Attention

0. Abstract

1. attention mech + recurrent X / conv X = transformer

1. Introduction

1. Recurrent Neural net
   Long Short-Term Memory
   Gated Recurrent NN
   ⟩ "Recurrent model" (RNN 류)
   → $h_t = f(h_{t-1}, x_t)$: 순차 의존성
   → 병렬화 X (∵순차적 처리)
   ↳ batch 시 seq 길이가 길면 메모리 부족

2. Attention mech
   ↳ only: Transformer (recurrent net 과 결합 X)

2. Background

1. 위치 i의 정보가 위치 j로 도달하는데 계산 그래프 상 거리는 연산 수↓
   but attention-weighted positions 로 인해 해상도 감소
   → Multi-head Attention

2. self-attention (intra-attention)
   → 1개 seq (문장) 내 서로 다른 위치들을 서로 연결 (ex) 문장 내 1개 단어(token)를 다른 단어들과 모두 결합)

3. Model Architecture

3.1 Encoder and Decoder stacks

1. 입력 seq $(x_1, \cdots, x_n)$ → (encoder) → $z = (z_1, \cdots, z_n)$ → (decoder) → $y = (y_1, \cdots, y_m)$
   순차적 출력 생성
   ↳ autoregressive (자기 회귀) : 이전 단계 출력을 이번단계 t의 입력으로 활용하여 출력 생성
   (항상 decoder에서 사용 X, transformer에서 autoregressive decoder 사용)

2.

Point-wise fully-connected layers (FFN)

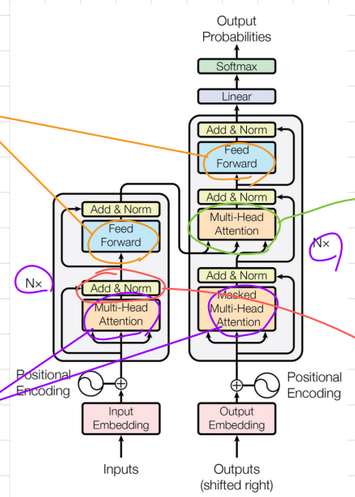


Figure 1: The Transformer - model architecture.

→ Cross-attention (self x)

Residual Add + Layer Norm

Stacked self-attention (내부에 여러 self-attention 블록 stack)

encoder decoder

→ encoder

↳ 2개 sub-layers → N=6

- multi-head self-attention
- point wise FFN (feed forward net)

→ 각각 뒤에 Residual Add + Layer Norm

= LayerNorm (x + Sublayer(x))

↳ 잔차 연결을 용이하게 하기위해 $d_{model} = 512$

→ decoder

↳ 2개 sub-layers → N=6

- Cross-attention : encoder의 2-layers add-Norm 후 출력은 multi-head attention + Residual Add + Layer Norm
- Masked multi-head attention (auto-regressive 보장)
  이후를 보지 X

3.2 Attention

3.2.1 Scaled Dot-Product Attention

1. query, key-value (모두 벡터)

↳ 출력 = weighted sum of values : key, query의 compatibility (호환성) 함수를 계산

= "Scaled Dot-Product Attention"

2.

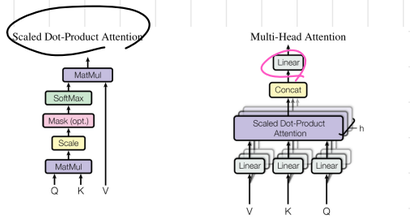


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

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

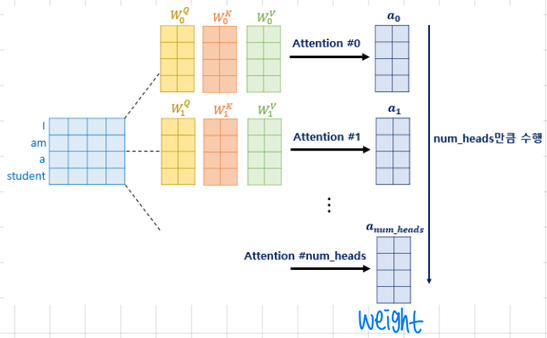
- QK^T → 내적 (dot-product) : 행렬 곱셈(메모리 효율)
- $\sqrt{d_k}$ → scaled : d_k ↑ 내적 크기 ↑ (softmax 시 기울기 소멸) ⇒ scaling
- softmax(...) → = weight

+) additive attention, (not-scaled) dot-product

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

3.2.2 Multi-Head Attention

1.



ex) 나는 오늘 점심을 먹었어.

↳ T=5 (토큰)

1) embedding : 각 token 별 d_model 차원의 실수를 가지는 벡터로 + positional encoding
→ $X \in R^{T \times d_{model}}$ = 임베딩 차원

2) Linear projection weight : token embedding 와 각 head 별 W 행렬곱 (Q,K,V 생성)
→ $Q = X \cdot W^Q = d_k$ dim
$K = X \cdot W^K = d_k$ dim $(d_k, d_v = \frac{d_{model}}{h})$
$V = X \cdot W^V = d_v$ dim

3) Attention score & weighted-sum
→ Attention score : $S = (\frac{QK^T}{\sqrt{d_k}})V$
→ weight : $A = \text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$ (head 개수 만큼)
→ weighted-sum : $Z = \sum A \cdot V$ ($T \times T$ · $T \times d_v$) (d_v 차원) ⇒ 최종 B × T × d_v (B: 배치)

4) Concat : head 별 Z 전부 concatenate ⇒ $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$
where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

5) Linear : 마지막 출력 projection

2. ↳ 서로 다른 선형 투영 (h개 = multi-head 개수)

= 서로 다른 representation subspace

= 서로 다른 위치 정보 attend (∵ head 1은 주어에 가중치, head 2는 동사 ···)

⇒ "병렬"

↳ $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

h = head 개수 (=8 선정)

$d_k = d_v = \frac{d_{\text{model}}}{h}$ (= $\frac{512}{8}$ = 64 선정)

3.2.3 Applications of Attention in our Model

1. 3가지 multi-head attention

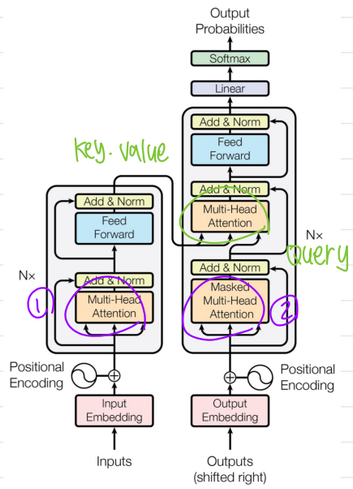


Figure 1: The Transformer - model architecture.

1) Encoder-Decoder attention

→ Query (직전 decoder 층) + key·value (encoder의 출력)

↳ decoder의 모든 위치가 입력 seq의 모든 위치에 Attention

2) ① Encoder self-attention

→ encoder (N층) 1층별 Q, K, V 계산

→ "self-": Q, K, V 모두 원천이 X seq

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

→ encoder 각 위치는 encoder 직전 층의 모든 위치에 attend (masked X)

3) ② Decoder self-attention (masked Multi-head attention)

→ decoder 각 위치는 그 위치를 포함하여 그 이전 decoder 내 모든 위치에 attend (이후 position = 미래 token → $-\infty$로 채우기)

masked = autoregressive 위함

3.3 Point-wise Feed-Forward Networks

1. attention sub-layers 후 FFN

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

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

→ 한 layer 안에서는 같은 FFN = 같은 가중치의 MLP
layer 바뀌면 가중치 달라짐

→ FFN은 position 섞임이 아닌 feature 차원(channel) 혼합 ≈ 1×1 conv 2개
선형 변환 2번 (Wx+b)

2. $d_{model} = 512$
$d_{ffn} = 2048$ 선정

3.4 Embeddings and Softmax

1. embedding layer 2개 (2개 parameter 행렬)
- encoder 입력
- decoder 입력

↳ $\times \sqrt{d_{model}}$ 로 Pos 정보가 token 정보를 압도하지 않도록 scaling

3.5 Positional Encoding

1. recurrent X, conv X : 순서 정보를 알 수 X

⇒ positional encoding

↳ 차원 d_model 이내의 더할 수 O

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

→ i = 차원

4. Why self-Attention

1. 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 (최대 경로 길이)
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 r개까지만 고려하도록 제한

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. 3가지 정규화

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

6. Results

(논문 참조)

7. Conclusion

Attention Visualizations

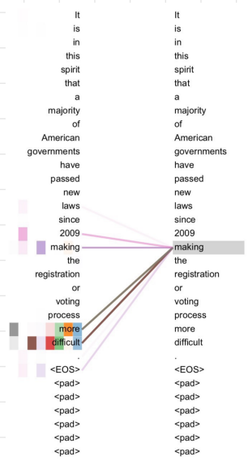


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word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

↳ encoder 6층 중 5층의 self-att 에서 making이 멀리떨어진 단어와 연결성

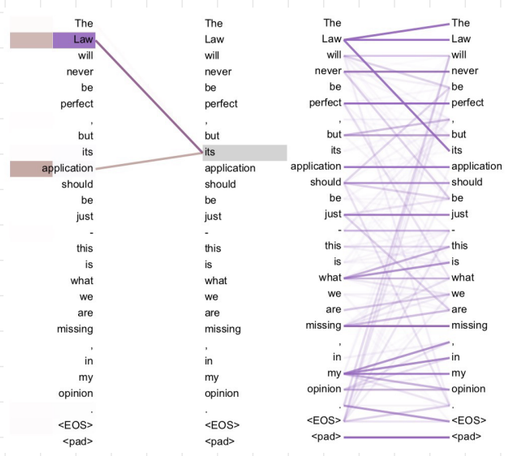


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

↳ 5층에서 its 에 대한 지시어 참조관계가 head 5, 6에서 담당

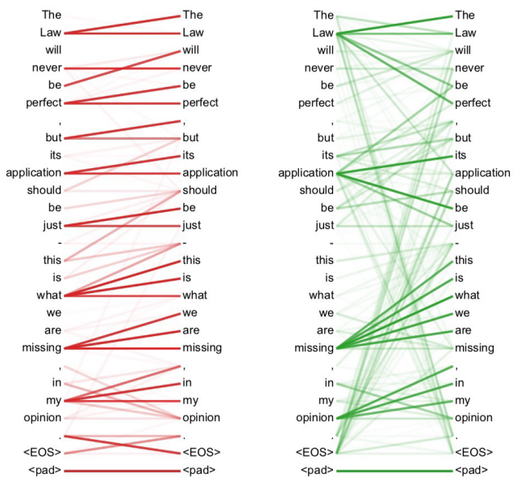


Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

↳ 문장 구조 파악

(장거리 의존성)

↳ 쌓은 head 만큼