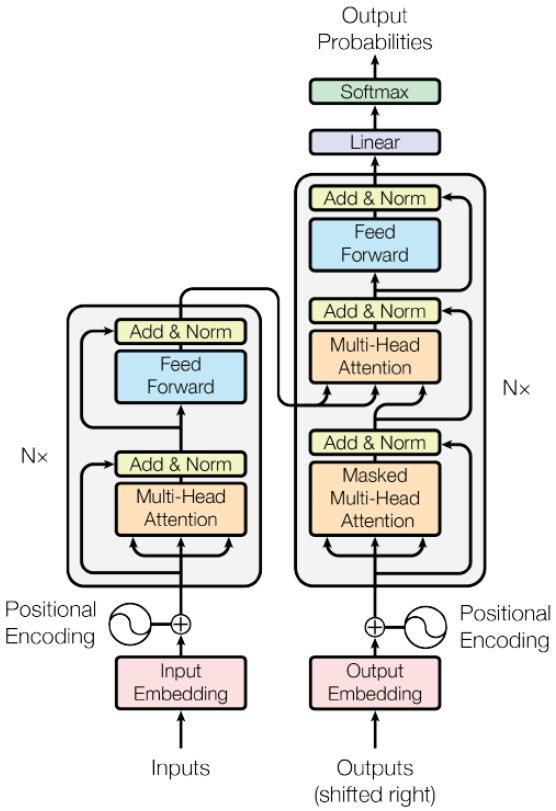
[6주차] Transformer : Attention Is All You Need

Transformer

트랜스포머(Transformer)는 2017년 구글이 발표한 논문 "Attention is all you need"에서 소개한 모델입니다. 기존의 seq2seq의 구조인 **인코더-디코더**를 따르면서도, 논문의 이름처럼 RNN을 사용하지 않고 **Attention만으로 구현**한 모델입니다.

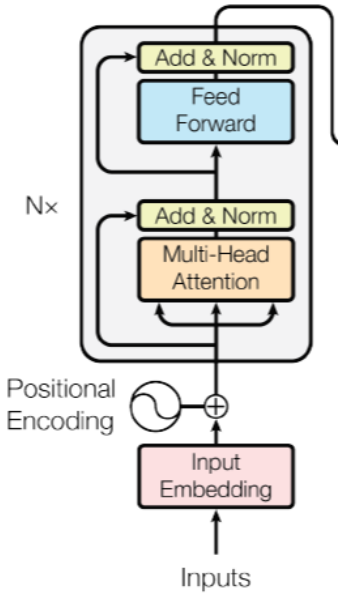
Model Architecture



Encoder and Decoder Stacks

(1) Encoder

위 그림에서 왼쪽에 해당하는 것으로 동일한 레이어의 인코더 6개(N=6) 를 병렬로 연결해 구성합니다. 인코더는 총 두 개의 서브층으로 이루어지는데, **멀티 헤드 어텐션**과 **피드 포워드 신경망**입니다. 각 서브층 이후에는 **드롭 아웃**, **잔차 연결**과 **층 정규화**가 수행됩니다.



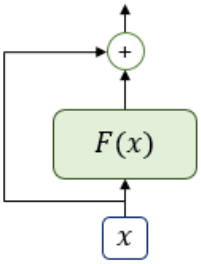
첫번째 서브층 : multi-head self- attention mechanism

두번째 서브층 : simple position-wise fully connected feed-forward network

그림에서 Add & Norm 에 해당하는 부분으로 두개의 서브층 각각에 **잔차연결(residual connection)과 층 정규화(layer normalization)**을 사용합니다.

- **잔차연결**

$$H(x) = x + F(x)$$



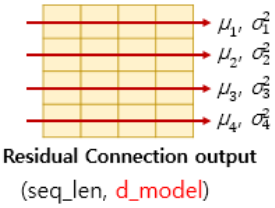
잔차 연결은 서브층의 입력과 출력을 더하는 것

컴퓨터 비전 분야에서 주로 사용되는 모델의 학습을 돕는 기법입니다.

ex) 서브층이 멀티 헤드 어텐션이었다면 잔차 연결 연산은 다음과 같습니다.

$$H(x) = x + multihead - attention(x)$$

- **층 정규화**



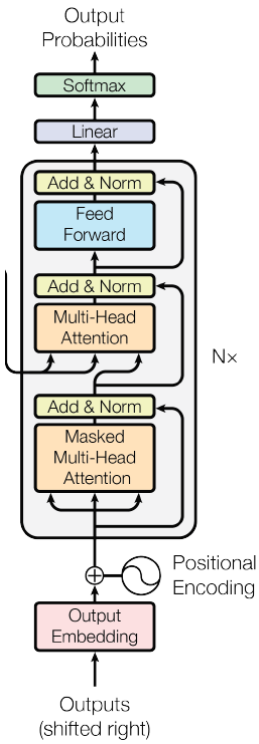
층 정규화는 **텐서의 마지막 차원**에 대해서 평균과 분산을 구하고, 이를 가지고 각 벡터 값을 정규화하여 학습을 돕습니다.

층 정규화를 적용해, 최종 함수는 다음과 같습니다.

$$LayerNorm(x + Sublayer(x))$$

(2) Decoder

위 그림에서 오른쪽에 해당하는 것으로 마찬가지로 디코더 6개(N=6)를 stack해 구성합니다. 디코더는 총 세 개의 서브층으로 구성됩니다. 인코더에서 봤던 2개의 서브층(**멀티 헤드 어텐션**과 **피드 포워드 신경망)**과 더불어 **마스크드 멀티 헤드 셀프 어텐션(Masked multi-head self-attention)**을 추가합니다. 3개의 서브층 각각 연산 후에는 **드롭 아웃**, **잔차 연결**, **층 정규화**가 수행됩니다.



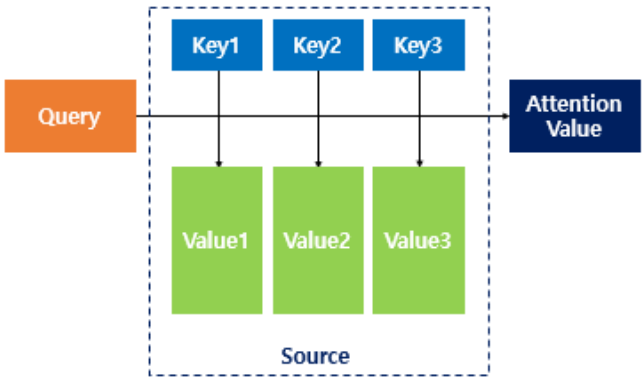
첫번째 서브층 : Masked multi-head self-attention

두번째 서브층 : multi-head attention mechanism

세번째 서브층 : simple position-wise fully connected feed-forward network

Attention

Attention은 **Query**에서 모든 **key**와의 유사도를 구하고, **key**에 해당하는 각 **value**를 유사도 가중합으로 **Attention value**가 계산됩니다**.** Query, Keys, Values는 sequence의 모든 단어 벡터들입니다.

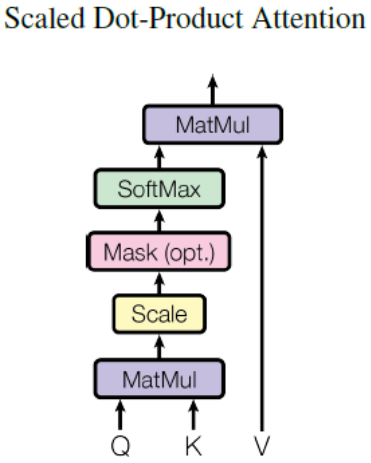


1. Query vector: 현재 처리하고자 하는 token을 나타내는 vector

2. Key vector: 일종의 label, 시퀀스 내에 있는 모든 토큰에 대한 identity

3. Value vector: Key와 연결된 실제 토큰을 나타내는 vector

(1) Scaled Dot- Product Attention



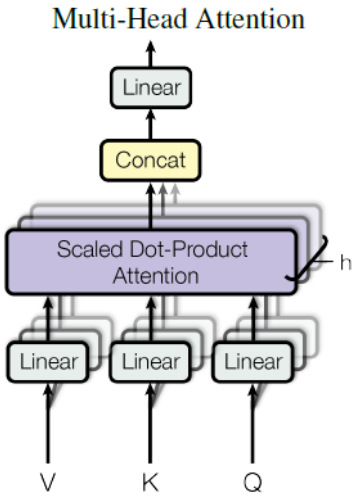
Scaled Dot- Product Attention은 어텐션 함수 종류 중 하나로 트랜스포머에서 사용되었습니다.

1. **queries,** 차원이 d_k인 **keys, value**를 입력합니다.
2. 두 행렬(Q, K)의 내적값에 d_k에 루트를 씌운 값으로 나누어줘 스케일링해줍니다.
3. <PAD> 토큰이 존재한다면 이에 대해서는 유사도를 구하지 않도록 마스킹(Masking)을 해 값을 가려줍니다.
4. 소프트맥스 함수를 사용하여 어텐션 분포(Attention Distribution)을 구합니다.
5. V행렬와 가중합하여 어텐션 값(Attention Value)을 구합니다.

최종 출력식은 다음과 같습니다.

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

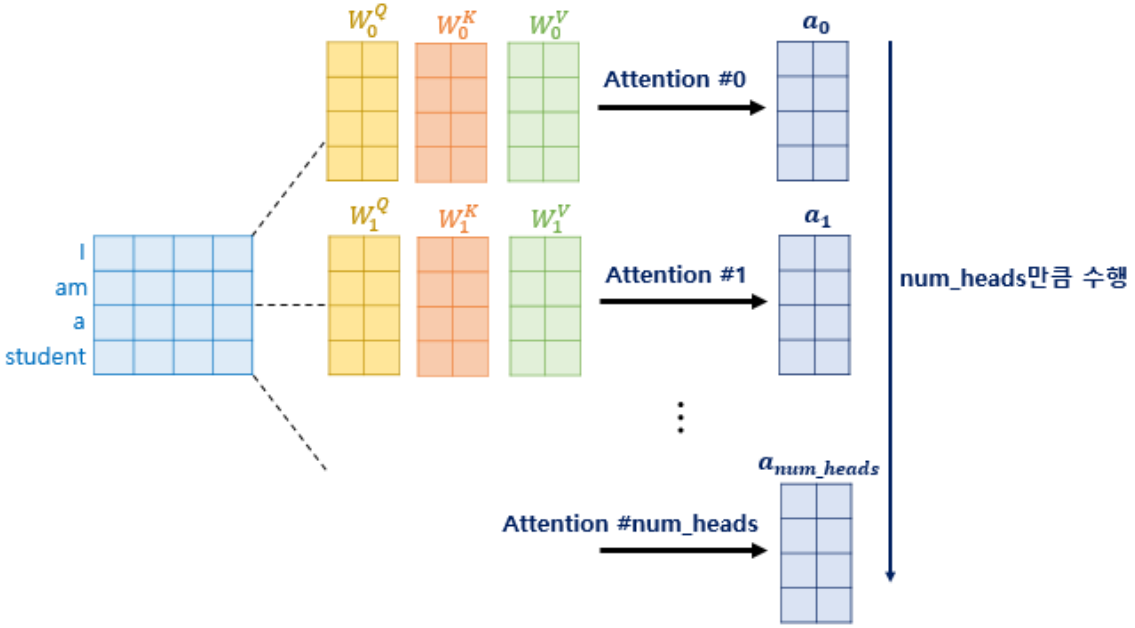
(2) Multi-Head Attention



입력 임베딩 차원의 queries, keys, values를 사용한 한번의 attention 함수보다, 입력 입베딩을 각각 d_v, d_k, d_q **차원으로 축소**해 attention을 수행하면 다른 시각으로 정보들을 수집할 수 있습니다. 즉 어탠션을 병렬적으로 수행합니다. 이때 각각의 어텐션 값 행렬을 **어텐션 헤드**라고 부릅니다. 결과적으로 d_v 차원의 어텐션 헤드들이 만들어지고 이것들을 합치고**(concatenated) 다시 투영(projection)해 최종 값**을 얻을 수 있습니다.

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

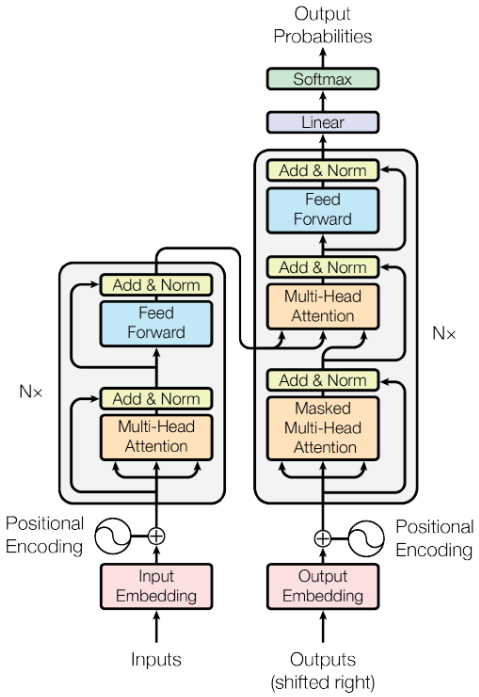


이는 서로 다른 positions에 서로 다른 representation subspaces에서 결합적으로(jointly) 정보에 접근할 수 있도록 합니다. 따라서 단어간의 연관도를 구할 때, 문법적 구조나 의미적 구조에 대해서 복합적으로 반영할 수 있습니다.

본 논문에서 **8개의 parallel attention layers**(h=8)을, 각 d_k, d_v는 **64차원**(512/8=64)을 사용했습니다.

(3) Applications of Attention in our Model

Transformer은 multi-head attention을 3가지 방법으로 사용했습니다.



- **디코더의 두번째 서브층, Multi-Head Attention:** encoder-decoder attention layer를 사용합니다. queries는 이전의 디코더 레이어에서 오고 memory keys와 values는 인코더의 출력으로부터 옵니다. 이것은 디코더가 입력 sequence의 모든 position을 고려할 수 있도록 합니다. 이는 sequence-to-sequence 모델의 일반적인 인코더-디코더 attention 매커니즘을 모방한 것입니다.
- **인코더의 첫번째 서브층, Multi-Head Attntion:** 인코더는 self-attention layers를 사용합니다. self-attention layer에서 모든 key, values, queries는 같은 곳(인코더의 이전 레이어의 출력)에서 옵니다. 인코더의 각 position은 이전 레이어의 모든 position을 고려할 수 있습니다.
- **디코더의 첫번째 서브층, Masked Multi-Head Attention:** 디코더의 self-attention layer로 디코더가 해당 position과 이전까지의 positon만을 고려할 수 있도록 합니다. 현재 시점의 단어와 이전 시점의 단어가 입력인 RNN계열과 달리 디코더의 입력은 문장행렬이기 때문에 뒤에 오는 단어를 참고하지 못하도록 **masking**을 사용했습니다. 즉 **i 시점의 단어가 오직 i 이전의 단어에만 의존**하도록

했습니다. 아주 작은 값에 수렴하도록 값을 주고 softmax를 적용하는 방법으로 미래의 position에 대한 영향을 masking out할 수 있습니다.

📌 인코더의 셀프 어텐션 : Query =**Key**= Value
디코더의 마스크드 셀프 어텐션 : Query =**Key**= Value
디코더의 인코더-디코더 어텐션 : Query : 디코더 벡터 /**Key**= Value : 인코더 벡터

Position-wise Feed-Forward Networks

인코더와 디코더의 각 레이어는 **Fully Connected feed-forward network(FFNN)**을 포함합니다. 각 단어 또는 문장에 독립적으로 동일하게 적용됩니다. 이것은 두개의 선형변환과 ReLU로 이루어져 있습니다.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

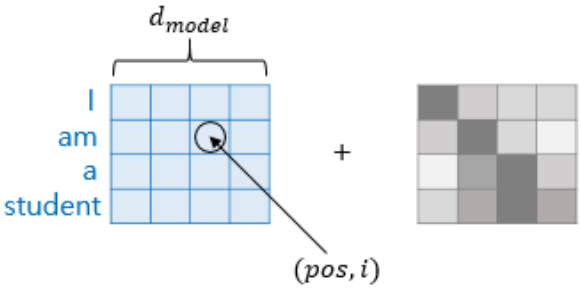
모수(W_1, W_2, b_1, b_2)는 하나의 인코더 층 내의 다른 단어 또는 다른 문장마다 동일한 값으로 선형변환을 하지만, 인코더 층이 달라지면 다른 값을 사용합니다. 논문에서 제시한 입력과 출력의 차원은 d_model = 512, 그리고 inner-layer의 차원은 d_ff = 2048입니다.

Embeddings and Softmax

다른 sequence 변환과 비슷하게, transformer도 입력 토큰과 출력 토큰을 차원이 d_model인 벡터로 전환하기 위해 learned embedding을 사용합니다. 디코더의 출력으로 next-token 확률을 예측하기 위해 선형변환과 소프트맥스 함수 사용합니다.

Positional Encoding

Transformer는 RNN과 CNN을 사용하지 않기 때문에 **sequence의 순서를 반영하기 위해서** 단어 토큰의 상대적 또는 절대적 위치에 대한 정보를 삽입하는 것이 필요합니다. 따라서 인코더와 디코더의 입력 임베딩에 **"Positional Encoding"**을 추가했습니다. **위치 정보를 가진 값을 만들기 위해** 다음 두 함수를 사용합니다.



$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

pos : 위치 (position)

i : 차원 인덱스(dimension)

각 위치 인코딩의 차원은 사인곡선(sinusoid)에 대응됩니다. **pos**는 입력 문장에서의 임베딩 벡터의 위치를 나타내며, **i**는 임베딩 벡터 내의 차원의 인덱스를 의미합니다.인덱스가 **짝수(2i)일 때는 사인함수**를 사용하고 **홀수(2i+1)일 때는 코사인 함수**를 사용합니다. 어느 k 값이라도 $PE_{(pos+k)}$는 선형 함수 $PE_{(pos)}$로 나타날 수 있기 때문에 사인·코사인 함수를 선택했습니다.

각 임베딩 벡터에 포지셔널 인코딩의 값을 더하면 같은 단어라고 하더라도 문장 내의 위치에 따라서 트랜스포머의 입력으로 들어가는 임베딩 벡터의 값이 달라집니다. 이에 따라 **트랜스포머의 입력은 순서 정보가 고려된 임베딩 벡터** 가 됩니다.

Conclusion

해당 논문에서는, encoder-decoder 구조에서 가장 흔하게 사용되는 recurrent layer를 multi-head self-attention으로 교체하면서, **전적으로 attention에만 의존하는 최초의 시퀀스 변환 모델, Transformer**를 제안하였습니다.

번역 task에서, Transformer는 recurrent, convolutional layer를 기반으로 하는 구조들보다 훨씬 빨리 학습될 수 있습니다. WMT 2014 English-to-German, WMT 2014 English-to-French 번역 task 둘 다, 새로운 SOTA를 달성하였습니다. 해당 논문에서의 최고의 모델은 이전에 연구되었던 모든 앙상블을 뛰어넘었습니다.

코드 실습

https://colab.research.google.com/drive/1Loe7uiTo31KdfcxkjIlzViZiv97eKJIq#scrollTo=hVH6RH-jvVZv

Reference

Attention Is All You Need https://arxiv.org/pdf/1706.03762.pdf

https://wikidocs.net/31379

https://velog.io/@changdaeoh/Transformer-논문리뷰

Attention Is All You Need (NIPS 2017) (tistory.com)