Attention Is All You Need

0. Abstract

보통의 sequence transduction model들은 인코더와 디코더를 포함하는 rnn 혹은 cnn을 기반으로 하거나 SOTA 모델은 attention 메커니즘을 통해 인코더와 디코더를 연결한다.

우리는 rnn과 cnn을 완전 배제하고 attention 메커니즘만을 사용하는 단순한 아키텍쳐인 transformer를 제안한다.

실험 결과, 우리 모델이 병렬화가 훨씬 용이하고 학습 시간이 훨씬 더 적지만 성능은 높다는 것을 보인다 (28.4 BLEU on the WMT 2014 Englishto-German translation task, 41.8 BLEU on the WMT 2014 English-to-French translation task).

추가적으로, Transformer가 다른 task에도 잘 일반화된다는 것을 보인다.

1. Introduction

RNN, LSTM(long short-term memory), gated recurrent neural networks은 언어 모델링 및 기계 번역과 같은 sequence modeling 및 transduction 문제에서 SOTA 방법론으로 제안되어왔다.

Recurrent model은 일반적으로 input 및 output sequence의 symbol positions을 따라 계산한다. 계산의 steps의 위치에 따라, 이전 hidden state ht-1과 position t가 input인 hidden state ht가 생성된다. 이러한 순차적인 특성은 병렬화를 막고, 긴 문장에 대해서 중요한 문제이다 (ht-1 처리 후에야 ht를 처리할 수 있는 순차적인 특성). 최근에는 factorization tricks과 conditional computation을 통해 개선했지만, 근본적인 순차적인 연산에 대한 한계점은 여전하다.

Attention 메커니즘은 input 또는 output sequence의 거리에 관계없이 의존성을 모델링함으로써 필수적이게 되었지만, 거의 대부분 recurrent 네트워크와 함께 사용되었다 (효율적인 병렬화 불가능).

본 연구는 recurrence를 제거하고 input과 output 사이의 global 의존성을 학습하기 위한 attention 메커니즘만을 사용한 모델 transformer를 제안한다. Transformer는 8개의 P100 GPU로 12시간 학습하여 병렬화 및 SOTA를 달성한다.

2. Background

sequential 연산을 줄이려는 많은 노력이 있었지만 (Extended Neural GPU, ByteNet, ConvS2S), 이는 모두 cnn을 사용한다. 이에 따라 원거리같이 의존성을 학습하는 것이 어렵다.

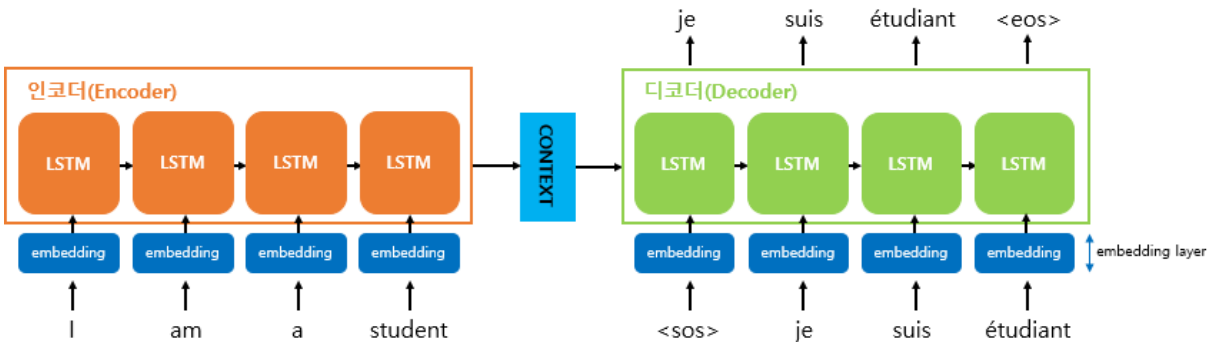
Self-attention(intra-attention)은 sequence의 representation을 계산하기 위해 single sequence의 다른 위치를 연관시키는 attention 메커니즘이다. Self attention은 다양한 task에서 성공적으로 사용되어 왔다.

End-to-End 메모리 네트워크는 sequence aligned recurrence 대신 recurrent attention mechanism을 기반으로 하며 single-language QA와 언어모델링 task에서 좋은 성능을 보인다.

Transformer는 rnn 혹은 cnn을 사용하지 않고 input과 output의 representation을 계산하기 위해 self-attention만을 사용하는 transduction model이다.

3. Model Architecture

대부분의 sequence trasduction model은 아래와 같은 encoder-decoder구조를 가진다 (아래는 seq2seq 예시이다). 이런 모델의 각 단계는 auto regressive이며, 이전에 생성된 output을 다음 단계에서 사용한다 (이전 단계가 완료가 된 후에 다음 단계를 수행할 수 있다는 말 = 순차적으로 진행 = 병렬적으로 처리가 불가능함).



본 논문에서 제안하는 Transformer는 아래 그림과 같은 아키텍처를 따른다.

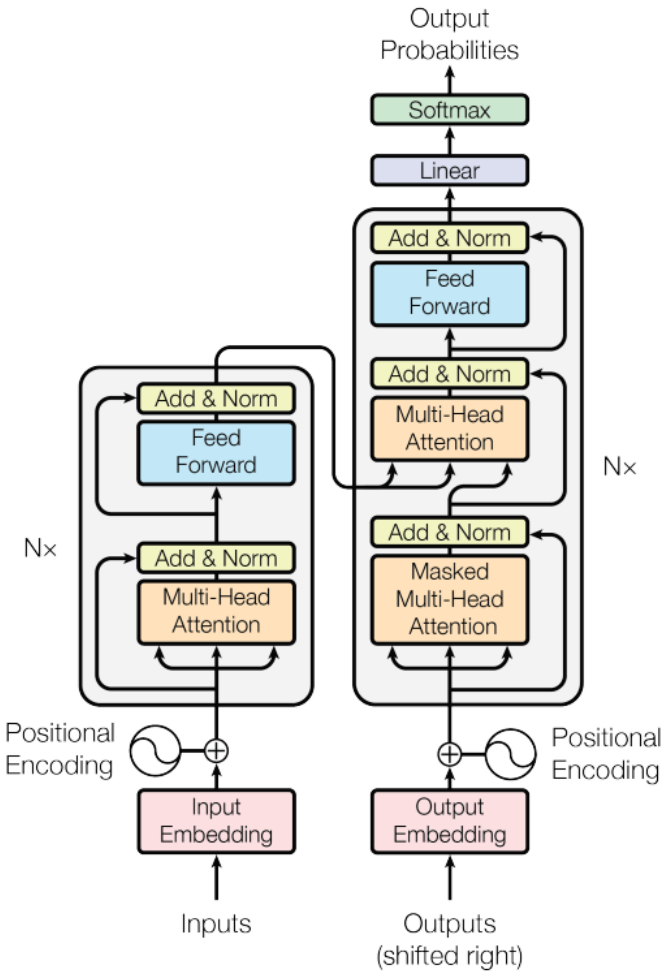


Figure 1: The Transformer - model architecture.

3.1 Encoder and Decoder Stacks

Encoder

encoder는 N=6개의 동일한 layer stack으로 구성된다. 각 layer에는 두 개의 sub-layer가 있다. 첫 번째는 multi-head self-attention 메커니즘이고, 두 번째는 FFN(feed foward network)이다. 각 sub-layer에 residual connection과 layer normalization을 사용한다. 즉, sub-layer의 output은 LayerNorm(x + Sublayer(x))이다. 여기서 Sublayer(x)는 sub-layer 자체에 의해 구현된 function (multi-head attention 혹은 FFN을 의미)이다. Residual connections을 용이하게 하기 위해 embedding layer 뿐만 아니라 모델의 모든 sub-layer는 d_model = 512 차원의 output을 생성한다 (input과 동일한 차원의 output을 생성하여 단순 + 연산을 쉽게 할 수 있음).

--> 즉, encoder = layer x 6이고, layer는 LayerNorm(x+Multi-head attention(x)) + LayerNorm(x+FFN(x))로 이루어져 있다.

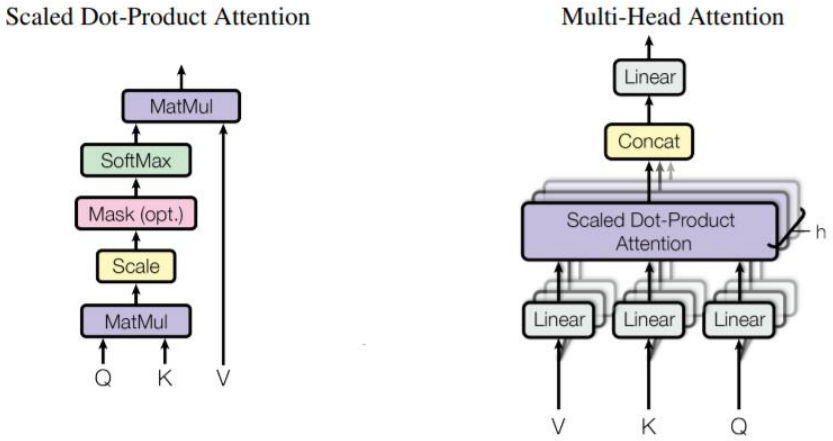
Decoder

디코더는 또한 N = 6개의 동일한 layer stack으로 구성된다. 각 encoder layer의 두 개의 sub-layer 외에도 decoder는 세 번째 sub-layer를 추가하여 encoder stack의 output에 대해 multi-head attention을 수행한다 (총 3개의 sub-layer). Encoder와 유사하게, 각 sub-layer에 residual connection과 layer normalization을 사용한다. 또한, decoder stack의 self-attention sub-layer를 수정하여 position이 subsequent positions에 attending하는 것을 막는다 (=masking). Output embedding이 한 위치의 offset이라는 사실과 결합한 masking은 position i에 대한 예측이 i보다 적은 수의 위치 (i 이전에 output된 위치)에서 알려진 output에만 의존할 수 있도록 한다 (미래 시점을 막아주는 역할).

3.2 Attention

Attention function은 vector query와 vector key-value 집합 쌍을 query, keys, values, output이 모두 벡터인 output에 mapping하는 것이다. Output은 각 values의 weighted sum으로 계산되며, 여기서 각 value에 할당된 weight는 query와 해당 key의 compatibility function에 의해 계산된다.

Q: 영향을 받는 벡터 K: 영향을 주는 벡터 V: 주는 영향의 가중치 벡터



3.2.1 Scaled Dot-Product Attention

Scaled Dot-Production attention은 위 그림의 왼쪽에 나와있다. **Input은 d_k차원의 query, key와 d_v차원의 value**로 구성된다. Key와 query의 dot product를 계산하고, 각각의 값을 √(d_k)로 나눈 후 softmax 함수를 적용하여 value에 대한 weight를 구한다.

Output matrix는 다음과 같이 계산한다.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

주로 사용되는 attention function은 additive attention과 dot-product (multiplicative) attention이다.

dot-product 계산법

values의 weight를 구하기 위해 query와 all key를 곱하고 d_k 를 각각 나누고 softmax 함수를 적용한다. 실제, Matrix Q로 변환해서 queries set를 동시에 attention function을 계산한다. Keys와 values 역시 matrices K와 V로 계산한다.

softmax를 적용하기 전 작은 gradients를 가지기 위해 $1/\sqrt{d_k}$ 을 곱해서 dot product을 조정한다.

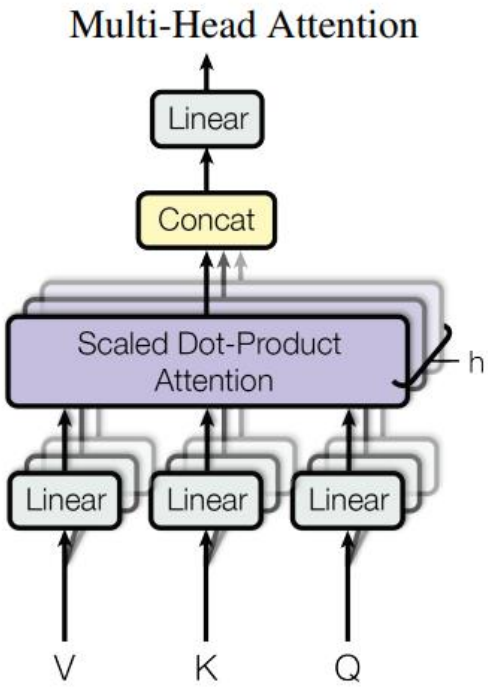
(논문에서는 dot products는 크기를 키우고, 큰 dk 는 좋지 않은 영향이 있을거라 의심하고 있다.)

두 방법은 이론적으로 복잡성이 비슷하지만, **dot-product attetion은 matrix를 통해 최적화된 연산을 구현할 수 있기 때문에 훨씬 빠르고 공간 효율적**이다.

d_k가 값이 작은 경우에는 dot-product와 scaled dot-product가 유사하게 수행되지만, 값이 큰 경우에는 후자가 우수하다. d_k 값이 클 때, dot-product의 size가 커지면서 softmax를 극도로 작은 gradient를 갖게끔 한다고 생각한다. 이를 개선하기 위해 **1/√(dk)만큼 스케일링**한다.

3.2.2 Multi-Head Attention

d_model 차원의 query, key, value를 사용하여 single attention을 수행하는 대신, 각각 d_k, d_k, d_v 차원에 대해 학습된 서로 다른 linear projection을 사용하여 query, key, value를 h회 linear projection하는 것이 유익하다는 것을 발견했다. 이러한 query, key, value의 각 projection version에서 attention funtction을 **병렬로 수행하여 d_v차원 output을 생성하고, 이를 concat하여 다시 d_model 차원의 output이 생성**된다.



Multi-head attention을 통해 모델은 다른 position의 서로 다른 representation subspaces의 정보에 공동으로 attend할 수 있다.

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

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 $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

h = 8개의 병렬 attention layer 혹은 head를 사용한다. d_k = d_v = d_model/h = 64를 사용한다. 각 head의 축소된 차원으로 인해 총 계산 비용은 전체 차원을 갖는 single-head attention 비용과 유사하다.

3.2.3 Applications of Attention in our Model

Transformer는 세 가지 방식으로 multi-head attnetion을 사용한다.

1. **"Encoder-Decoder Attention layer"** 에서 **query는 이전 decoder layer에서 얻고, key와 value는 encoder의 output에서 얻는다**. 이를 통해 decoder의 모든 position이 input sequence의 모든 position에 배치될 수 있다. 이는 sequence-to-sequence 모델에서 **일반적인 encoder-decoder attention 메커니즘과 동일**하다.
2. **"Encoder Self-Attention layer"** 는 **encoder에 존재하며 query, key, value가 동일**하며, 이는 encoder에 있는 이전 layer의 output이다. Encoder의 각 position은 이전 layer의 모든

position에 attend 할 수 있다.

3. **"Masked Decoder Self-Attention layer"**는 decoder에 존재하며, 마찬가지로, decoder의 self-attention layer는 각 position이 해당 position의 위치까지 docoder의 모든 position에 attned하도록 한다. auto-regressive propert를 유지하기 위해 decoder에서 leftward information flow을 막아야 한다 **(미래 시점의 단어를 볼 수 없도록 하는 것)**. 이를 위해 매우 작은 수를 부여하여 softmax 결과 0에 수렴하도록 하여 **masking을 수행**한다.우리는 잘못된 연결에 해당하는 소프트맥스 입력의 모든 값을 마스킹(-)로 설정)하여 스케일링된 도트 제품 주의의 내부에서 이를 구현한다.

3.3 Position-wise Feed-Forward Networks

attention sub-layer 외에도 encoder와 decoder의 각 layer에는 FFN가 포함되어 있으며, 이는 각 position에 개별적으로 동일하게 적용된다. 이것은 중간에 ReLU activation이 있는 두 가지 linear transformation으로 구성된다.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

linear transformation은 여러 position에서 동일하지만 layer마다 다른 파라미터를 사용한다. 이것을 설명하는 또 다른 방법은 커널 크기가 1인 두 개의 convolution을 사용하는 것이다. Input과 output의 차원은 d_model=512이고, 내부 layer의 차원은 d_ff=2048이다.

3.4 Embeddings and Softmax

다른 sequence trasduction 모델과 유사하게, 학습된 embedding을 사용하여 input token과 output token을 d_model차원의 벡터로 변환한다. 또한, 일반적으로 학습된 linear transformation과 softmax 함수를 사용하여 decoder output을 예측된 다음 token 확률로 변환한다. 본 모델에서, 두 embedding layer와 softmax 이전 linear transformation 사이에서 동일한 weight matrix를 공유한다. Inner layer에서 이러한 weight를 가중치에 √(d_model)을 곱한다.

3.5 Positional Encoding

Transformer는 순차적인 특성이 없고 이에 따라 **sequence의 위치 정보가 없기 때문에 positional 정보를 추가**해줘야한다. 이를 위해, **encoder와 decoder input embedding에 "positional encoding'을 추가**한다. Positional encoding은 embedding과 동일한 차원을 가진다.

본 연구에서는 다음과 같이 sin, cos 함수를 사용한다.

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

여기서 *pos*는 token의 위치를 의미하고, i는 차원을 의미한다. 즉, positional encoding의 각 차원은 사인파에 해당한다. geometric progression wavelengths 은 2π에서 10000까지이다. **PE_pos+k가 PE_pos의 선형 함수로 표현**될 수 있기에 모델이 상대적인 위치를 쉽게 학습할 수 있을 것이라 가정했기 때문에 이 함수를 사용한다.

학습가능한 positional embedding을 사용해 봤지만 동일한 성능을 보이는 것을 확인할 수 있었고, 더 긴 시퀀스에서도 추론이 가능한 사인파 버전을 사용한다.

4. Why Self-Attention

Self-attention layers와 recurrent and convolution layers와의 비교를 수행한다. 비교 방법은 symbol representations(x1, ..., xn)의 one variable-length sequence를 같은 길이 (z1, ..., zn)으로 mapping 이다. 우리가 self-attetion을 사용하는 이유는 세가지이다.

layer별 총 연산의 complexity.

1. 필요한 최소 sequential operations으로 측정한 병렬처리 연산량.
2. Network에서 long-range dependencies(장거리 의존성) 사이 path length. long-range dependencies의 학습은 번역 업무에서 핵심 task이다. 이러한 dependencies을 학습하는 능력에 영향을 미치는 한 가지 핵심 요소는 전달해야 하는 forward 및 backward signal의 길이다. Input의 위치와 output의 위치의 길이가 짧을수록 dependencies 학습은 더욱 쉬워진다. 그래서 서로 다른 layer types로 구성된 네트워크에서 input과 output 위치 사이 길이가 maximum 길이를 비교한다.