Attention Is All You Need

Original slide author: Daiki Tanaka
Translator(Eng to Kor): huffon

Original Source
: https://www.slideshare.net/DaikiTanaka7/attention-is-all-you-need-127742932

Thanks to Tanaka, again :)

Hoon Heo , Undergraduate student at Hankuk University of Foreign Studies

@DaikiTanaka7 Of course I will do so :) Thanks for sharing slide again !

less than a second ago

DaikiTanaka7

Hi Heo. Yes, It's okay. And if you don't care, please clarify my original slides here in your slides.

3 minutes ago

Hoon Heo , Undergraduate student at Hankuk University of Foreign Studies

Hi, Tanaka! I'm undergraduate student studying NLP and Deep Learning. Can I translate your slide to Korean and share it to korean guys? This is fabulous slide :)

16 minutes ago

Attention is All you need

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.

Background : 이전 모델들은 **" 병렬 처리 "** 에 취약 !

- ***Past*** : RNN, LSTM, GRU는 Sequence modeling에서 항상 SOTA 성능
- RNN은 time step t 이전의 hidden state들과
  timestep t의 input을 조합해 **새로운 hidden state** 생성
- 이러한 Sequential한 성질은 Sequence의 길이가 길 때,
  Training sample들을 **병렬적으로 처리하지 못하게 하는** 장애 요소로 작용
- ***Recent*** : Attention 메커니즘이 최근 자연어 처리 모델들에 있어 핵심적 요소
- 그러나, Attention 조차 **여전히 Recurrent network와 함께 사용**되고 있음

Background : 연산 처리에 대한 문제

- 이처럼 병렬 처리를 저해하는 Sequential 연산을 줄이고자 하는 목표는 **CNN**을 이용해서, 혹은 **hidden state를 일정 부분 병렬적으로 연산**함으로써 소정 달성해옴

- 그러나, 이러한 모델들에서 역시 Input position과 Output position의 Dependency를 찾는 연산의 수는 distance에 비례해 크게 증가

"즉, 거리가 먼 위치에 있는 단어들 간 학습이 어렵다 ! "

What's next?

- Attention 메커니즘을 사용하면, Input과 Output 간 **Global dependency**를 "*상수*" 만큼의 연산을 통해 얻을 수 있음

- 이번 연구에서는 RNN, CNN 등을 **전혀 사용하지 않은** Transformer 모델을 제시할 것이며, 해당 모델은 번역 품질 태스크에서 SOTA를 달성하였음

Problem Setting : Sequence to sequence

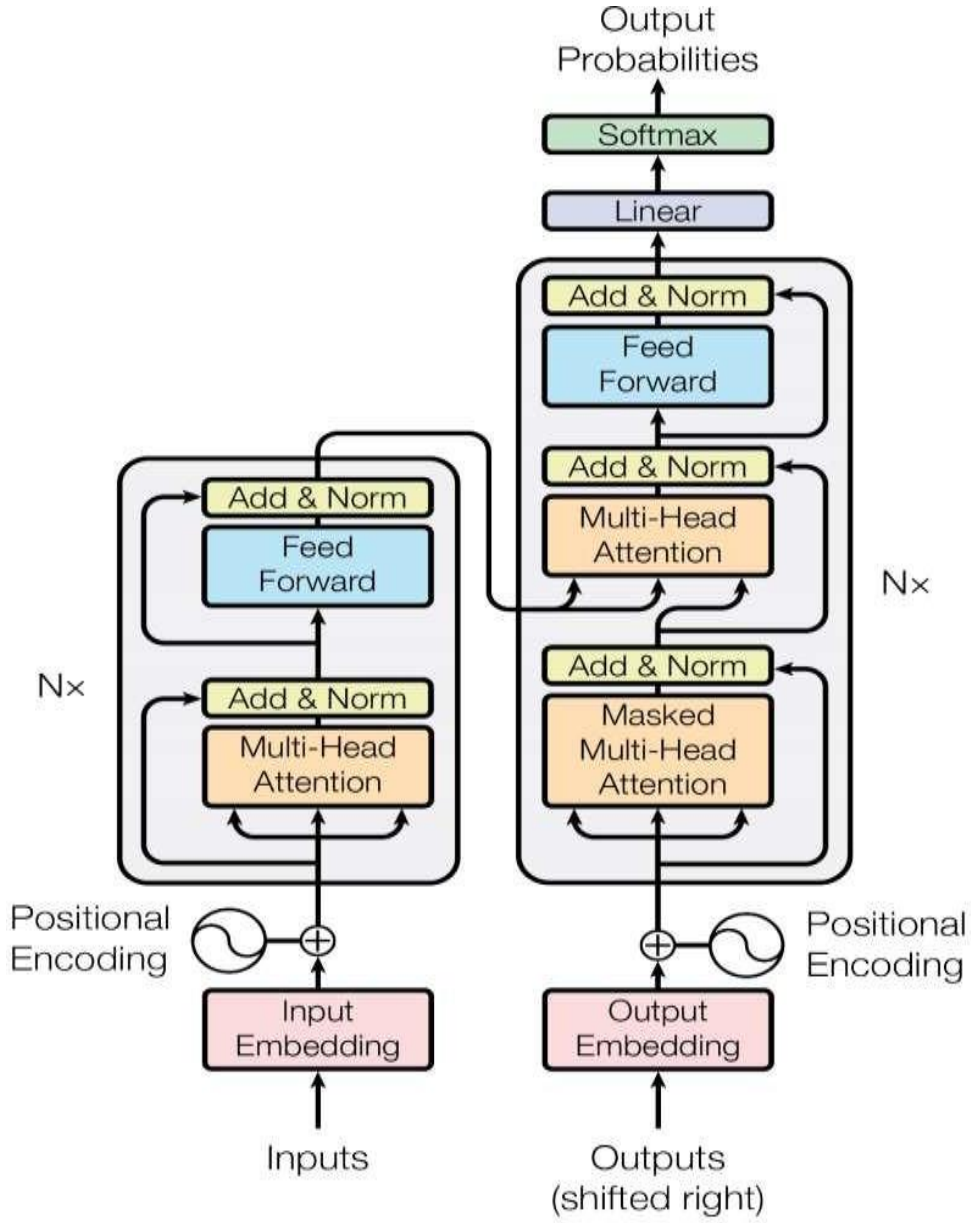
- Input data:
  - Sequence of symbol representations ($x_1, x_2, .., x_n$)
    - “This is a pen.”

- Output data:
  - Sequence of symbol representations ($y_1, y_2, .., y_n$)
    - 「이것은 펜이다. 」

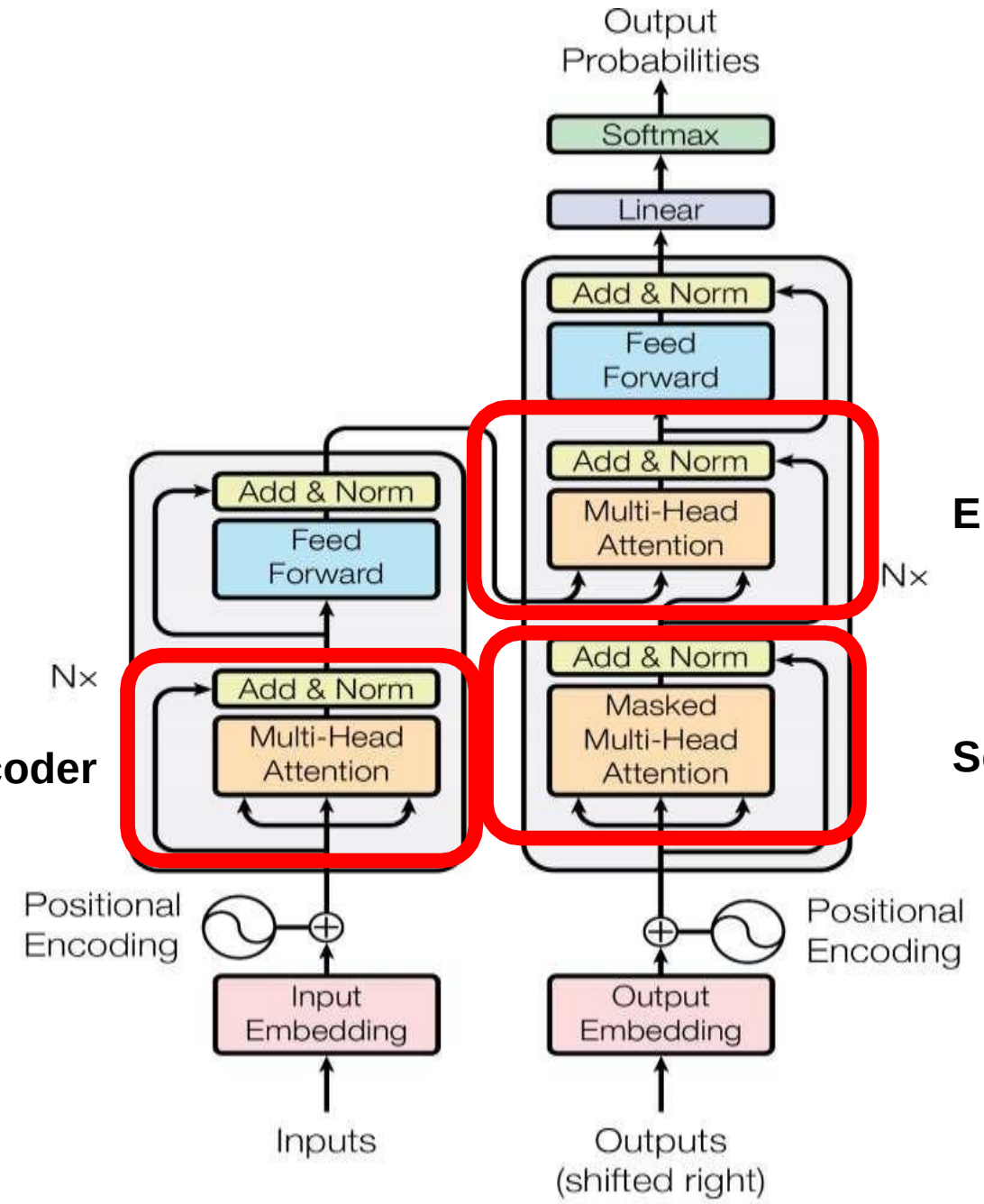
Entire Model Architecture

- 좌측 Blocks: Encoder
- 우측 Blocks: Decoder

- Consisting layers:
  - **Multi-Head Attention layer**
  - **Position-wise Feed-Forward layer**
  - **Positional Encoding**
  - (Residual Adding and Normalization layer)



1. Attentions



Encoder-Decoder attention

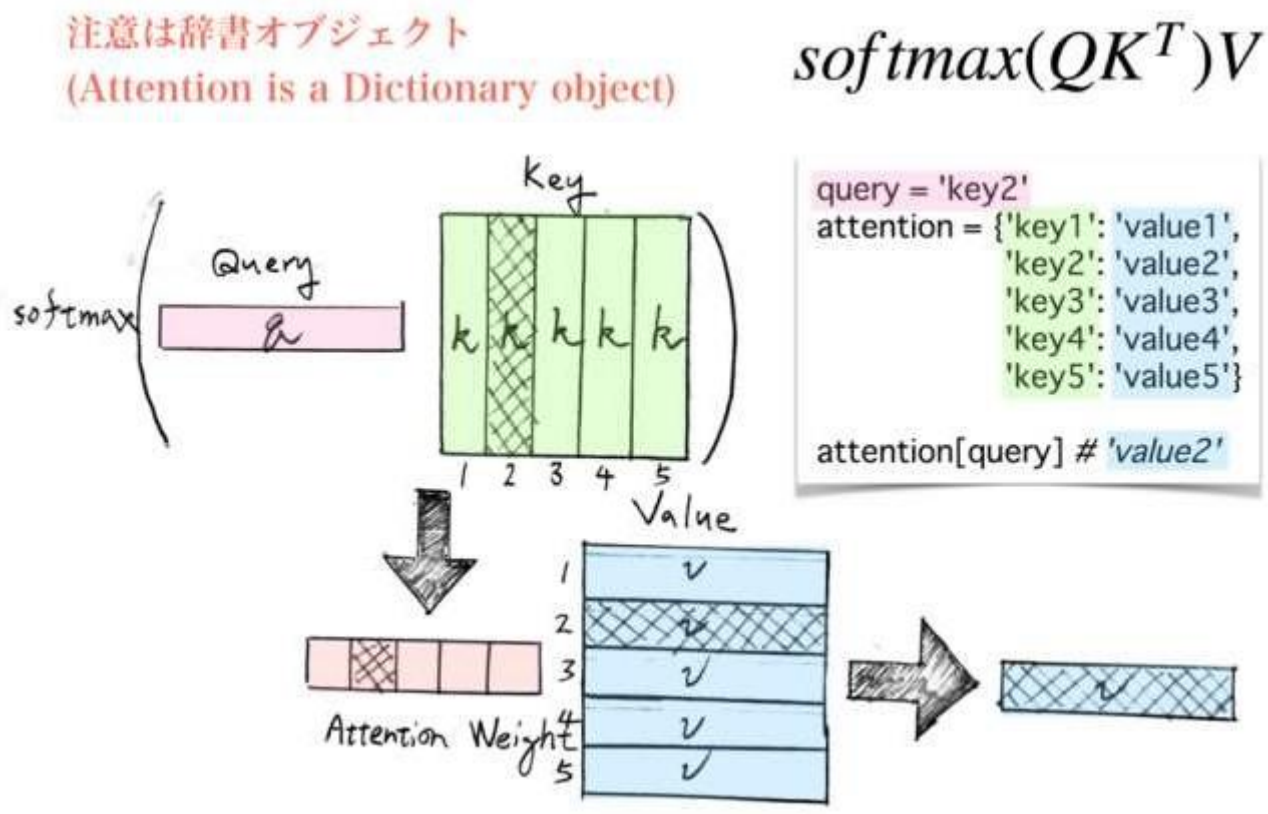
Self-attention in Encoder

Self-attention in Decoder

What is Attention?: 문장 내 단어들 간 관계 정도를 계산

- Attention : (단어 vector q, 문장 matrix K, 문장 matrix V) → context vector v

$$Attention = (query, Key, Value) = Softmax\left(query \cdot Key^{T}\right) \cdot Value$$



What is Attention?

Attention Weight

- $Attention(query, Key, Value) = Softmax(query \cdot Key^T) \cdot Value$

query 벡터와

유사한 벡터 key를 탐색,

그에 상응하는 value 반환

```
query = 'key2'
attention = {'key1': 'value1',
             'key2': 'value2',
             'key3': 'value3',
             'key4': 'value4',
             'key5': 'value5'}

attention[query] # 'value2'
```

즉, Attention function은 **dictionary 자료형**

(Key : Value)과 같은 개념으로 생각할 수 있음 !

What is Attention?: 단어가 아닌 문장 전체를 입력 값으로 넣을 수 있음

- Single query Attention : (vector, matrix, matrix) → vector

$$Attention\ (\ query, Key, Value\ ) = Softmax\ (\ query \cdot Key^T\ ) \cdot Value$$

- Matrix 이용해 Single-query(단어) 아닌 **Multiple-query(문장)** 수행:

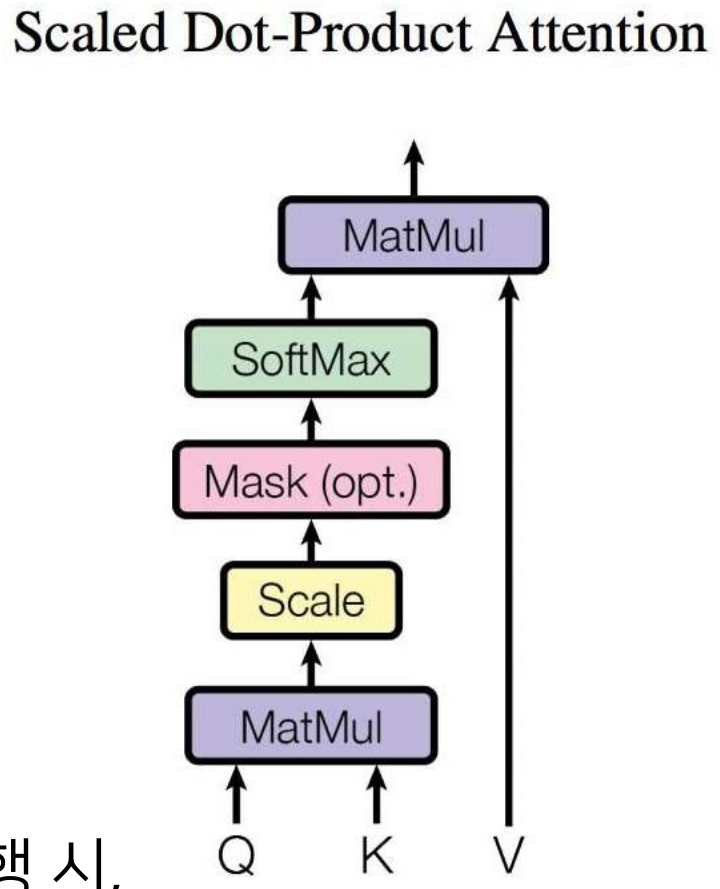
(matrix, matrix, matrix) → matrix

$$Attention\ (\ Query, Key, Value\ ) = Softmax\ (\ Query \cdot Key^T\ ) \cdot Value$$

Scaled Dot-Product Attention

- Attention에는 널리 사용되는 두 가지 기법이 있음
  - Additive attention : $Softmax(\sigma(W[Q,K]+b))$
  - Dot-product attention : $Softmax\ [\ Q\,K^T]$
- 둘 중, **Dot-product attention**이 훨씬 빠르고, 메모리면에서도 효율적
- **"Scaled" Dot-product Attention**:

$$Attention(Q,K,V) = Softmax\left(\frac{Q\,K}{\sqrt{d_k}}\right)V$$



$\sqrt{d_k}$ 가 scaling factor로 사용되며, 이는 softmax function 수행 시,
특정 값이 극도로 작은 경사에 빠지지 않도록 막아주는 **일종의 정규화** 기능 수행

Multi-Head Attention

- 한 번의 Dot-product attention이 아니라

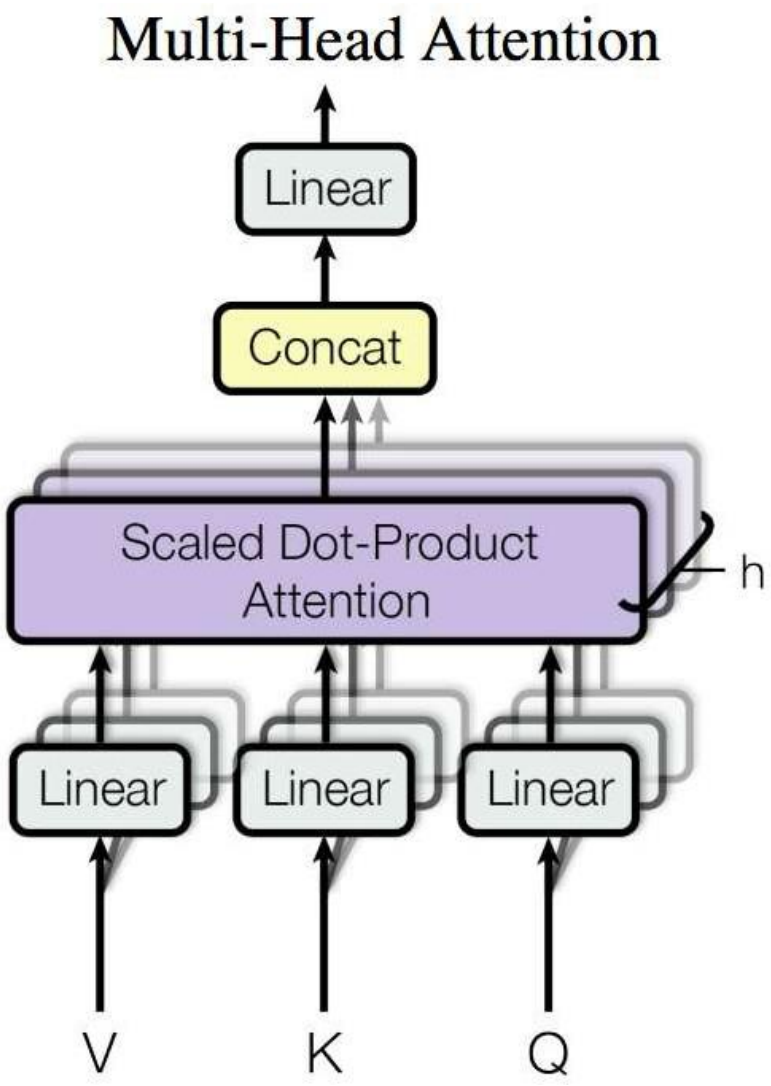
  여러 번의 attention 연산을 수행하자 ! (for example, h = 8)

  : 더 다양한 Input representation으로부터 head 학습 가능하도록!

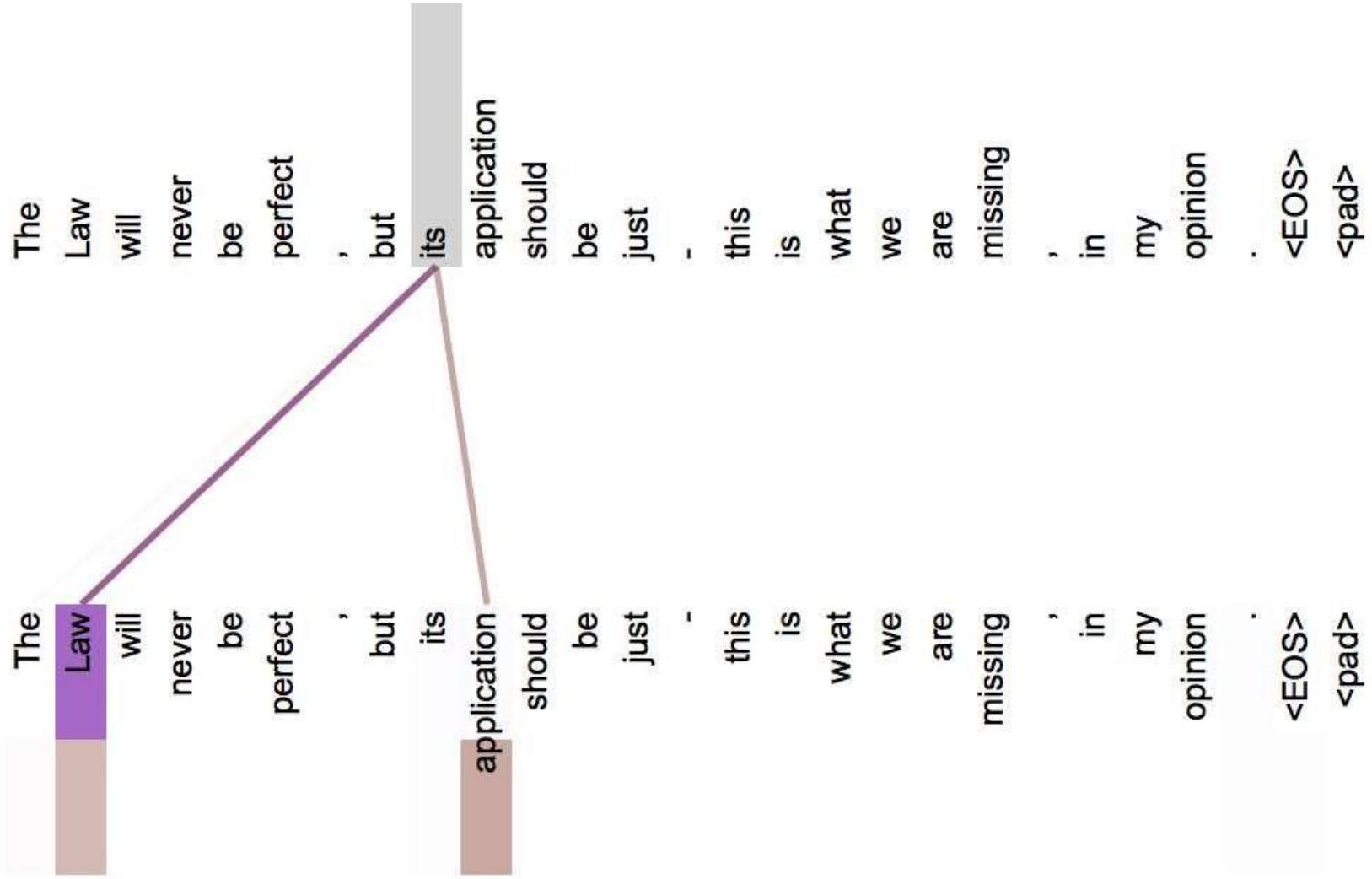
- Q, K, V의 Embedding vector를 나누어 h번만큼 Linear Layer에 투과시킨 후, Attention 결과들을 concat!

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

$$\text{where } \mathrm{head}_\mathrm{i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Experiment – Self-attention visualization



2개의 multi-head가 각기 다른 attention을 취하고 있음 !

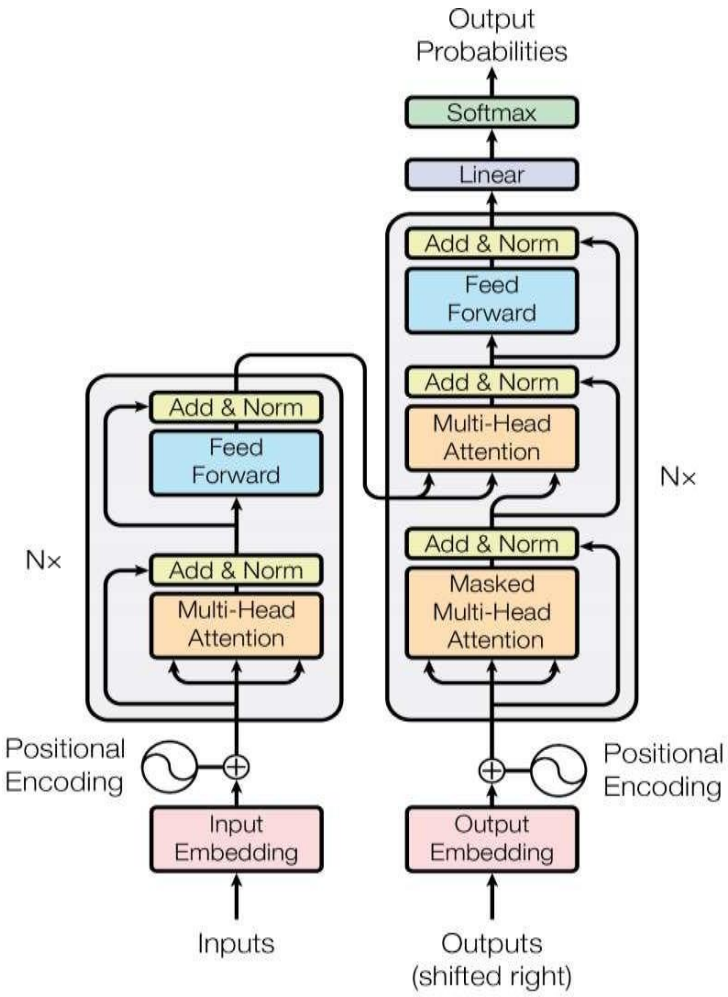
1) What is it of its? Law
2) its what? application

Applications of Multi-Head Attention in model

1. Encoder-decoder attention : Queries(Q)는 **이전 decoder layer**로부터 받아 사용하며, Keys(K)과 Values(V)는 전통적인 Attention 메커니즘과 마찬가지로 **encoder의 output**을 사용

2. Self-attention layers in encoder : Keys(K), Values(V) 그리고 Queries(Q)는 모두 같은 Matrix를 사용. 즉, **이전 encoder layer가 계산한 output matrix**를 **Q=K=V**로 사용

3. Self-attention layers in decoder : 마찬가지로 K, V, Q 모두는 **이전 decoder layer가 계산한 output matrix**를 사용.
   이 때, 전체 문장을 참조하지 않고, 자신보다 이전에 등장한 단어만 참조하도록 하는 "**마스킹 기법**"을 추가적으로 고려해야 함



Self-Attention in Decoder

- Decoder에서는, self-attention layer가 **output sequence의 이전에 위치한 단어들만 참조**할 수 있어야 함. 그리고 이는 현재 **output sequence 이후에 위치한 단어들을 Masking**함으로써 해결할 수 있음. 이를 위해, softmax를 적용하기 전 현재 output sequence 이후 단어들에 **–inf** 값을 주어 softmax 적용 후 값이 0에 가깝게 만들어 줌

- **ex)** 만약 "I play soccer in the morning" 이라는 문장이 Decoder에 주어졌고, "soccer"에 self-attention을 적용하고자 한다면 ("soccer"가 query)
  soccer의 self-attention을 구하기 위해서는 **이미 등장한 "I", "play"**만 사용 가능
- **"in", "the", "morning"은 아직 등장하지 않은 단어**이므로 attention 불가능 !

Why Self-Attention?

1. **Layer 당 처리해야 하는 연산량**의 감소
2. Matrix 연산이 주를 이루기 때문에 **병렬 처리 가능**한 연산의 증가
3. Target 단어와 상응하는 Source 단어를 찾는 **Maximum path length 감소**
4. +) 부가적으로 Self-attention은 전체 모델 아키텍쳐를 이해하기 쉽게함!

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

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

Data Flow in Attention (Multi-Head)

1) This is our input sentence*

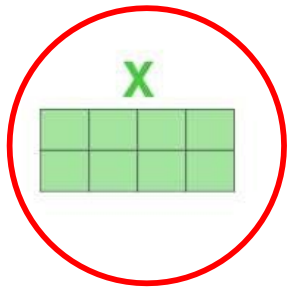
2) We embed each word*

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

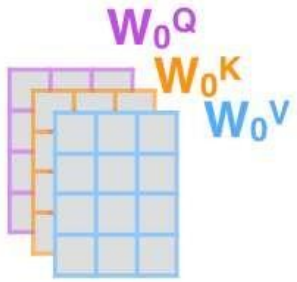
4) Calculate attention using the resulting Q/K/V matrices

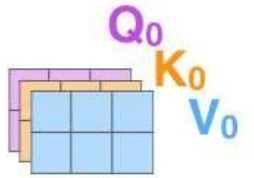
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

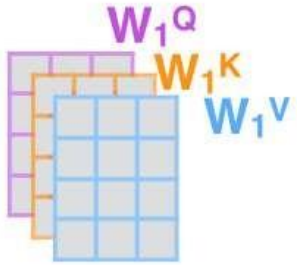
Thinking Machines

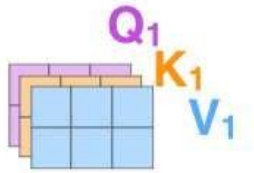


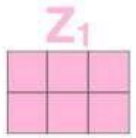
Input







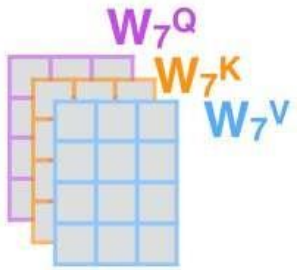


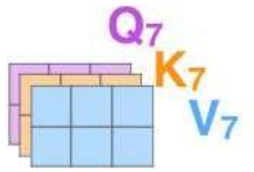


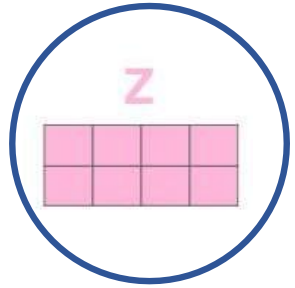
...

...

...

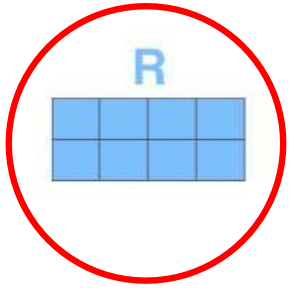




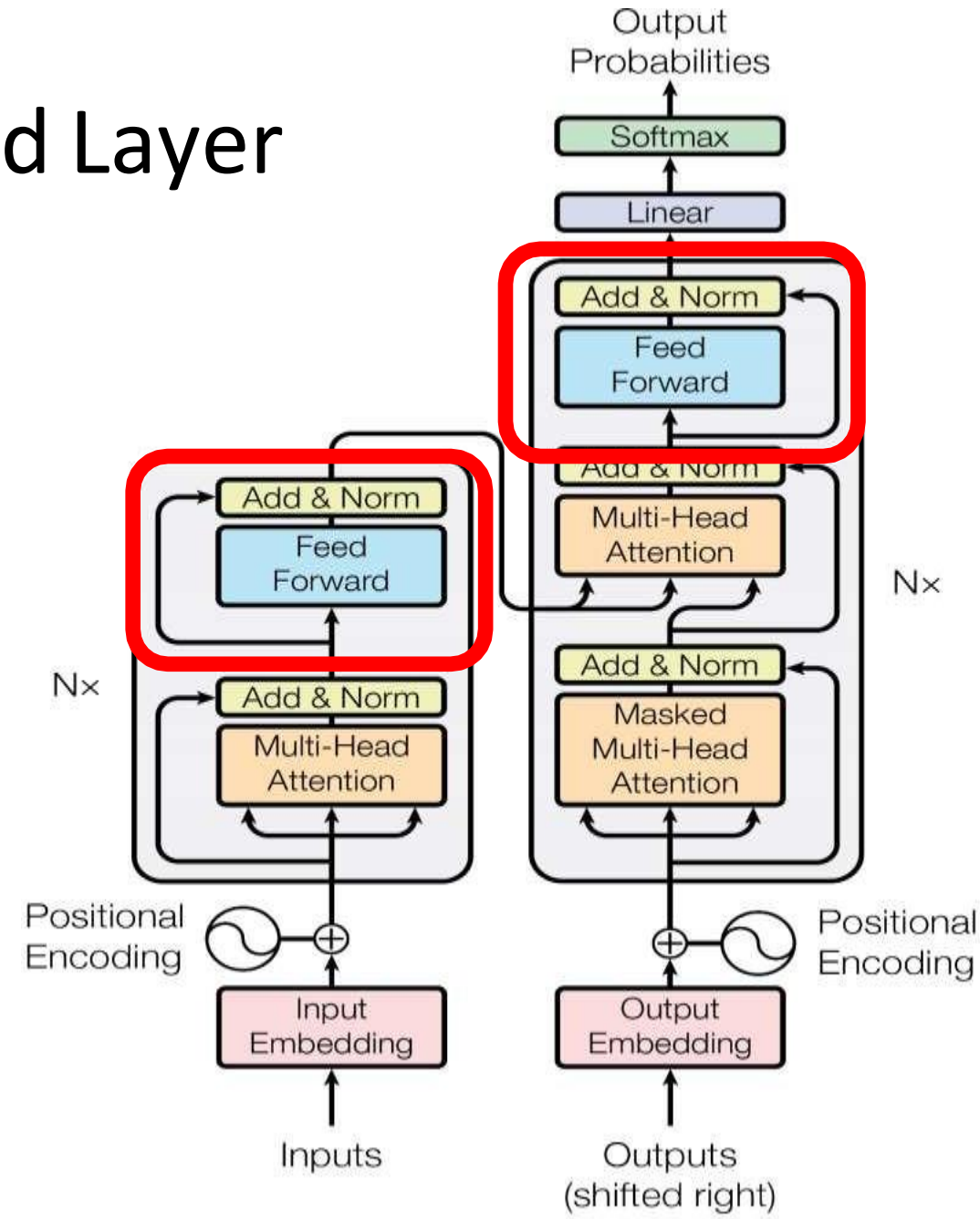


Output

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



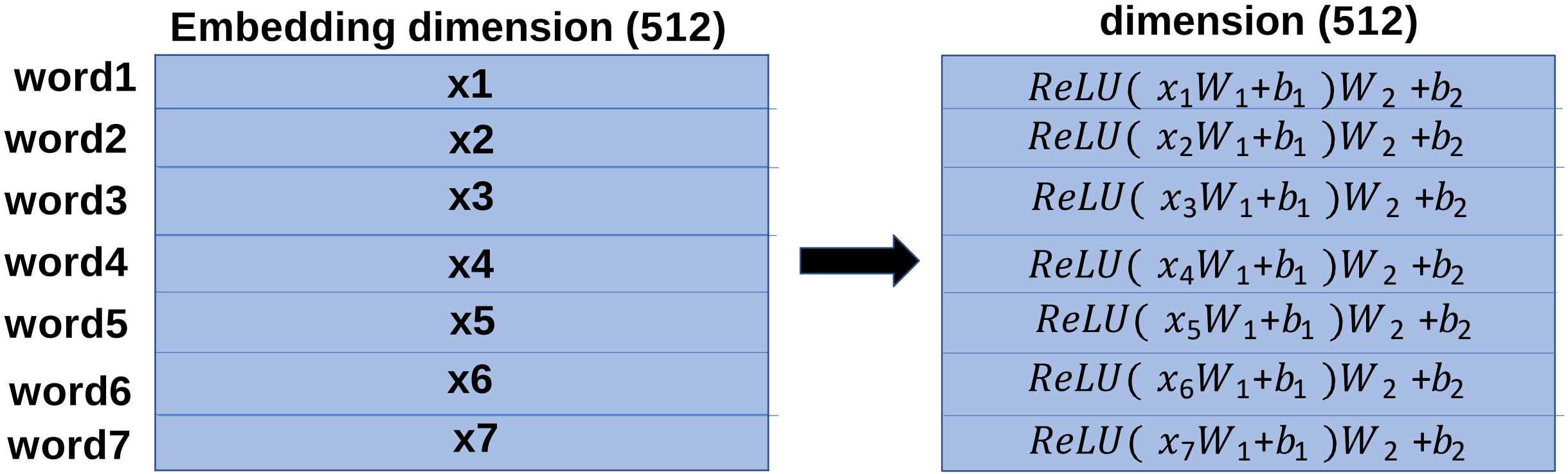
2. Feed Forward Layer



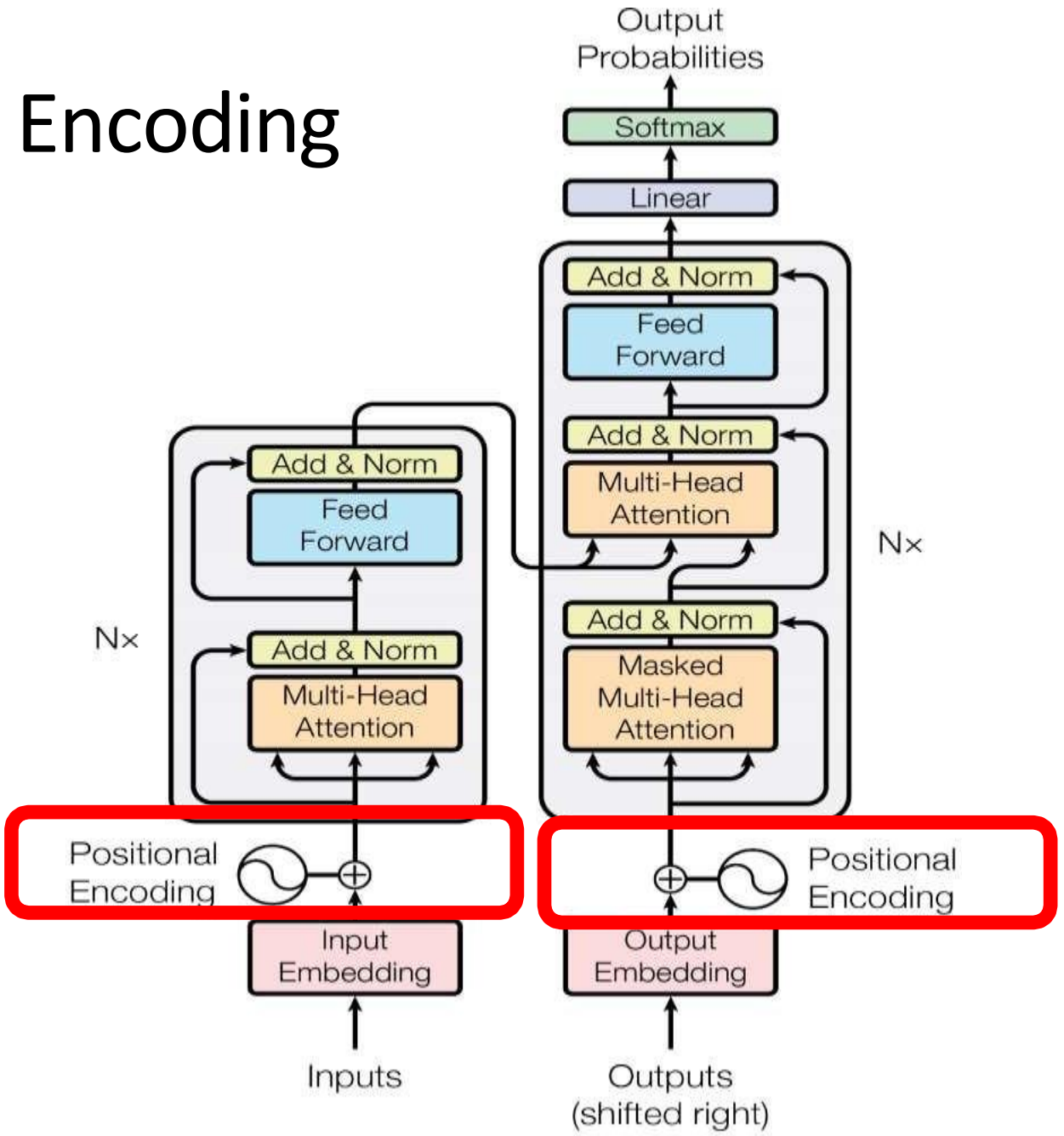
Position-wise Feed-Forward Networks

- Feed-forward networks를 각 단어들에 순차적으로 적용!

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2$$



3. Positional Encoding



Positional Encoding

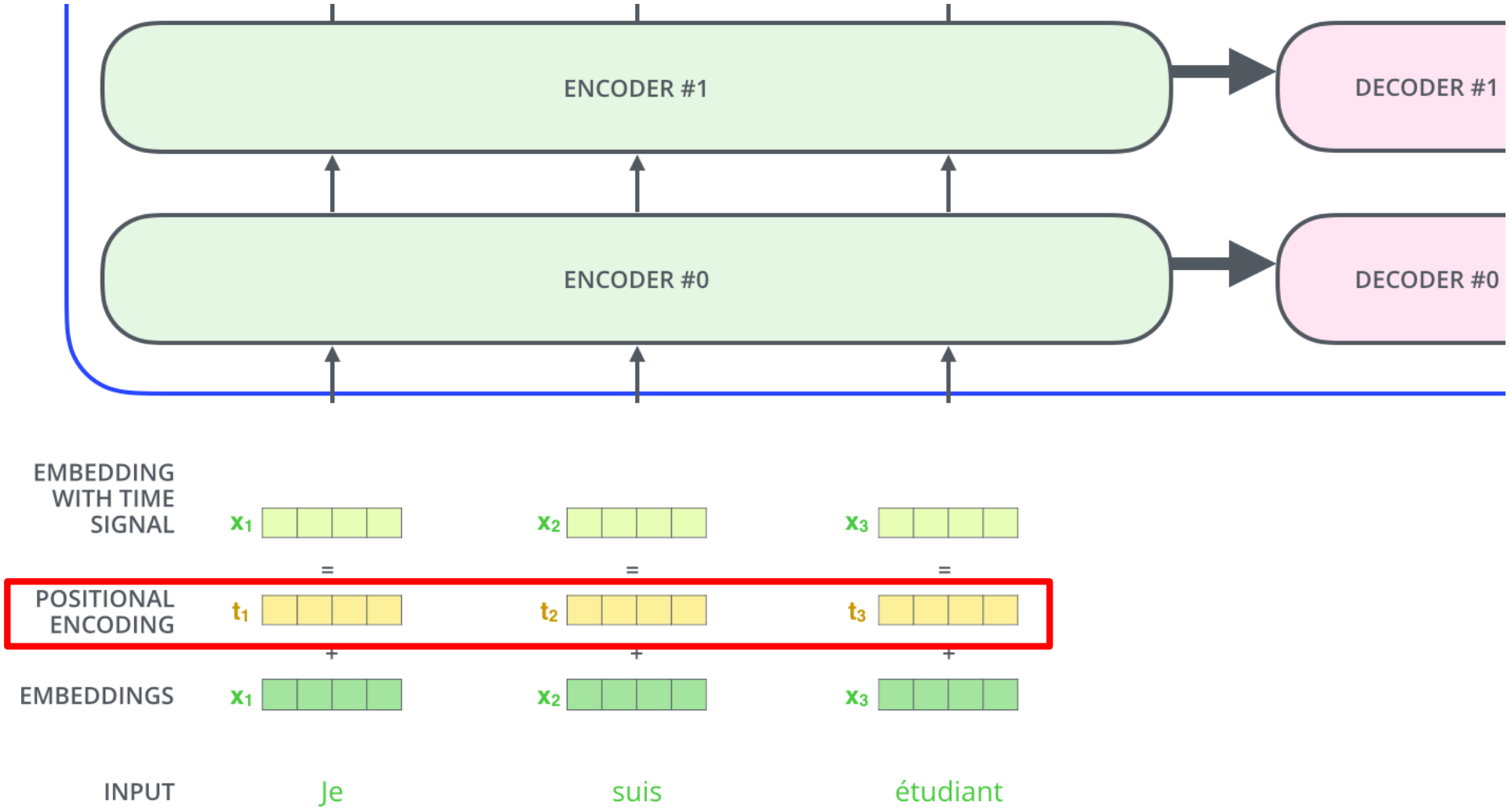
- Transformer는 RNN 혹은 CNN Layer를 사용하지 않기 때문에,
  단어들의 위치 정보를 따로 알 수 있는 방법이 없음
  : 따라서, 문장에서의 **단어들의 위치 정보를 인위로 주입**해 알려주어야 함
- 해당 위치 정보 주입 위해 Positional Encoding을 Input Embedding에 추가
- 각각의 PE는 다음과 같이 구성:

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

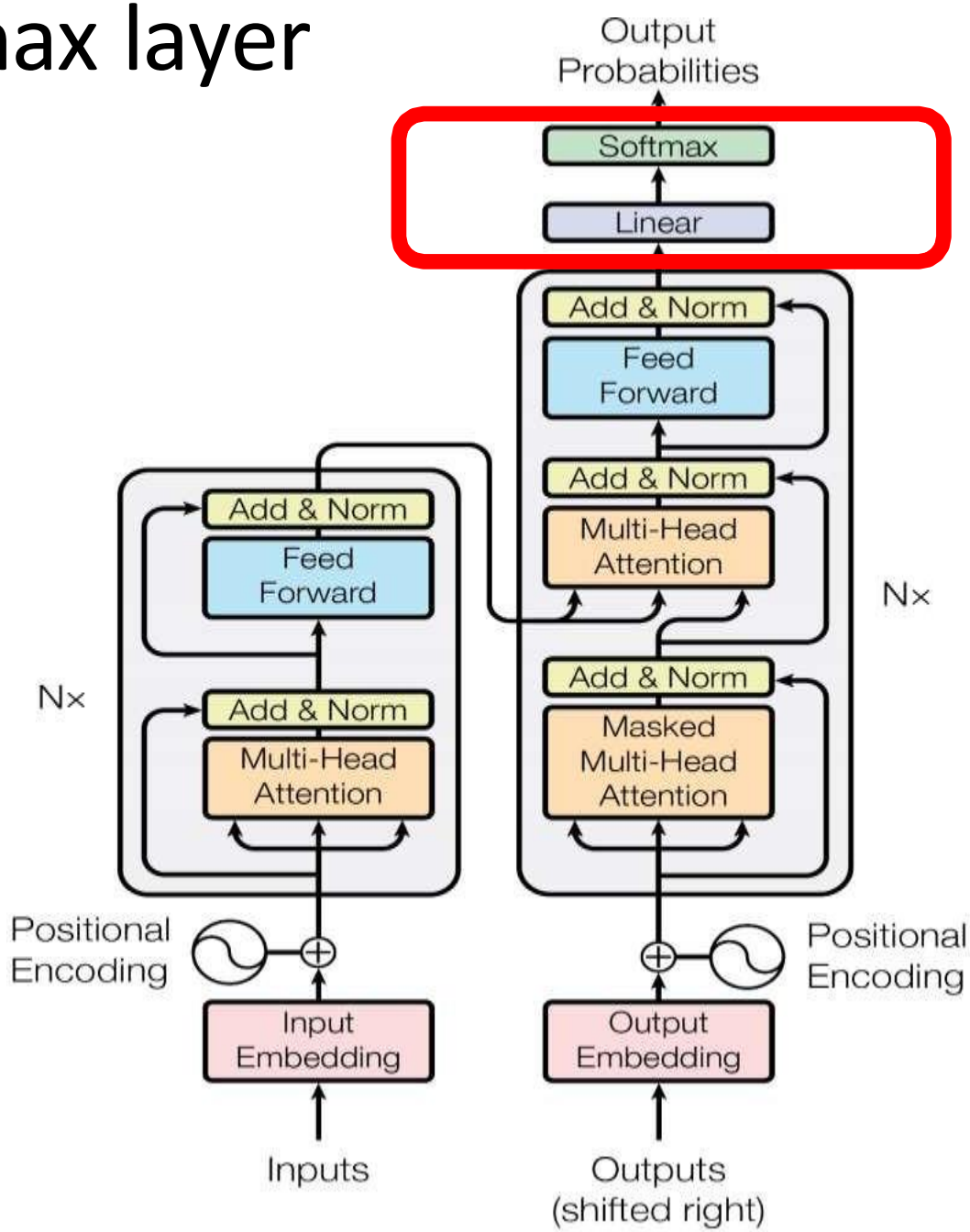
  - pos is the location of the word, i is the index of dimension in word embedding

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

Positional Encoding

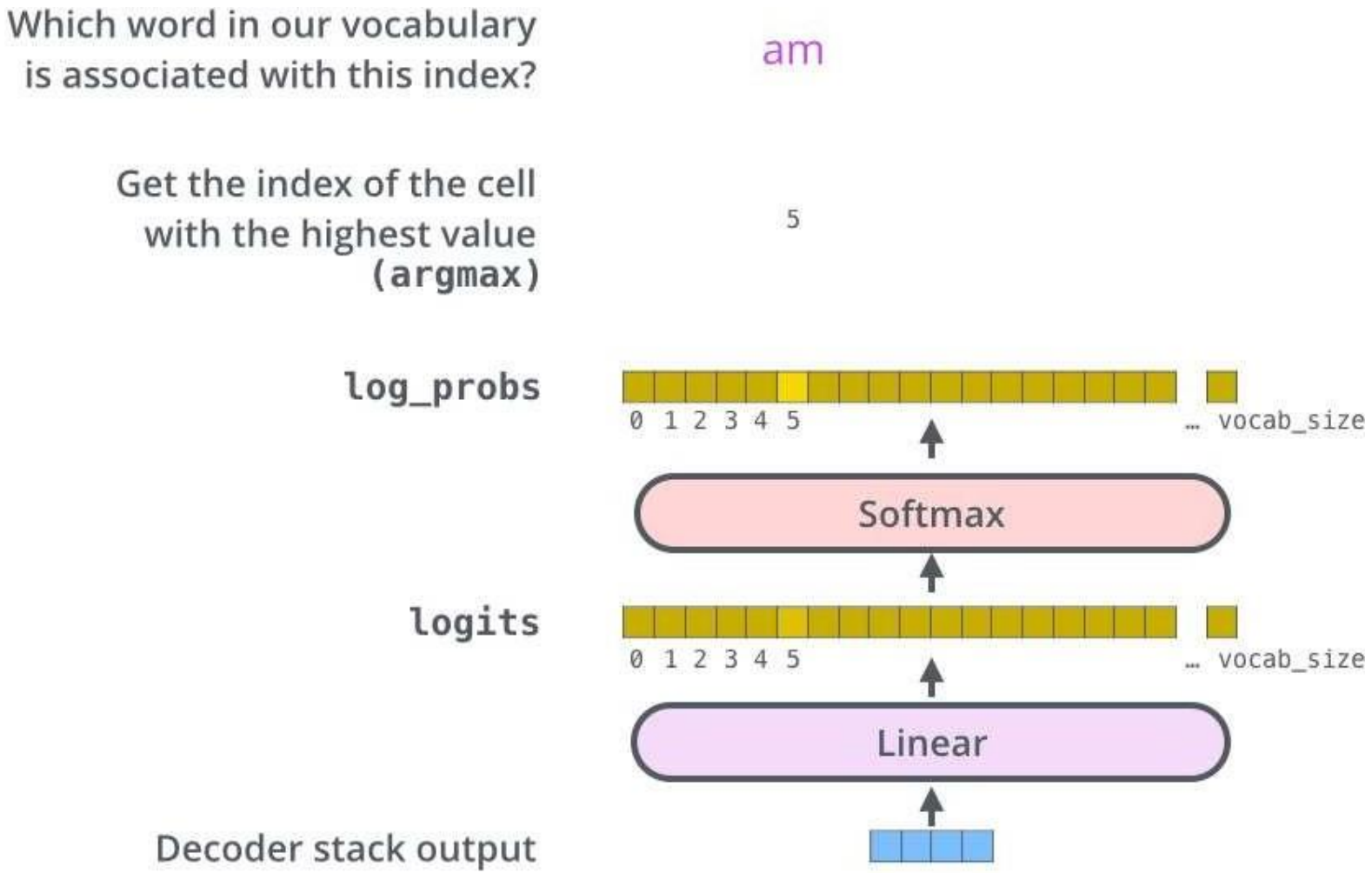


4. FC and Softmax layer

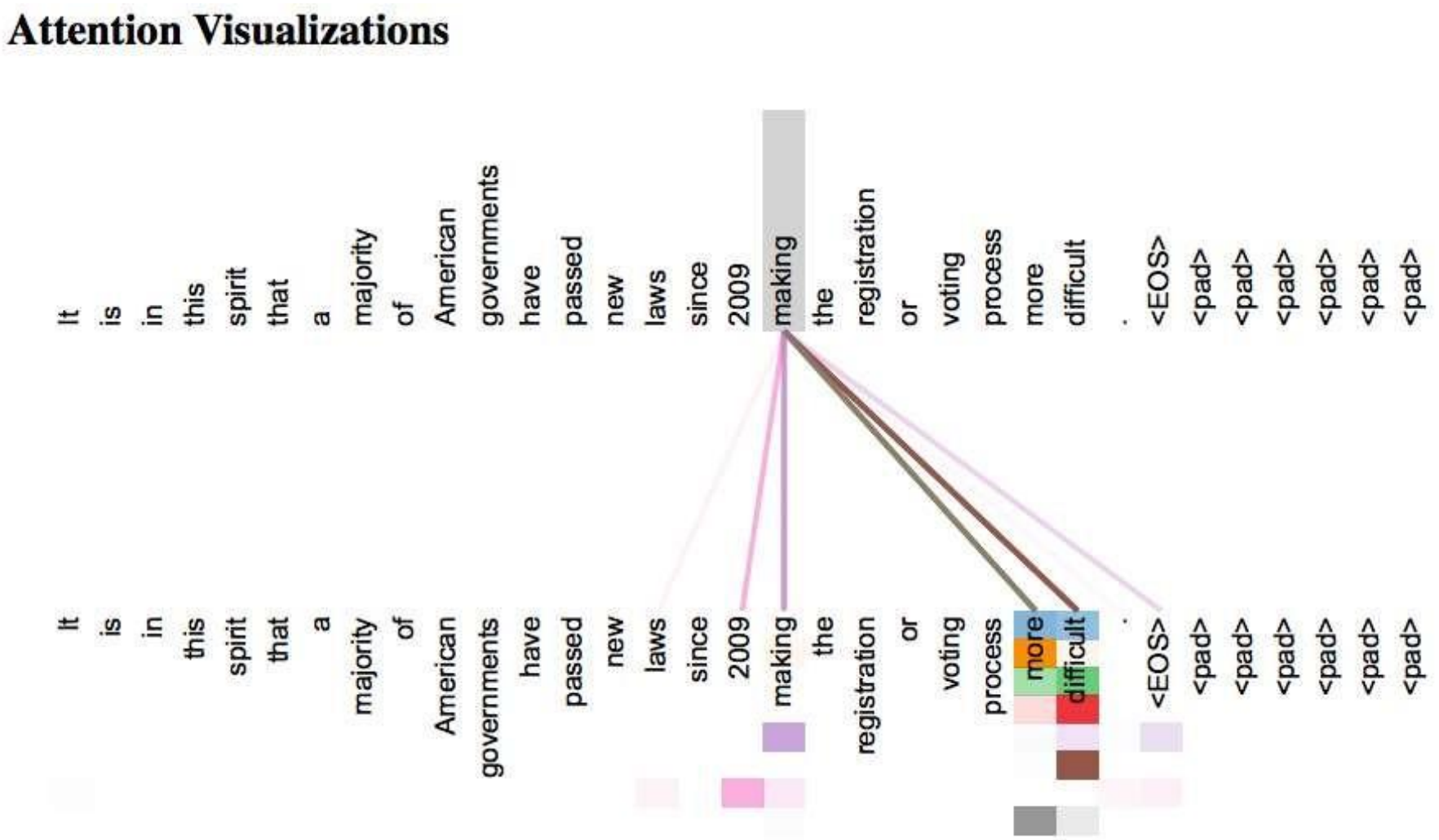


Cited from : https://jalammar.github.io/illustrated-transformer/

Final FC and softmax layer



Experiment – Self-attention visualization(with. multi-head)



Completing the phrase "making … more difficult"

Conclusion

- 역사적으로 Encoder-decoder 모델에서 핵심적으로 사용되던 RNN에 전혀 의존하지 않고, **Attention 메커니즘만 사용**한 Transformer 모델 제안

- Transformer는 RNN 혹은 CNN을 기반으로 한 모델들 보다 **빠르게 훈련**이 가능하며, 이는 병렬 처리 가능 연산량의 증가 덕분

< EOP >