# Attention Is All You Need

Status | Papers

https://github.com/edwinjungwoo/papers/blob/main/attentionisallyouneed.pdf

## “Attention Is All You Need”

(read on 06/13/22)

### Abstract

Sequence transduction model은 encoder와 decoder를 포함한 복잡한 recurrent or convolutional neural networks에 기초하고 있습니다. 가장 성능이 좋은 sequence model도 attention mechanism을 이용하는 encoder와 decoder의 구조를 따릅니다.

본 논문은 오로지 **attention mechanism**에 기초하는 간단하고 강력한 **Transformer**라는 새로운 아키텍처를 제안합니다.

기계 번역 작업에 대한 실험 결과에 따르면, Transformer는 병렬 처리가 가능하여 학습 시간이 크게 단축되는 동시에 좋은 성능을 가진 것으로 나타납니다. 또한, 크고 제한된 훈련 세트로 영어 문법 해석에 성공적으로 적용됨에 따라, 다른 tasks에도 좋은 일반화 성능을 가짐을 알 수 있습니다.

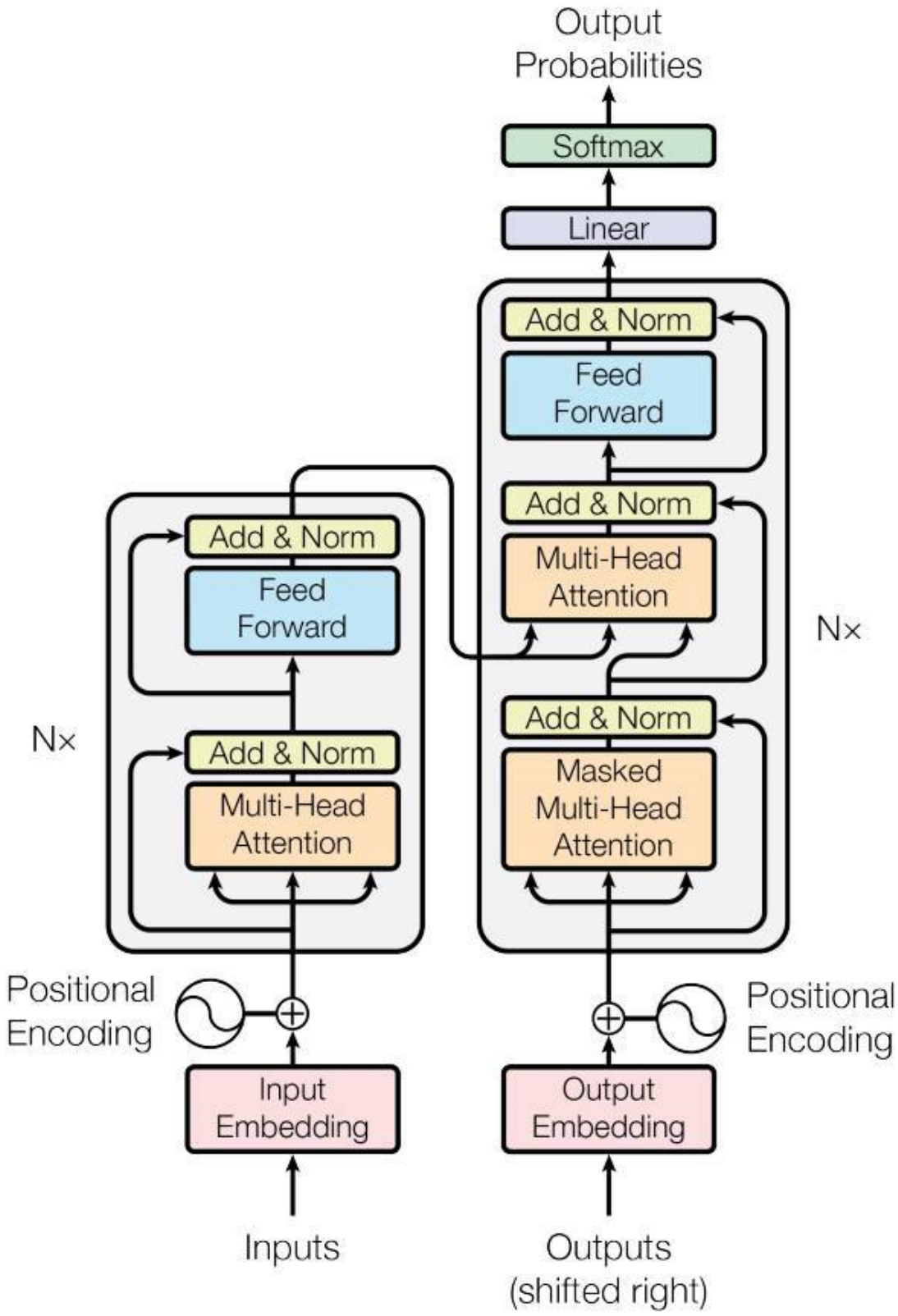
### Introduction

언어 모델링 및 기계 번역과 같은 sequence modeling에서 RNN, LSTM, GRU의 적용은 지배적입니다.  Recurrent model은 데이터에 대한 병렬처리가 불가능하다는 근본적인 제약이 존재

하지만 sequence modeling에서 대부분은 attention mechanism도 recurrent network와 함께 사용되었습니다.

앞서 말한 recurrent model의 근본적인 제약에서 벗어나고 입력과 출력 사이에 전역 의존성을 도출하기 위해 병렬화가 가능하고 높은 성능의 Transformer를 제안합니다.

## Model Architecture


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

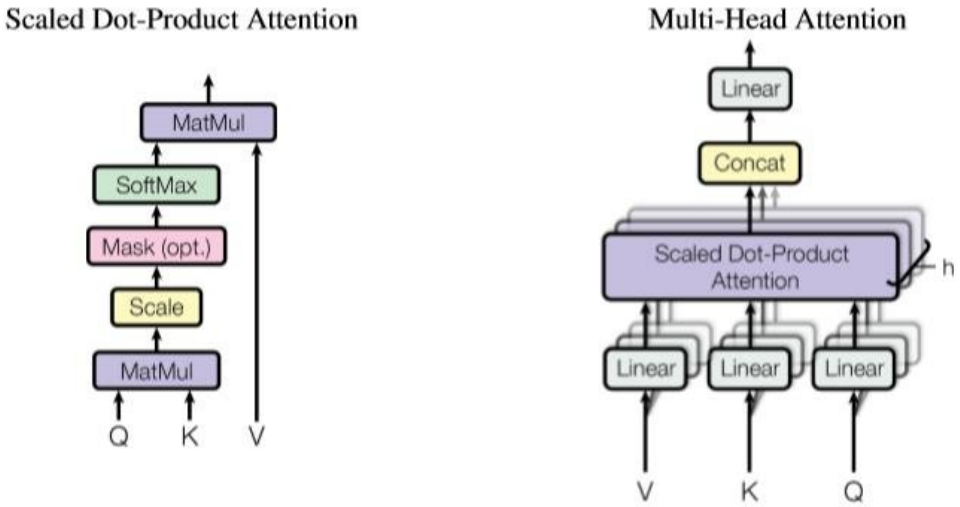
## Encoder

- Encoder는 N = 6개의 identical layers로 구성되어 있습니다. Input이 첫 layer에 들어가고 다음 layer는 이전 layer의 결과값을 받는 식으로 진행됩니다.
- 각 layer는 ‘multi-head attention’과 ‘feed forward’라는 2개의 sub-layers로 구성되어 있습니다.
- Sub-layers에 residual connection을 채택하며 layer normalization을 진행합니다.
- 각 sub-layers의 출력은 LayerNorm(x + Sublayer(x))이며, residual connection을 용이하게 하기 위해 model의 모든 layer는 d = 512로 embedding합니다.

## Decoder

- Decoder도 Encoder와 마찬가지로 N = 6개의 identical layers로 구성되어 있습니다. Residual connection을 수행하고 layer normalization을 진행하는 것도 동일합니다.
- Decoder는 각 층에 2개의 sub-layers 이외에도 Encoder 스택 출력을 통해 ‘multi-head self-attention’을 수행하는 또 다른 sub-layer를 삽입합니다.
- 또한, Decoder는 cheating을 방지하고 순차적으로 결과를 만들어야 하기 때문에 masking을 통해 position i보다 작은 output에만 의존합니다.

## Attention



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

scaled

입력은 dk 차원의 Query와 Key 그리고 dv 차원의 Value가 있습니다. (이하 Q, K, V)

Q와 K를 dot products하고 scaled된 dk로 나누어줍니다.

그 후, softmax 함수를 적용해 V에 대한 가중치를 구합니다.

- encoder-decoder attention의 경우, Q: decoder의 이전 layer hidden state, K: encoder의 output state, V: encoder의 output state
- self-attention의 경우, Q = K = V: encoder의 output state

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

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

dmodel-dimensional의 Q, K, V로 단일 attention function을 수행하는 것 보다 dq, dk, dv dimensions에 대해 학습된 Q, K, V Linear를 h번 연산하는 것이 vector의 크기를 줄이고 병렬처리할 수 있기 때문에 좋습니다.

각각의 head 즉 V, K, Q를 h개로 나눈 값들의 attention을 구하고 concatenate합니다.

Attention sub-layers 외에도 encoder와 decoder의 각 layer는 fully connected feed-forward network를 포함하며 이는 ReLU를 activation function으로 사용하는 두 번의 linear transformations로 구성됩니다.

Transformer는 RNN이나 CNN을 사용하지 않기 때문에 input sequence의 위치에 관한 정보를 주입해야 합니다. 따라서 encoder와 decoder stack 하단에 **positional encoding**을 추가합니다.

## Why Self-Attention

1. Layer당 연산량이 줄어듭니다.
2. 병렬처리가 가능한 계산이 늘어납니다.
3. 장거리 학습이 가능합니다.

| 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) | $O(r \cdot n \cdot d)$ | $O(1)$ | $O(n/r)$ |

## Training

WMT 2014 English-German dataset을 사용했고 NVIDIA P100 GPUs 8대로 기본 모델은 12시간 동안 big 모델은 3.5일간 학습시킨 결과입니다.

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | **$3.3 \cdot 10^{18}$** | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |

성능에 비해 빠른 속도로 우수한 BLEU scores를 기록했고 EN-DE에서는 당시 SOTA (State-of-the-art)를 달성했습니다.

## Implementation