.net/book/2155

위의 그림은 트랜스포머 모델의 예를 나타내는 것으로 인코더와 디코더가 각각 6개로 구성되어 있고 이들이 모여 하나의 encoders, decoders 로 작동하는 것을 나타낸다. 이 encoders, decoders는 기존 seq2seq처럼 encoders 값을 토대로 하여 decoders에서 <sos>를 시작 입력값으로 하여 <eos>가 출력될 때까지 작업을 수행하는 방식으로 진행된다. 이제부터는 기존 seq2seq모델과의 차이점을 보다 자세하게 알아보자.

트랜스포머는 RNN모델을 사용하지 않기 때문에 RNN을 사용한다면 얻을 수 있는 각 단어의 위치 정보를 별도로 벡터에 추가해주는 작업이 필요하다. 이를 포지셔널 인코딩(positional encoding)이라고 한다. 포지셔널 인코딩은 기존 임베딩된 벡터들이 입력값으로 들어갈 때 포지셔널 인코딩값을 임베딩 벡터에 더해줌으로써 작업이 진행된다. 이를 그림으로 표현해보면 다음 그림과 같다.



출처 : https://wikidocs.net/book/2155

포지셔널 인코딩 값들은 임베딩 벡터 내의 차원 인덱스에 따라 cos, sin함수로 구성되어 있고 그 수식은 다음과 같다.

$PE_{(pos,\ 2i)} = \sin(pos/10000^{2i/d})$

$PE_{(pos,\ 2i+1)} = \cos(pos/10000^{2i/d})$

이렇게 포지셔널 인코딩까지 마친 벡터는 이제 인코더와 디코더의 입력 벡터로 들어가게 된다. 트랜스포머에는 아까 언급한 바와 같이 N개의 인코더와 디코더가 존재한다. 각 인코더는 2개의 서브층인 멀티-헤드 어텐션과 피드 포워드 신경망으로 나뉘어진다. 또한 각 디코더는 2개의 멀티-헤드 어텐션 층과 하나의 피드 포워드 신경망으로 구성되어 총 3개의 층으로 구성되어 있다.

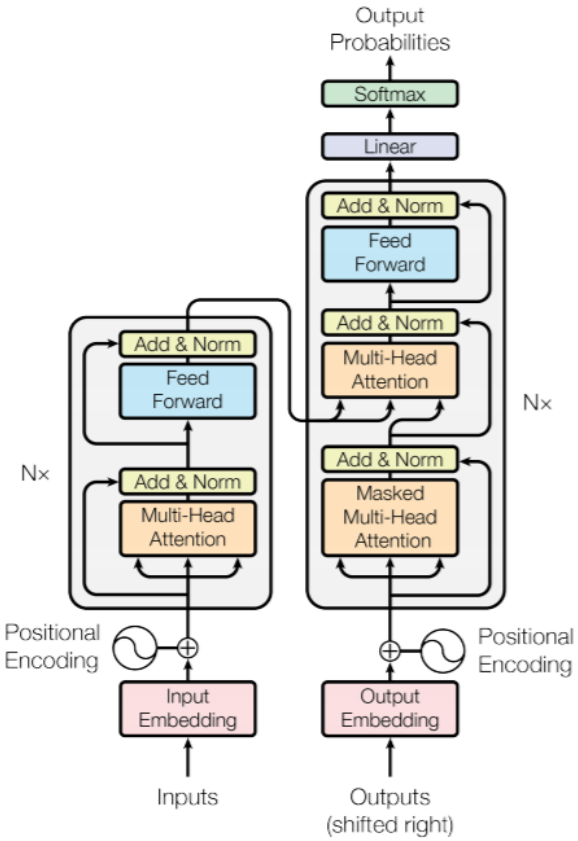


Figure 1: The Transformer - model architecture.

출처 : https://medium.com/platfarm/어텐션-메커니즘과-transfomer-self-attention-842498fd322

멀티 헤드 어텐션은 셀프 어텐션이라는 어텐션 기법을 활용한다. 앞에서 다뤘던 어텐션 기법은 t 시점에서의 디코더 셀 은닉 상태와 모든 시점의 인코더 셀 은닉 상태들을 활용하여 어텐션 값을 도출하였다면 셀프 어텐션에서는 입력 문장의 모든 단어 벡터들만을 가지고 어텐션 값을 구한다는 차이점이 있다. 셀프 어텐션은 입력 문장 내의 단어들끼리 유사도를 구함으로써 문장 내의 모호한 표현(대명사 등)을 자기 문장 속 단어와 비교해 그 의미를 파악할 수 있다는 장점이 있다. 구체적인 방법은 복잡하고 여기서 세세하게 다루기는 어려운 것 같아 관심 있는 사람들은 참고 문헌 속 사이트에 들어가서 살펴보면 좋을 것 같다. 멀티 헤드라는 의미는 셀프 어텐션을 한

번만 진행하는 것이 아니라 벡터들을 세분화하고 이들을 따로 각각 계산하여 셀프 어텐션 값을 여러 번 구하고 이들을 종합하는 작업을 의미한다. 이를 통해 단일 어텐션보다 더 다양한 관점에서 문맥을 파악할 수 있고 보다 나은 결과값을 가질 수 있게 된다.

앞서 멀티 헤드 어텐션의 결과로 나온 행렬(벡터들의 모임)을 가지고 피드 포워드 신경망에서는 가중치 행렬과 활성화 함수 ReLU를 활용하여 결과값을 도출해내고 이 결과값은 다음 인코더로 들어가 위와 같은 연산을 반복한다.

디코더에서는 2개의 멀티 헤드 어텐션 층과 1개의 피드 포워드 신경망으로 구성된다. 피드 포워드 신경망은 인코더의 내용과 같으므로 생략하고 2개의 멀티 헤드 어텐션 층에 대해 알아보도록 하자.

디코더의 첫 번째 멀티 헤드 어텐션층은 Masked Multi-head Attention 으로 ‘Masked’라는 용어가 추가되었다. 이 ‘masked’라는 용어가 의미하는 바는 다음과 같다. 입력층에서는 모든 단어 시퀀스가 한 번에 주어져 기준 단어의 전후 모든 단어에 대해 어텐션값을 구할 수 있지만 디코더 부분에서는 이전 단어들만을 고려할 수 있으므로 어텐션 값을 구할 때 기준 시점 이후의 단어 시퀀스 부분에 대해서는 마스크를 씌워 고려하지 않도록 해준다. 이외의 부분에 대해서는 인코더에서 진행했던 셀프 어텐션 구하는 방법을 디코더 입력 벡터에 대해 동일하게 진행하는 형식이므로 자세한 설명은 생략하도록 하겠다.

디코더의 두 번째 멀티 헤드 어텐션층은 디코더와 인코더의 셀프 어텐션 값에 대하여 어텐션 값을 구하는 것으로 셀프 어텐션이라고 볼 수 없고 기존 어텐션 메커니즘에서 보았던 어텐션 방법과 동일하다고 볼 수 있다. 즉, 인코더와 디코더의 어텐션 층을 정리해보면 다음과 같다.

인코더의 Multi-head Attention: 인코더 입력 벡터에 대해 Self-Attention이 일어나는 부분

디코더의 Masked Multi-head Attention: 디코더의 입력 벡터 Self-Attention이 일어나는 부분

디코더의 Multi-head Attention: 인코더와 디코더의 Attention이 일어나는 부분

인코더와 디코더의 사이사이에는 Add & Norm이라는 단계가 존재하는 것을 알 수 있는데 이는 층 정규화(layer normalization)와 잔차 연결(residual connection)을 통해 vanishing gradient 문제와 gradient가 exploding하는 문제를 완화시켜준다.

※ 본 자료는 딥 러닝을 이용한 자연어 처리 입문(https://wikidocs.net/book/2155)을 주로 참고하여 제작되었습니다.