

Attention

0. Abstract

1. attention mech + recurrent X = transformer
conv X

1. Introduction

1. Recurrent Neural net
Long Short-Term Memory
Gated Recurrent NN

) "Recurrent Model"
(RNN 등)

→ $h_t = f(h_{t-1}, x_t)$: 순차 의존성
→ x_t : 순차적 차이
↳ batch size seq 길이가 같은 미션 가능

2. Attention Mech

↳ only: Transformer (recurrent net 과 결합 X)

2. Background

1. 위치 i의 정보가 위치 j로 전달하는데 계산 2m² 상 가치는 연산 수↓

but attention-weighted positions 를 이용해 해결 가능

→ Multi-head Attention

2. self-attention (intra-attention)

→ m seq (m size 다른 위치들을 서로 연결 (ex. 문장 내同一 단어를 다른 단어들과 모두 연결))
문장 token

3. Model Architecture

3.1 Encoder and Decoder stacks

1. 입력 seq $(x_1, \dots, x_n) \rightarrow z = (z_1, \dots, z_m) \rightarrow y = (y_1, \dots, y_m)$

encoder decoder
순차적 축적 생성

↳ autoregressive (자기 회귀) : 이전 단계 출력을 이용하여 t의 입력으로 활용하여 출력하기
(정상 decoder나 seq2seq, transformer에서 autoregressive decoder 사용)

2.

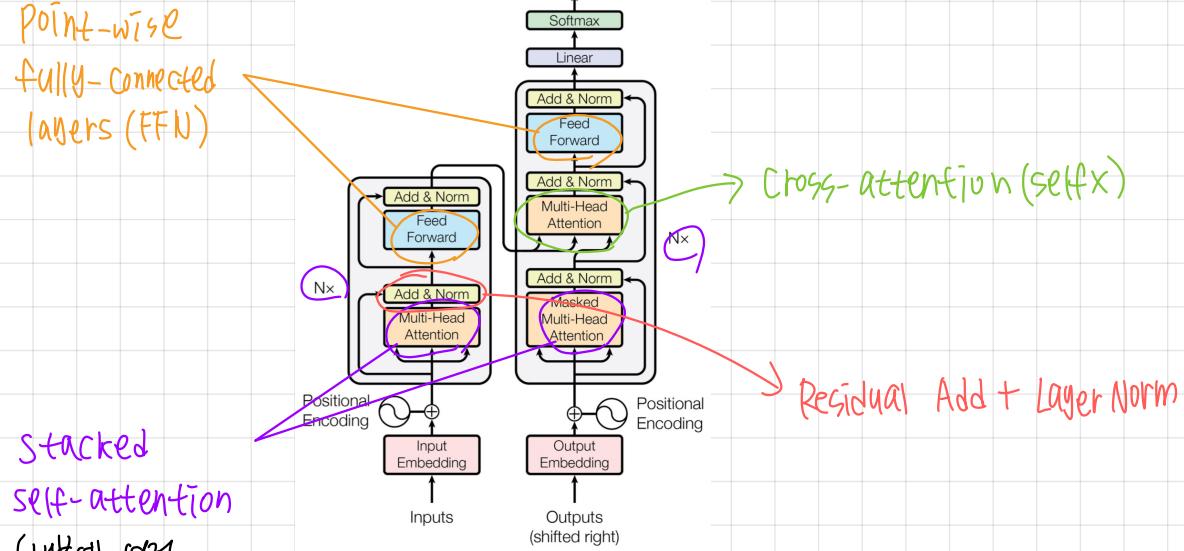


Figure 1: The Transformer - model architecture.

Stacked
self-attention
($\text{input of } 21$
self-attention
 $\frac{\text{blocks}}{\text{stack}}$)

encoder decoder

→ encoder

↳ 27n sub-layers → $N=6$

Multi-head self-attention
Point wise FFN (feed forward net)

→ 27n sub-layers Residual Add + Layer Norm
= Layer Norm (\downarrow sublayer(a))

↳ 전부 연결을 풀이하기하기하기위한 $d_{\text{model}} = 512$

→ decoder

↳ 27n sub-layers → $N=6$

Cross-attention : encoder의 2-layers add-Norm \Rightarrow Multi-head attention + Residual Add + Layer Norm
Masked multi-head attention (auto-regressive 예상)
output \times

3.2 Attention

3.2.1 Scaled Dot-Product Attention

1. query, key-value (값 셈)

↳ $\text{sum} = \text{Weighted sum of values}$: key, query \Rightarrow compatibility (tương ứng) \rightarrow 계산
= "Scaled Dot-Product Attention"

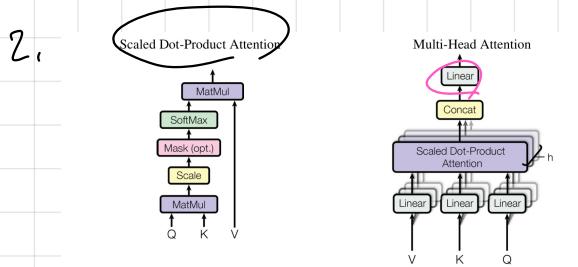


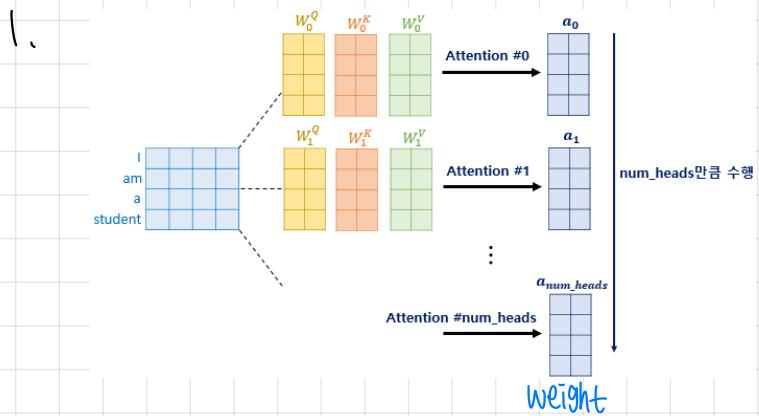
Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

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

↳ Q (Query) : d_k dim
 K (key) : d_k dim
 V (value) : d_v dim

내적 (dot-product) : 행렬 곱셈(행렬의 합성)
scaled : $d_k \uparrow$ (내적 크기 \uparrow (softmax 시 기울기 소실) \Rightarrow Scaling) +) additive attention, (not-scaled) dot-product

3.2.2 Multi-Head Attention



ex) 나는 오늘 점심을 먹었어.

↳ T=5 (토론)

1) embedding : 각 token 별 d-model 차원의 실수를 가지는 벡터로 + Positional encoding

$\rightarrow \mathbf{X} \in \mathbb{R}^{Tx\text{d}_{\text{model}}}$: 이미지 차원

2) Linear projection weight: token embedding \mathbf{q} + \mathbf{k} + \mathbf{v} head by \mathbf{W}_Q , \mathbf{W}_K , \mathbf{W}_V ($Q, K, V \in \mathbb{R}^{d \times d_h}$)

$$\rightarrow Q = X \cdot W^Q = d_K \dim$$

$$k = X \cdot W^K = d_K \dim \quad (d_K, d_V = \frac{d_{\text{model}}}{h})$$

$$V = X \cdot W^V = d_V \dim$$

3) Attention score & weighted-sum

→ Attention score: $S = \frac{QK^T}{\sqrt{d_k}}V$

→ **Weight**: $A = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$ (head および softmax)

→ Weighted-Sum : $Z = \sum_{T \times T} A \cdot V_{d_v \text{ 차원}} \Rightarrow \text{크기 } B \times T \times d_v$

4) Concat : head $\oplus_1 \dots \oplus_d$ concatenate \Rightarrow

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

2. ↳ M을 다른 선형 특성 (hH = multi-head 개수)

= M을 다른 representation subspace

= K를 다른 위치에 attend (\because head 1은 3번에 가능, head 2는 동사 ...)

\Rightarrow "방법"

$$\left. \begin{array}{l} W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k} \\ W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k} \\ W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v} \\ W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}} \end{array} \right) \begin{array}{l} h = \text{head 수} \\ (= 8 \text{ 개}) \\ d_k = d_v = \frac{d_{\text{model}}}{h} \\ (= \frac{512}{8} = 64 \text{ 개}) \end{array}$$

3.2.3 Applications of Attention in our Model

1. ① Multi-head attention

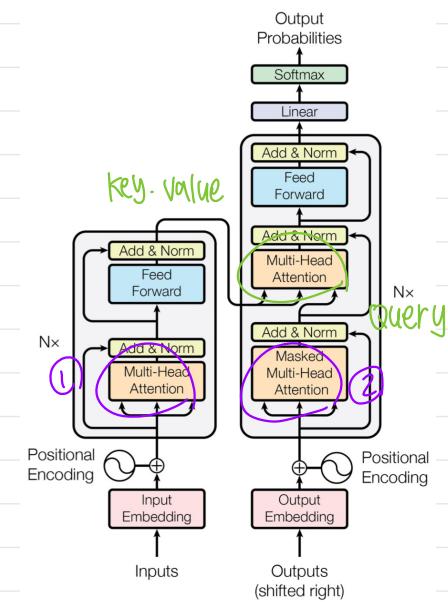


Figure 1: The Transformer - model architecture.

1) Encoder-Decoder attention

\rightarrow Query(직전 decoder 풀) + key·value(encoder의 출력)

\hookrightarrow decoder의 모든 위치가 입력 seq의 모든 위치에 attention

2) ① Encoder self-attention

\rightarrow encoder (N 층) 1 층별 Q, K, V 계산

\rightarrow "self-": Q, K, V 모두 원천이 X seq

\hookrightarrow 다음 층은 이전 층의 self-att + FFN + Add & Norm

\rightarrow encoder 각 위치는 encoder 직전 층의 모든 위치에 attend (masked x)

3) ② Decoder self-attention (Masked multi-head attention)

\rightarrow decoder 각 위치는 그 위치를 포함하여 그 이후 decoder的所有 위치에 attend (이후 position = 0/n token \rightarrow $-\infty$ 로 치우침)

masked = auto-regressive 위치

3.3 Point-wise Feed-Forward Networks

1. attention sub-layers 후 FFN

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

ReLU

$$= W_2 \cdot \text{ReLU}(xW_1 + b_1) + b_2$$

→ 한 layer 안에서 같은 FFN = 같은 가중치의 MLP
layer마다 다른 가중치 딜레이션

→ FFN은 position embedding이나 feature channel 흐름 $\approx 1 \times 1 \text{ Conv}$
한 번 (wx+b)

$$d_{\text{model}} = 512$$

≈ 256

$$d_{\text{ffn}} = 2048$$

3.4 Embeddings and Softmax

1. embedding layer 256 (256 parameter ≈ 256)

[encoder o_{256}] $\hookrightarrow \times \sqrt{d_{\text{model}}}$ \approx pos embedding token 범위를 축소하지 않도록 scaling
decoder o_{256}

3.5 Positional Encoding

1. recurrent X : seq 순서를 알 수 X
conv X

⇒ Positional encoding

↪ 차원 d_{model} 이거나 더 많거나 0

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

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

→ $\lambda = \pi/10000$

4. Why self-Attention

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

↪ 이웃을 뒤까지는 고려하지 않고 한

5. Training

5.1 Training Data and Batching, 5.2 Hardware and Schedule

5.3 Optimizer

```

1. optimizer = optim.Adam(
    model.parameters(),
    lr=lr,
    betas=(0.9, 0.98), # β1, β2
    eps=1e-9,          # ε
)
→ lrate =  $d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$ 

```

5.4 Regularization

1. 정규화

- (Residual dropout (0.1) : residual 정규화 dropout)
- (embedding + positional encoding 정규화 dropout)
- (label smoothing ($\epsilon_{\text{ls}} = 0.1$)

6. Results

(논문 참조)

7. Conclusion

Attention Visualizations

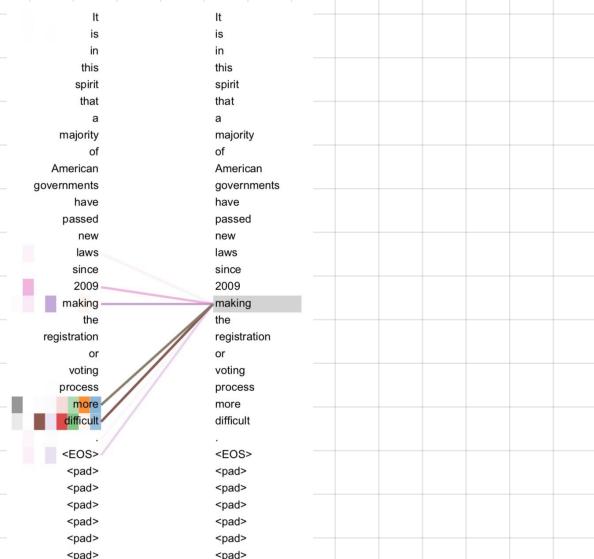


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 or 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

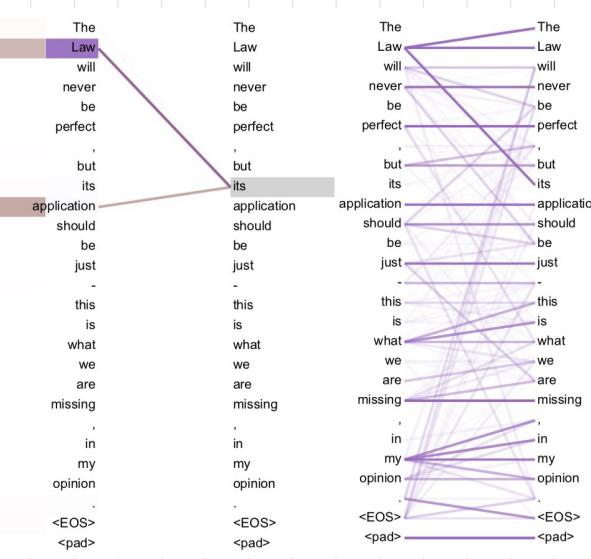


Figure 4: Two attention heads, also in layer 5 or 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

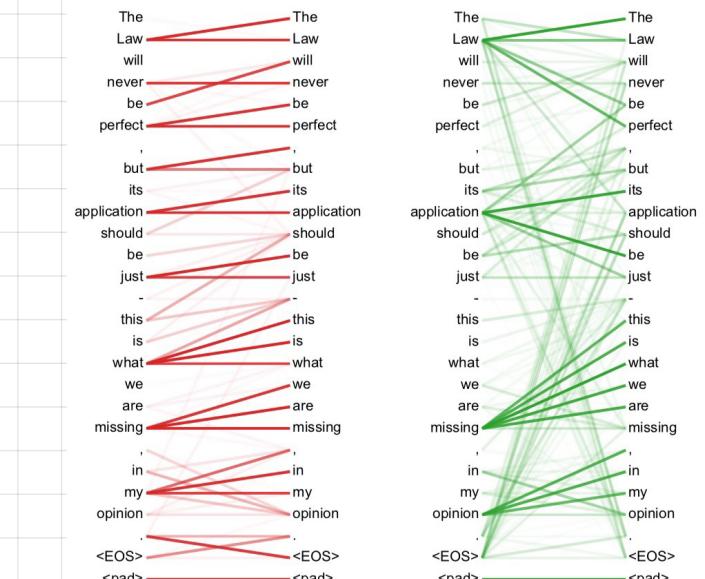


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 or 6. The heads clearly learned to perform different tasks.

↳ encoder 6 층 중 5 층의 self-att 히트
making(10)에 대한 멀리 떨어진 단어와 연결됨

↳ 5 층에서 its(11)에 대한 각각의 히트를 확인
head 5, 6의 힙당

↳ 문장 구조 파악

(장기(의논형))

↳ 같은 head