Attention is All You Need

2022-05-08

Hajin Lee

Table of Contents

Introduction

- Recurrent Models
  - Sequential nature -> parallelization in training not possible -> too much memory

- Attention mechanism
  - Input과 output 간의 거리와 상관없이 dependency를 모델링할 수 있음
  - 대부분의 경우 recurrent network와 함께 사용되었음

- Transformer
  - Recurrence 완전히 배제하고 온전히 attention mechanism에만 의존
  - 상당히 더 많은 parallelization 가능

Transformer Model Structure

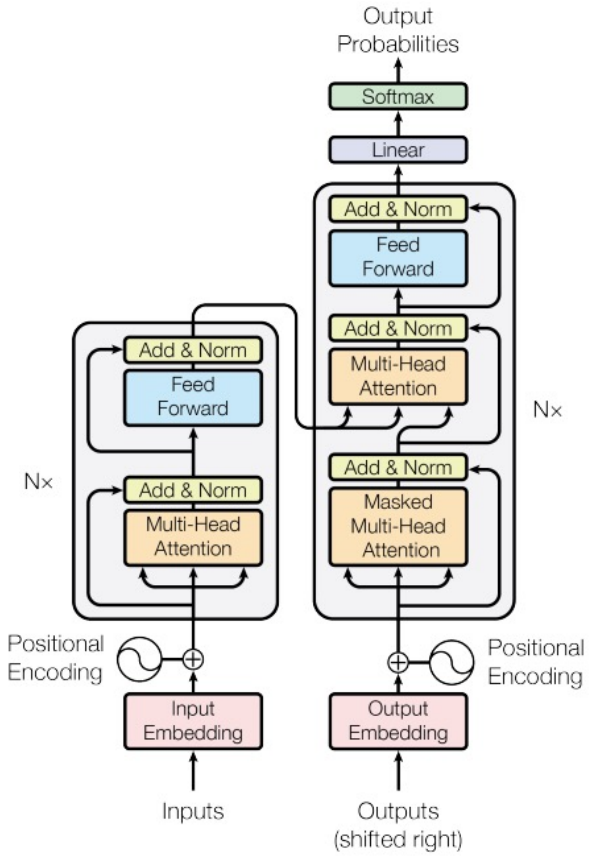


Figure 1: The Transformer - model architecture.

- Encoder – Decoder 형식으로 구성
- Auto-regressive
- 온전히 attention에 기반한 첫 sequence transduction 모델

Transformer Model Structure: Encoder

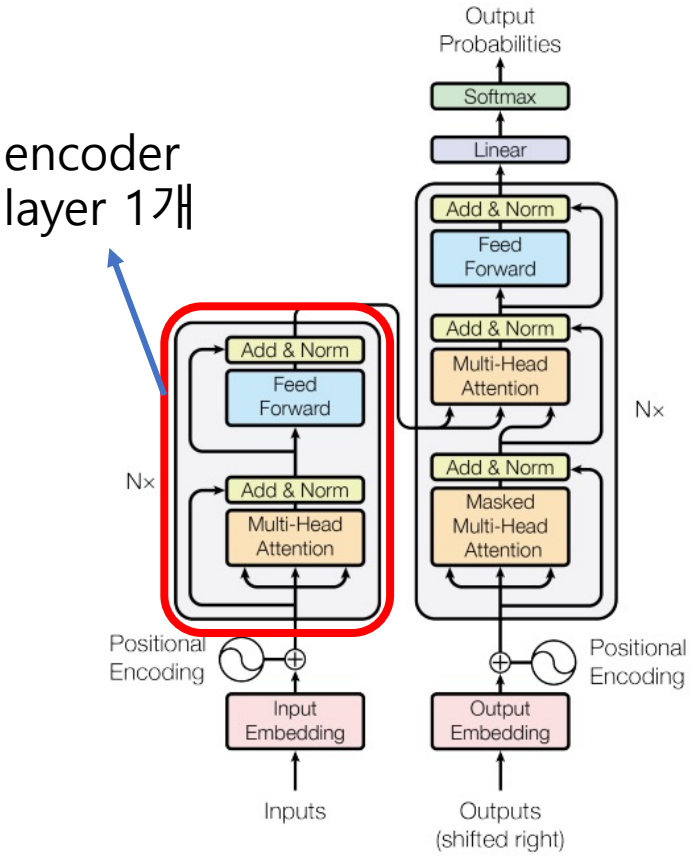


Figure 1: The Transformer - model architecture.

- N=6개의 동일한 layer로 구성
- 각 layer는 2개의 sub-layer로 구성
  - Multi-head self-attention sub-layer
  - Simple, position-wise fully connected feed-forward network
- 각각의 sub-layer에 residual connection과 layer normalization 적용
- 각 sub-layer의 output: $LayerNorm(x + Sublayer(x))$

Transformer Model Structure: Decoder

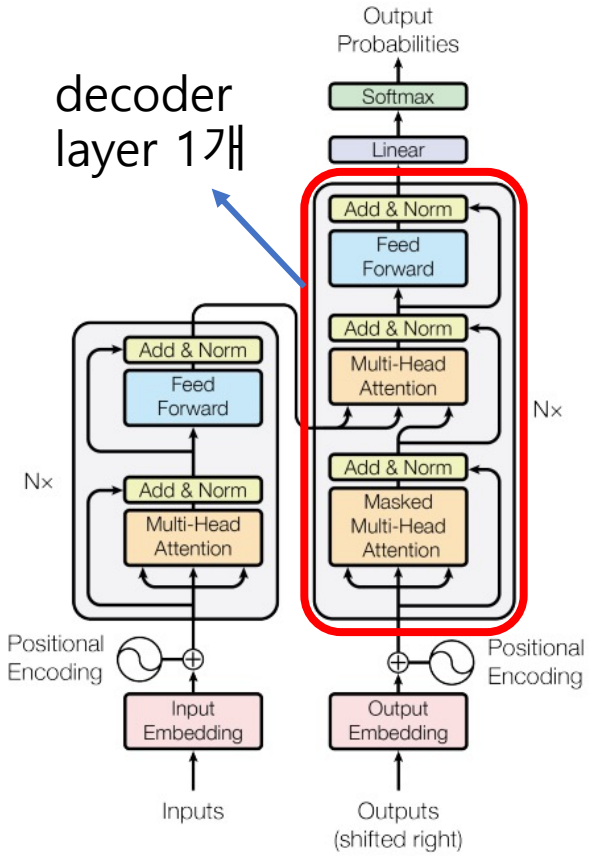


Figure 1: The Transformer - model architecture.

- N=6개의 동일한 layer로 구성
- 각 layer는 3개의 sub-layer로 구성
  - Masked multi-head self-attention sub-layer
  - Multi-head self-attention sub-layer on Encoder stack output
  - Simple, position-wise fully connected feed-forward network
- 각각의 sub-layer에 residual connection과 layer normalization 적용
- Masked multi-head self-attention sub-layer
  - Prevent positions from attending to subsequent positions
  - Position i에 대한 예측이 이미 알려진(Position이 i보다 작은) 값에만 의존

Attention

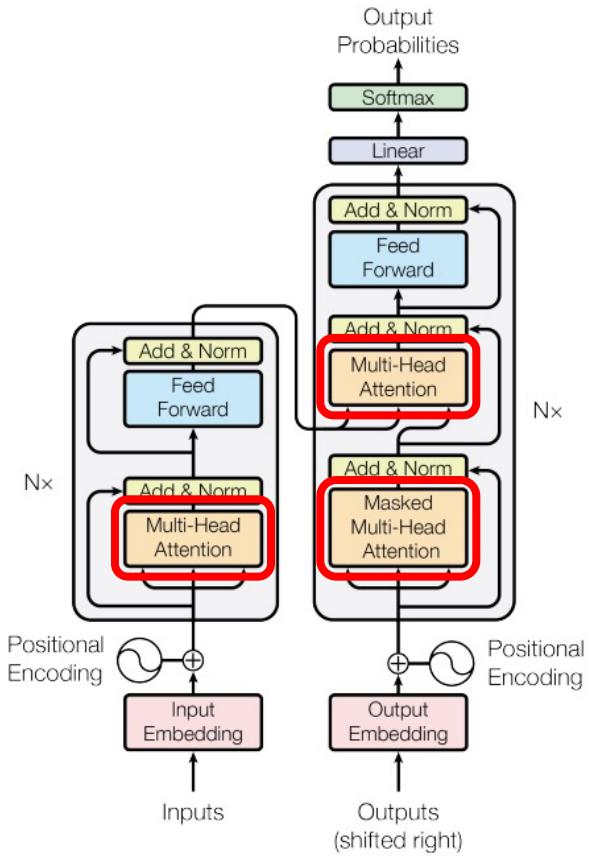
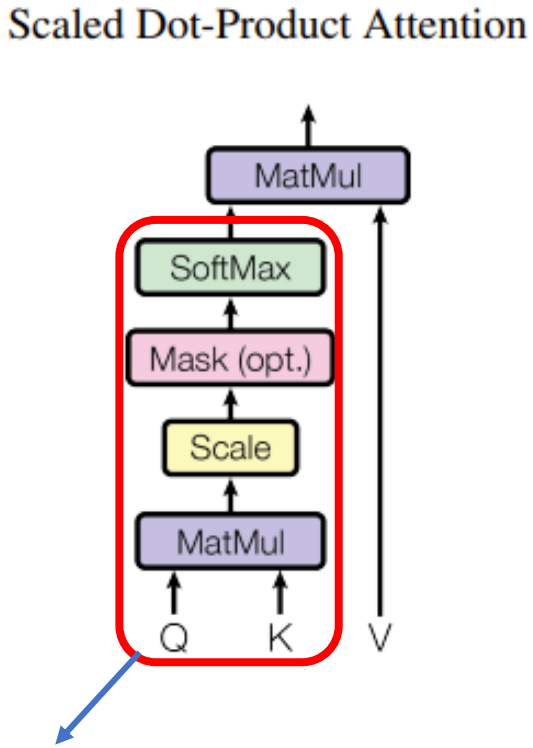


Figure 1: The Transformer - model architecture.

(1)Query와 (2)key-(3)value pair을 output으로 mapping하는 함수

- Query, key, value는 모두 vector
- Output은 value의 weighted sum
- Weight 값은 compatibility function에 의해 산출
  - Compatibility Function: query와 그에 대응하는 key의 함수
- 2 Types of Attention
  - Scaled Dot-Product Attention
  - Multi-Head Attention

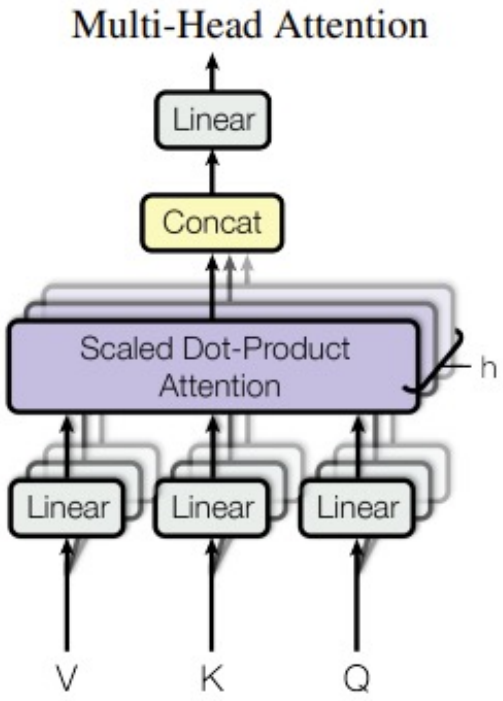
Attention: Scaled Dot-Product Attention



Compatibility Function
(output dimension d_v)

- Query와 key는 dimension d_k, value는 dimension d_v
- Weight 산출하기
  - Query와 모든 key들의 내적값들을 구한 후 $\sqrt{d_k}$로 나눈다
  - Softmax 함수 적용
- Matrix notation: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$
- Additive Attention vs. Dot-product Attention
  - Dot-product Attention은 위와 동일하지만 $\sqrt{d_k}$로 나눠서 scaling하지 않음
  - Additive Attention의 compatibility function은 hidden layer 1개인 feed-forward network
  - 작은 d_k 에 대해서는 성능 비슷, 큰 d_k 에 대해서는 성능 Additive가 나음
  - 이것을 상쇄하기 위해 $\sqrt{d_k}$로 나눈다

Attention: Multi-Head Attention



- Different, learned linear projection을 통해 queries, keys, values를 각각 dimension d_k, d_k, d_v로 h번 projection 시킴
- 각 queries, keys, values에 대해 scaled dot-product attention 함수 적용해서 d_v dimension의 output을 h개 얻음
- Matrix Notation:

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

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

- Allows model to jointly attend to information from different representation subspaces at different positions

Feed-Forward Network

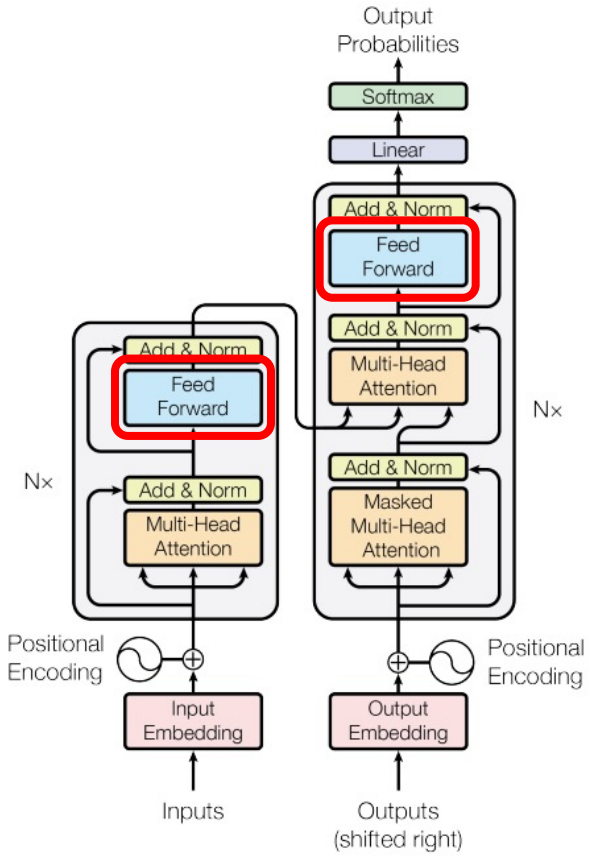


Figure 1: The Transformer - model architecture.

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

- Layer마다 다른 parameter 값을 사용

Embedding, Softmax and Linear Layer

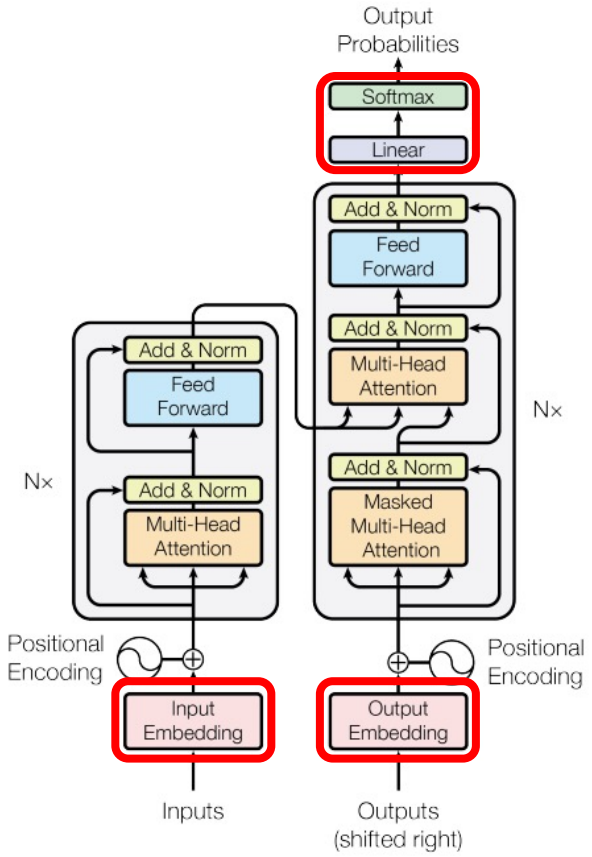


Figure 1: The Transformer - model architecture.

- 학습된 embedding 사용해서 d_{model} dimension 벡터로 변환
- 학습된 linear transformation과 softmax 함수 사용해서 decoder output을 next-token 확률 예측값들로 변환
- 여기서 사용한 모형에선 2개의 embedding layer과 linear transformation layer이 같은 weight matrix 공유
- Embedding Layer에선 weight들을 $\sqrt{d_{model}}$로 곱함

Positional Encoding

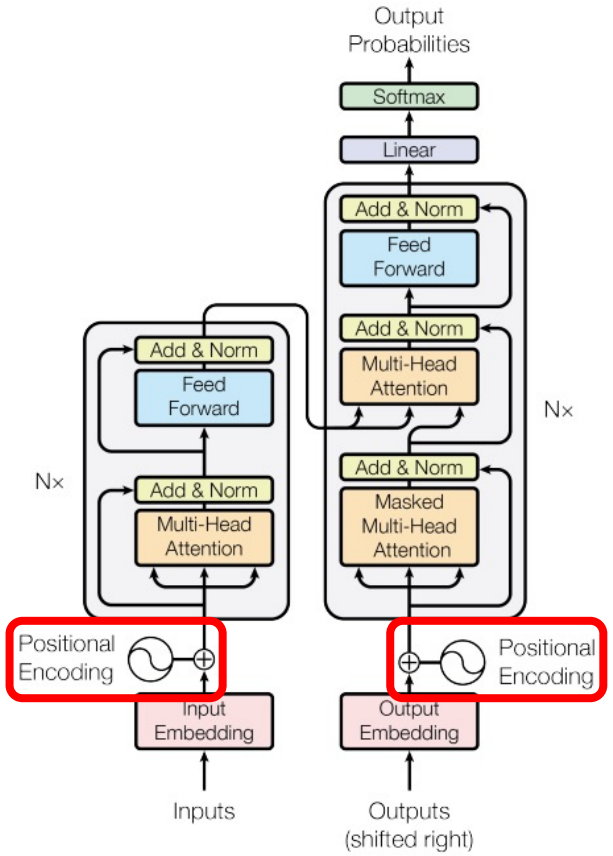


Figure 1: The Transformer - model architecture.

- 모델에 recurrence와 convolution이 없음
  - token들의 상대/절대적 position에 대한 정보를 넣어줘야 함
- Embeddings에 "positional encoding"을 더함
  - Positional encoding은 embedding과 더할 수 있도록 같은 d_{model} dimension을 가짐
- 여기서 사용한 positional encoding $PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$ $PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$
- 학습된 PE도 사용해봤지만 거의 비슷한 결과 도출됨

Why Use Self-Attention?

- Attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence

- 3가지 관점에서의 self-attention의 필요성
  - Total computational complexity per layer
  - Amount of computation that can be parallelized
  - Path length between long-range dependencies in the network

- Side benefit: self-attention은 상대적으로 interpretation이 쉬운 장점이 있다

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Conclusion

- Transformer은 recurrent/convolutional layer에 기반한 모델에 비해 훈련시간이 상당히 짧음
- Translation task에서 이전 모든 모형들에 비해 우수한 성능을 보임
- 앞으로의 목표
  - Transformer의 사용을 text 외로 확대하여 이미지,음성,영상 등의 큰 입력/출력값들을 다루기 위한 local, restricted attention mechanism에 대한 연구
  - Generation을 덜 sequential하게 만들기