Attention is all you need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

논문 review

김소희

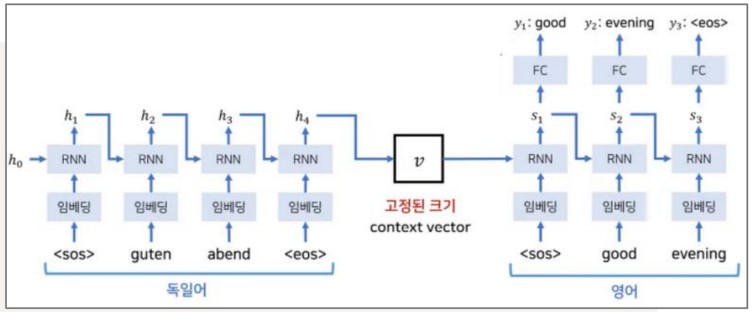
Abstract

- 기존의 sequence transduction model들은 복잡한 RNN / CNN 기반 모델들이다.
- 이제까지 가장 뛰어난 성능의 모델들 = encoder / decoder 구조 + Attention 메커니즘

- 본 논문에서는 recurrence와 convolution을 완전히 제거하고
  Attention 메커니즘에 전적으로 기반을 둔 Transformer 아키텍처를 제안한다.

- 두 가지 기계번역 task 에서 이 transformer 모델들은 기존 최상의 모델들보다 성능이 더 우월했으며, 병렬화를 통해 train 시간을 줄였다. 기계번역 뿐만 아니라 다른 task에서도 일반화가 잘됐다.

1. Introduction



- RNN, LSTM, GRU 등의 여러 딥러닝 모델 존재 : sequence 모델링에 효과적으로 이용
  - 그러나, Recurrent 한 모델들은 각 step에 따라 위치를 정렬시키고,
    이전 hidden state h_{t-1}과 t step의 input을 바탕으로 h_t를 생성한다.
  - 이렇게 순차적으로 처리하는 특성때문에 병렬 처리에 어려움이 있음
  - 메모리의 제약으로 Input sentence의 길이가 길어질수록 성능이 떨어짐
- Attention 기법 : input과 output sentence의 길이에 상관없이 dependency를 모델링 할 수 있음
  - 대부분 RNN에 결합되어 사용

- **Recurrence 특성 자체를 완전히 제거, Attention 기법에만 의존하는 transformer 모델 제안**
  - Transformer는 더 많은 병렬 처리가 가능
  - 8개의 P100 GPU를 사용했을 때 12시간만에 좋은 성능을 얻을 수 있다.

2. Background

- 이제까지 Sequential 연산을 줄이기 위해 Extended Neural GPU, ByteNet, ConvS2S 등 다양한 모델 제안
  - 이 모델들은 CNN을 사용하여 모든 input, output 위치에 대한 hidden representation을 병렬적으로 계산
  - 하지만, 이는 두 위치가 멀어지면 이를 계산하기 위한 추가적인 연산량이 증가 => 학습이 어려움

- **Self attention**
  - 특정 문장이 있을 때 자기 자신의 문장 스스로에게 attention을 수행해 학습하는 것
  - 한 시퀀스의 representation을 계산하기 위해 한 시퀀스 내에서 서로 다른 위치에 있는 요소들을 관련시킴
  - 예) I am a teacher → 4개의 단어는 서로에게 attention 수행, 가중치 부여

- Transformer는 RNN이나 CNN을 사용하지 않고 self-attention으로만 구성된 최초의 transduction(변환) 모델이다.

3. Model Architecture

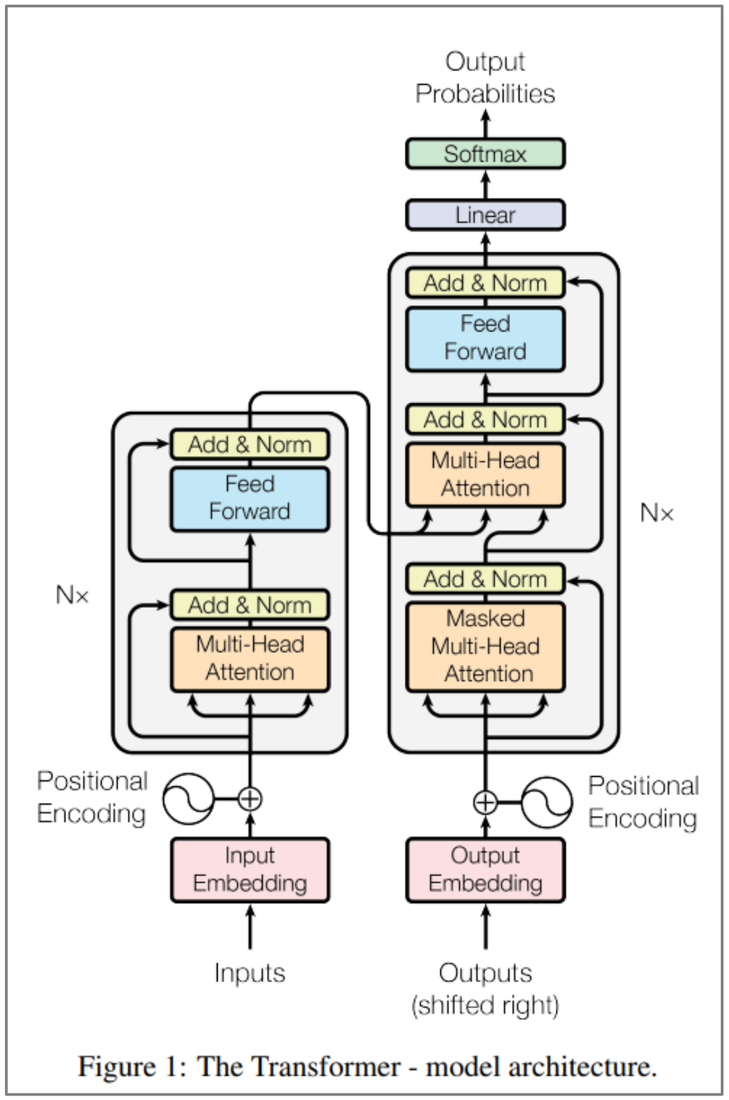


Figure 1: The Transformer - model architecture.

- 대부분 성능이 좋은 시퀀스 모델 : Encoder-Decoder 구조를 지님
  - Encoder : input 시퀀스 $(x_1, \dots, x_n)$ 를
    연속적인 representation인 $\mathrm{z} = (z_1, \dots, z_n)$로 매핑
  - Decoder : z에 대해 $(y_1, \dots, y_m)$ 의 결과를 제공

◆ **Transformer 모델** : 동일한 구조를 유지

- 모델을 recurrent하게 이용하지 않고 Self-attention 활용해 sequence에 대한 정보를 한번에 입력

3. Model Architecture – 1. Encoder and Decoder Stacks

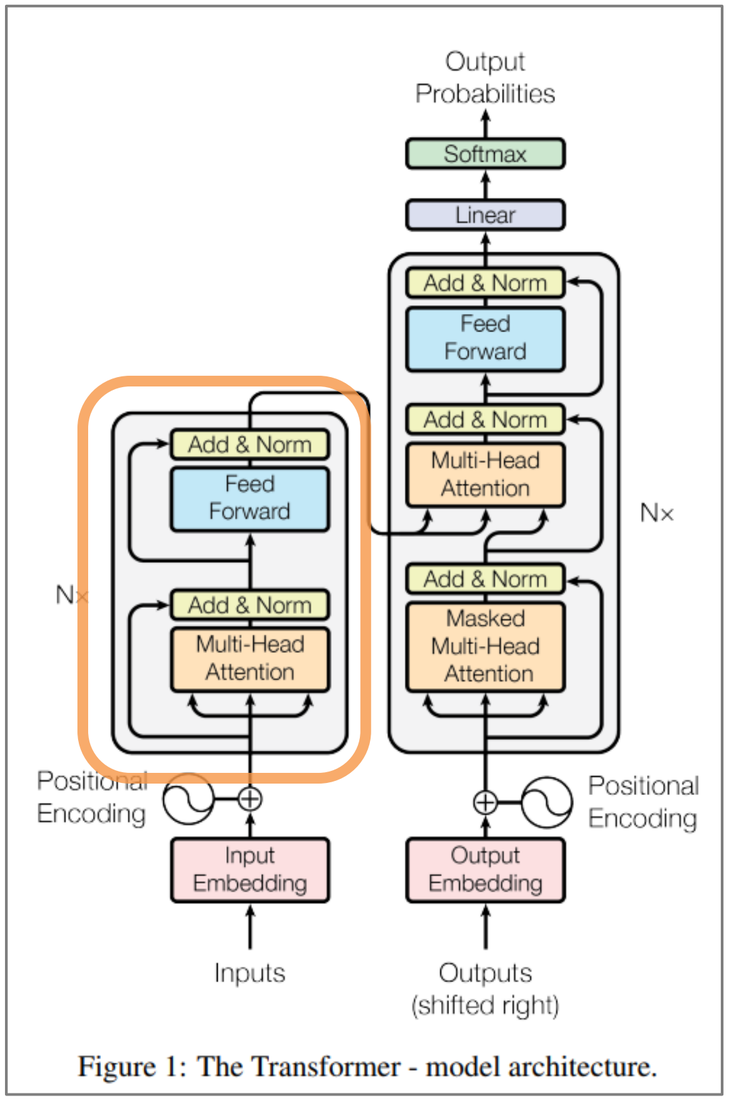


Figure 1: The Transformer - model architecture.

◆ Encoder

- N=6개의 동일한 layer로 구성
- 각각의 encoder는 Multi-Head Attention + position-wise fully connected feed-forward 2개 sub-layer로 구성
- 각 sub-layer마다 residual connection을 적용, layer normalization(정규화) 수행
  - Embedding층을 포함하는 모든 sub-layer의 output 차원 통일
    : d_{model} = 512

3. Model Architecture – 1. Encoder and Decoder Stacks

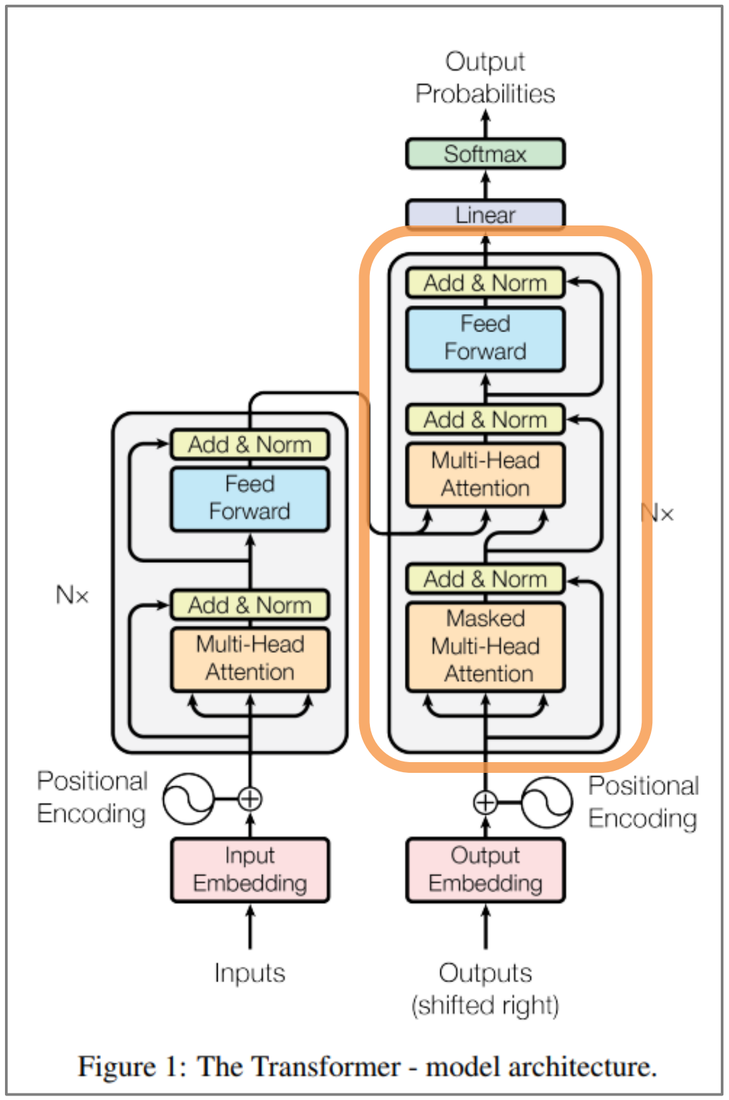


Figure 1: The Transformer - model architecture.

◆ Decoder

- N=6개의 동일한 layer로 구성
- Encoder 2개의 sub-layer + **multi-head attention 수행하는 sub-layer** 삽입
- 3개의 sub-layer에 각각 residual connection 적용, layer normalization 수행

➢ **Masked Multi-head Attention**

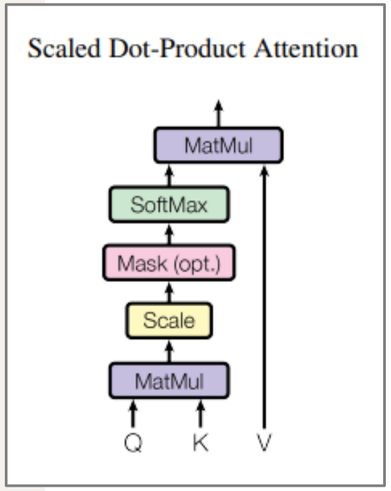
- **뒤의 입력들에 대한 정보를 미리 알지 못하도록** Self-attention 방식을 수정
- Masking 방식 : i번째 예측은 i번째 이전의 결과에만 의존적이게 함

3. Model Architecture – 2. Attention

- Attention은 query와 key-value의 쌍을 output에 mapping 하는 것 (query, key, value, output 모두 벡터)
- query와 그에 대응되는 key의 연관성함수에 의해 계산된 attention 가중치들로 value들을 weighted sum하여 최종 출력이 계산된다.

◆ Scaled Dot-product Attention

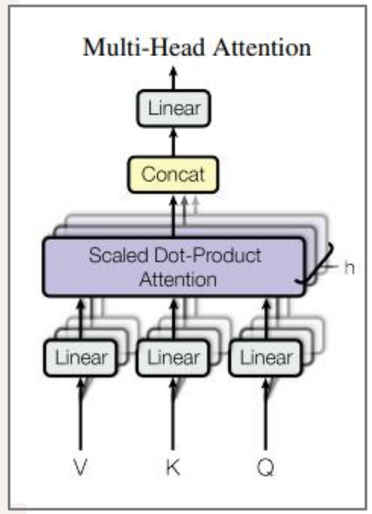
- Query와 Key는 d_k차원이고,
  value는 d_v차원으로 구성
- Query와 모든 Keys에 대한 dot-product를 구하고, $\sqrt{d_k}$로 나눔
- dot-product attention
  : 속도 빠르고 공간 효율적인 방법
- Scaling 하는 이유 : d_k가 커질 경우
  결과값도 커지는 경향이 있어, 결과적으로 softmax 함수에서 gradient가 매우 작아지게 됨
  => $\sqrt{d_k}$로 나눔



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

◆ Multi-Head Attention

- Query, Key, Value에 각각d_q, d_k, d_v차원의 linearly project를 병렬로 h번 수행
  => d_v차원의 output
- 이 값들은 concatenate해서 최종 결과로 사용
- Multi-head 모델이 서로 다른 domain들의 서로 다른 representation 부분 공간들로부터 결합적으로 정보에 접근하도록 한다.
- 본 논문에선 h=8개의 헤드 사용,
  각 헤드마다 $d_k = d_v = 64$ 로 지정



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

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

3. Model Architecture – 2. Attention

◆ Applications of Attention in our Model

- **Encoder self-attention layer**
  - key, value, query 모두 이전 인코더의 결과로부터 옴
  - 각 인코딩 층의 position은 이전 인코딩 층의 모든 position에 접근 가능

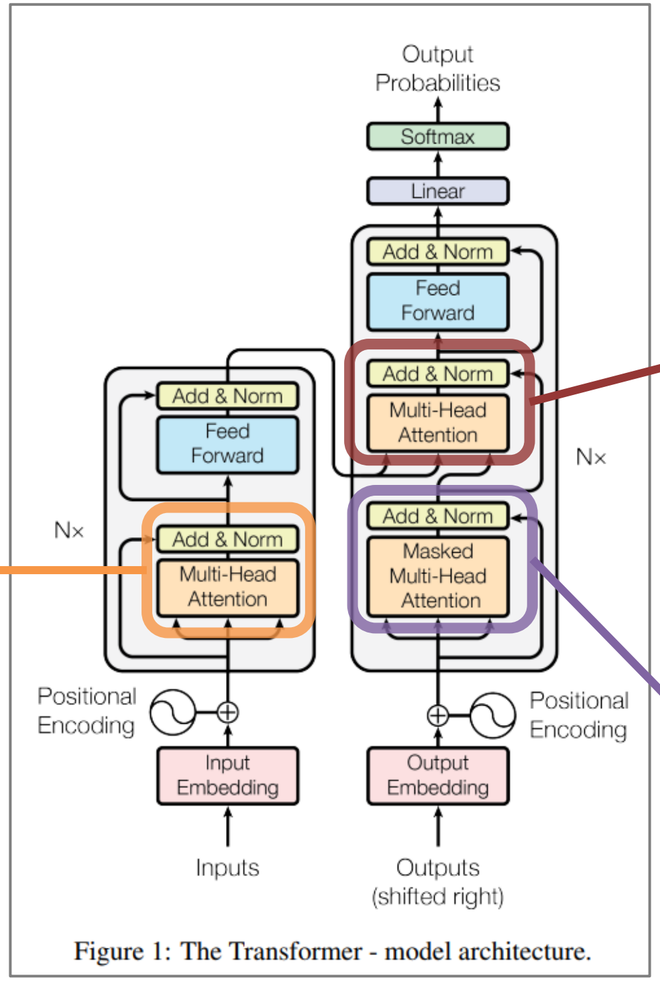


Figure 1: The Transformer - model architecture.

- **Encoder-decoder attention**
  - 인코더의 결과가 key, value, 이전 디코더의 결과가 query 역할 수행
  - 현재 디코더의 결과가 인코더의 모든 출력에 대해서 attention을 수행

- **Decoder self-attention layer**
  - Masking을 해서 미래의 모든 값들에 대한 attention 불가

3. Model Architecture – 3. Position-wise Feed-Forward Networks

- Encoder, decoder의 각 attention sub-layer는 모두 fully connected feed-forward network를 포함한다.
- 각 position 마다 독립적으로 적용되므로 position-wise다.

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

- 이와 같이 두개의 선형 변환과 ReLU activation function으로 구성되어 있다. (max 함수)
  - x에 선형 변환을 적용한 후 ReLU를 거쳐 다시 선형 변환을 한다.
- 선형변환은 position마다 같은 parameter를 사용하지만 layer가 달라지면 다른 parameter를 사용한다.
- 각 층의 Input과 output의 차원은 $d_{model} = 512$이고 inner-layer의 차원은 $d_{ff} = 2048$이다.

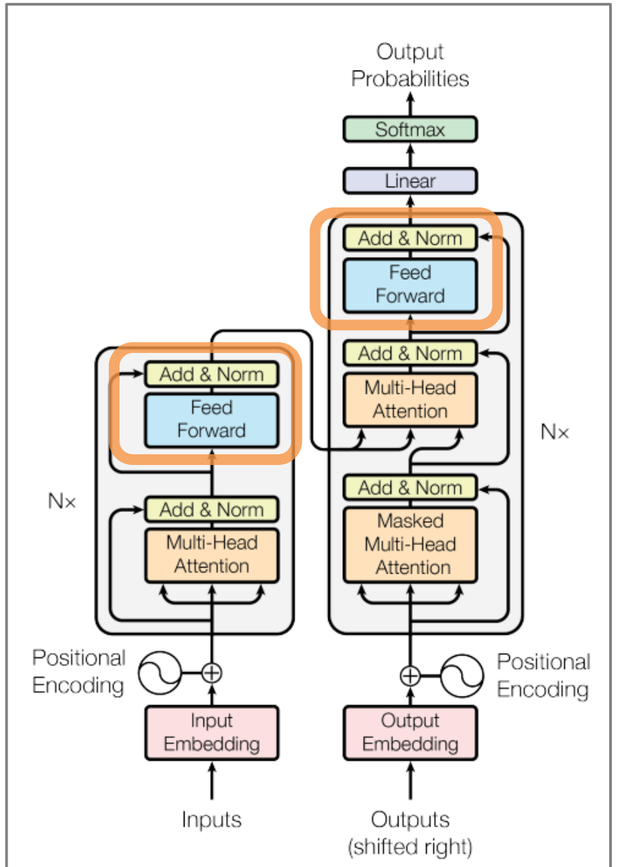


Figure 1: The Transformer - model architecture.

3. Model Architecture – 4. Embeddings and Softmax. 5.Positional Encoding

◆ Embeddings and Softmax

- 다른 sequence transduction model과 유사하게 input과 output 토큰을 $d_{textmodel}$차원을 갖는 벡터로 변환하여 embedding을 학습한다.
- 일반적으로 학습된 선형변환과 softmax 함수를 사용해 디코더의 output을 다음 토큰을 예측하기 위한 확률로 변환한다.
- 본 논문의 모델에서는 두 embedding layer와 softmax 이전의 linear transformation 에서 동일한 가중치 행렬을 공유한다. Embedding layer에서는 가중치에 $\sqrt{d_{model}}$을 곱한다.

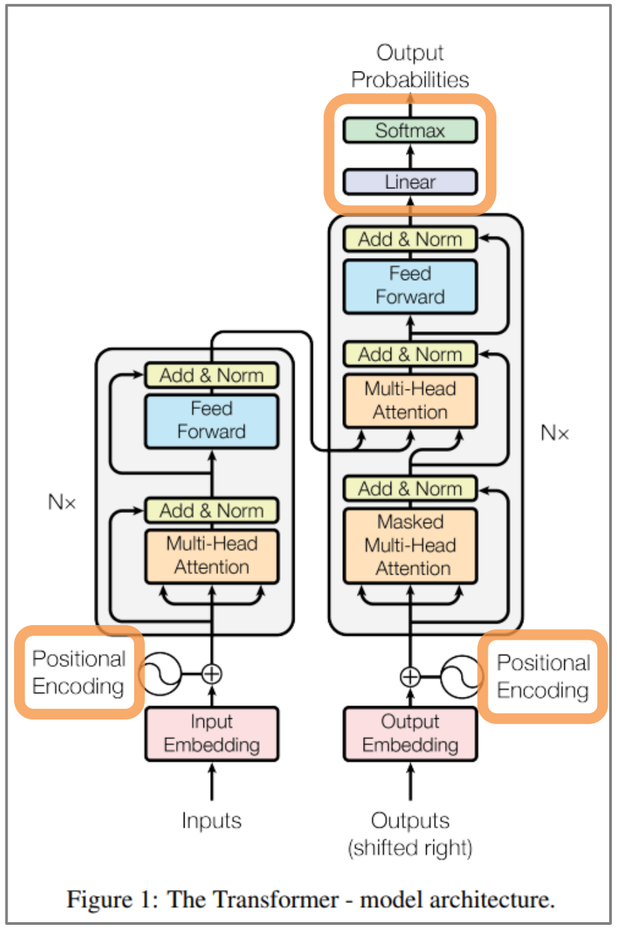


Figure 1: The Transformer - model architecture.

◆ Positional Encoding

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

- Transformer 모델은 recurrence나 convolution을 포함하지 않는다.
  - 따라서 위치 정보를 sequence 내의 토큰에 주입하여야 한다.
- 이를 위해 인코더와 디코더의 input embedding에 “positional encoding”을 더해준다.
- 이는 embedding과 더해질 수 있게 $d_{textmodel}$로 같은 차원을 갖는다.
- 본 논문에서는 다른 주기함수(sin, cos)를 사용하였다. (pos는 position, i는 차원을 의미한다.)
- 이 함수를 통해 상대적인 위치를 기반으로 attention을 더 쉽게 학습할 수 있다.
  - 그 어떠한 고정된 offset값 k에 대해서도 PE_{pos+k} 가 PE_{pos} 의 선형함수로써 표현될 수 있어서

4. Why Self-Attention

- Transformer의 핵심은 기존과 다르게 attention 메커니즘으로만 이루어져 있다는 점이다.
- 따라서, 왜 self-attention을 사용하는 것이 기존의 RNN, CNN보다 강점을 가지는지?
  1. Layer 별로 필요한 계산 복잡도가 줄어들었다.
  2. 순차적으로 진행되는 작업의 횟수가 감소하면서, 병렬적으로 진행할 수 있는 연산량이 증가하였다.
  3. 멀리 떨어진 원소들 사이의 path length가 감소하였다.
     - 각 토큰별로 일정한 연산량이 병렬적으로 이루어지기 때문에, long-range dependency에 구애받지 않고 빠르게 연산을 수행할 수 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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

1. 시퀀스길이 n < 임베딩 차원 d
   ⇒ 계산 복잡도 줄어듬

2. 상수의 복잡도
   ⇒ 선형 복잡도보다 좋음

3. 문장내의 모든 단어에 한번에 접근 = O(1)
   ⇒ 다른 것 보다 좋음

5. Training

◆ **Training Data and Batching**

- Standard WMT 2014 English-German dataset (4.5million 문장 쌍) 사용
- WMT 2014 English-French dataset도 사용
- 각 batch마다 약 25000개의 source token과 target token을 포함하는 문장 쌍을 가지도록 하였다.

◆ **Hardware and Schedule**

- 전체 모델을 8개의 NVIDIA P100 GPU를 이용해서 학습시켰다.
- 각 training step은 0.4초 정도 걸렸고, base model은 100,000step 동안 학습시켰다.
- Big model은 한 step에 1.0초가 걸렸고, 300,000step 동안 학습을 진행했다.

◆ **Optimizer**

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

- 학습하는 과정에서 Adam Optimizer를 사용했다.
- 옆의 식에서 Learning rate는 첫번째 warmup_steps동안 step_num에 비례하여 선형적으로 증가하고, 그 이후로는 step_num의 역제곱근에 비례하여 감소한다.
- 본 논문에서는 warmup_steps = 4000을 사용했다.

◆ **Regularization**

1. Residual Dropout - Sub-layer의 출력에 대해서 dropout을 적용했다.
2. 인코더와 디코더의 스택에 embedding과 positional encoding의 합에 대해서도 dropout을 적용했다.
3. Label Smoothing – 학습이 진행될 동안 $\epsilon_{ls} = 0.1$ 값을 갖는 label smoothing을 적용했다.

6. Results

◆ Machine Translation

- WMT 2014 English-to-German translation task
  – big transformer 모델은 BLEU 점수를 2.0정도 증가시킴
- WMT 2014 English-to-French translation task
  – big transformer 모델은 ¼의 학습비용만으로 이전에 최고 성능의 모델들을 뛰어 넘었다.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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	41.0	$2.3 \cdot 10^{19}$	

◆ Model Variations

- Transformer의 다양한 요소들의 중요성을 파악하기 위해 base model을 다양한 방식으로 수정했다.
  (English-to-German translation 성능 기반으로 평가)
- (A) : head가 너무 많으면 오히려 성능이 떨어진다.
- (B) : key size가 모델의 성능에 영향을 미침
- (C),(D) : 큰 모델들의 성능이 더 좋고, overfitting을 예방하기 위해 dropout이 효과적
- (E) : 기존의 sinusoids 대신 positional embedding으로 교체
  => 기존의 성능과 유사했다.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

7. Conclusion

- Transformer는 Attention 기법만 이용한 최초의 Sequence transduction model이다.
- 기존의 encoder-decoder 구조에 존재하던 recurrent한 요소들을 모두 multi-headed self-attention으로 대체했다.
- Translation task에 대해서는 recurrent, convolutional layer에 기반을 둔 구조들보다 훈련이 유의미하게 빨랐으며, 기존의 다른 모델들보다 성능이 월등히 높았다.
- Attention based model 들이 다른 다양한 task들에 적용되기를 기대한다.