Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit

Ⅰ. Introduction

RNN, 특히 LSTM과 GRU는 sequence modeling이나 machine translation과 language modeling 같은 transduction problems에서 state-of-art로 자리잡았다. Recurrent 모델의 순차적인 특성은 학습에서 병렬화를 제한하는데 메모리의 제약으로 batch에도 제한이 생겨 sequence의 길이가 길어질수록 문제가 생긴다. 다른 연구에서 factorization trick이나 conditional computation을 통해 연산 효율을 향상시켰으나 sequence 연산의 기본 제약은 여전히 남아있다.

Attention은 input, output sequence 길이에 상관없이 dependency를 모델링해줄 수 있어서 sequence modeling이나 transduction modeling에서 중요한 요소가 되었다. 그러나 대부분 RNN에 결합되어 사용된다.

이 논문에서는 recurrence를 피하고 input과 output 사이의 global dependency를 찾는 attention 메커니즘만 사용하는 Transformer를 제안한다. Transformer는 더 많은 병렬 처리가 가능하고 state-of-art 수준을 보인다.

Ⅱ. Background

Sequence 연산을 줄이기 위해 Extended Neural GPU, ByteNet, ConvS2S 등이 제안되었다. 이 모델들은 모두 CNN을 기본으로 사용하여 모든 input과 output 위치에 대한 hidden representation을 병렬로 계산한다. 그러나 임의의 input 또는 output을 연결하는데 필요한 number of operation은 거리에 따라 증가한다.(ConvS2S는 linear하게, ByteNet은 logarithmical하게) 이는 distant position에 있는 dependency를 학습하는데 어려움이 있다. Transformer에서는 attention-weighted positions을 평균화함에 따라 number of operation이 일정한 수로 줄어든다. 하지만 effective가 감소하는데 이는 Multi-head Attention으로 극복한다.

Self-attention(intra-attention)은 sequence의 representation을 계산하기 위해 단일 sequence의 서로 다른 위치들을 관련시키는 attention 메커니즘이다. Self-attention은 reading comprehension, abstractive summarization, textual entailment, learning task-independent sentence representations 등 다양한 작업에서 사용된다.

End-to-end memory network는 sequence-aligned recurrence 대신 recurrent attention 메커니즘을 기반으로 simple-language question answering과 같은 language modeling task에서 좋은 성능

을 내었다.

그러나 Transformer는 sequence aligned RNNs 또는 CNN을 사용하지 않고 input 및 output의 표현을 계산하는데 self-attention에 완전히 의존하는 최초의 transduction model이다.

Ⅲ. Model Architecture

뛰어난 성능을 가진 neural sequence transduction model은 encoder-decoder 구조로 이루어져 있다. Encoder는 input sequence $(x_1,...,x_n)$을 continuous representation인 z=$(z_1,...,z_n)$으로 변환한다. 주어진 z에 대해 decoder는 output sequence $(y_1,...,y_m)$을 하나씩 생성한다. 각각의 step에서 모델은 다음 symbol을 생성하기 위해 이전에 생성한 symbol을 추가적인 input으로 사용하는데 이를 auto-regressive라고 한다.

Transformer는 self-attention(각 단어끼리 얼마나 관계가 있는지를 계산하여 반영)과 point-wife fully connected layer르 쌓아 만든 encoder와 decoder로 구성되어 있다.

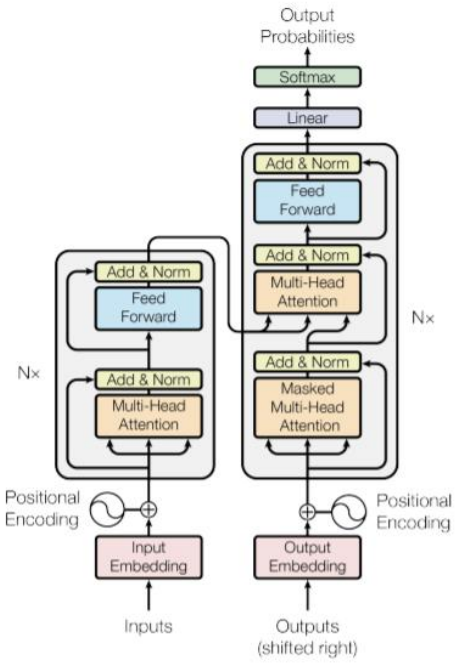


Figure 1: The Transformer - model architecture.

1. Encoder and Decoder Stacks

   - Encoder : encoder는 N=6개의 동일한 layer로 구성되어있다. 각 layer는 multi-head self-attention 메커니즘과 간단한 position-wise fully connected feed-forward로 구성된 두 개의 sub-layer를 갖는다. 각 sublayer에 $LayerNorm(x + Sublayer(x))$ residual connection을 사용한다. 이를 적용하기 위해 sublayer의 output dimension과 embedding dimension이 동일해야 하는데 여기서는 512로 설정하였다. 그 후 layer

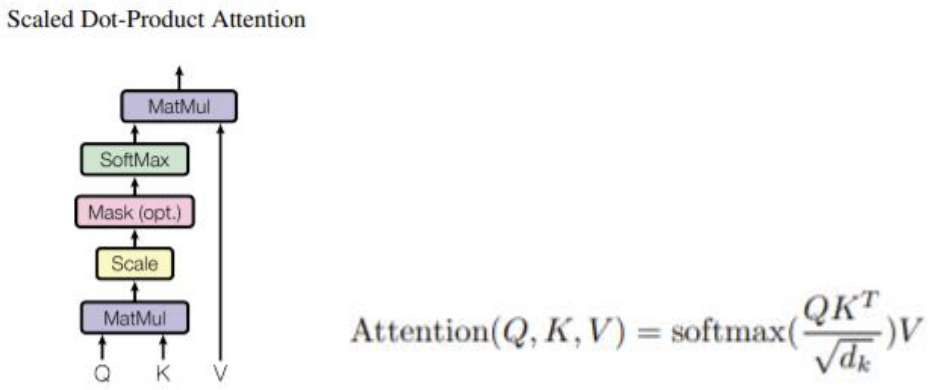
normalization을 적용한다.

- Decoder : N=6개의 동일한 layer로 구성되어 있다. Decoder는 encoder에 존재하는 두 개의 sub-layer와 추가로 encoder stack의 output에 대한 multi-head attention을 수행하는 세 번째 sub-layer를 가진다. Encoder와 유사하게 각 sublayer에 residual connection을 사용하고 layer normalization을 적용한다. Decoder의 self-attention sub-layer에서는 현재 위치보다 뒤에 있는 요소에 attention을 적용하지 못하도록 masking을 추가해준다. 이는 position i에 대한 예측을 위해 i 이전에 있는 정보들만 사용하도록 하는 것이다.

2. Attention

Attention function은 query와 key-value 쌍을 output에 mapping하는 것으로 설명할 수 있다. query, key, value, output은 모두 벡터로 이루어진다. Output은 weighted sum으로 계산되는데 weight는 해당 key와 query의 compatibility function으로 계산된다.

1) Scaled Dot-Product Attention



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

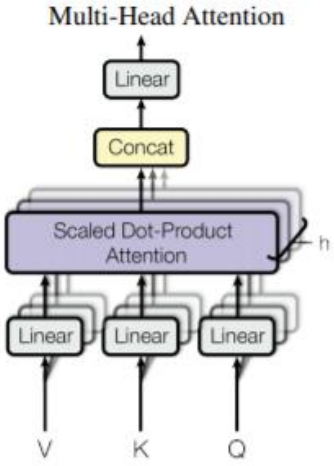
input으로 d_k 차원의 query, key, d_v 차원의 value를 가진다. query와 key의 dot product를 계산하고 $\sqrt{d_k}$로 나눈 값에 softmax 연산을 적용해 value에 대한 weight를 얻는다.

가장 자주 쓰이는 attention function은 additive attention과 dot-product(multiplicative) attention이다. dot-product attention은 scaling factor $\frac{1}{\sqrt{d_k}}$만 제외하면 논문의 알고리즘과 동일하다. Additive attention은 compatibility function을 단일 hidden layer의 feed-forward network를 사용해 계산한다. 두 연산의 이론적 complexity는 비슷하지만 matrix multiplication에 대한 최적화된 코드가 많기 때문에 dot-product attention이 더 빠르고 공간 효율성이 좋다.

작은 값의 d_k에 대해서는 두 메커니즘이 비슷하게 동작하지만 scaling이 없는 큰 d_k에 대해서는 additive function이 더 좋은 성능을 보인다. 큰 값의 d_k에 대해서는

dot product가 크게 증가하여 softmax 함수에서 gradient가 아주 작은 부분으로 가게 된다. 이를 방지하기 위해 $\frac{1}{\sqrt{d_k}}$로 dot product에 scale을 해준다.

2) Multi-Head Attention



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

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

d_{model} 차원의 key, value, query로 하나의 attention을 계산하는 대신 query, key, value에 대해서 서로 다르게 학습된 d_k, d_k, d_v 차원의 linearly project를 h번 수행하는 것이 더 효과적이다. project된 query, key, value에 대해 attention을 병렬적으로 계산해 d_v 차원의 output을 산출한다. 이 값들은 concatenate된 뒤에 다시 project 연산을 거쳐 최종 값을 얻게 된다. Multi-head attention을 사용해 다른 위치에 있는 representation subspace 정보를 attention할 수 있다. Single attention head는 이것을 억제한다.

이 논문에서는 h=8개의 parallel attention layer(head)를 사용했다. 각 벡터는 $d_k = d_v = \frac{d_{model}}{h} = 64$ 각 head에서 사용하는 dimension이 줄어들었기 때문에 총 연산량은 single-head attention과 비슷하다.

3) Applications of Attention in our Model

Transformer는 multi-head attention을 다음과 같은 방법으로 사용했다.

- Encoder-decoder attention layer에서 query는 이전 decoder layer에서 오고 key, value는 encoder의 output에서 온다. Decoder의 모든 position에서 input sequence(encoder output)의 모든 position에 attention을 적용할 수 있도록 한다.

- Encoder는 self-attention layer를 포함한다. Self-attention layer에서 모든 key, value, query는 encoder의 이전 layer output에서 온다. Encoder의 각 position에서 encoder의 이전 layer에 대한 모든 position에 대해 attention을 적용할 수 있다.

- Decoder의 self-attention layer에서는 decoder의 각 position에 대해 그 position 이전까지의 position에 대해서만 attention을 적용할 수 있도록 한다. Auto-regressive한 속성을 유지하기 위해 scaled dot-product attention에 masking out을 적용했다. Masking out은 i번째 position에 대한 attention을 계산할 때 i+1번째 이후의 모든 position은 softmax의 input을 -∞으로 설정하여 attention을 얻지 못하도록 하는 것이다.

3. Positoin-wise Feed-Forward Networks

Encoder, decoder의 각 attention sub-layer는 fully connected feed-forward network를 포함한다. 각 position마다 독립적으로 적용되므로 position-wise다. 두 개의 선형 변환과 ReLU activation function으로 구성되어 있다.

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

x에 선형 변환을 적용한 후 ReLU를 거쳐 다시 선형 변환을 한다. 선형 변환은 position마다 같은 파라미터를 사용하지만 layer가 달라지면 다른 파라미터를 사용한다. Input과 output의 차원은 d_{model}=512이고 inner-layer의 차원은 d_{ff}=2048이다.

4. Embeddings and Softmax

다른 sequence transduction model과 유사하게 input과 output token을 $d_{textmodel}$차원을 갖는 벡터로 변환하여 embedding을 학습한다. 일반적으로 학습된 선형 변환과 softmax 함수를 사용해 decoder의 output을 다음 token을 예측하기 위한 확률로 변환한다. 두 embedding layer와 softmax 이전의 linear transformation에서 동일한 weight 행렬을 공유한다. Embedding layer에서는 weight에 $\sqrt{d_{model}}$을 곱한다.

5. Positional Encoding

Transformer 모델은 recurrence나 convolution을 포함하지 않는다. 따라서 sequence의 순서성을 이용하기 위해서는 position에 대한 정보를 sequence의 token에 주입하여야 한다. 이를 위해 encoder와 decoder의 input embedding에 positional encoding을 더해준다. Positional encoding은 embedding과 더해질 수 있게 $d_{textmodel}$로 같은 차원을 갖는다. 논문에서는 다른 frequency를 갖는 sin, cos함수를 이용했다.

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

pos는 position, i는 차원이다. 이 함수를 통해 모델이 relative position에 대해 attention을 더 쉽게 학습할 수 있다. 고정된 offset k에 대해서 위 둘은 선형 변환으로 나타낼 수 있다.

IV. Why 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. Layer의 계산 복잡도가 감소한다.
2. 더 많은 양을 병렬처리 할 수 있다.
3. 긴 거리의 dependency를 잘 학습할 수 있다.

V. Training

1. Training Data and Batching

   4.5M개의 문장 쌍이 있는 WMT 2014 English-German dataset을 이용했다. 36M개의 문장 쌍을 포함하는 WMT 2014 English-French dataset도 이용했다. 각 batch마다 약 25000개의 source token과 target token을 포함하는 문장 쌍을 가지도록 하였다.

2. Hardward and Schedule

   8 NVIDIA P100 GPUs를 가진 하나의 기계에서 훈련하였다. 기본 모델의 경우 하이퍼 파라미터에 대해 각 훈련 step은 약 0.4초가 소요되었다. 총 100,000단계 또는 12시간 동안 기본 모델을 훈련하였다. 큰 모델의 경우 step 시간은 1.0초였다. 큰 모델은 300,000 step으로 훈련되었다.(3.5일)

3. Optimizer

   $\beta_1 = 0.9, \beta_2 = 0.98, \varepsilon = 10^{-9}$ 를 파라미터로 갖는 Adam optimizer를 사용하였다. 또한 $lrate = d_{model}^{-0.5} \cdot \min({step_{num}}^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$ 이 식과 같이 변화하는 learning rate를 사용하였다. 처음에는 warmup_steps 동안 step_num에 비례하여 선형적으로 증가하고 이후에는 step_num의 역제곱근에 비례하여 감소한다. warmup_steps=4000을 사용하였다.

4. Regularization

   1) Residual Dropout

      Sub-layer의 각 output이 input으로 사용되거나 normalized 되기 전에 dropout을 적

용했다. 추가로 각 encoder, decoder stack에 embedding과 positional encoding을 더해 dropout을 적용했다. 기본적으로 P_{drop}=0.1을 사용했다.

2) Label Smoothing

학습이 진행되는 동안 ϵ_{ls}=0.1을 갖는 label smoothing을 적용했다.

VI. Results

1. Machine Translation

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

2. Model Variants

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	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

A를 보면 attention heads의 수와 key와 value 차원을 다르게 하였는데 single-head attention이 최상의 결과보다 0.9BLEU 더 나쁘지만 헤드가 너무 많으면 좋지 않았다.

B를 보면 d_k의 크기를 줄이는 것은 결과를 나쁘게 했다. 이는 호환성을 결정하는 것이 쉽지 않으며 내적보다 더 정교한 호환성 기능이 도움이 될 수 있음을 시사했다.

C와 D를 보면 예상대로 더 큰 모델이 더 좋은 성능을 보였으며 dropout이 overfitting을 피하는데 매우 도움이 된다는 것을 보였다.

E에서 sinusoidal을 positional embedding으로 대체해도 기본 모델과 거의 동일한 결과를 보였다.

3. English Constituency Parsing

Transformer가 다른 작업으로 일반화할 수 있는지 평가하기 위해 영어 구성 구문 분석에 대한 실험을 하였다. 이 실험은 output이 강력한 구조적 9가지 제약 조건을 따르며 input보다 훨씬 길다. 게다가, RNN sequence-to-sequence 모델은 이 작은 데이터 영역에서 최상의 결과를 얻을 수 없었다.

Penn Treebank의 Wall Street Journal(WSJ) 부분에서 대략 40K 훈련 문장을 d_{model}=1024로 4 layer transformer를 훈련시켰다. 또한, 약 1700만 개의 문장으로 이루어진 더 높은 신뢰도와 Berkley Parser 말뭉치를 사용하여 semi-supervised 환경에서 훈련했다. WSJ 전용 설정에서는 16K 토큰 어휘를 사용하고 semi-supervised 설정에서는 32K 토큰 어휘를 사용했다.

Dropout과 attention, residual, learning rate, beam size을 선택하기 위해서 오직 적은 수의 실험을 수행하였다. 다른 모든 매개 변수는 영어에서 독일어로의 기본 번역 모델에서 변경되지 않았다. 추론하는 동안 최대 output 길이를 +300으로 늘렸다. WSJ와 semi-supervised 설정 모두 beam size 21과 learning rate 0.3을 사용하였다.

Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

작업 별 tuning이 없음에도 불구하고 transformer가 놀랍게 잘 수행되어 recurrent neural network grammar를([8]) 제외하고 이전에 보고된 모든 모델보다 더 나은 결과를 산출하였

다. RNN sequence-to-sequence 모델([37])과 달리 transformer는 40K 문장으로 구성된 WSJ 학습 세트에서만 학습하는 경우에도 Berkeley Parser([29])를 능가한다.

Ⅶ. Conclusion

이 논문에서는 recurrent layer를 multi-head self-attention으로 대체한 첫번째 sequence transduction model인 transformer를 제안하였다. translation task에서 transformer는 다른 recurrent나 convolution 모델보다 더 빠르게 학습하고 더 좋은 성능을 보였다.

https://arxiv.org/pdf/1706.03762.pdf