TRAIN AND TEST

[CS224N] Lecture 13
Contextual Word Embeddings

INDEX

Reflections on word Representations and Pretraining

Topic1: Reflections on word Representations and Pretraining

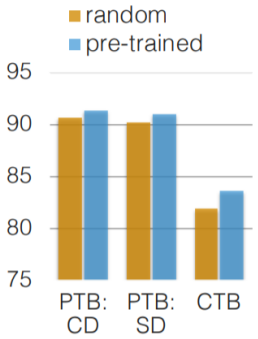
Pretraining word Vector

Q1) Pre-training?

: 내가 풀고자 하는 문제와 비슷하면서 "사이즈가 큰 데이터"로 학습된 것!

Q2) Fine-tuning?

: 기존에 학습되어져 있는 모델을 기반으로 아키텍쳐를 변형하고 이미 학습된 모델 Weights로 부터 학습을 업데이트하는 방법



	POS WSJ (acc.)	NER CoNLL (F1)
State-of-the-art*	97.24	89.31
Supervised NN	**96.37**	**81.47**
Unsupervised pre-training followed by supervised NN**	**97.20**	**88.87**
+ hand-crafted features***	97.29	89.59

기존의 모델을 바로 적용했을 때 보다 Pretraining을 사용했을 때, 성능이 향상하였다!

Unknown Token (Feat. <unk>)

Q1) Unknown Token이 뭐예요?

: Training 단계에서 보통 5번 이상 등장한 token만을 embedding 시킴!

그래서, training 되지 못한 단어에 대해서는 모델인 인식 못함!

=> 이것을 Out-of-Vocabulary (OOV) 라고 부른다

Q2) 왜 문제가 돼?

: 말 그대로 '중요 단어'인데 빠졌을 수도 있잖아

Q3) 해결방법은?

: Char-level-embedding model, pretraining 된 word vector

random vector

Topic1: Reflections on word Representations and Pretraining

기존에 배웠던 Word Vector

: Word2Vec, Glove, fastText 등

이들은 모두 One representation of Word 라는 점에서 문제가 있다!

One representation of word?

: 하나의 단어에 하나의 의미만을 나타낸다

Q1) 그래서 뭐가 문제?

A) 그러면 동음이의어는 어떻게 할건데? (Feat. 먹는 배,, vs 타는 배,,)

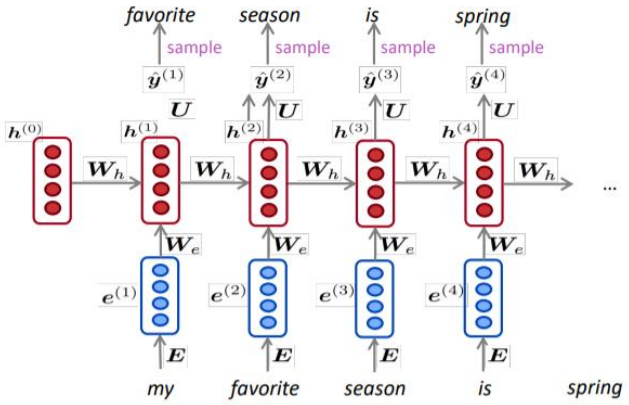
A) 그리고, 문맥마다 단어의 의미가 다른데? (Feat. '이불' 킥, '이불')

Topic1: Reflections on word Representations and Pretraining

기존에 배웠던 Word Vector

: Word2Vec, Glove, fastText 등

이들은 모두 One representation of Word 라는 점에서 문제가 있다!



LSTM의 경우, 다음 단어를 예측하는 시스템

'Context-specific word representation'이

가능하다는 것!

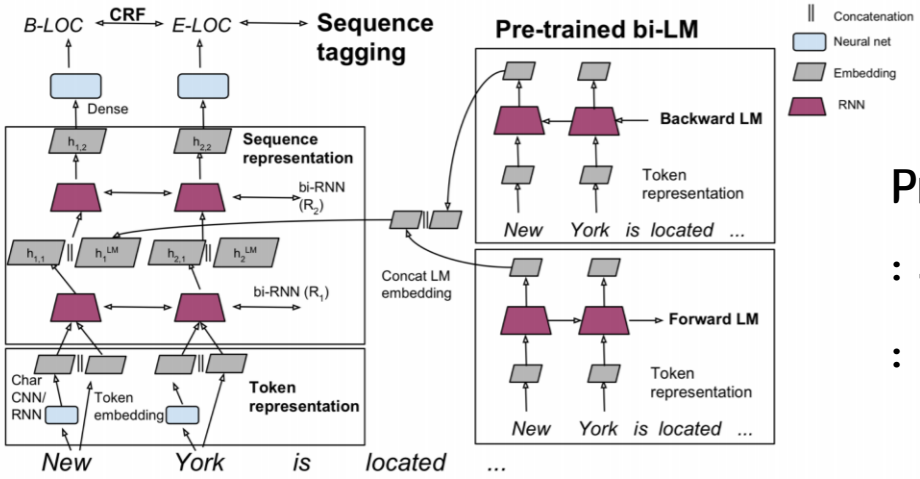
Pre-ELMo and ELMO

Topic2: Pre-ELMo and ELMO

Pre-ELMo

: RNN 모델을 통해, 'Context-specific word representation' 을 확보하고자 했으나, 학습하기에 데이터셋이 너무 작다,,

=> 그것의 답은 'Pretraining'



Pre-trained bi-LM

: 순방향과 역방향 모두 학습하여 concat

: 학습을 통해 얻어진 hidden과도 concat

$$\mathbf{h}_{k,1} = [\overrightarrow{\mathbf{h}}_{k,1}; \overleftarrow{\mathbf{h}}_{k,1}; \mathbf{h}_k^{LM}]$$

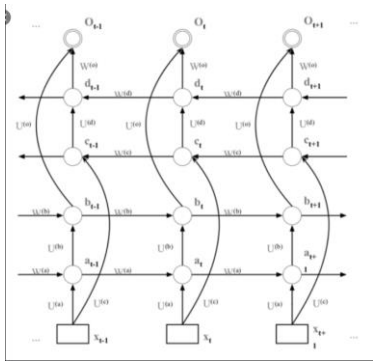
ELMO (Embedding from Language Model)

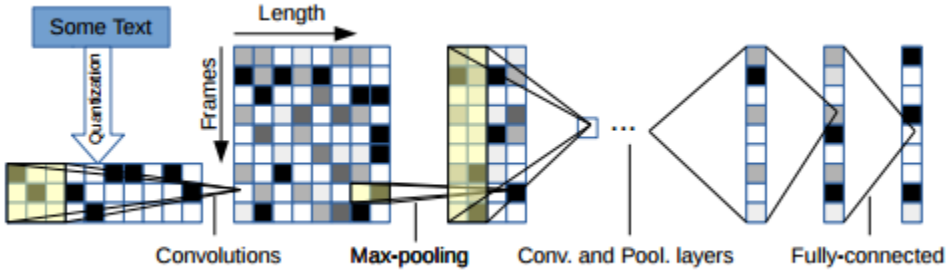
Q) 특징을 간략하게 살펴보면?

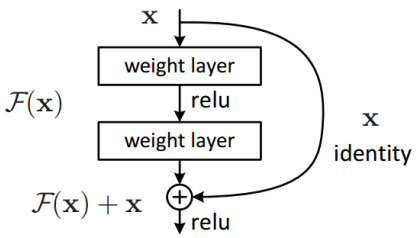
: Train a bidirectional LM (2biLSTM layers)

: Use character CNN to build initial word representation

: Use reisidual connection





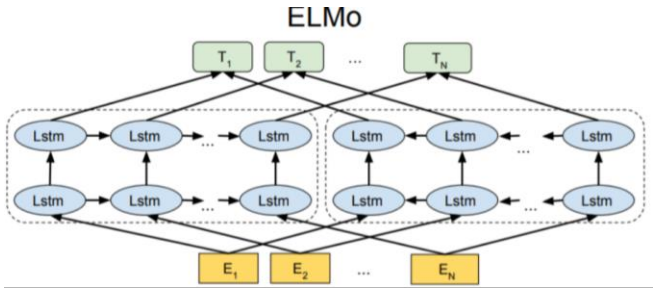


ELMO (Embedding from Language Model)

Q) 그렇다면 ELMO의 수식 해석은?

$$
\begin{aligned}
R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
&= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},
\end{aligned}
$$

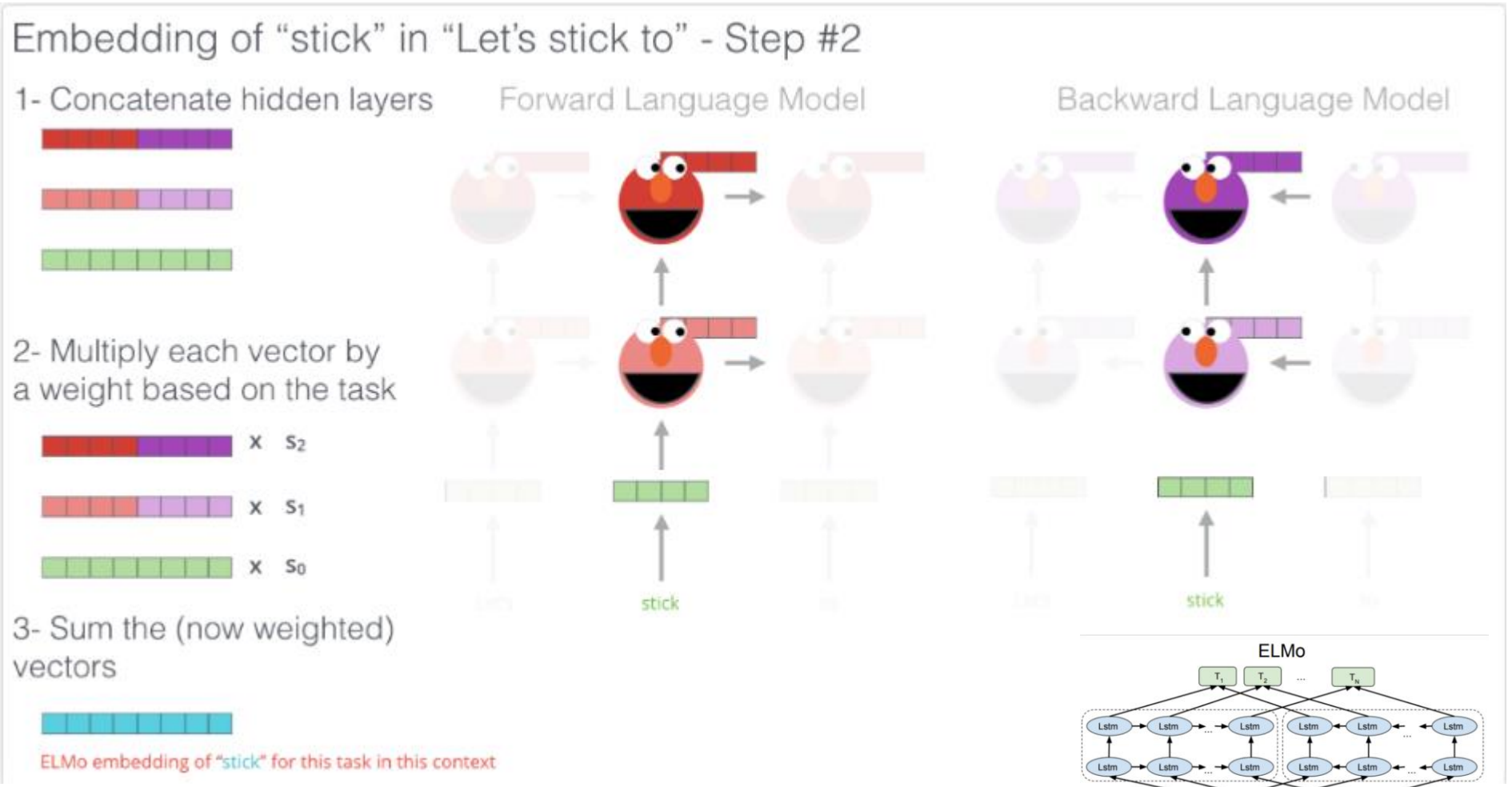
$$
\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}
$$



임베딩 값을 'weight sum' 한것 과 같다!

: X (기존의 임베딩), h (순방향 임베딩), h (역방향 임베딩)

Topic2: Pre-ELMo and ELMO



Transformer architectures

Transformer architectures

!요즘 핫한 BERT, GPT-3 모두 Transformer 기반 모델!

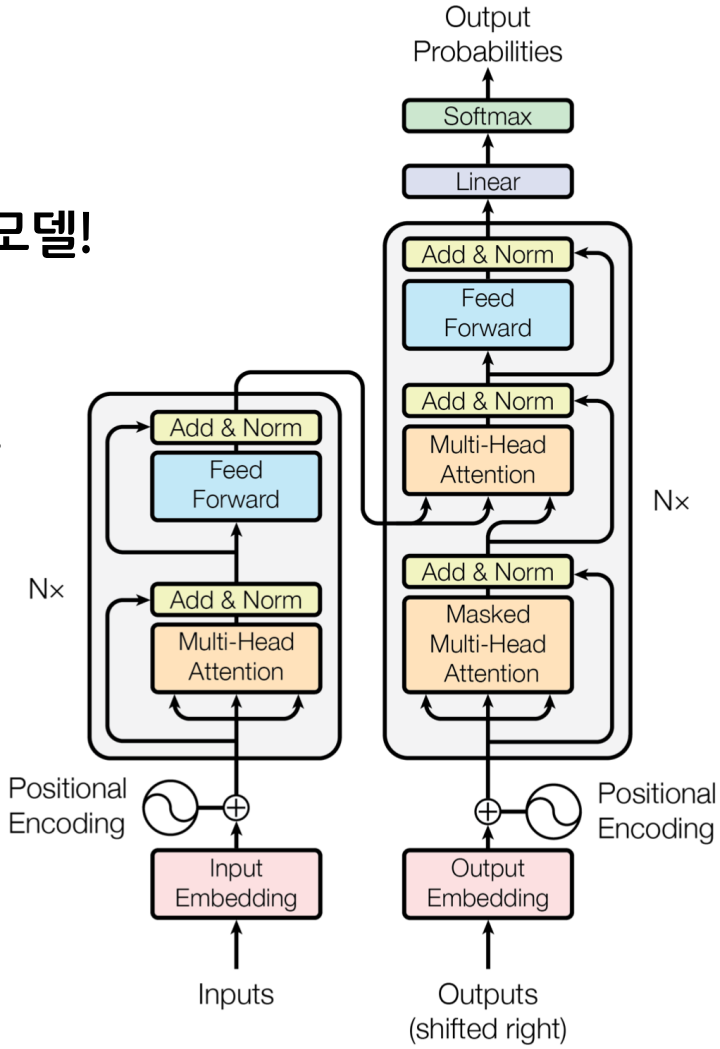
Q) 그렇다면 Transformer의 구조는 어떻게 생김?

: 기본적으로 Encoder와 Decoder로 나눠져 있음!

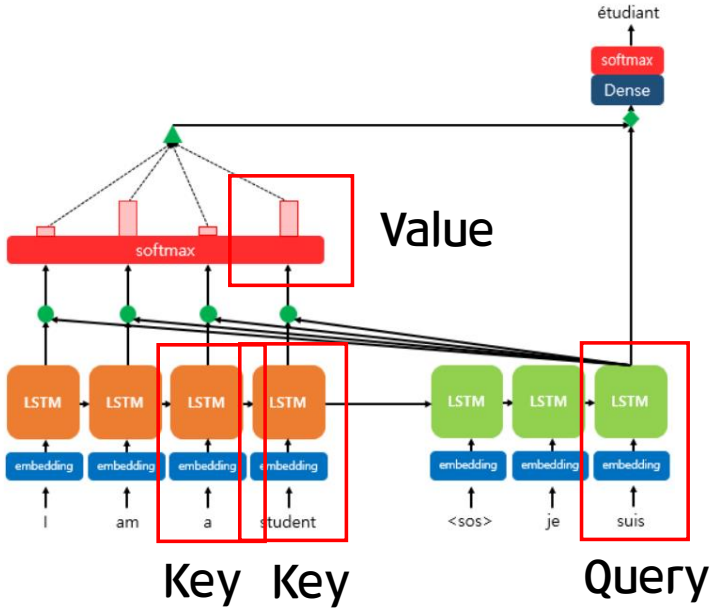
Attention-based-model

Q) Attention이 뭐가 부족해서 생겼을까?

: 기본적으로 속도의 차이!가 심각하게 난다.



Attention architectures (복습)



<Attention 개념의 핵심 용어>

* Key, Query, Value

Query : '디코더'의 값

Key : '인코더'의 값

Value : Query와 Key의 유사도 값

한 Query에 대해 유사도를 계산하기 위해 우리는 모든 Key들을 다시 봐야 한다는 것이 문제!

즉, 무조건 인코더 계산이 이루어진 후에, 우리는 디코더에 대한 계산이 된다. (parallelization)

Transformer architectures

!요즘 핫한 BERT, GPT-3 모두 Transformer 기반 모델!

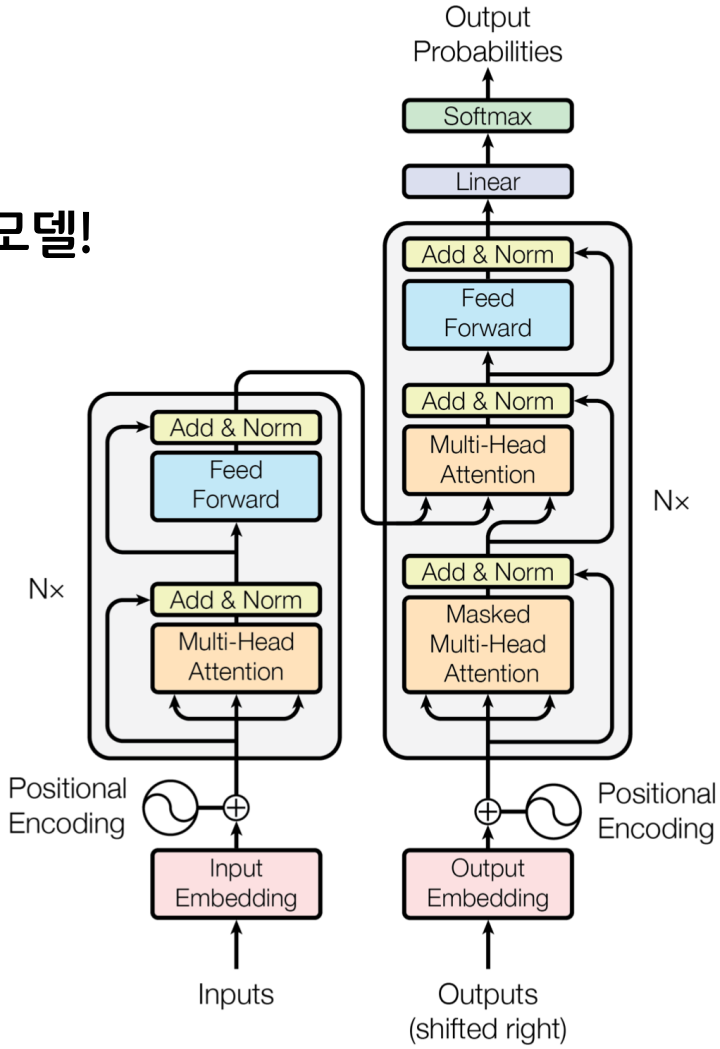
각각이 무엇인지부터 파악해보자.

Encoder & Decoder stack

Positional Encoding

Multi-Head Attention

Feed Forward



Transformer architectures

1) Encoder & Decoder stack

: 동일한 layer(회색) N번 반복

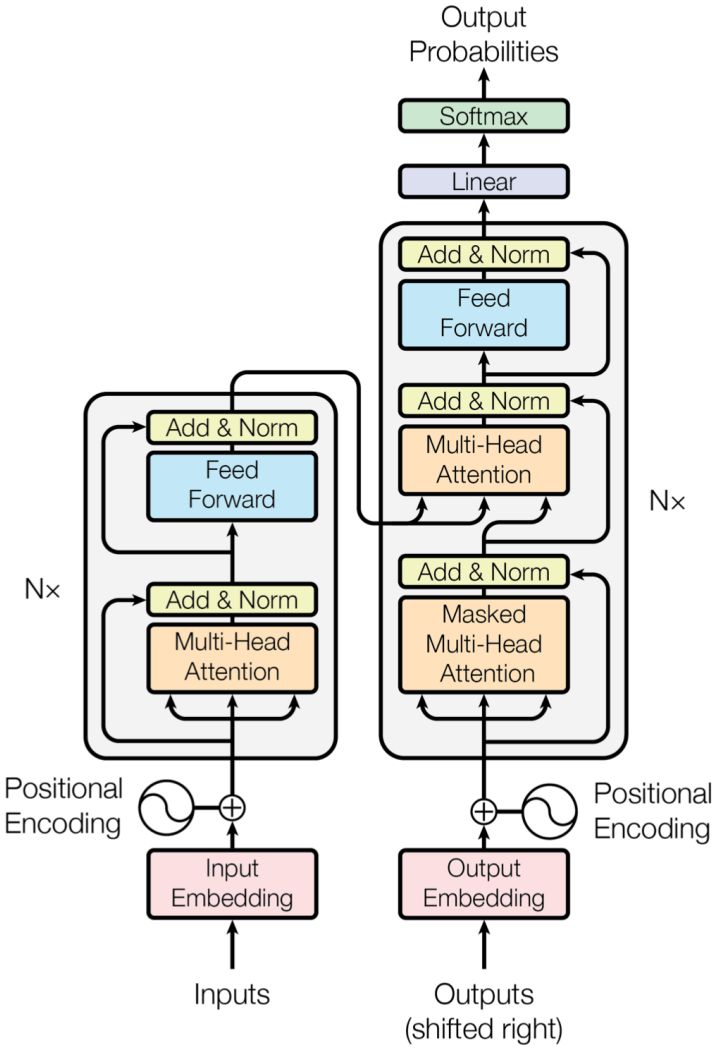
Encoder: 두 개의 sub-layer로 구성

Decoder: Encdoer의 output에

Multi-Head attention 추가

전체적으로, residual connection 추가

residual 값을 더한 후 Normalize



Transformer architectures

2) Positional Encoding

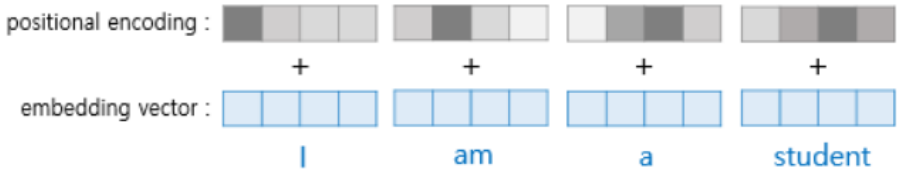
: "앞서, 우리는 CNN, RNN을 쓰지 않았다" = Transformer

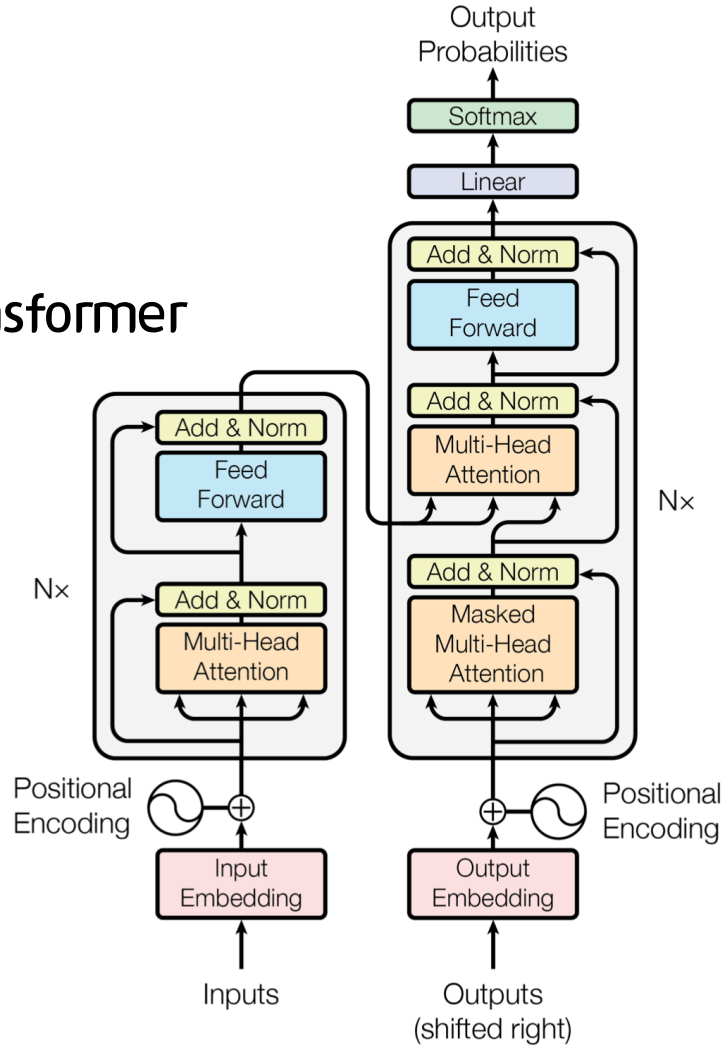
Q) 어떻게 순서를 기억할 건데?

A) positional encoding 쓸거임

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

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

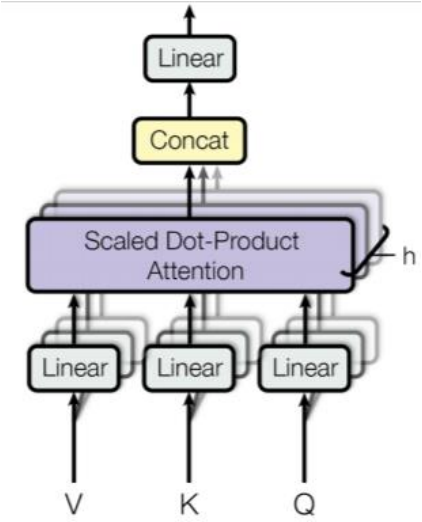




Transformer architectures

3) Multi-Head Attention

: 말 그대로, attention head를 여러 개 가지는 것

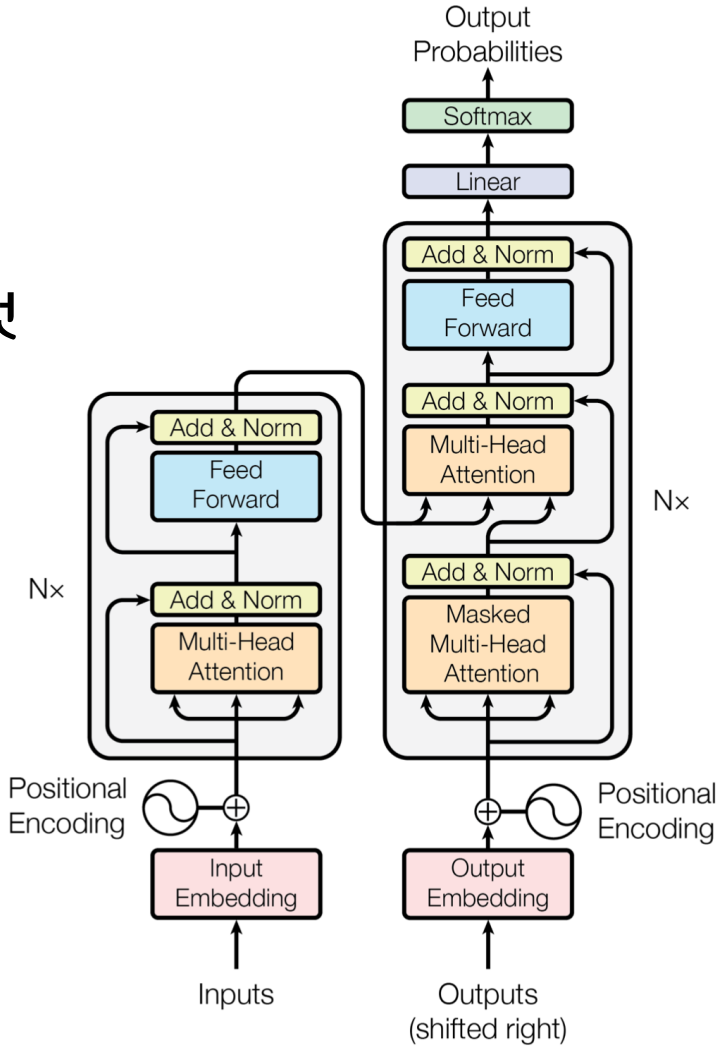


<기존의 attention>

Query : '디코더'의 값

Key : '인코더'의 값

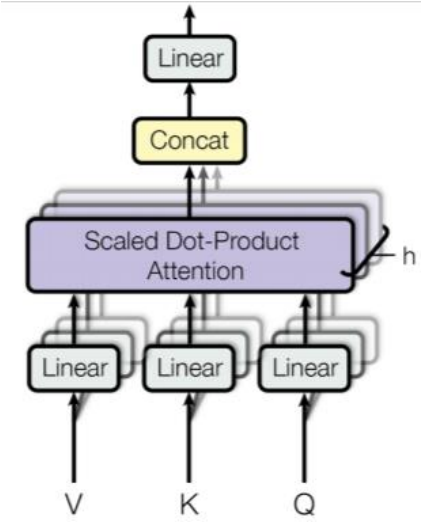
Value : Query와 Key의 유사도



Transformer architectures

3) Multi-Head Attention

: 말 그대로, attention head를 여러 개 가지는 것

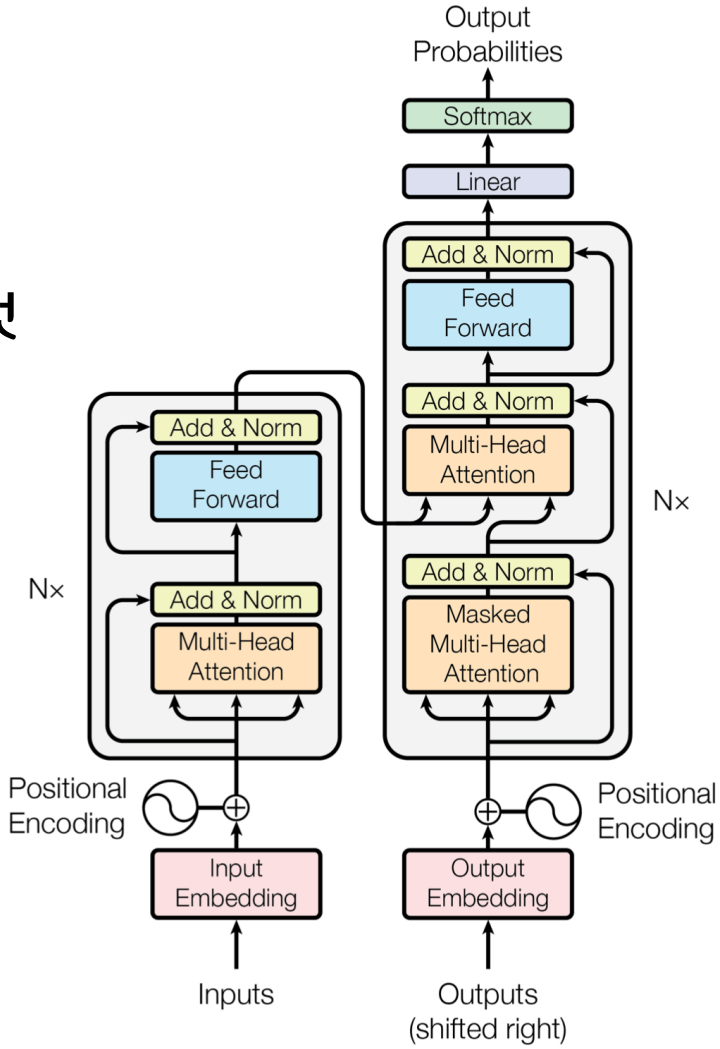


<Self attention>

Query : '인코더'의 값

Key : '인코더'의 값

Value : '인코더'의 값



<Self Attention 예시>

Transformer architectures

2) Multi-Head Attention

'나는 이홍정' 이라는 문장이 있다면,

: 말 그대로, attention head를 여러 개 가지는 것

Query : '나'

Key : '나', '는', '이홍정'

Value : '나' X '나', '나' X '는', '나' X '이홍정'

<Self attention>

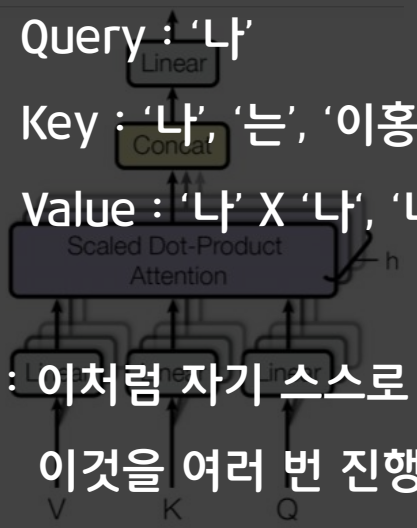
Query : '인코더'의 값

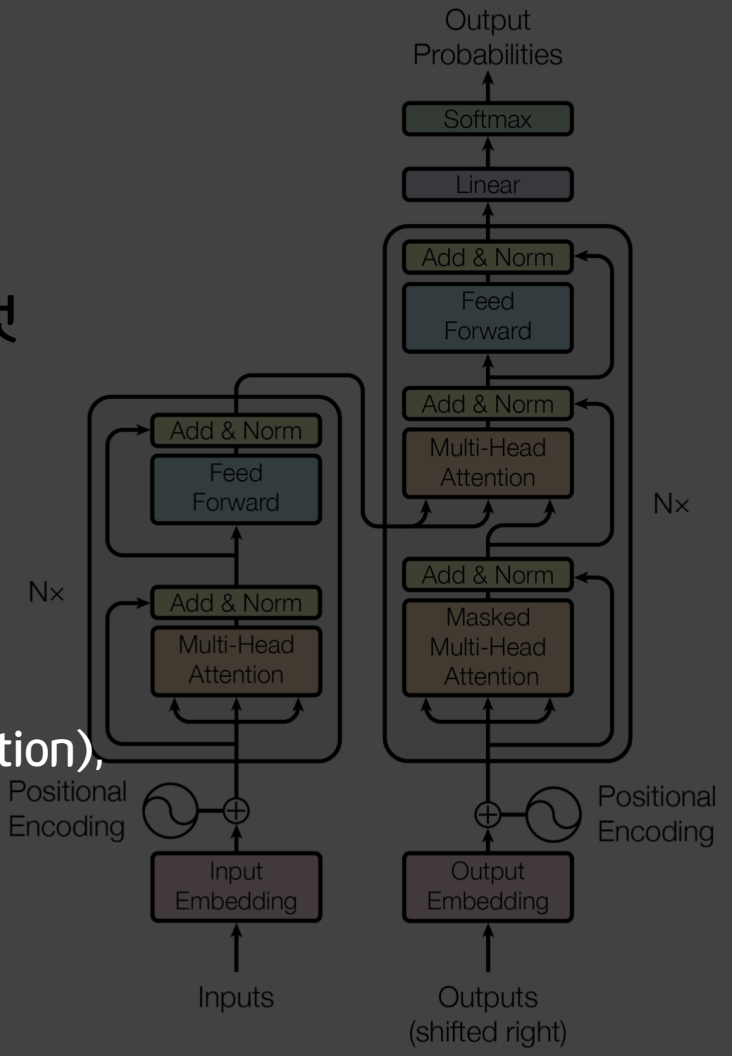
Key : '인코더'의 값

Value : '인코더'의 값

: 이처럼 자기 스스로 attention을 진행하며 (self attention),

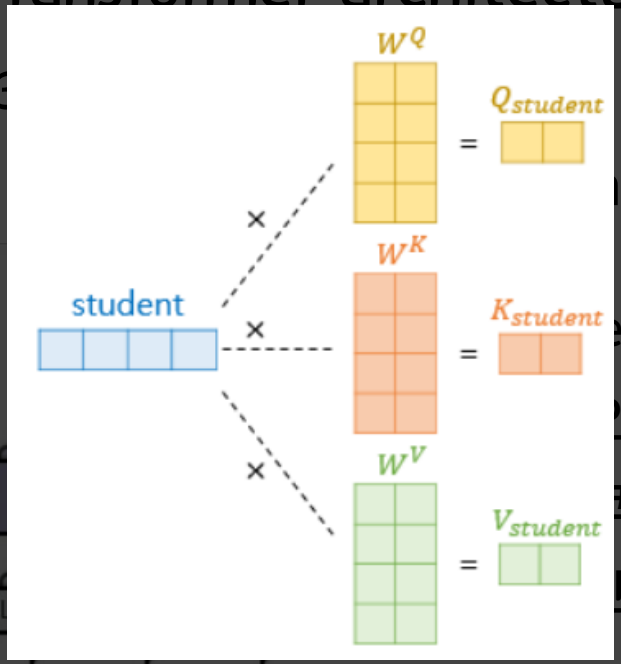
이것을 여러 번 진행한다. (Multi head)





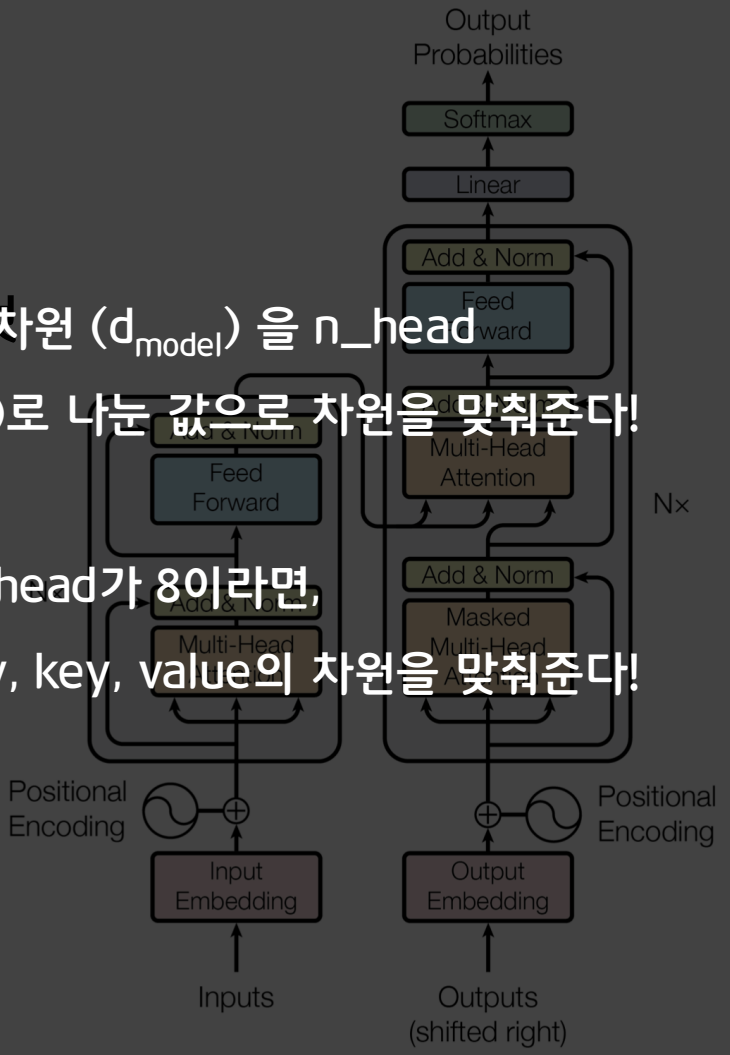
<Multi Head Attention 예시>

Transformer architectures



기존의 embedding 차원 (d_{model}) 을 n_head (multi head의 개수)로 나눈 값으로 차원을 맞춰준다!

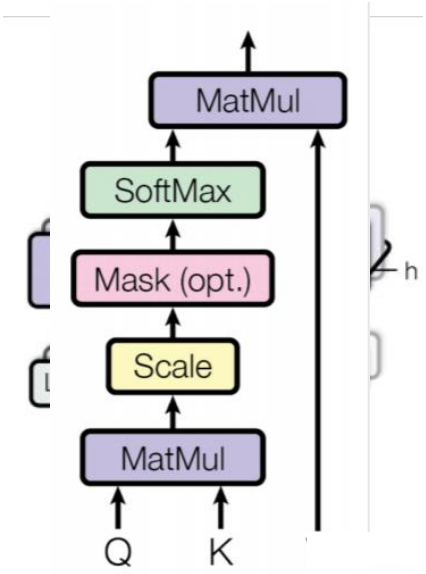
d_{model}이 512, multi head가 8이라면, 64 (512/8)로 query, key, value의 차원을 맞춰준다!



Transformer architectures

3) Multi-Head Attention

: 말 그대로, attention head를 여러 개 가지는 것



<scaled Dot-Product Attention>

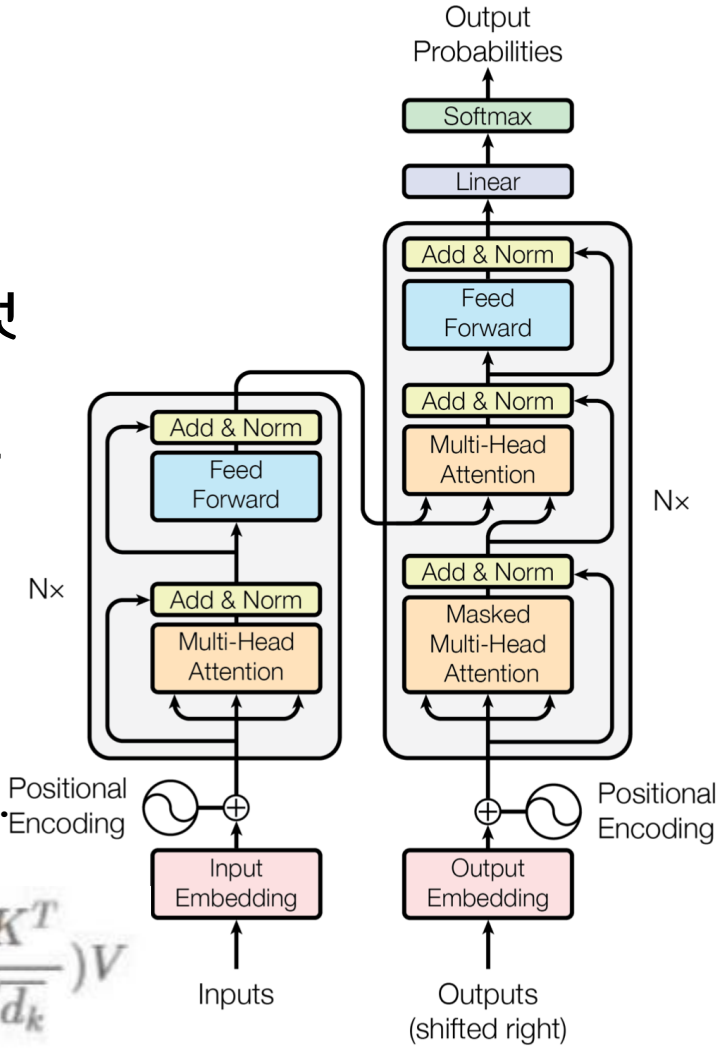
Query와 Key의 유사도를 구한 후,

Scale를 해준다!

그냥 softmax를 취하면

기울기 변하지 않는 곳으로 향한다.

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



Transformer architectures

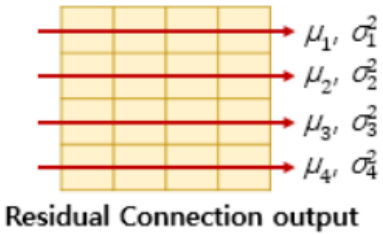
4) Feed Forward

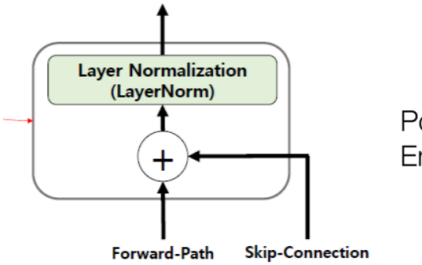
: 이전부터 해왔던 단순히 Fully Connected layer 후 relu

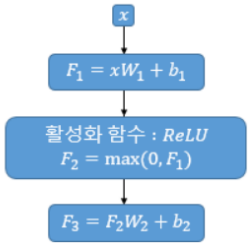
5) Add & Norm

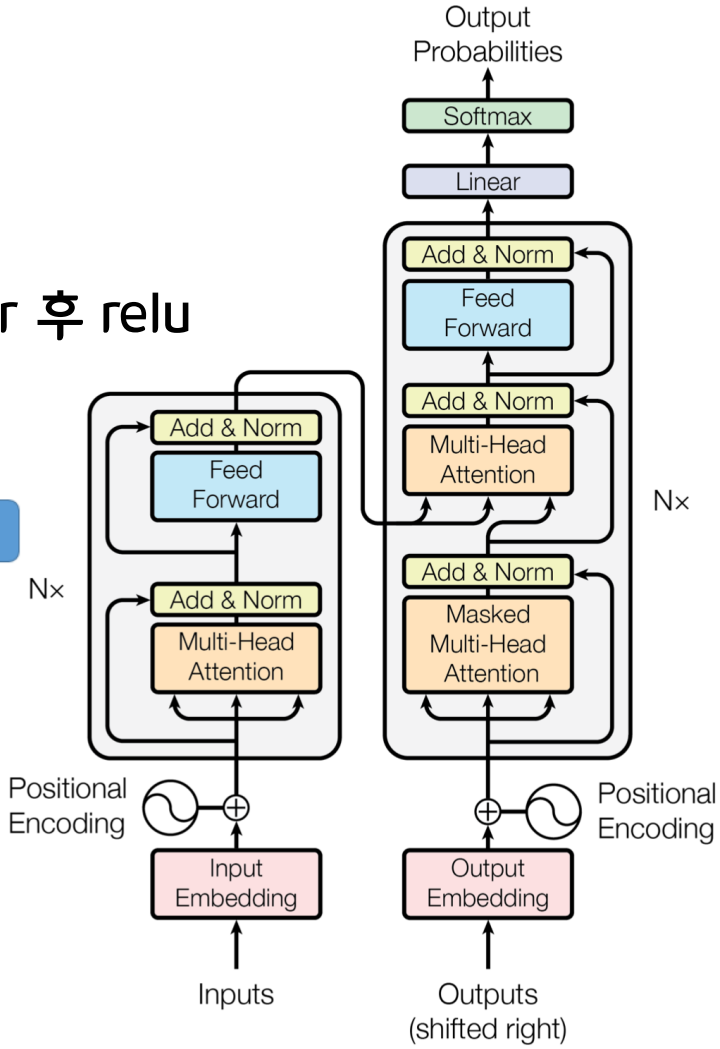
: Skip connection을 받고,

정규화를 한다!









Transformer architectures

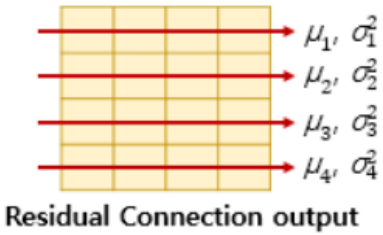
4) Feed Forward

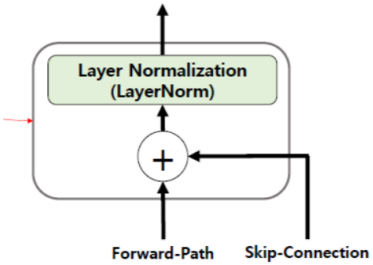
: 이전부터 해왔던 단순히 Fully Connected layer 후 relu

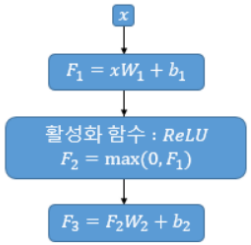
5) Add & Norm

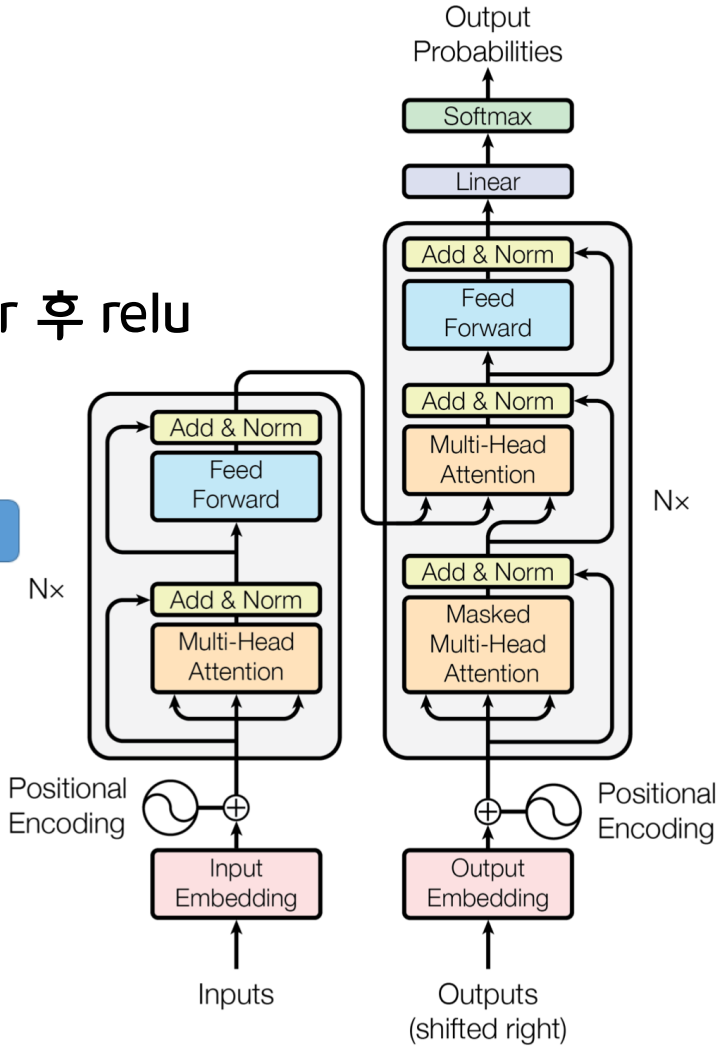
: Skip connection을 받고,

정규화를 한다!









Transformer architectures

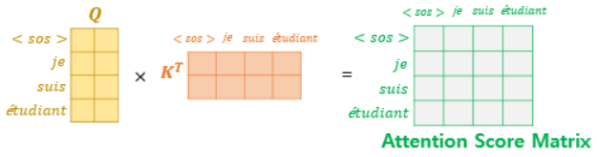
전체적인 구조

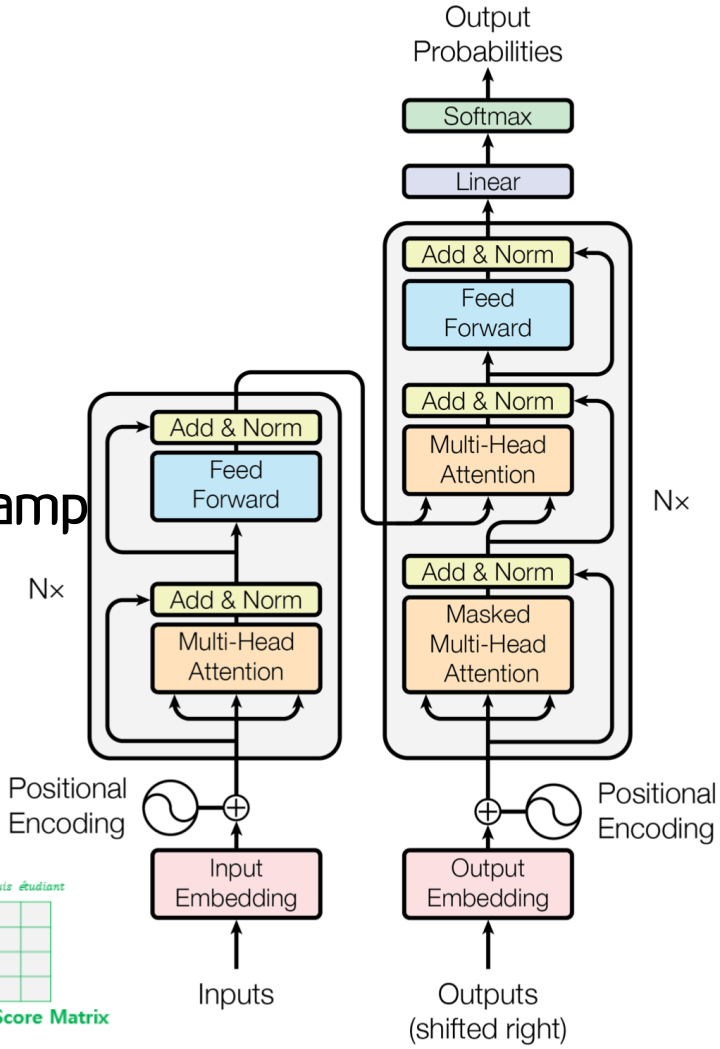
Encoder Block (한국어-예시)

Decoder Block (영어-예시)

: Masking을 통해 query 단어가 이전의 timestamp 의 key 단어만 보도록 설정

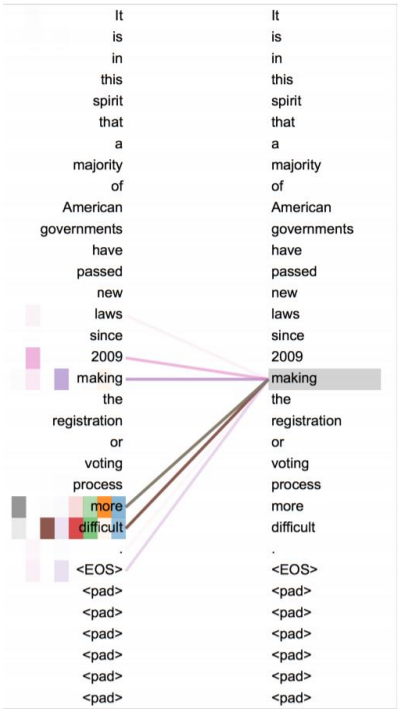
: 두번째 block에서 Enconder의 output을 key, value로 받아 target query와 self-attention 진행





Transformer architectures

결과해석

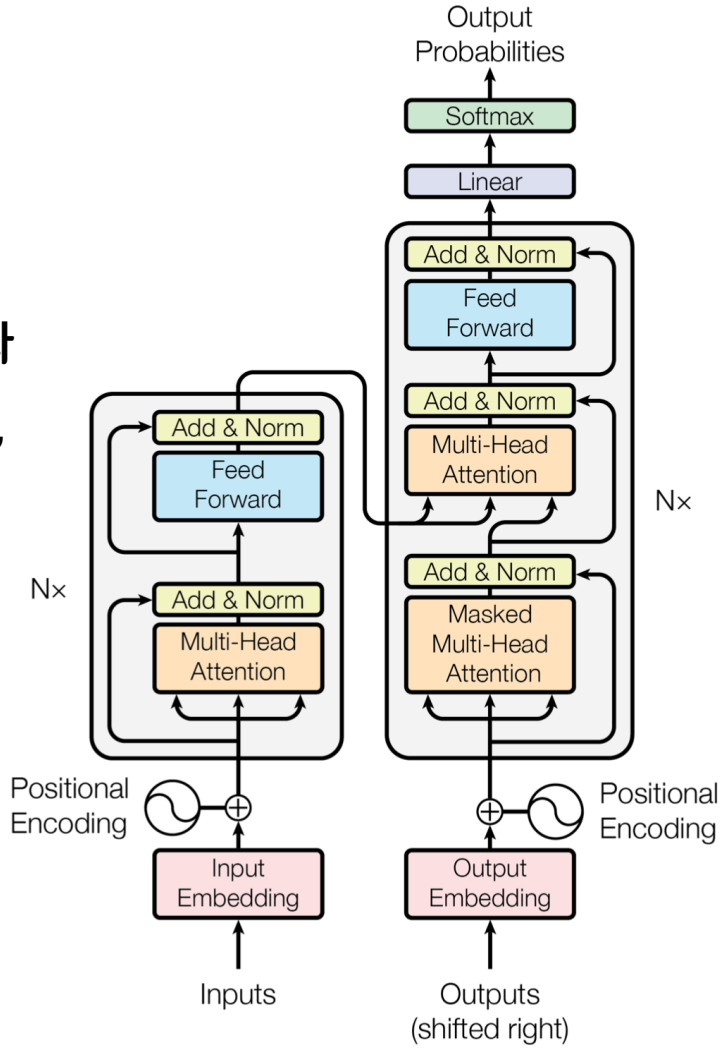


'Making'이 self attention한 결과

우리는 8개의 multi head를 이용,

따라서, 8개의 조각마다 어디에

집중했는지 파악할 수 있다!



TRAIN AND TEST