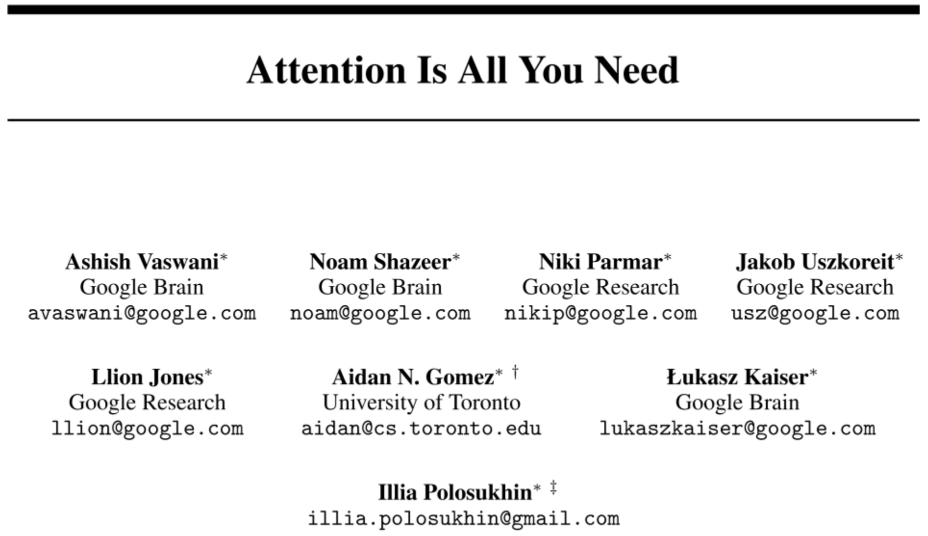
Attention Is All You Need

2019.07.06
김보섭

Agenda

Abstract

Encoder decoder architecture에서 대표적으로 활용된 building block인 RNN, CNN을 self-attention으로 모두 대체한 Transformer architecture를 제시



Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

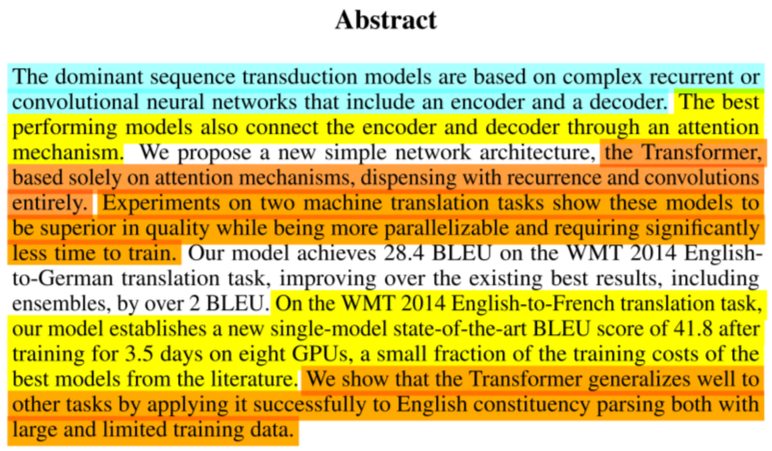
Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

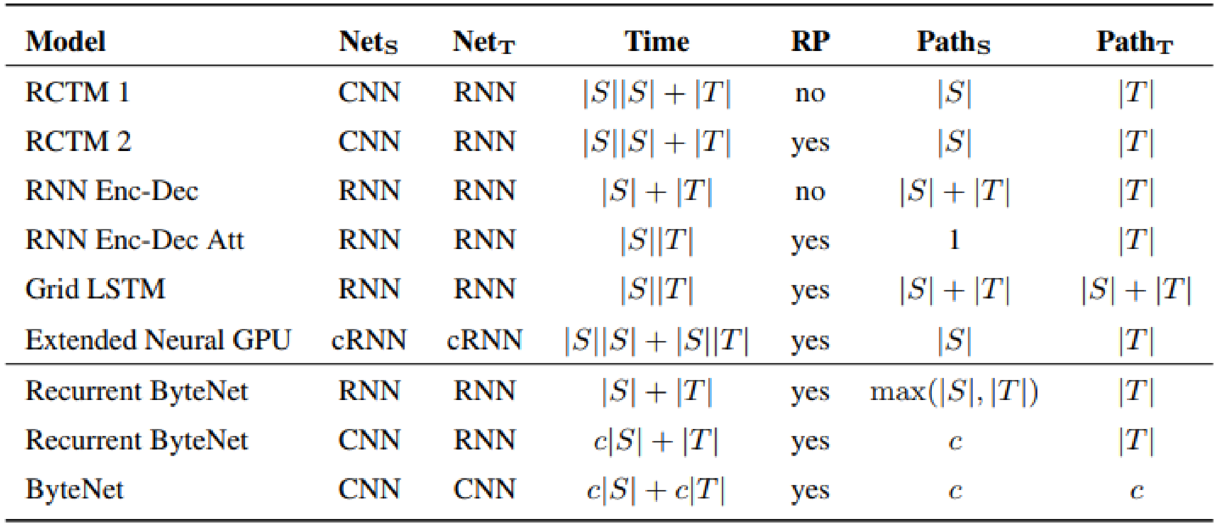


Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Introduction, Back ground, Why Self-attention? (1/3)

기존의 sequence transduction model의 대표적 architecture는 encoder decoder architecture에 building block으로 RNN, CNN을 쓰고, attention으로 resolution preserving을 갖춤



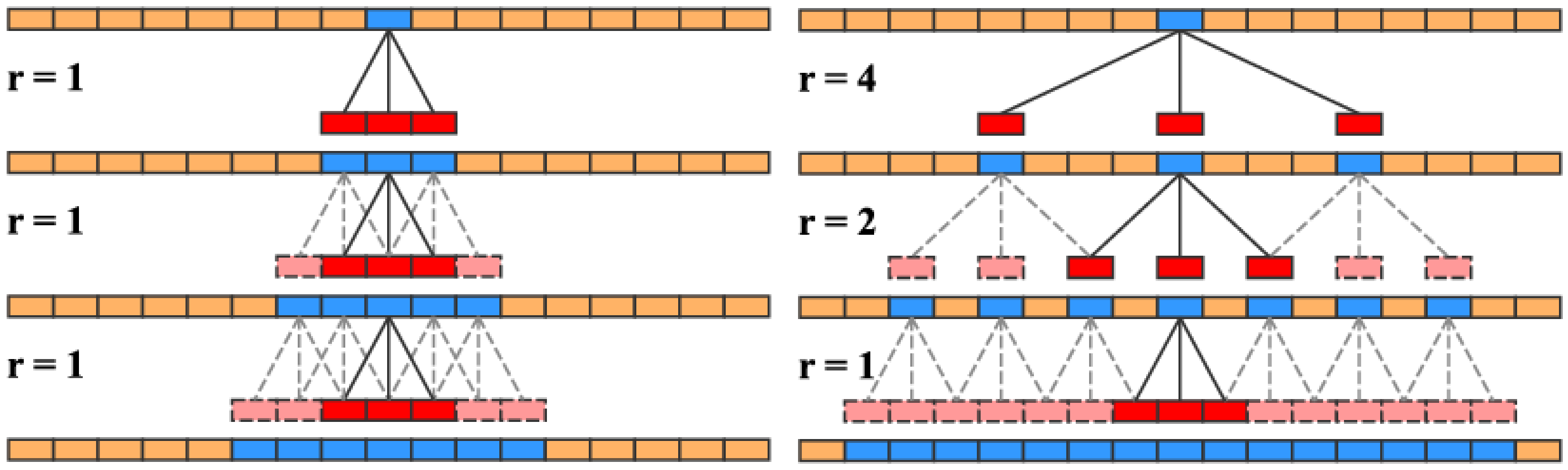
Model	Net_S	Net_T	Time	RP	Path_S	Path_T
RCTM 1	CNN	RNN	$\|S\|\|S\| + \|T\|$	no	$\|S\|$	$\|T\|$
RCTM 2	CNN	RNN	$\|S\|\|S\| + \|T\|$	yes	$\|S\|$	$\|T\|$
RNN Enc-Dec	RNN	RNN	$\|S\| + \|T\|$	no	$\|S\| + \|T\|$	$\|T\|$
RNN Enc-Dec Att	RNN	RNN	$\|S\|\|T\|$	yes	1	$\|T\|$
Grid LSTM	RNN	RNN	$\|S\|\|T\|$	yes	$\|S\| + \|T\|$	$\|S\| + \|T\|$
Extended Neural GPU	cRNN	cRNN	$\|S\|\|S\| + \|S\|\|T\|$	yes	$\|S\|$	$\|T\|$
Recurrent ByteNet	RNN	RNN	$\|S\| + \|T\|$	yes	$\max(\|S\|, \|T\|)$	$\|T\|$
Recurrent ByteNet	CNN	RNN	$c\|S\| + \|T\|$	yes	c	$\|T\|$
ByteNet	CNN	CNN	$c\|S\| + c\|T\|$	yes	c	c

Table 1. Properties of various neural translation models.

- Resolution preserving: the size of source representation should be linear in the length of the source string

Introduction, Back ground, Why Self-attention? (2/3)

Building block으로서 RNN은 sequential operation이므로 parallel computing이 힘들며 CNN은 parallel computing이 가능하지만, source가 길면 receptive field 문제로 stacking이 필요함 → parameter 증가



(a) Conventional convolutions (b) Dilated convolutions

Introduction, Back ground, Why Self-attention? (3/3)

Building block으로 self-attention을 활용하면, parallel computing은 물론 source가 길어도 long range dependency를 modeling하기위해 stacking이 필요하지 않음

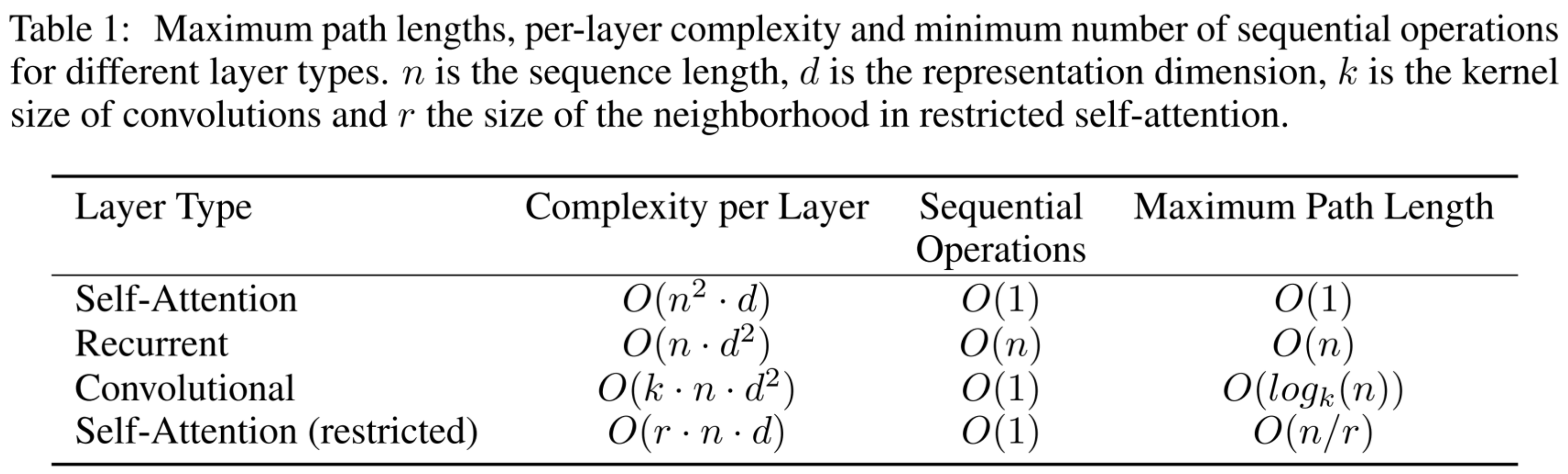
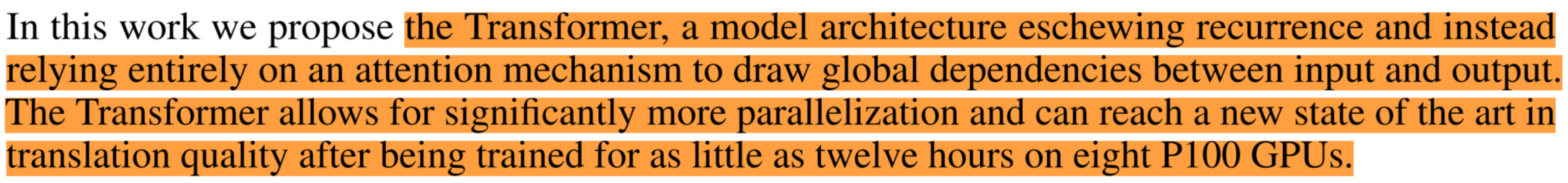


Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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



In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

Model architectures – Encoder and Decoder Stacks

Transformer는 크게 보면 기존의 encoder-decoder architecture에서 encoder, decoder를 self-attention을 stacking하여 구성한 형태

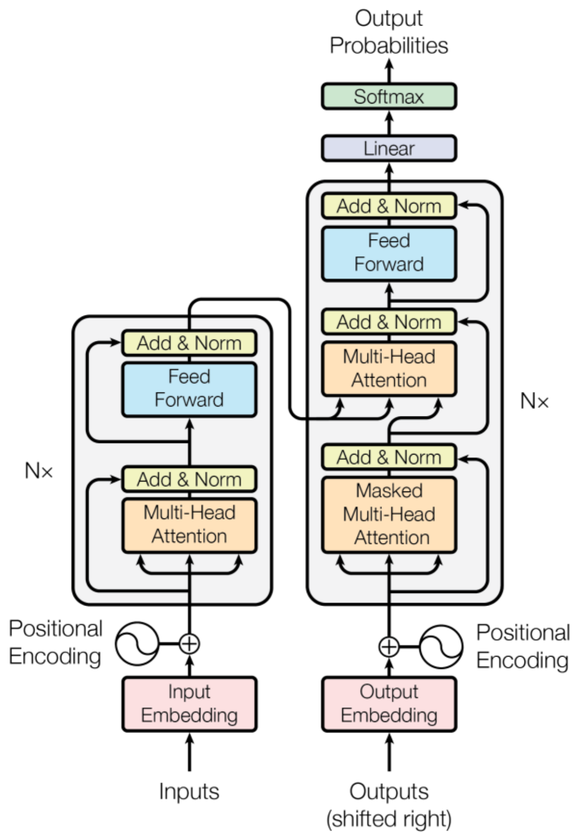
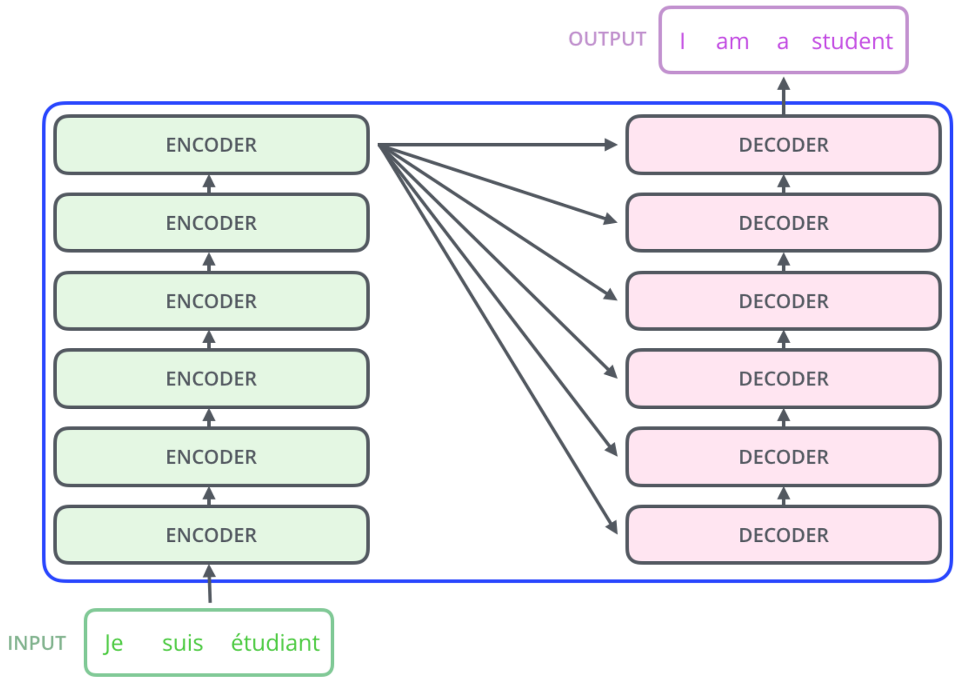


Figure 1: The Transformer - model architecture.



The illustrated Transformer (http://jalammar.github.io/illustrated-transformer/)

Model architectures – Input Representation

Self-attention은 operation에 sequential한 정보를 반영하기위해, token의 position과 embedding vector dimension으로부터 생성되는 정현파의 특정 값을 position 별, dimension 별로 더함 → Positional Encoding

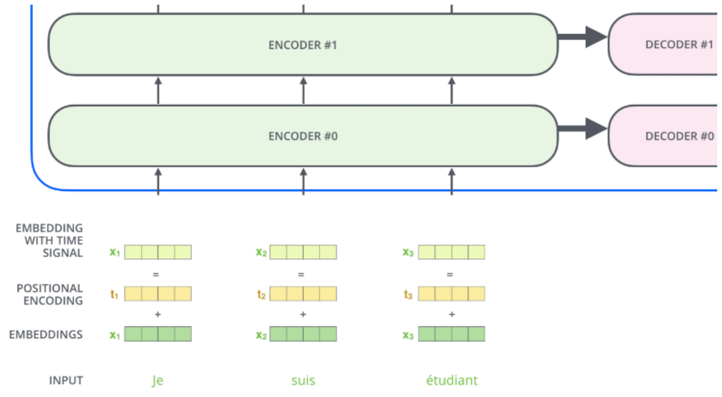
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

Representing The Order of The Sequence Using Positional Encoding

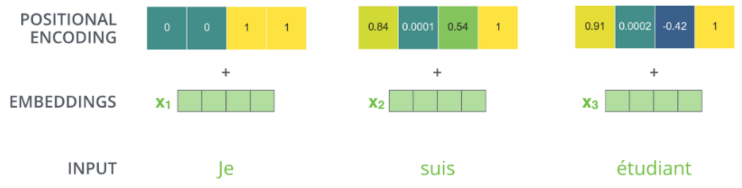
One thing that's missing from the model as we have described it so far is a way to account for the order of the words in the input sequence.

To address this, the transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.



To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

If we assumed the embedding has a dimensionality of 4, the actual positional encodings would look like this:

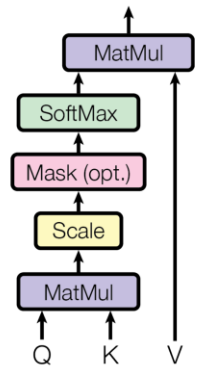


A real example of positional encoding with a toy embedding size of 4

Model architectures – Scaled Dot-Product Attention

Scaled Dot-Product Attention은 sequence의 token에 해당하는 vector가 query matrix, key matrix, key matrix와의 연산 통해서 query vector, key vector, value vector가 생성되면, 아래와 같이 계산함

Scaled Dot-Product Attention

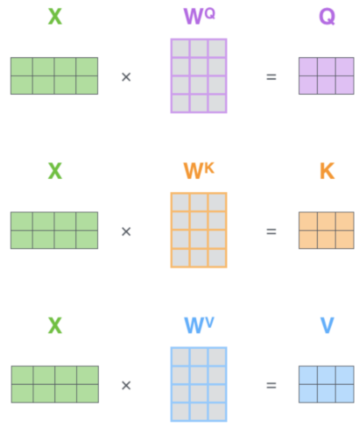


$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

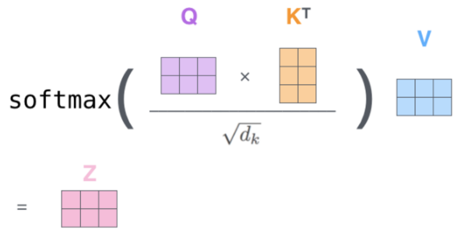
[4]To illustrate why the dot products get large, assume that the components of q and k are independent random variables with mean 0 and variance 1. Then their dot product, $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$, has mean 0 and variance d_k.

Matrix Calculation of Self-Attention

The first step is to calculate the Query, Key, and Value matrices. We do that by packing our embeddings into a matrix X, and multiplying it by the weight matrices we've trained (WQ, WK, WV).



Finally, since we're dealing with matrices, we can condense steps two through six in one formula to calculate the outputs of the self-attention layer.

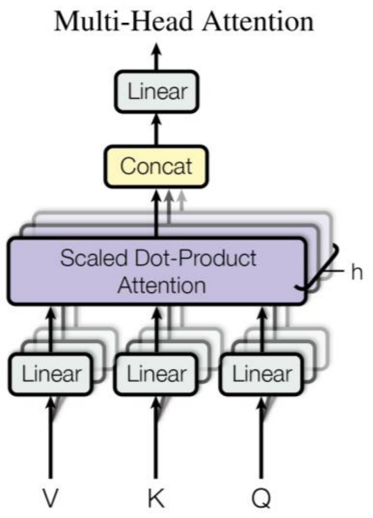


The self-attention calculation in matrix form

The illustrated Transformer (http://jalammar.github.io/illustrated-transformer/)

Model architectures – Multi-Head Attention

Scaled Dot-product attention이 하나의 head가 되는 형태 (head는 layer가 stacking되도 같음) → 8개의 head를 concatenate하여 사용

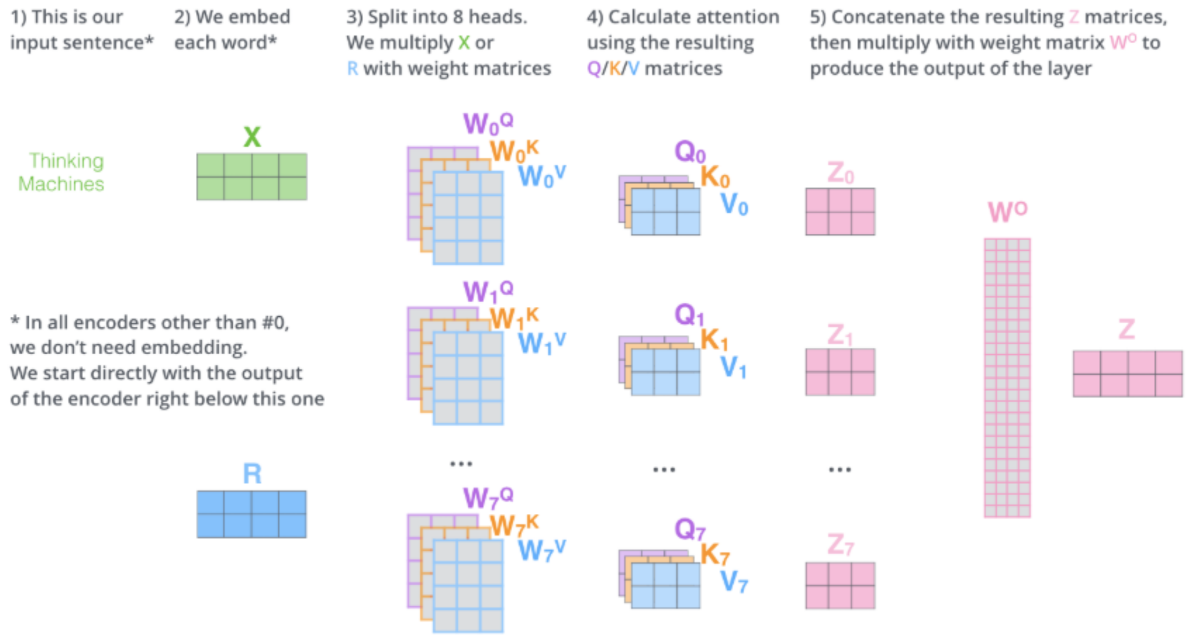


$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

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

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

In this work we employ $h = 8$ parallel attention layers, or heads. For each of these we use $d_k = d_v = d_{\text{model}}/h = 64$. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality.


1) This is our input sentence*

2) We embed each word*

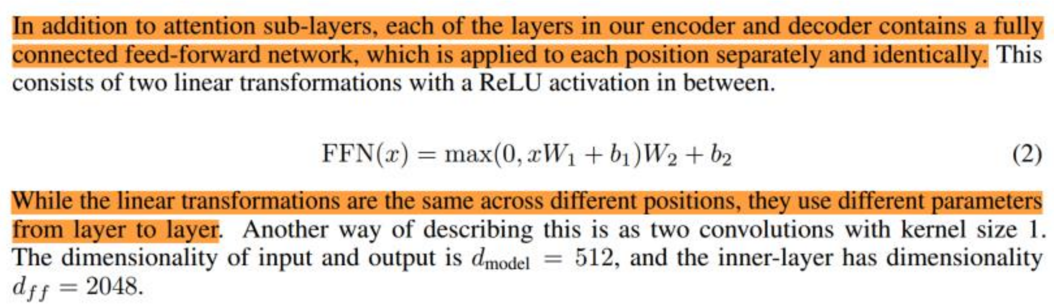
3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

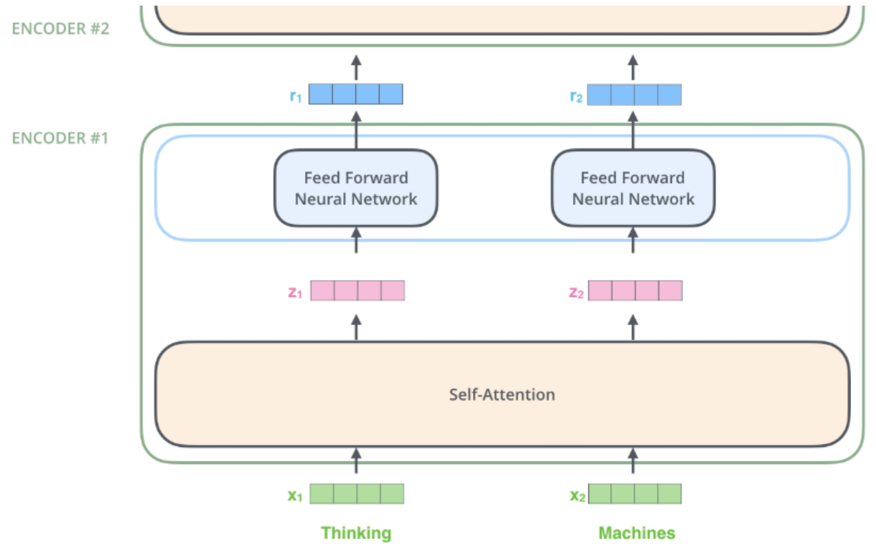
Model architectures – Position-wise Feed-Forward Networks



In addition to attention sub-layers, 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. This consists of two linear transformations with a ReLU activation in between.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \qquad (2)$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

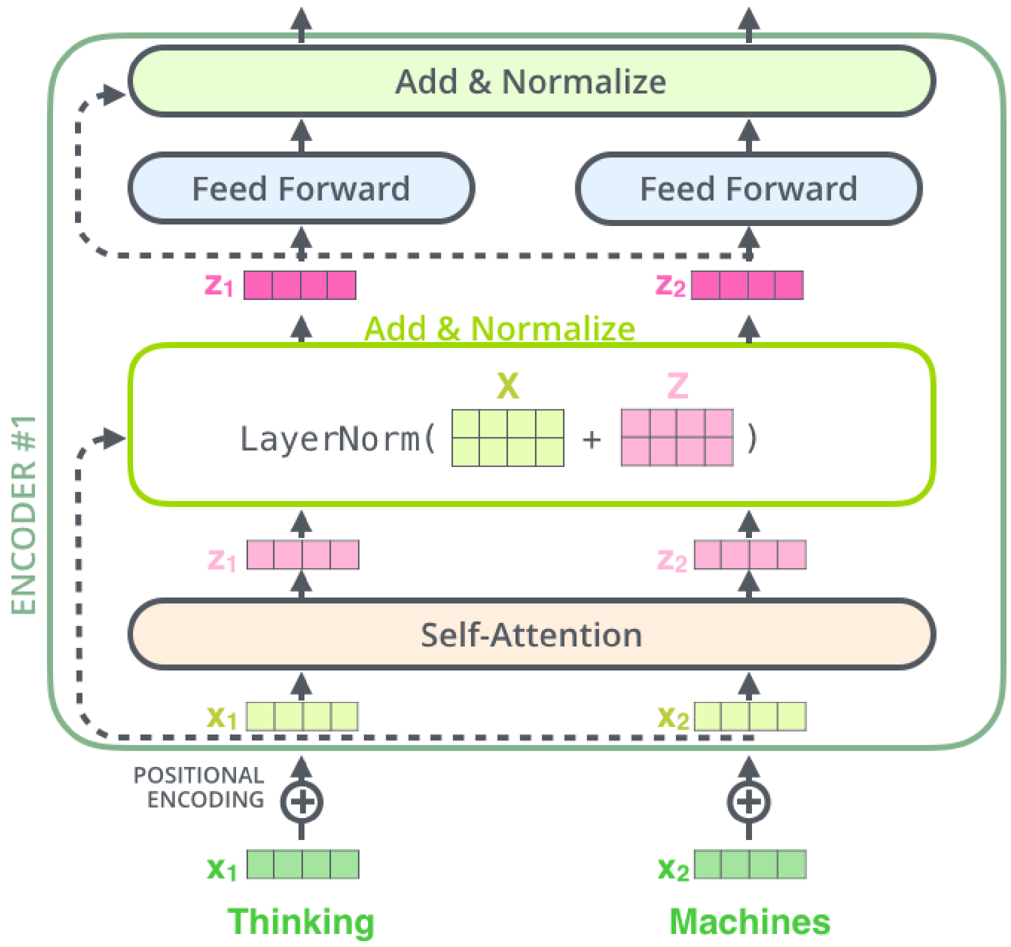


The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

Model architectures – Building block

Building block은 LayerNorm(x + Sublayer(x)) 형태로 구성됨

Sublayer: Multi-Head Attention, Position-wise Feed-Forward Neural Network



Model architectures – Applications of Attention in our Model

Building block을 바탕으로 encoder와 decoder를 구성한 Transformer에서 Attention은 아래와 같이 활용됨

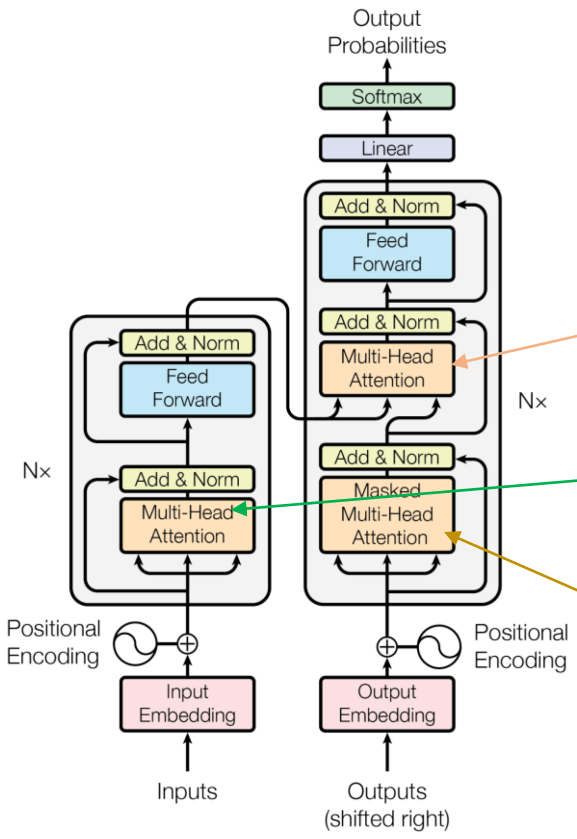


Figure 1: The Transformer - model architecture.

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. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models such as [38, 2, 9].
- 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.
- Similarly, 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. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

Model architectures – Embedding and Softmax

Output embedding가 pre-softmax linear transformation의 weight를 tying
→ 학습해야 할 parameter 감소

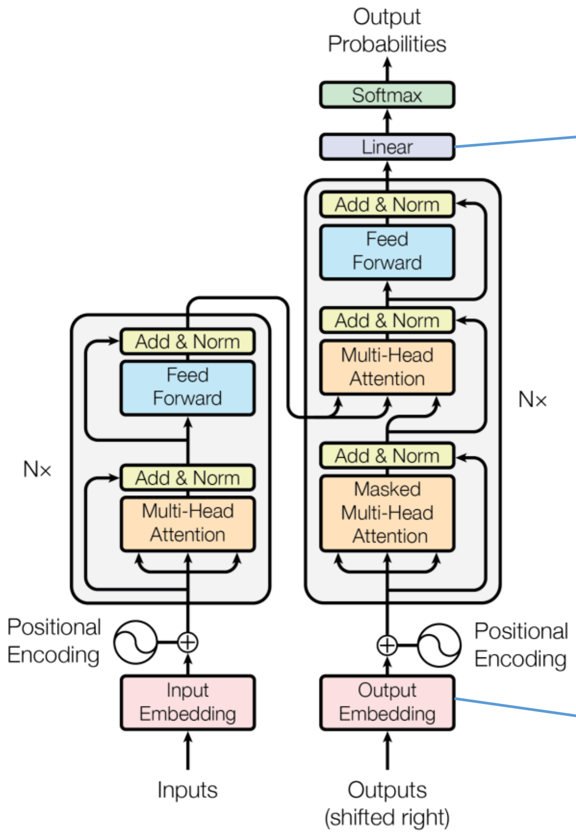


Figure 1: The Transformer - model architecture.

$W: (d_{model}, V)$

$d_{modle}: 512,$
$V: the\ number\ of\ token$

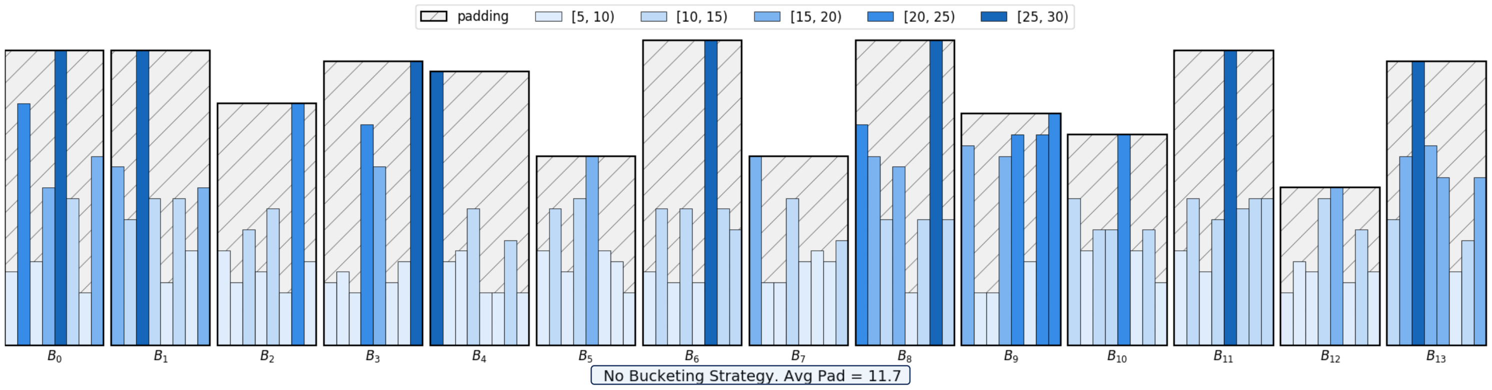
$E: (V, d_{model})$

$d_{modle}: 512,$
$V: the\ number\ of\ token$

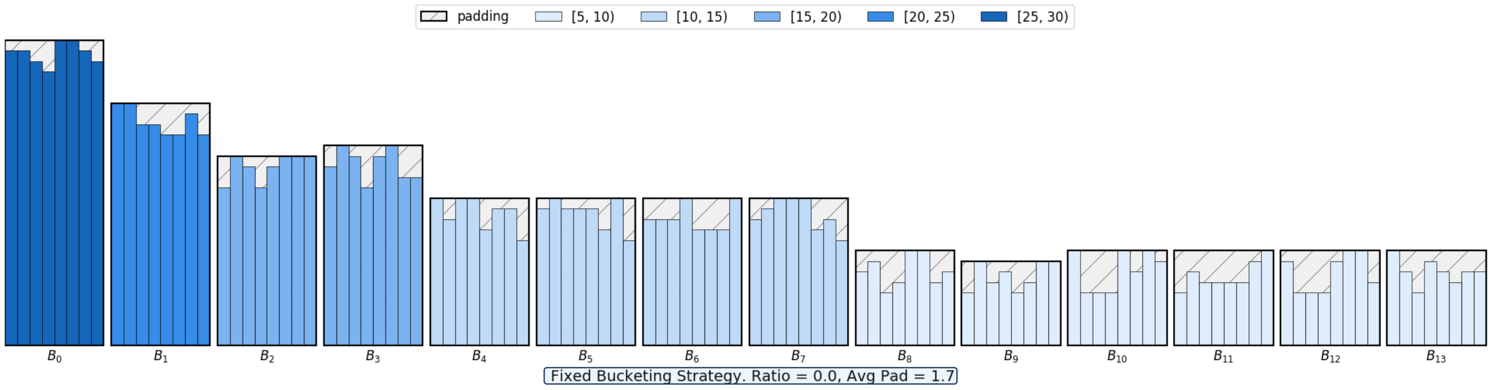
$$E^T = W$$

Training (1/3)

Source와 target을 합쳐서 Byte Pair Encoding으로 하나의 Vocabulary를 공유,
mini-batch 구성 시 bucketing을 활용



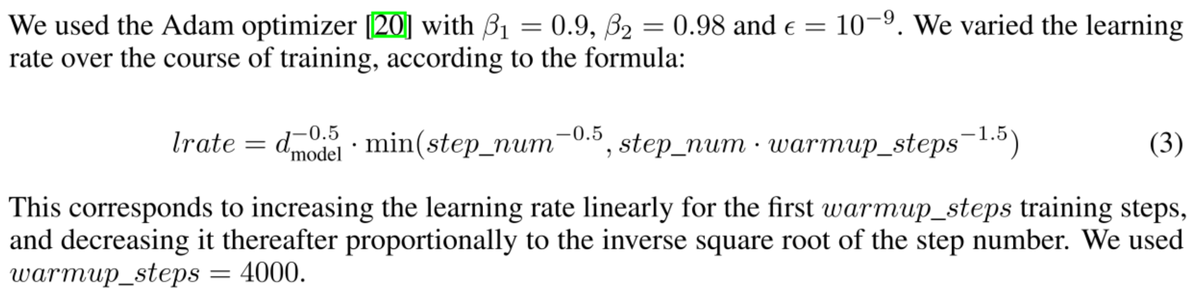
Bucketing으로 ‘<pad>’ token이 줄어서 효율적인 training이 가능



https://gluon-nlp.mxnet.io/api/notes/data_api.html

Training (2/3)

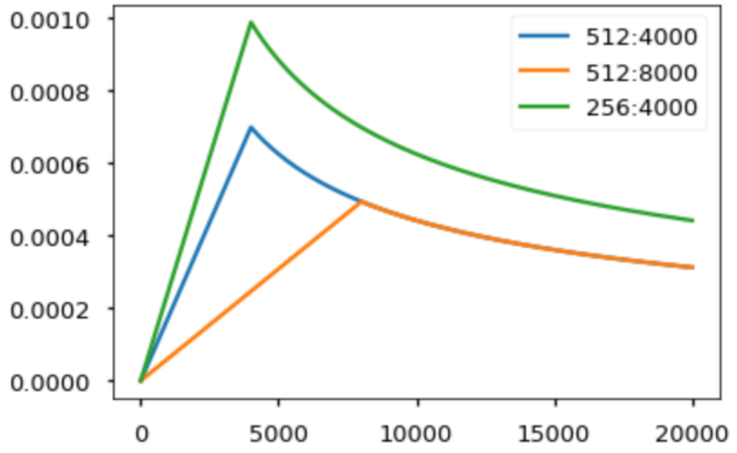
Training 초기에는 빠르게 적당한 parameter space에 들어가고, 해당 영역에서 세밀하게 optima를 찾기를 원함 → warm up learning rate를 사용



We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

This corresponds to increasing the learning rate linearly for the first *warmup_steps* training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.



Training (3/3)

Regularization으로 label smoothing을 활용하고, dropout은 sublayer의 output, input representation (embedding + positional encoding)에 활용

Model이 prediction한 label distribution

$$H(q',p) = -\sum_{k=1}^{K} \log \boxed{p(k)}\boxed{q'(k)} = (1-\epsilon)H(q,p)+\epsilon H(u,p)$$

$$q'(k) = (1-\epsilon)\boxed{\delta_{k,y}} + \frac{\epsilon}{K}$$

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

$$u(k) = \frac{1}{K}$$

실제 label distribution (ground truth)

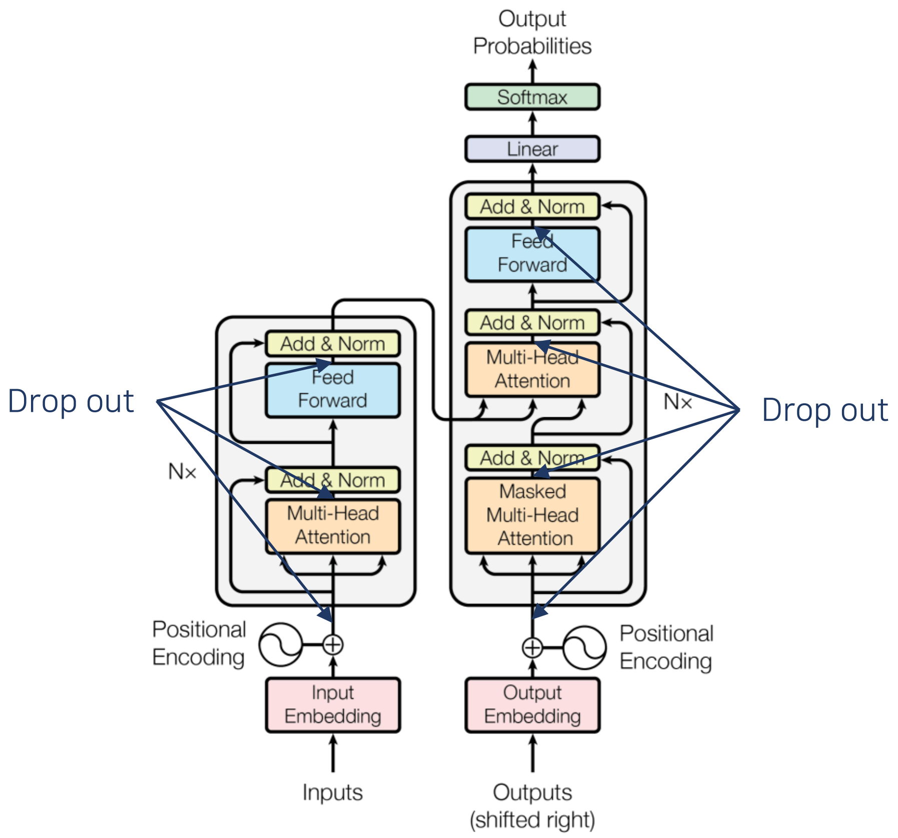


Figure 1: The Transformer - model architecture.

Results (1/3)

Training cost가 타 model 대비 훨씬 적은 single model로도 이전의 모든 architecture의 성능을 압도 → architecture 자체가 좋음

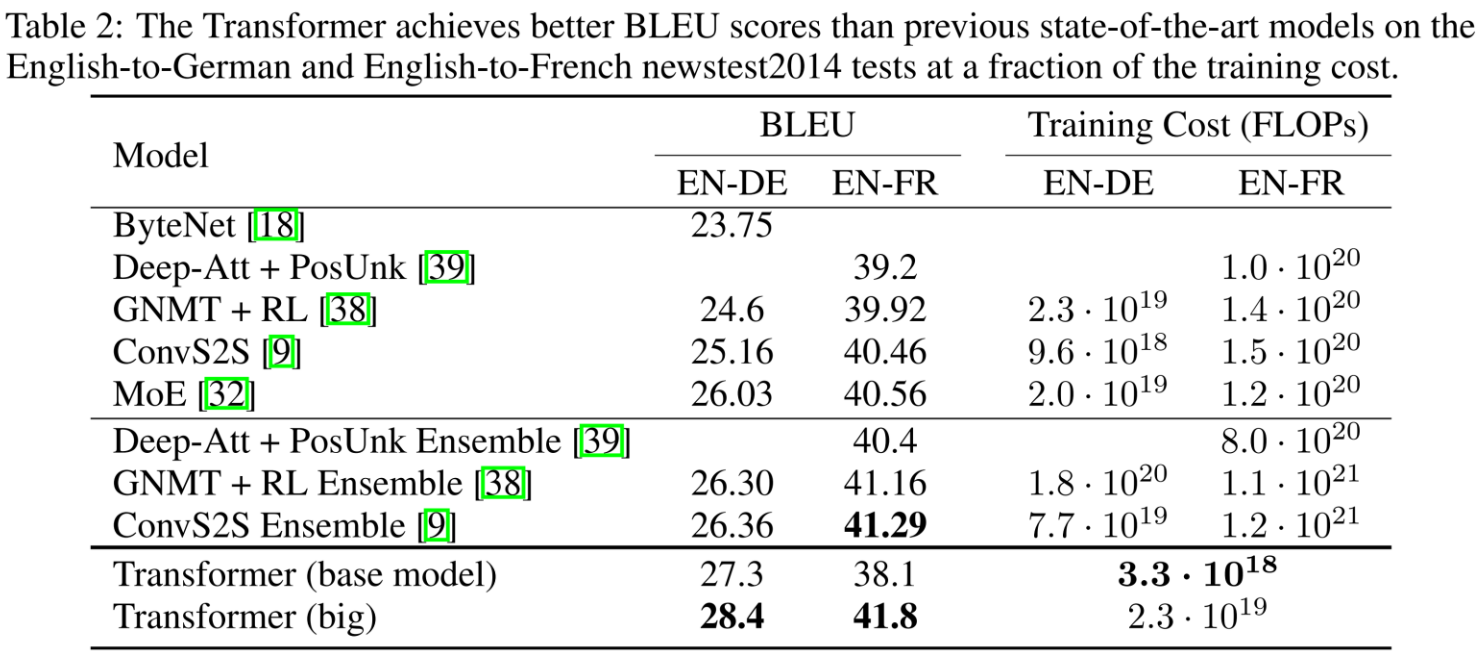


Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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**	$2.3 \cdot 10^{19}$	

Results (2/3)

Head가 너무 많으면 오히려 성능 저하, query와 key의 dimension이 적으면 성능이 감소, positional encoding 대신 positional embedding을 해도 성능은 비슷

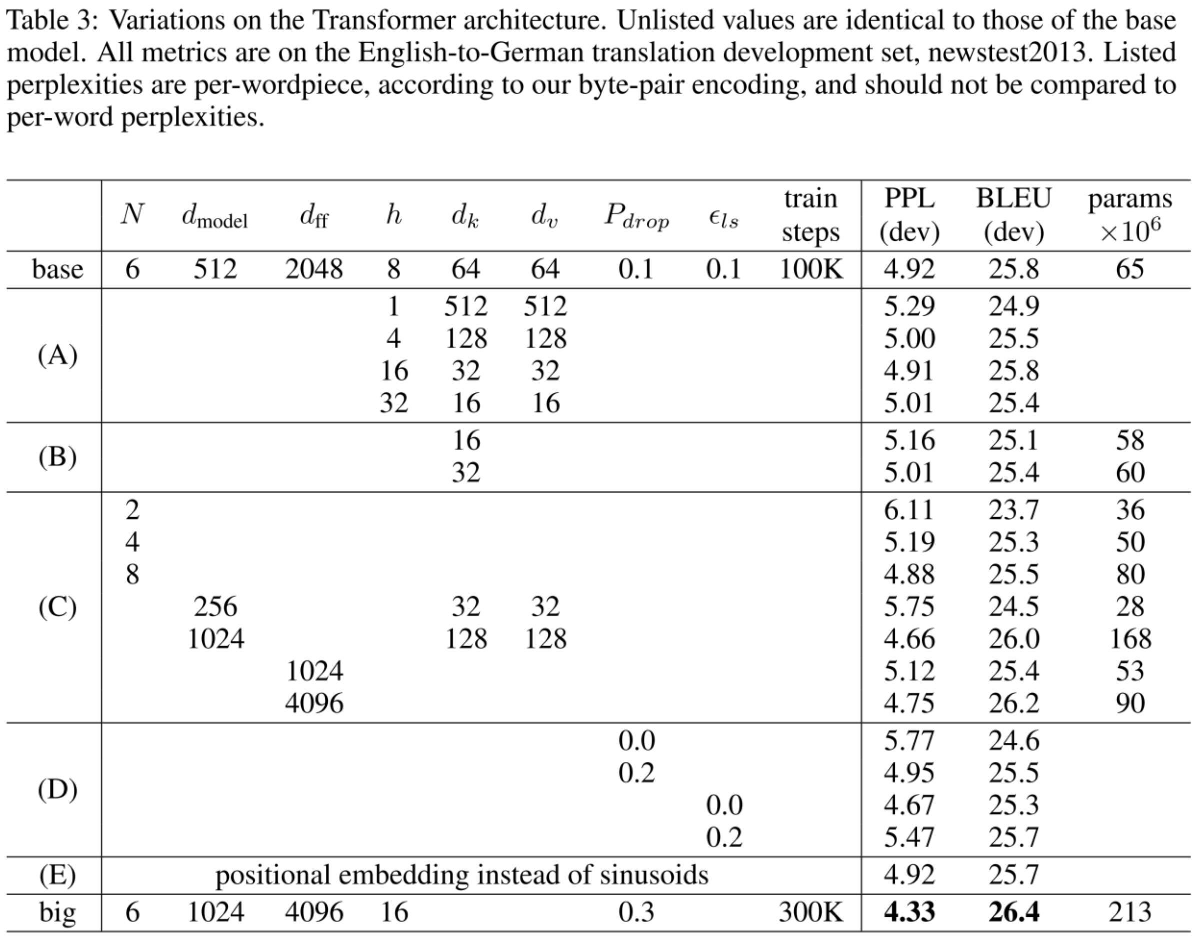


Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to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

Results (3/3)

Task-specific tuning이 상대적으로 다른 benchmark model에 비해 적었지만 성능이 비슷하고 더 좋기도 함

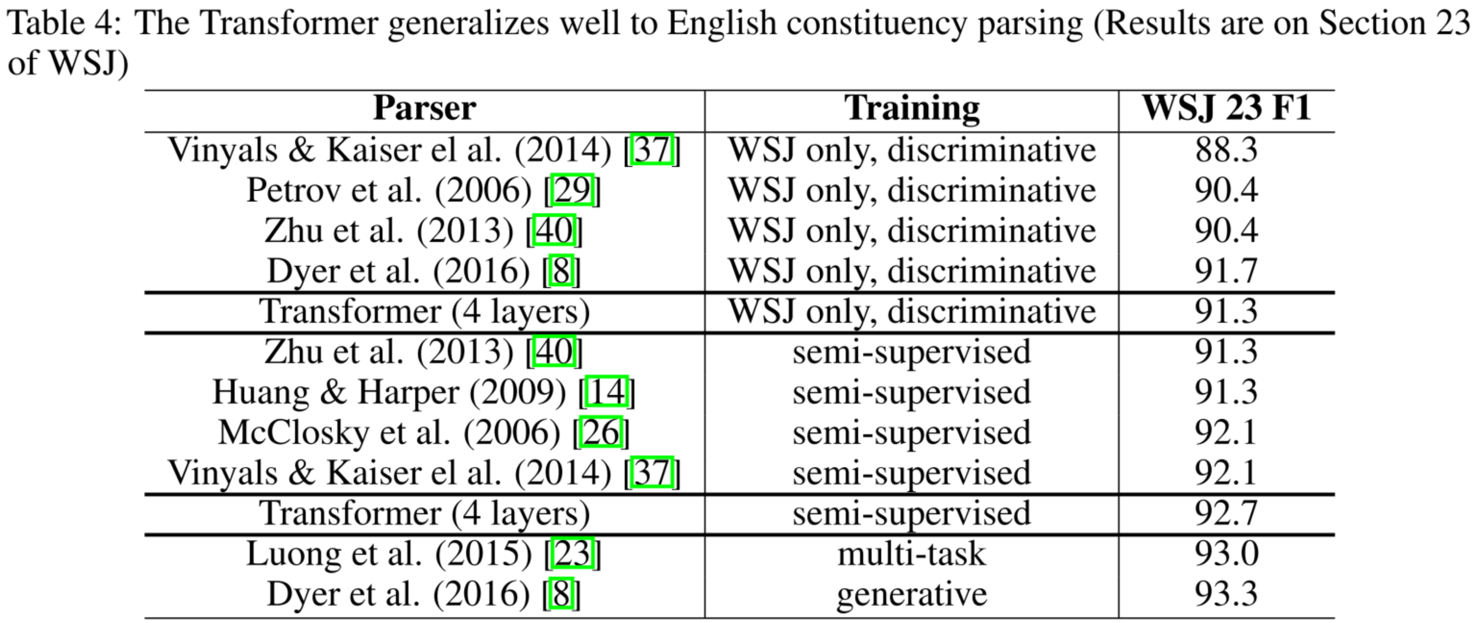
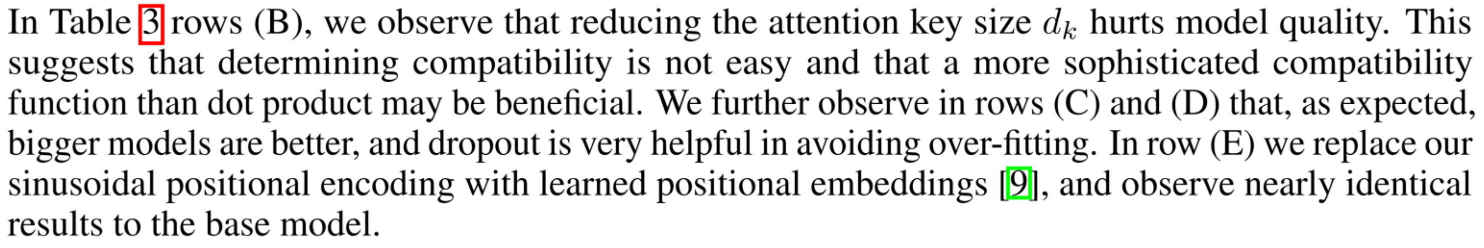


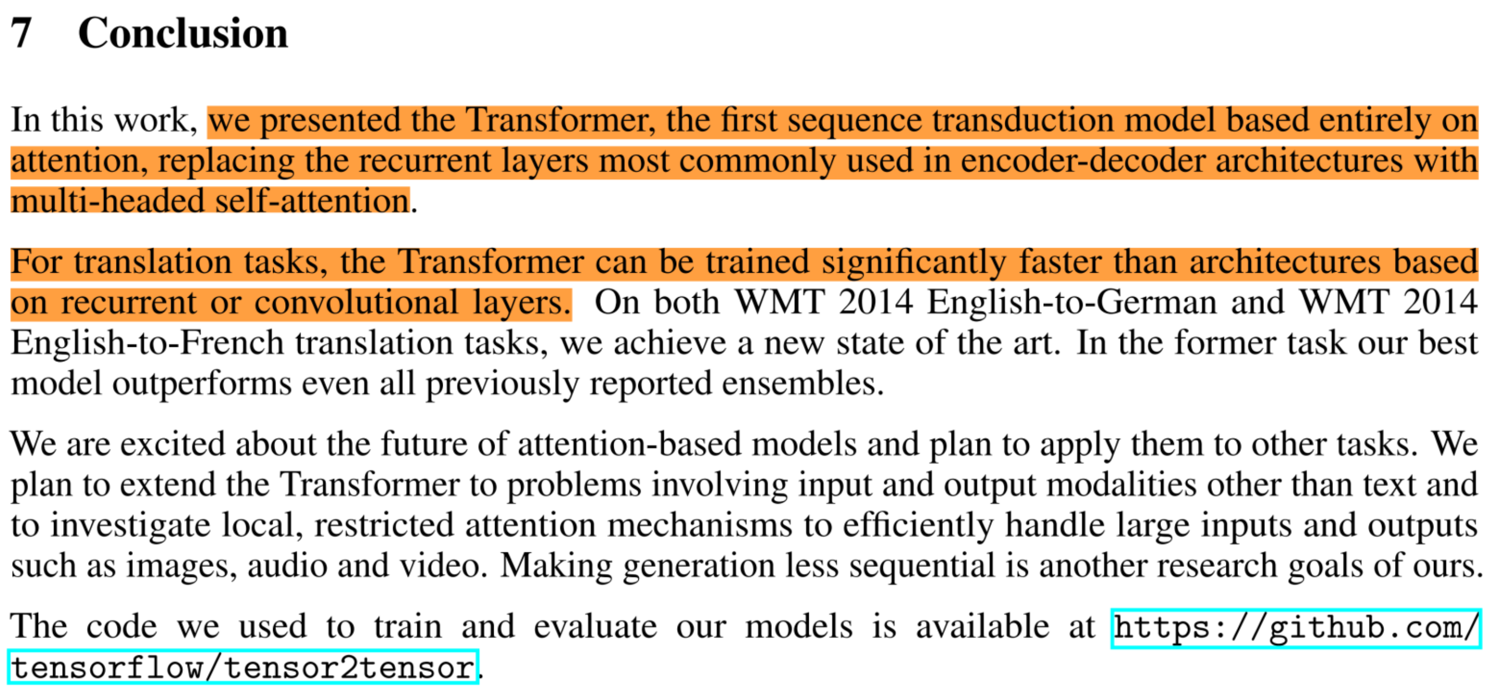
Table 4: The Transformer generalizes well to English constituency parsing (Results are on Section 23 of WSJ)

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



In Table 3 rows (B), we observe that reducing the attention key size d_k hurts model quality. This suggests that determining compatibility is not easy and that a more sophisticated compatibility function than dot product may be beneficial. We further observe in rows (C) and (D) that, as expected, bigger models are better, and dropout is very helpful in avoiding over-fitting. In row (E) we replace our sinusoidal positional encoding with learned positional embeddings [9], and observe nearly identical results to the base model.

Conclusion



7 Conclusion

In this work, we presented the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.

For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art. In the former task our best model outperforms even all previously reported ensembles.

We are excited about the future of attention-based models and plan to apply them to other tasks. We plan to extend the Transformer to problems involving input and output modalities other than text and to investigate local, restricted attention mechanisms to efficiently handle large inputs and outputs such as images, audio and video. Making generation less sequential is another research goals of ours.

The code we used to train and evaluate our models is available at `https://github.com/tensorflow/tensor2tensor`.

Q & A