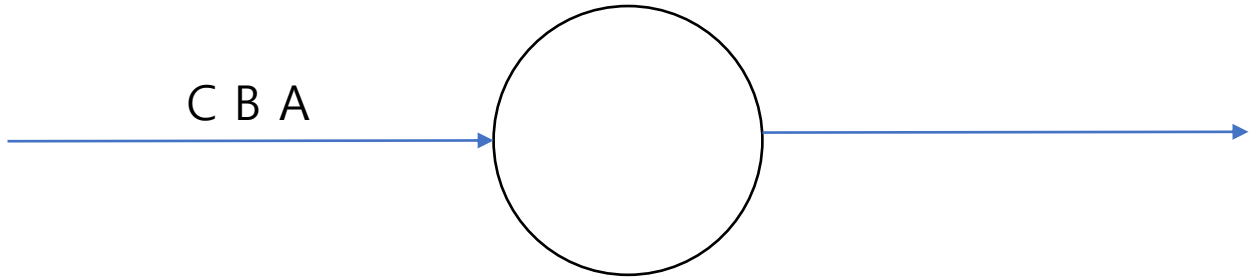
Attention Is All You Need

Self-Attention mechanism에 기반한 자연어 처리 모델 "Transformer"

인천대학교 컴퓨터공학부 강병하

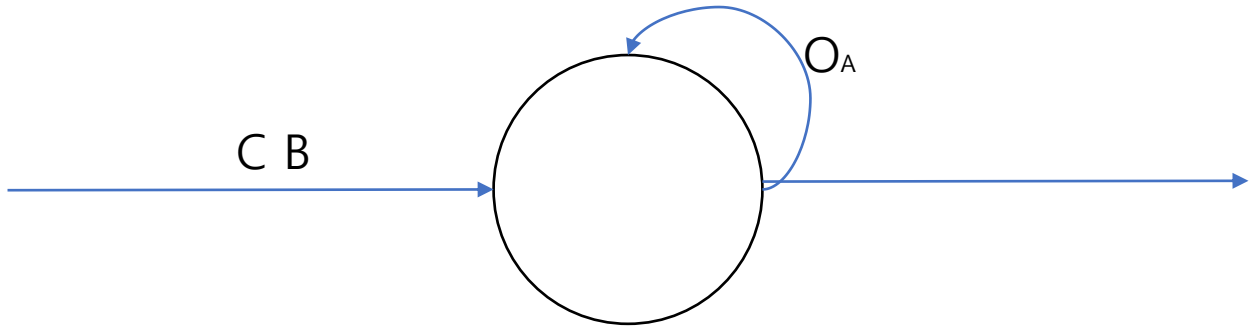
| 순환 신경망

- 텍스트 데이터는 순차데이터. 단어의 순서가 유지되어야 함.
- **이전에 입력한 데이터를 기억하는 기능이 필요 -> 순환신경망**



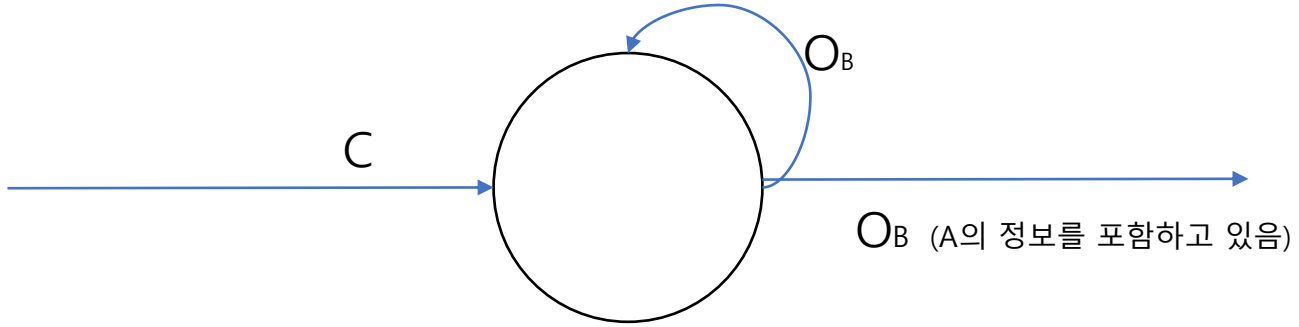
| 순환 신경망

- 텍스트 데이터는 순차데이터. 단어의 순서가 유지되어야 함.
- **이전에 입력한 데이터를 기억하는 기능이 필요 -> 순환신경망**



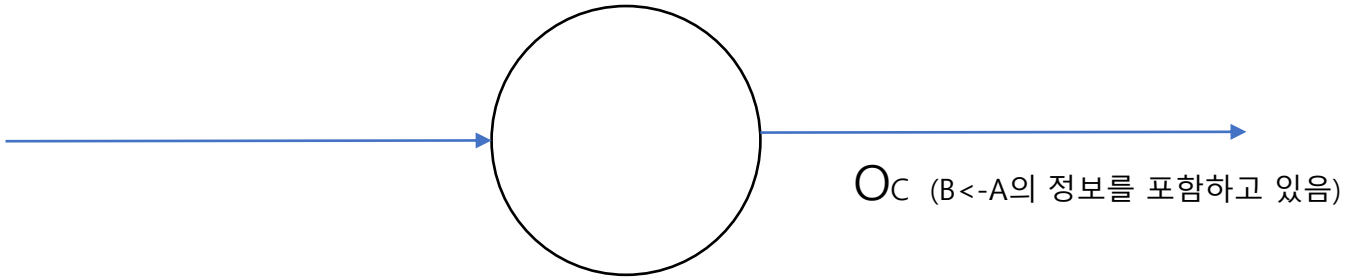
| 순환 신경망

- 텍스트 데이터는 순차데이터. 단어의 순서가 유지되어야 함.
- **이전에 입력한 데이터를 기억하는 기능이 필요 -> 순환신경망**



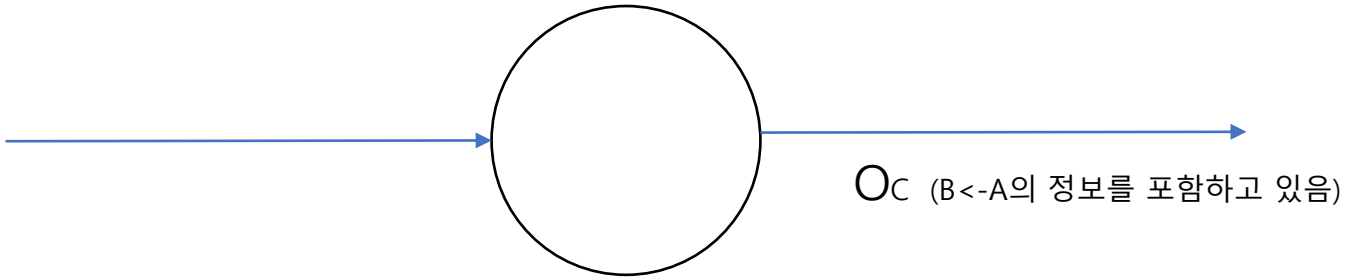
| 순환 신경망

- 텍스트 데이터는 순차데이터. 단어의 순서가 유지되어야 함.
- **이전에 입력한 데이터를 기억하는 기능이 필요 -> 순환신경망**



| 순환 신경망

- 텍스트 데이터는 순차데이터. 단어의 순서가 유지되어야 함.
- **이전에 입력한 데이터를 기억하는 기능이 필요 -> 순환신경망**

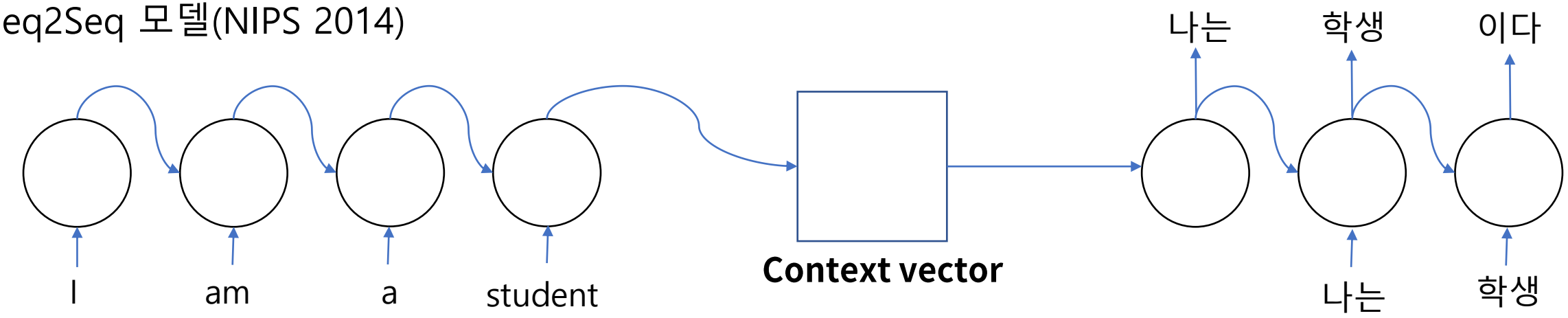


입력과 이전 타임스텝의 **은닉상태(hidden state)**를 사용하여 순서에 대한 정보를 포함한다.

| 순환 신경망(RNN) 기반 모델의 한계

- RNN(1986), LSTM(1997), GRU 등이 자연어처리 분야에서 주로 사용되었음.
- 하지만 RNN기반 모델은 **긴 문장을 처리하는 데 한계를 가짐.**

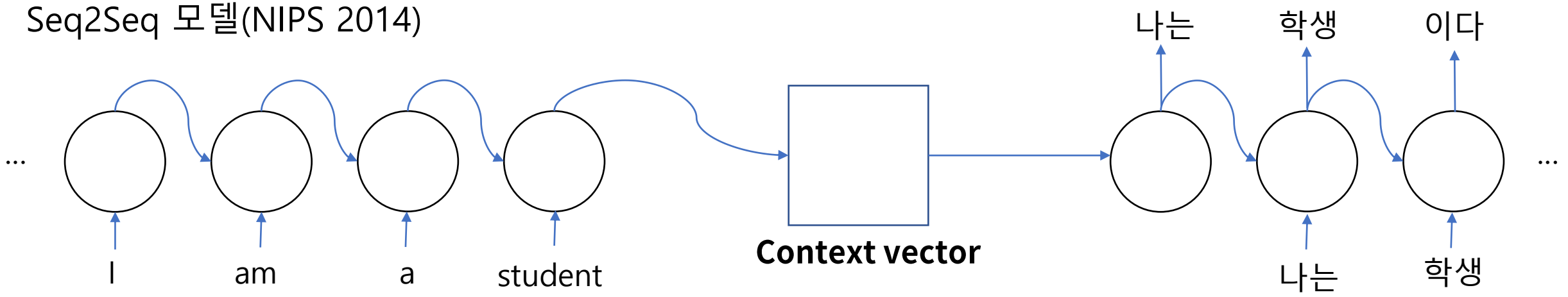
Seq2Seq 모델(NIPS 2014)



| 순환 신경망(RNN) 기반 모델의 한계

- 앞 단의 셀에 의존적(**병렬화** 불가능) -> **연산 속도 느려짐**
- **고정된** 크기의 Context vector에 **압축(정보의 손실)** -> **정확도 떨어짐**

Seq2Seq 모델(NIPS 2014)



문제와 해결 방안

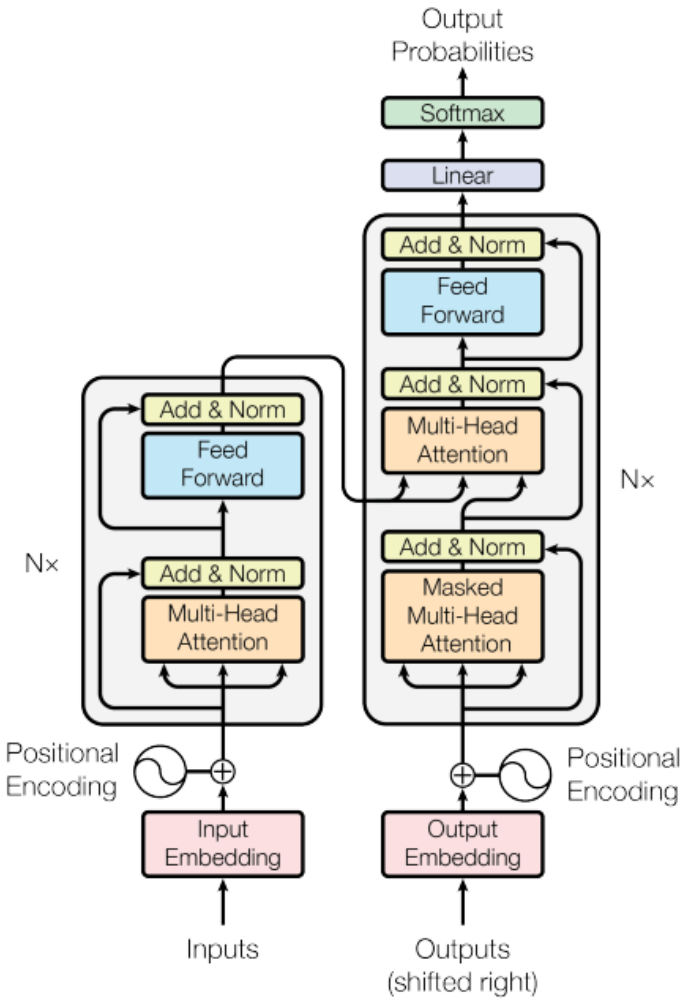
문제 : RNN의 순차적인 구조

해결 : RNN을 사용하지 않는 새로운 구조의 기계 번역 모델 개발 -> **Transformer**

- RNN없이 **어떻게** 순차 데이터의 **순서 정보**를 포함할 것인가?
- 은닉상태(hidden state)를 사용하지 않고 **어떻게 문맥에 대한 정보를 참조**하여 정확도를 높힐 것인가?

| Transformer의 구조

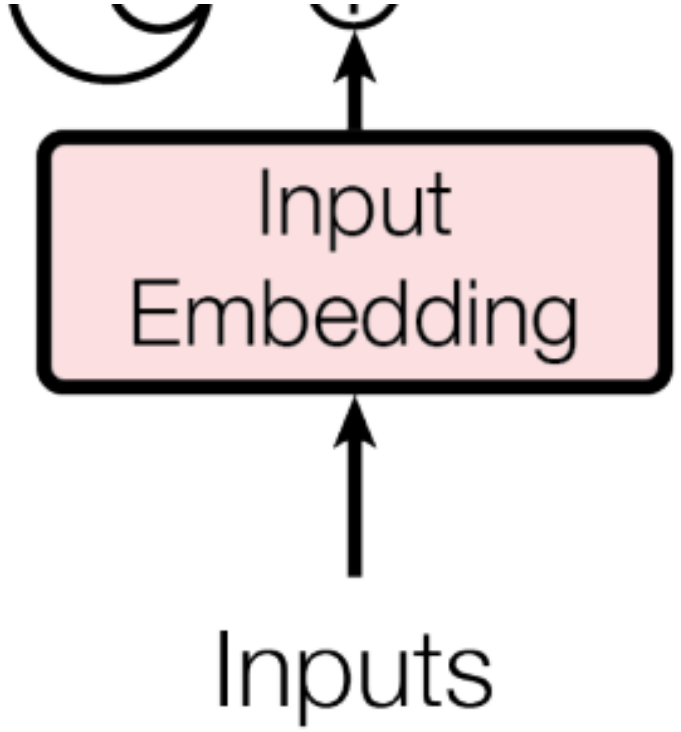
- Embedding
- Positional Encoding
- N(=6)개의 인코더와 N개의 디코더
- Linear
- Softmax



| Transformer의 구조 : Embedding

텍스트를 컴퓨터가 적절히 연산할 수 있도록

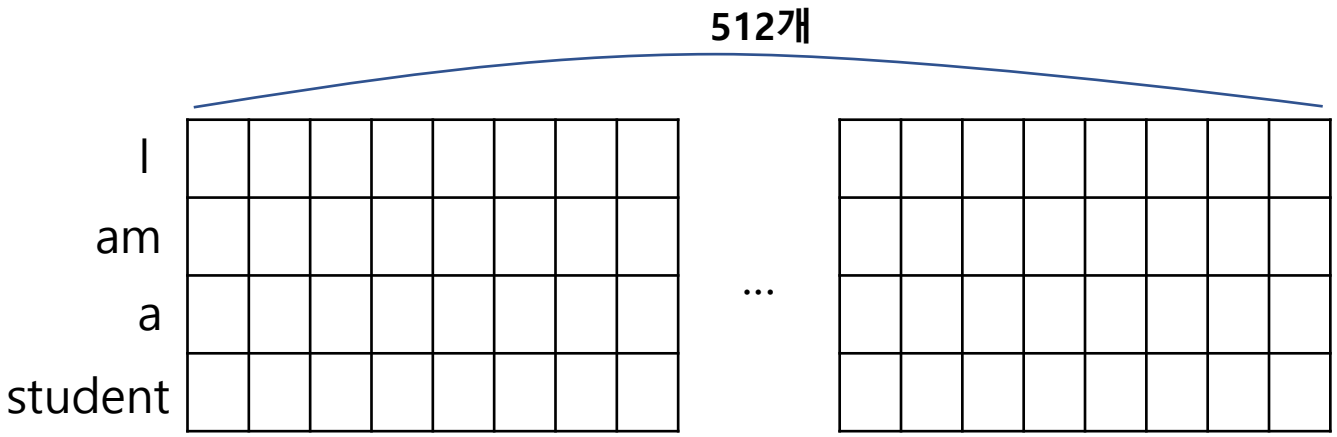
입력된 문장을 **수치화(벡터화)** 하는 작업



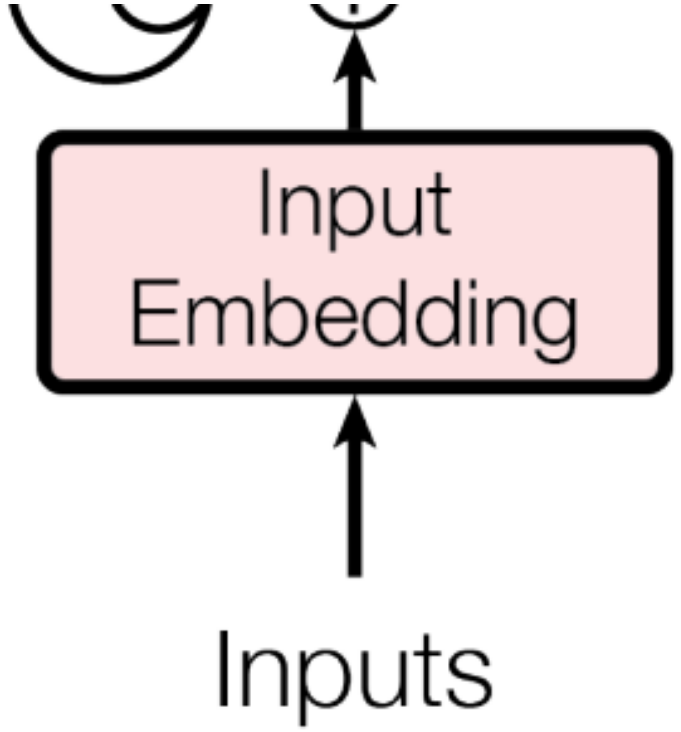
| Transformer의 구조 : Embedding

텍스트를 컴퓨터가 적절히 연산할 수 있도록

입력된 문장을 **수치화(벡터화)** 하는 작업



dimension $d_{\text{model}} = 512.$

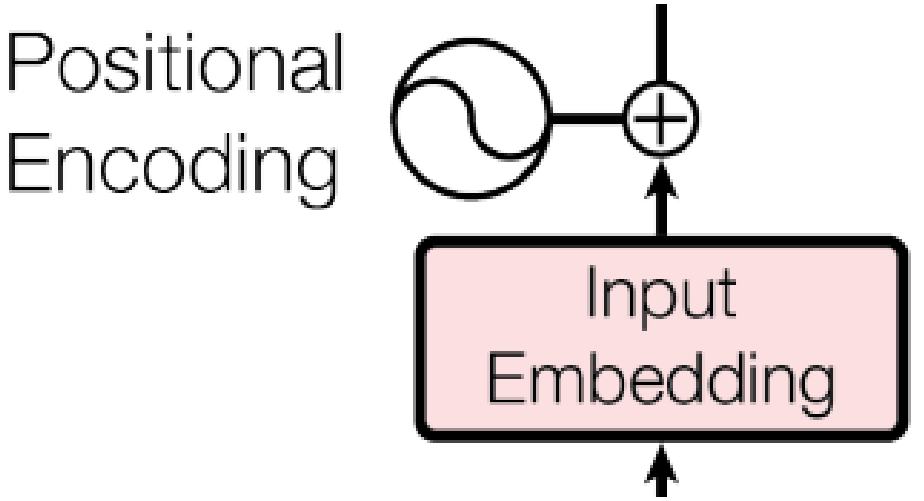


| Transformer의 구조 : Positional Encoding

RNN없이 각 단어의 **순서(위치) 정보를 포함시키는 방법**

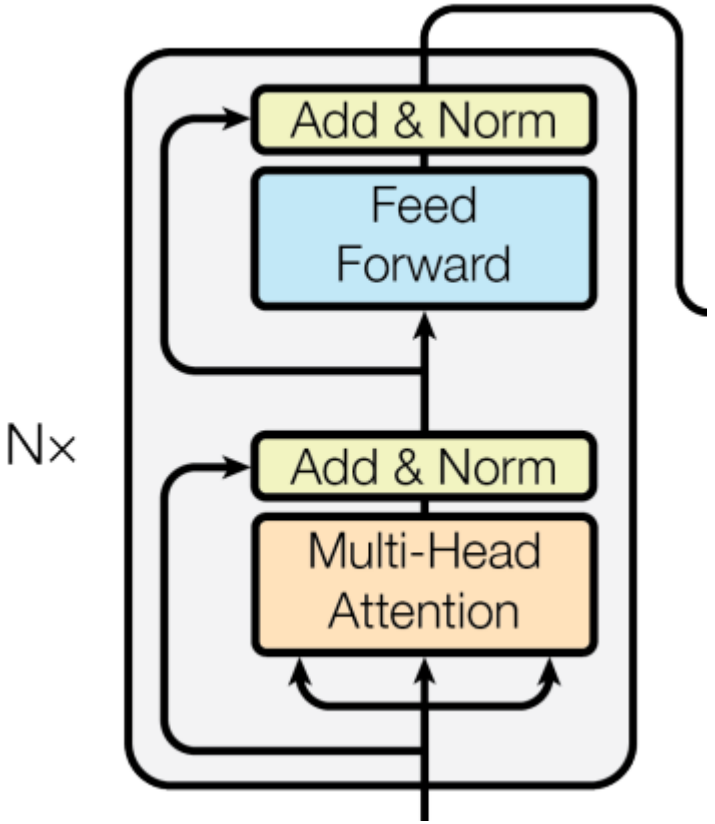
임베딩된 행렬과 같은 크기의 위치 정보를 담은 행렬과
Embeding Matrix를 더해준다. (Element Wise Sum)

-> **RNN을 제거하였을 때 생기는 문제(위치 정보 손실) 해결**



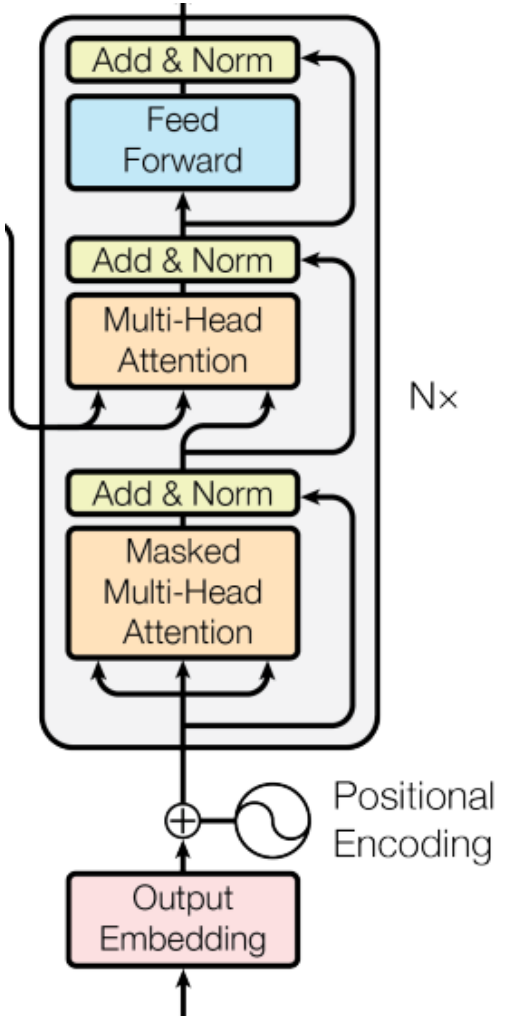
| Transformer의 구조 : Encoder Layer

- N(=6)개의 레이어
- **Multi-Head Attention**와 **Feed Forward**로 구성
- 각 작업이 끝나고 잔여학습과 정규화
- 이전층의 출력이 다음층의 입력으로 들어감.



| Transformer의 구조 : Decoder Layer

- 인코더와 마찬가지로 N(=6)개의 레이어
- **인코더 마지막 레이어의 출력 값**을 참조하는 Multi-Head Attention
  (각 레이어가 모두 참조)
- **출력되고 있는 문장의 정보**를 참조하는 Masked Multi-Head Attention
- Feed Forward
- 각 작업이 끝나고 잔여학습과 정규화



| Attention?

- 주의, 주목, 집중
- 각 단어가 문장 내에서 어떤 단어와 연관성을 가지는가 = 어떤 단어에 주의, 주목, 집중해야 하는가
- **Self-Attention**

The student **who** likes to study.

다른 단어보다 student에 보다 주목해야 함

| Multi-Head Attention

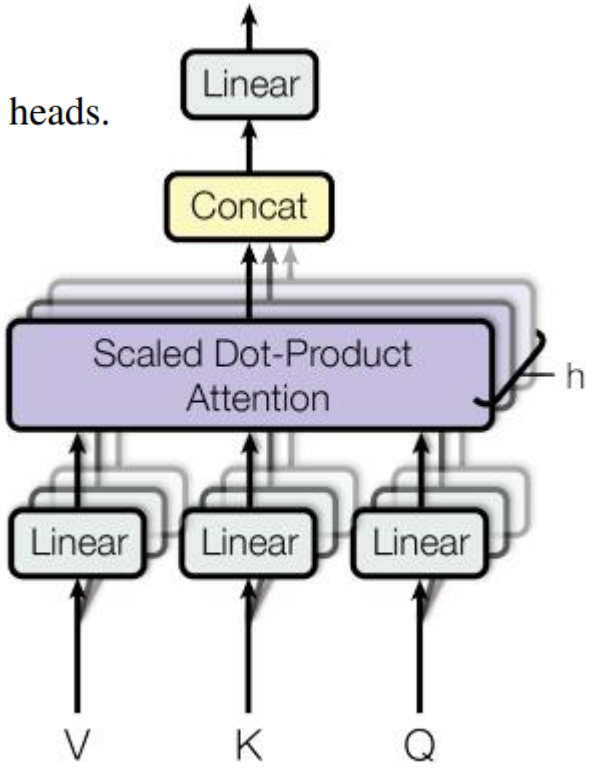
we employ $h = 8$ parallel attention layers, or heads.

Scaled Dot-Product Attention을 여러 개 사용 (8개)

Why?

-> 자기 자신이 아닌 다른 단어에 attention 하는 능력 향상

-> 각각의 Head가 서로 다른 단어에 attention 하기에 이들을 조합하여 보다 정교한 단어 간의 연관성 파악 가능

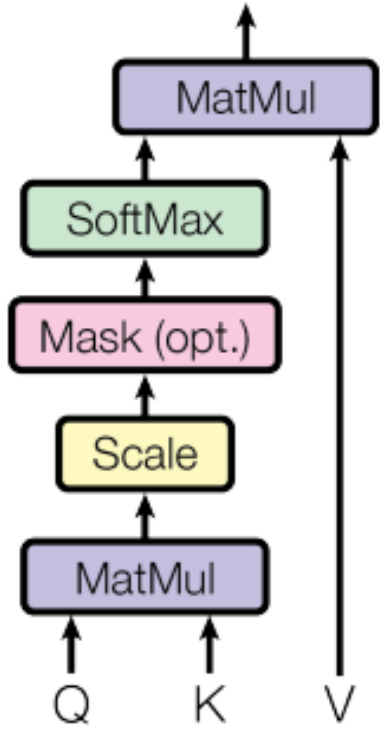


Scaled Dot-Product Attention

Q(query): 어디에 Attention해야 할지를 묻는 주체 단어

K(key): Attention의 대상(문장의 각 단어)

V(value): Attention을 산출할 때 사용



Scaled Dot-Product Attention

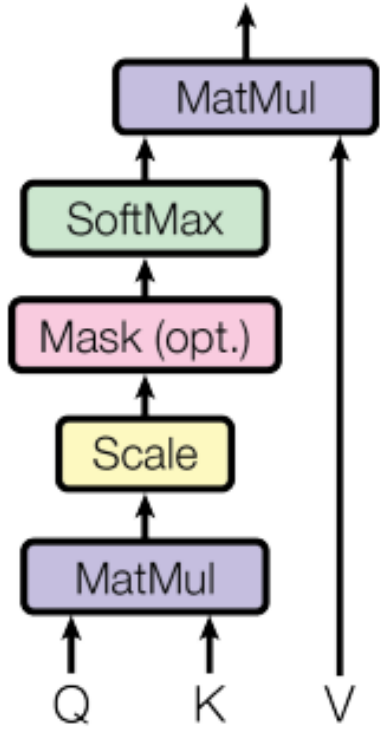
Q(query): 어디에 Attention해야 할지를 묻는 주체 단어

K(key): Attention의 대상(문장의 각 단어)

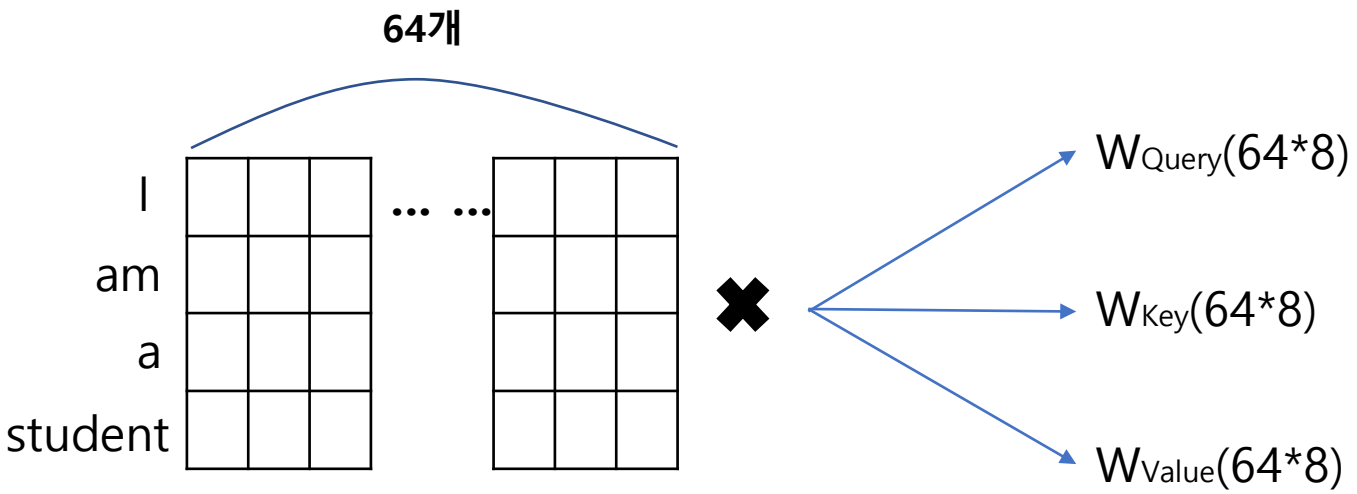
V(value): Attention을 산출할 때 사용

512차원의 문장벡터를 헤더의 개수만큼 나눔 -> 512/8 = 64

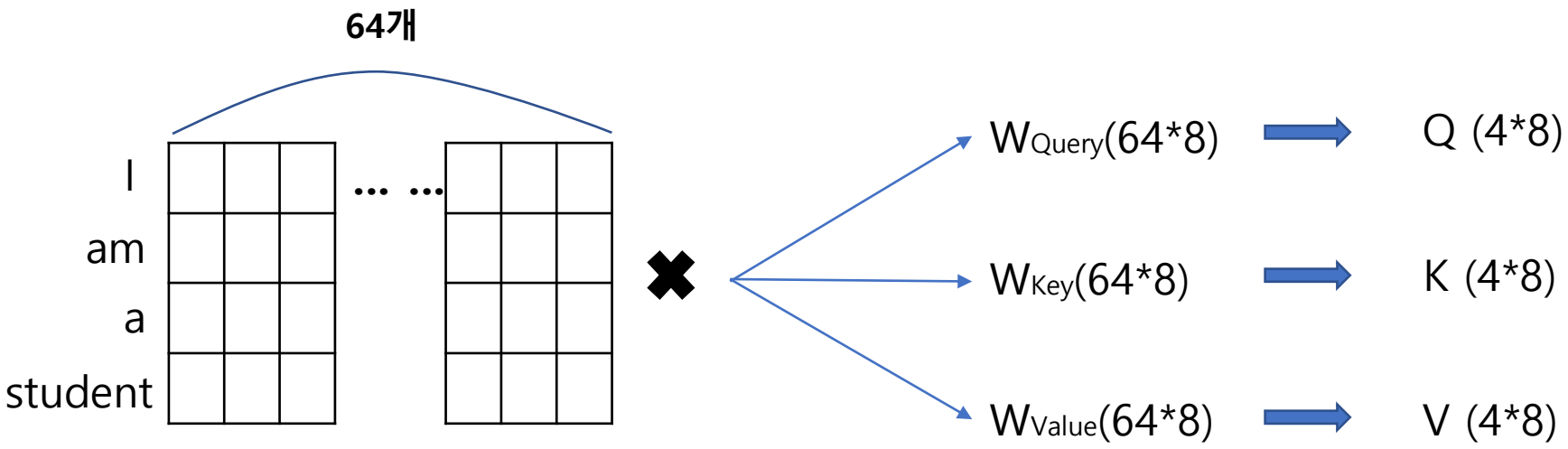
Query, Key, Value벡터를 각각 구하여 attention 수행



| Scaled Dot-Product Attention – Q, K, V 구하기



| Scaled Dot-Product Attention – Q, K, V 구하기



| Scaled Dot-Product Attention – Attention 수행

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

Q (4*8) × **K** (8*4) ÷ $\sqrt{d_k}$ × **V**(4*8) + ⟶ Attention (4*8)

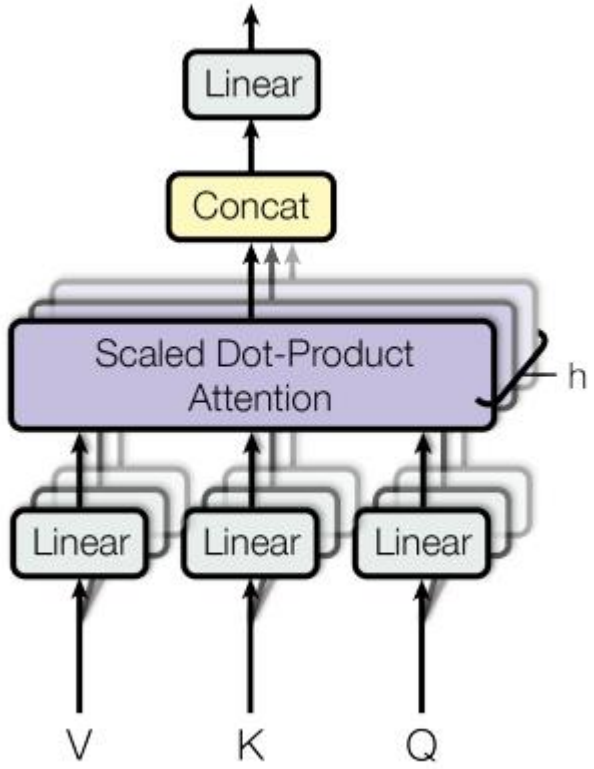
(4*4)

⬆

softmax()

| Multi-Head Attention

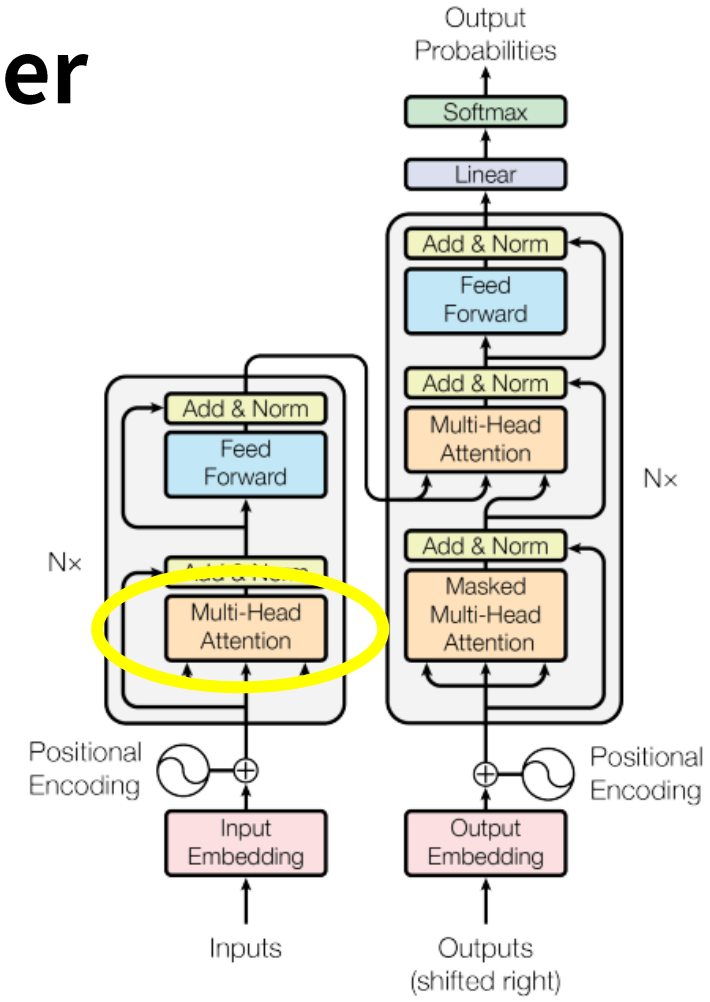
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$



| Multi-Head Attention in Encoder

인코더의 **Multi-Head Attention**

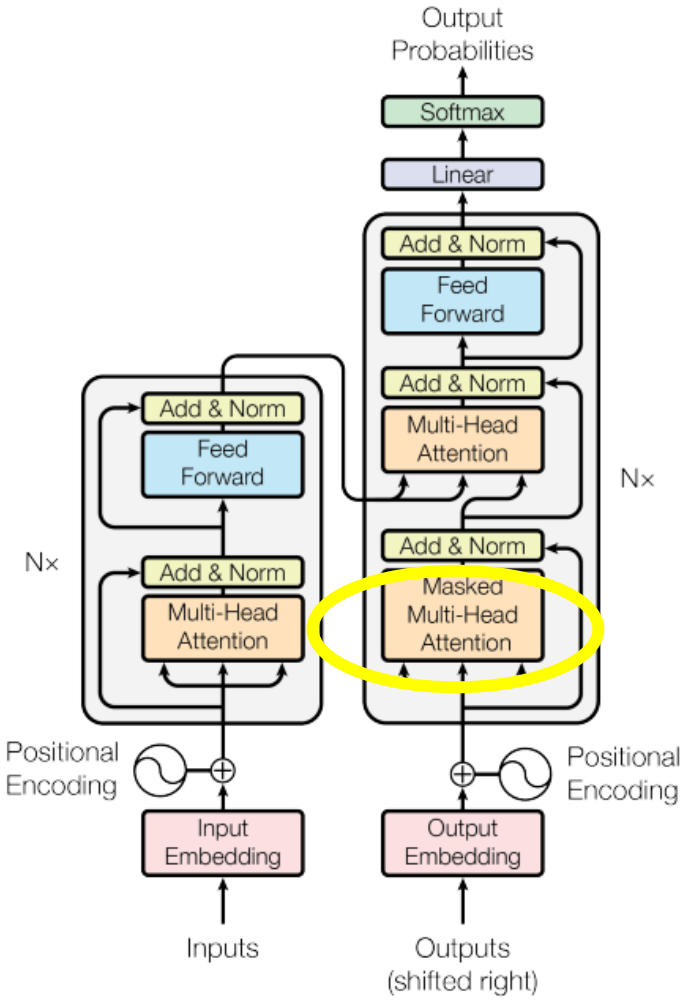
-> 입력된 문장의 전체적인 연관성 파악



| Masked Multi-Head Attention

디코더의 아웃풋을 참조하는
디코더의 Masked Multi-Head Attention

->앞에서 출력되었던 단어들과의 연관성만 파악



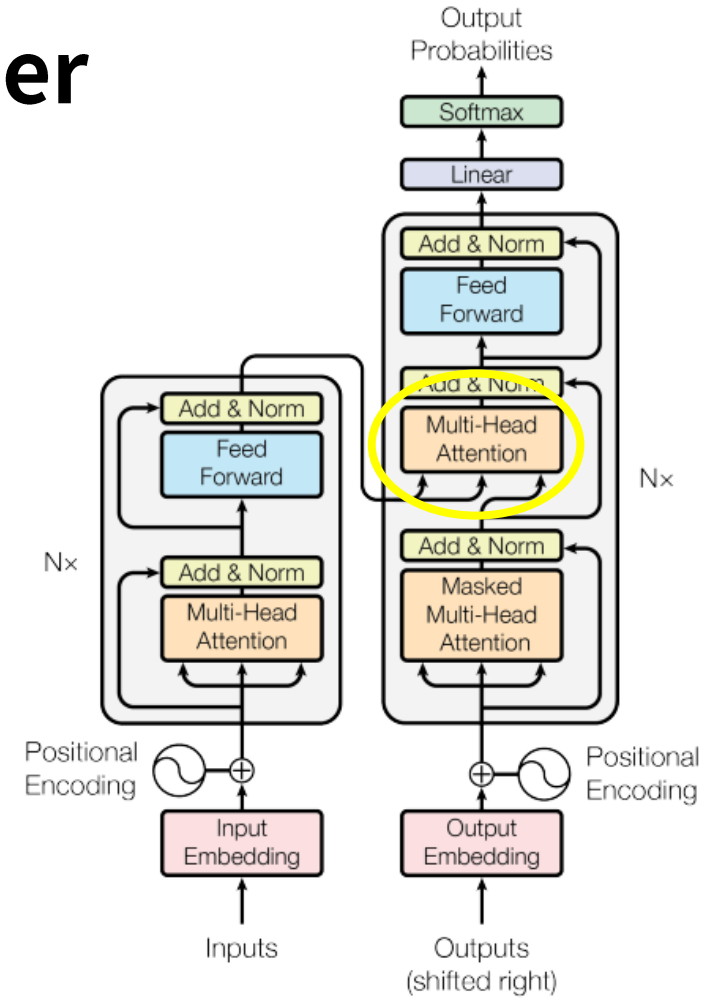
| Multi-Head Attention in Decoder

인코더의 아웃풋을 참조하는
디코더의 Multi-Head Attention

인코더 마지막 레이어의 결과를 **Key**와 **Value**로 참조

Query는 **디코더**에서 참조.

-> **출력하고 있는 단어**가 **소스 문장의 단어**들과 어떤 연관성이 있는지 파악



| Why Self-Attention

- 계산의 병렬화 -> 한번에 많은 계산을 할 수 있음
- 긴 문장에서 멀리 있는 단어에도 연관성을 찾아낼 수 있음 -> **long-term dependency** 문제 해결

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

| BLUE score 비교

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

결론

RNN이나 CNN을 사용하지 않고 오직 Attention에만 의존하는 새로운 모델, **Transformer**

Self-Attention으로 병렬화하여 **긴 문장도 빠르고 정교하게 처리**

현대 자연어 처리 분야에 중대한 영향