Attention is All You Need

 Propose the Transformer

- based solely on attention mechanism

  →on recurrence and convolutions
- more parallelizable
- less time to train

 기존 모델의 한계점

1. Seq2Seq 모델
   - 병목 현상 발생

     →context vector에 문장 정보를 압축하기 때문
   - 소스 문장 압축 불가피

     →context vector의 고정된 크기 때문
2. RNN 모델
   - 병렬화 불가피

     → h_{t-1}이 hidden layer h_t으로 입력받는 순차연산

 Background

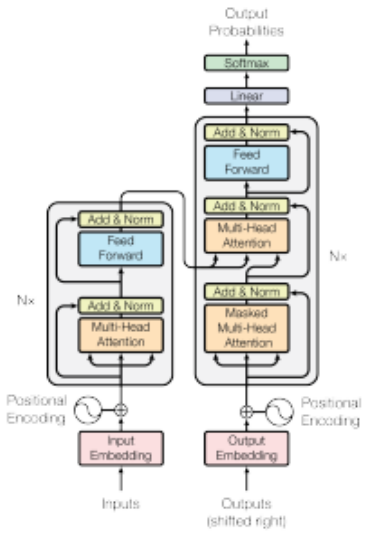
1. Self - Attention

- 목표 : 입력 문장에서 각 단어가 어떤 다른 단어와 연관성이 높은지 파악
- 핵심 구현 : single sequenec에서 different position of single sequence가 서로 가중치를 부여하며 관계 정도를 효과적으로 학습

2. Attention을 위한 3가지 입력 요소
    - Query : 해당 단어와 다른 단어의 연관성을 구하는 주체
    - Key : 해당 단어
    - Value : 다른 단어

Model Architecture & Transformer 동작 원리

Entire Model



좌측 : Encoder

우측 : Decoder

Layer :

Positional Encoding layer

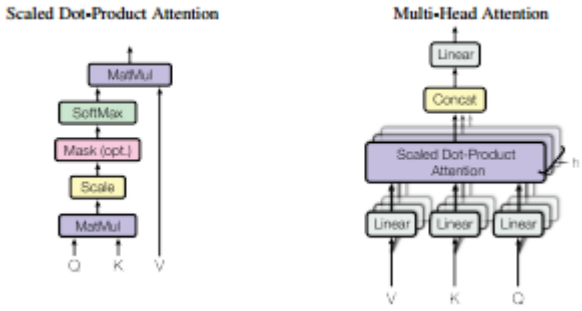
Multi-head Attention layer

Residual Adding & Normalization layer

Positional Wise Feed Forward layer

1. Attention

    Query와 key - value 값을 이용해 weight sum과 softmax 계산으로 통해 output 산출



- Scaled Dot-Product Attention

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

Input : Query, Key → d_k차원, Value → d_v차원

QK^T : 내적을 통해 높은 연관성 가지는 것을 파악

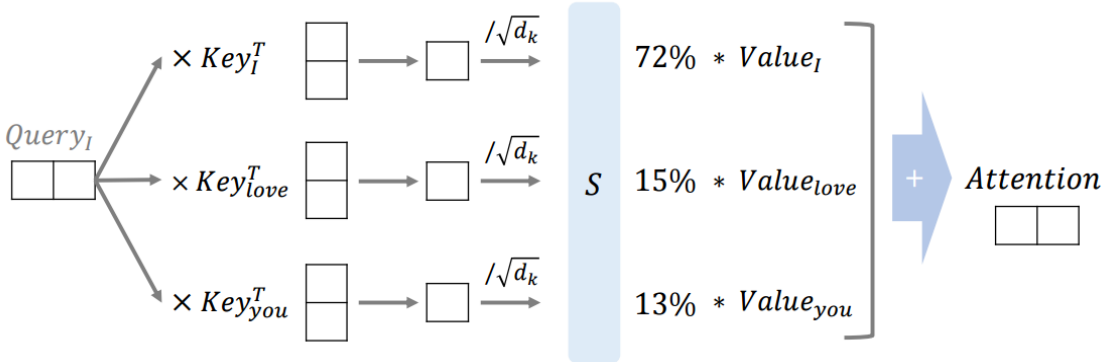
$\frac{QK^T}{\sqrt{d_k}}$: $\sqrt{d_k}$ 로 나눠 Softmax의 gradient vanishing 해결

$softmax(\frac{QK^T}{\sqrt{d_k}})$: Attention Distribution

(softmax를 취해 각 Key에 대한 연관성을 확률로 표현)

$softmax(\frac{QK^T}{\sqrt{d_k}})V$: Attention Value

(Attention Distribution에 Value를 곱함)



scaled dot-product attention 과정 벡터 차원의 도식화

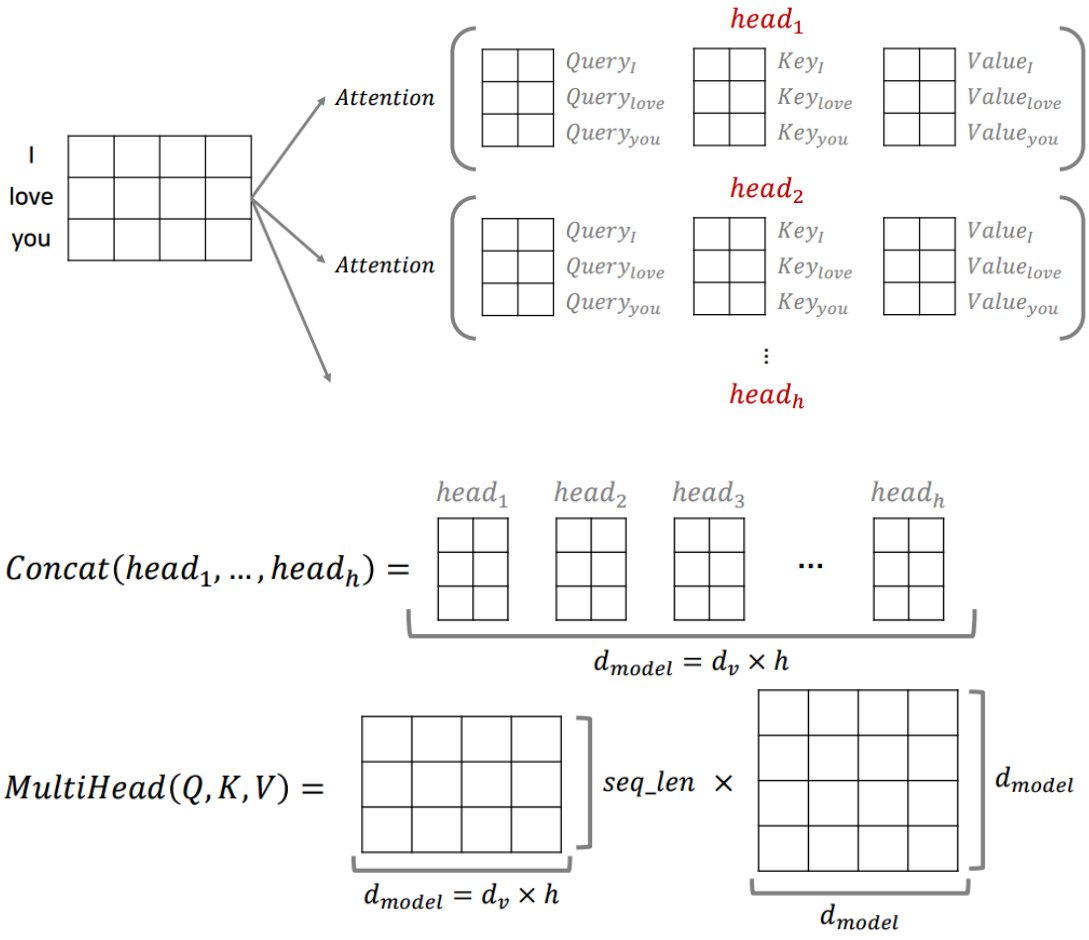
- Multi-Head Attention

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

여러 번의 attention연산을 head마다 h번 수행

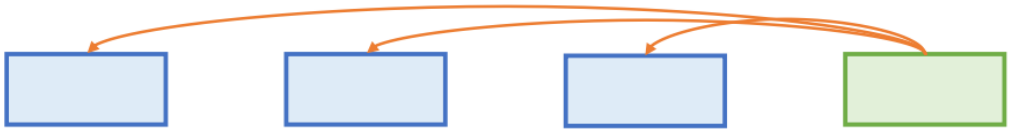
$$MultiHead(Q,K,V) = Concat(head_1, ..., head_h)W^o$$

Q,K,V의 Embedding Vector를 concat(합쳐서) linear layer를 거쳐 output값 도출



Multi-Head Attention 과정 벡터 차원의 도식화

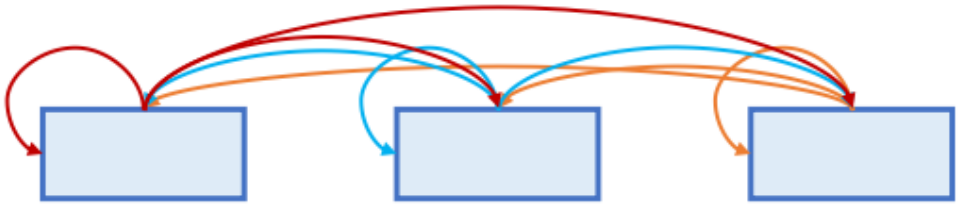
- Attention applied to Model
  1. Enconder-decoder attention



이전 decoder output인 query & encoder output인 key, value 받음

→allow decoder to attend all input sequence position

2. Self-attention layers in encoder



이전 encoder output인 Query, Key, Value 받음

→allow encoder to attend all position in encoder's previous layer

3. Self-attention layers in decoder

이전 decoder output인 Query, Key, Value 받음

→ allow decoder to attend all position in decoder up to

⭐Masking

→뒤쪽의 단어를 미리 알지 못하도록 하는 방식

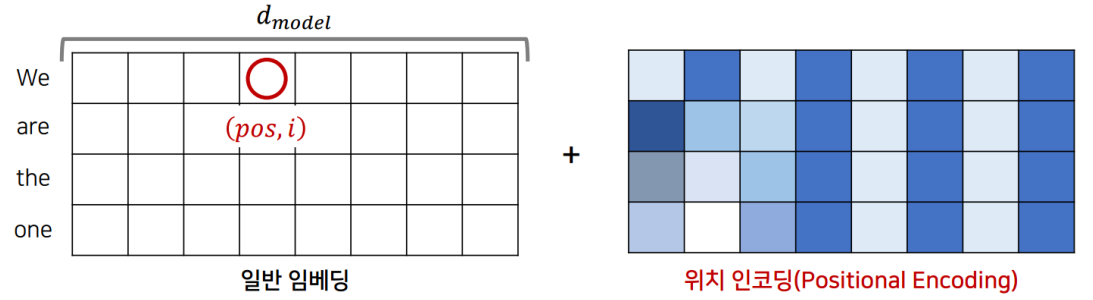
2. Positional Encoding

단어들의 상대적인 위치에 대한 정보를 알려주기 위해 주기 함수를 활용한 공식 사용

$$PE_{(pos,2i)} = sin(pos/1000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = sin(pos/1000^{2i/d_{model}})$$

sequence의 순서를 위해서 각 단어의 상대적인 위치 정보를 담아 모델에 입력



pos와 i를 positional encoding 함수에 입력해 입력값과 동일한 차원을 가지는 위치 인코딩을 만들어 element wise의 덧셈을 진행해 encoder layer와 decoder layer의 input으로 사용

3. Position-wise Feed Forward Networks

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

two linear transformations with ReLU activiation

⭐ Reason for Using Self - Attenton

1. layer에서 이루어지는 연산 감소
2. 병렬적으로 연산 가능
3. 경로의 길이에 따른 long - range 학습의 용이함

   The shorter length of the paths forward and backward between any combination of positions in the input and output sequences, the easier it is to learn long-range dependencies

⭐ Conclusion

The Transformer는RNN 또는 CNN 없이 오직 attention으로만 구현된 최초의 시퀀스 변환 모델

Reference:

이미지 : 나동빈 자연어 처리 : 트랜스포머

https://greeksharifa.github.io/nlp(natural language processing) / rnns/2019/08/17/Attention-Is-All-You-Need/