# Transformer

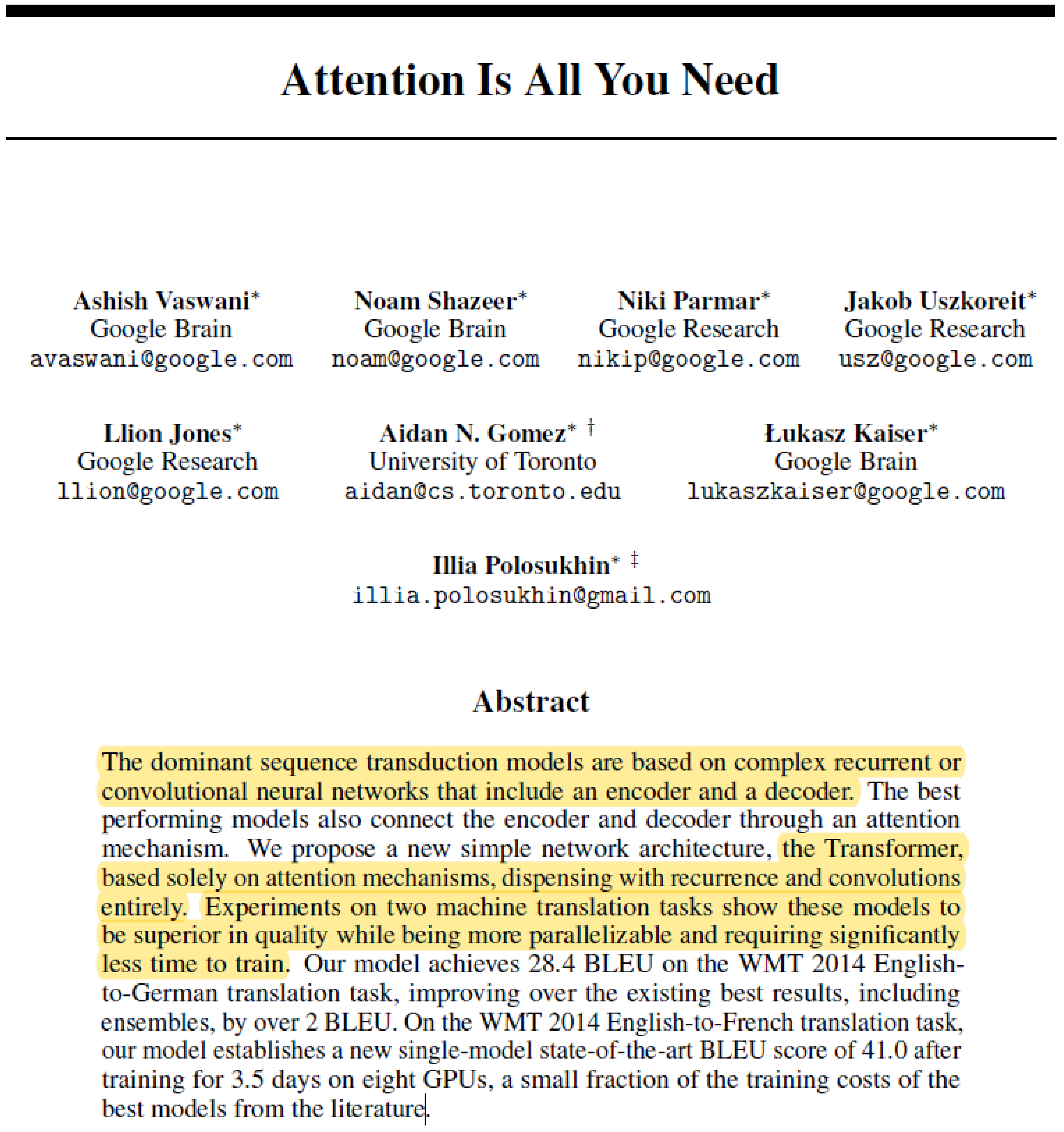
**Attention Is All You Need, NIPS 2017**

Email : ghy546@naver.com

GitHub : github.com/changyong93

Velog : https://velog.io/@changyong93

## 0. Abstract



# 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**

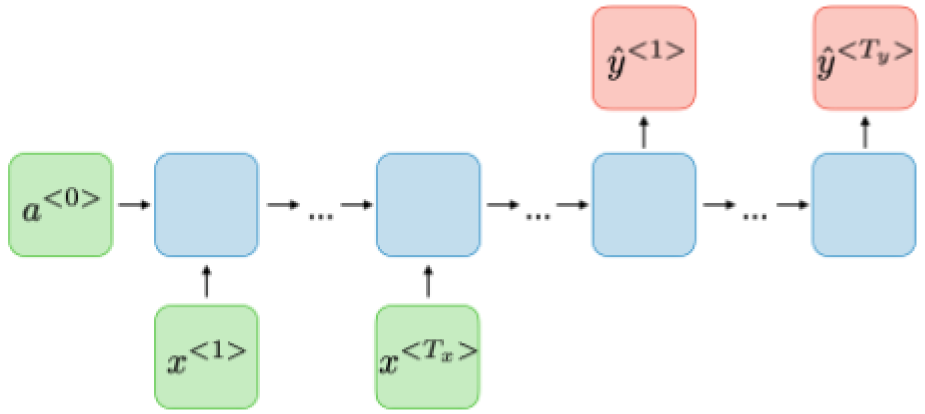
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, 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 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

- ✓ 기존 Seq2seq model은 RNN & CNN with Encoder-decoder가 지배적
- ✓ Transformer는 RNN이나 CNN이 아닌 attention으로만 구성
- ✓ 뛰어난 기계 번역 성능 ➔ 28.4 BLUE (EN-DE) , 41.0 BLEU (EN-FR, p.8)
- ✓ 병렬화가 용이하며 학습 시간이 단축됨 ➔ Training for 3.5 days on 8-GPUs

## 1. Introduction

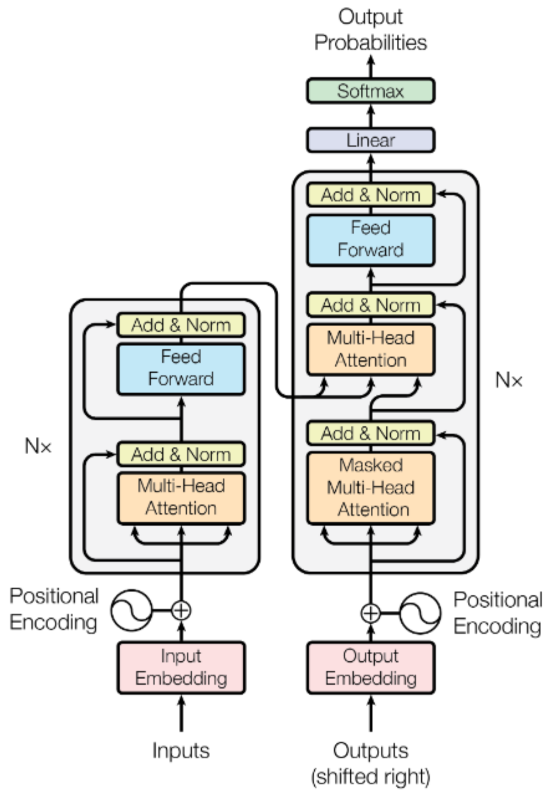
### Old) Recurrent Neural Network

- ✓ E.g.)Many-to-many(machine translation)
- ✓ This inherently sequential nature precludes parallelization
- ✓ critical at longer sequence length



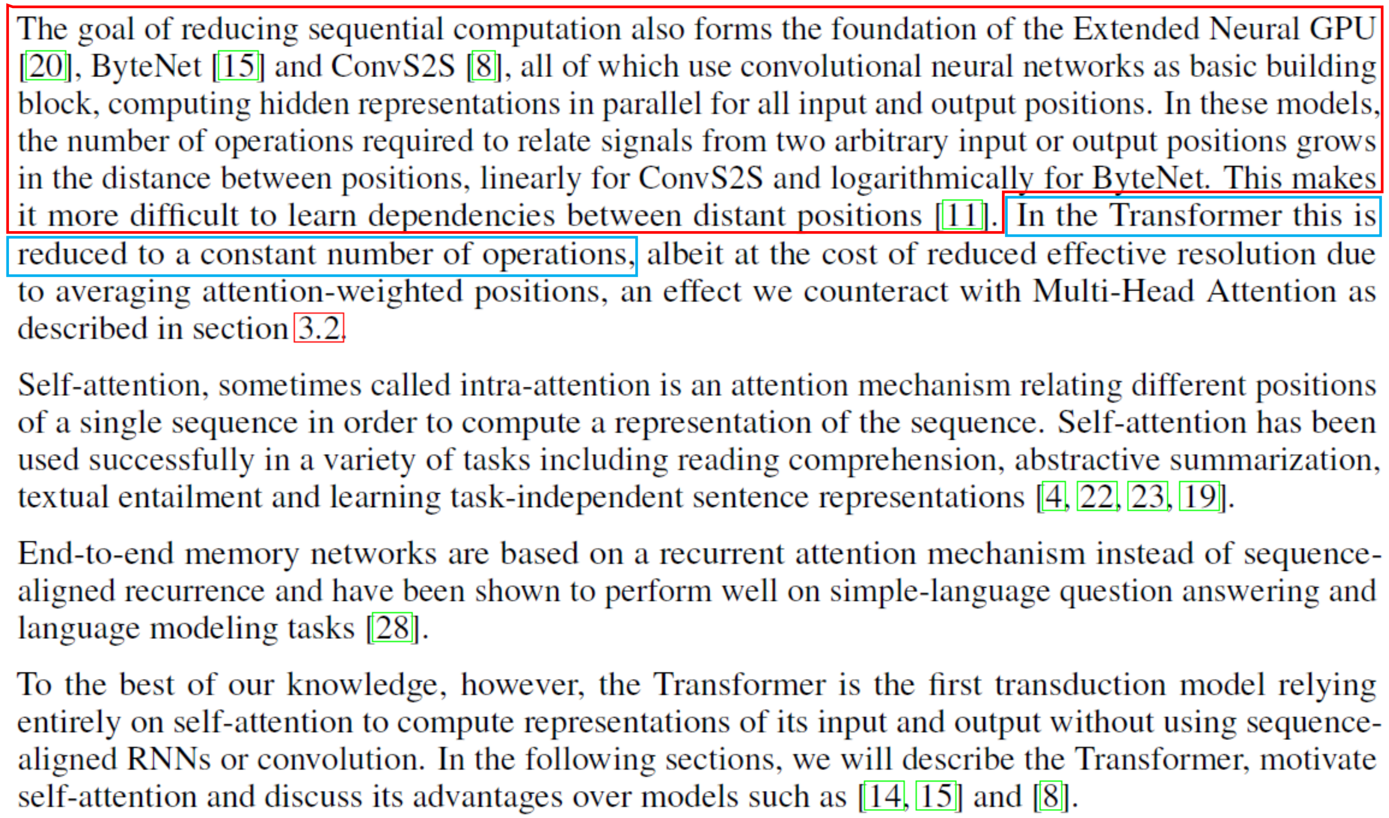
### New) Transformer

- ✓ become an integral part of compelling sequence modeling and transduction models in various tasks.
- ✓ modeling of dependencies without regard to their distance (not all cases, e.g. Natural Language Inference)
- ✓ relying entirely on an attention mechanism



- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networksThis inherently
- http://jalammar.github.io/illustrated-transformer/

## 2. Background



The goal of reducing sequential computation also forms the foundation of the Extended Neural GPU [20], ByteNet [15] and ConvS2S [8], all of which use convolutional neural networks as basic building block, computing hidden representations in parallel for all input and output positions. In these models, the number of operations required to relate signals from two arbitrary input or output positions grows in the distance between positions, linearly for ConvS2S and logarithmically for ByteNet. This makes it more difficult to learn dependencies between distant positions [11]. In the Transformer this is reduced to a constant number of operations, albeit at the cost of reduced effective resolution due to averaging attention-weighted positions, an effect we counteract with Multi-Head Attention as described in section 3.2.

Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. Self-attention has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations [4, 22, 23, 19].

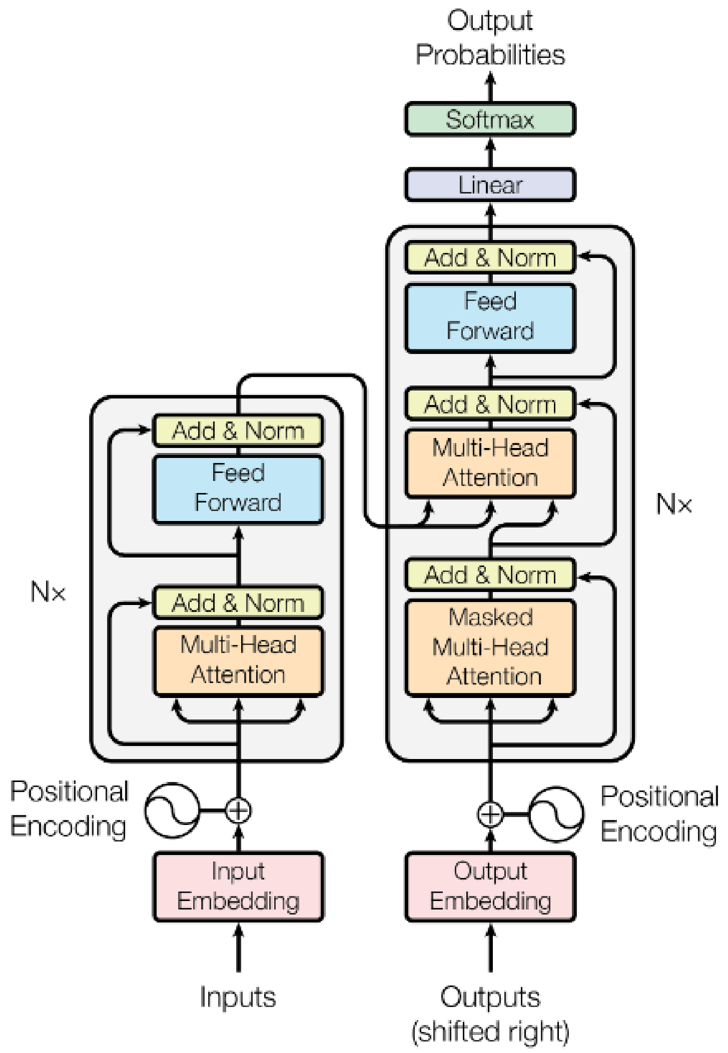
End-to-end memory networks are based on a recurrent attention mechanism instead of sequence-aligned recurrence and have been shown to perform well on simple-language question answering and language modeling tasks [28].

To the best of our knowledge, however, the Transformer is the first transduction model relying entirely on self-attention to compute representations of its input and output without using sequence-aligned RNNs or convolution. In the following sections, we will describe the Transformer, motivate self-attention and discuss its advantages over models such as [14, 15] and [8].

- ✓ Sequence length에 따른 연산량을 줄이기 위한 CNN 기반 다수의 모델들
- ✓ Computing hidden vectors in parallel for all input and output.
- ✓ Input과 output의 거리에 따라 연산량 증가 ➔ linear(ConvS2S), log(ByteNet)
- ✓ Transformer에서는 연산량 고정 ➔ 거리와 연관 X

- https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networksThis inherently

## 3. Model Architecture

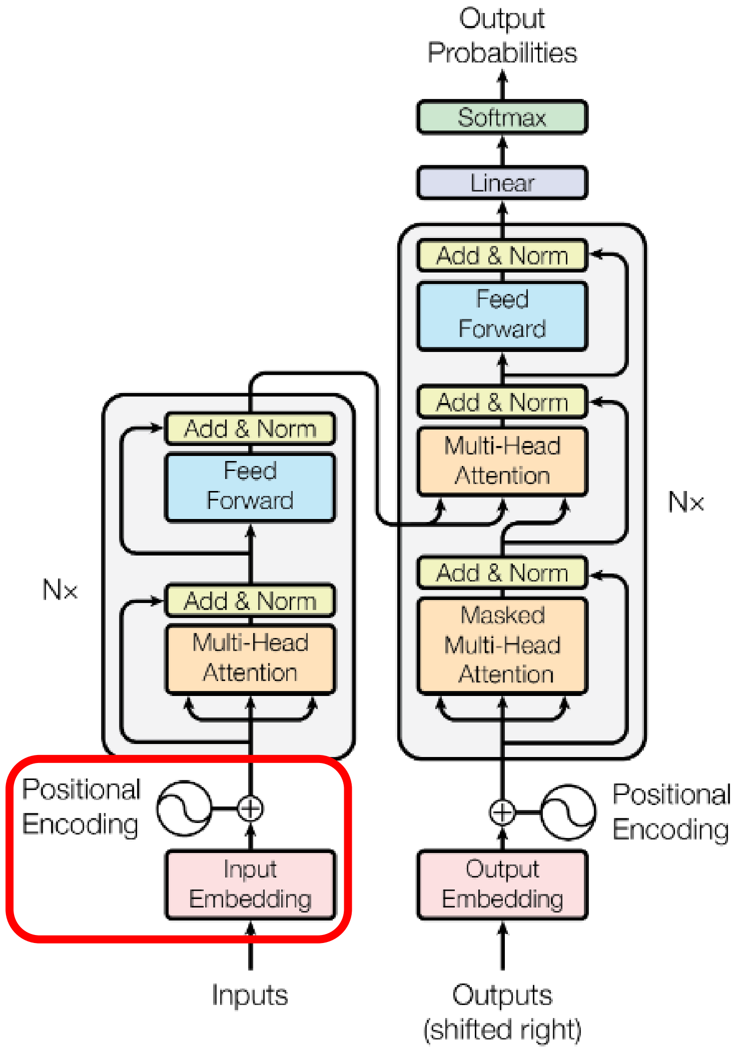


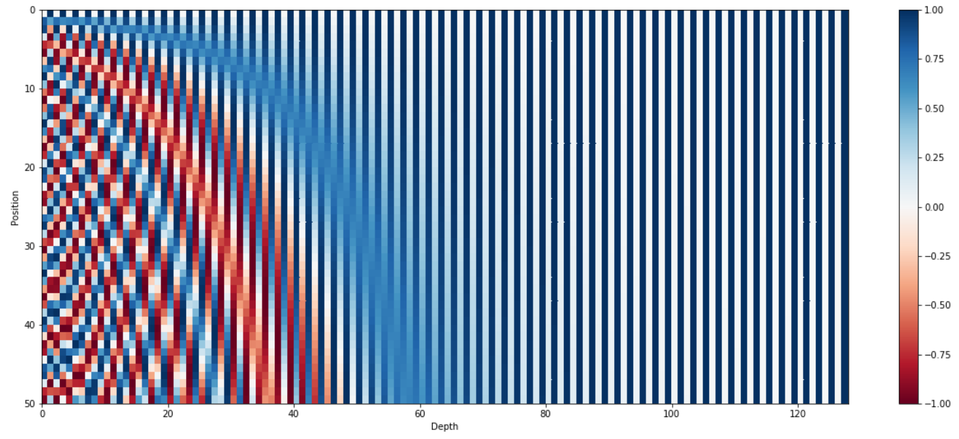
- ✓ Trnasformer = encoder *6 + decoder*6
- ✓ Encoder :
  Input + Positional encoding ➔ MHA + skip connection ➔ Layer Norm ➔
  FFN + skip connection ➔ Layer Norm ➔ output
- ✓ Decoder :
  output for previous step + Positional Encoding ➔ Masked MHA + skip connection ➔ Layer Norm ➔
  MHA with Q,K of encoder + skip connection ➔ Layer Norm ➔ FFN ➔ Softmax ➔ output

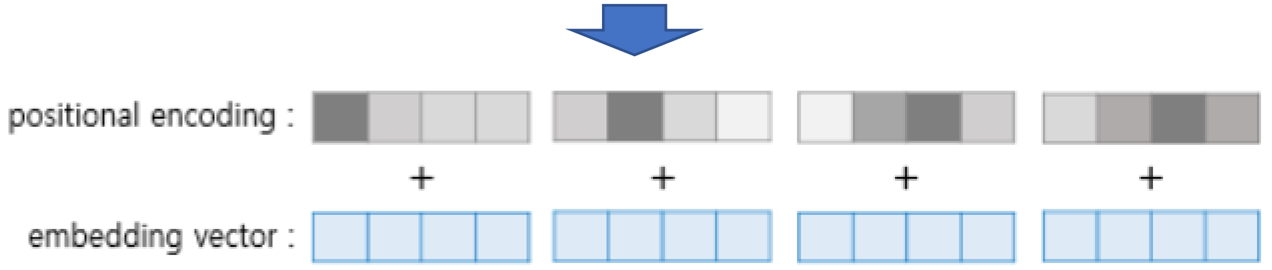
## 3.1 Model Architecture - encoder - Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \qquad PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)$$

현재 과정





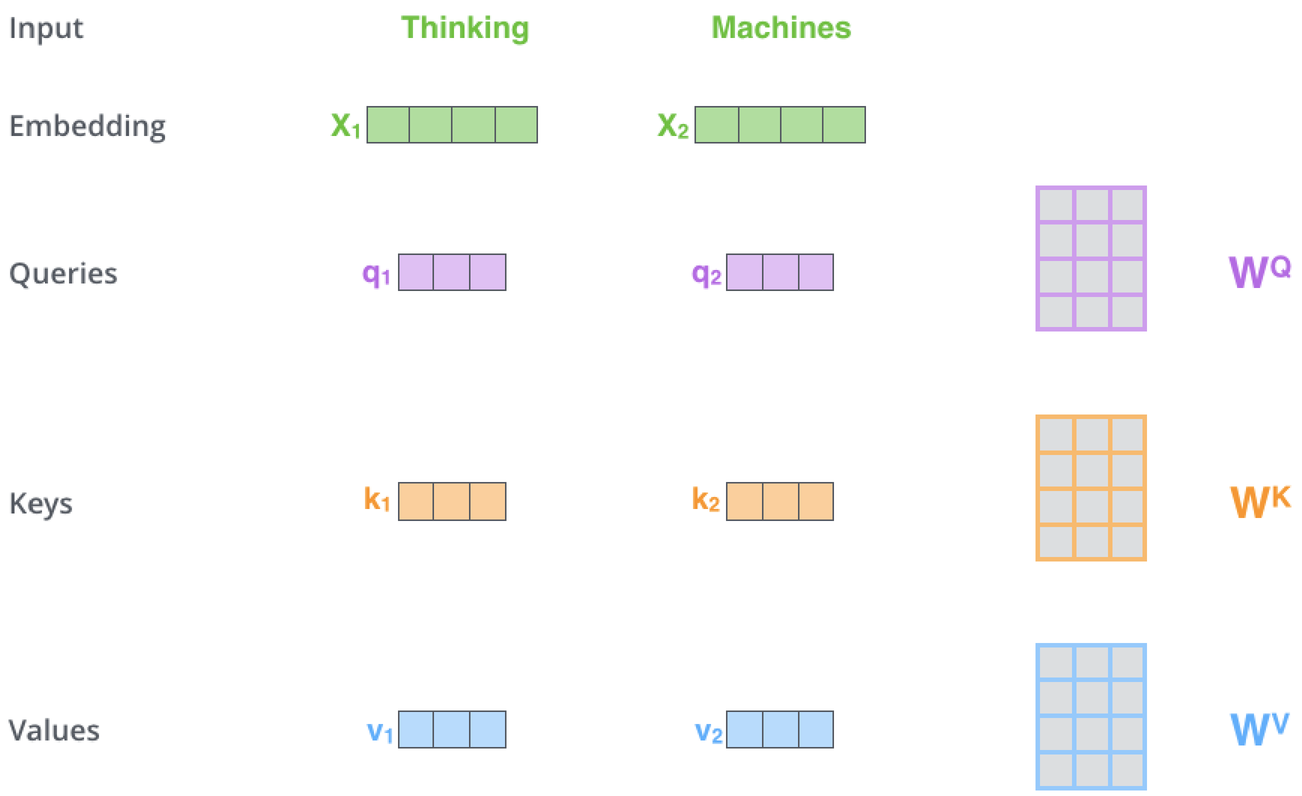


- ✓ pos: sequence 내 index
- ✓ i : dimension의 index

- ✓ 실제로는 $PE_{\{pos+K\}}$로 사용 ➔ pos에 k offset을 추가함으로써 $PE_{pos}$의 linear function으로 볼 수 있어서 학습하기가 용이하다고 함..(잘 이해는 안감)

- http://nlp.seas.harvard.edu/2018/04/03/attention
- https://wikidocs.net/31379

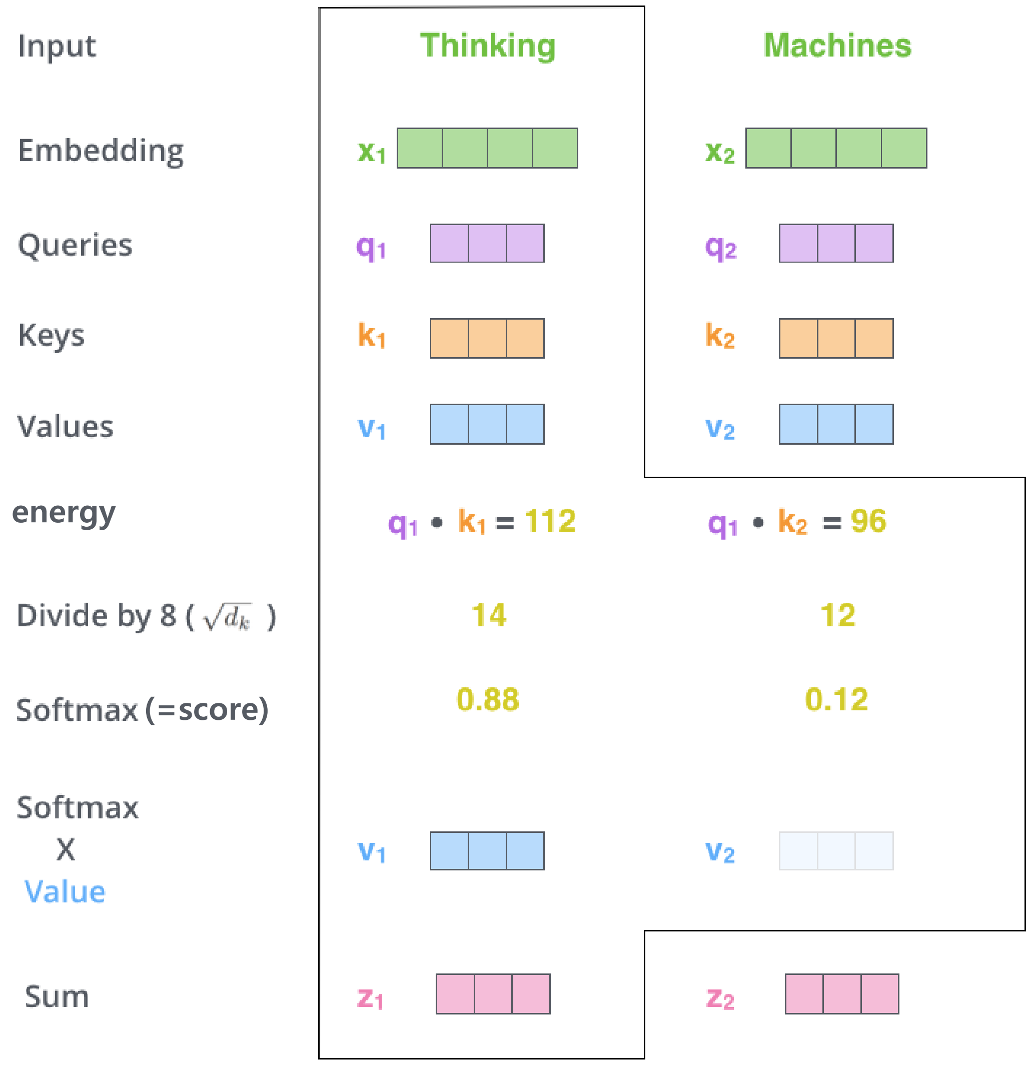
## 3.2 Model Architecture - encoder - Self-attention



✓ liear transformation ➔ Query, Key, Value Vectors

- https://jalammar.github.io/illustrated-transformer/

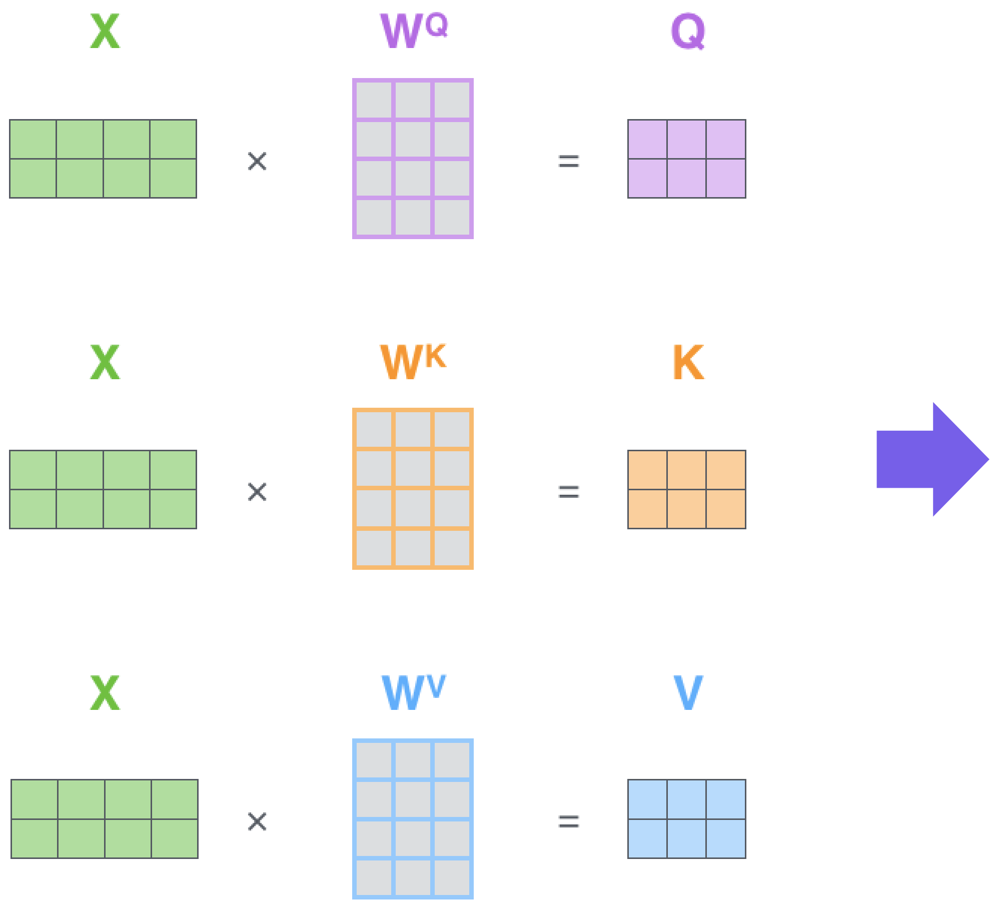
## 3.2 Model Architecture - encoder - Self-attention

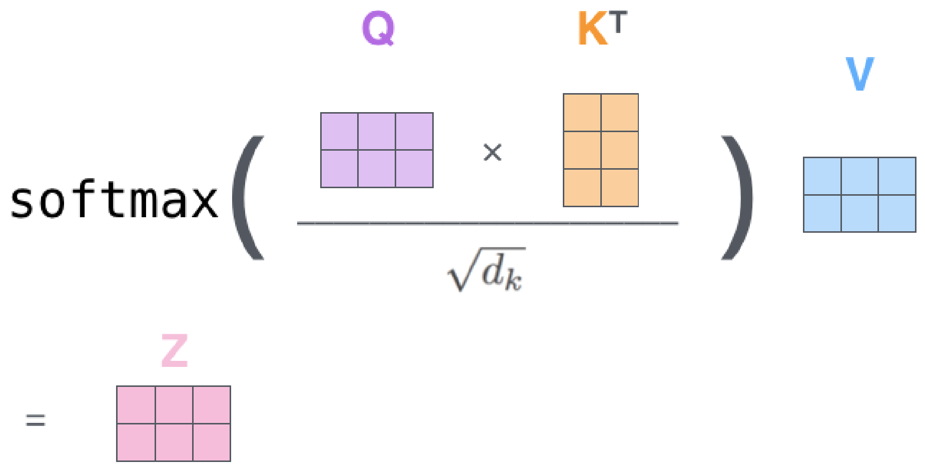


- ✓ 각 Query vector를 모든 Key vector와 내적 후 $\sqrt{d_k}$로 나눔 ➔ attention score
- ✓ 해당 값들을 softmax ➔ attention distribution
- ✓ Attention score와 Value vector의 matrix multiplication ➔ attention value

- https://jalammar.github.io/illustrated-transformer/

## 3.2 Model Architecture - encoder - Self-attention



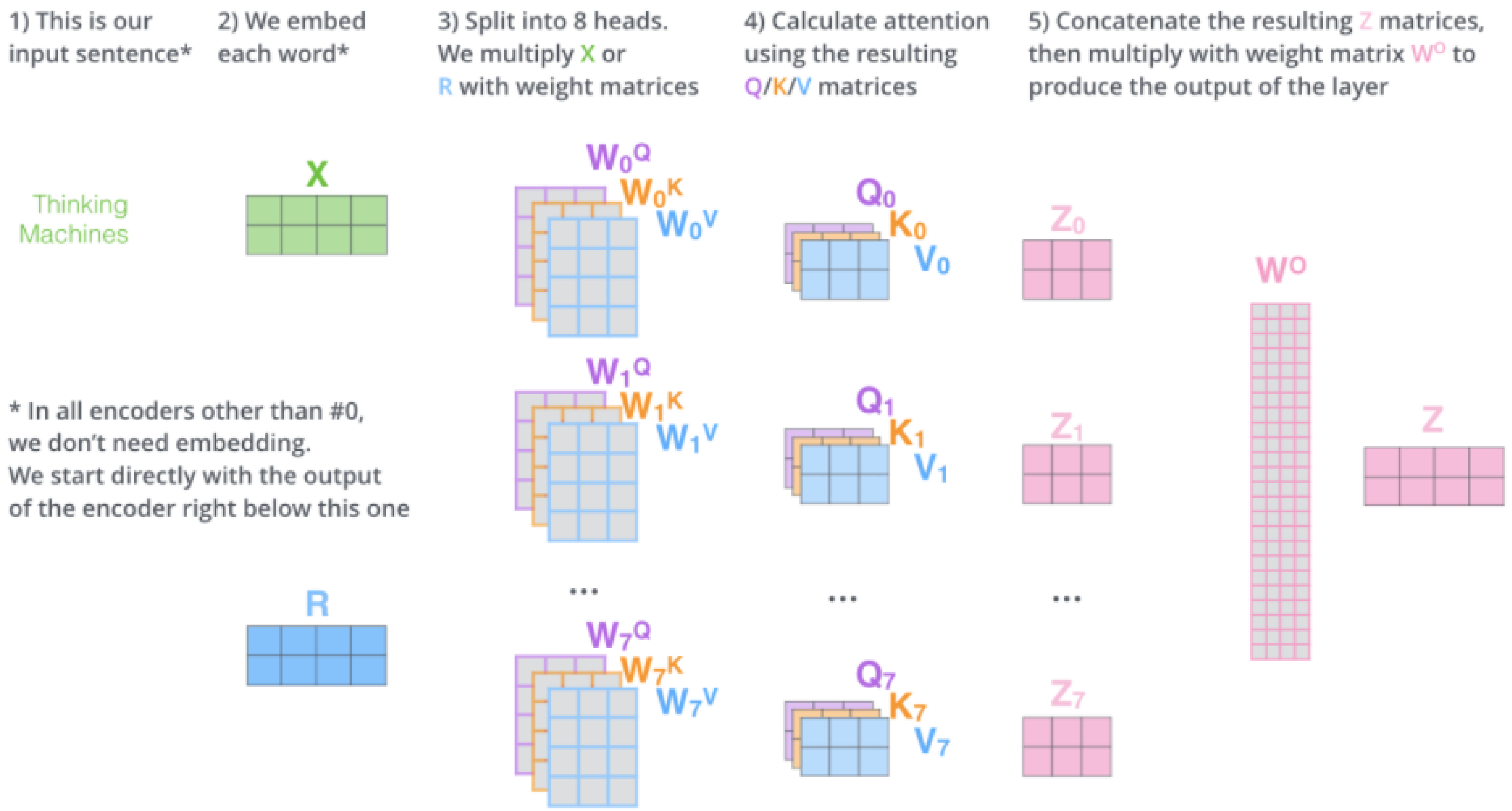


The self-attention calculation in matrix form

✓ 실제 연산은 행렬 연산으로 일괄 처리

- https://wikidocs.net/31379

## 3.2 Model Architecture - encoder - Multi-Head attention = self-attention*head



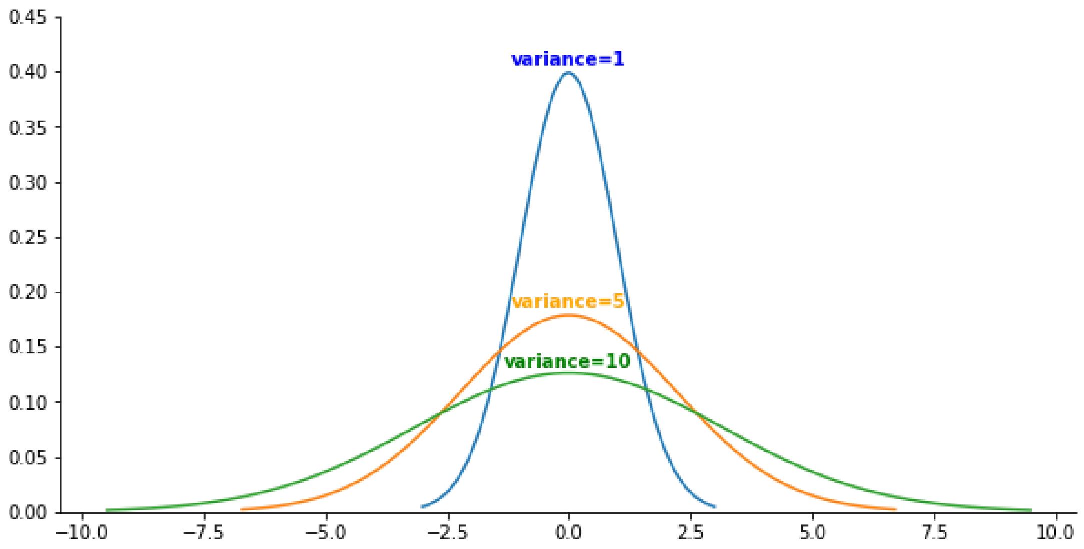
1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply X or R with weight matrices
4) Calculate attention using the resulting Q/K/V matrices
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer
Thinking Machines
X
R
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W0Q W0K W0V
W1Q W1K W1V
W7Q W7K W7V
Q0 K0 V0
Q1 K1 V1
Q7 K7 V7
Z0
Z1
Z7
WO
Z

- ✓ 이전의 구조를 하나의 Head ➔ Head를 여러 개로 구성 ➔ MHA
- ✓ 논문 상에선, embedding dimension을 512로 하고, 8개의 head로 나눠서 사용 ➔ 64 x 8

## 3.2 Model Architecture - encoder - Multi-Head attention

### Q) Why divide by $\sqrt{d_k}$ ?

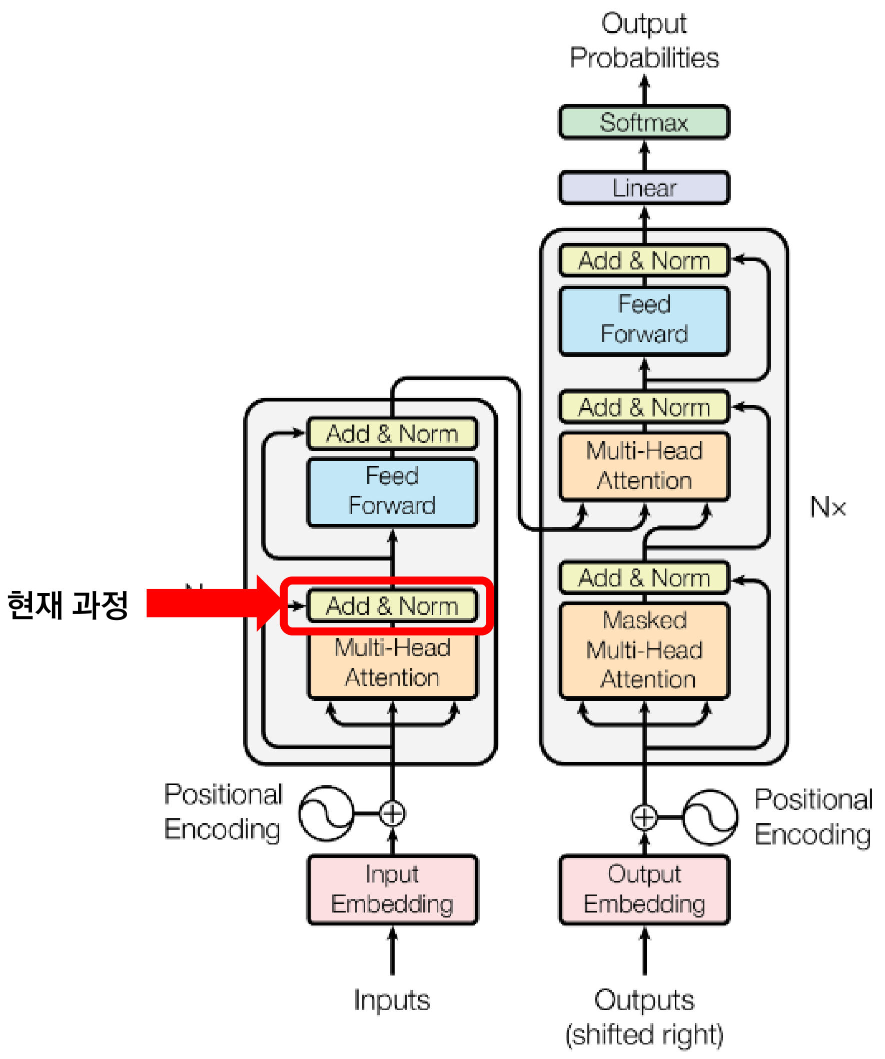
- ✓ Q, K vecto의 값이 평균과 분산이 0,1로 random하게 할당되었을 때
- ✓ Q, K 내적 시 평균과 분산은 각각 0과 dimension
- ✓ Softmax 연산은 지수함수를 사용한 연산으로 값이 클수록 softmax probability가 증가
- ✓ 따라서, devide by $\sqrt{d_k}$로, 평균과 분산을 0,1로 만들기 위함



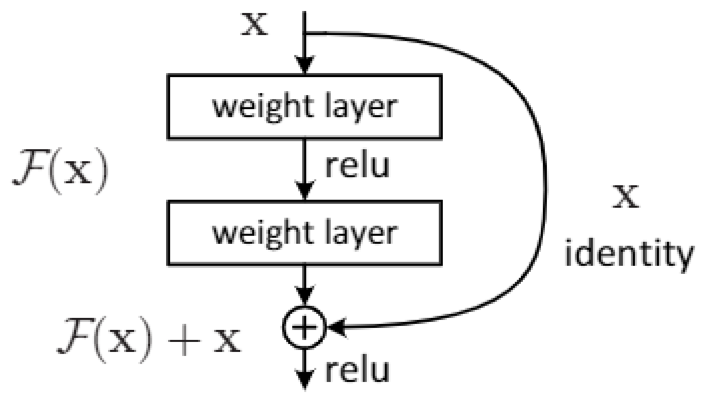
### Q) Q, K, V dimension

- ✓ Q, K vector 내적 ➔ 동일 차원
- ✓ Value vector는 가중치가 곱해지기에 달라도 무방
- ✓ 단, 이후 MHA 이후 residual connection ➔ input과 동일 차원

## 3.3 Model Architecture - encoder - Layer Norm



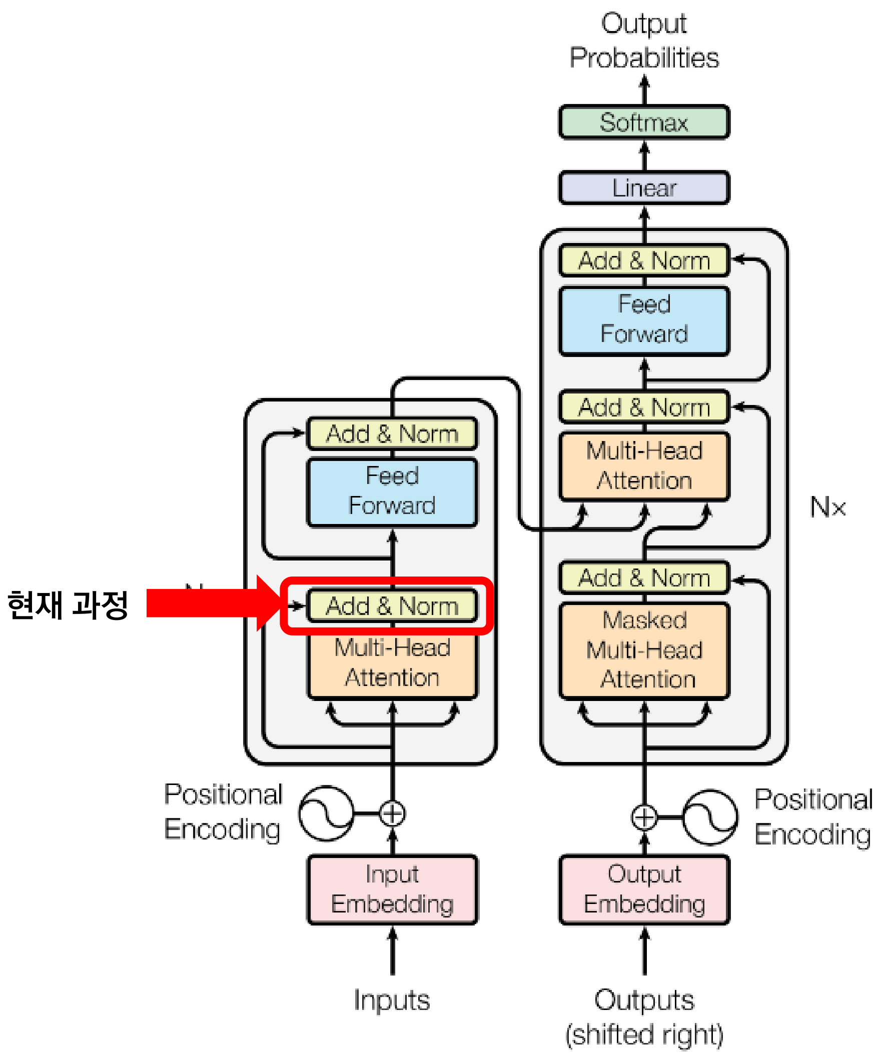
Add & Norm: Residual Connection ➔ Layer Normalization



Deep Residual Learning for Image Recognition, CVPR 2015)

- ✓ F(x) :  기존 정보를 보존하지 않고 변형시켜 새롭게 생성하는 정보
- ✓ x : 기존 정보
- ✓ F(x)+x : 기존 정보를 보존한 상태로, 추가적으로 학습한 정보
- ✓ 장점 : 깊은 신경망을 쌓을 때 발생하는 장기기억손실 문제 해결 (RNNs의 문제인 장기기억손실 문제 해결)

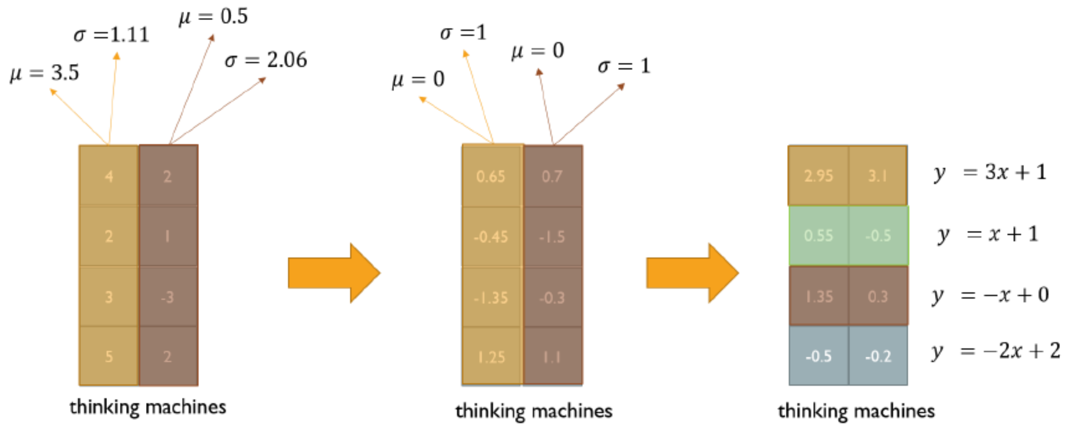
## 3.3 Model Architecture - encoder - Layer Norm



Add & Norm: Residual Connection ➔ Layer Normalization

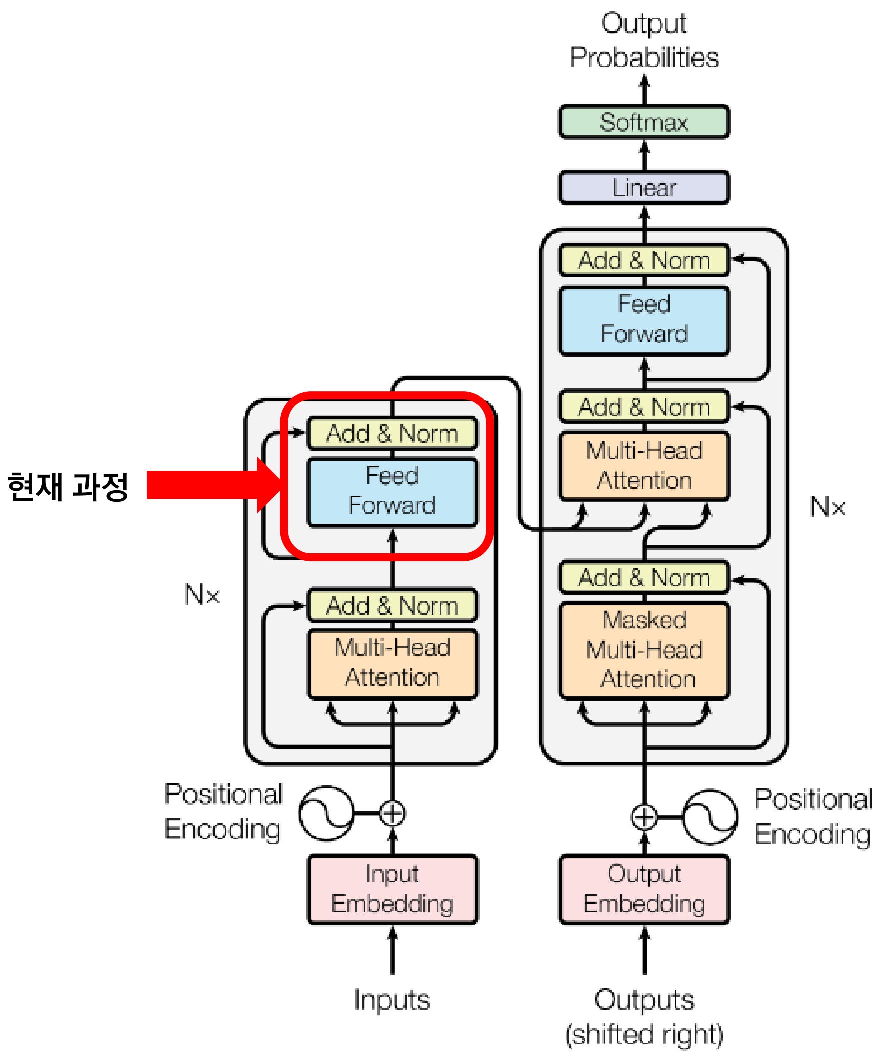
$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l, \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}, \qquad h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$

Group Normalization, ECCV 2018



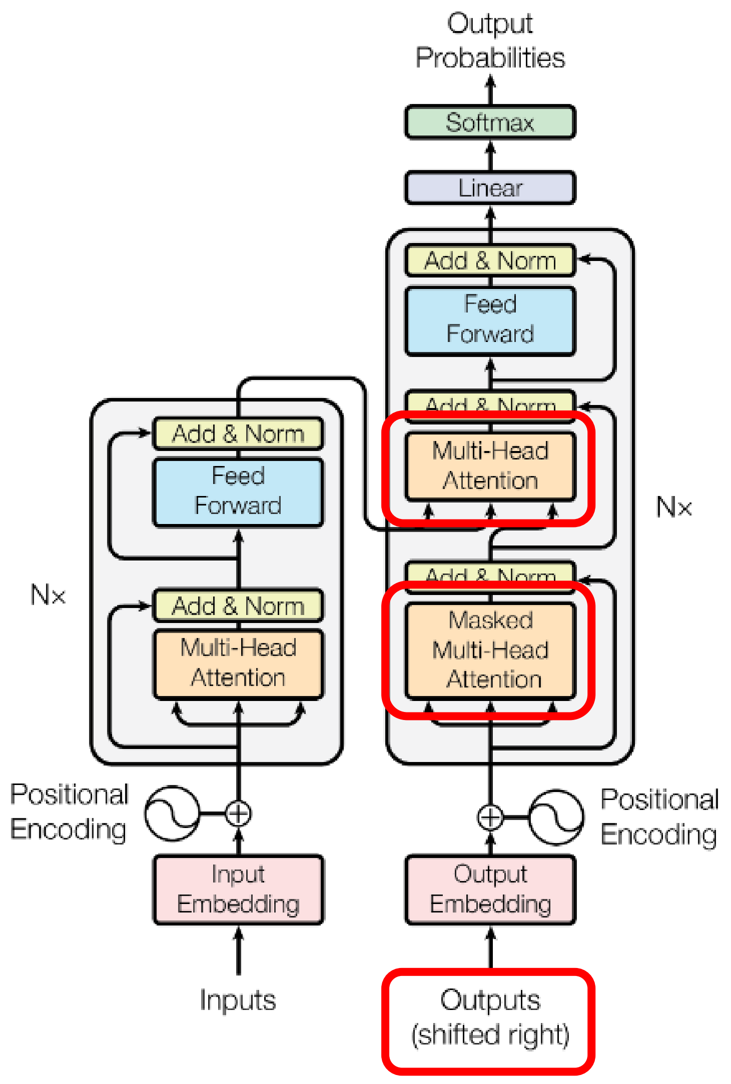
✓ 각 Word의 dimension를 평균과 분산이 0,1로 normalization
✓ Affine transformation 적용(trainable)
➔ y = ax +b에서, 각 성분이 x(평균0, 분산1) 에 들어가며, 평균이 b로 분산이 a로 변환

## 3.4 Model Architecture - encoder - FFNN + Add & Norm



- ✓ Position-wise Fead Forward Networks를 통과 후
  (Dense layer(512,2048) ➔ ReLU ➔ Dense Layer(2048,512)
- ✓ 다시 residual connection + Layer Norm을 통해 encode의 output 반환
- ✓ 단, Transformer는 6개의 encoder를 쌓인 arichtecture

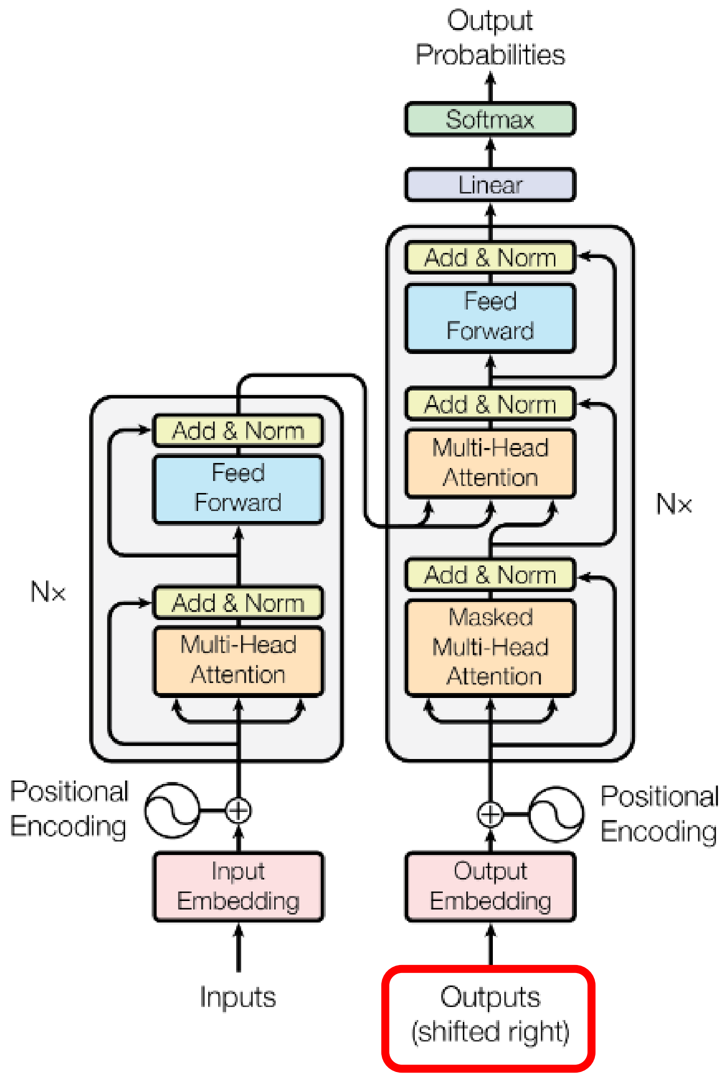
## 3.4 Model Architecture - decoder



✓ Decoder는 encoder와 유사한 구조

✓ 차이점
  ✓ Decoder의 Input
  ✓ 2-layers의 MHA
  ✓ 첫 번째 MHA : Masked MHA

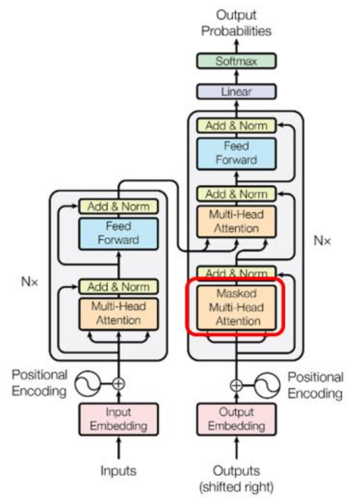
## 3.4 Model Architecture - decoder - Inputs



- ✓ Decoder는 input 예측로 다음 단어를 예측
- ✓ 첫 input은 <SOS> token
- ✓ 이후 input은 두 가지 ➔ decoder의 output(feature forcing) vs ground truth
- ✓ Pos and neg
  - ✓ Decoder의 output
    - ✓ Pos : 실제 생성 과정과 동일 환경에서 학습
    - ✓ Neg : 이전 과정에서 단어를 예측하면 이후 과정은 잘못된 단어로 학습
  - ✓ Ground truth
    - ✓ Pos : 예측 단어와 상관없이 정상적인 값으로 학습
    - ✓ Neg : 실제 생성 과정과 다른 환경
- ✓ 따라서, 초기 학습엔 Ground Truth를 적용, 어느 정도 학습 후엔 teacher forcing 적용

- http://www.clungu.com/Teacher-Forcing/

## 3.4 Model Architecture - decoder – Masked MHA



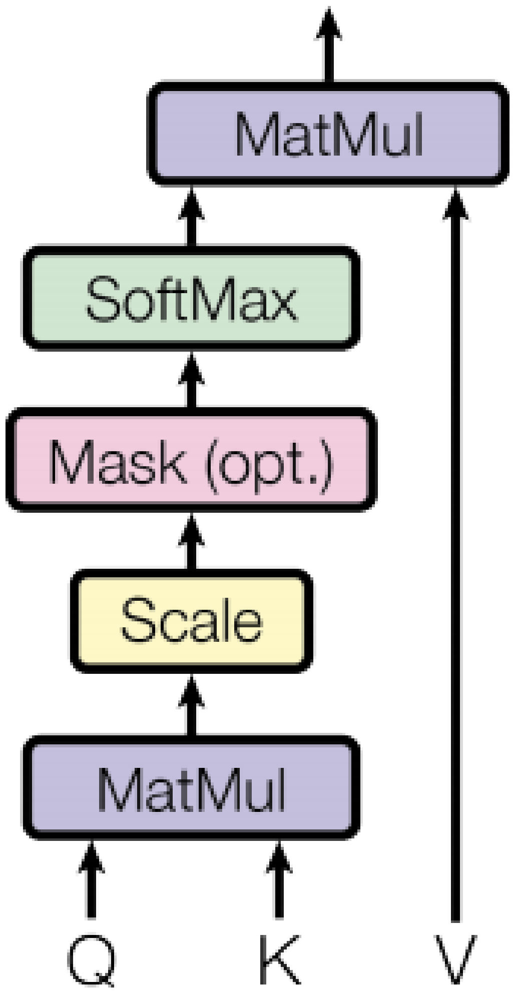


Table 1 Softmax result(without Masking)

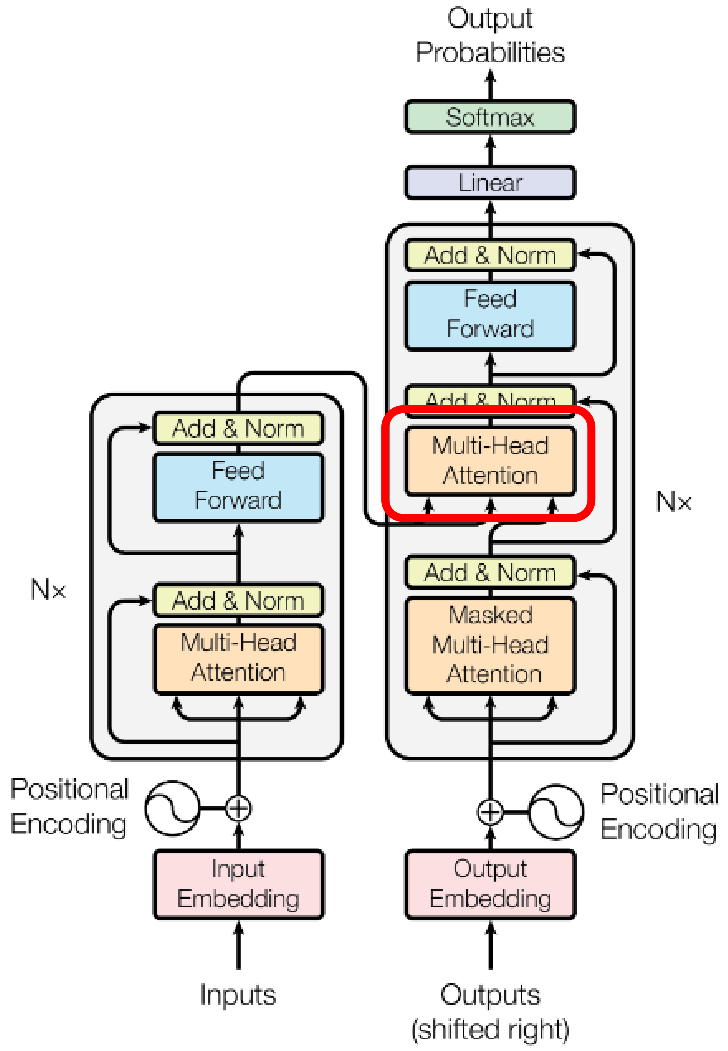
| | I | go | home |
|---|---|---|---|
| I | 0.8 | 0.15 | 0.05 |
| go | 0.3 | 0.6 | 0.1 |
| home | 0.1 | 0.3 | 0.6 |

Table 2 Softmax result(Masking)

| | I | go | home |
|---|---|---|---|
| I | 1.0 | ~~0.15~~ | ~~0.05~~ |
| go | 0.33 | 0.66 | ~~0.1~~ |
| home | 0.1 | 0.3 | 0.6 |

- ✓ 기존 MHA과 동일하게 softmax까지 진행
- ✓ 단, encoder의 MHA(without masking)은 입력 sequence를 모두 참고
- ✓ 단, decoder의 학습 및 실제 생성 과정에서 다음 단어를 참고하여 예측
- ✓ 다음 단어가 현재 단어의 학습 과정에 포함되선 안됨
- ✓ 따라서, softmax 결과에 대각성분의 상위 성분들(다음 단어들에 의한 값)들을 −inf값으로 변환 후 normalization
- ✓ 그렇게 되면 이전 단어들 및 현재 단어만으로 softmax 결과가 출력됨
- ✓ 이후 과정은 다시 MHA와 동일

## 3.4 Model Architecture - decoder - outputs

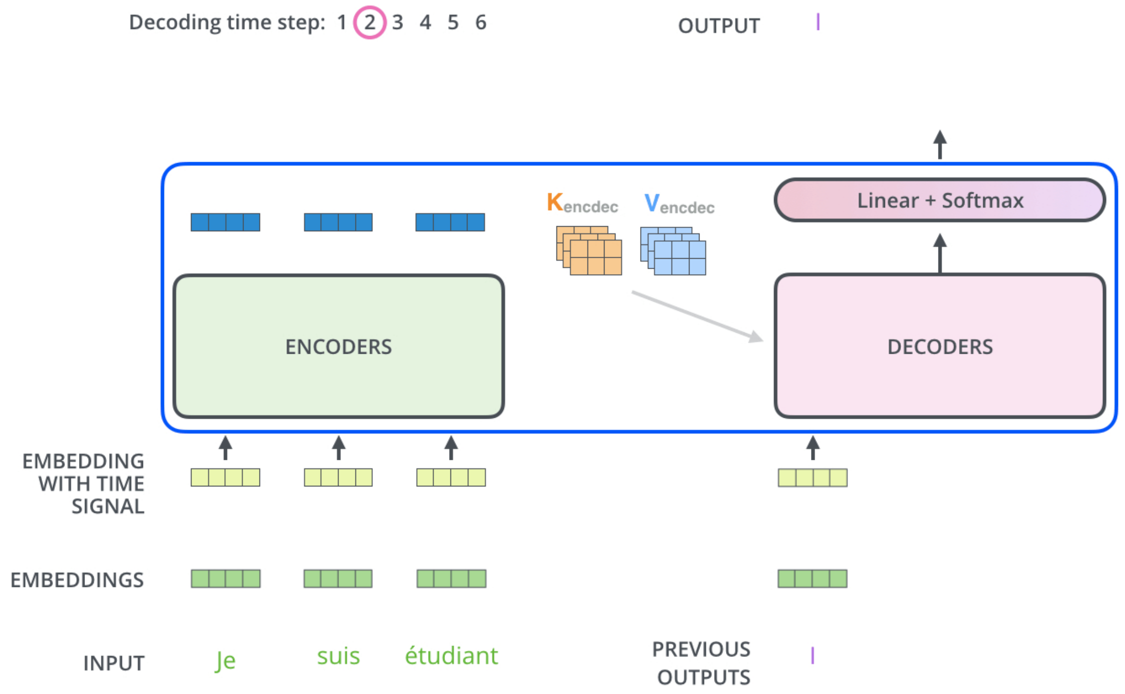


✓ 이후 과정은 동일하게 position-wise FFNN layer를 거치고,

✓ Vocab size만큼 dimension을 늘리기 위한 linear Transformation

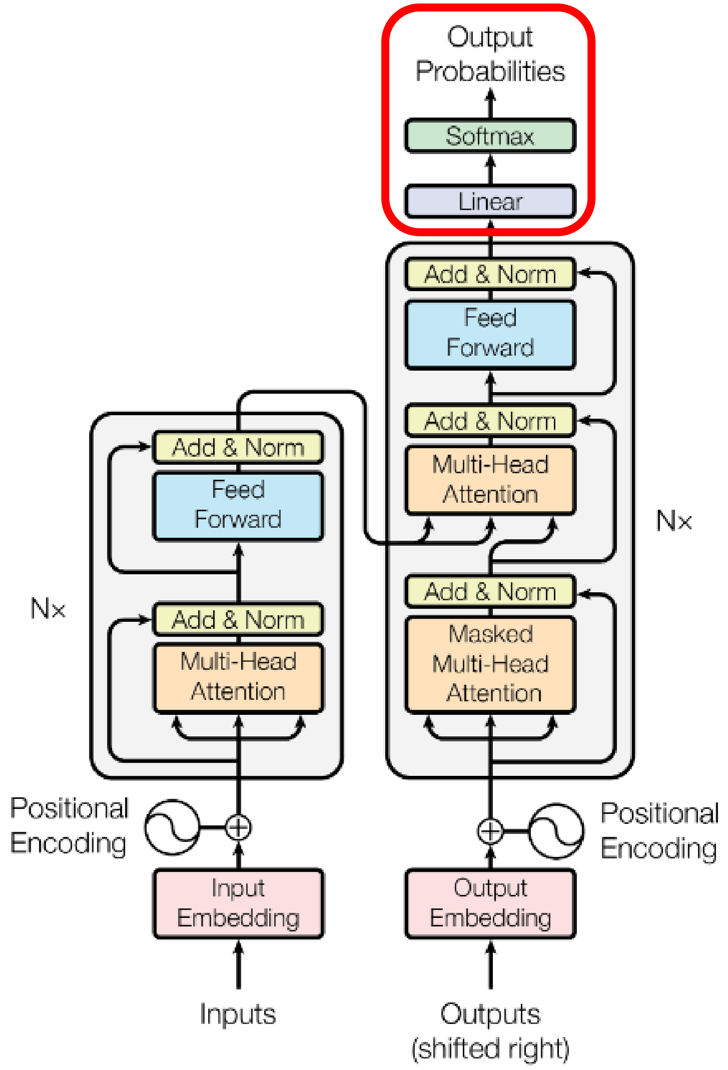
✓ 그리고, 확률변수를 출력하기 위한 softmax

✓ 아래 animation은 Transformer의 전 과정



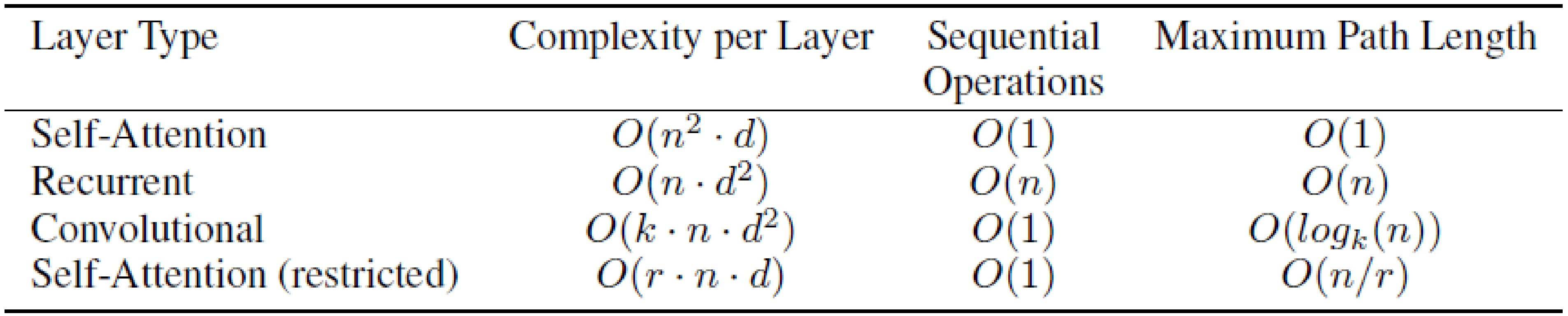
- https://jalammar.github.io/illustrated-transformer/

## 3.4 Model Architecture - decoder - MHA(encoder-decoder attention)



- ✓ encoder−decoder attention layer는 encode의 attention layer와 동일
- ✓ 단, Query vector는 이전 layer의 output이며,
- ✓ Key & value Vector는 encoder의 output이 변형된 key & value Vector를 사용
- ✓ Why?
  - ✓ i 번째 단어를 만들 때, i 번째 단어의 query vector와 모든 key vector 를 내적
  - ✓ 거기에 value vector를 weighted sum
  - ✓ 즉, input에 있는 단어들에 대한 출력의 attention map을 만들려면 input 단어와 연산을 할 key vector와 value vector가 필요

## 4. Complexity



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

n : is the sequence length
d : is the representation dimension
h : is the kernel size of convolutions
r : the size of the neighborhood in restricted self-attention.

Complexity per Layer
- ✓ Self-Attention : $Q * K^T$ ➔ ( n x d ) x ( d x n) ➔ $n^2 \cdot d$
- ✓ Recurrent : hidden vector를 구하기 위해 이전 hidden vector dimension과 행과 열의 길이가 동일한 weight matric와 연산
  ➔ d x d, 그리고 이 과정을 sequence 길이만큼 수행하므로 ➔ $d^2 \cdot n$

Sequential Operations
- ✓ Self-Attention : GPU가 무한하다고 가정할 때, 한번에 모든 연산을 진행
- ✓ Recurrent : time-step마다 수행해야 하므로 n

Maximun Path Length
- ✓ Self-Attention : query vector와 direct로 연결
- ✓ Recurrent : Time-step 진행되었으므로, 첫번째 input과 최종 output 사이에는 seq len만큼 path가 길어짐

Convolutional은 RNN과 똑같이 sequence 길이만큼 수행되는데, 단 크기가 k인 1-d conv이 kerne이 이동하며 연산을 수행
Self-Attention(restricted)는 주변 단어의 길이를 전체 n에서 r로 변경

## 5. Optimizer, Scheduler, etc..

### Optmizer : Adam

- ✓ $\beta_1$ : 0.9 (momentum)
- ✓ $\beta_2$ : 0.98 (EMA of gradient Squares)
- ✓ $\epsilon : 10^{-9}$

Momentum

EMA of gradient squares

$$\boxed{m_t} = \beta_1 m_{t=1} + (1 - \beta_1) g_t$$

$$\boxed{v_t} = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

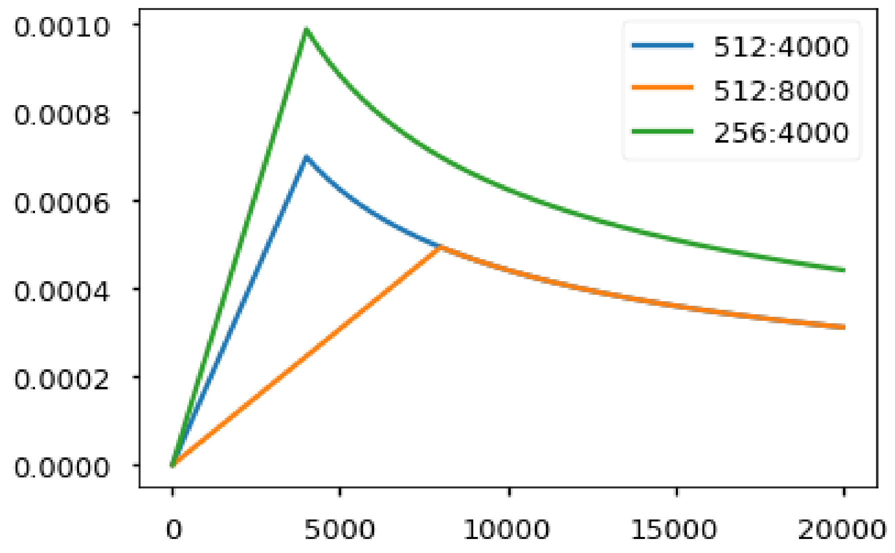
Stepsize

$$W_{t+1} = W_t - \frac{\boxed{\eta}}{\sqrt{\boxed{v_t}} + \epsilon} \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \boxed{m_t}$$

### Scheduler : Warm-up Learning Rate Scheduler

- ✓ warmup_steps = 4000
- ✓ 초기엔 learning rate 지속적으로 증가
- ✓ 이후 어느 정도 step을 거치면 지속적으로 감소

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

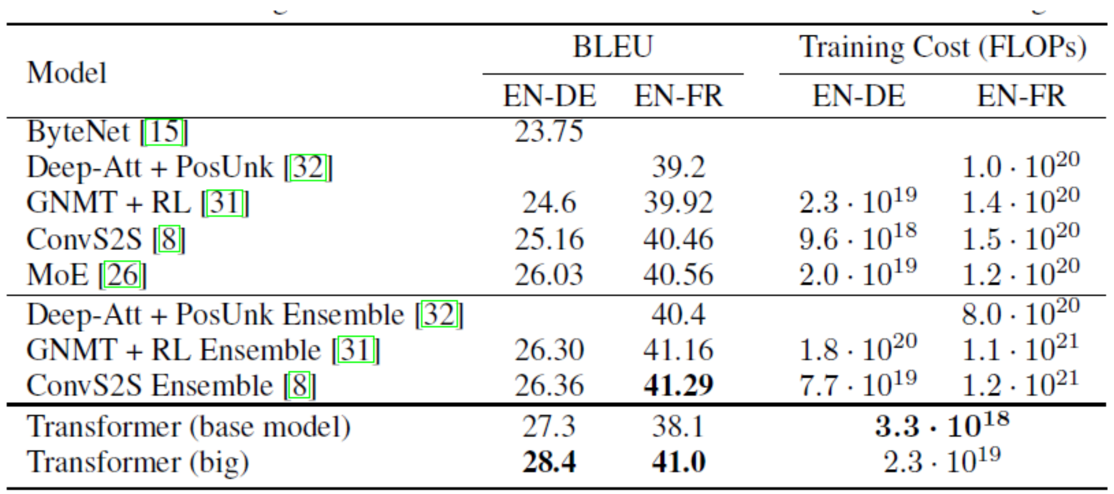


### etc

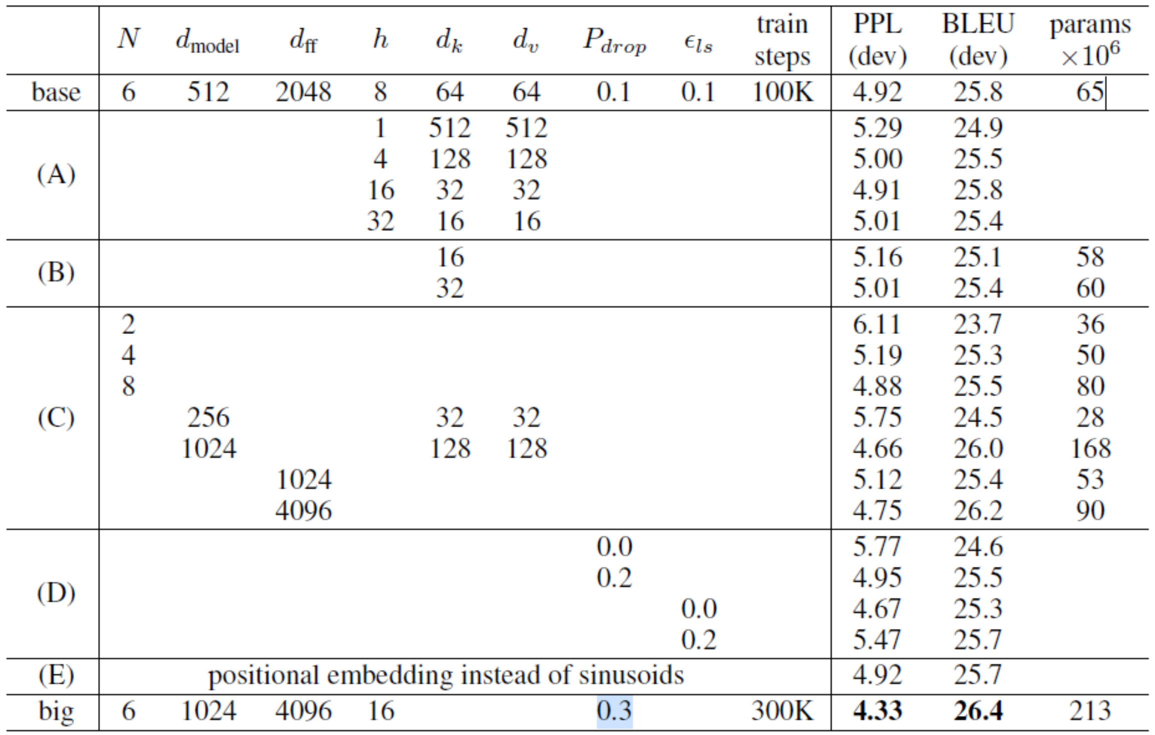
- ✓ Label smoothing
- ✓ Residual dropout

• http://nlp.seas.harvard.edu/2018/04/03/attention

## 6. Score



| 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 | $\mathbf{3.3 \cdot 10^{18}}$ |  |
| Transformer (big) | **28.4** | **41.0** | $2.3 \cdot 10^{19}$ |  |



|  | $N$ | $d_{\text{model}}$ | $d_{\text{ff}}$ | $h$ | $d_k$ | $d_v$ | $P_{drop}$ | $\epsilon_{ls}$ | 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 |

Transformer hyperparameters and scores

## 7. Code

harvardnlp
- ✓ http://nlp.seas.harvard.edu/2018/04/03/attention
- ✓ Ipynb
- ✓ Pytorch
- ✓ 설명과 함께 코드를 보려면 이 자료를 추천

다수의 star와 clone이 된 github
- ✓ https://github.com/jadore801120/attention-is-all-you-need-pytorch
- ✓ Pytorch로 구현
- ✓ 패키지화
- ✓ 논문 발표 후 이틀 뒤부터 구현하기 시작한 거 같네요
- ✓ 아마도 처음으로 구현한 사람이지 않을까…
- ✓ 패키지로 구성된 결과를 보려면 이 자료를 추천

# 끝으로... Transformer 및 참고하여 공부하면 좋을 자료들..

- Trasformer
  ➔https://greeksharifa.github.io/nlp(natural%20language%20processing)%20/%20rnns/2019/08/17/Attention-Is-All-You-Need/#33-position-wise-feed-forward-networks

- Bert와 Transformer 차이 비교하며 설명
  ➔https://docs.likejazz.com/bert/#masked-attention

- Attentions
  ➔https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

- 딥러닝을 이용한 자연어처리 입문(저자 : 유원준)
  https://wikidocs.net/31379

- 모든 Transformer 강의(다른 교육 단체도 포함)에서 강의자료에서 필수적으로 참고하는 글
  https://jalammar.github.io/illustrated-transformer/