어텐션 구조와 메커니즘 조사 분석

201904213 심성빈
201904201 김찬민

메커니즘

Seq2seq 모델은 기존의 RNN 모델과 같이, 고정된 길이의 벡터를 입출력하는 구조이다. 그러나 이 구조는 고정된 길이의 벡터 압축으로 시퀀스의 길이가 길어질 수록 정보 소실이 발생한다

➔ 이러한 한계를 극복하기 위해 어텐션 메커니즘을 도입

➔ 어텐션 메커니즘은 입력 문장의 모든 단어를 동일한 가중치로 취급하지 않고, 출력 문장에서 특정 위치에 대응하는 입력 단어들에 더 많은 가중치를 부여하는 것이다.

➔ 입력과 출력의 길이가 다른 경우에도 모델이 더욱 정확하고 유연하게 작동될 수 있다.

Background

Transformer에서 Muti-Head Attention을 통해 상수 시간의 계산만으로 가능

->

Self – attention

- 자신에게 수행하는 어텐션 기법으로, 다일 시퀀스 안에서 서로 다른 위치에 있는 의존성을 찾아냄
- 메커니즘 독해, 추상적 요약, 텍스트 포함, 학습 과제, 독립적인 문장 표현을 포함한 다양한 task에서 성공적으로 사용

End-to-end memory network

- 시퀀스가 배열된 recurrence보다 recurrent attention mechanism 기반
- 간단한 언어 질문 답변 및 언어 모델링 작업에서 좋은 성능을 보임

장거리 의존성 문제

- 대부분의 자연어처리는 Encoder-decoder 구조를 가지는 recurrent model인 RNN,CNN등이 주로 사용

➔ 자연스럽게 문장의 순차적인 특성이 유지되지만 먼 거리에 있는 의존성을

파악하기에 취약하다는 단점이 존재

➔ RNN의 경우, 시퀀스 길이가 길어질수록 정보 압축 문제 존재

➔ CNN의 경우, 합성곱의 필터 크기를 넘어서는 문맥은 알아내기 어렵다.

- 반면 Transformer는 순환업이 어텐션 메커니즘만을 이용해 의존성을 찾을 수 있음(Self-attention에만 의존하는 최초 모델)

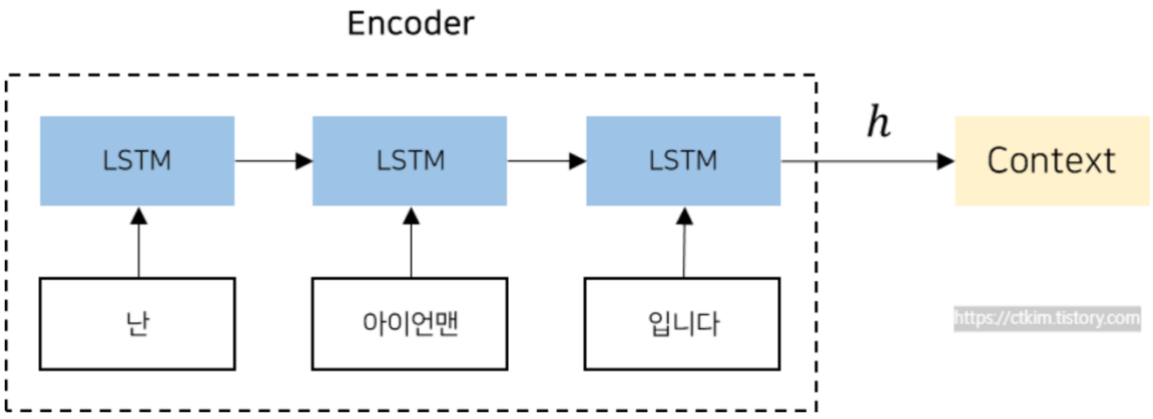
모델 구조

1. seq2seq 구조

- 한 시퀀스를 다른 시퀀스로 변환하는 작업을 수행하는 딥러닝 모델
- 인코더와 디코더라는 모듈을 가지고 있다.

#인코더

- 인코더는 일반적으로 RNN, LSTM, GRU등의 순환 신경망 구조를 사용하여 입력 시퀀스를 고정 길이의 벡터로 변환하는 역할



➔ 입력 시퀀스의 각 단어를 순차적으로 처리하면서 각 단계에서 hidden state를 업데이트하고, 최종적으로 전체 입력 시퀀스를 대표하는 고정 길이의 벡터를 생성

➔ 그림과 같이 문장을 인코딩하여 고정 길이 벡터 h를 생성 이후 디코더에 전달

➔ 인코더가 생성한 벡터 h는 LSTM의 마지막 hidden state

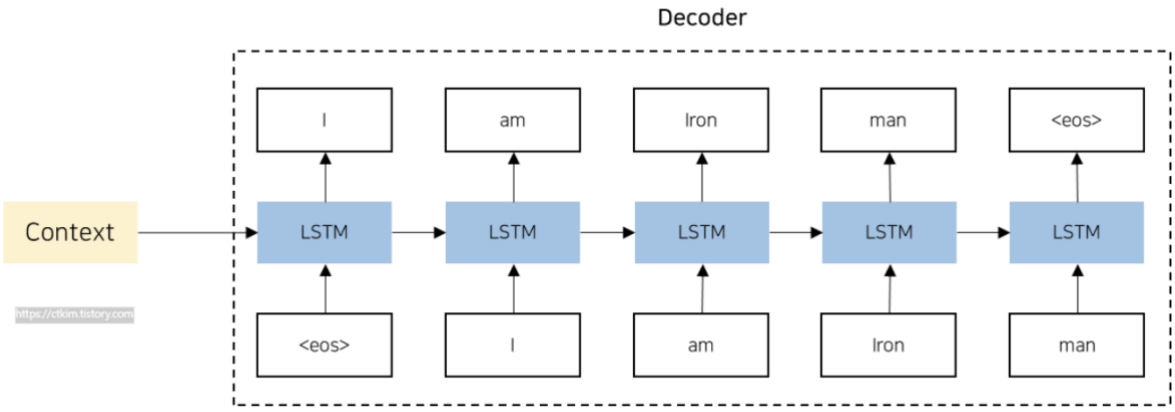
➔ 인코더가 인코딩한다는 것은 임의의 길이의 문장을 고정 길이 벡터로 변환하는 작업

#디코더

- 디코더는 인코더의 출력인 고정 길이의 벡터를 기반으로 원하는 출력 시

퀀스를 생성하는 역할 수행

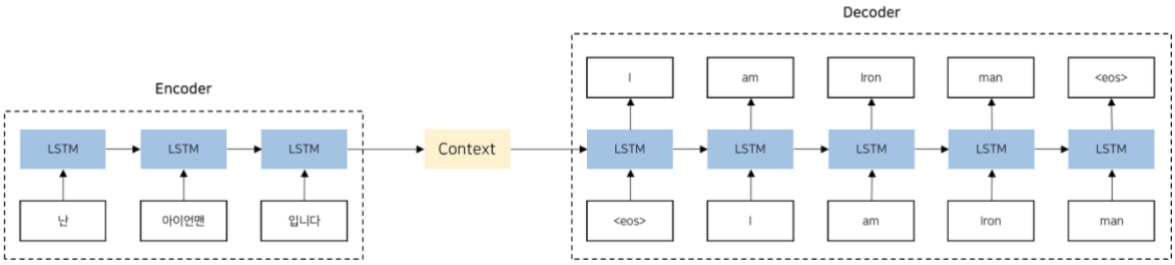
- 기존 순환 신경망 구조와 다른 점은 벡터 h를 입력으로 받는다
- 디코더 예측은 일반적으로 소프트맥스 활성화 함수를 통해 확률 분포로 변환되며, 가장 확률이 높은 단어가 선택된다 -> 이 과정을 반복하여 최종적으로 출력 시퀀스가 생성된다



➔ <eos>기호는 디코더에게 문장 생성 시작과 종료를 알리는 신호로 사용

➔ 구분 기호는 <go>, <start>, <s>등을 이용하기도 한다

#seq2seq 구조



- 인코더와 디코더 역할을 하는 두개의 LSTM으로 구성
- 인코더와 디코더 사이에는 컨텍스트 벡터가 존재하며. 이를 통해 인코딩된 정보가 디코더로 전달
- 역전파 과정에서는 컨텍스트 벡터를 통해 기울기가 디코더에서 인코더로 전달되어 모델이 학습

#seq2seq 한계

- 고정된 길이의 벡터로 압축으로 인해 입력 시퀀스의 길이가 길어질 수록 정보의 손실이 발생

- RNN의 고질적인 문제인 경사 소실 또는 폭발 문제가 존재

➔ 극복하기 위해 어텐션 메커니즘 방법 선택

2. Transformer

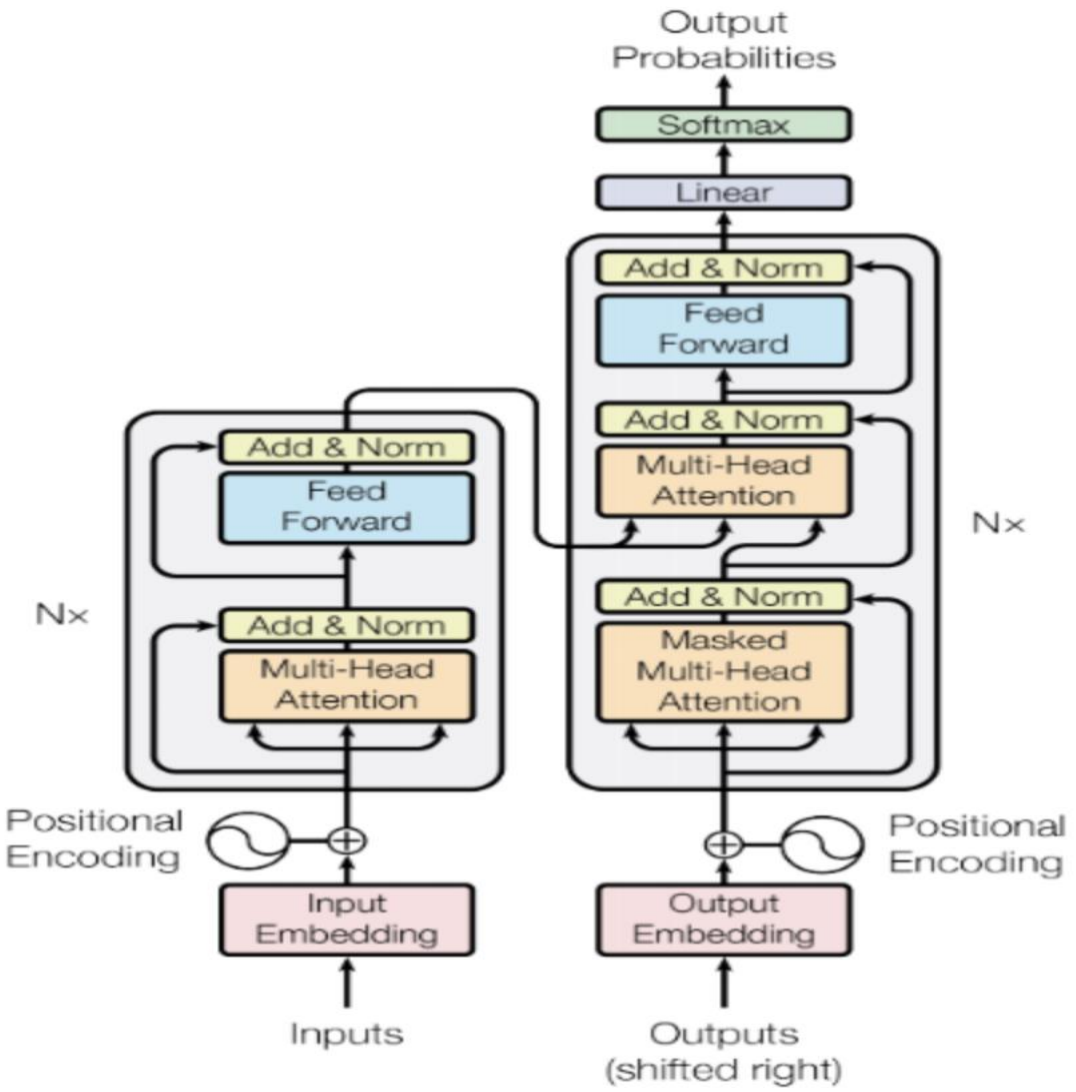


Figure 1: The Transformer - model architecture.

Embedding 단계

- Data를 임의의 N-dimension data오 만들어주는 단계
- 자연어 처리에서는 단어 등의 단위를 표현하는 N-dimension 데이터로 만들어주는 단계

Positional 인코딩

- 시퀀스내에서 해당 정보의 위치 정보와 Embedding된 데이터를 Sin함수와 Cos 함수 형태로 만들어 다음 레이어의 input으로 전달

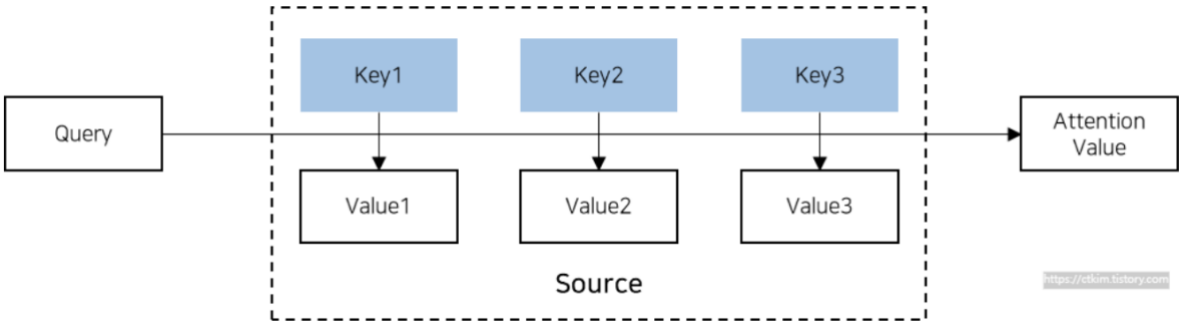
➔ Embedding을 할 때 위치 정보까지 함께 넣어주자
➔ Transformer는 해당 input의 위치를 반영한 pre-processing을 할 수 있게 된다
➔ 이렇게 정리된 input data들은 인코더로 들어간다
➔ Encoder는 Multi-head Self Attention / Add & Normalize / Position-wise FFNN라는 모듈들로 구성
➔ Multi-head Self Attention은 앞에서 정리한 Attention과 비교했을 때 살짝 차이가 있다
➔ Attention이 인코더의 Hidden state와 디코더의 Hidden state를 연산해서 Attention score 값들을 계산했다면, Self-Attention은 Encoder로 들어간 벡터와 인코더로 들어간 모든 벡터를 연산해서 Attention score 값을 구한다
➔ 이 부분이 매우 헷갈렸지만, 이런 Self-Attention을 통해서 각 input data들의 유사도를 추정할 수 있다
➔ 어텐션 함수의 Query (Q), Key (K), Value (V)를 의미하는 Weight 행렬과 ' scaled Dot-Product Attention'방식으로 연산하여 Attention Value Matrix (a)를 만들어낸다
➔ 이 과정에서 d_model개의 차원을 num_heads로 나눠지는 갯수만큼의 그룹으로 묶어서 d_model/num_head 개의 Attention value matrix를 뽑아낸다
➔ 직관적으로는 여러 head를 가지는 Attention value matrix를 뽑아냄으로써 다양한 관점으로 연산을 할 수 있게 만드는 것이다
➔ 이렇게 여러 개로 나온 Attention value matrix는 다시 d_model 차원으로 합쳐진다.

Attention Function

- 입력 시퀀스의 각 단어들에 대한 가중치를 계산하는 함수
- 각 단어의 중요도를 측정하여 출력 결과에 반영
- 세가지 요소로 구성

➔ 첫 번째는 Query(현재 출력 단어를 나타내는 벡터(t시점의 디코더 셀에서의 은닉 상태))
➔ 두 번째는 Key
➔ 세 번째는 Value
➔ Key와 Value는 입력 시퀀스의 각 단어에 대응하는 벡터(모든 시점의 인코

더 셀의 은닉 상태들)



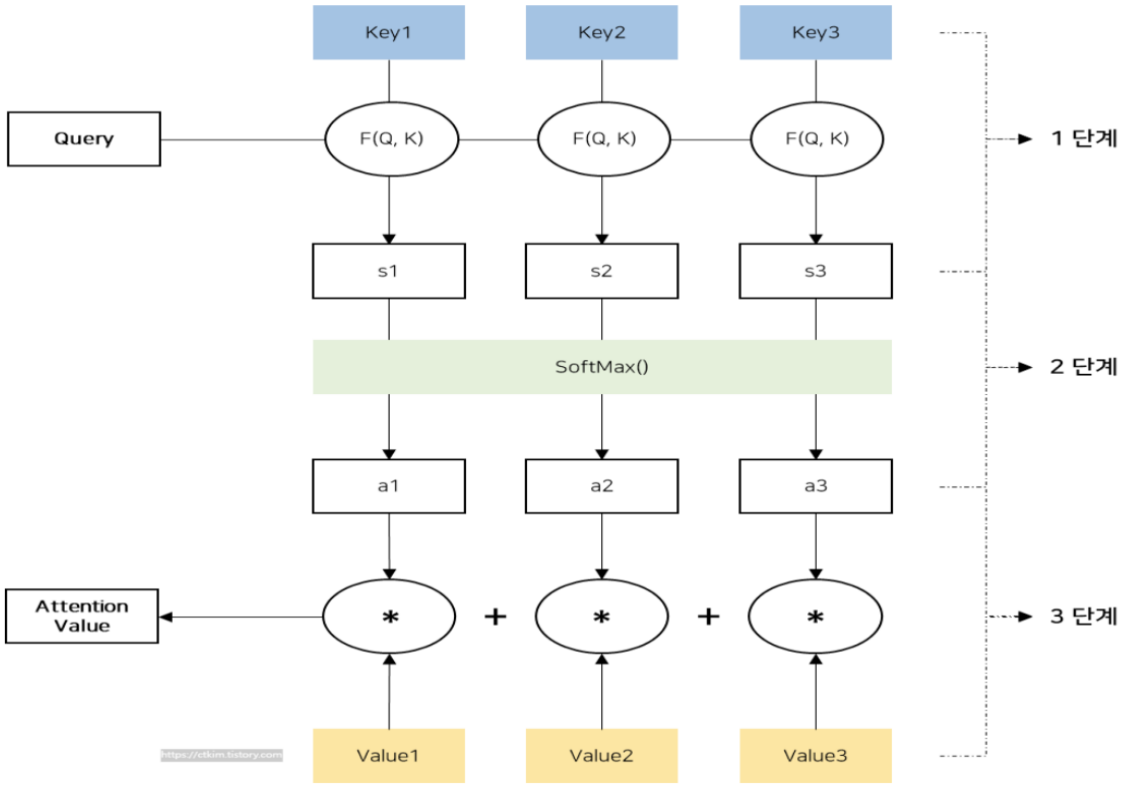
➔ Attention Function은 Query벡터와 Key벡터 간의 유사도 측정 (내적, 외적 등 다양한 방법으로 계산 될 수 있다)

➔ 가중치는 Value 벡터와 곱해져 최종 출력 벡터 생성

Scaled Dot-Product Attention

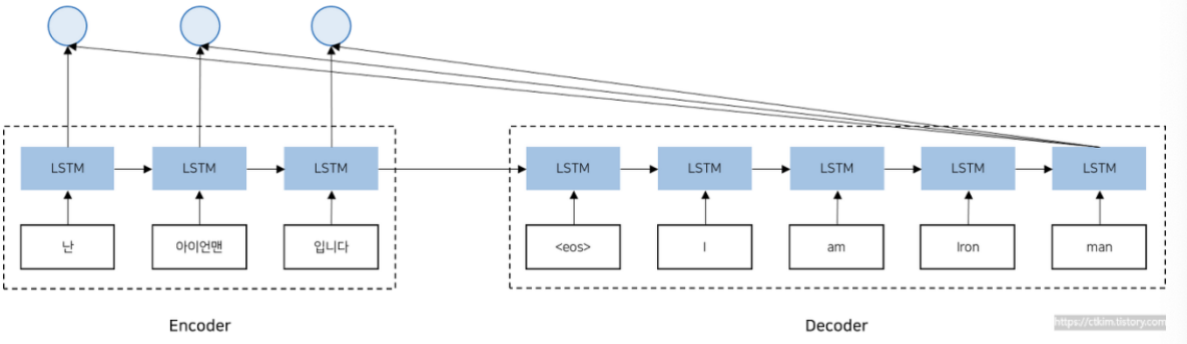
- 어텐션 매커니즘이 입력 시퀀스 내에서 특정 토큰에 집중할 수 있도록 도와준다
- Softmax 함수를 통해 정규화를 어텐션 분포 조절, 가중치를 균등하게 분배할 수 있도록 도와준다.

➔ Key 벡터의 차원 수가 증가할 수록, 내적 값이 커지기 때문에 softmax 함수 사용



1단계 : Attention Score 계산

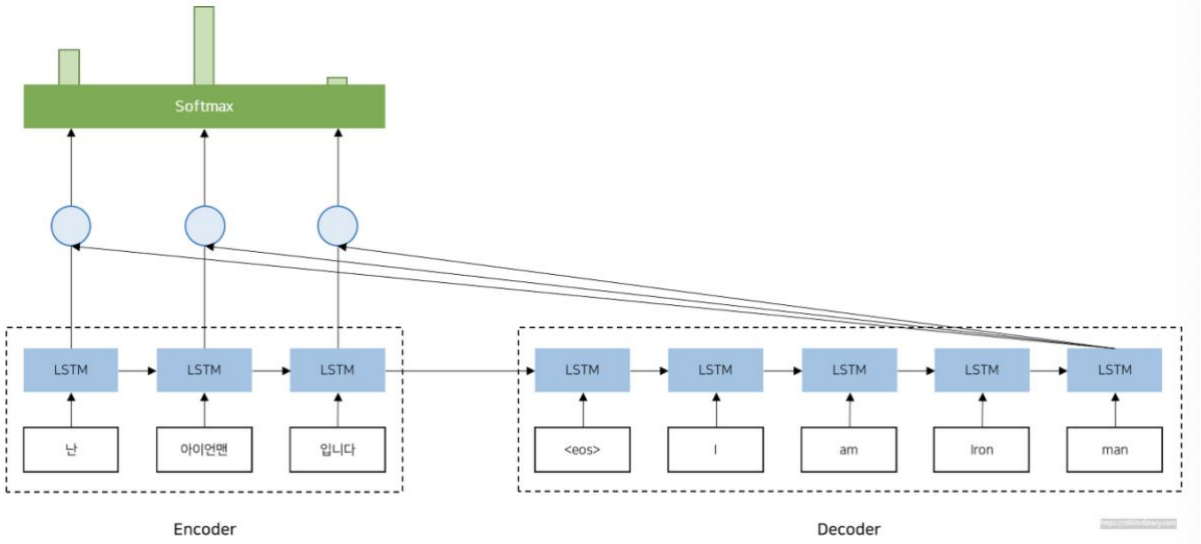
- 쿼리와 키 사이의 유사도를 나타내는 값
- 각 인코더 시점의 은닉 상태와 디코더의 t 시점의 은닉 상태 간의 어텐션 스코어가 계산



2단계 : Attention Distribution 계산

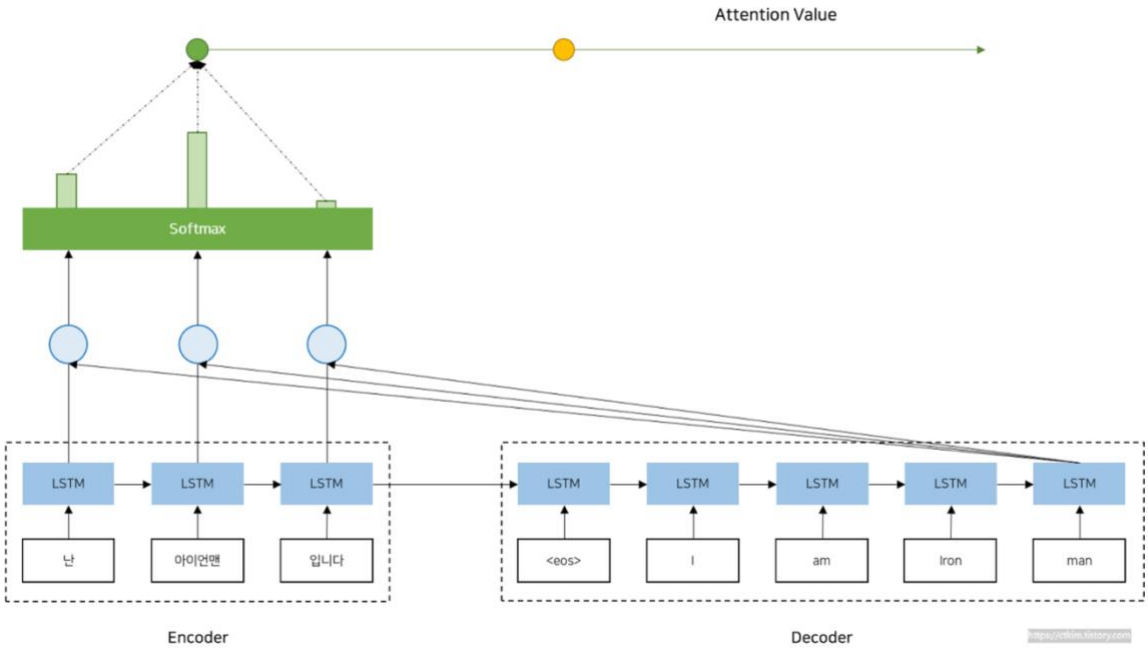
- 어텐션 스코어를 소프트맥스 함수에 적용하면, 각 인코더 시점의 은닉 상태에 대한 가중치를 얻을 수 있다
- 이 가중치는 디코더의 t시점에 인코더의 각 시점의 정보가 얼마나 중요한지를 나타낸다

- 소프트맥스 함수를 통해 얻은 값은 모든 인코더 시점에 대해 합이 1이되는 확률 분포를 가진다



3단계 : Attention Value 계산

- 어텐션 가중치와 인코더의 값 벡터를 곱한 후, 모든 인코더 시점에 대해 합산하여 어텐션 값을 구한다
- 여기서 값 벡터는 인코더의 각 시점의 은닉 상태를 의미
- 어텐션 값은 인코더의 정보를 디코더에 전달하는 역할을 하며, 가중치를 고려해 각 인코더 시점의 정보를 조합한 결과물



4단계 : Attention Value와 디코더의 은닉 상태 concatenate

- 어텐션 값과 디코더의 t시점의 은닉상태를 연결하여, 새로운 입력으로 사용

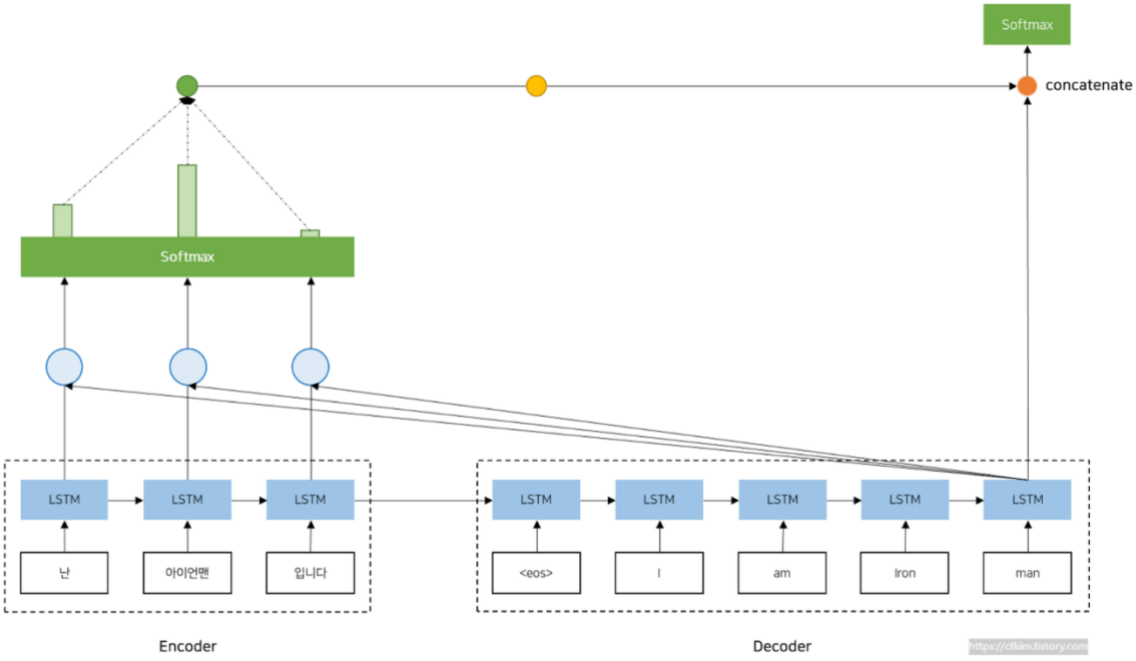
➔ 어텐션 메커니즘을 통해 인코더와 디코더가 상호 작용하며, 입력 문자의 정보를 공유

- 디코더의 현재 시점의 은닉 상태를 ht, 어텐션 메커니즘의 결과를 ct
- Concatenate한 후 출력을 생성하는 방식

➔ 두 벡터를 Concatenate : [ht : ct]

➔ 연결된 벡터를 적절한 크기의 출력 벡터로 변환 ot = f([ht ; ct])
(f는 디코더의 출력을 생성하기 위해 사용되는 활성화 함수)

➔ 출력 벡터를 소프트맥스 함수에 적용하여 최종 확률 분포를 얻음
Pt = softmax(ot)



Scaled Dot-Product Attention의 특징

1) 계산이 빠르다

➔ 내적 연산을 고속으로 처리할 수 있기 때문에 다른 어텐션 방법보다 계산 속도가 빠르다

2) 정규화를 위해 스케일링이 사용

➔ 내적 연산을 통해 계산된 유사도를 정규화하는데, 이때 내적 연산 결과의 차원 수를 나누어 스케일링을 수행

➔ 가중치의 분산을 조절하여 학습을 안정화시키고, 어텐션 메커니즘의 성능을 향상

3) 병렬 처리가 가능

➔ Scaled dot-Product Attention은 입력 시퀀스의 각 단어 간에 독립적으로 계산이 가능하므로, 병렬 처리에 용이하다

➔ 이를 통해 더 빠른 학습 속도를 달성

4) 다양한 입력 시퀀스 길이를 처리할 수 있다

➔ Scaled dot-Product Attention은 입력 시퀀스의 길이에 대해 제한이 없기 때문에, 다양한 길이의 입력 시퀀스를 처리할 수 있다.

다양한 종류의 어텐션

1) Dot-Product Attention

- 쿼리와 키의 내적을 사용하여 어텐션 스코어를 계산

$$Score(Q, K) = QK^T$$

2) Scaled Dot-Product Attention

- Dot-Product Attention의 변형으로, 내적 값을 키 벡터의 차원 수의 제곱근을 루트 d의 k로 나누어 스케일링
- 각 차원에 대한 영향력이 과도하게 커지는 것을 방지하고 학습의 안정성을 향상시킨다

$$Score(Q, K) = QK^T/\sqrt{dk}$$

3) Additive Attention

- 쿼리와 키를 결합한 후, 학습 가능한 가중치 행렬을 적용하여 어텐션 스코어를 계산하는 방식
- 활성화 함수를 적용한 결과를 가지고 어텐션 스코어를 구한다

$$Score(Q, K) = v^T * tanh(W1 * Q + W2 * K)$$

4) Multi-Head Attention

- 여러 개의 독립적인 어텐션을 병렬로 수행한 후, 결과를 결합하는 방식
- 각 어텐션 헤드는 서로 다른 가중치를 사용하여 입력에 대한 다양한 관점을 학습할 수 있다
- 일반적으로 Scaled Dot-Product Attention을 기반으로 한다

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h) * W^O$$

$$head_i = Attention(Q * W_i^Q, K * W_i^K, V * W_i^V)$$