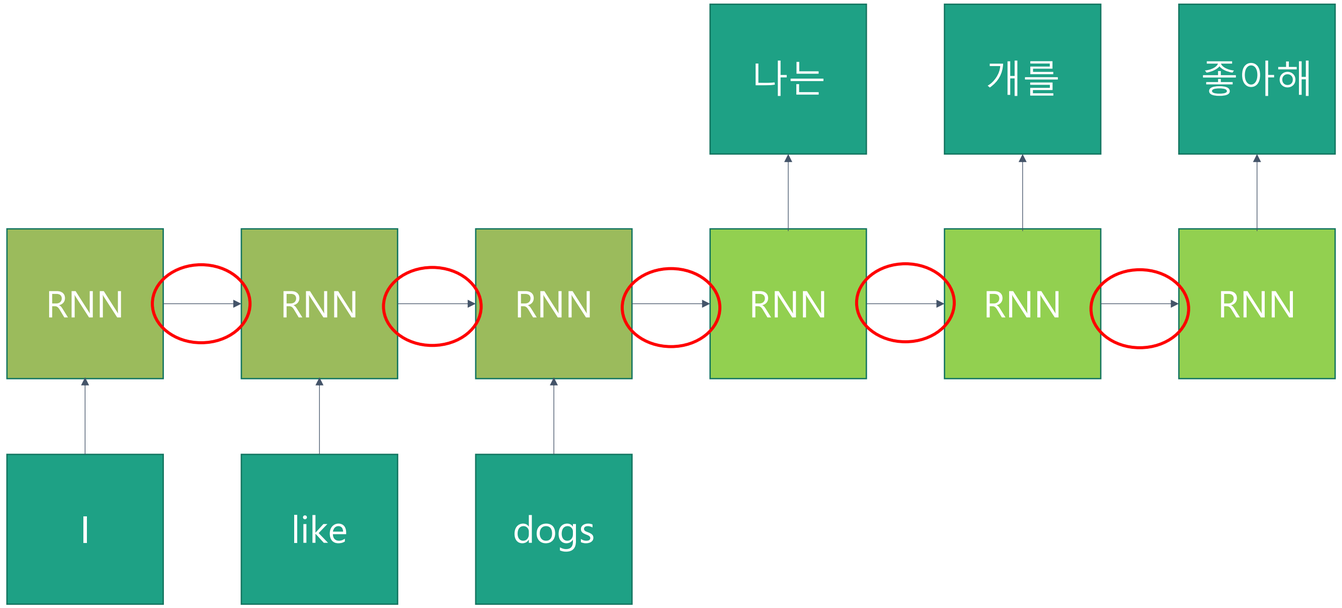
Attention is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Pro- cessing Systems, pages 6000–6010.

소개

- 기존에는 LSTM[2], GRU[3] 등의 RNN 모델들을 순차열 모델링이나 변환 문제에 접근하기 위해 많이 사용
- 하지만 기존 RNN 모델은 구조 상 데이터를 순차적으로 계산 -> 학습 시 병렬 계산을 할 수 없어 오랜 시간이 걸린다는 단점이 존재

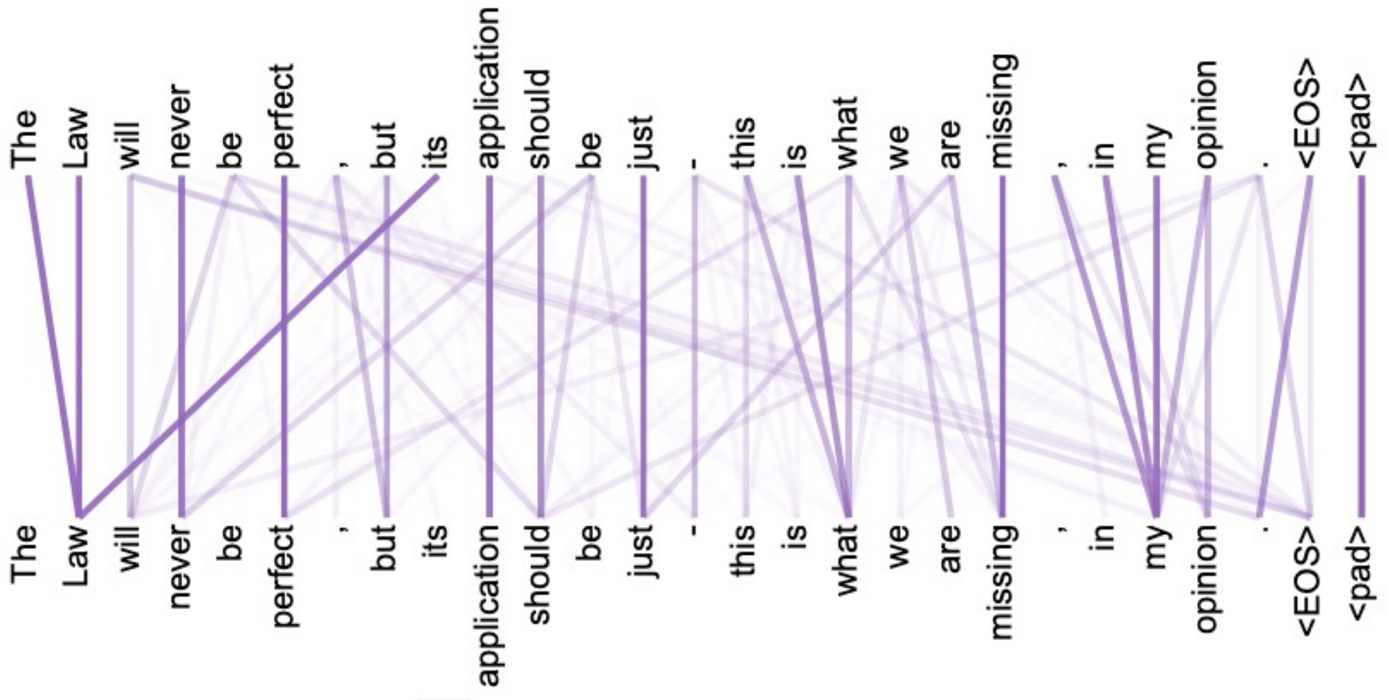
기존 RNN



2. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.
3. Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555, 2014.

소개

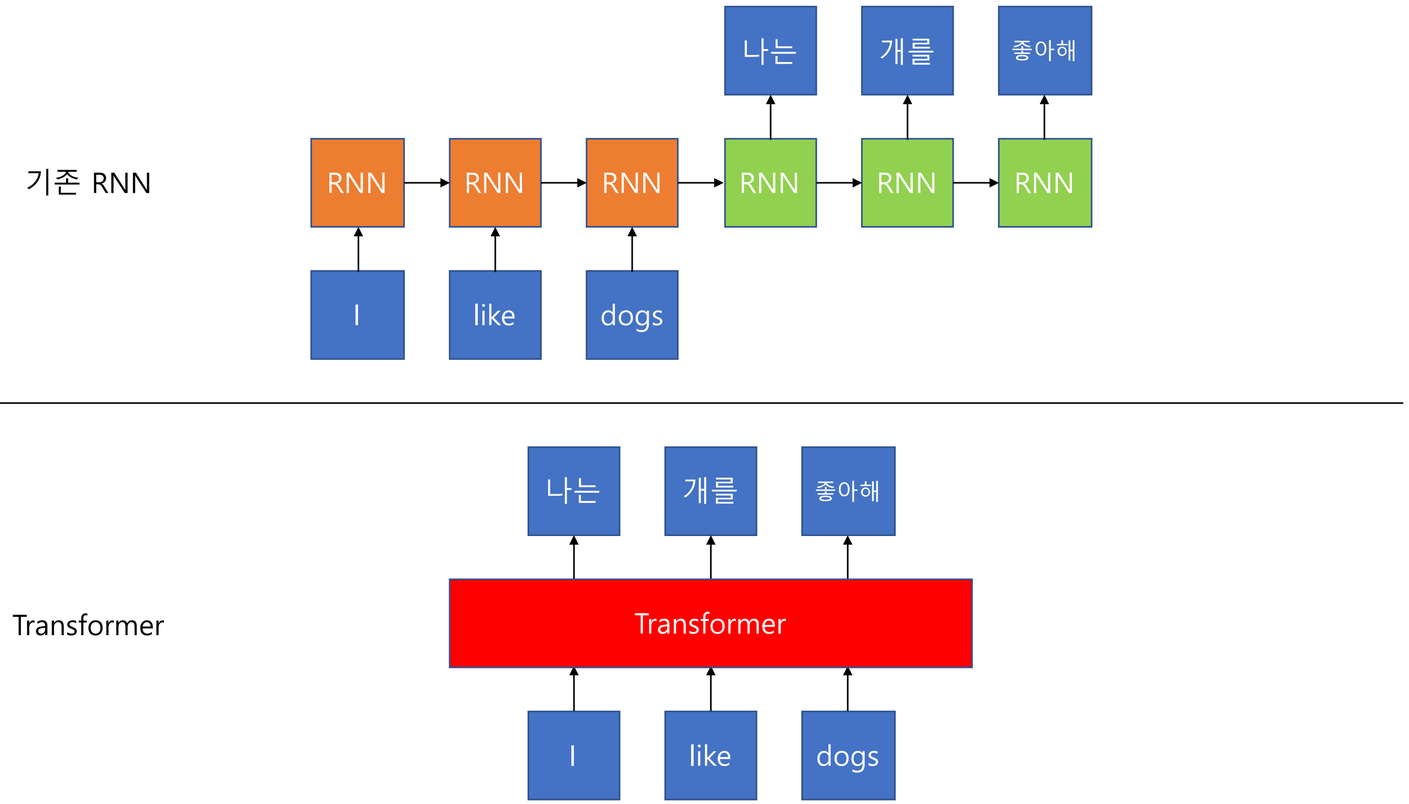
- 어텐션 메커니즘[4]은 입력 데이터와 출력 데이터 간 거리에 상관 없이 두 데이터 사이의 의존성을 모델링할 수 있게 하는 능력이 있어 최근 필수적으로 사용됨
- 기존 어텐션 메커니즘은 주로 RNN과 함께 사용됨



4. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

소개

- 병렬 학습을 진행하면서도 데이터 간 의존성을 반영하기 위해 RNN을 제외하고 어텐션 메커니즘만을 가져와 이에 의존한 **Transformer** 모델을 제시



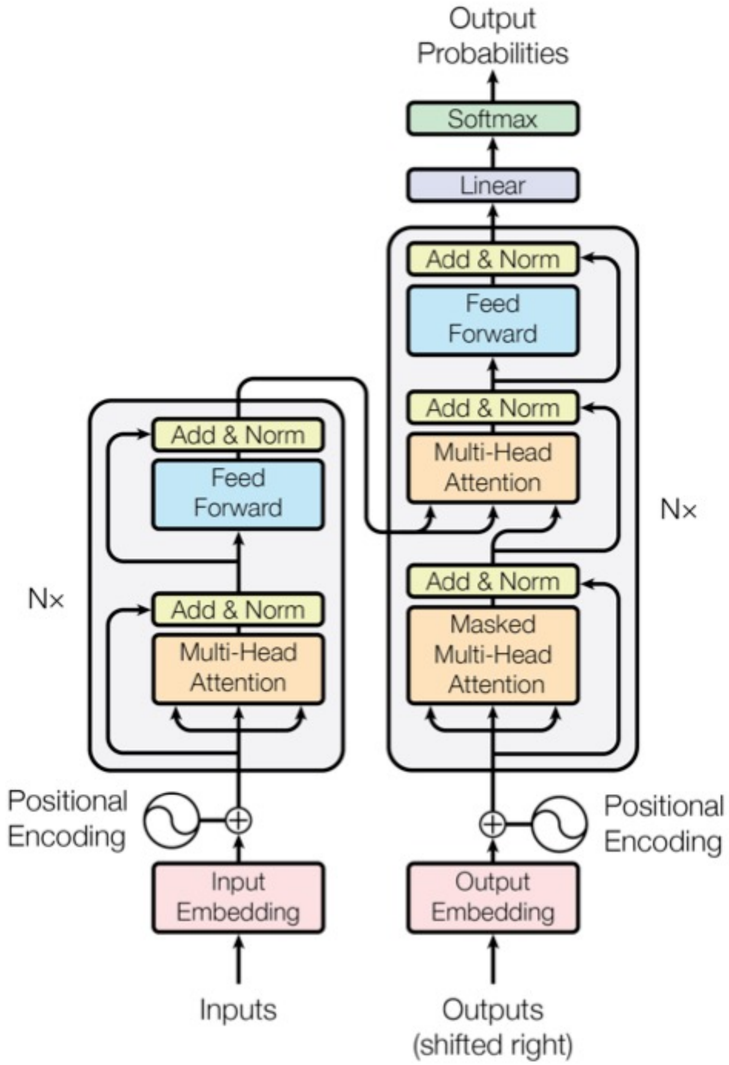
배경

- 기존에도 순차 계산 문제를 해결하기 위해 CNN 기반의 모델들이 제시됨
- 하지만 이들은 병렬 학습이 가능한 대신 입력 위치와 출력 위치 사이의 거리에 따라 계산량이 늘어나 먼 거리 사이의 의존성을 학습하기 어렵다는 문제가 존재
- Transformer는 병렬 학습이 가능하면서도 먼 거리 사이의 의존성을 학습할 수 있음

배경

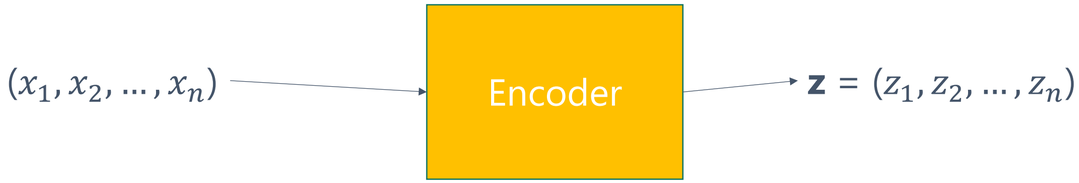
- 한 순차열 데이터의 representation을 계산하기 위해 해당 순차열의 각기 다른 위치를 고려하는 **셀프 어텐션**은 이미 성공적으로 많이 사용됨
- 하지만 RNN이나 convolution 없이 셀프 어텐션만 사용한 것은 Transformer가 최초

모델 전체 구조


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

모델 전체 구조

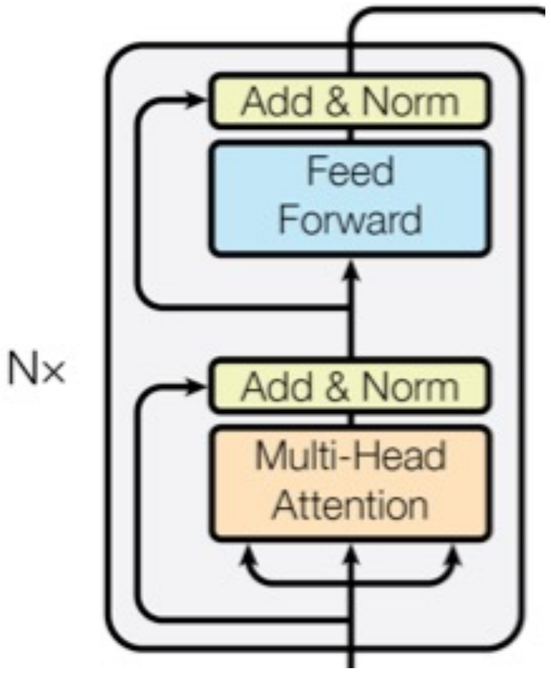
- 경쟁력 있는 모델들에서 사용되는 encoder-decoder 구조를 사용
- 입력으로 들어온 순차열을 Encoder로 1차 변환(encoding, 부호화)



- Encoder의 출력을 decoder로 2차 변환(decoding, 해독)

$x_i, z_i, y_i \in \mathbb{R}^d$

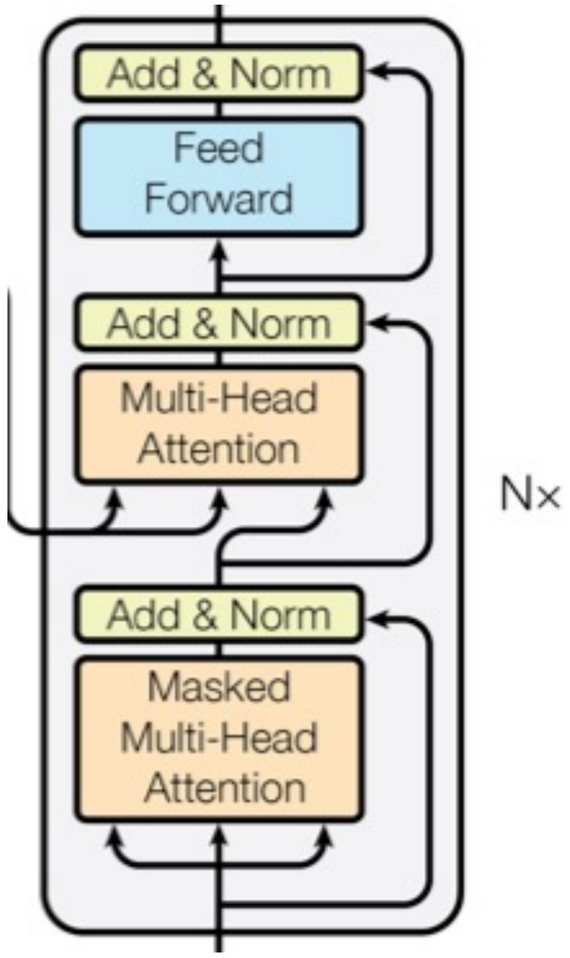
Encoder 구조



- 같은 구조의 layer 6 개(hyperparameter)로 구성
- 각 layer는 2 개의 sub-layer로 구성
  - Multi-head self-attention
  - Fully connected feed-forward network
- 한 sub-layer를 지날 때마다 residual connection[5], layer normalization[6] 시행
- Residual connection을 위해 embedding layer를 포함한 모든 sub-layer의 출력을 같은 크기로 조정($d_{\{model\}} = 512$)

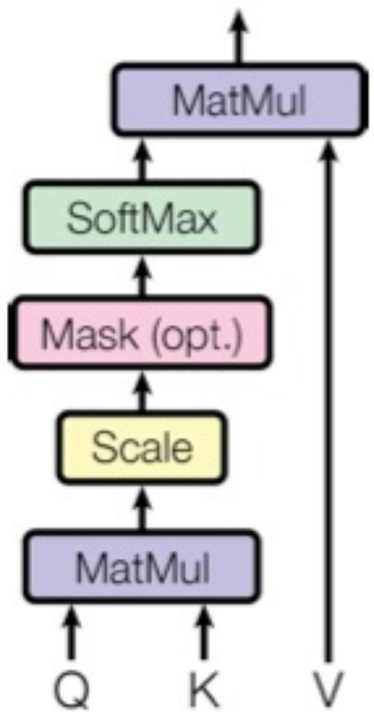
5. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for im- age recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.
6. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

Decoder 구조



- 같은 구조의 layer 6 개(hyperparameter)로 구성
- 각 layer는 3 개의 sub-layer로 구성
  - Masked multi-head self-attention
  - Multi-head attention over the output of the encoder stack
  - Fully connected feed-forward network
- 한 sub-layer를 지날 때마다 residual connection, layer normalization 시행
- Residual connection을 위해 embedding layer를 포함한 모든 sub-layer의 출력을 같은 크기로 조정($d_{model} = 512$)
- Self-attention 시행 시 현재 위치 뒤의 부분을 참고하지 않도록 masking

Scaled Dot-Product Attention



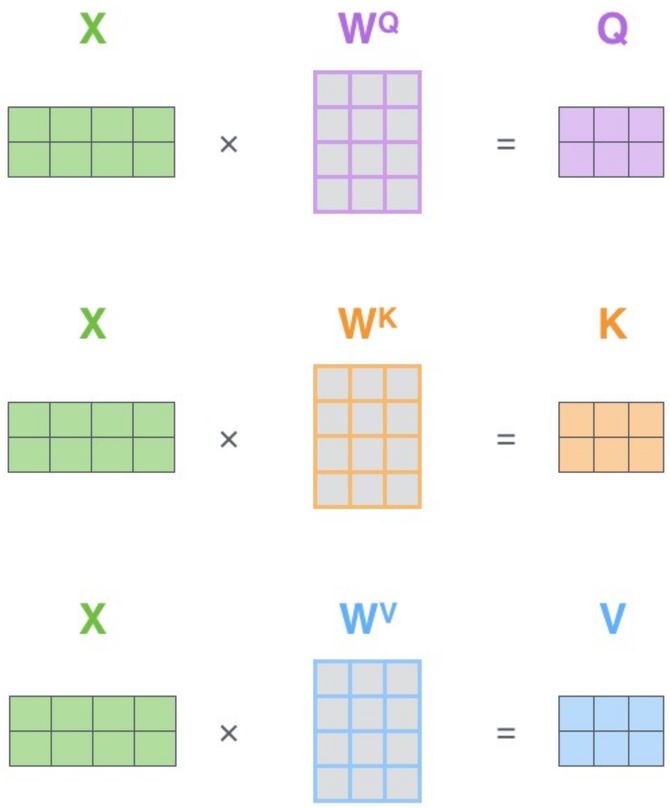
- 입력은 세 가지로 구성
  - Queries(dimension = d_k)
  - Keys(dimension = d_k)
  - Values(dimension = d_v)
- Query 하나가 주어졌을 때, 모든 key들과 내적을 구함
  - query와 key들 사이 유사도 계산
- 위의 내적을 $\sqrt{d_k}$로 나눔
  - 내적이 너무 커져 softmax로 입력될 때 gradient가 작은 지역으로 들어가지 않게 하기 위함
- Softmax 적용
  - 유사도를 바탕으로 weighted sum을 취하기 위해 합이 1이 되도록 변경
- 위의 결과를 weight로 value들의 weighted sum을 산출

Scaled Dot-Product Attention

- 앞의 과정을 모든 query에 대해 시행
- 실제로는 행렬 형태로 한 번에 계산

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

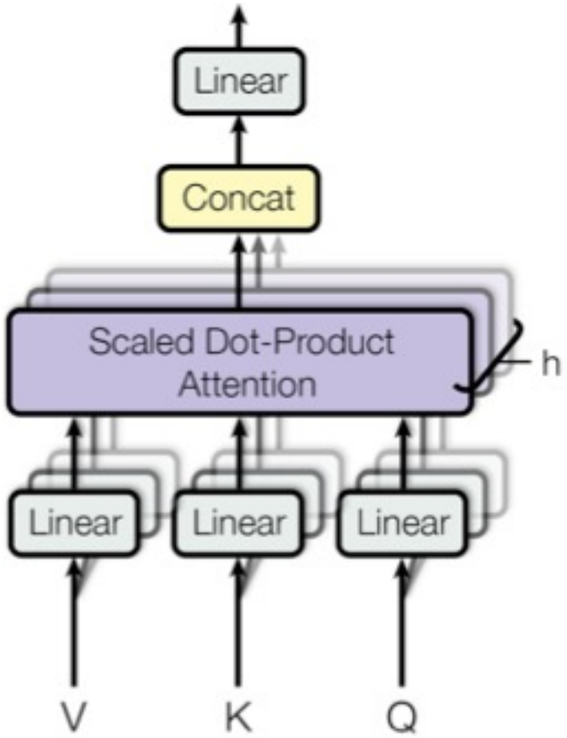
입력 -> (Query, Key, Value)



- 입력 데이터를 (Query, Key, Value) 형태로 변형하는 법[7]
- X = 입력 행렬
- W^Q, W^K, W^V = 학습할 weight 행렬들

7. Korean, NLP. "The Illustrated Transformer". Nlpinkorean.Github.Io, 2020, https://nlpinkorean.github.io/illustrated-transformer/.

Multi-Head Attention

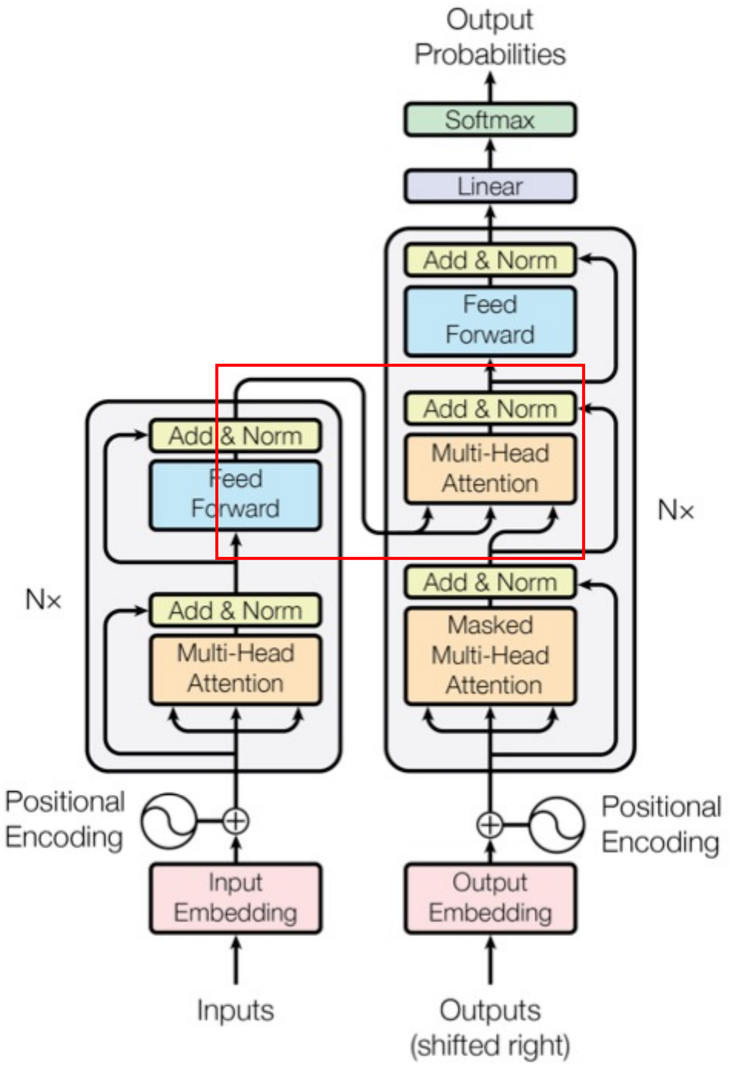


- Scaled Dot-Product Attention을 d_{model} 차원의 queries, keys, values에 한 번만 시행하는 것이 아니라 d_{model}/h 차원의 (queries, keys, values) set을 h 개 만들고 각각에 대해 Scaled Dot-Product Attention을 시행하는 방법을 제시
- h 개의 (queries, keys, values) set을 만들 때 각각 다르게 학습된 weight 사용
- 최종적으로 위 h 개의 결과를 concat한 뒤 한 번 더 linearly project
- 저자는 h = 8, $\frac{d_{model}}{h} = d_k = d_v = 64$ 사용

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

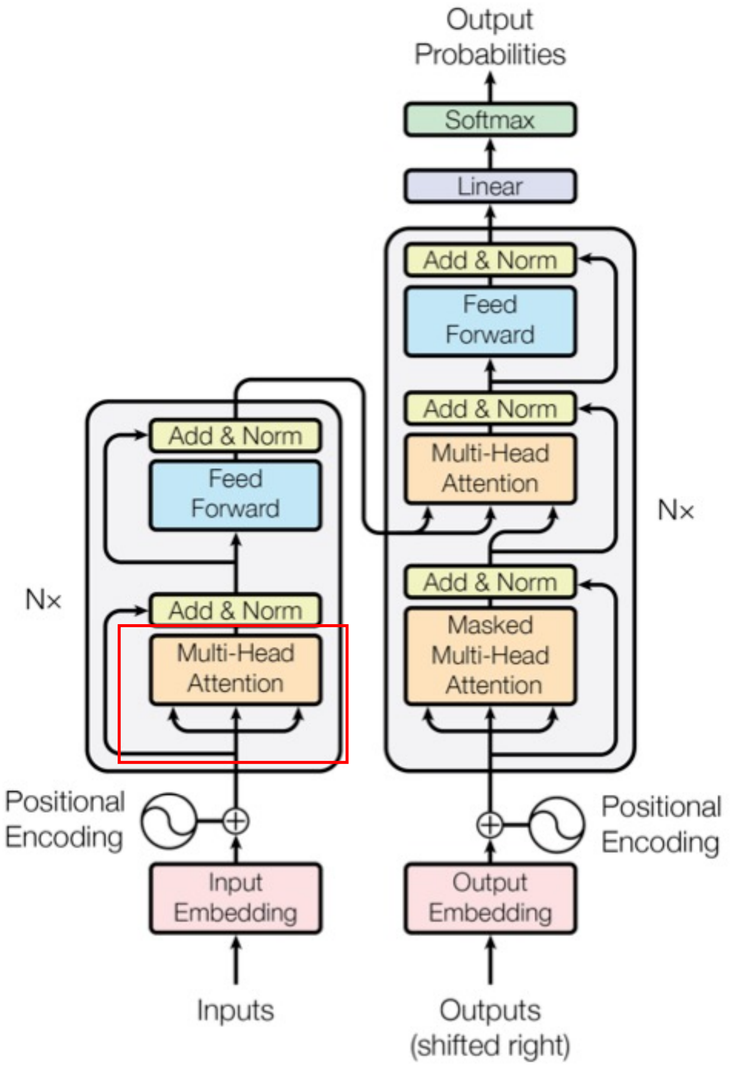
Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

Transformer에서의 attention 사용



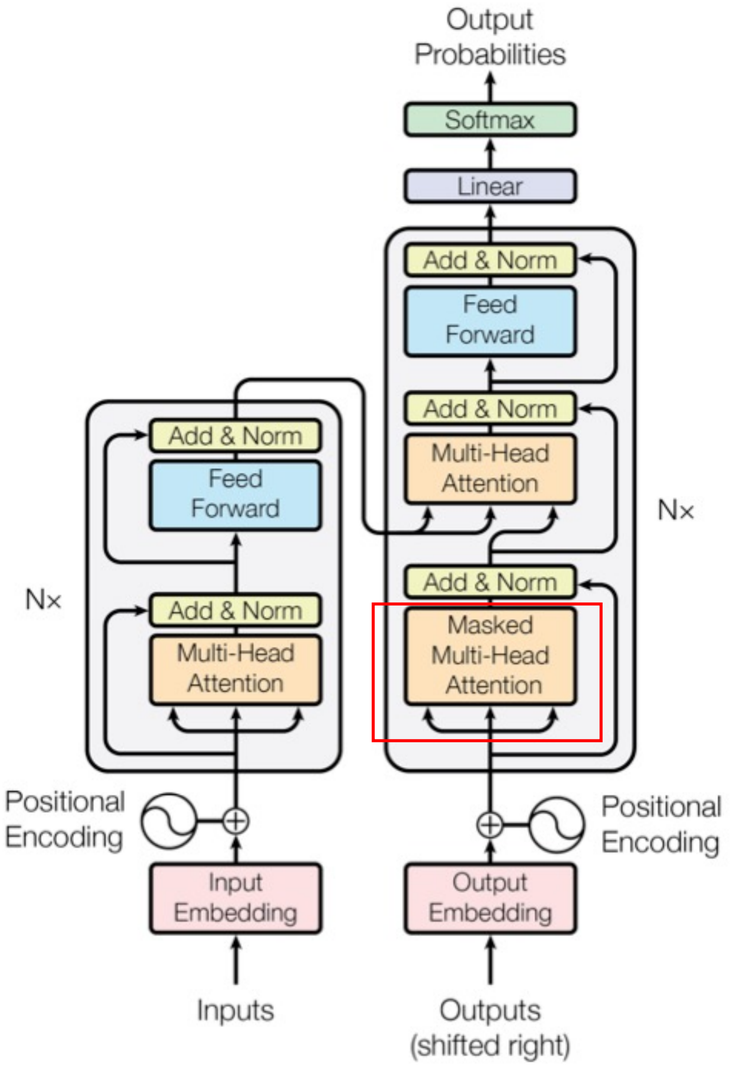
- “encoder-decoder attention”
  - Decoder에서만 사용
  - Queries는 이전 단계 decoder 층의 출력으로부터 산출
  - Keys, Values는 encoder의 출력으로부터 산출
  - Decoder의 모든 위치에서 입력 데이터의 모든 위치를 참고할 수 있도록 만듦

Transformer에서의 attention 사용

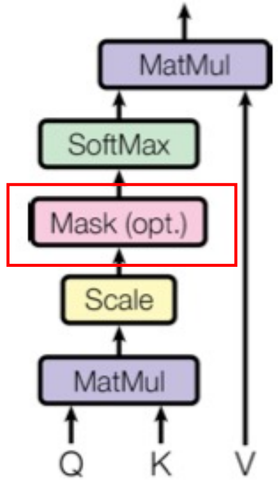


- Encoder self-attention
  - Queries, Keys, Values 모두 이전 단계 encoder 층의 출력으로부터 산출
  - Encoder의 각 위치에서 이전 층 encoder의 모든 위치를 참고할 수 있도록 만듦

Transformer에서의 attention 사용



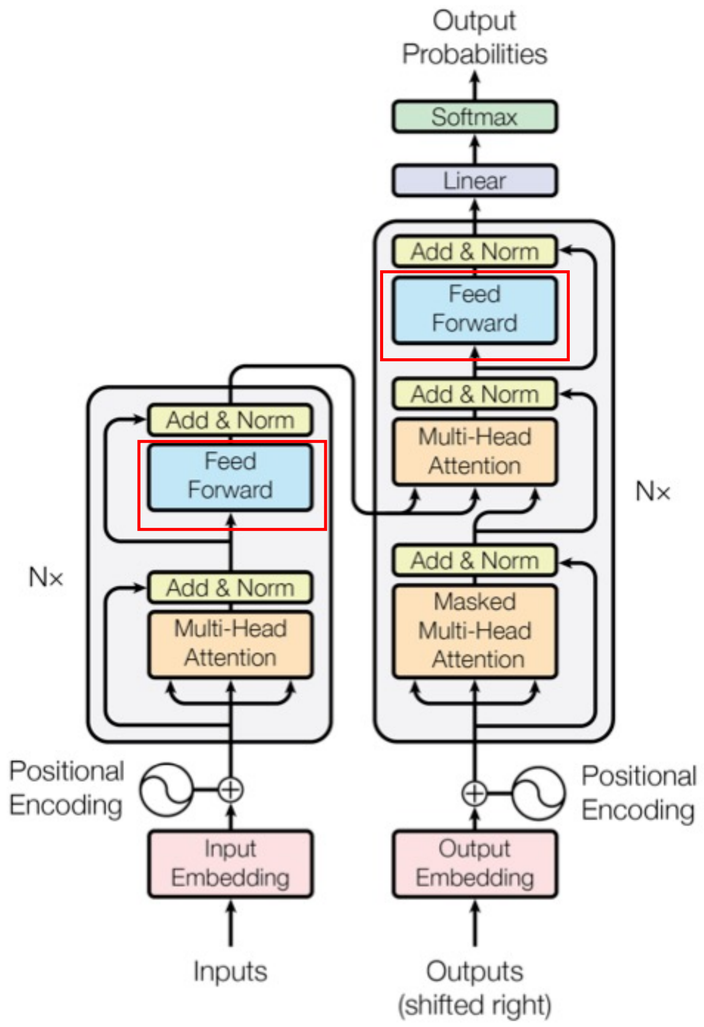
- Decoder self-attention
  - Queries, Keys, Values 모두 이전 단계 decoder 층의 출력으로부터 산출
  - Decoder의 각 위치에서 이전 층 decoder의 현재 위치까지 참고할 수 있도록 만듦
  - 현재 위치 뒤의 부분을 참고하지 않도록 scaled dot-product 단계에서 masking 시행



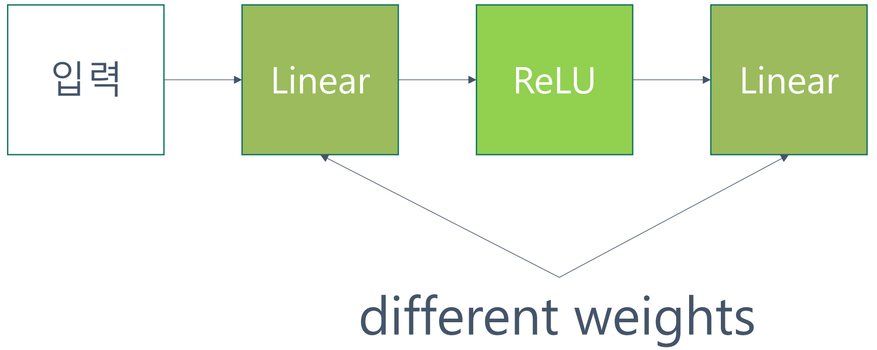
Decoder 병렬화

- Inference 시 현재 위치 뒤의 데이터를 알 수 없기 때문에 현재 위치의 계산이 완료되어야 다음 위치의 계산을 진행할 수 있음(Sequential)
- Decoder self-attention의 masking이 이를 처리하기 위한 것
- 하지만 training 시에는 모든 위치 데이터를 이미 알고 있으므로 병렬적으로 계산하여 학습 가능

Fully connected feed-forward network



- 각 층은 어텐션 이후 fully connected feed-forward network 적용



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

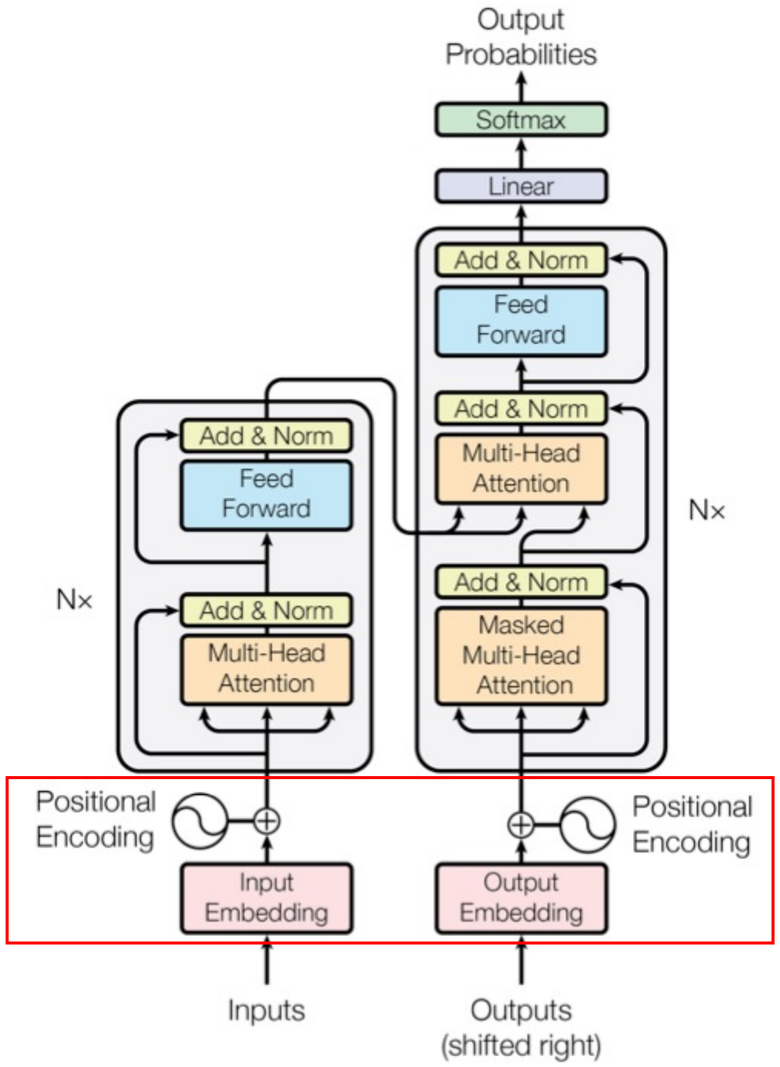
- 저자는 다음의 hyperparameter 사용
  - input, output dimension = d_{model} = 512
  - inner layer dimension = d_{ff} = 2048

Embedding

- 입력 token과 출력 token을 d_{model} 차원 벡터로 변환할 때 사전 학습된 embedding 사용
- Embedding layer와 마지막 softmax 전 linear transformation에서 같은 weight를 사용하는 weight tying 기법[8] 사용
- 다만 embedding layer 에서는 위 weight에 $\sqrt{d_{model}}$을 곱해서 사용

8. Ofir Press and Lior Wolf. Using the output embedding to improve language models. arXiv preprint arXiv:1608.05859, 2016.

Positional Encoding



- Transformer는 recurrence나 convolution을 사용하지 않기 때문에 데이터의 순서를 고려하지 못함
- 이에 대응하기 위해 positional encoding을 각 embedding 뒤에 더해줌
- Positinal encoding 결과는 embedding과 더해져야 하기 때문에 d_{model} 차원을 가짐
- 여러 가지 positional encoding이 있지만 저자는 다음의 positional encoding을 사용

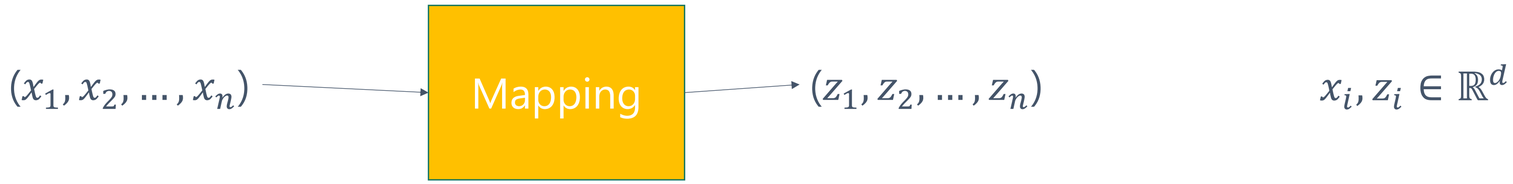
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

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

- 학습 데이터보다 긴 데이터가 들어와도 위의 공식은 처리할 수 있기 때문에 사용

Self-Attention의 장점

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



- 최신 machine translation 분야 모델들에서 사용되는 sentence representation의 경우 대부분 n < d -> Self-Attention이 복잡도 낮음
- Sequential Operations의 경우 Recurrent보다 Self-Attention이 낮음 -> 병렬화 가능한 연산이 많음
- Maximum Path Length의 경우 Self-Attnetion이 가장 낮음 -> 먼 거리 사이 의존성을 학습하기 쉬움

실험에 사용한 학습 환경

- 학습 데이터
  - 450만 문장 쌍의 WMT 2014 영어-독어 데이터셋
    - Byte-pair encoding
    - 37,000 tokens
  - 3,600만 문장의 WMT 2014 영어-불어 데이터셋
    - 32,000 word-piece 어휘
- 하드웨어 및 학습 시간
  - 8 개의 NVIDIA P100 GPU를 가진 기계 하나로 학습
  - Base model
    - 한 스텝 당 0.4 초
    - 총 100,000 스텝 or 12 시간 학습
  - Big model
    - 한 스텝 당 1 초
    - 300,000 스텝 학습(3.5 일)

실험에 사용한 학습 환경

- Optimizer
  - Adam 사용
  - $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$
  - Learning rate

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

- Regularization
  - Dropout
    - 각 sub-layer 이후 residual connection, layer normalization 전에 dropout 적용
    - Dropout 비율 = 0.1
  - Label Smoothing
    - $\epsilon_{ls} = 0.1$ 적용하여 label smoothing 시행
    - Label smoothing 결과 perplexity는 안 좋아졌지만 accuracy와 BLEU는 향상됨

기계 번역 결과

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

모델 변형 결과

	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

모델 변형 결과

- 영어-독어 번역 task로 성능 측정
- Attention head 개수는 너무 작아도 성능이 낮아지지만, 너무 높아도 성능이 낮아짐(attention key, value dimension은 일정한 계산 비용을 유지하도록 attention head 개수에 맞춰 조정)
- 다른 parameter가 모두 같을 때 attention key dimension을 내리면 성능 하락
- 전반적으로 모델이 커지면 성능이 향상됨
- Positional encoding 방법을 바꾸는 건 성능에 큰 영향 없음

Constituency Parsing 결과

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

- 번역 외 다른 task에서도 generalize가 잘 되는지 보기 위한 실험

결론

- 기존 encoder-decoder 구조 모델에서 많이 사용되던 recurrent 층을 없애고 attention에만 의존한 모델 제시
- 번역 task의 경우 적은 학습 시간으로 향상된 성능을 보여줌
- 추후 다른 task, 다른 modality의 데이터에도 적용해볼 것
- Sequential 계산을 줄이는 연구도 계속할 것