Attention Is All You Need

2021-01-07

전세희

E-DEEP

Introduction

Sequence modeling, Transduction problems(ex. language modeling, machine translation) 에서 RNN(Recurrent Neural Networks), LSTM(Long Short-Term Memory), Gated Recurrent Neural Networks는 SOTA로 굳건히 자리 잡고 있음.

👉 But, 기존 Recurrent models은 이전 시간 state를 input으로 하기 때문에 sequential nature가지고 있음

👉 병렬화 불가

👉 문장의 길이가 길 때 큰 치명적.

(비록 현재 factorization을 통해 효율성을 확보한 모델도 있지만 여전히 sequential computation의 제약 조건이 남아있다.)

Introduction

<Attnetion>

다양한 task에서 필수 파트가 되었다. input이나 output sequence에서 dependencies의 거리에 상관없이 모델링함. But, recurrent network와 함께 사용됨.

👉🏻 Transformer 라는 모델 제안!

✔️ Recurrence 🙅🏻‍♀️

✔️ Attention만 이용

👉🏻 더 병렬화 가능, Translation에서도 적은 training 시간으로 SOTA 수준의 품질

Model Architecture

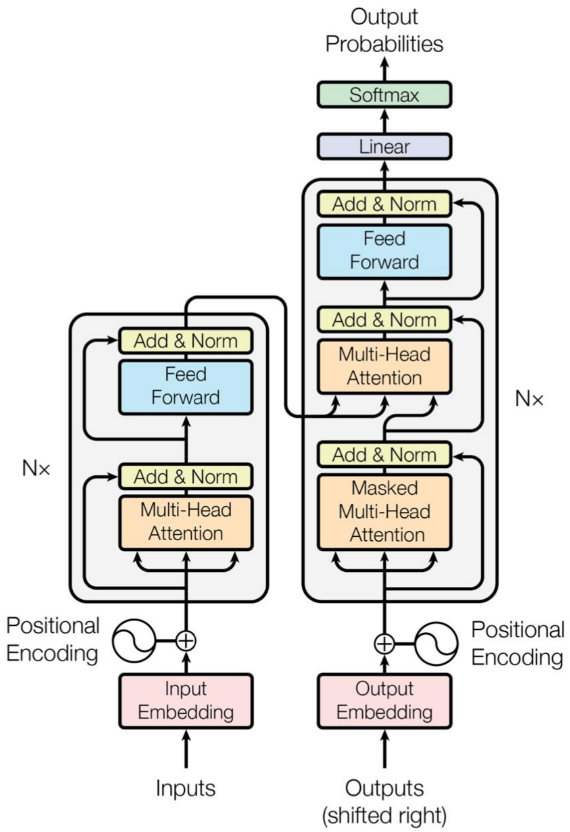


Figure 1: The Transformer - model architecture.

Model Architecture

- **Encoder**

  1. Each layer has 2 sub-layers
  (multi-head self-attention & position-wise fully connected feed-forward network)

  2. Residual connection & layer normalization

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

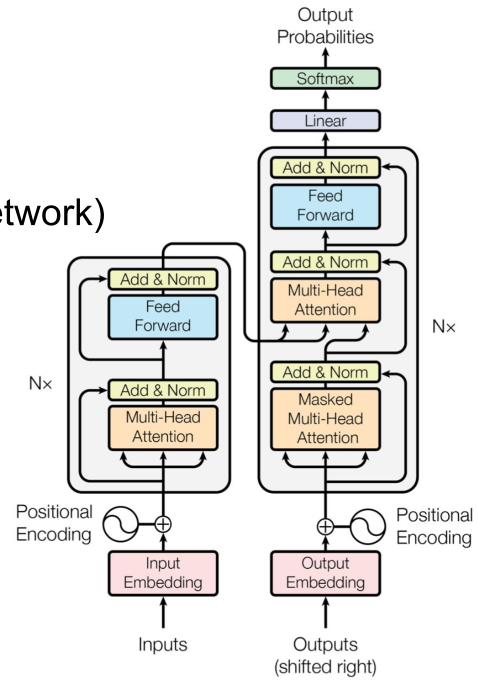


Figure 1: The Transformer - model architecture.

Model Architecture

- **Decoder**

1. multi-head self-attention & position-wise fully connected feed-forward network)

2. a third sub-layer, which performs multi-head attention over the **output** of the encoder stack.

3. Residual connection & layer normalization

4. Masking (for preventing positions from attending to subsequent positions)

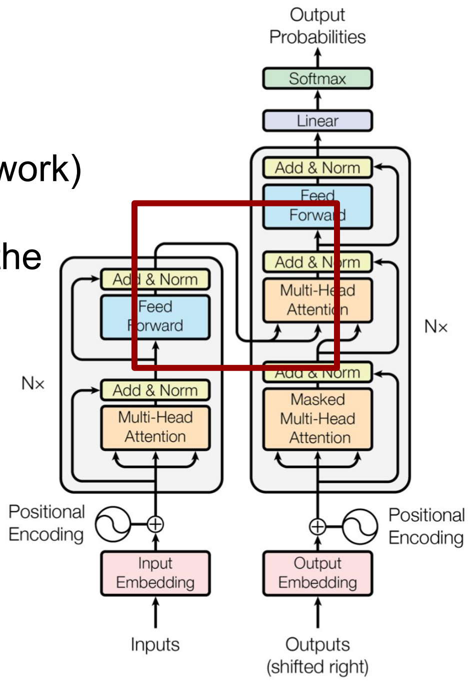
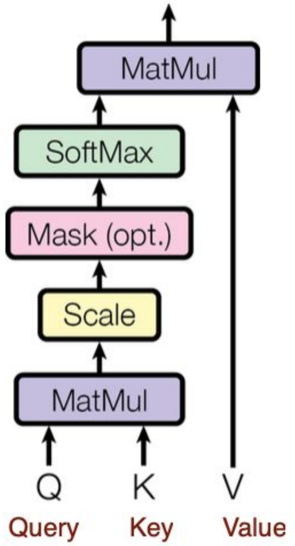


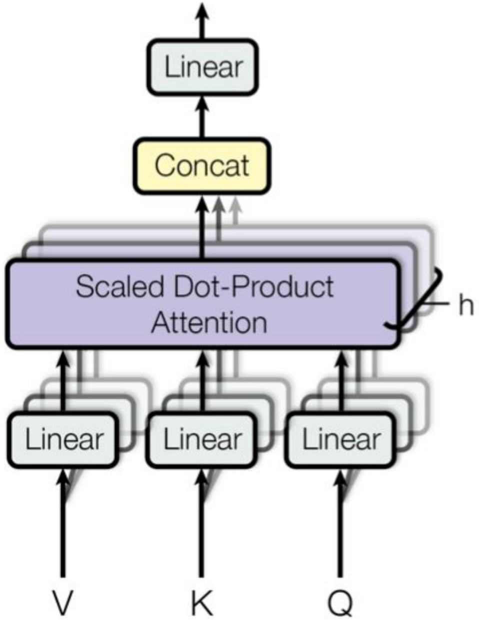
Figure 1: The Transformer - model architecture.

Attention

Scaled Dot-Product Attention



Multi-Head Attention

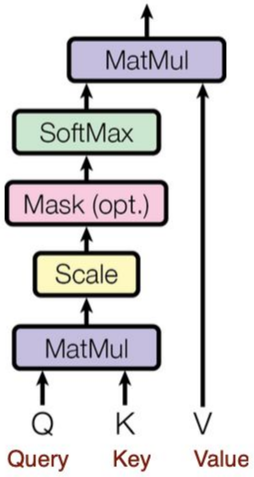


Attention

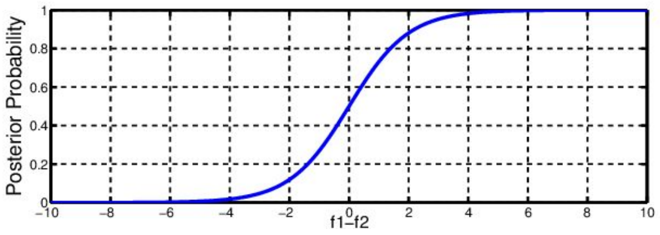
1. Scaled Dot-Product Attention

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

Scaled Dot-Product Attention



Q. scale하는 이유?



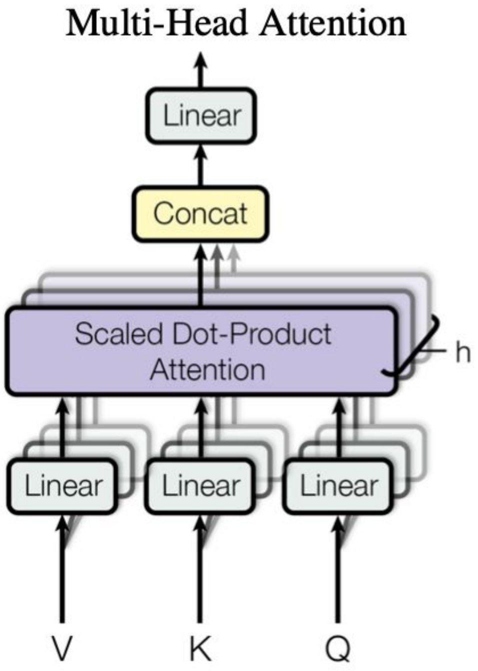
2. Multi-Head Attention

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1},...,\mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

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

$$QW_i^Q = [d_Q \text{x} d_{model}]\text{x}[d_{model}\text{x}d_k] = [d_Q\text{x}d_k]$$
$$KW_i^K = [d_K \text{x} d_{model}]\text{x}[d_{model}\text{x}d_k] = [d_K\text{x}d_k]$$
$$VW_i^V = [d_V \text{x} d_{model}]\text{x}[d_{model}\text{x}d_v] = [d_V\text{x}d_v]$$
$$\downarrow$$
$$Attention(QW_i^Q, KW_i^K, VW_i^V) = [d_V\text{x}d_v]$$
$$\downarrow$$
$$Concat(QW_i^Q, KW_i^K, VW_i^V)W^O = [d_V\text{x}hd_v]\text{x}[hd_v\text{x}d_{model}] = [d_V\text{x}d_{model}]$$

https://pozalabs.github.io/transformer/



3. Applications of Attention in our Model

The Transformer uses **multi-head attention** in three different ways:

- In "encoder-decoder attention" layers, **the queries come from the previous decoder layer**, and the **memory keys and values come from the output of the encoder.**

  👉🏻 Every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms.

- The encoder contains self-attention layers. In a self-attention layer all of the keys, **values and queries come from** the same place, in this case, **the output of the previous layer in the encoder**.

  👉🏻 Each position in the encoder can attend to all positions in the previous layer of the encoder.

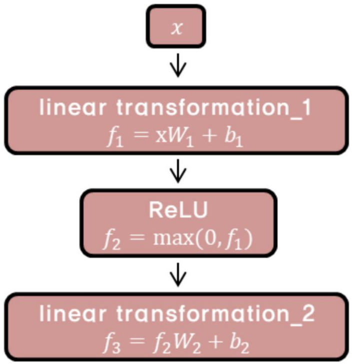
- Self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. 👉🏻Prevent leftward information flow in the decoder to preserve the auto-regressive property.

(토큰마다 적용되기 때문에 Position-wise)

Position-wise Feed-Forward Networks

Each of the layers in our encoder and decoder contains ***a fully connected feed-forward network***, which is applied to each position separately and identically.

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



https://pozalabs.github.io/transformer/

While the linear transformations are the same across different positions,

they use *different parameters from layer to layer.*

Positional Encoding

🙅🏻‍♀️ No Recurrence

🙅🏻‍♀️ No Convolution

👉🏻 **sequence의 순서정보는 어떻게 이용?**

In this work, we use sine and cosine functions of different frequencies:

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

왜 삼각함수를 사용? 👀

$$PE_{pos} = [cos(pos/1), sin(pos/10000^{2/d_{model}}), cos(pos/10000)^{2/d_{model}}, \dots, sin(pos/10000)]$$

- 이때 PE_{pos+k}는 PE_{pos}의 linear function으로 나타낼 수 있습니다. 표기를 간단히 하기 위해 $c = 10000^{\frac{2i}{d_{model}}}$ 라고 해봅시다.
  $sin(a+b) = sin(a)cos(b) + cos(a)sin(b)$이고
  $cos(a+b) = cos(a)cos(b) - sin(a)sin(b)$ 이므로 다음이 성립합니다.

$$PE_{(pos,2i)} = sin(\frac{pos}{c})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{c})$$

$$PE_{(pos+k,2i)} = sin(\frac{pos+k}{c}) = sin(\frac{pos}{c})cos(\frac{k}{c}) + cos(\frac{pos}{c})sin(\frac{k}{c}) = PE_{(pos,2i)}cos(\frac{k}{c}) + cos(\frac{pos}{c})sin(\frac{k}{c})$$

$$PE_{(pos+k,2i+1)} = cos(\frac{pos+k}{c}) = cos(\frac{pos}{c})cos(\frac{k}{c}) - sin(\frac{pos}{c})sin(\frac{k}{c}) = PE_{(pos,2i+1)}cos(\frac{k}{c}) - sin(\frac{pos}{c})sin(\frac{k}{c})$$

https://pozalabs.github.io/transformer/

❓ Why Self-Attention

1. computational complexity

2. 병렬화 가능한 계산의 양 👍

3. long-range dependencies사이의 path length

❓Why Self-Attention

3. long-range dependencies사이의 path length

많은 sequence transduction task에서 long-range dependency problem 존재

👉🏻 path가 짧을수록 long-range dependency를 러닝하기 쉬워짐
(즉 거리가 먼 토큰들 사이에도 관계를 찾기 쉬워짐)

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

1. computational complexity
👉🏻 대부분의 경우에 n < d 이기 때문에 self-attention의 complexity가 가장 작음
👉🏻 만약 n이 매우 큰 seq라면, 즉 길이가 매우 긴 seq의 경우에는 neighbor size를 r로 제한시켜 사용함.
(💢 그만큼 max path length가 O(n/r)로 커지는 문제가 있음)

2. 병렬화 가능한 계산의 양 👍 : Sequential하지 않기에 가능

3. long-range dependencies사이의 path length

Training

1. Optimizer : Adam Optimizer

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$

$$warmup\_steps = 4000$$

👉 warmup_steps까지는 learning rate가 linear하게 증가하다가 warmup_steps를 뛰어넘으면
step_num의 sqrt의 역수로 증가.

Training

2. Regularization

- **Residual Dropout**

👉 각 sub-layer의 output & the sums of the embeddings & positional encodings 에 적용

($P_{drop} = 0.1.$ for the base model)

- **Label Smoothing** **(모델이 덜 confident하게 만들어 overfitting방지)**

$$\epsilon_{ls} = 0.1$$

$$q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$$

Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Conclusion

"Transformer" : Based entirely on attiention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

😂 **Side benefit**

self-attention could yield **more interpretable models**