<Attention Is All You Need 논문 리뷰>

- Keyword를 바탕으로 작동원리 설명 및 사용 이유 중심으로 리뷰 -

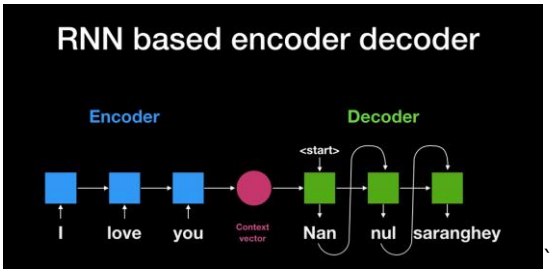
keyword: Masked attention, Self attention, Multi head attention, Positional encoding

- Abstract: Transformer의 차별점

■ 지금까지 sequence transduction model은 RNN 기반이었다. 그러나 Transformer는 RNN을 사용하지 않고 오직 attention mechanism만으로 translation task에서 높은 성능과 훈련 시간 감소의 결과를 얻어냈다.

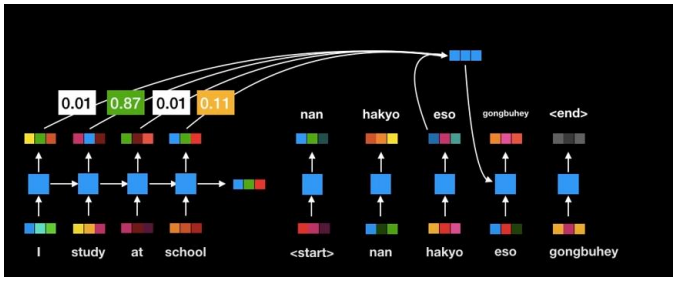
- Introduction

■ RNN / LSTM / GRU는 language modeling과 machine translation과 같은 문제를 해결하는데 성과를 보였고, recurrent language model과 encoder-decoder 구조로 진행되어왔다. 그러나 recurrent model은 input과 output 문장의 위치에 따라 순차적으로 계산을 하는데, 이는 병렬처리를 못하게 하고, 메모리 제약으로 인해 긴 문장을 처리하는 것이 어렵다는 문제가 있다.



* 전통적인 encoder/decoder 방식: 문맥 벡터(Context vector)가 고정적인 크기
-> 긴 문장일 경우 고정된 문맥 벡터에 저장하기가 어려움. 성능 떨어짐

■ Attention mechanism 등장: input 또는 output 문장의 거리와 상관없이 dependency를 모델링하게 함으로써 attention mechanism은 compelling sequence modeling과 transduction model에 필수적인 부분이 되었다.

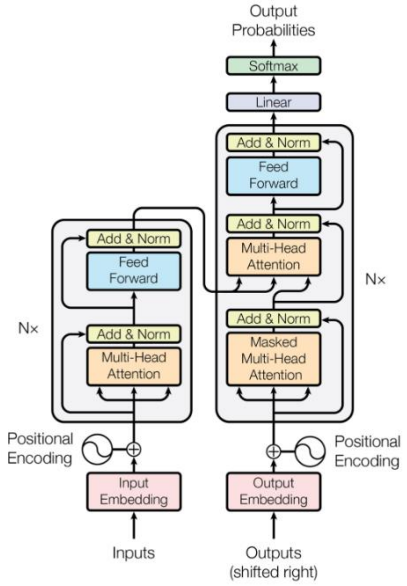


* context vector 대신 encoder의 모든 상태값을 사용함 -> 긴 문장의 번역 성능 향상

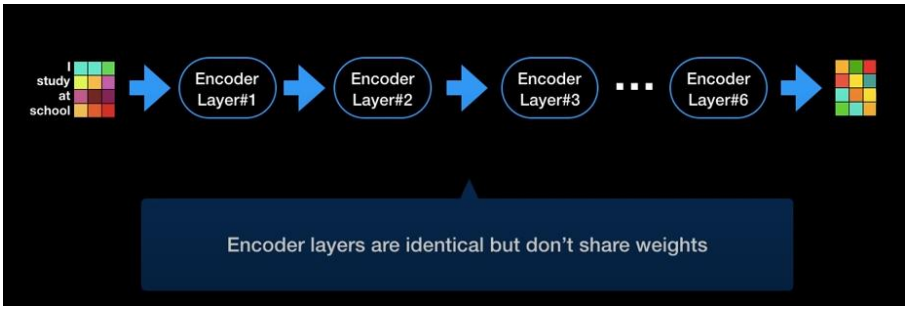
■ 기존까지는 attention 구조와 RNN이 결합되어 사용되어 왔는데, Transformer는 recurrence를 사용하지 않고 attention mechanism만을 사용해, 병렬처리가 가능하고 보다 적은 훈련 시간으로 높은 번역 성능을 얻을 수 있다.

- **Model Architecture**

■ Transformer는 stacked self-attention과 point-wise FC layer를 사용해서 구성되어 있다.



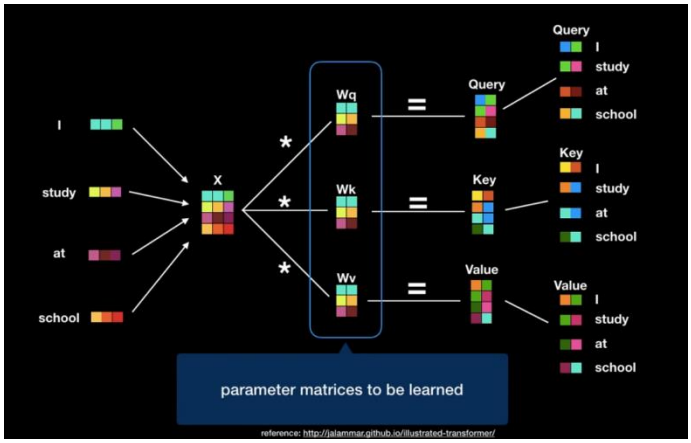
■ **Encoder**: 6개의 동일한 layer로 스택되어 있고, 각 layer는 multi-head self attention과 point-wise FC layer로 구성되어 있다. 또한 residual connection과 layer normalization 이 적용되어 있다.



■ **Decoder**: Encoder과 마찬가지로 6개의 동일한 layer로 스택되어 있다. 그러나 encoder와 다른 점은 encoder의 출력값에 multi-head attention을 수행하는 sub-layer가 추가되었다. 또한 decoder의 경우 self-attention에 masking을 추가하는데, **masked attention**은 decoder layer에서 지금까지 출력된 값들에 대해서만 attention 을 적용하기 위해서 붙여진 이름으로, 아직 출력되지 않은 미래의 단어에 attention 을 적용하는 것을 방지한다.

■ **Attention**: attention function의 output은 value에 대한 weighted sum으로 구해지고, 이 weight은 key에 대한 query의 compatibility function을 수행해 얻어진다.

* 추가 설명: query, key, value는 벡터의 형태로 Wq, Wk, Wv 행렬에 의해 각각 생성이 되는데, 행렬들은 단순히 weight matrix로서 딥러닝 모델 학습과정을 통해 최적화된다.



attention function의 경우, 현재의 단어(ex. I)는 query에 해당하고, 특정 단어(I, study, at, school 등)와의 상관관계를 구하고 싶을 때, 특정 단어의 key값과 곱하면, 그 값은 attention score(scalar형태)가 된다. score가 높을수록 연관성이 높다고 할 수 있다. 이후, attention score를 0~1까지의 확률값으로 바꾸기 위해 softmax를 적용한다. 본 논문에서는 key의 차원의 수가 늘어날수록 dot product 값이 증대되는 문제를 보완하기 위해서 softmax를 적용하기 전에, score에 key의 square root로 나눠서 계산하였다.

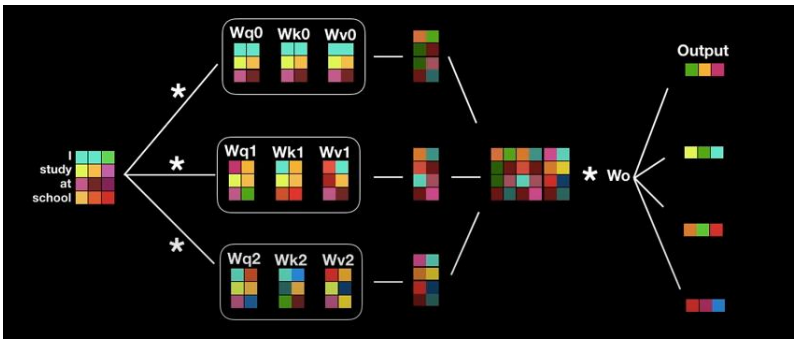
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

이 후, 각 확률값을 value와 곱해준다. 그 결과, 연관성이 거의 없는 단어는 희미해지게 되고, 최종적으로 모든 값을 더해주면 최종 벡터는 단순히 해당 단어가 아닌 '문장 속에서 해당 단어가 지닌 전체적인 의미를 지닌 벡터'로 간주된다.

* 참고로, key와 value는 사실상 같은 단어를 의미한다. 두 개를 구분하는 이유는, key값을 위한 vector와 value를 위한 vector를 따로 만들어서 사용하기 위해서이다. key를 통해서는 각 단어와 연관성의 확률을 계산하고, value는 그 확률을 사용해서 attention 값을 얻는 용도이다.

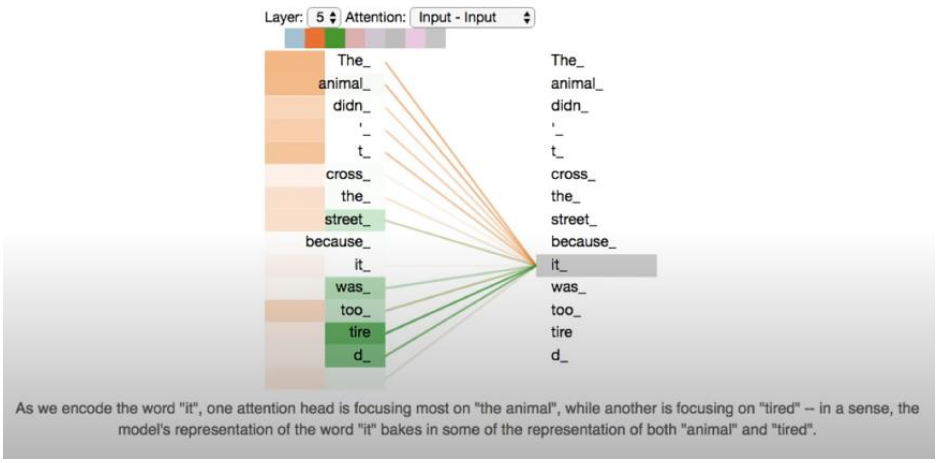
■ **Multi Head Attention**: 기존의 attention은 전체 dimension에 대해서 하나의 attention만 적용시켰다. 본 논문은 전체 dimension을 h로 나눠서 attention을 h번 적용하는 것이 더 효과적임을 발견했다. 각 query, key, value의 vector는 linear하게 h개로 project된다. 이후 각각 나눠서 attention을 시킨 후 만들어진 h개의 vector를 concat한다. 마지막으로 vector의 dimension을 맞춰 주기 위해 matrix를 곱한다.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



* 위 그림에서는 3개의 attention layer지만 본 논문에서는 8개의 attention layer 사용

* 병렬 처리함으로써 얻는 이점: 사람의 문장은 모호할 때가 있는데 한 개의 attention layer로는 충분히 encoding하기 어렵기 때문에 multi head attention을 통해 연관된 정보를 다른 관점에서 수집하고 보완할 수 있다.



Transformer에서는 multi-head attention을 세가지 방법으로 사용하였다.

1) query들은 이전 decoder layer로부터 얻고, encoder의 출력값을 key와 value로 사용한다. 따라서 decoder의 모든 위치의 token은 input sequence의 어느 곳이든 attend할 수 있다. (decoder의 다음 단어에 가장 적합한 단어를 출력하는 과정이라고 생각하면 됨)

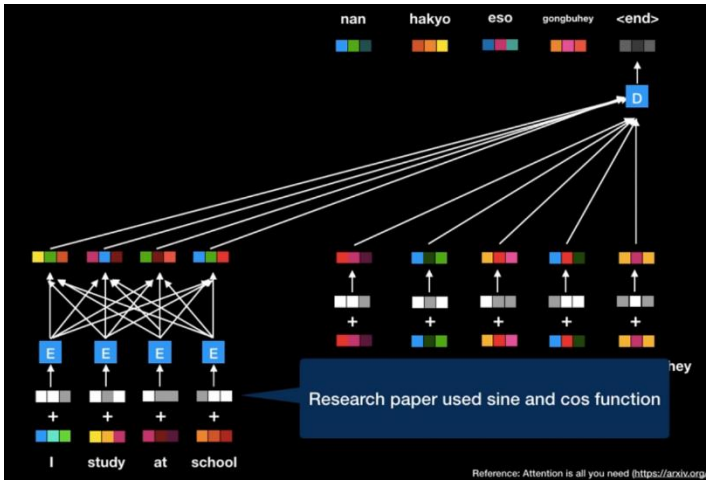
2) encoder는 self-attention layer를 갖는다. 모든 key, value, query는 같은 sequence 에서 오고 정확히는 이전 layer의 output에서 온다. 따라서 encoder는 이전 layer 의 전체 위치를 attend할 수 있다.

3) decoder의 self-attention layer도 이전 layer의 모든 position을 attend할 수 있는데, 정확히는 자신의 position 이전의 position까지만 attend할 수 있다. scaled dot-product를 masking함으로써 이를 구현하였다.

- Position-wise Feed-Forward Network: attention sub-layer에 이어서 FC network를 거치게 되는데 이 network는 두 개의 linear transformation으로 구성되어 있고, 사이에 ReLU함수를 사용한다.

- Embeddings and Softmax: embedding vector를 사용하는데, 입력 token을 linear transformation을 적용해 dimension vector로 바꿔주고, softmax로 나온 decoder의 결과값을 predicted next-token으로 다시 linear transformation해서 바꿔준다. 이 때 사용되는 weight matrix는 동일하다.

- **Positional Encoding**: 본 논문에서는 recurrence를 사용하지 않기 때문에 추가적으로 위치정보를 넣어줘야 한다. 각 위치에 대해서 embedding과 동일한 dimension을 가지도록 encoding을 해준 후 그 값을 embedding 값과 더해서 사용한다. 본 논문에서는 sin과 cos함수를 이용해 positional encoding값을 구하였다.



- Why self-attention

1) layer 당 전체 연산량이 줄어든다. -> 속도 향상

2) 병렬화가 가능한 연산이 늘어난다.

3) long-range의 term들의 dependency도 잘 학습할 수 있게 된다.

- **Conclusion**

  - Transformer는 translation task에서 RNN 또는 CNN layer를 기반한 구조보다 눈에 띄게 빠르게 학습할 수 있었다. 뿐만 아니라 이전의 모델들보다 좋은 성능을 보여주었다. Transformer는 image, audio, video 등 큰 input을 가진 task에도 적용할 수 있을 것이다.

- 참고

  - 허민석, 트랜스포머(어텐션 이즈 올 유 니드)

    https://www.youtube.com/watch?v=mxGCEWOxfe8&t=12s

  - Jay alarmmar, The Illustrated Transformer

    http://jalammar.github.io/illustrated-transformer/

    한글 번역본

    https://nlpinkorean.github.io/illustrated-transformer/

  - 논문 번역

    https://reniew.github.io/43/

<train 성공 캡쳐 화면>