ACADENTIAL

Section 16.
Attention과 Transformer

ACADENTIAL

목차

- 섹션 14. Convolution Neural Network (CNN)
- 섹션 15. Recurrent Neural Network (RNN)
- 섹션 16. Attention과 Transformer

ACADENTIAL

Objective
학습 목표

1부: Attention의 기본 개념

- Attention의 개념과 정의
- Attention을 행렬의 곱으로 이해하기 (Vectorized)

2부: Attention의 계보

- Neural Machine Translation by Jointly Learning to Align and Translate
- Attention is all you need
- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

3부: Transformer

- Scaled dot product
- Positional Embedding
- Multi-head attention
- Transformer의 Encoder, Decoder

ACADENTIAL

16-1. Attention의 기본 개념

ACADENTIAL

Attention

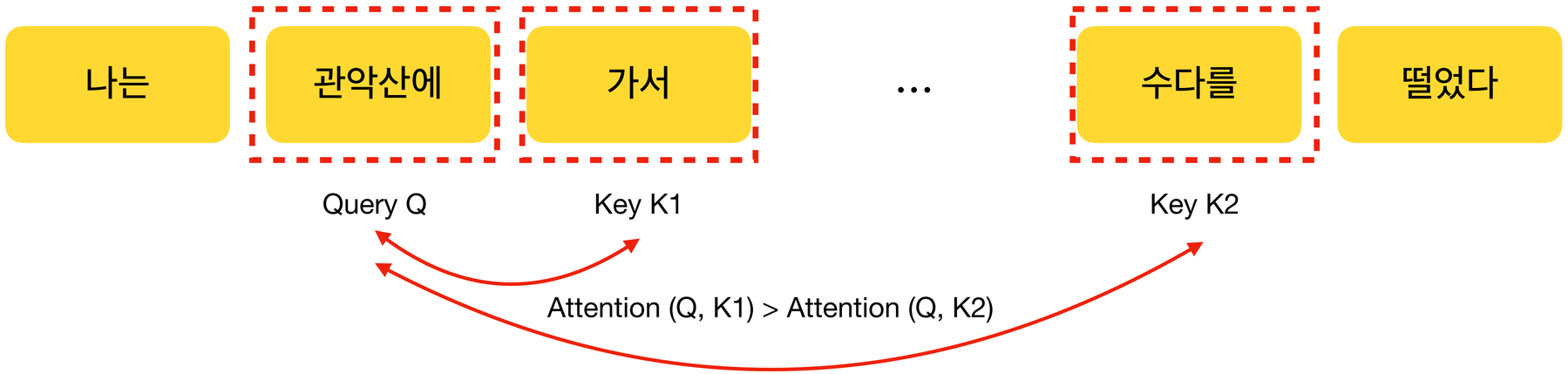
Introduction

- Attention == “관심, 주의, 주목”
- 배열 상의 어떤 단어 (query)와 다른 단어들 (key)간에 연관성이 서로 얼마나 높을까?

Attention

Introduction

- Attention == “관심, 주의, 주목”
- 배열 상의 어떤 단어 (query)와 다른 단어들 (key)간에 연관성이 서로 얼마나 높을까?
- 즉, attention은 어떤 단어 (query)와 다른 단어 (key)간의 연관성이다!



Attention

- Attention은 3가지 요소로 구성됨:
  - Query Q
  - Key K
  - Value V

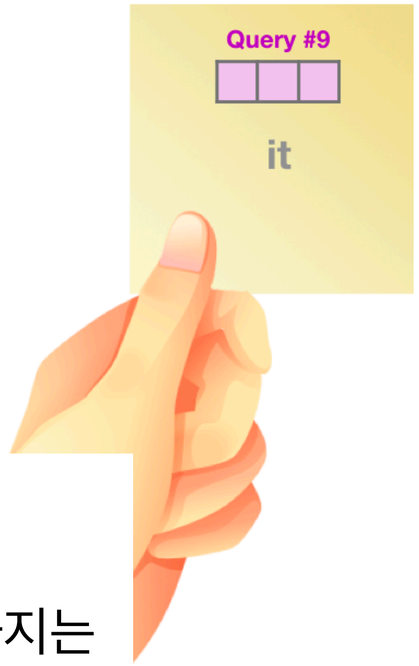
Attention

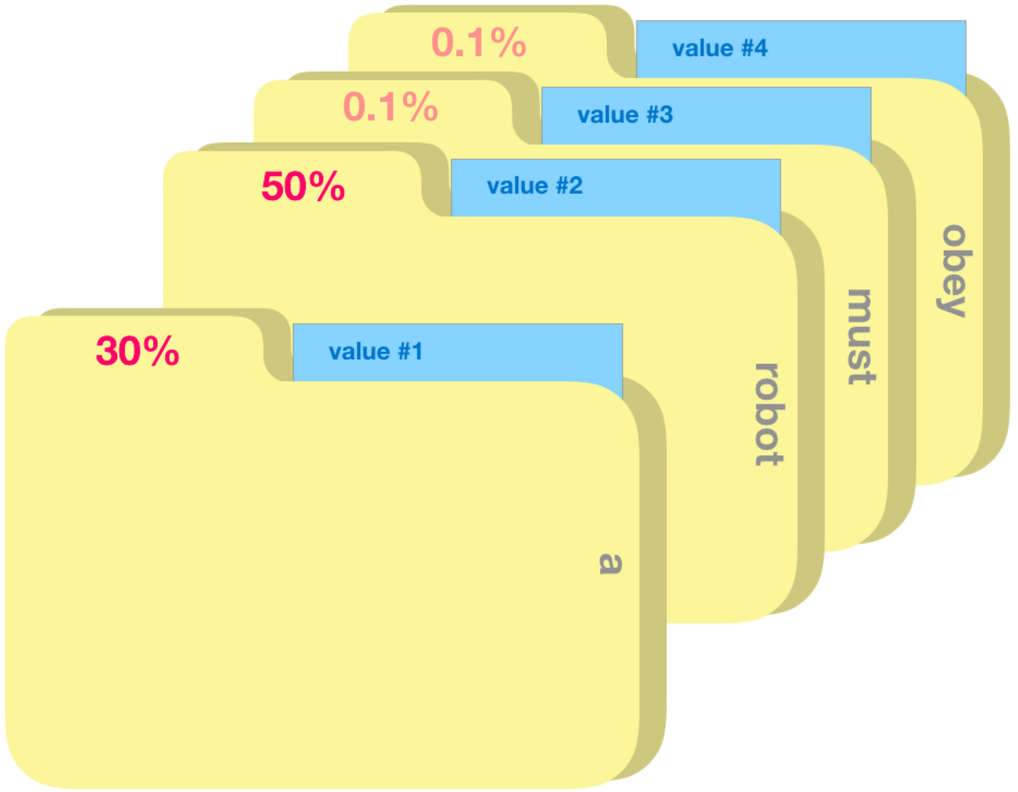
- Attention은 3가지 요소로 구성됨:
  - Query Q: 쿼리 (질문)
  - Key K: 키 (답안)
  - Value V: 값 (내용)

ACADENTIAL

Attention

- Query Q: 쿼리 (질문)
- Key K: 키 (답안)
- Value V: 값 (내용)





1. “it”을 표현하는 “Query #9”
2. **Query**을 각 **Key**와 비교
3. **Query**와 **Key**가 가장 높은 연관성을 가지는 것을 고른다.
4. 해당 **Key**에 해당되는 **Value**을 출력한다.

출처: https://jalammar.github.io/illustrated-gpt2/

ACADENTIAL

Attention

- 그렇다면 Attention의 구성 요소인 **Query, Key, Value**은 어떻게 구하는가?
  - 2가지 방법: “Self-attention”, “Attention”
- **Self-attention**:

  동일한 hidden state 혹은 activation 값으로부터 linearly project한 경우
- **Attention**:

  서로 다른 hidden state 혹은 activation 값으로부터 linearly project한 경우

ACADENTIAL

Attention

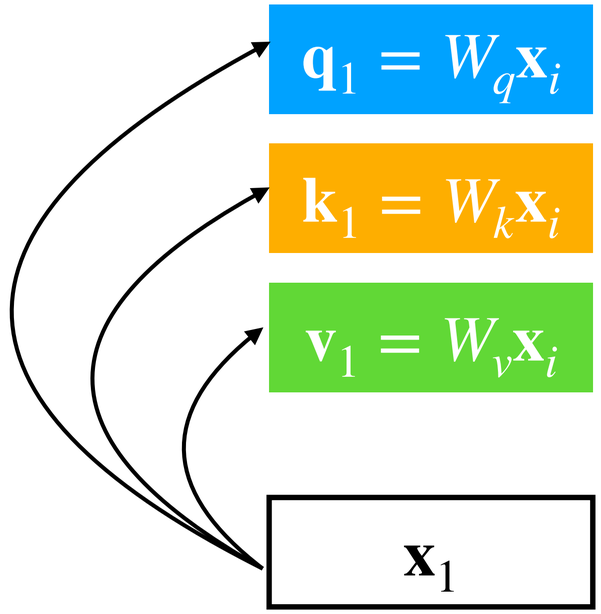
Self-attention:

동일한 hidden state 혹은 activation 값으로부터 linearly project

- $\mathbf{q}_i = W_q \mathbf{x}_i$
- $\mathbf{k}_i = W_k \mathbf{x}_i$
- $\mathbf{v}_i = W_v \mathbf{x}_i$

동일한 state $\mathbf{x}_i$ 로 부터 비롯됨!

물론 weight은 query, key, value 각각에 대해서 따로 정의된다.



ACADENTIAL

Attention

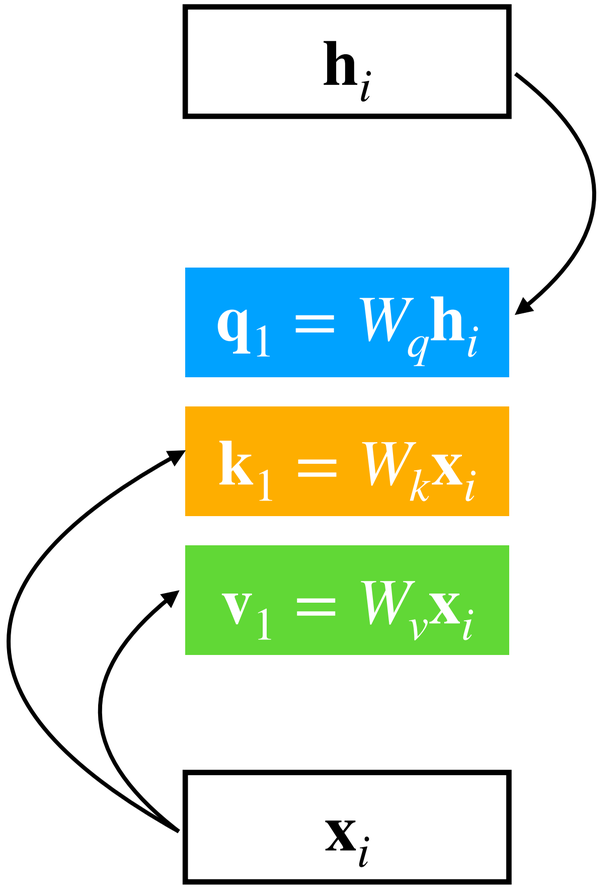
Attention:

서로 다른 hidden state 혹은 activation 값으로부터 linearly project

- $\mathbf{q}_i = W_q \mathbf{h}_i$
- $\mathbf{k}_i = W_k \mathbf{x}_i$
- $\mathbf{v}_i = W_v \mathbf{x}_i$

state $\mathbf{x}_i$, $\mathbf{h}_i$로 부터 비롯됨!

(참고로, projection 되는 input들이 모두 다 다를 필요는 없다.)



ACADENTIAL

Attention

- 그렇다면 Self-attention이 계산되는 과정을 한 번 살펴보자!
- 참고로, Attention의 경우 input만 다를 뿐 나머지 계산 과정은 self-attention과 동일하다.

ACADENTIAL

Attention

Self-attention

Mapping to $\mathbf{q}, \mathbf{k}, \mathbf{v}$

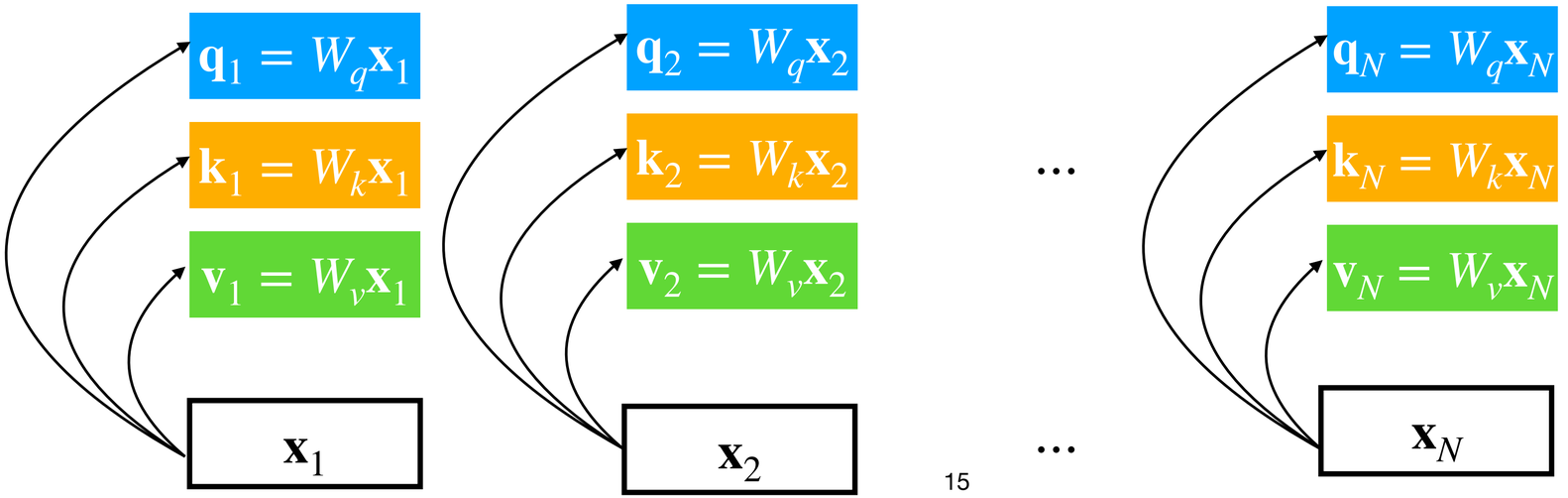
- $\mathbf{x}_i \in \mathbb{R}^D$은 i 번째 토큰에 대한 embedding 혹은 hidden state이다.
- W_q, W_k, W_v : $\mathbf{x}_i$을 query $\mathbf{q}_i$, key $\mathbf{k}_i$, value $\mathbf{v}_i$ vector에 mapping 해주는 weight matrix들이다.
- $W_q, W_k, W_v \in \mathbb{R}^{H \times D}$ (참고로 H 은 query, key, value vector들의 dimension이다)

Attention

Self-attention

Mapping to $\mathbf{q}, \mathbf{k}, \mathbf{v}$

- $\mathbf{x}_i \in \mathbb{R}^D$은 i 번째 토큰에 대한 embedding 혹은 hidden state이다.
- W_q, W_k, W_v : $\mathbf{x}_i$을 query $\mathbf{q}_i$, key $\mathbf{k}_i$, value $\mathbf{v}_i$ vector에 mapping 해주는 weight vector들이다.
- $W_q, W_k, W_v \in \mathbb{R}^{H \times D}$ (참고로 H 은 query, key, value vector들의 dimension 이다)
- $\mathbf{q}_i = W_q \mathbf{x}_i, \quad \mathbf{k}_i = W_k \mathbf{x}_i, \quad \mathbf{v}_i = W_v \mathbf{x}_i$



Attention

Self-attention

Attention score 및
Attention weight 계산

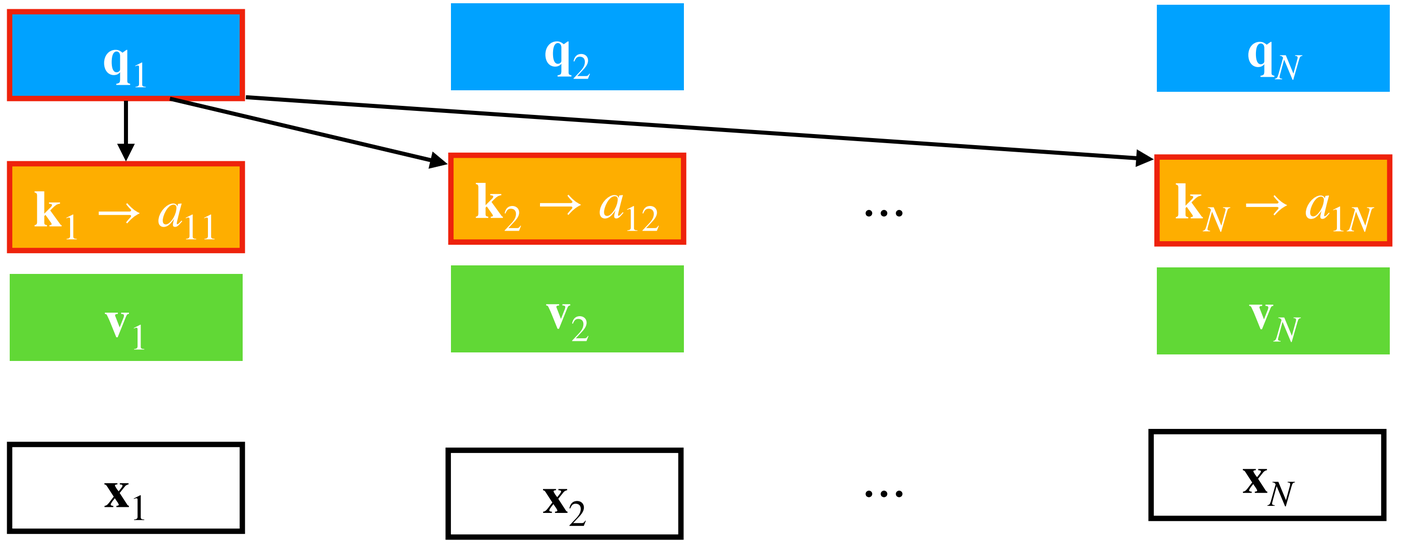
- attention score을 계산:

$$e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$$

- attention weight을 계산:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_k \exp(e_{ik})}$$

- a_{ij} 은 "i 번째 토큰에 대해서 j 번째 토큰이 얼마나 연관있는지 나타내는 값"이다!



Attention
Self-attention

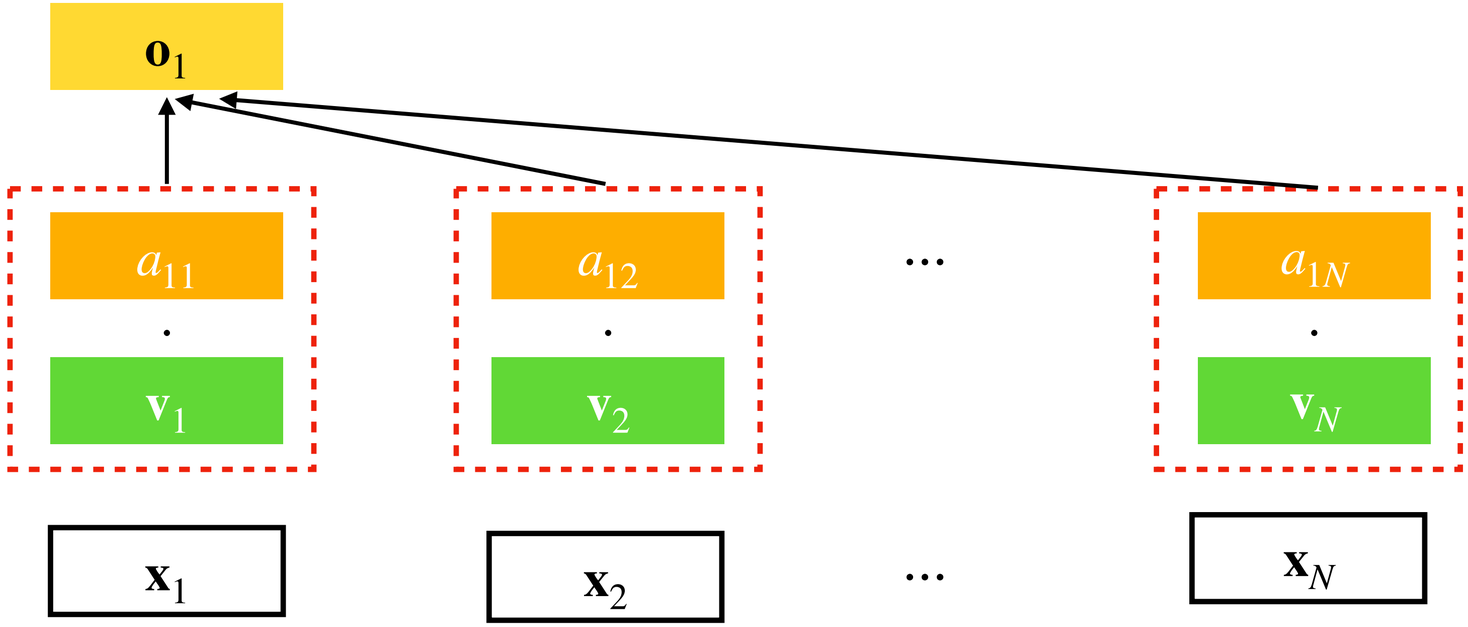
ACADENTIAL

Attention-weighted 합 구하기

- **Output 값**을 다음과 같이 구함:

$$\mathbf{o}_1 = \sum_j a_{1j} \cdot \mathbf{v}_j$$

- 즉, attention이 큰 (연관성이 높은) token의 value에 가중치를 더 높게 부여해서 value에 대한 weighted sum을 구하는 것!
- attention은 일종의 value에 대한 “weight”라고 볼 수 있다!



ACADENTIAL

16-2. Vectorize된 Self-attention

Attention

Vectorize: query, key, value

- “Vectorize”이란?:
  - (속된말로…) “for loop”을 사용해서 iterative하게 일일이 계산하는 것이 아니라 Vector와 Matrix의 곱으로 표현하는 것!
  - 효과: Matrix의 곱은 GPU 상에서 병렬적으로 계산할 수 있어서 “for loop”으로 계산하는 것보다 효율적이고 더 빠르다!
  - 그렇다면 어떻게 $\mathbf{q}_i = W_q\mathbf{x}_i,$ $\mathbf{k}_i = W_k\mathbf{x}_i,$ $\mathbf{v}_i = W_v\mathbf{x}_i$ 의 계산을 vectorize할까?

ACADENTIAL

Attention

Vectorize: query, key, value

- “Vectorize”이란?:
  - (속된말로…) “for loop”을 사용해서 iterative하게 일일이 계산하는 것이 아니라 Vector와 Matrix의 곱으로 표현하는 것!
  - 효과: Matrix의 곱은 GPU 상에서 병렬적으로 계산할 수 있어서 “for loop”으로 계산하는 것보다 효율적이고 더 빠르다!
  - 그렇다면 어떻게 $\mathbf{q}_i = W_q \mathbf{x}_i,$ $\mathbf{k}_i = W_k \mathbf{x}_i,$ $\mathbf{v}_i = W_v \mathbf{x}_i$ 의 계산을 vectorize할까?

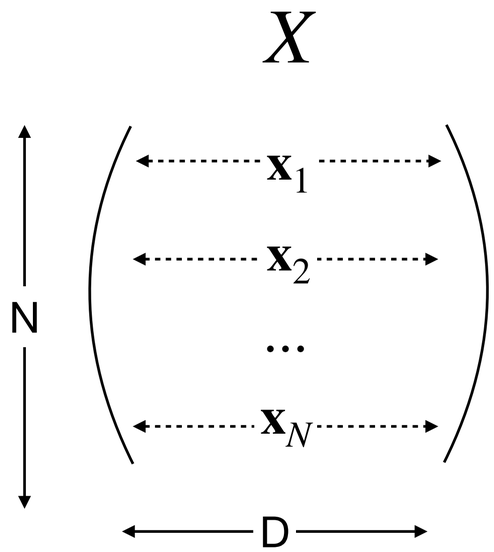
정답: $\mathbf{x}_i$을 row-wise stack해서 matrix X을 만드는 것! ($\mathbf{x}_i \rightarrow X$)

ACADENTIAL

Attention

Vectorize: query, key, value

1. $\mathbf{x}_i$을 row-wise stack해서 matrix X을 만든다



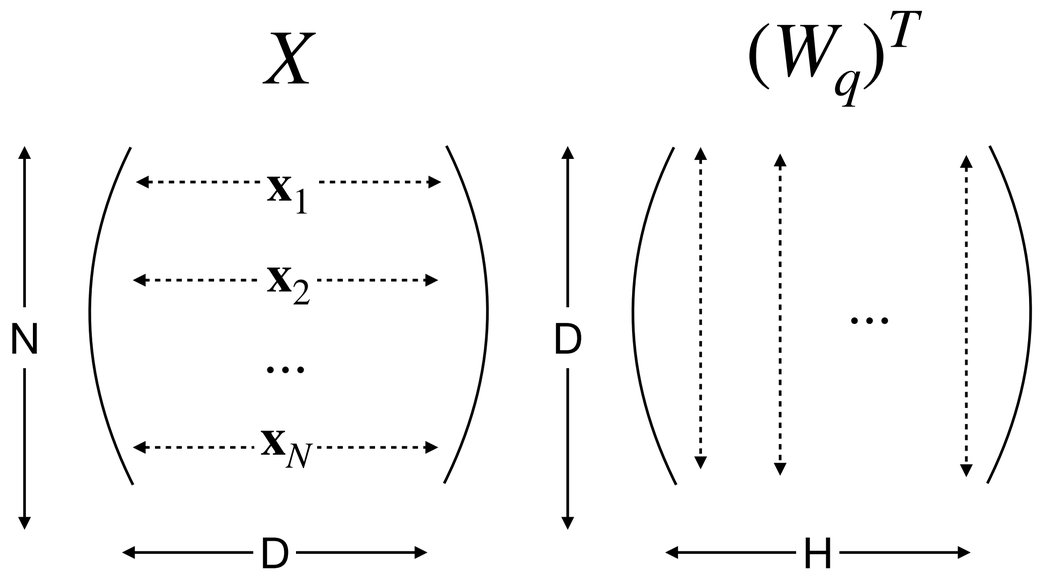
ACADENTIAL

Attention

Vectorize: query, key, value

1. $\mathbf{x}_i$을 row-wise stack해서 matrix X을 만든다

2. Query Weight을 traspose하여 X와 곱해준다



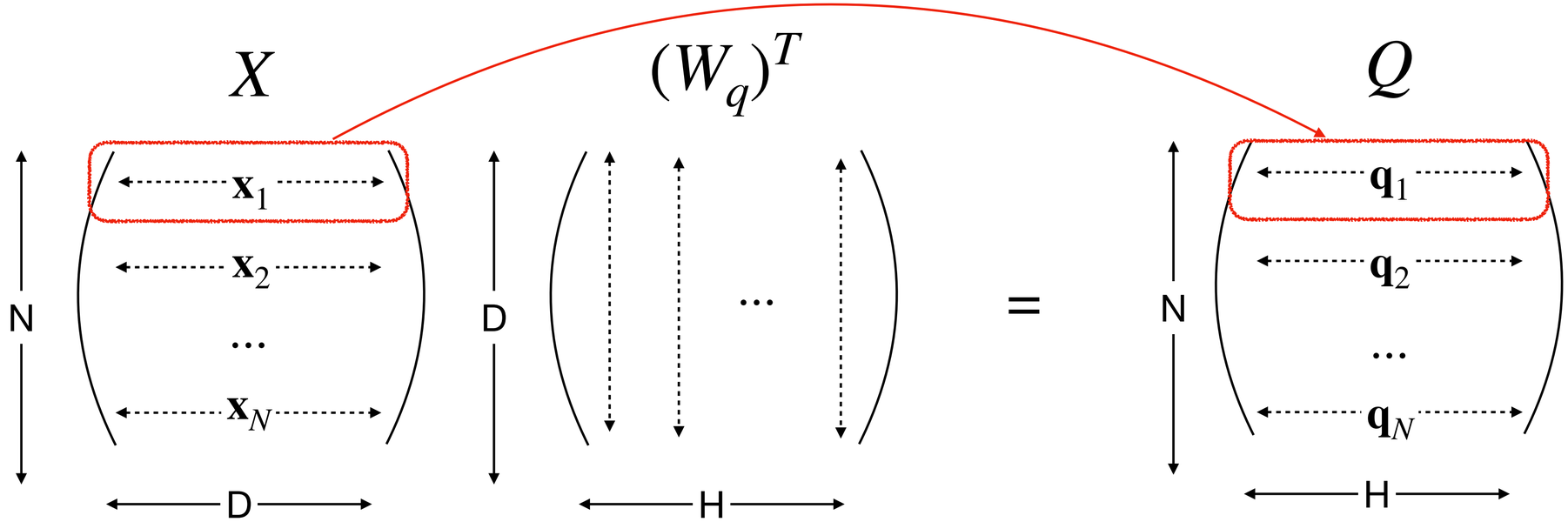
Attention

Vectorize: query, key, value

1. $\mathbf{x}_i$을 row-wise stack해서 matrix X을 만든다

2. Query Weight을 traspose하여 X와 곱해준다. 곱해져서 나온 행렬이 곧 Query matrix이다.

3. 이때 Query matrix의 i 번째 row가 곧 $\mathbf{q}_i$가 된다!



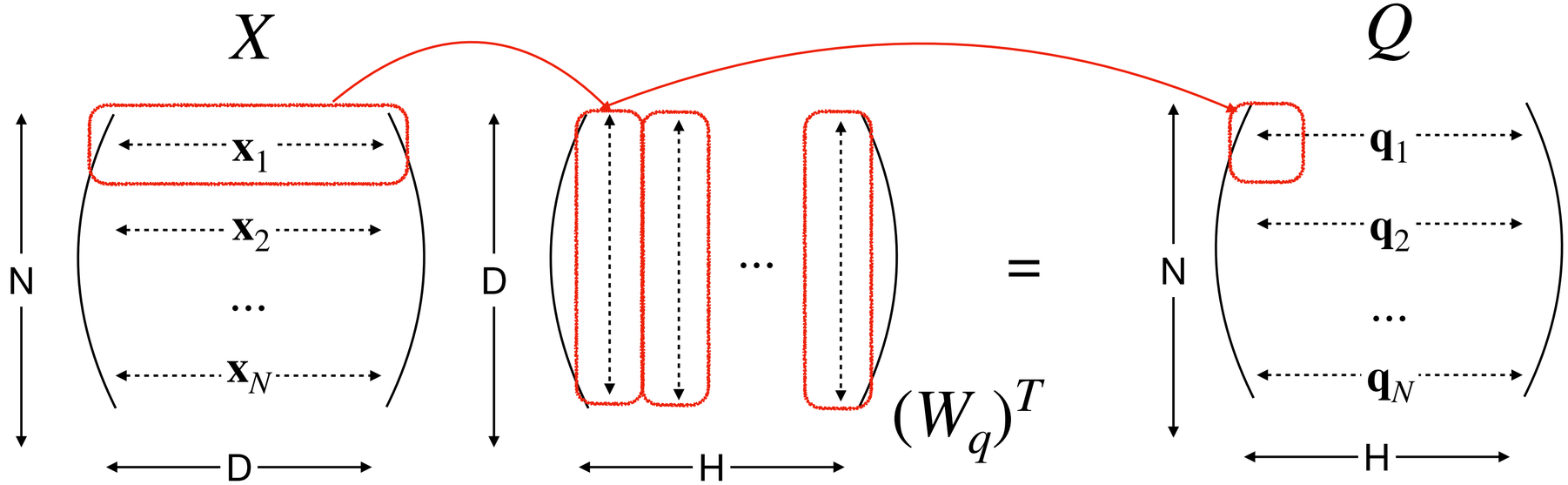
Attention

Vectorize: query, key, value

1. $\mathbf{x}_i$을 row-wise stack해서 matrix X을 만든다

2. Query Weight을 traspose하여 X와 곱해준다. 곱해져서 나온 행렬이 곧 Query matrix이다.

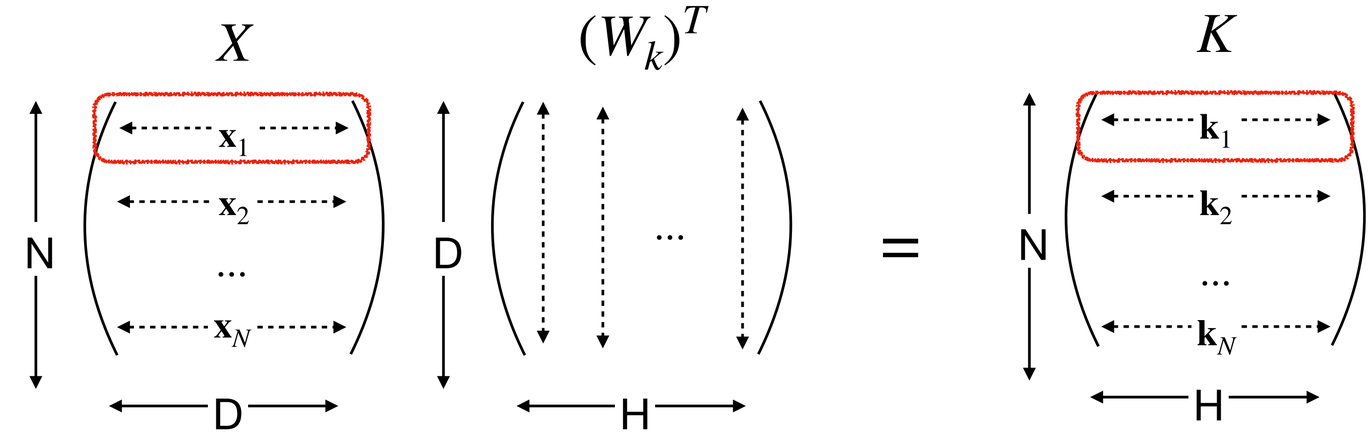
3. 이때 Query matrix의 i 번째 row가 곧 $\mathbf{q}_i$가 된다!

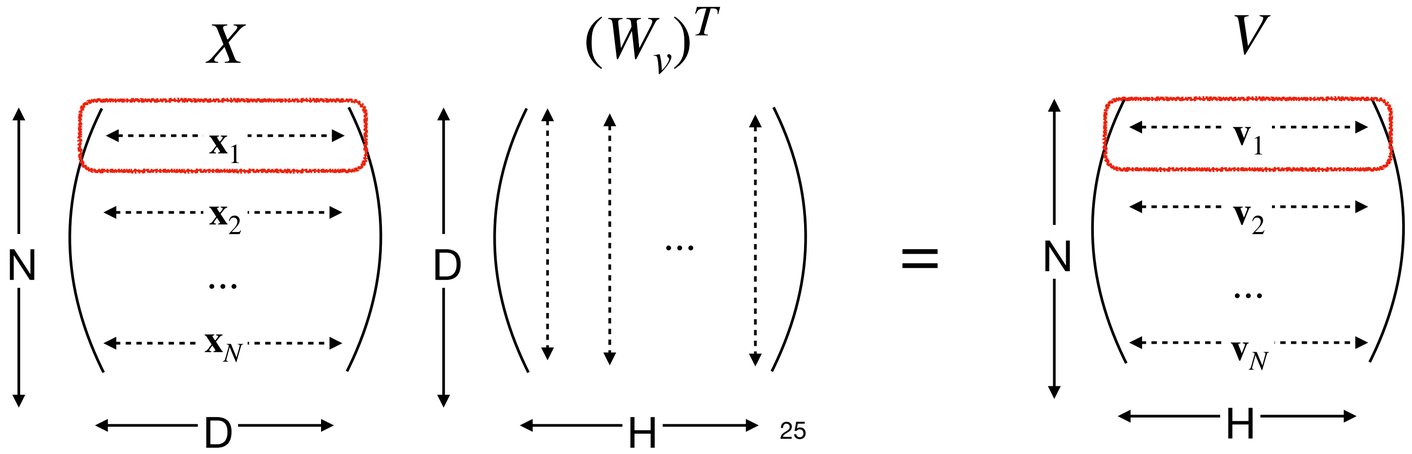


Attention

Vectorize: query, key, value

Key와 Value에 대해서도 동일하게 vectorize해서 구할 수 있다!





Attention

Vectorize: query, key, value

정리해보면:

$$Q = X(W_q)^T$$

$$K = X(W_k)^T$$

$$V = X(W_v)^T$$

where:

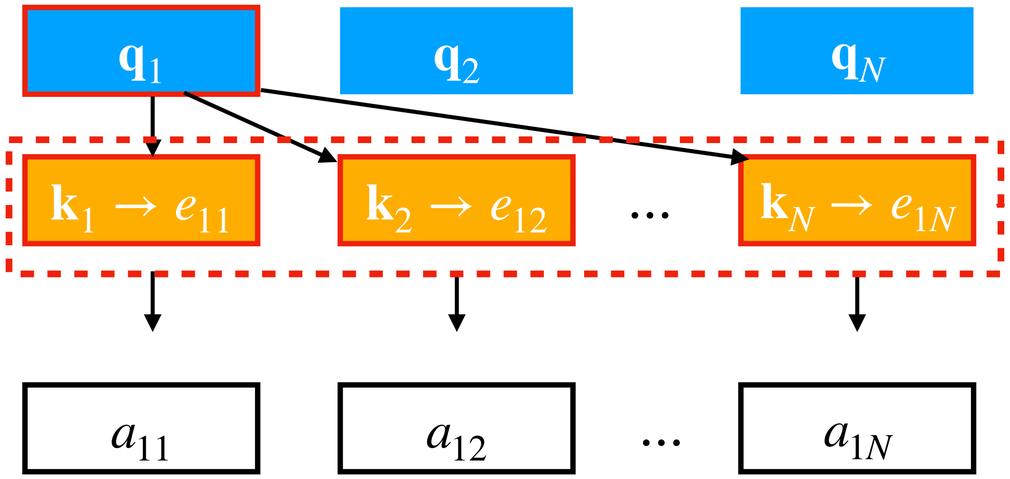
$$X \in \mathbb{R}^{N \times D}$$

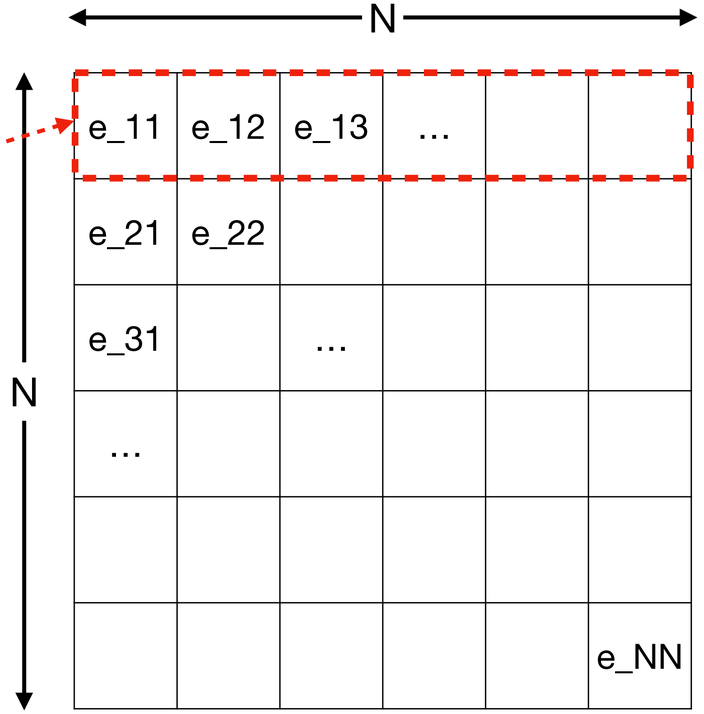
$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

Attention

Vectorize attention (attention map)



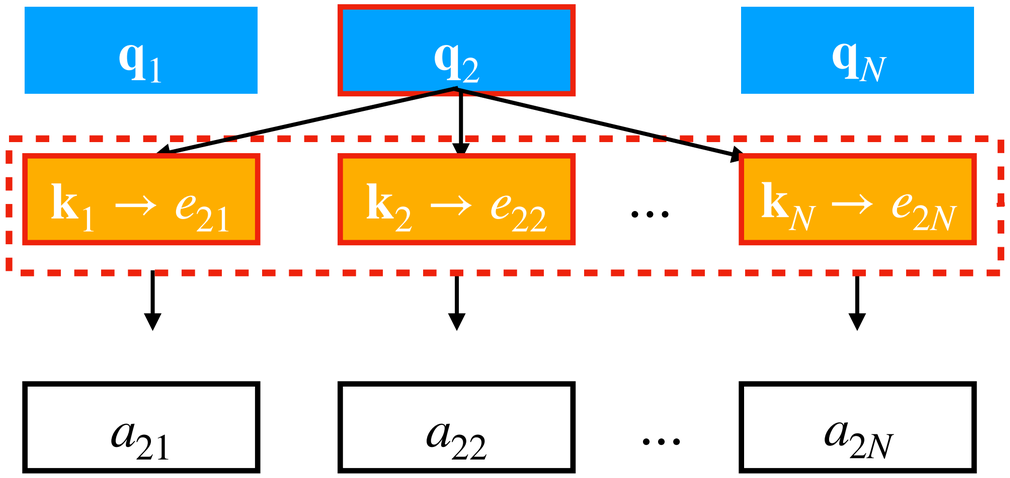


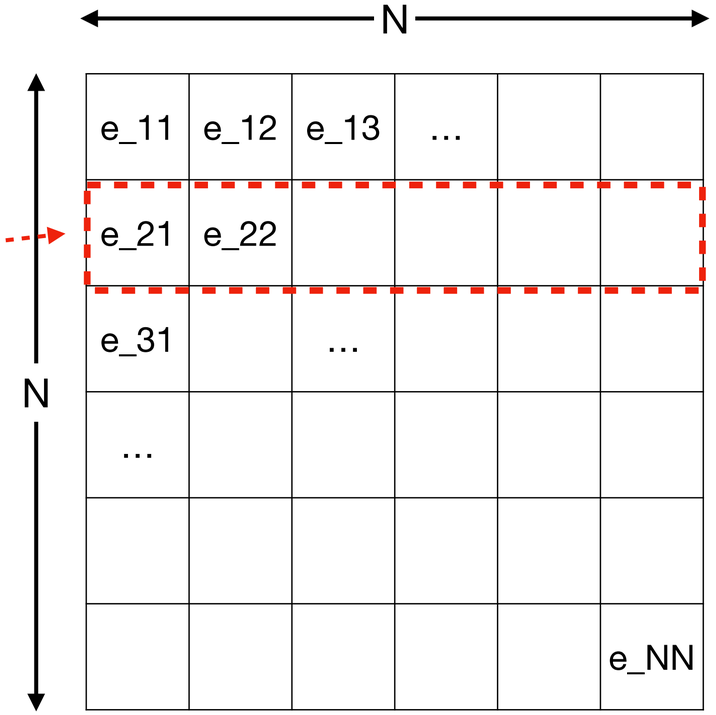
Attention Score Matrix E

ACADENTIAL

Attention

Vectorize attention (attention map)

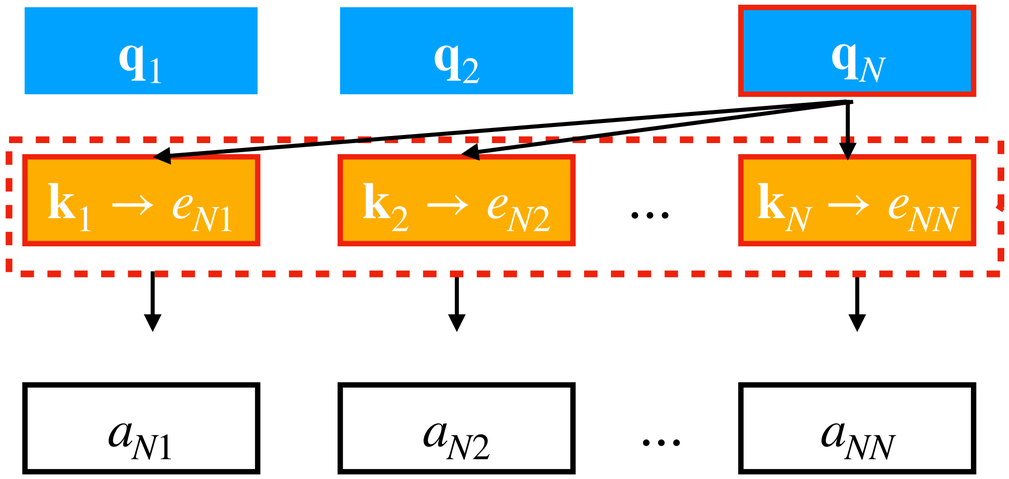


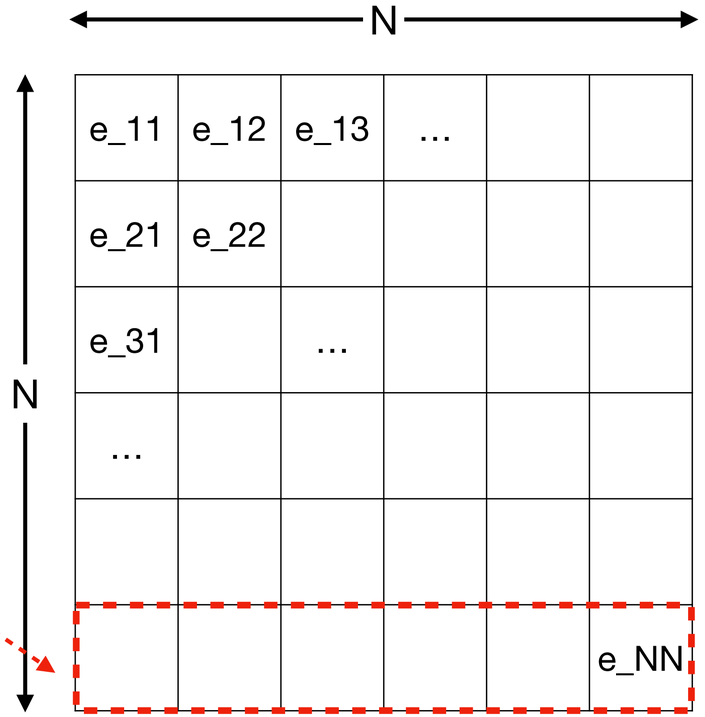


Attention Score Matrix E

Attention

Vectorize attention (attention map)



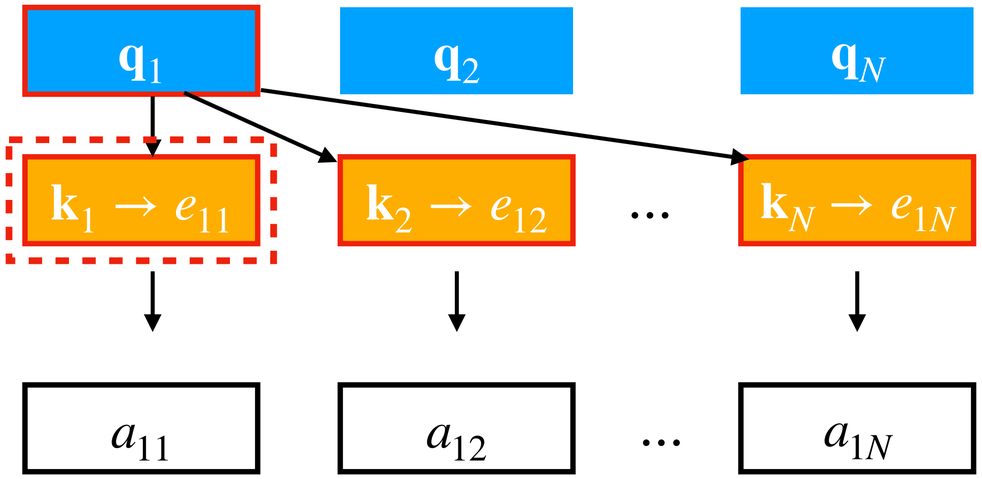


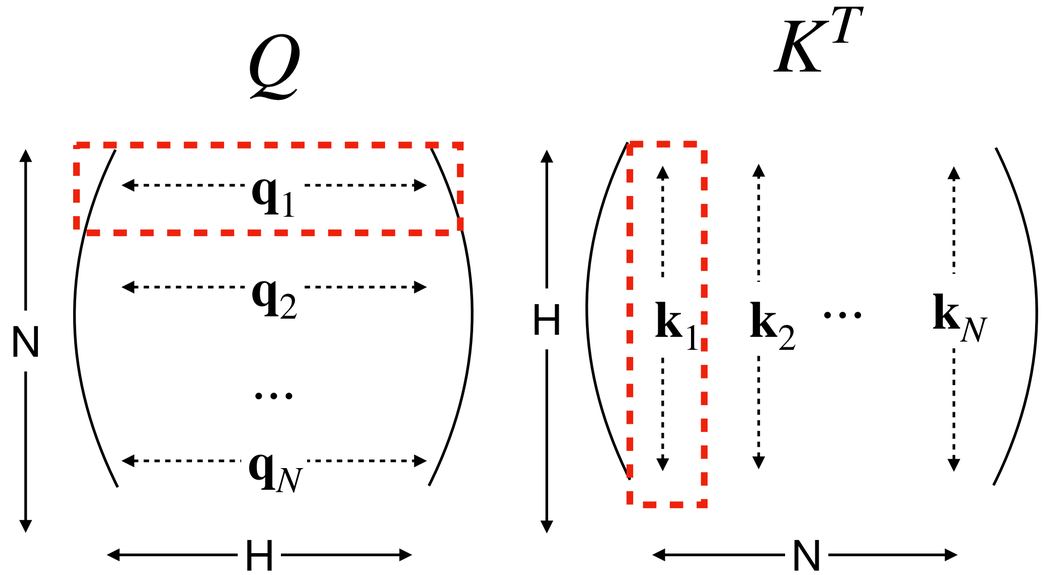
Attention Score Matrix E

ACADENTIAL

Attention

Vectorize attention (attention map)





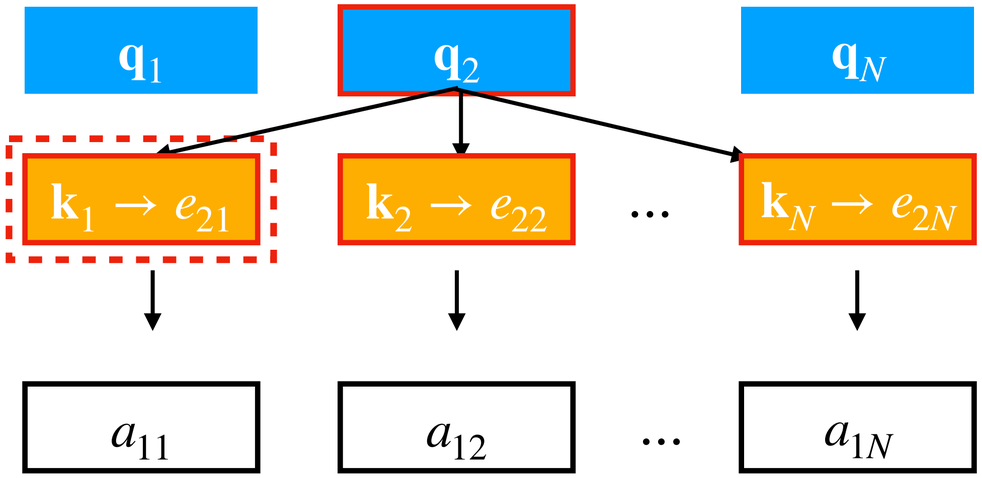
$$E = Q(K)^T$$

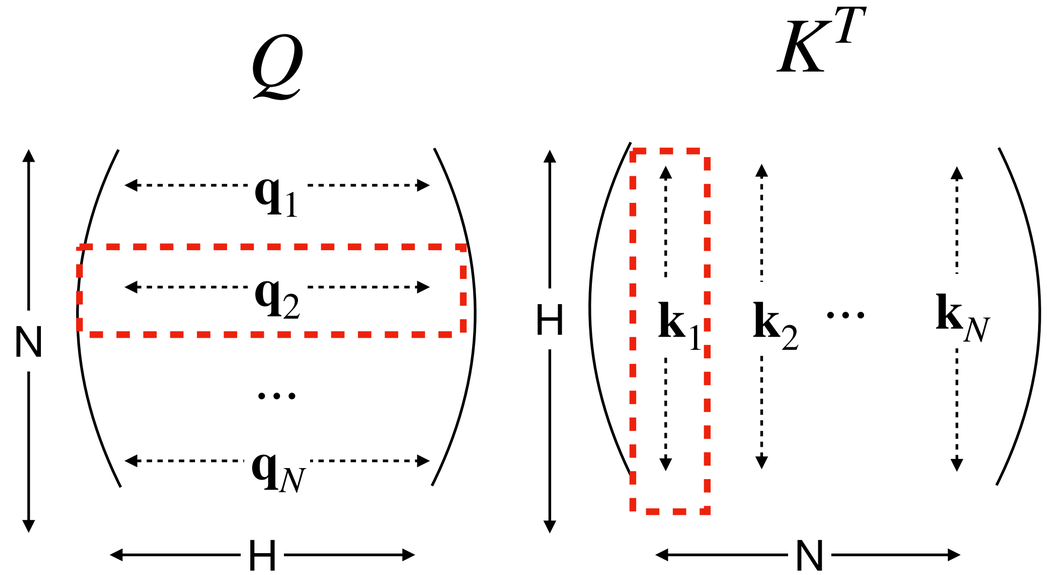
$$= \begin{pmatrix} e_{11} & \cdots \\ \cdots & \\ & E \end{pmatrix}$$

ACADENTIAL

Attention

Vectorize attention (attention map)





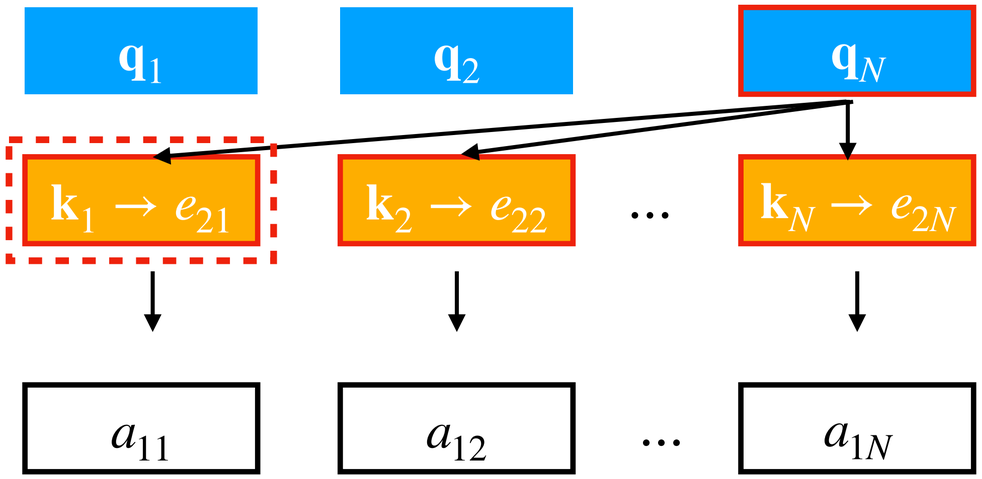
$$E = Q(K)^T$$

$$= \begin{pmatrix} e_{11} & \cdots \\ e_{21} & \cdots \\ & E \end{pmatrix}$$

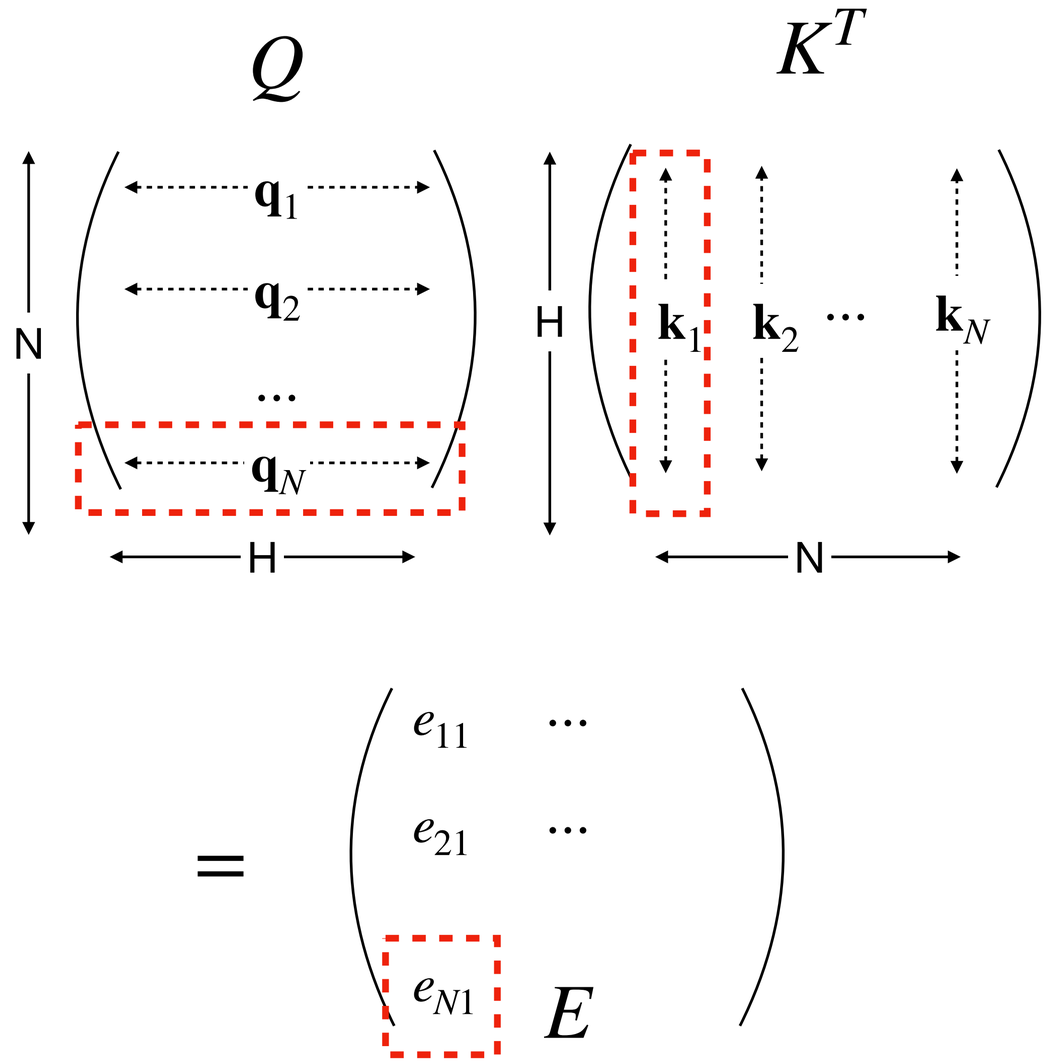
ACADENTIAL

Attention

Vectorize attention (attention map)



$$E = Q(K)^T$$



Attention

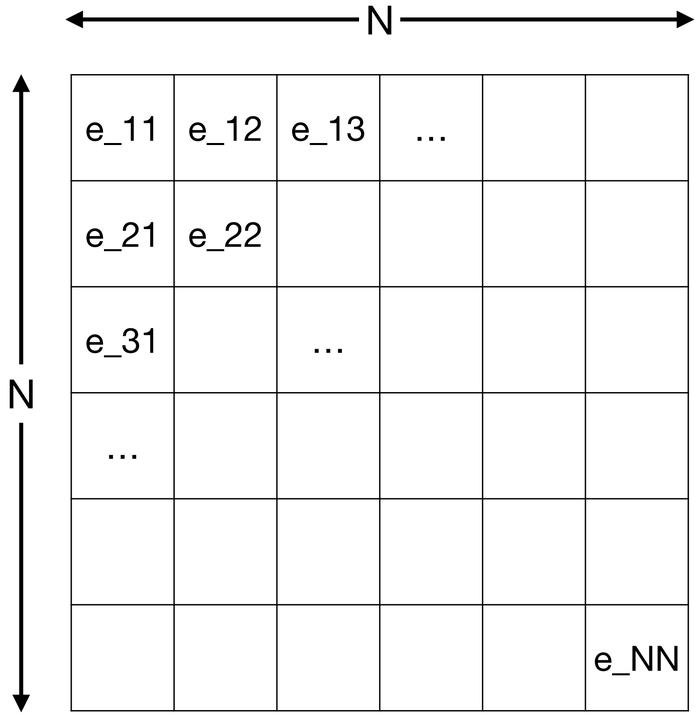
Vectorize attention (attention map)

Attention score의 계산을 vectorize해보면:

$$E = QK^T$$

Attention weight의 값은:

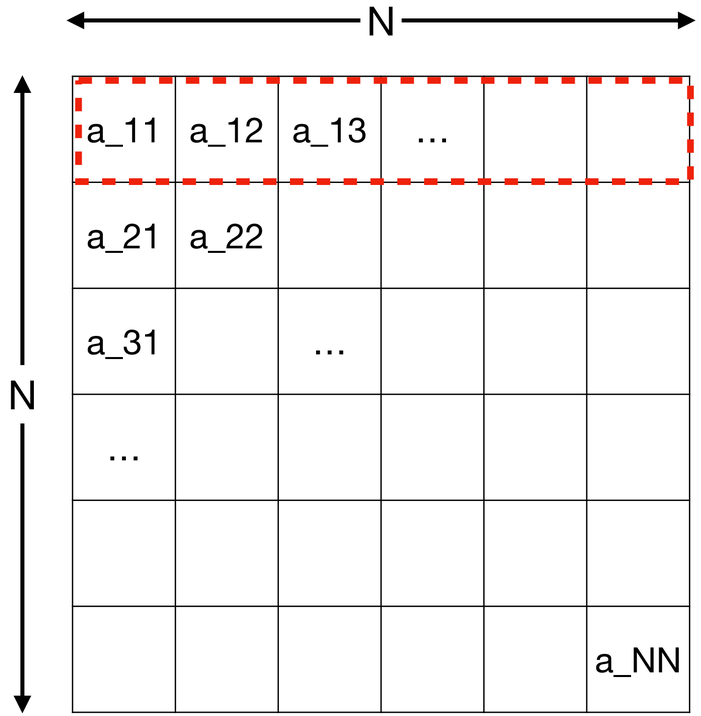
$$A = \text{softmax}(E)$$



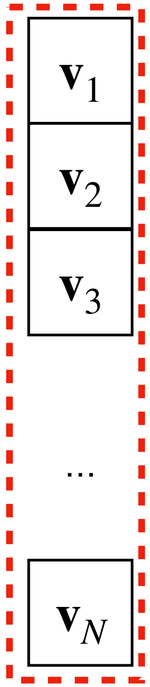
Attention Score Matrix E

Attention

Vectorize output calculation



Attention Weight Matrix A



Value Matrix V

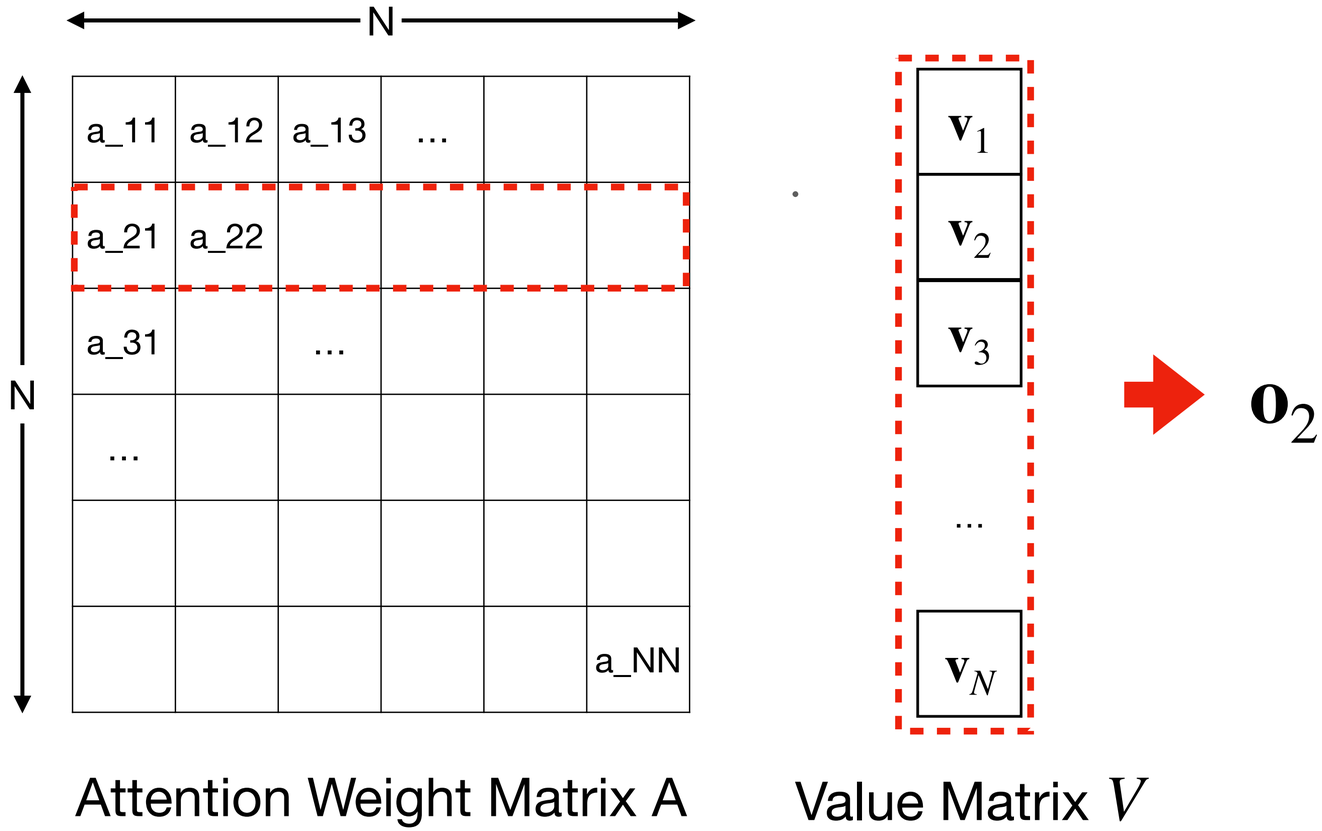
$\mathbf{o}_1$

- Output값:

$$\mathbf{o}_i = \sum_j a_{ij} \cdot \mathbf{v}_j$$

Attention

Vectorize output calculation



Attention Weight Matrix A

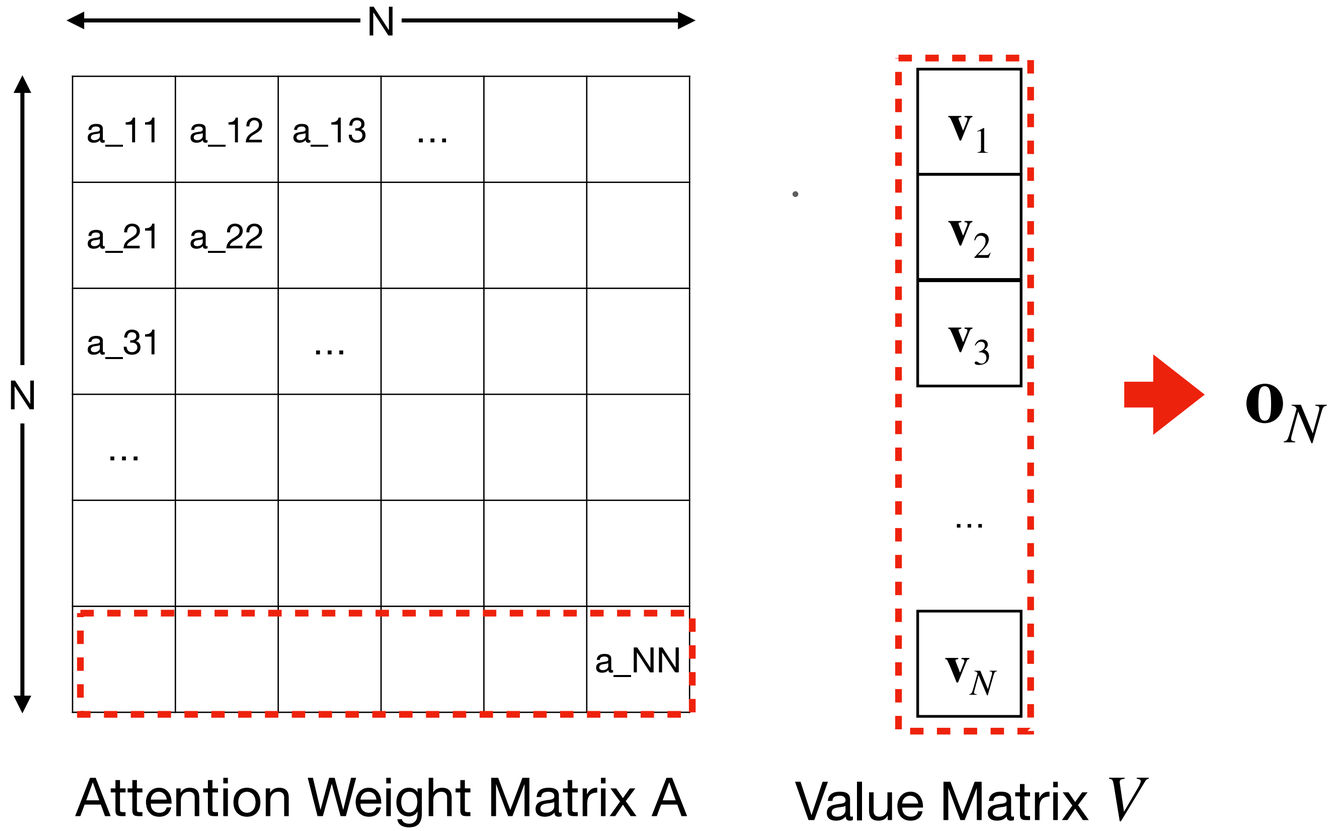
Value Matrix V

- Output값:

$$\mathbf{o}_i = \sum_j a_{ij} \cdot \mathbf{v}_j$$

Attention

Vectorize output calculation



Attention Weight Matrix A

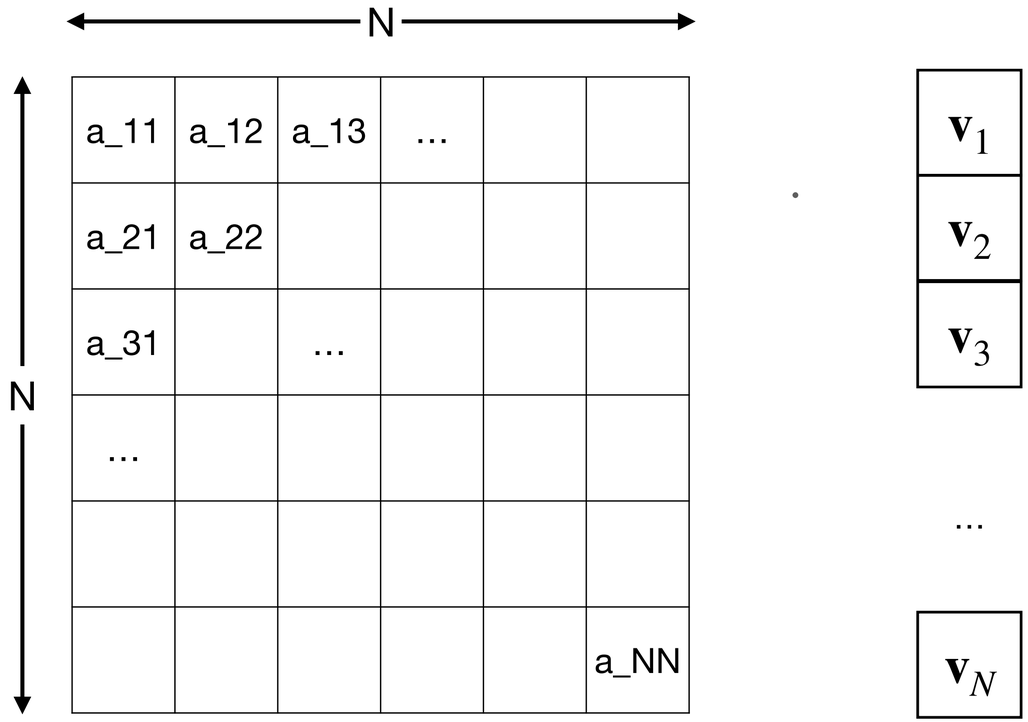
Value Matrix V

- Output값:

$$\mathbf{o}_i = \sum_j a_{ij} \cdot \mathbf{v}_j$$

Attention

Vectorize output calculation



Attention Weight Matrix A

Value Matrix V

- Output값:

$$\mathbf{o}_i = \sum_j a_{ij} \cdot \mathbf{v}_j$$

- 위 식을 Attention Weight Matrix A 행렬과 Value Matrix V 행렬간의 곱으로 표현할 수 있다!

Output Matrix $O = AV$

Attention

Self-Attention summary

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV$$

- 즉,

$$O = \text{softmax}(QK^T)V$$

ACADENTIAL

Attention

Self-Attention summary

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T$$

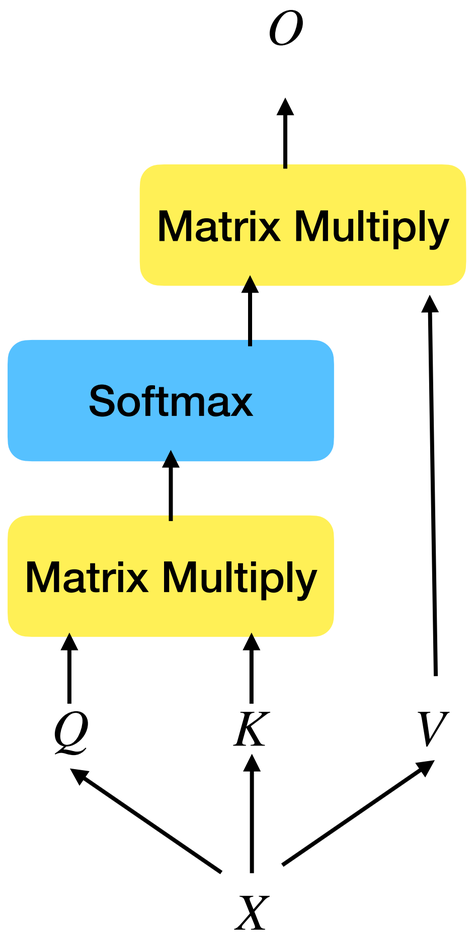
- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV$$

$$O = \text{softmax}(QK^T)V$$



Attention

Attention의 발전 과정

- Attention을 사용하는 모델로 Transformer, BERT 등이 있다.
- Transformer와 BERT 둘은 어떤 차이인가?
- Attention과 Self-Attention은 서로 어떤 차이인가?
- 이것들을 이해하기 위한 맥락으로 Attention의 발전 과정을 살펴보자!

ACADENTIAL

16-3. Attention의 발전 과정

Neural Machine Translation by Jointly Learning to Align and Translate

Attention

Attention의 계보

- Attention이 가장 처음으로 제안된 논문은:

"Neural Machine Translation by Jointly Learning to Align and Translate"

(Bahdanau et al. ICLR 2015)

ACADENTIAL

Attention

Attention의 계보

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

Attention이 가장 처음으로 제안된 논문!

Attention

Attention의 계보

- Attention이 가장 처음으로 제안된 논문은:

  “Neural Machine Translation by Jointly Learning to Align and Translate”

  (Bahdanau et al. ICLR 2015)

- 특징은
  - RNN Encoder-Decoder 구조에서 Attention이 Decoder에 적용된 것
  - “Additive” attention이 사용됨
  - Machine Translation task에 적용됨.

ACADENTIAL

Attention

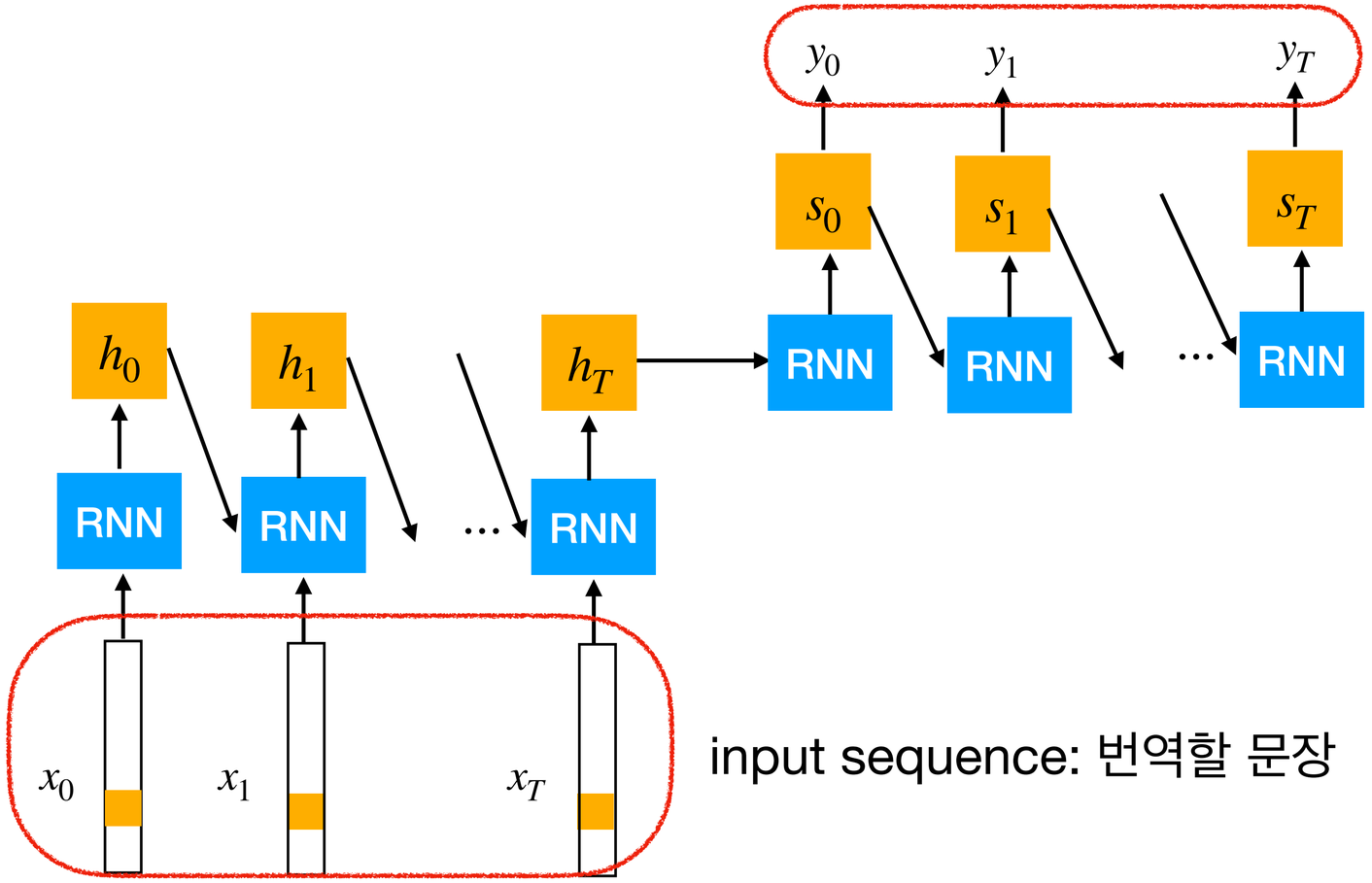
Neural Machine Translation by Jointly Learning to Align and Translate

- 먼저 기존에 Machine Translation에 흔히 사용되었던 RNN기반의 Encoder-decoder 구조를 살펴보자.

ACADENTIAL

Attention

일반적인 RNN 기반의 Encoder-Decoder 구조



target sequence: 번역된 문장

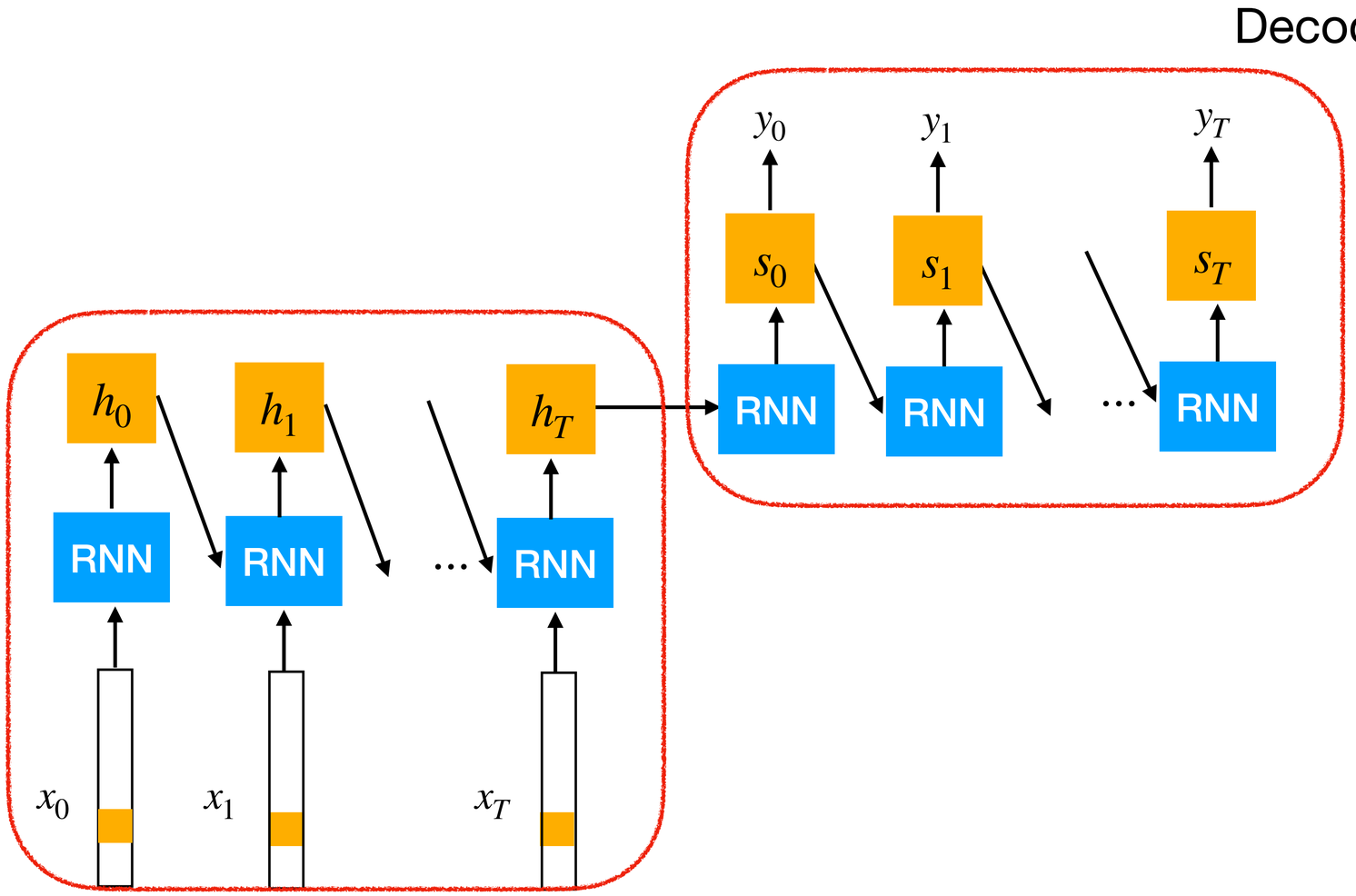
input sequence: 번역할 문장

Encoder란 인풋을 encoding하는 역할을 가진 모듈

Decoder란 encoding된 정보로부터 아웃풋을 출력해주는 역할을 가진 모듈

Attention

일반적인 RNN 기반의 Encoder-Decoder 구조



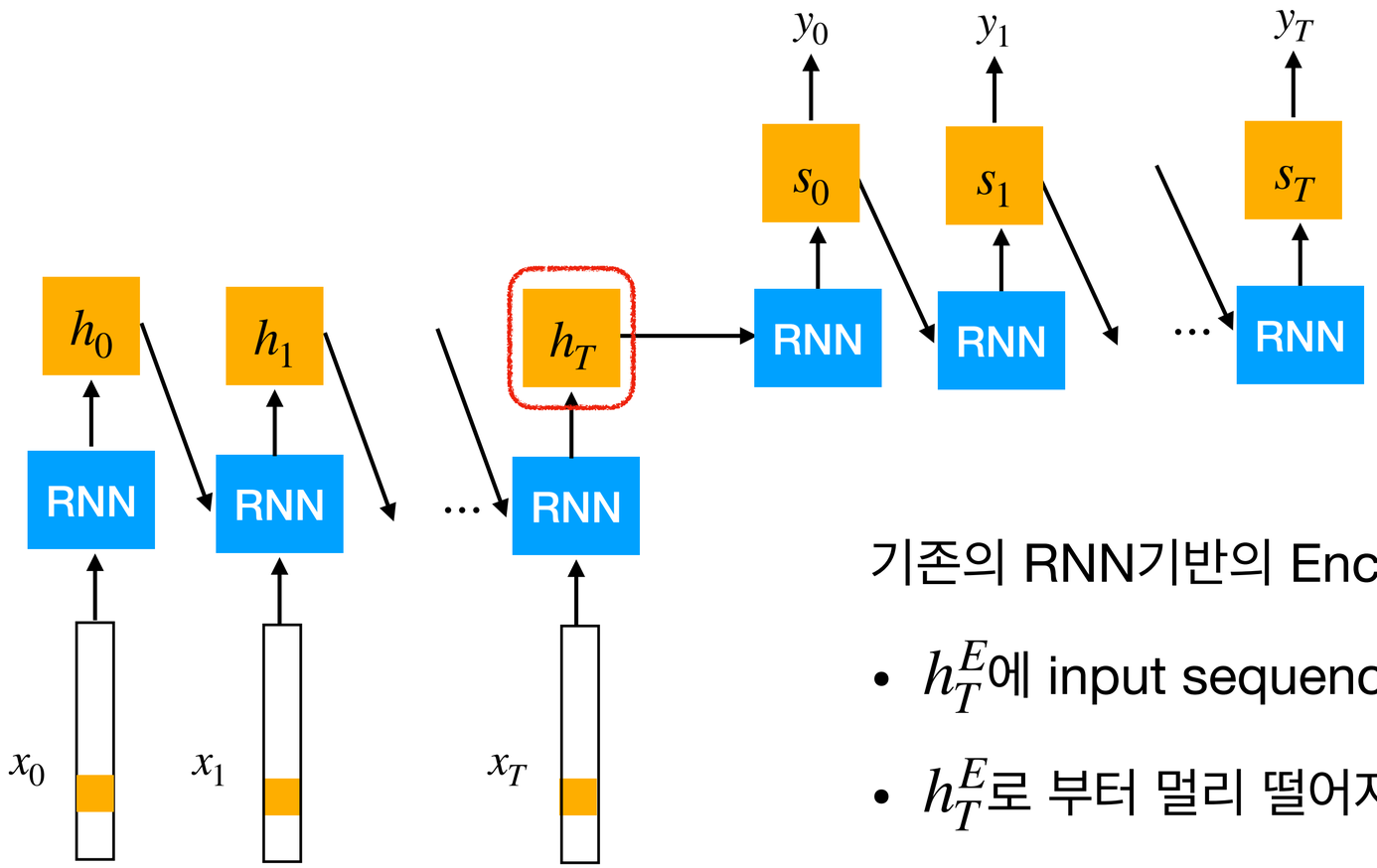
Encoder란 인풋을 encoding하는 역할을 가진 모듈

Decoder란 encoding된 정보로부터 아웃풋을 출력해주는 역할을 가진 모듈

ACADENTIAL

Attention

RNN 기반의 Encoder-Decoder 구조

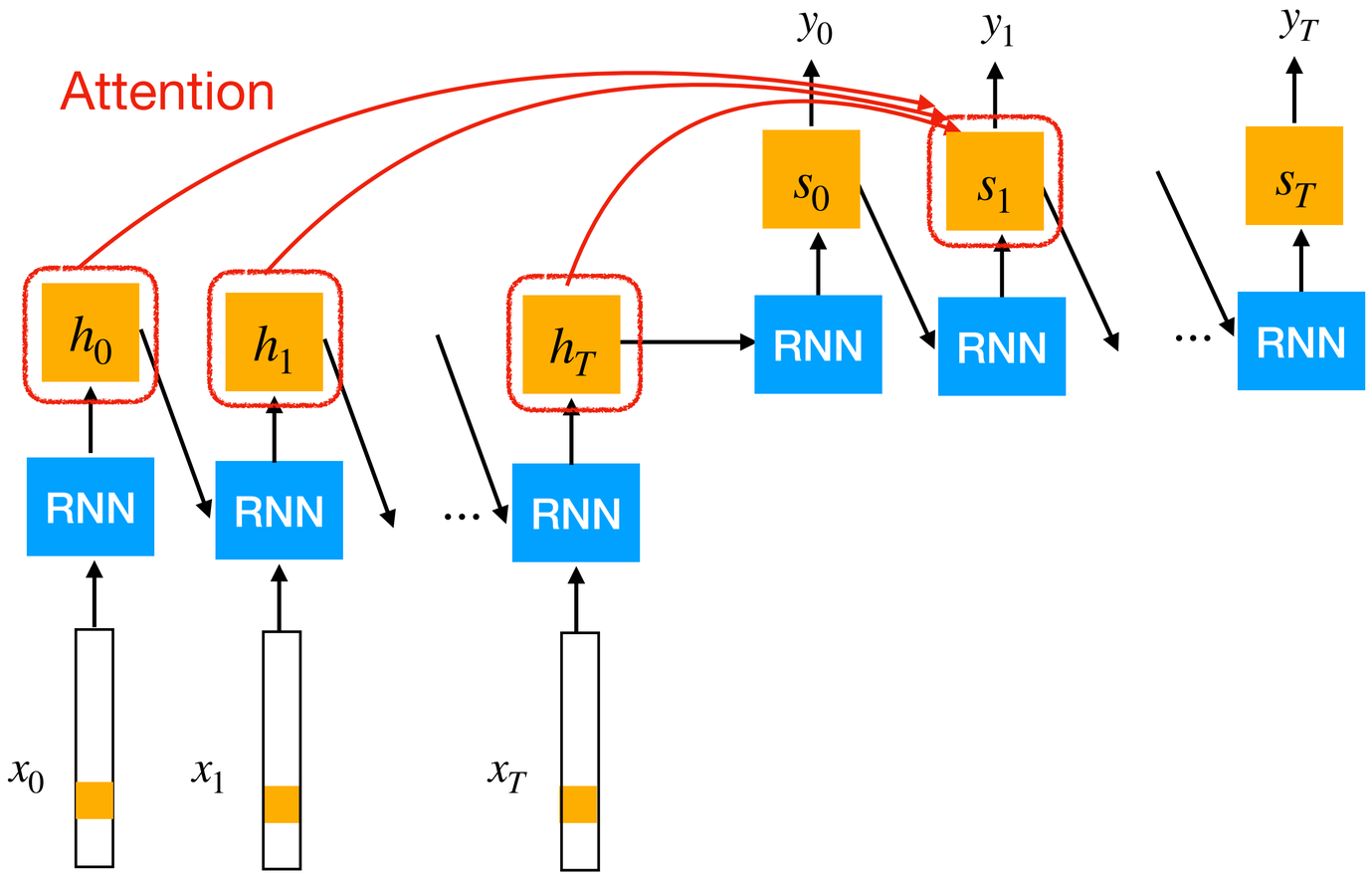


기존의 RNN기반의 Encoder-Decoder 구조의 문제점:

- h_T^E에 input sequence에 대한 정보를 모두 담아내야함.
- h_T^E로 부터 멀리 떨어져있는 input token의 정보는 “희석” 될 수 있다.

Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al 2015.에서 제안된 방법)



"희석"되는 문제점을 해소하기 위한 방안으로

Bahnadau et al 2015.에서는 **attention**을 처음으로 도입하였다!

이것을 더 구체적으로 살펴보자!

ACADENTIAL

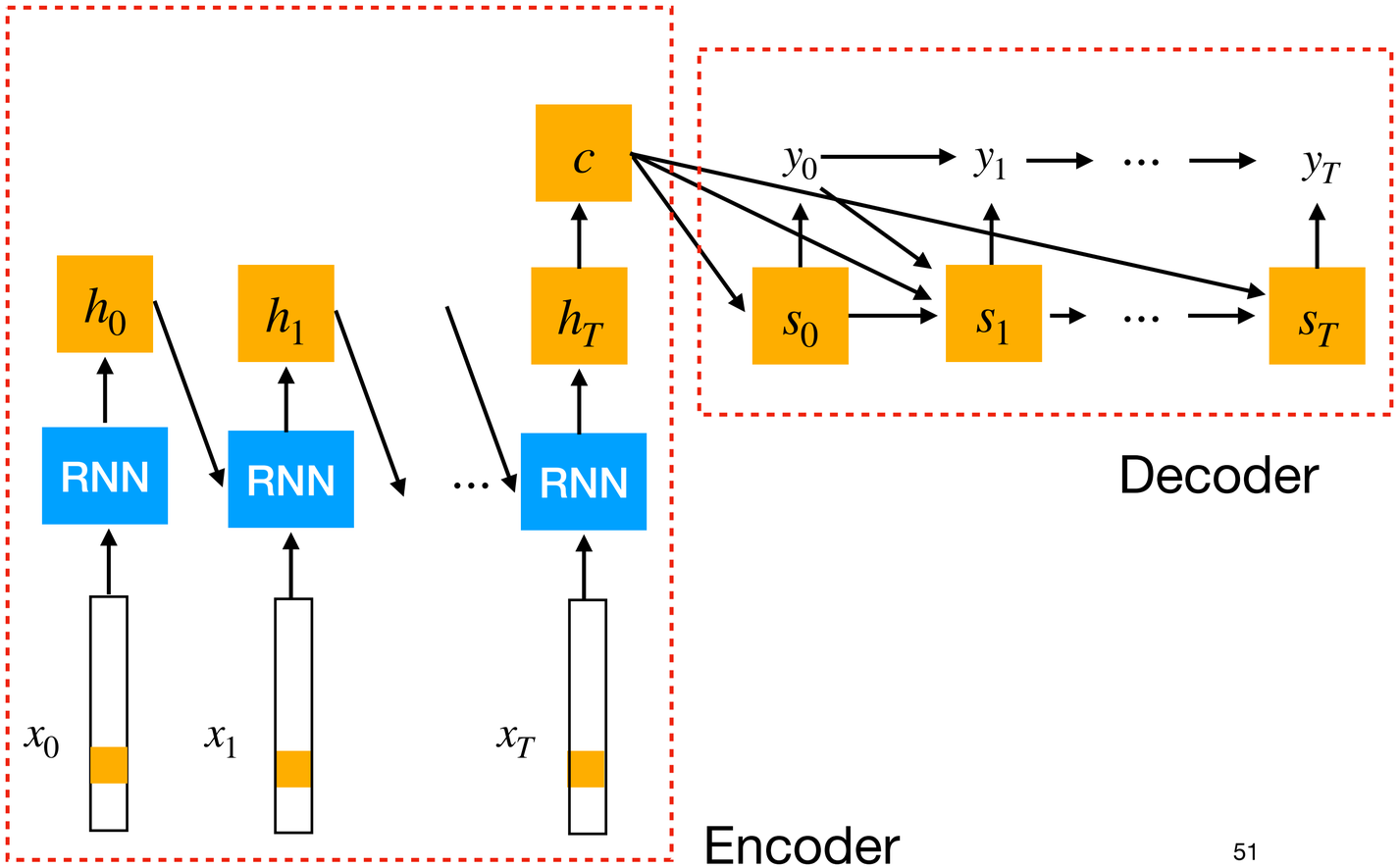
Attention

RNN 기반의 Encoder-Decoder 구조

- *“Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation”*에서 제안한 **RNN Encoder-Decoder 구조에 기반**하고 있다.
  - 일단, 이 논문의 **RNN Encoder-Decoder 구조**를 간략히 살펴보자.

Attention

Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation (Cho et al 2014)



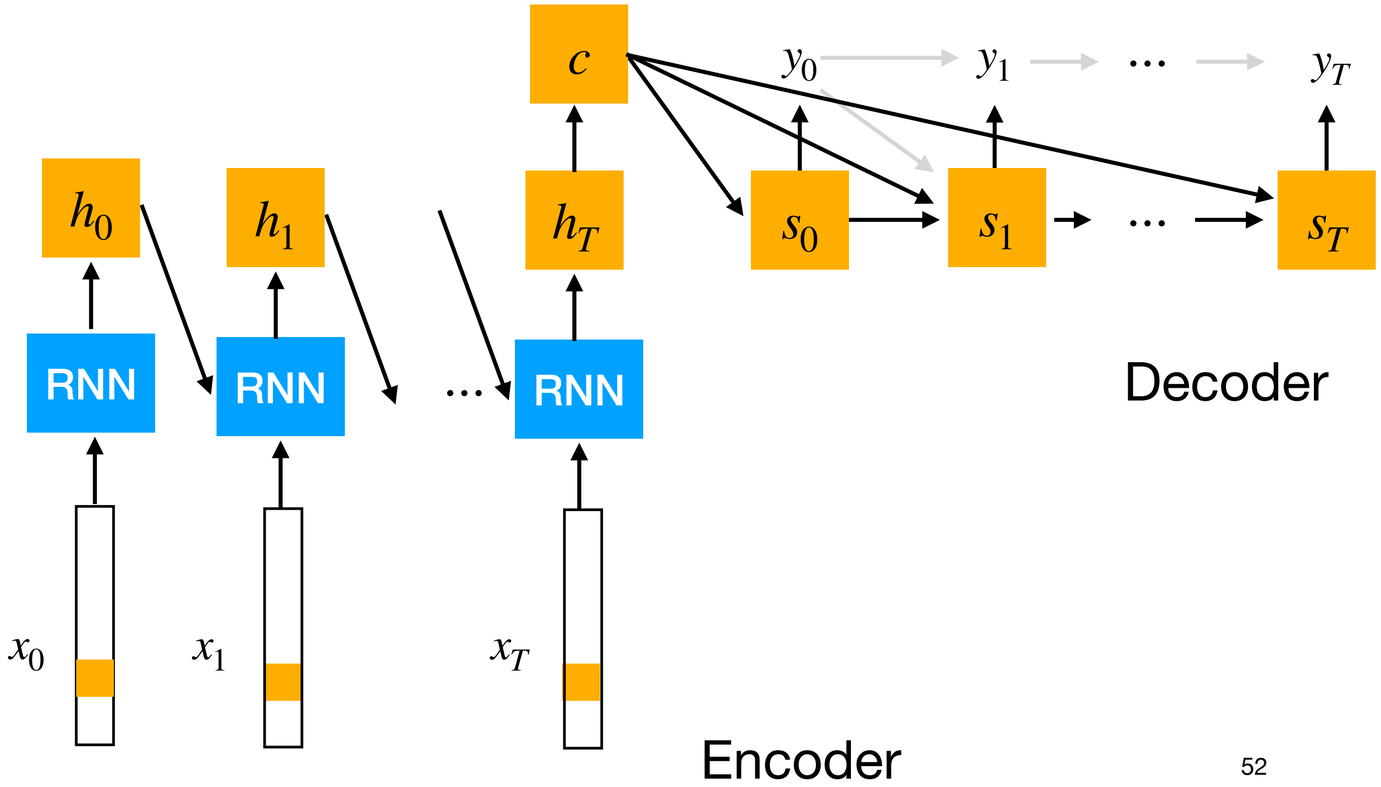
Decoder의 구조가 조금 복잡하다…

이해를 돕기 위해서 몇 가지 부분을 단순화해 보자.

Attention

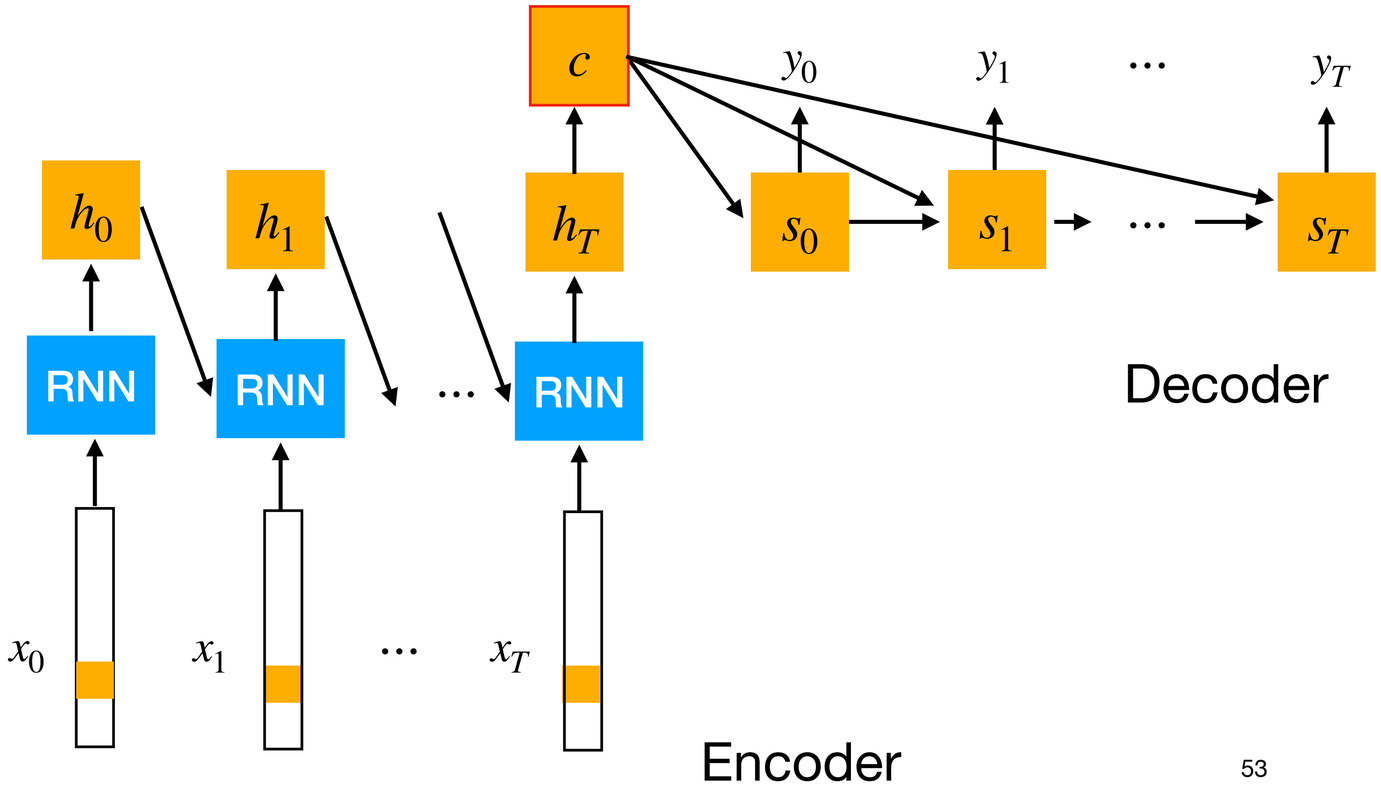
Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation (Cho et al 2014)

하얀색 화살표 생략



Attention

Learning Phrase Representation using RNN Encoder-Decoder for Statistical Machine Translation (Cho et al 2014)



context vector c에 $x_0, \ldots, x_T$ 의 정보들을 전부 포함시키려고 한다.

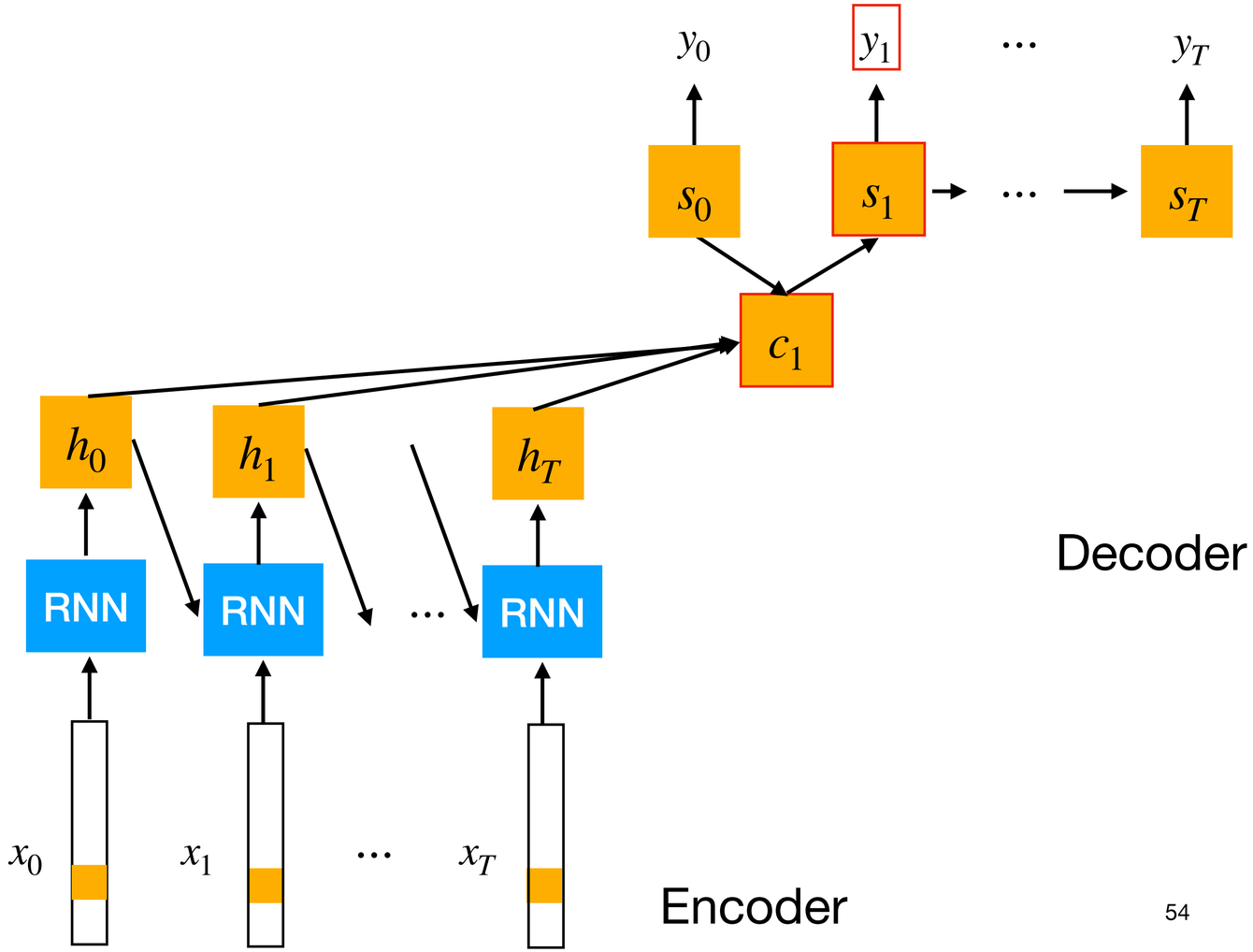
즉, $x_{t \ll T}$ 에 대해서는 정보가 “희석”된다.

그렇다면, (Bahnadau et al 2015) 어떻게 이 문제를 해결할 수 있었을까?

ACADENTIAL

Attention

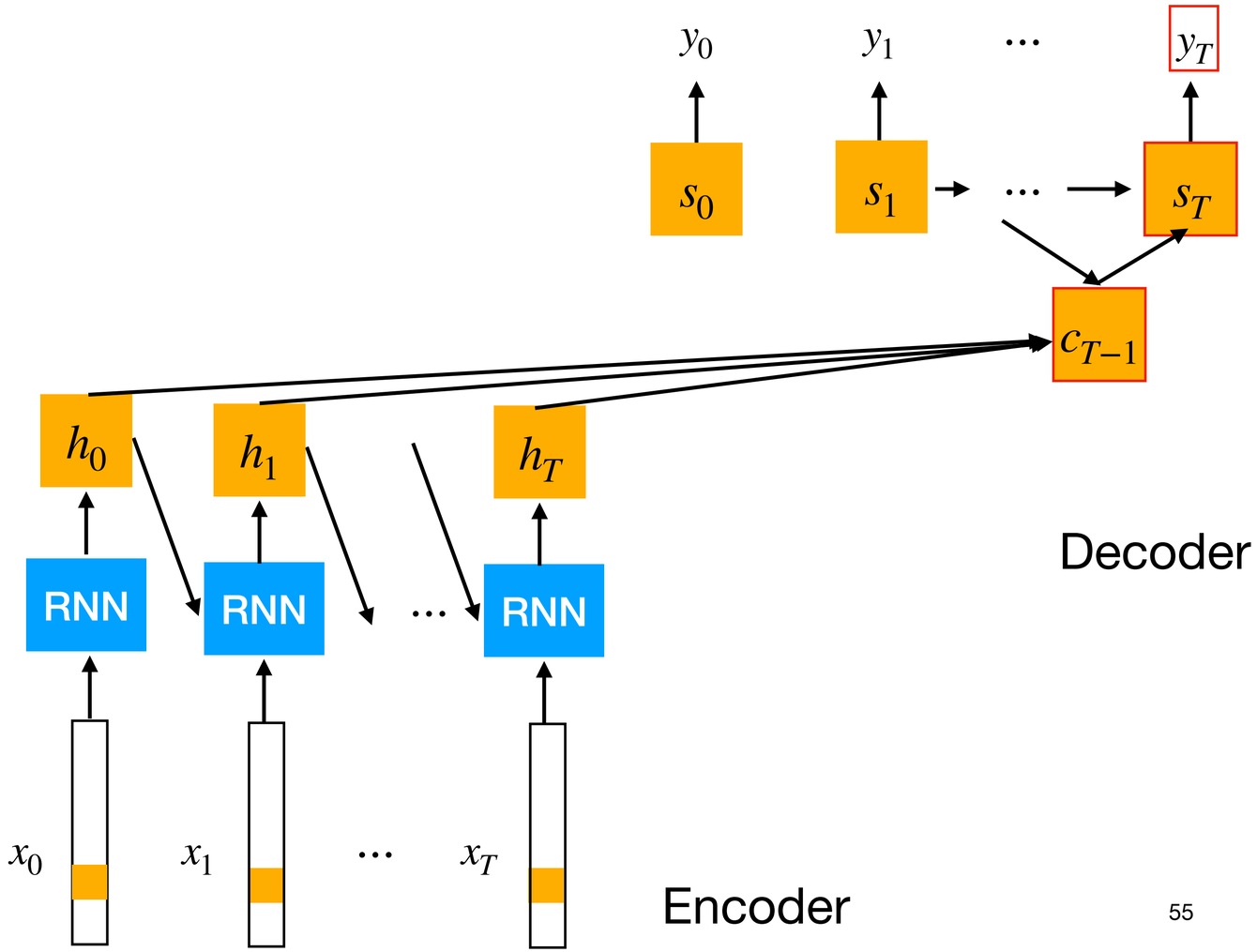
Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)



ACADENTIAL

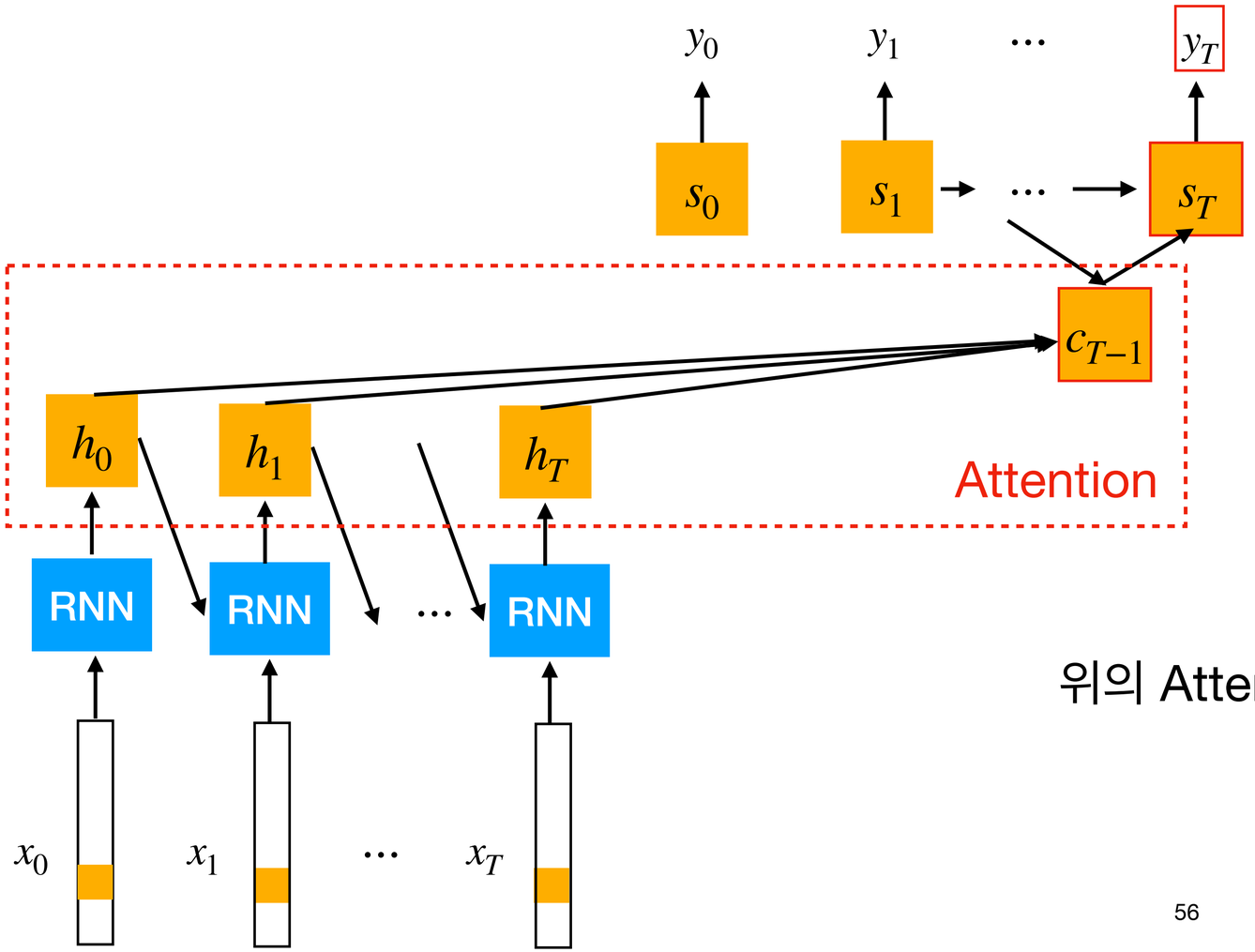
Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)



Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

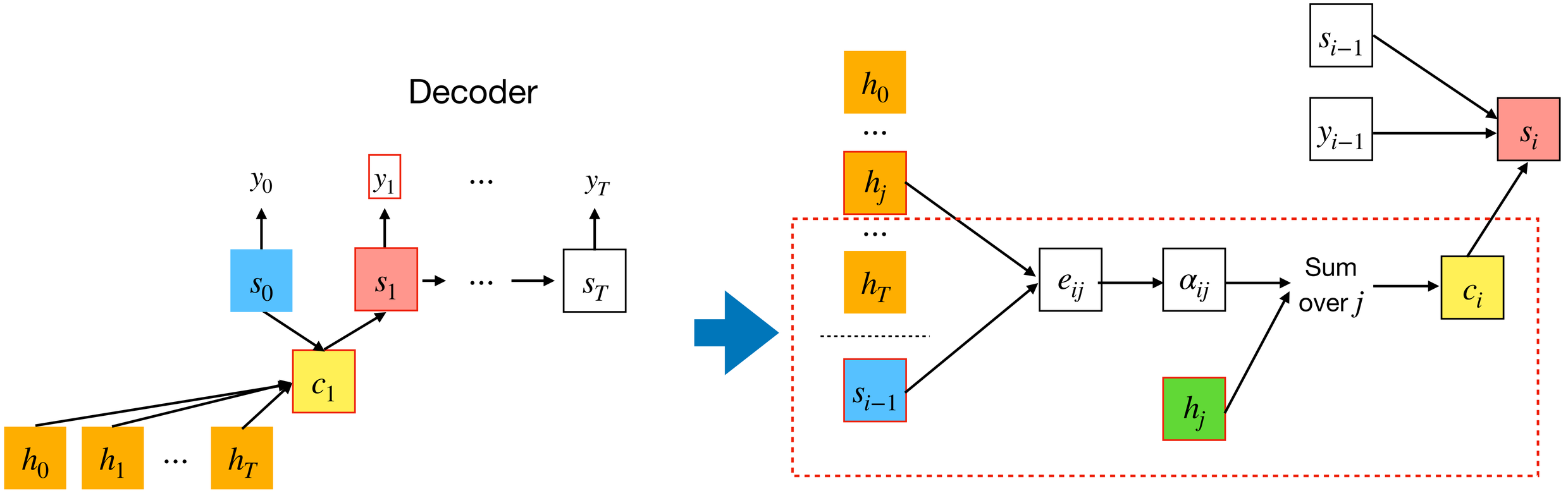


위의 Attention의 과정을 자세하게 살펴보자

ACADENTIAL

Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)



여기서 c_i은 일종의 "h_j 의 Attention-weighted 평균값"이다

(query: s_{i-1}, key: h_j, value: h_j)

Attention

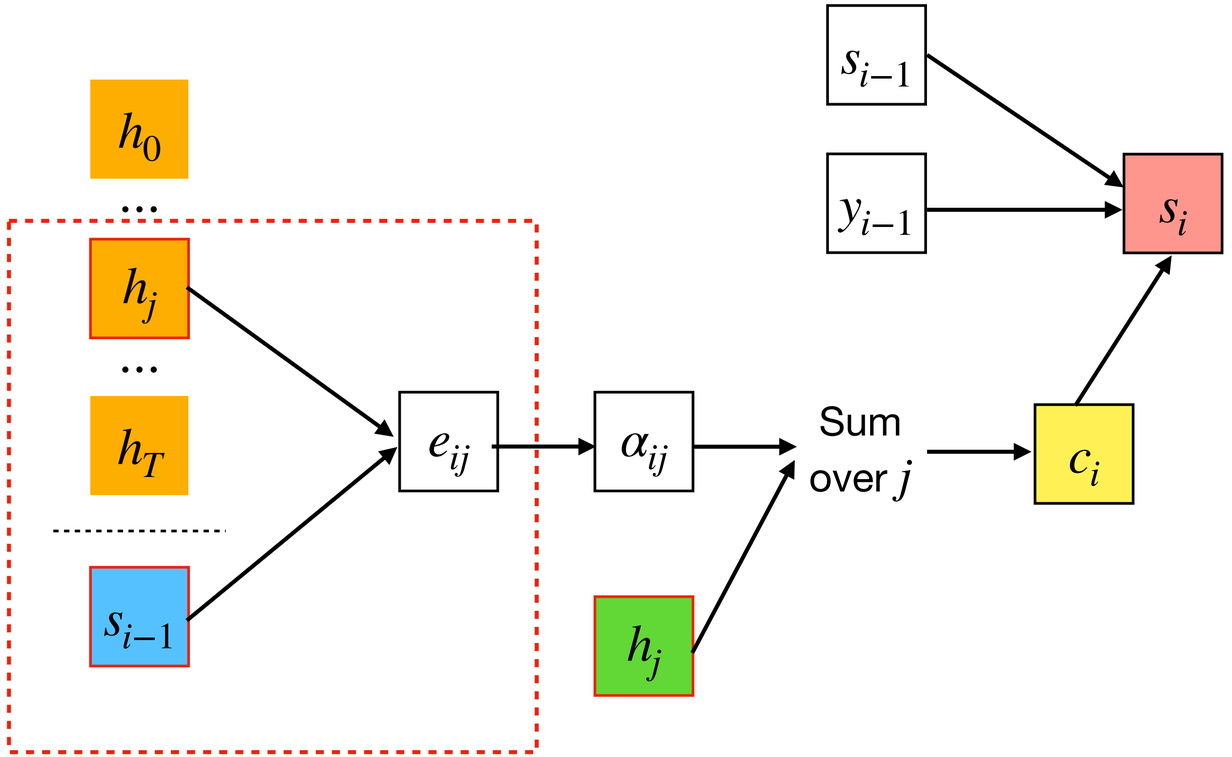
Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

Attention score은 다음과 같다:

$$e_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

참고로, 여기서는 ***additive attention***을 사용한다!

그리고 (query = s_{i-1}) 와 (key = h_j) 이다.

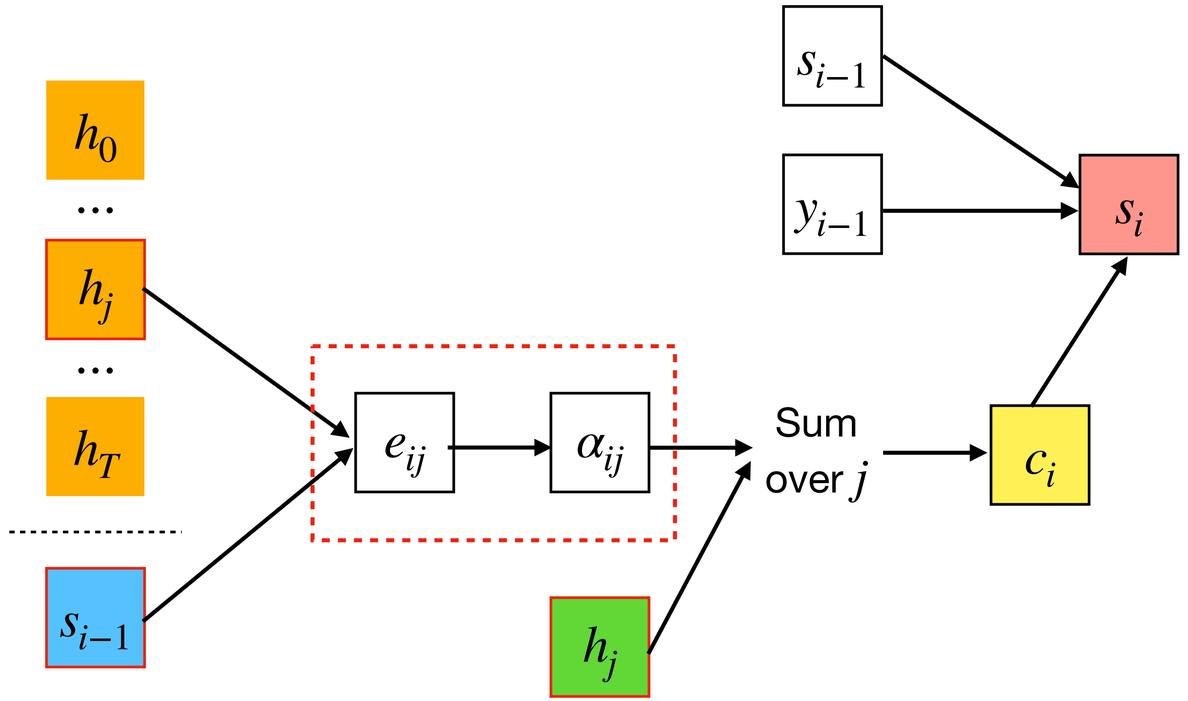


Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

Attention weight α_{ij}은 다음과 같다:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ij})}$$



Attention

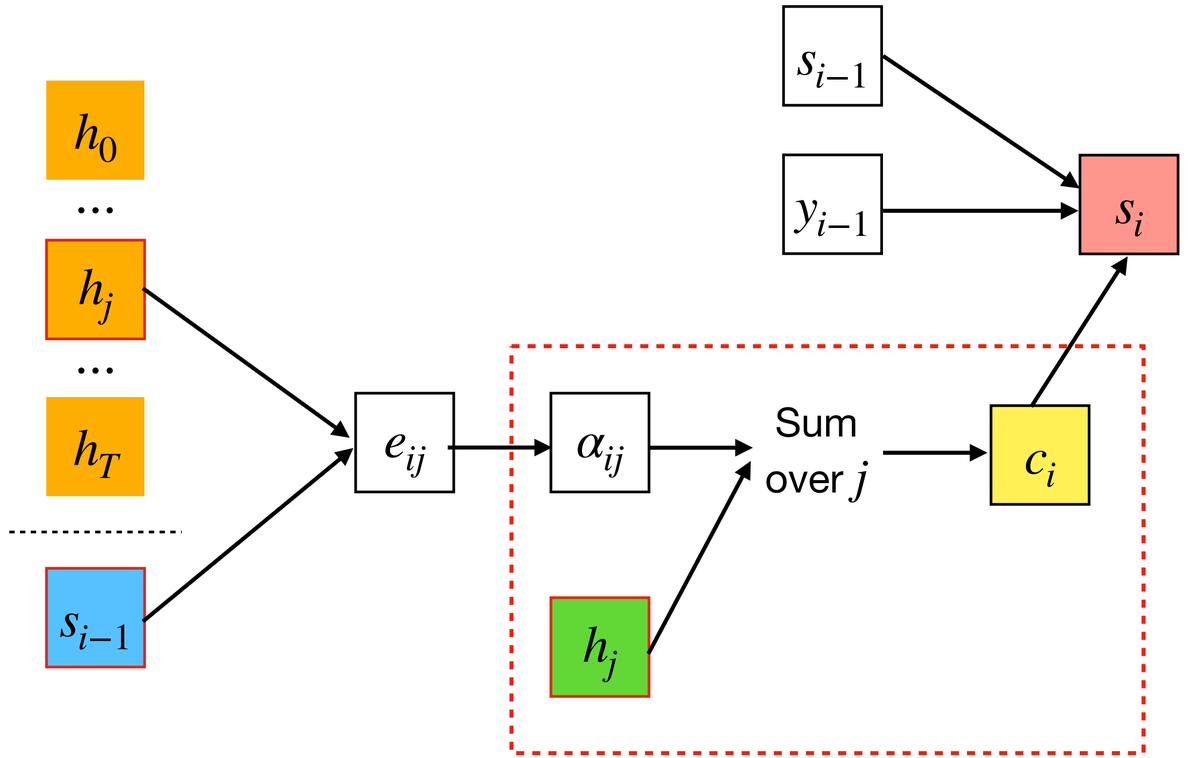
Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

Context vector c_i은 다음과 같다:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

즉, h_j (Encoder에서 $t = j$의 hidden state)

을 attention weighted한 평균.



h_j가 *"value"* 의 역할을 하고 있다!

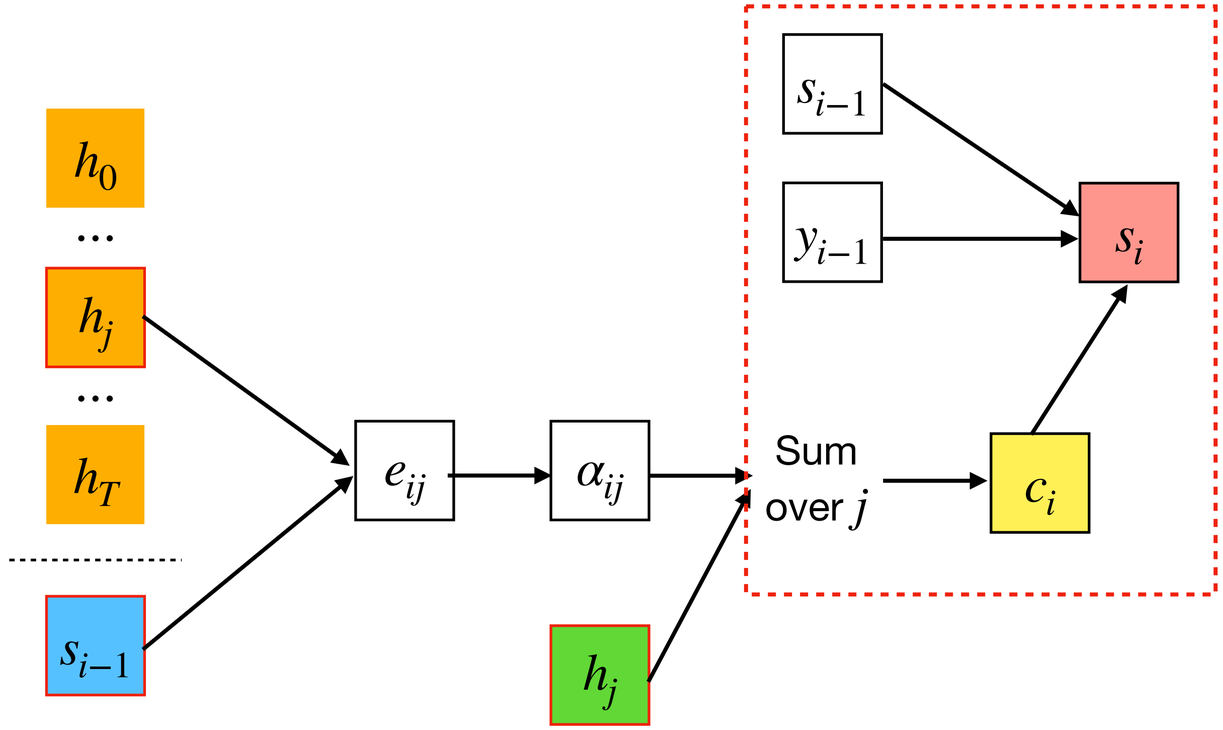
Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

Decoder에서 time-step i 에서 **hidden state** s_i 은:

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

(자세한 공식은 생략)



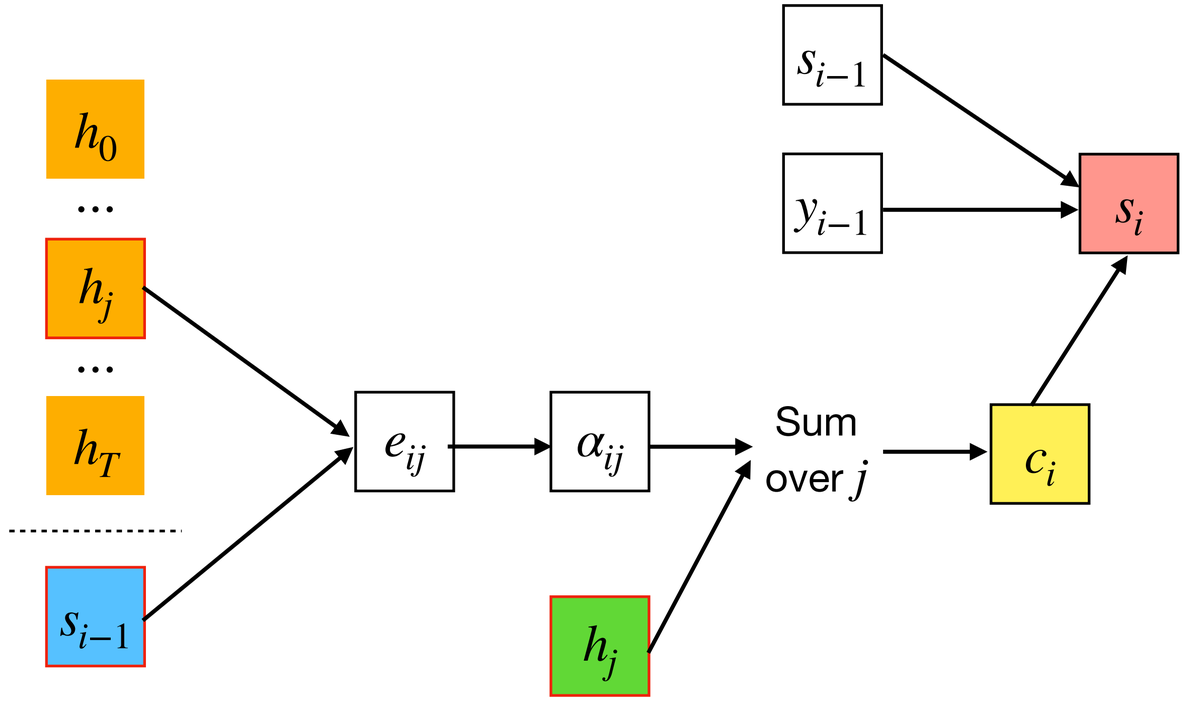
Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

c_i은 일종의 "h_j 의 Attention-weighted 평균값"이다

- query: s_{i-1}
- key: h_j
- value: h_j

요약하자면,
(Bahnadau et al 2015)에서는 **RNN기반**의 Encoder-decoder 구조의 **decoder에 attention을 도입**한 것이다!



Attention

Neural Machine Translation by Jointly Learning to Align and Translate (Bahnadau et al. 2015)

1부에서 배운 self-attention 과는 몇 가지 차이점들 있음:

1. Additive Attention을 사용
2. Value에 대한 weight matrix가 따로 정의되어 있지 않음
3. Query, Key Value가 동일한 feature가 아닌 다른 feature들로 비롯됨
   - Query은 decoder의 hidden state
   - Key, Value은 encoder의 hidden state에서 비롯됨
   - “원래의 attention”은 다른 feature로부터 query, key, value을 계산한다는 의미를 가지게 됨
   - 구분짓기 위해서 “self-attention”은 동일한 feature로부터 query, key, value을 계산한다는 의미를 가지게 됨

ACADENTIAL

16-4. Transformer 모델

Attention is all you need

ACADENTIAL

Attention

Attention is all you need (Vaswani et al. 2017)

Transformer 모델을 제안한 논문!

NeurIPS 2017 published

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

ACADENTIAL

Attention

Attention is all you need
(Vaswani et al. 2017)

특징:

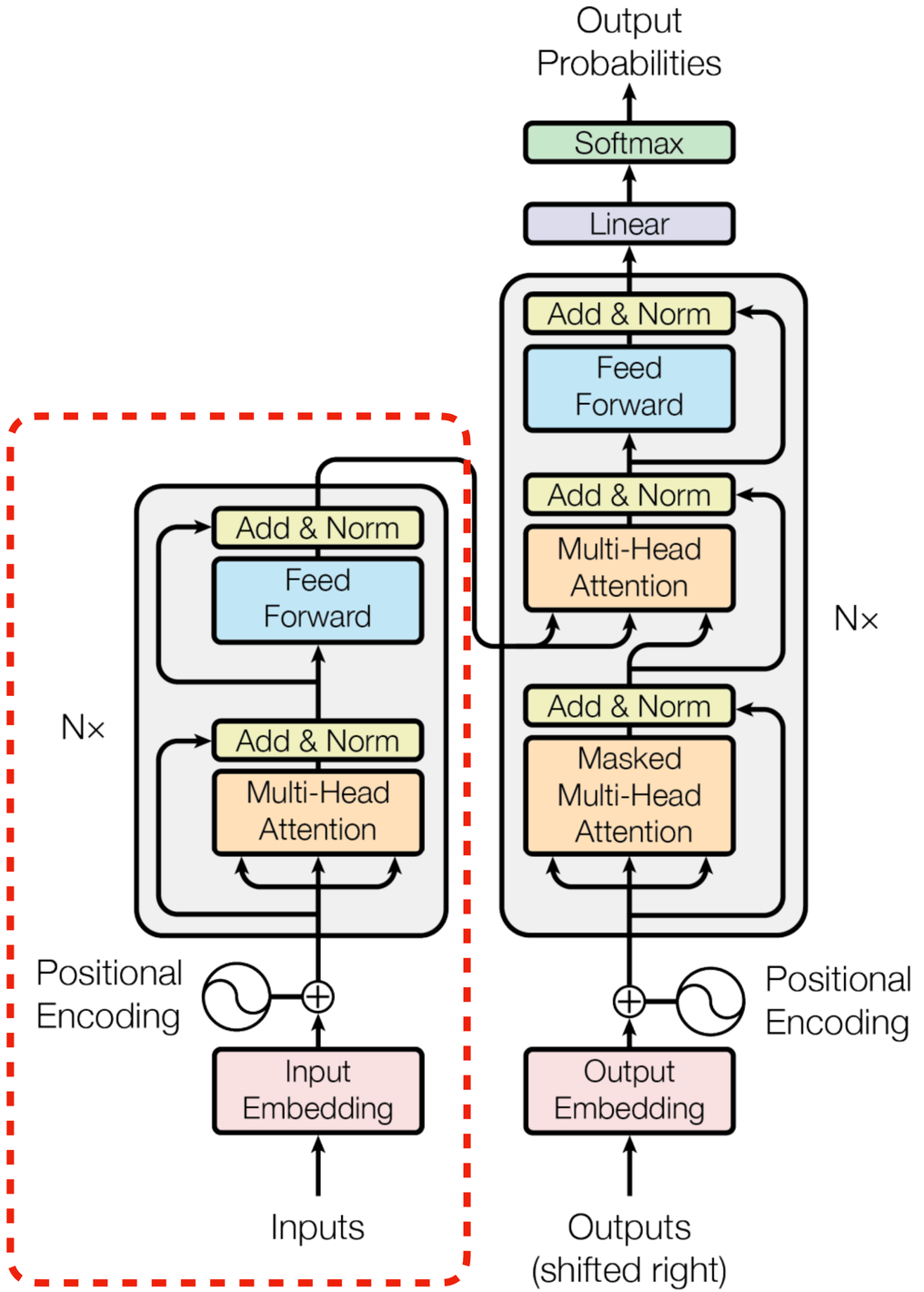
- Transformer 모델을 제안함!
- 기존에는 attention이 CNN 혹은 RNN과 함께 사용되었다.
- Transformer 모델은 오로지 **attention으로만** 구성된 모델이다!
- Encoder와 Decoder로 구성되어 있다.
- **Machine Translation (번역)**에 대해서 적용된 모델이다.

(Transformer 모델의 자세한 구조는 3부에서 다룰 예정)

Attention

Attention is all you need
(Vaswani et al. 2017)

- **Encoder**은 다음 구성 요소를 가진다:
  - Multi-head self-attention
  - FC layer
  - Residual connection
  - Layer norm
  - Positional encoding

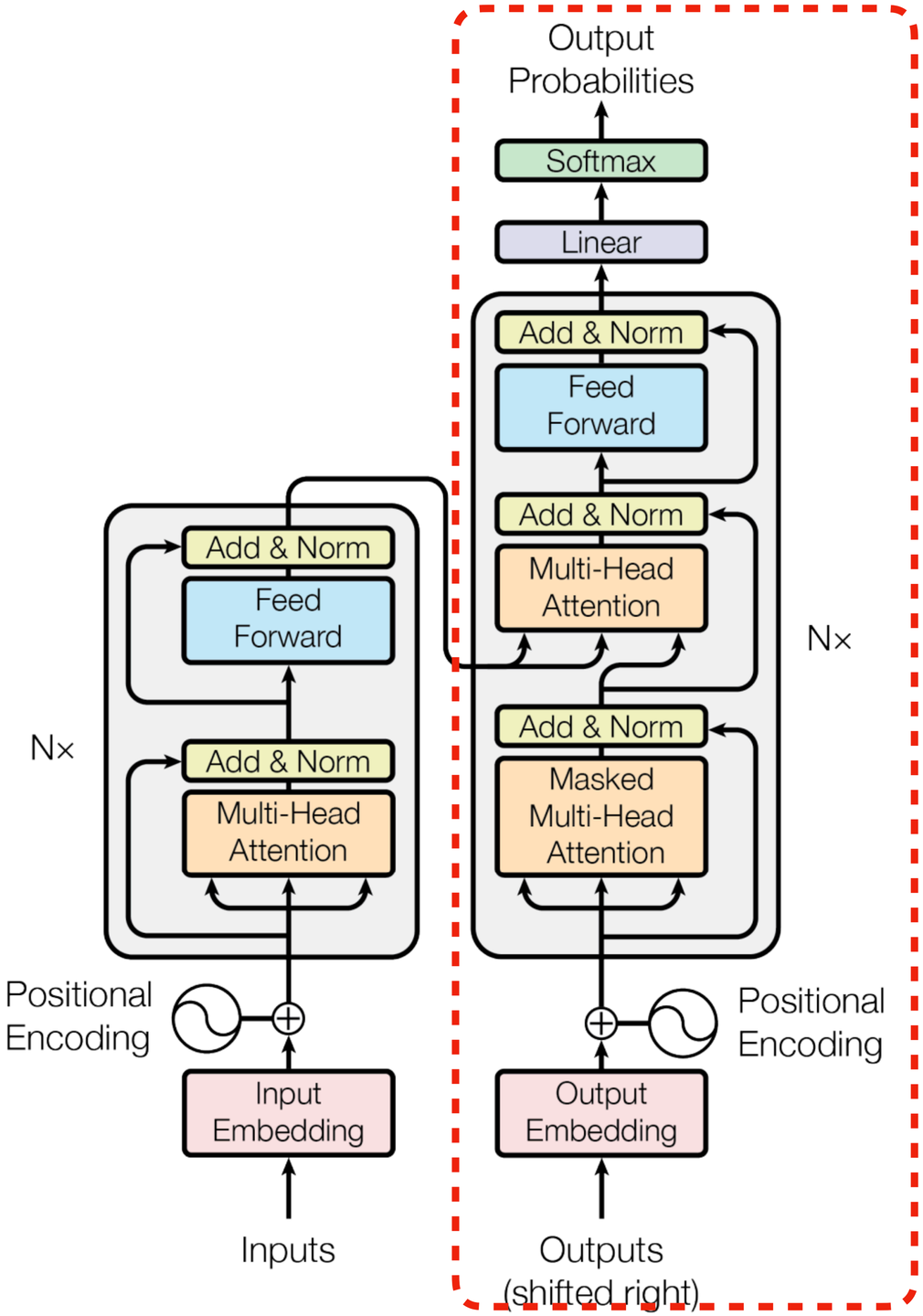


ACADENTIAL

Attention

Attention is all you need
(Vaswani et al. 2017)

- **Decoder**은 다음 구성 요소를 가진다:
  - Multi-head self-attention
  - FC layer
  - Residual connection
  - Layer norm
  - Positional encoding
  - **Multi-head attention**



ACADENTIAL

ACADENTIAL

16-5. BERT 모델

“BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”

ACADENTIAL

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

BERT 모델을 제안한 논문!

NAACL-HLT 2019 published

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

특징:

1. **BERT** (Bidirectional **Encoder** representations from **Transformers**) 모델을 제안함!
    - **BERT**은 **Transformer의 Encoder만 떼어낸 것**이다!

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

특징:

1. **BERT** (Bidirectional **Encoder** representations from **Transformers**) 모델을 제안함!
    - **BERT**은 **Transformer의 Encoder만 떼어낸 것**이다!
2. **Masked Language Modeling pre-training** (MLM 사전 학습 방법)을 처음으로 제안함!
    - MLM pre-training → Fine-tuning
    - **Fine-tuning**하는 과정에서는 **마지막 layer만 교체.**

ACADENTIAL

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

“Masked Language Modeling (MLM)”은 무엇인가?

아래 예시를 살펴보자

닥스훈트는 | 강아지의 | 한 | 부류이다

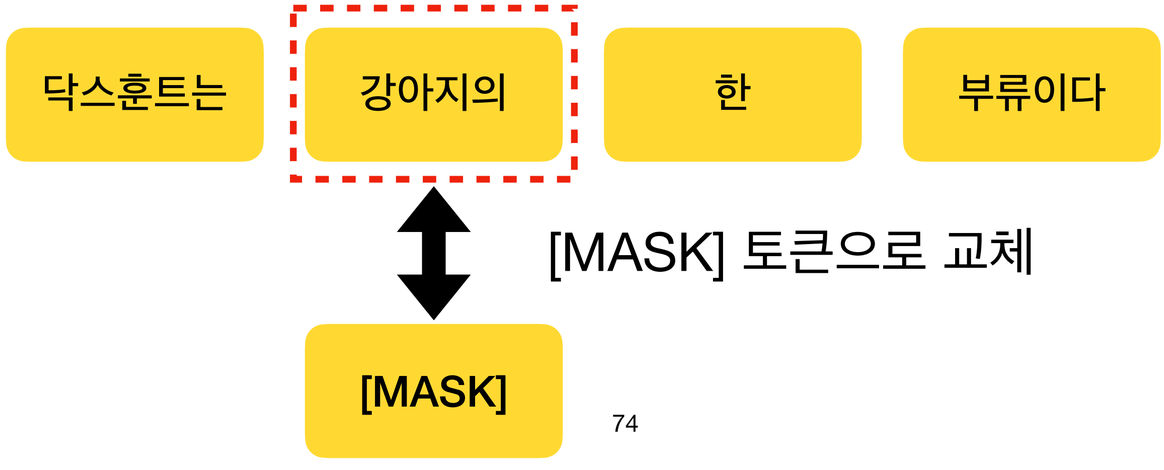
ACADENTIAL

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

“Masked Language Modeling (MLM)”은 무엇인가?

MLM에서는 random한 확률로 토큰들 중 일부를 “Mask”한다.



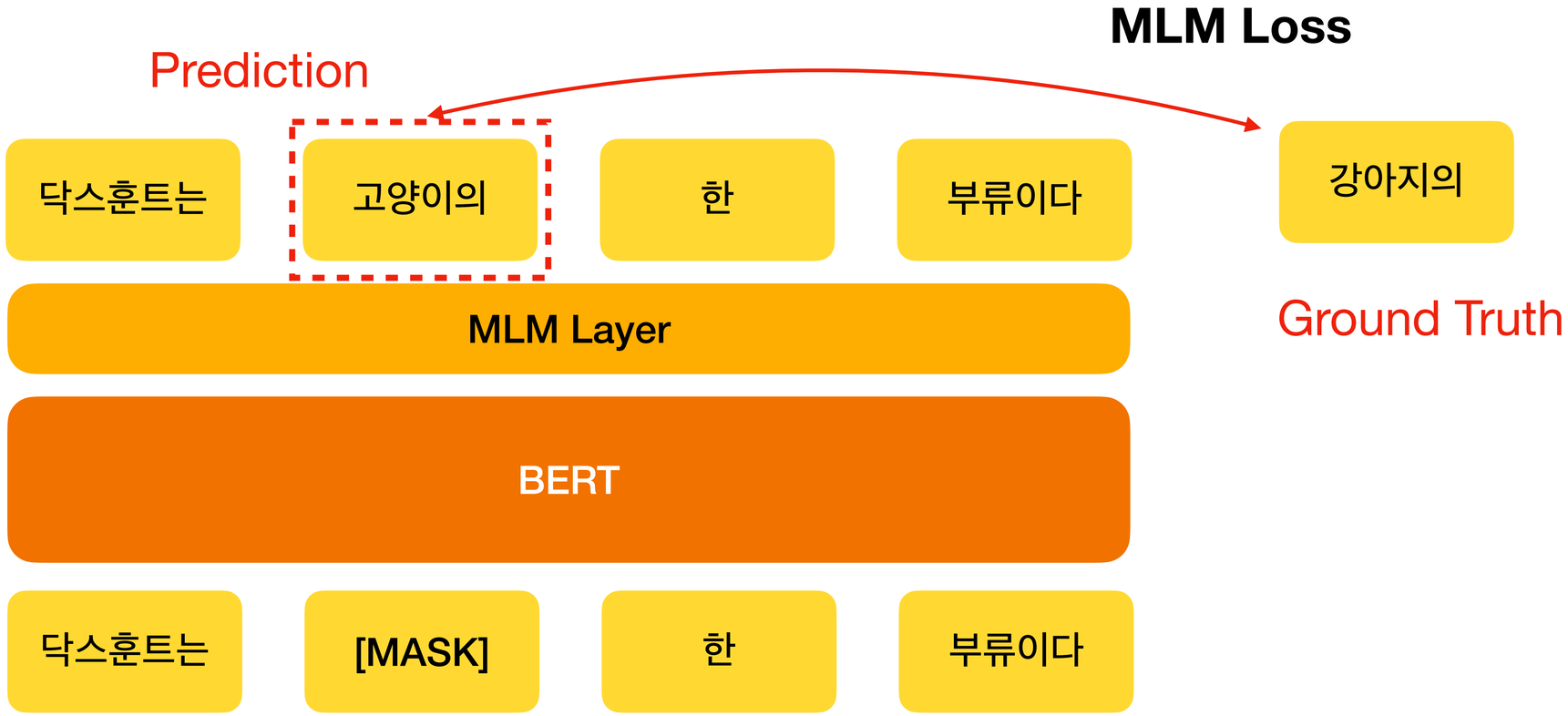
ACADENTIAL

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

“Masked Language Modeling (MLM)”은 무엇인가?

“[MASK]”된 토큰을 맞추도록 모델을 학습시킨다!



Attention

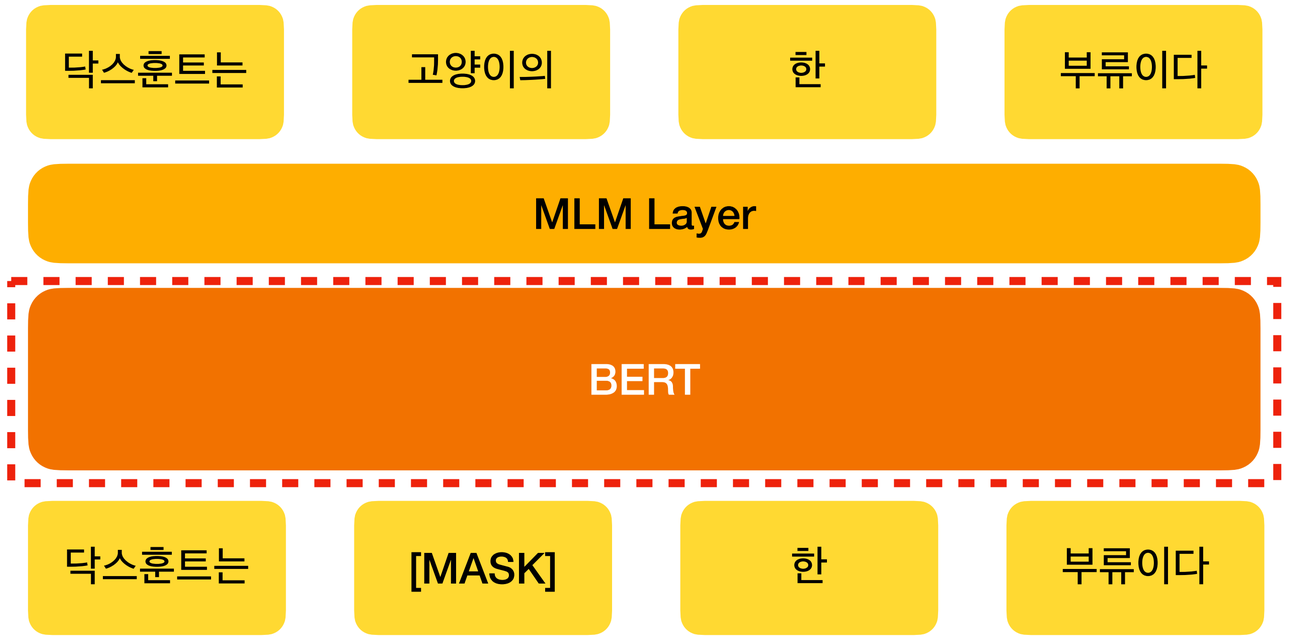
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

MLM으로 사전 학습 (pre-train)된 모델은 **target task에 대해서 fine-tuning (조율 학습)**한다.

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

MLM으로 사전 학습 (pre-train)된 모델은 **target task에 대해서 fine-tuning (조율 학습)**한다.

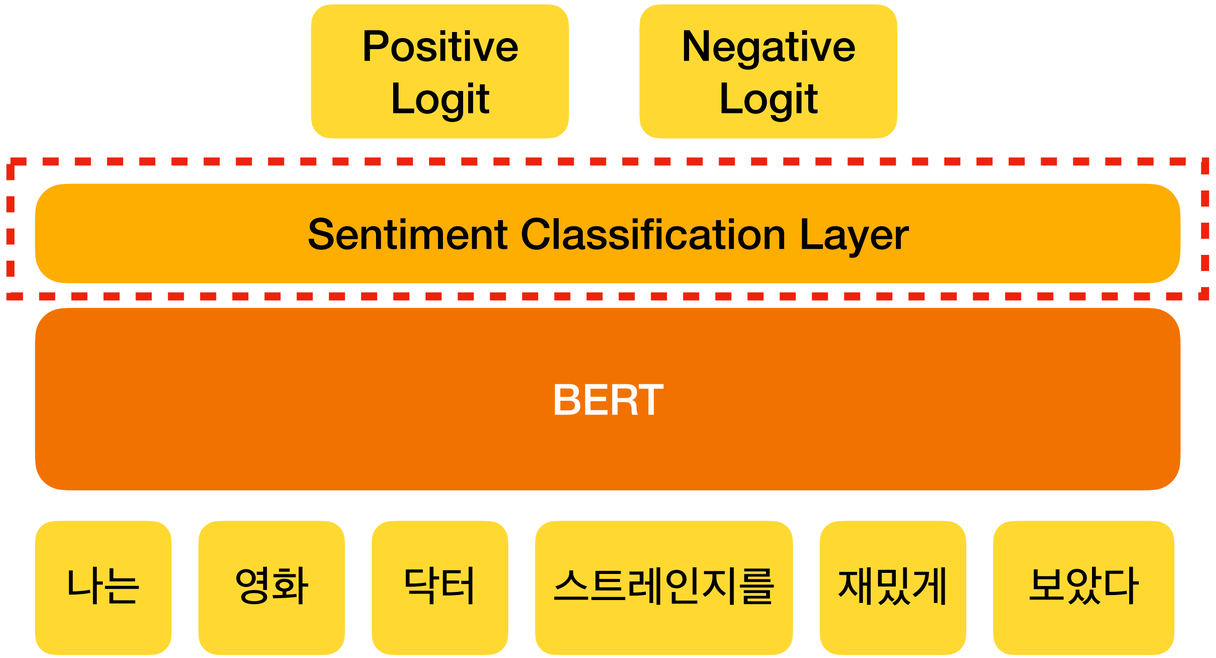


1. MLM Layer이 제외된 부분을 “떼어낸다”.

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

MLM으로 사전 학습 (pre-train)된 모델은 **target task에 대해서 fine-tuning (조율 학습)**한다.



1. MLM Layer이 제외된 부분을 “떼어낸다”.
2. target task에 적합한 layer을 추가해서 해당 task에 대해서 fine-tune한다.

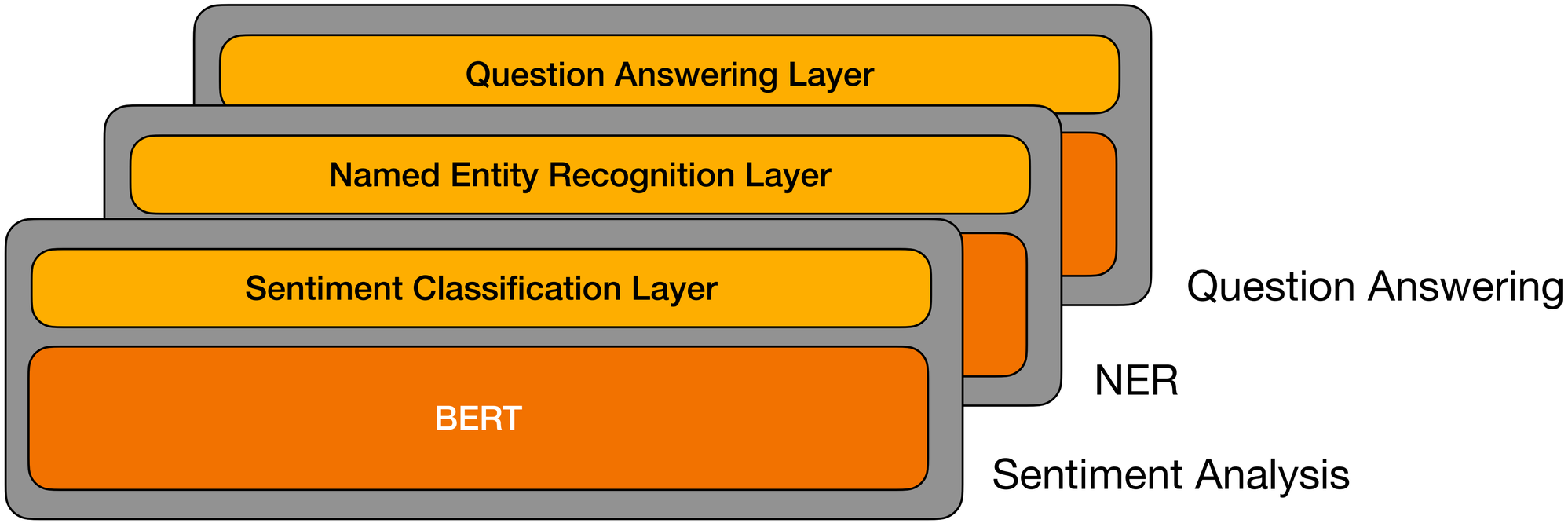
ACADENTIAL

Attention

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Devlin et al. 2018)

BERT 모델을 base로 공통적으로 가지고 마지막 layer만 task-specific하다!

즉, “NLP에서 다양한 task에 대해서 unified된 framework”을 제안한 셈이다!



ACADENTIAL

16-6. Transformer의 Encoder

ACADENTIAL

Attention

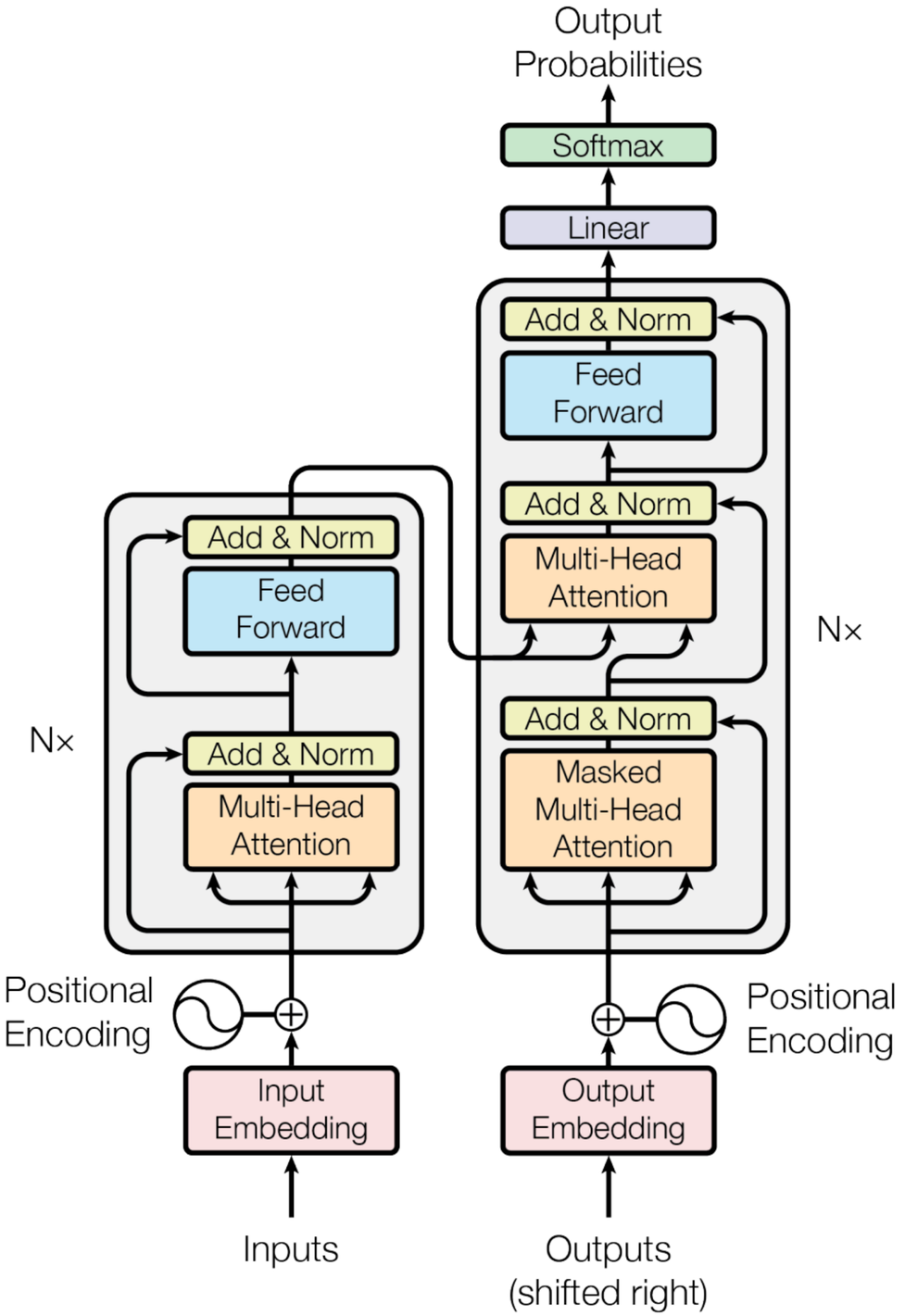
Transformer

- **1부**에서는 **self-attention, attention**을 살펴보았다.
- **2부**에서는 **Attention의 계보**를 살펴보았다.
  - (Bahnadau 2015)에서 처음 제안된 Attention 기반의 RNN 모델.
  - (Vaswani et al 2017)에서 제안된 Transformer 모델.
  - (Devlin et al 2018)에서 제안된 BERT (Transformer의 Encoder을 MLM 사전 학습) 모델.
- **3부**에서는 **Transformer모델의 구조**를 자세하게 살펴보자!

Attention

Transformer

- Transformer 모델은 다음과 같이 구성됨:
  - Encoder
  - Decoder

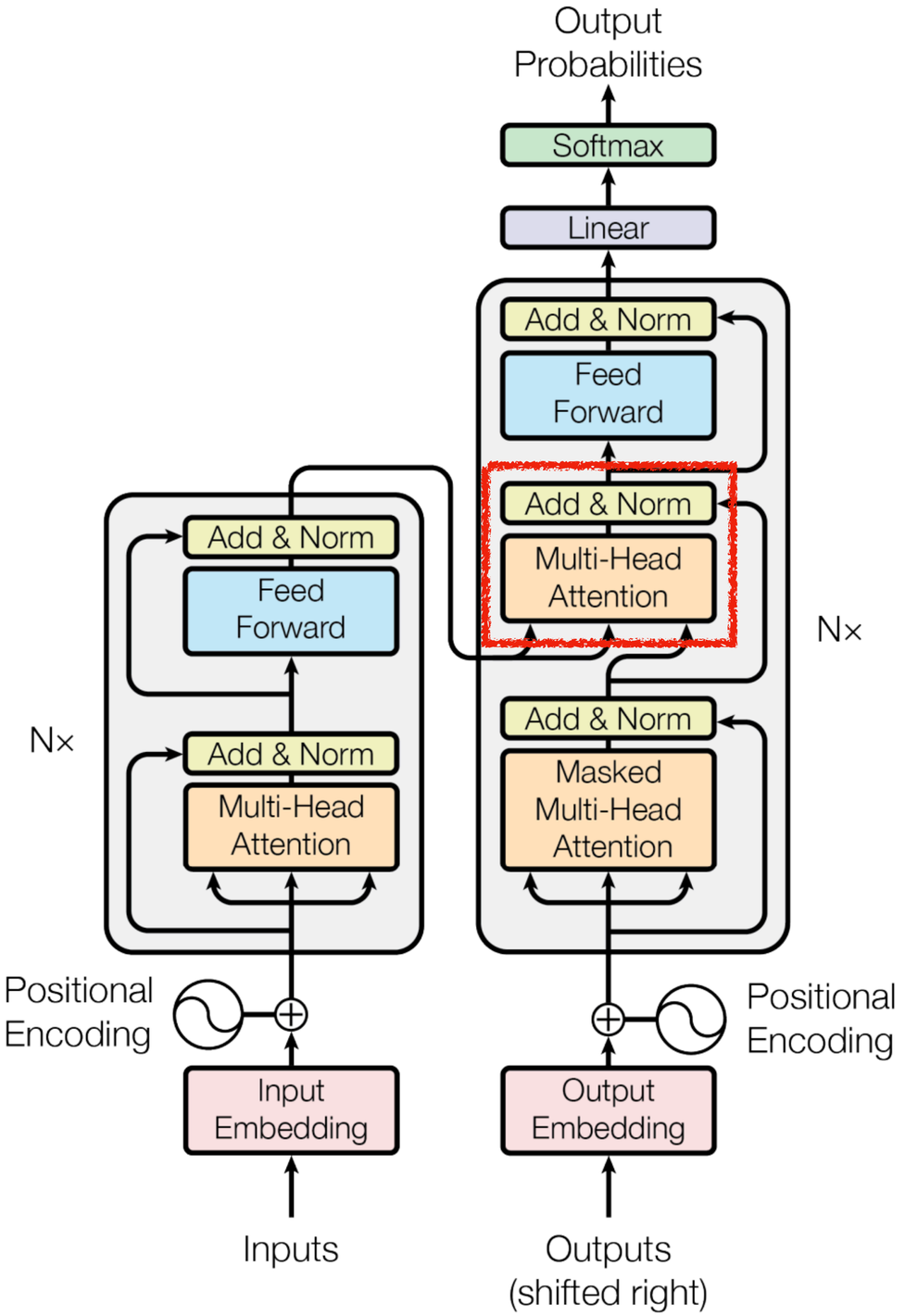


ACADENTIAL

Attention

Transformer

- Transformer 모델은 다음과 같이 구성됨:
  - Encoder
  - Decoder
- 먼저 Encoder을 자세하게 살펴보자!
- Decoder은 Encoder의 구조와 거의 유사함 (빨간색 표시만 차이)



ACADENTIAL

ACADENTIAL

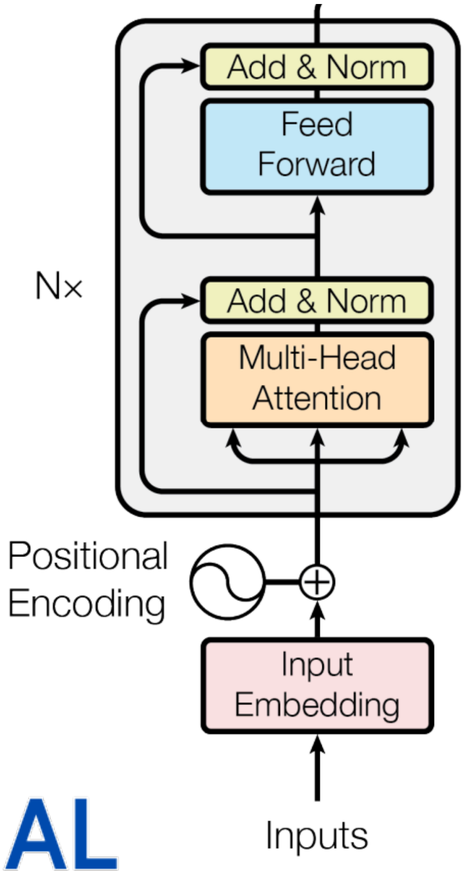
Attention

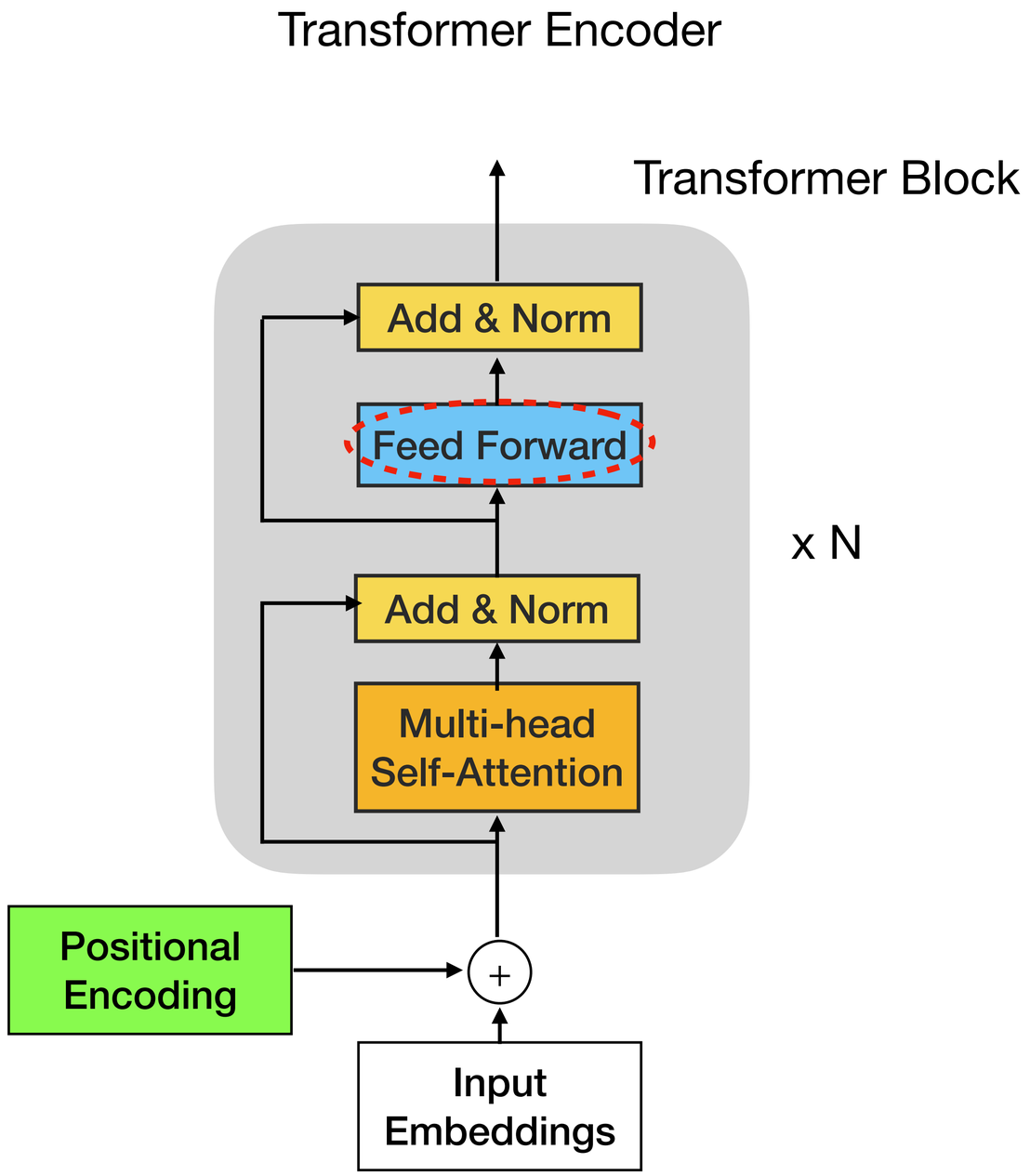
Transformer Encoder의 구성 요소

- Feed forward layer
- Layer Norm
- Residual connection
- Scaled dot product
- Positional Embedding
- Multi-head attention

Attention

Transformer Encoder의 구성 요소



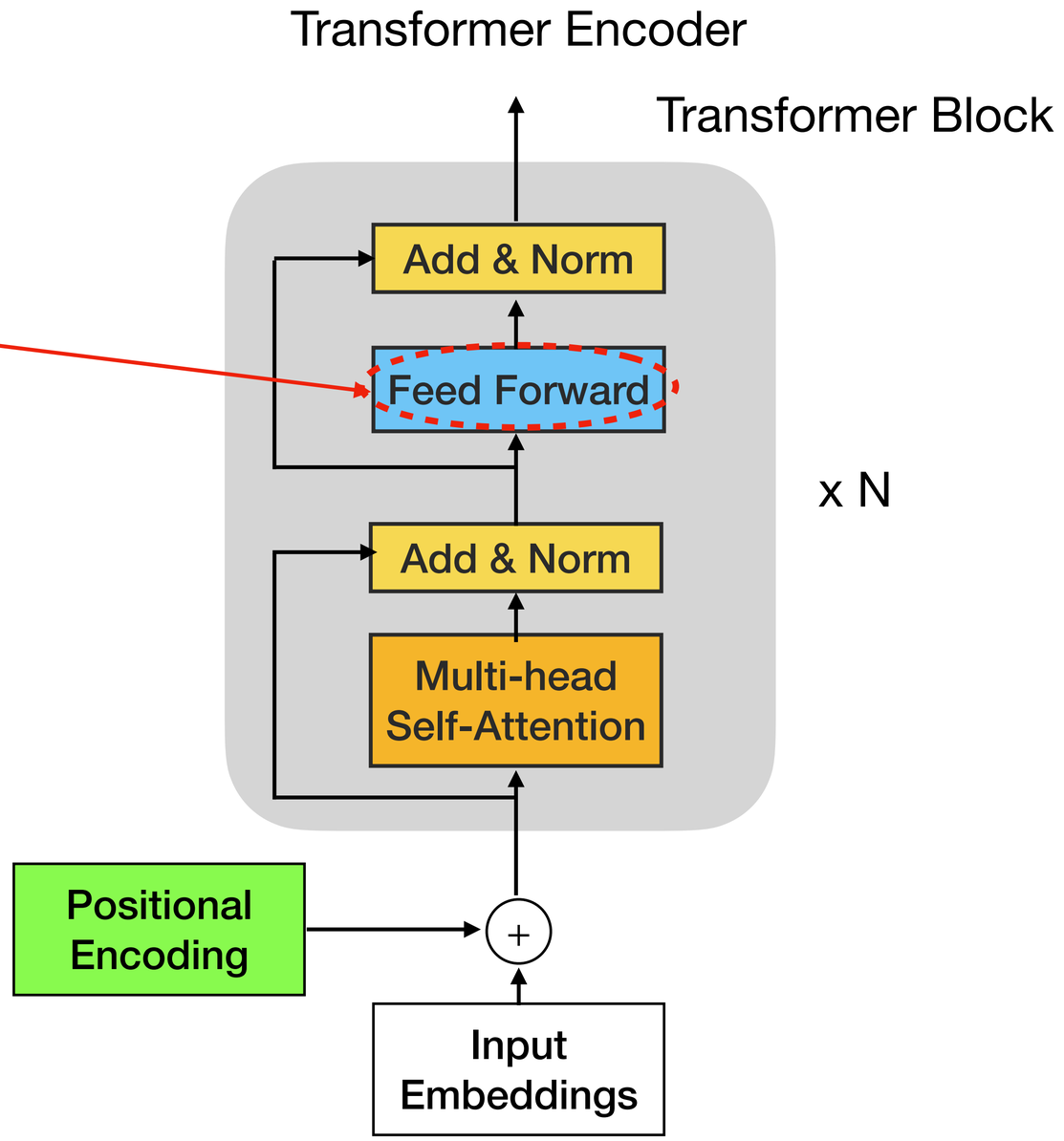


ACADENTIAL

Attention

Transformer Encoder의 구성 요소

- Feed forward layer

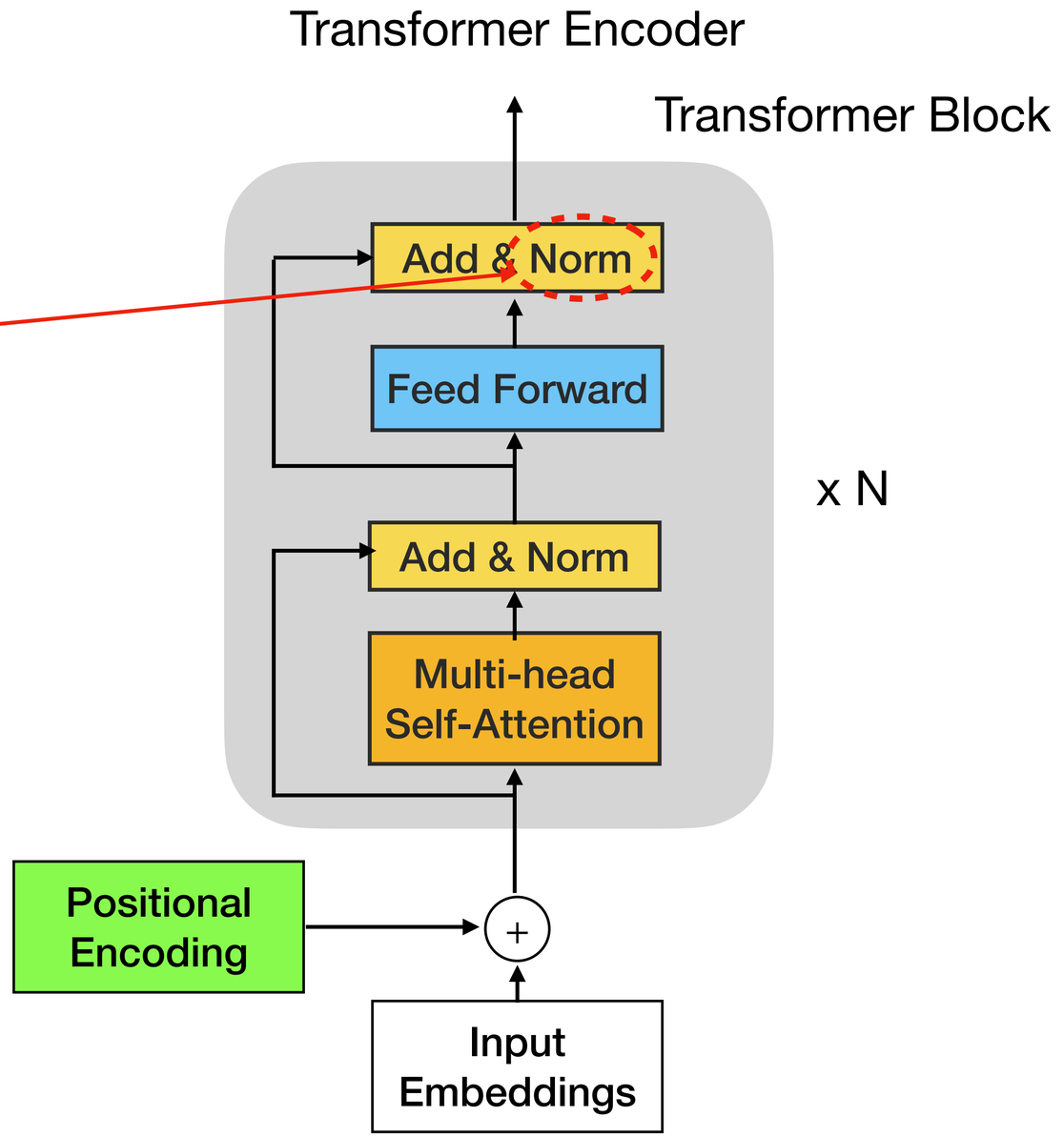


ACADENTIAL

Attention

Transformer Encoder의 구성 요소

- Feed forward layer
- Layer norm layer

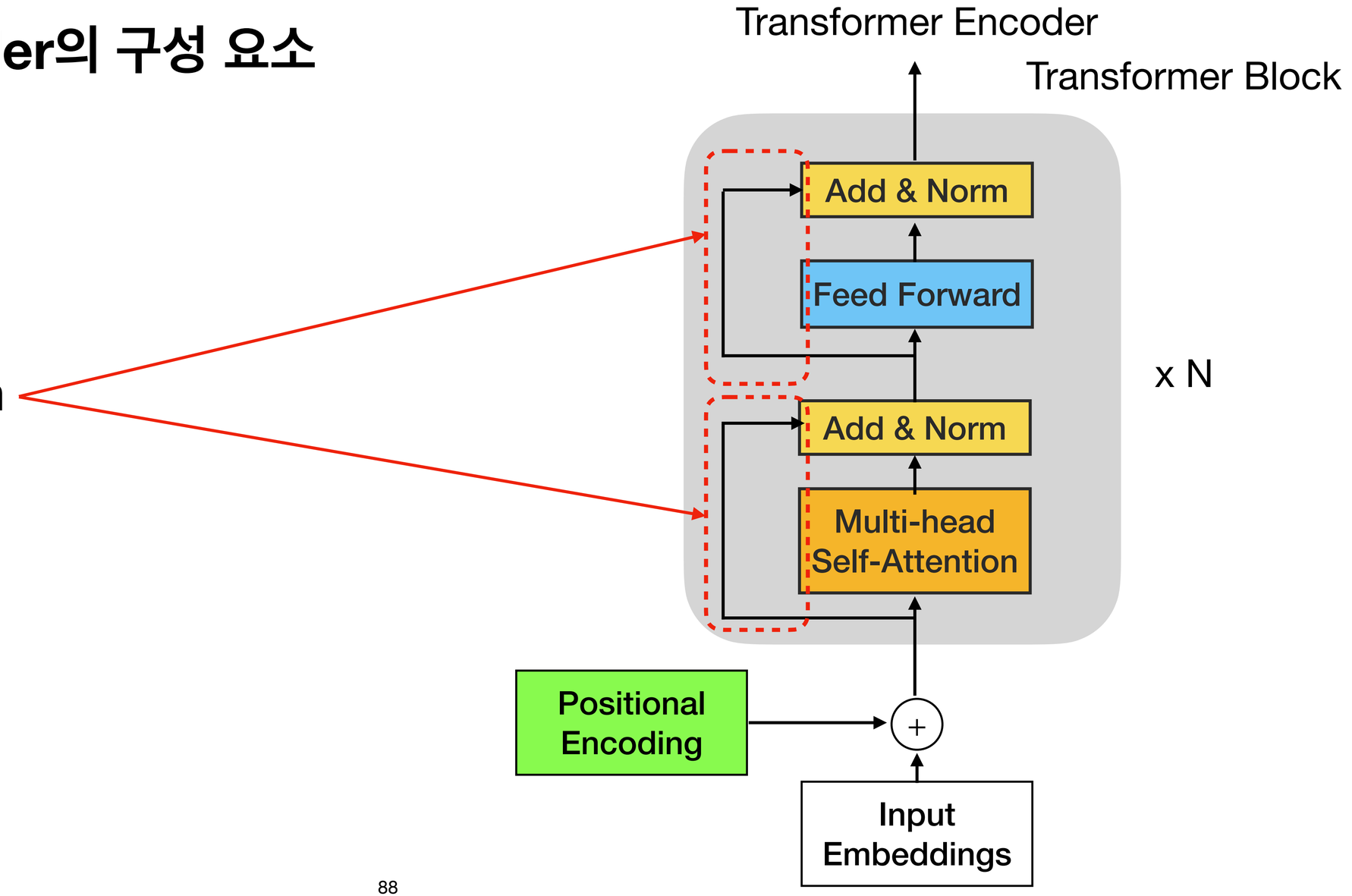


ACADENTIAL

Attention

Transformer Encoder의 구성 요소

- Feed forward layer
- Layer norm layer
- Residual connection

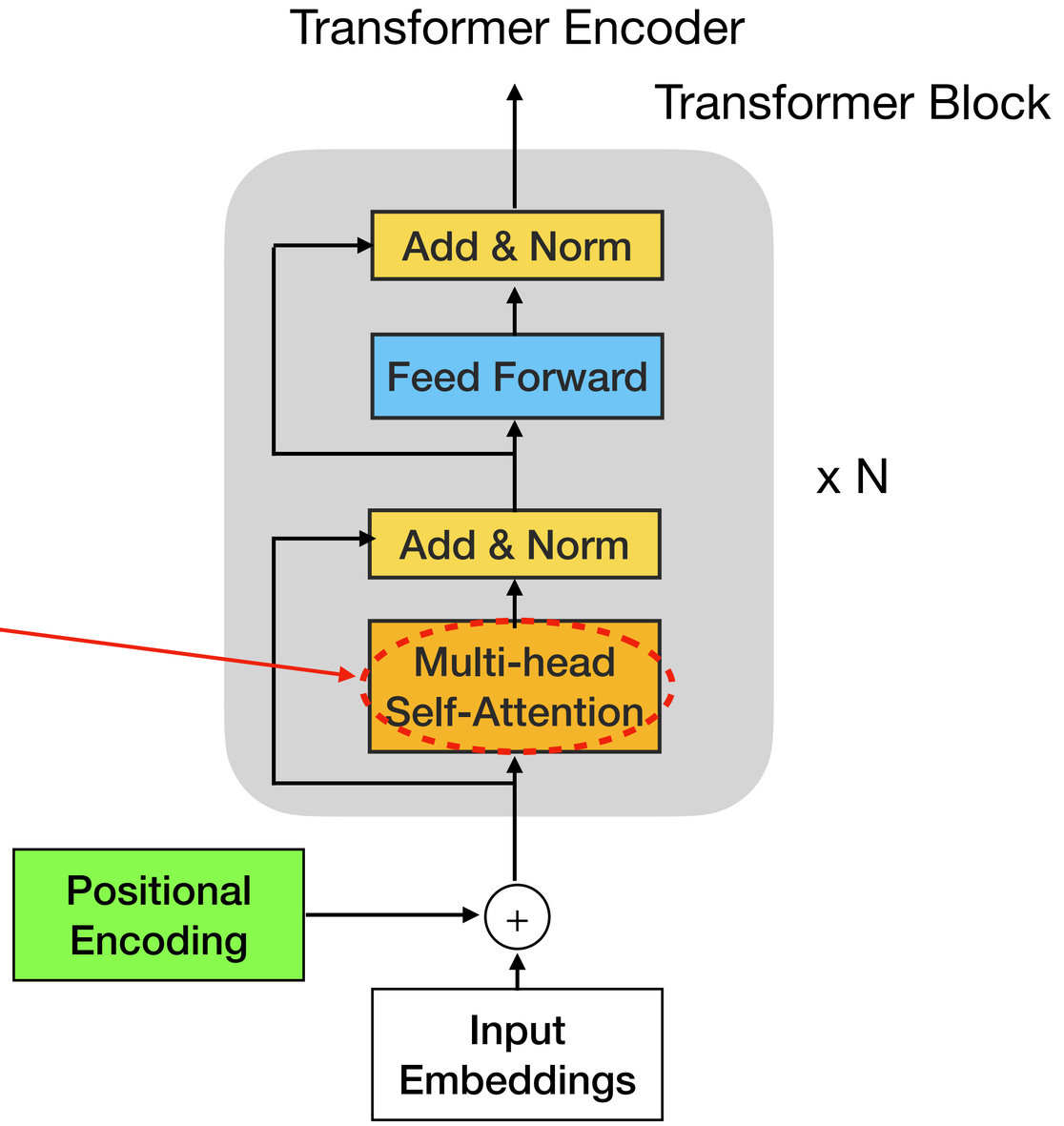


ACADENTIAL

Attention

Transformer Encoder의 구성 요소

- Feed forward layer
- Layer norm layer
- Residual connection
- Scaled Dot Product, multi-head attention



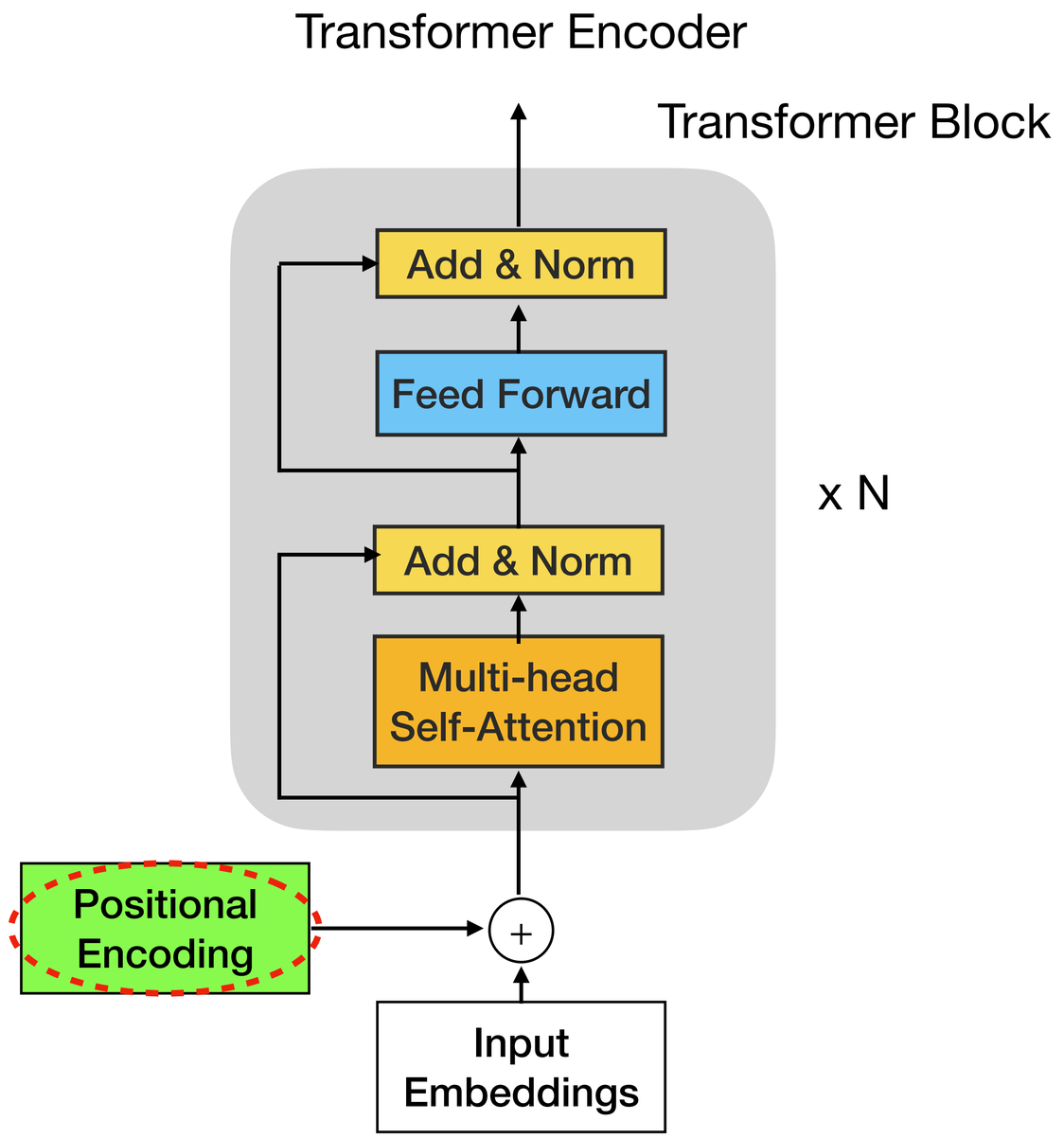
ACADENTIAL

Attention

Transformer Encoder의 구성 요소

- Feed forward layer
- Layer norm layer
- Residual connection
- Scaled Dot Product, multi-head attention
- Positional Encoding

각각 왜 필요한지 살펴보자!



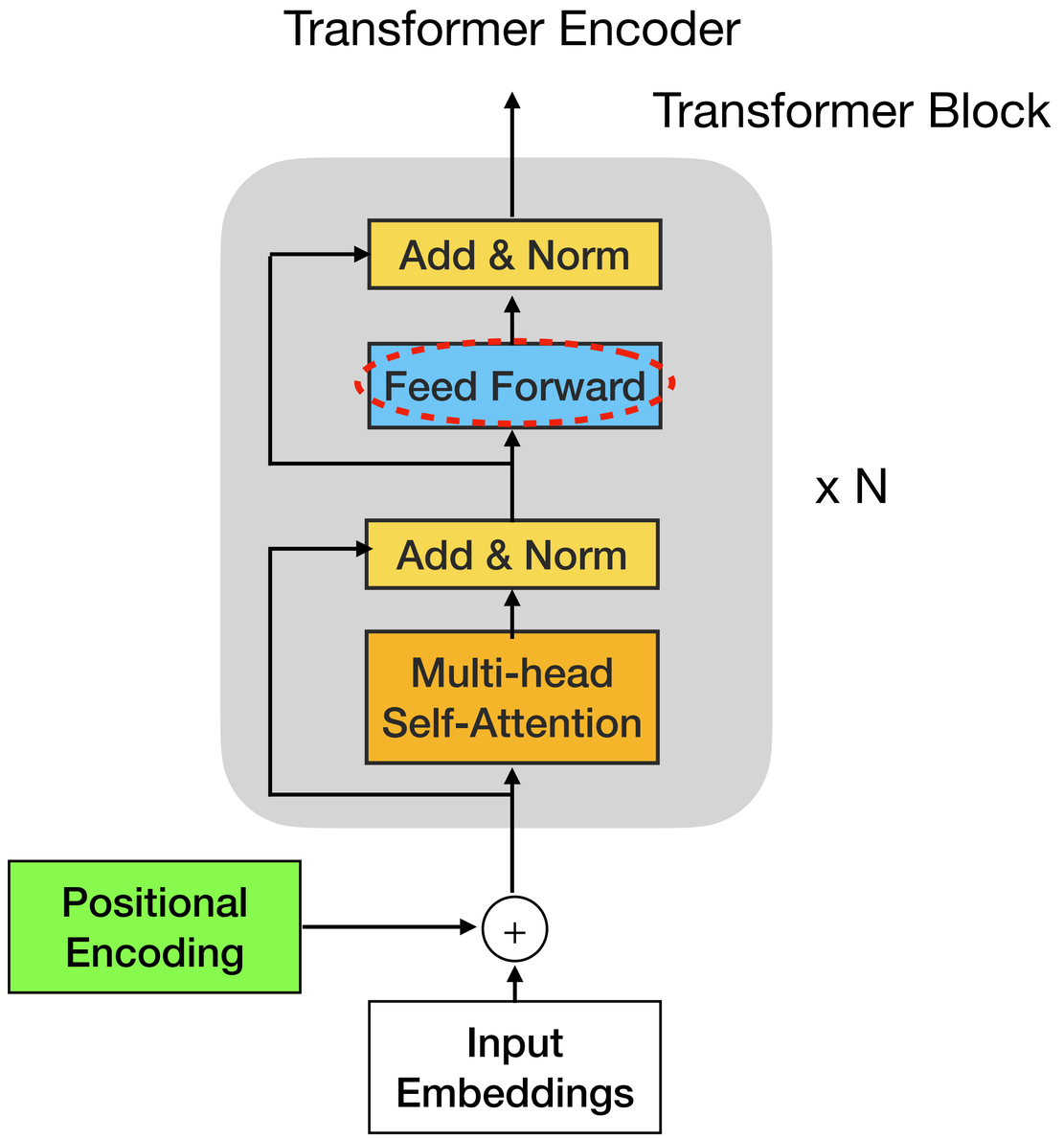
Attention
Feed forward layer

Feed forward layer

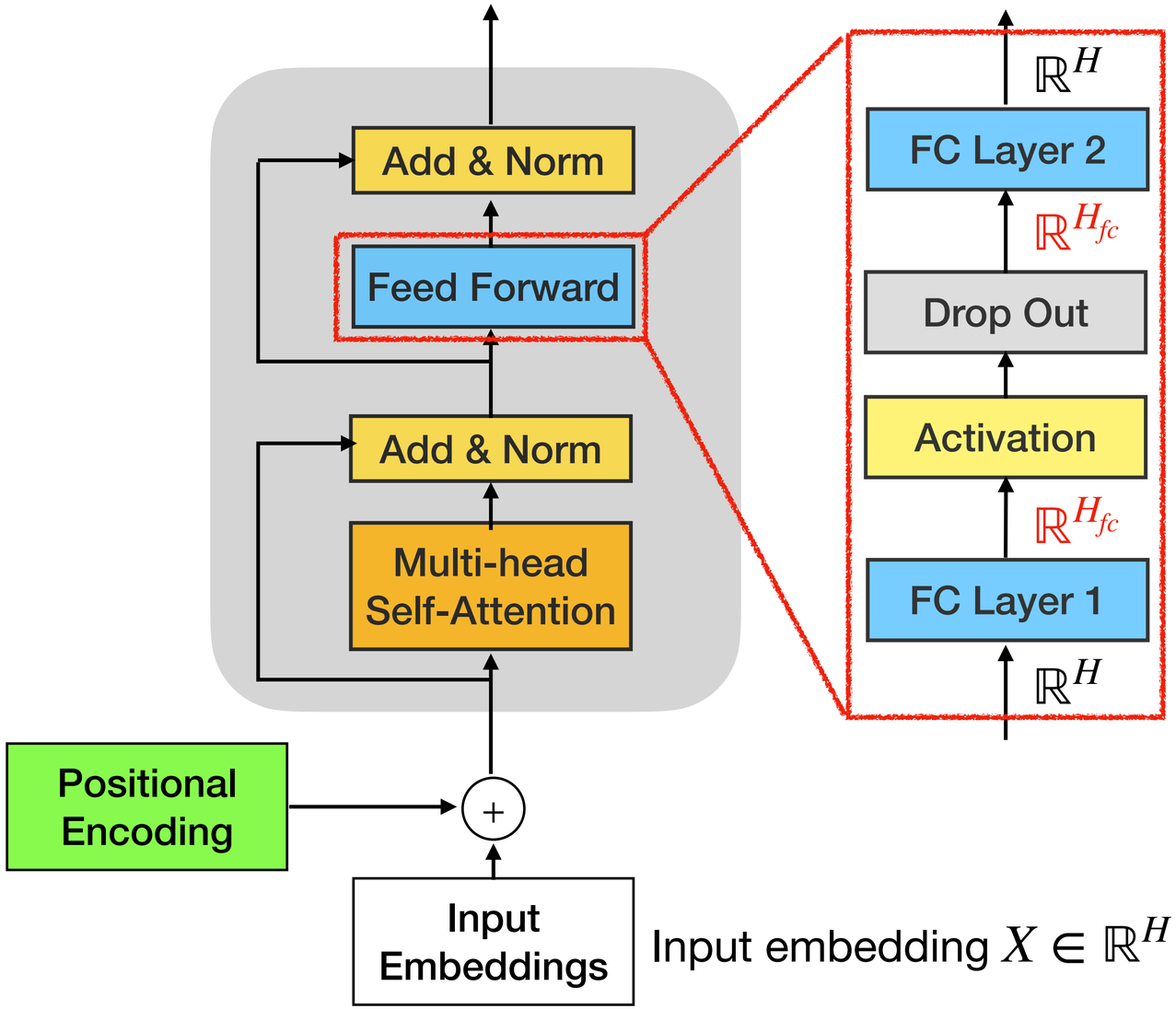
- 왜 필요한가?
- 먼저 Self-attention을 recap해보자:

$$O = \text{softmax}(QK^T)V = AV = A(XW_v^T)$$

- 즉, output은 Value에 대한 “attention weighted average”이다.
- Input X에 대한 non-linearity가 없다!
- 하지만, NN이 복잡한 decision boundary을 학습하려면 non-linearity은 필수이다!



Transformer Encoder's block

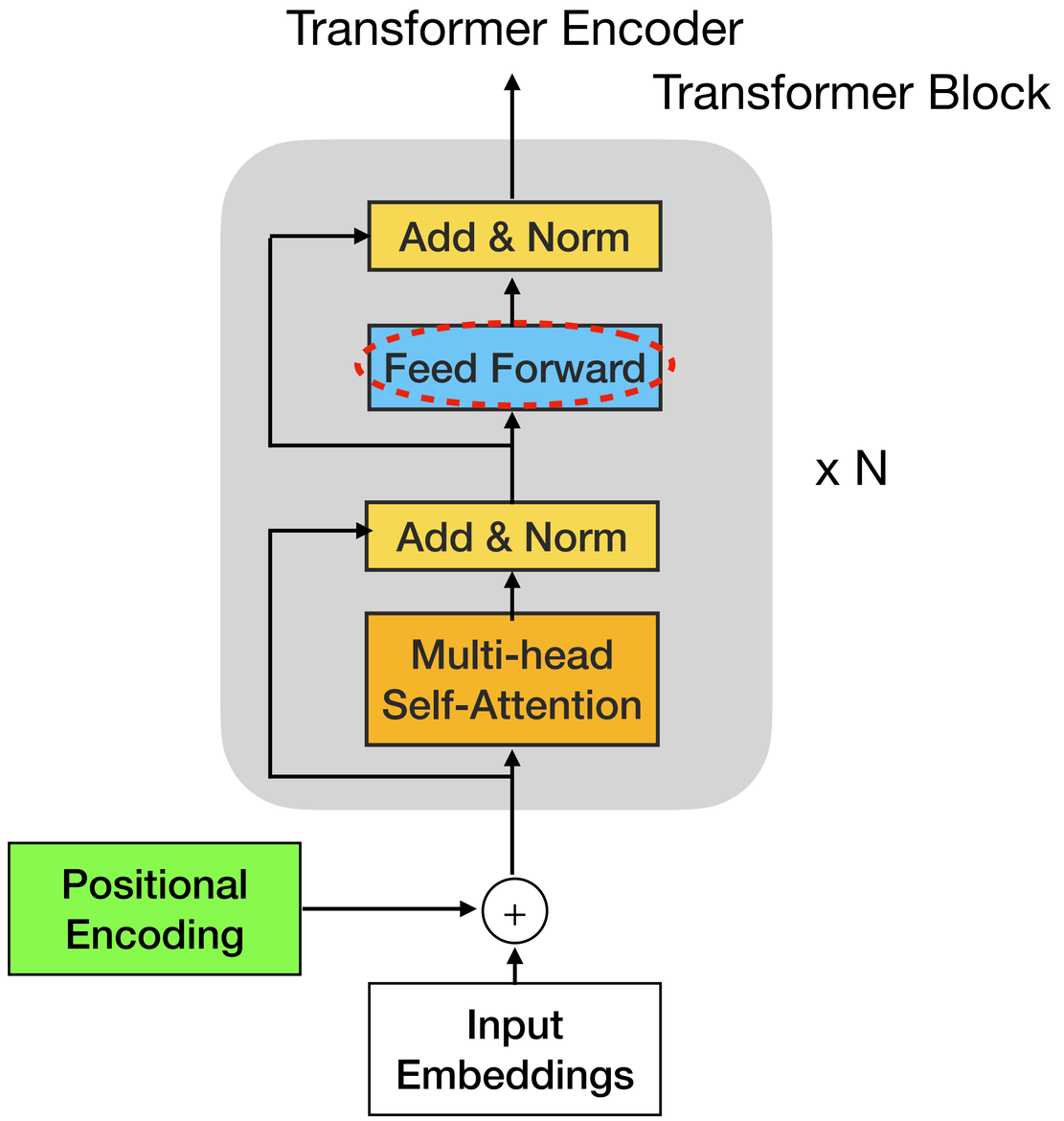


- (참고로) Feed forward block은 **두 개의 FC layer들로 구성되어 있다.**

Attention
Feed forward layer

Feed forward layer

- 일반적으로 Feed forward layer은
  - “FC layer → ReLU → FC Layer” 로 구성.
  - $y_i = W_2 \cdot ReLU(W_1 \cdot x_i + b_1) + b_2$

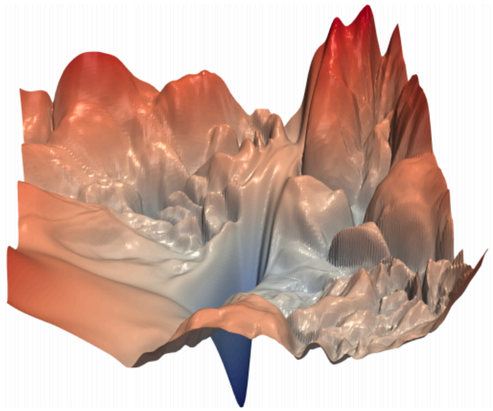


Attention

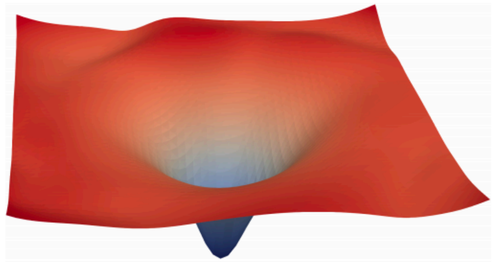
Layer Norm

Layer Norm

- 왜 필요한가?
  - “Normalization chapter”에서 살펴봤듯이 Normalization은 Loss surface을 smooth해주는 효과가 있다!

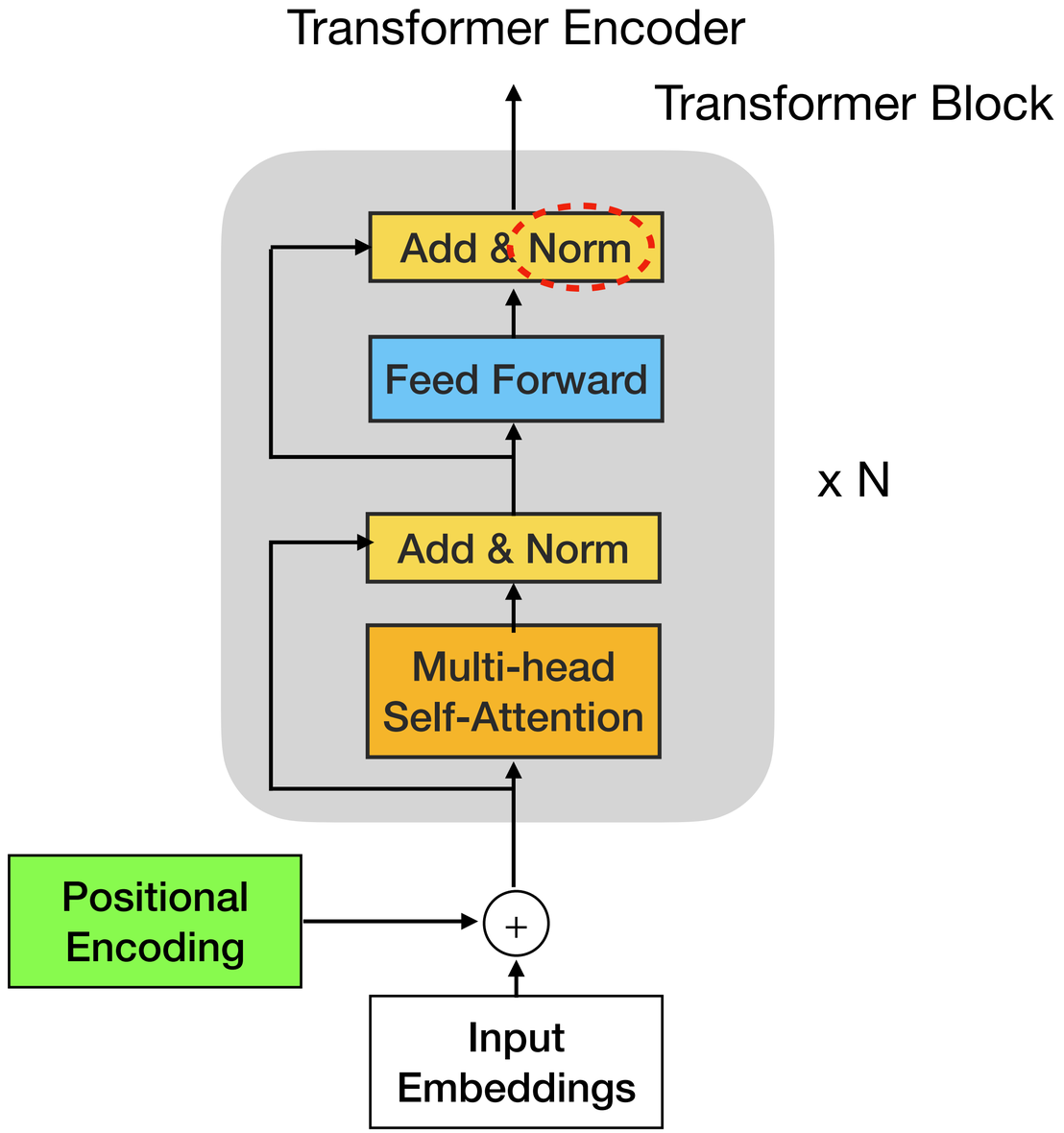


without norm layer



with norm layer

Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., 2018. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, *31*.

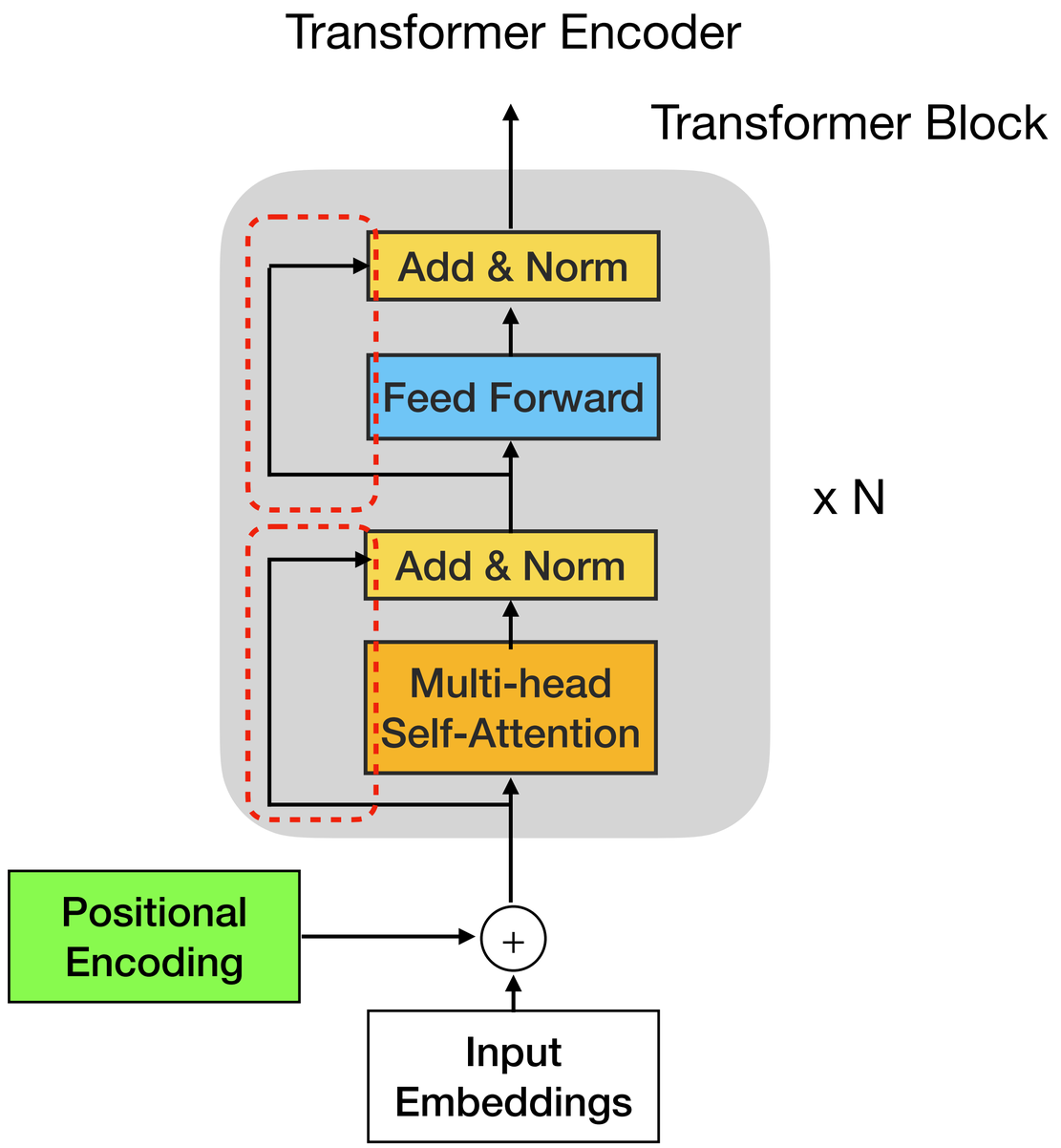


Attention

Residual Connection

Residual Connection

- 왜 필요한가?
  - “ResNet chapter”에서 살펴봤듯이 Residual Connection은 Degradation 문제와 Vanishing Gradient 문제를 해소하는데 도움이 되기 때문이다.



Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T., 2018. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, *31*.

Attention

Scaled dot product

Scaled dot product 이란?

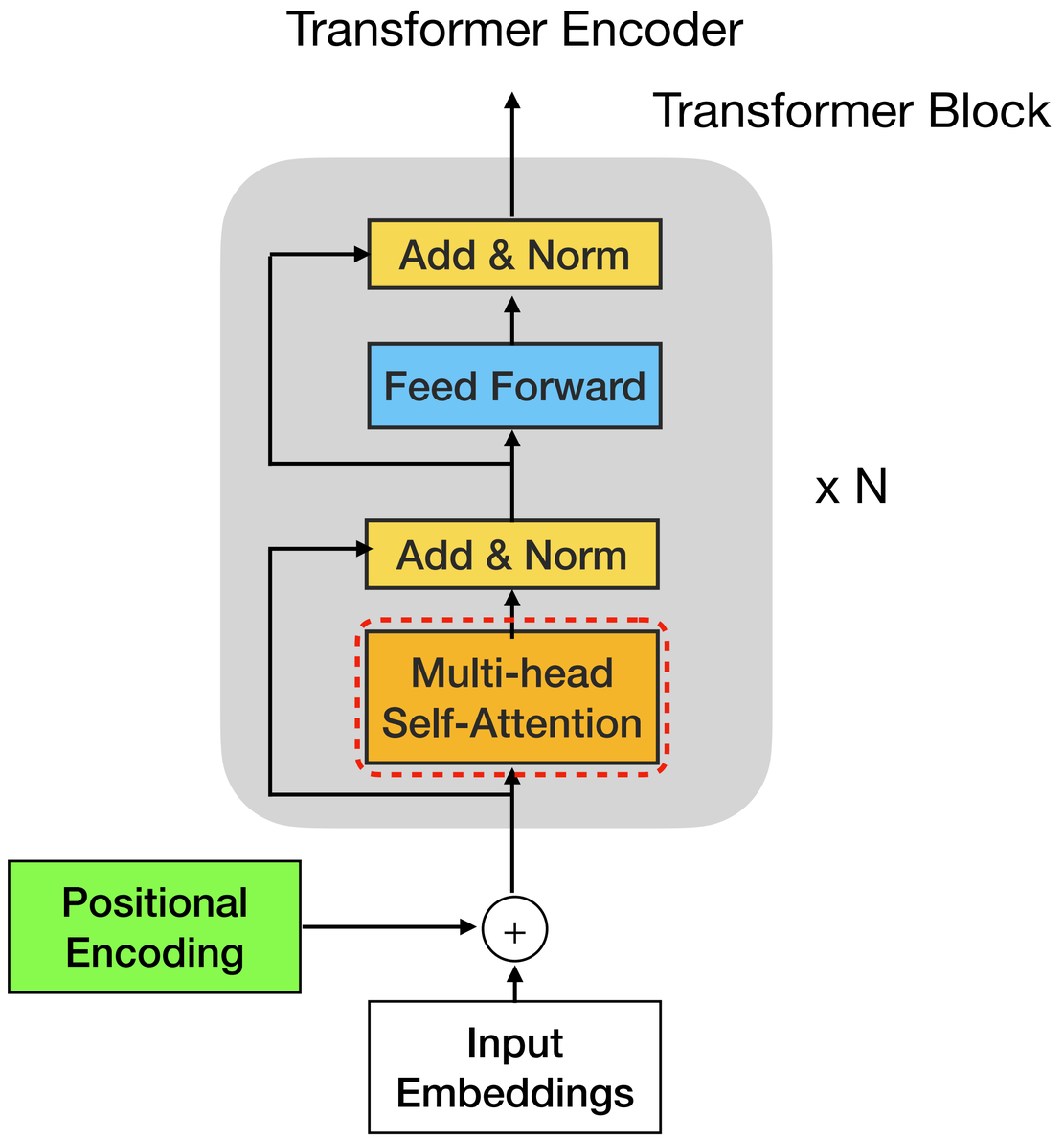
- 기존의 self-attention에서

$$O = \text{softmax}(QK^T)V$$

- 다음과 같이 scale 하는 것:

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

참고로 H은 Q, K, V의 dimension (즉, $Q, K, V \in \mathbb{R}^H$)



ACADENTIAL

Attention

Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 먼저 attention score은 $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$ 이다

ACADENTIAL

Attention

Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 먼저 attention score은 $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$ 이다
  - 여기서 query와 key vector의 각 element은 normal distribution (mean = 0, standard deviation = 1)

ACADENTIAL

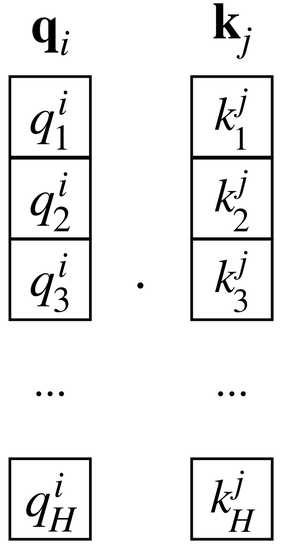
Attention

Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 먼저 attention score은 $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$ 이다
  - 여기서 query와 key vector의 각 element은 normal distribution (mean = 0, standard deviation = 1)
  - 즉, $q_h^i, k_h^j \sim N(0,1)$



ACADENTIAL

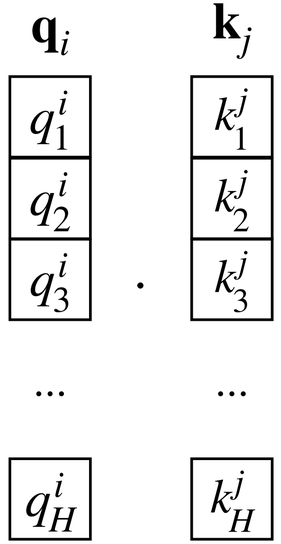
Attention

Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 먼저 attention score은 $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$ 이다
  - 여기서 query와 key vector의 각 element은 normal distribution (mean = 0, standard deviation = 1)
  - 즉, $q_h^i, k_h^j \sim N(0,1)$
  - 따라서, $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j \sim N(0,H)$



ACADENTIAL

Attention

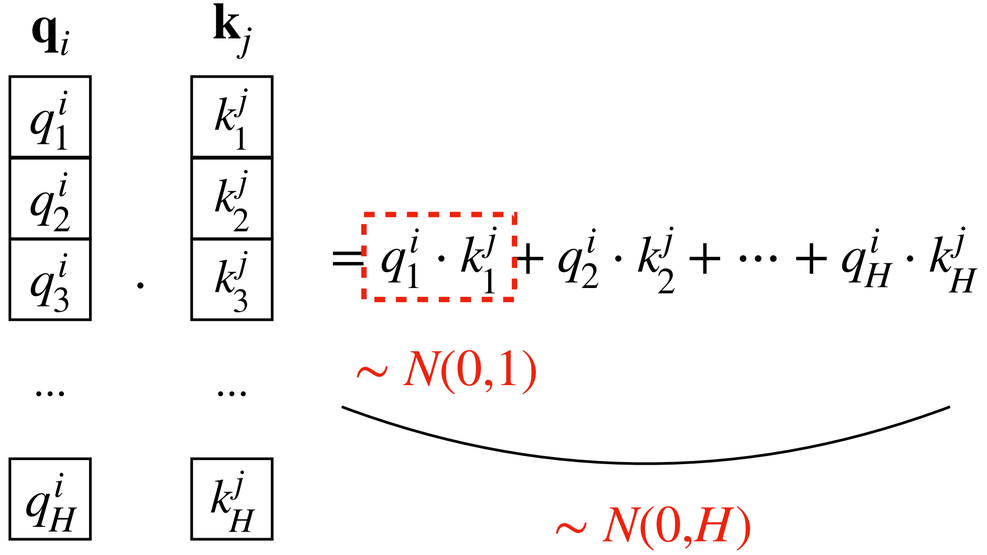
Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 먼저 attention score은 $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j$ 이다
  - 여기서 query와 key vector의 각 element은 normal distribution (mean = 0, standard deviation = 1)

- 즉, $q_h^i, k_h^j \sim N(0,1)$
- 따라서, $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j \sim N(0,H)$



$N(0,1)$을 H번 더했기 때문에!

Attention

Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 따라서, $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j \sim N(0,H)$
  - (H 가 클수록 더 퍼져있는셈이다)

Attention

Scaled dot product

Scaled dot product 이란?

$$O = \text{softmax}(QK^T/\sqrt{H})V$$

- 왜 필요한가?
  - 따라서, $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j \sim N(0,H)$
  - (H 가 클수록 더 퍼져있는셈이다)
  - 따라서 이를 교정하기 위해서 $1/\sqrt{H}$ scale해준다!
  - $e_{ij} = \mathbf{q}_i \cdot \mathbf{k}_j/\sqrt{H} \sim N(0,1)$

$$\mathbf{q}_i \cdot \mathbf{k}_j \rightarrow \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{H}}$$

$$\text{softmax}\left(QK^T\right)V \rightarrow \text{softmax}\left(\frac{QK^T}{\sqrt{H}}\right)V$$

Attention

Positional Encoding

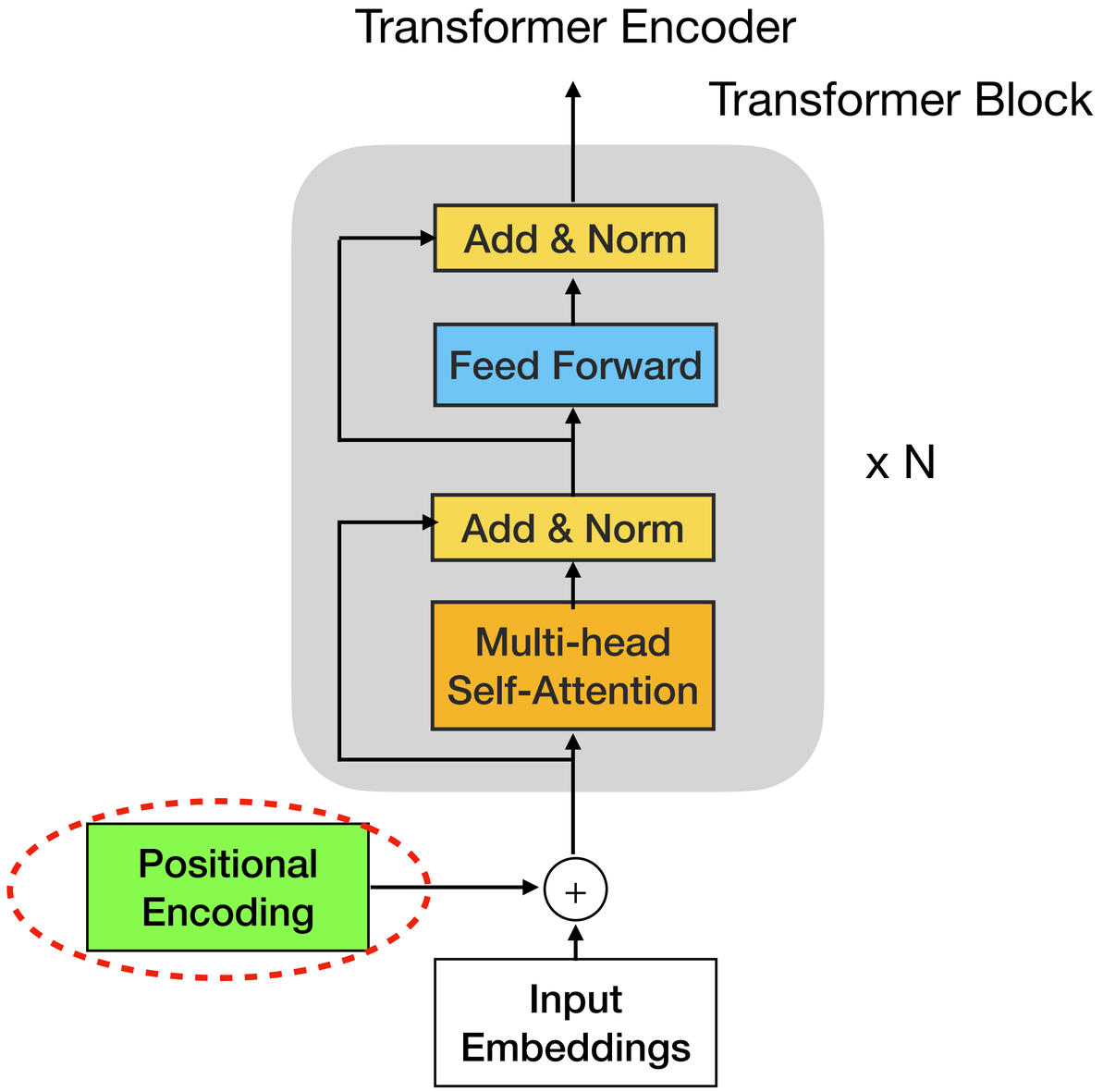
Positional Encoding

- 왜 필요한가?
  - Self-attention은 sequence order (단어의 순서)에 dependent하지 않다!

"나는 카페에서 책을 보면서 커피를 마신다"

"카페는 책에서 커피를 보면서 나를 마신다"

즉, self-attention은 위 두 문장을 구분하지 못한다!



Attention

Positional Encoding

- 해결책: **Positional Encoding**

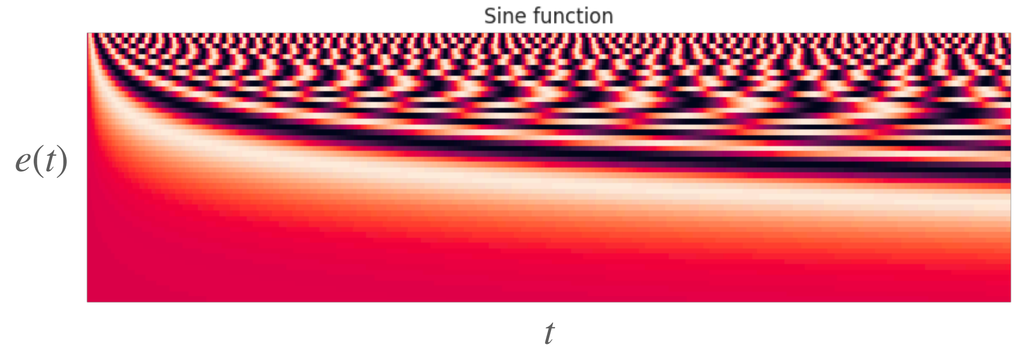
1. Sinusoidal Positional Representation
2. Learned positional embedding

position을 의미하는 position encoding (feature)을 따로 두는 것!

token의 position에 따라 해당 feature의 값은 다르도록 설정하는 것이다!

Attention

Sinusoidal Positional Representation



Sinusoidal Positional Representation

token의 position t에 대한 positional encoding을 **sinusoidal 함수**로 표현

encoding의 각 element는 **다른 period의 sine, cosine 함수로 구성**됨

$$e(t) = \boldsymbol{E}_{t,:} := \begin{bmatrix} \sin\left(\frac{t}{f_1}\right) \\ \cos\left(\frac{t}{f_1}\right) \\ \sin\left(\frac{t}{f_2}\right) \\ \cos\left(\frac{t}{f_2}\right) \\ \vdots \\ \sin\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \\ \cos\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \end{bmatrix}, \quad f_m = \frac{1}{\lambda_m} := 10000^{\frac{2m}{d_{\text{model}}}}.$$

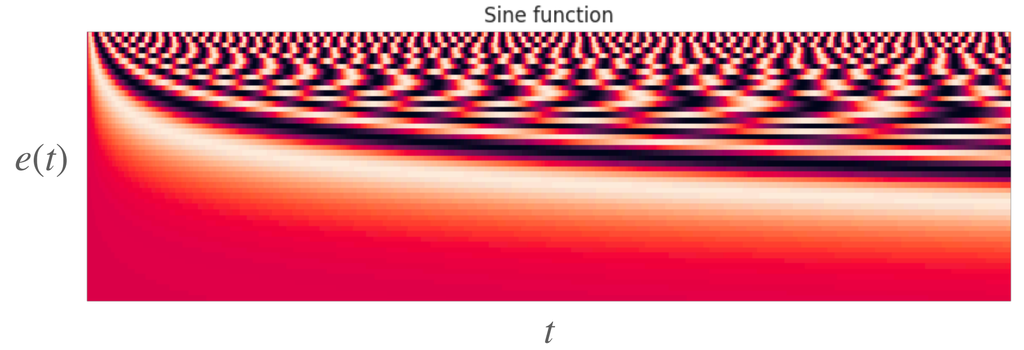
그렇다면 이 positional encoding을 기존의 input embedding $\mathbf{x}_i$에 어떻게 합칠 수 있을까?

ACADENTIAL

출처: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Attention

Sinusoidal Positional Representation



기존의 input embedding에 element-wise addition 혹은 concatenation 해줌

Element-wise addition (일반적으로 사용됨)

- $\mathbf{x}_i \rightarrow \mathbf{x}_i + \mathbf{e}_i$

Concatenation

- $\mathbf{x}_i \rightarrow \text{concat}[\mathbf{x}_i, \mathbf{e}_i]$

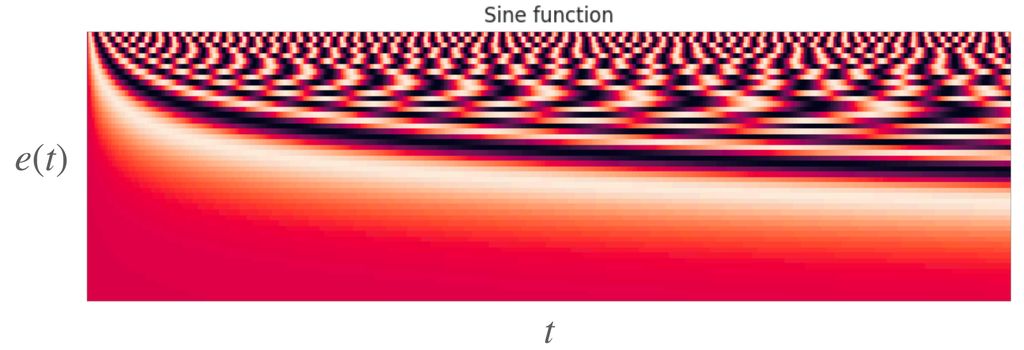
$$e(t) = \boldsymbol{E}_{t,:} := \begin{bmatrix} \sin\left(\frac{t}{f_1}\right) \\ \cos\left(\frac{t}{f_1}\right) \\ \sin\left(\frac{t}{f_2}\right) \\ \cos\left(\frac{t}{f_2}\right) \\ \vdots \\ \sin\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \\ \cos\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \end{bmatrix}, \quad f_m = \frac{1}{\lambda_m} := 10000^{\frac{2m}{d_{\text{model}}}}.$$

ACADENTIAL

출처: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

Attention

Sinusoidal Positional Representation



Sinusoidal Positional Representation

단점:

- Learnable 하지 않다 (즉, 고정되어 있음)
- t_{max} 보다 더 긴 sequence에 대해서도 extrapolate 가능할까? 실제로는 그렇지 않다.

$$e(t) = \boldsymbol{E}_{t,:} := \begin{bmatrix} \sin\left(\frac{t}{f_1}\right) \\ \cos\left(\frac{t}{f_1}\right) \\ \sin\left(\frac{t}{f_2}\right) \\ \cos\left(\frac{t}{f_2}\right) \\ \vdots \\ \sin\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \\ \cos\left(\frac{t}{f_{\frac{d_{\text{model}}}{2}}}\right) \end{bmatrix}, \quad f_m = \frac{1}{\lambda_m} := 10000^{\frac{2m}{d_{\text{model}}}}.$$

ACADENTIAL

출처: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

ACADENTIAL

Attention

Learned Positional Encoding

Positional Embedding $P \in \mathbb{R}^{H \times T}$

(H: feature dimension, T: max sequence length)

즉, 각 position t 에 대해서 $\mathbf{p}_t \in \mathbb{R}^H$ **embedding vector**을 정의한다!

이를 $\mathbf{x}_t$에 element-wise 더해준다!

Attention

Learned Positional Encoding

Positional Embedding $P \in \mathbb{R}^{H \times T}$

각 position t 에 대해서 $\mathbf{p}_t \in \mathbb{R}^H$ **embedding vector**을 정의한다! (가장 일반적인 positional encoding 방법)

장점:

- Learnable
- 유연하다

단점

- 1, …, T 외의 다른 position에 대해서는 extend하지 못한다

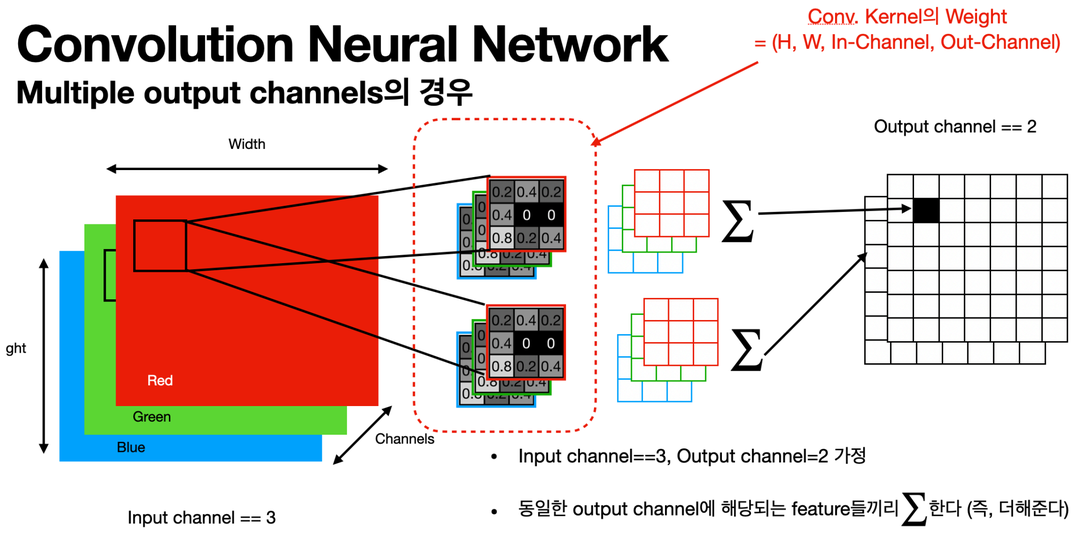
Attention

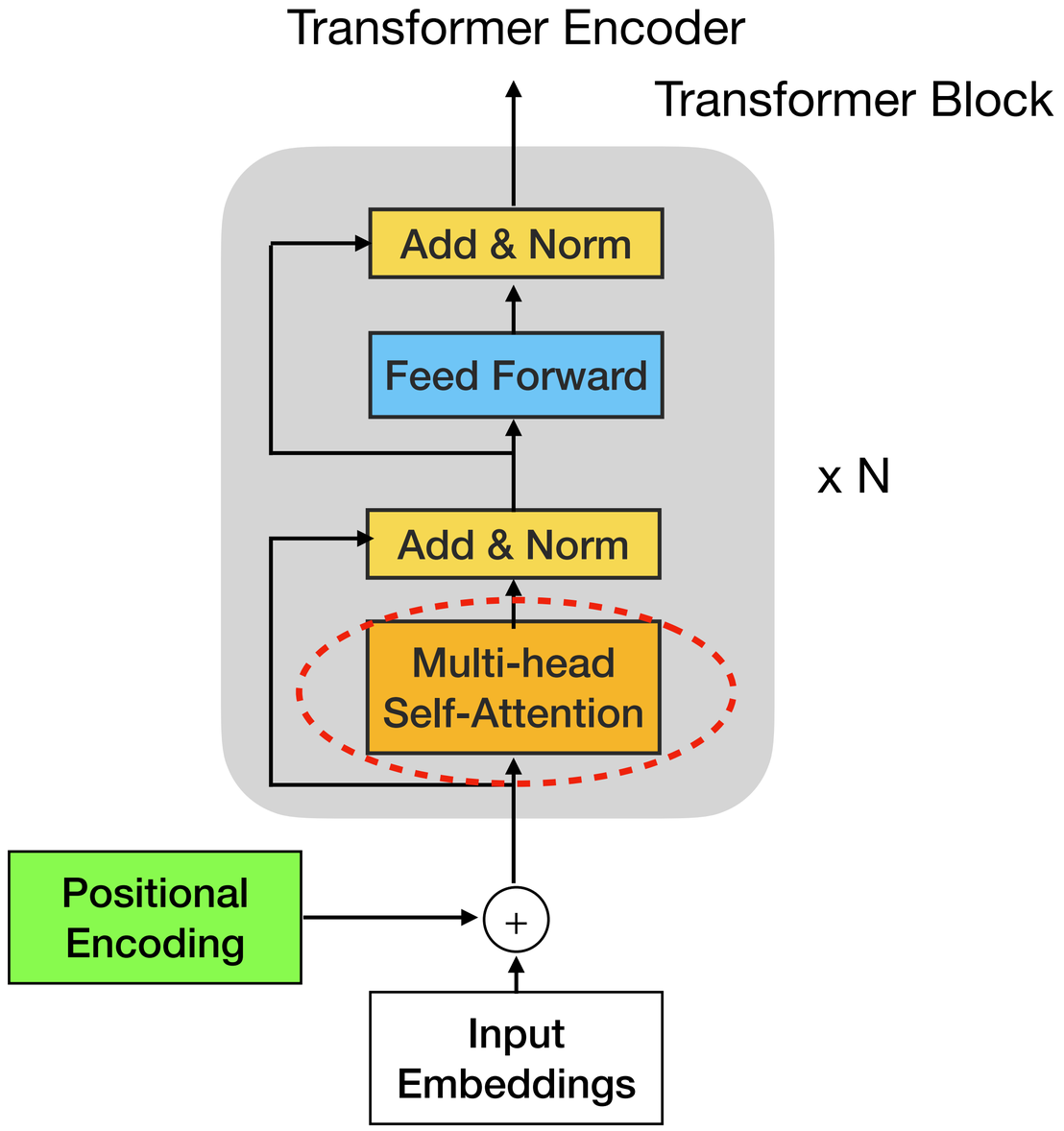
Multi-head attention

Multi-head attention

CNN에서 multi-channel과 유사한 개념이다.

Multi-head attention은 여러 개의 self-attention을 서로 stack 시킨 것이다!

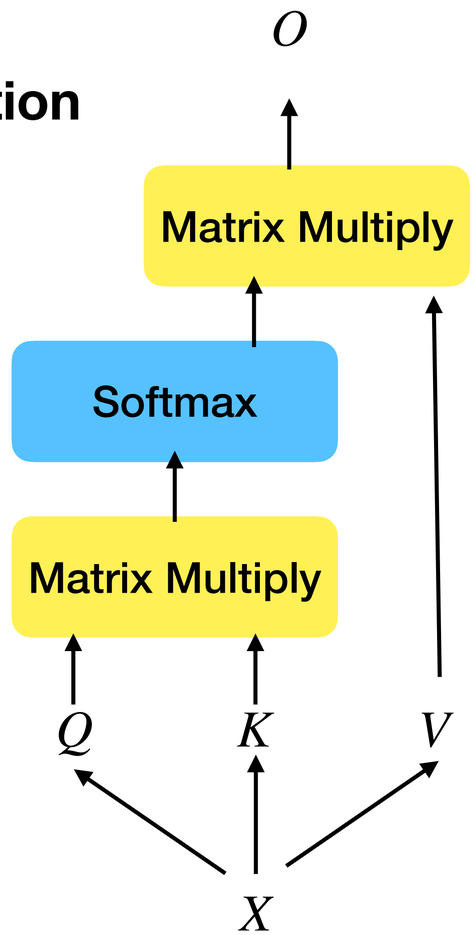




Attention

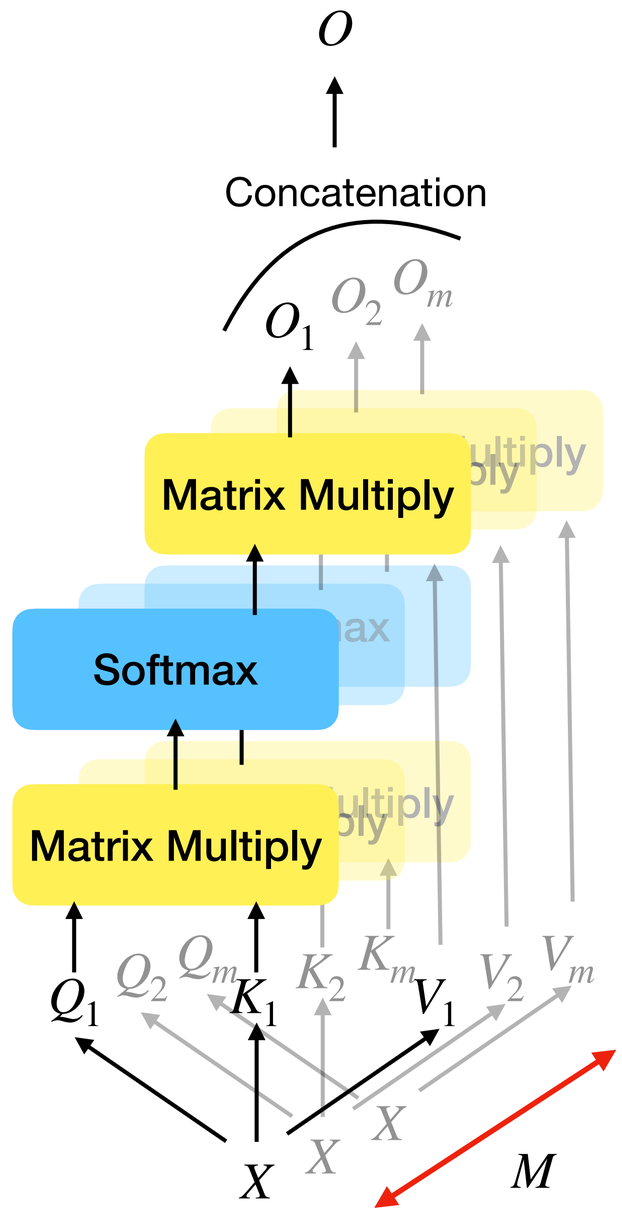
Multi-head attention

Self-attention



Multi-head attention

(head 개수 = M)



Attention

Multi-head attention

즉, M개의 독립적인 query, key, vector들을 따로 둔다:

$$Q_m, K_m, V_m \in \mathbb{R}^{N \times \frac{H}{m}}$$

따라서 총 M개의 attention map을 구하게 된다:

$$A_m \in \mathbb{R}^{N \times N} \quad \text{for m in [1, …, M]}$$

각 head 마다 output O_m을 구함:

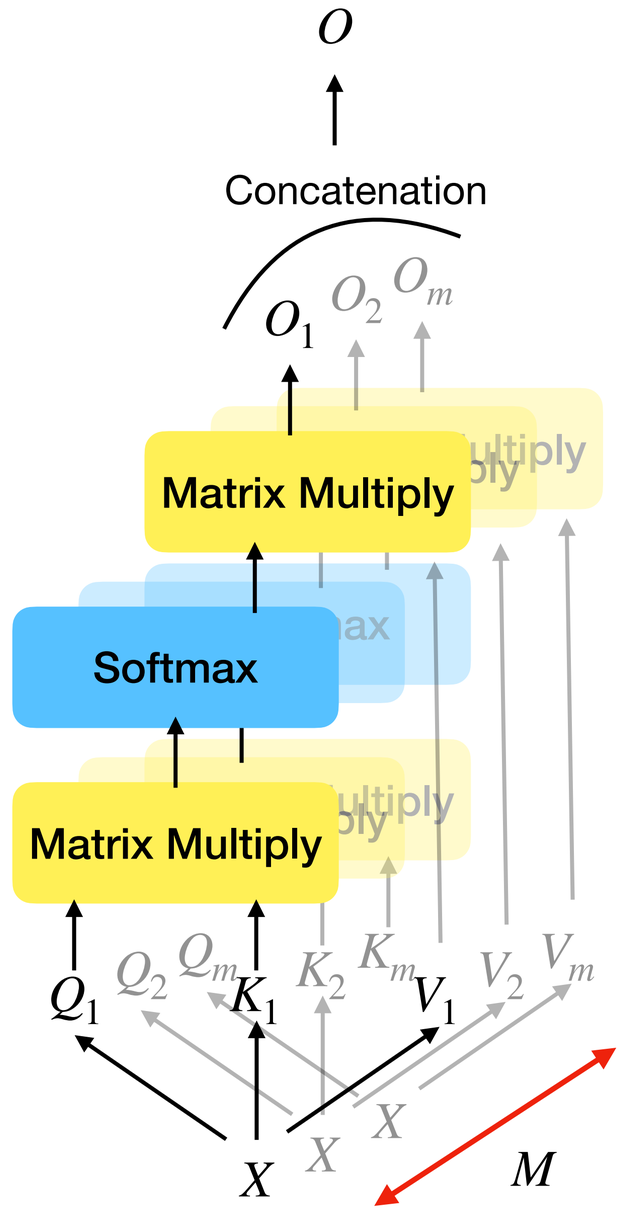
$$O_m \in \mathbb{R}^{N \times \frac{H}{m}}$$

O_m을 concatenation해서 최종 출력값을 구함:

$$O \in \mathbb{R}^{N \times H}$$

Multi-head attention

(head 개수 = M)



ACADENTIAL

Attention

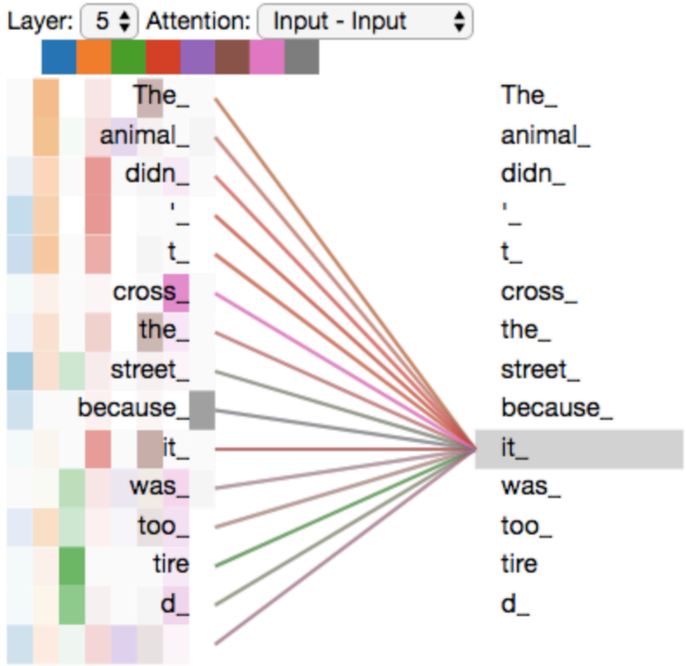
Multi-head attention

총 M개의 attention map을 구하게 된다:

$$A_m \in \mathbb{R}^{N \times N} \quad \text{for m in [1, …, M]}$$

단어간의 상호 연관성이 다양할때 여러 단어들을 동시에 고려해야 할 수 있다!

Multi-head attention은 각 head가 독립적으로 attention을 구하기 때문에 다양한 상호 연관성을 포착할 수 있다!



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

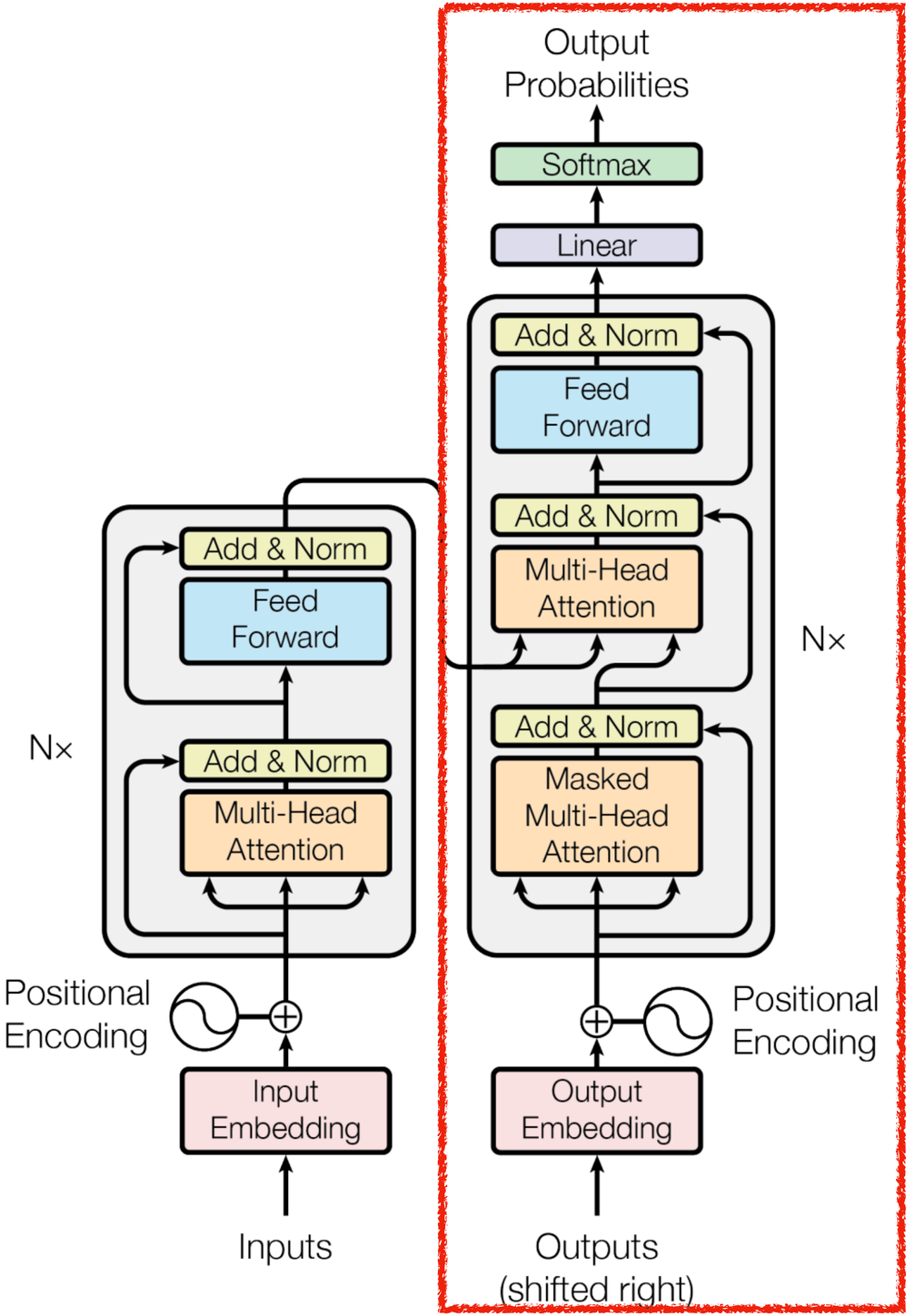
ACADENTIAL

16-8. Transformer의 Decoder

Attention

Transformer Decoder

- Transformer 모델의 구조:
  - Encoder
  - Decoder
- 앞서서 Encoder에 대해서 살펴보았다!
- 이제 Decoder에 대해서 살펴보자!

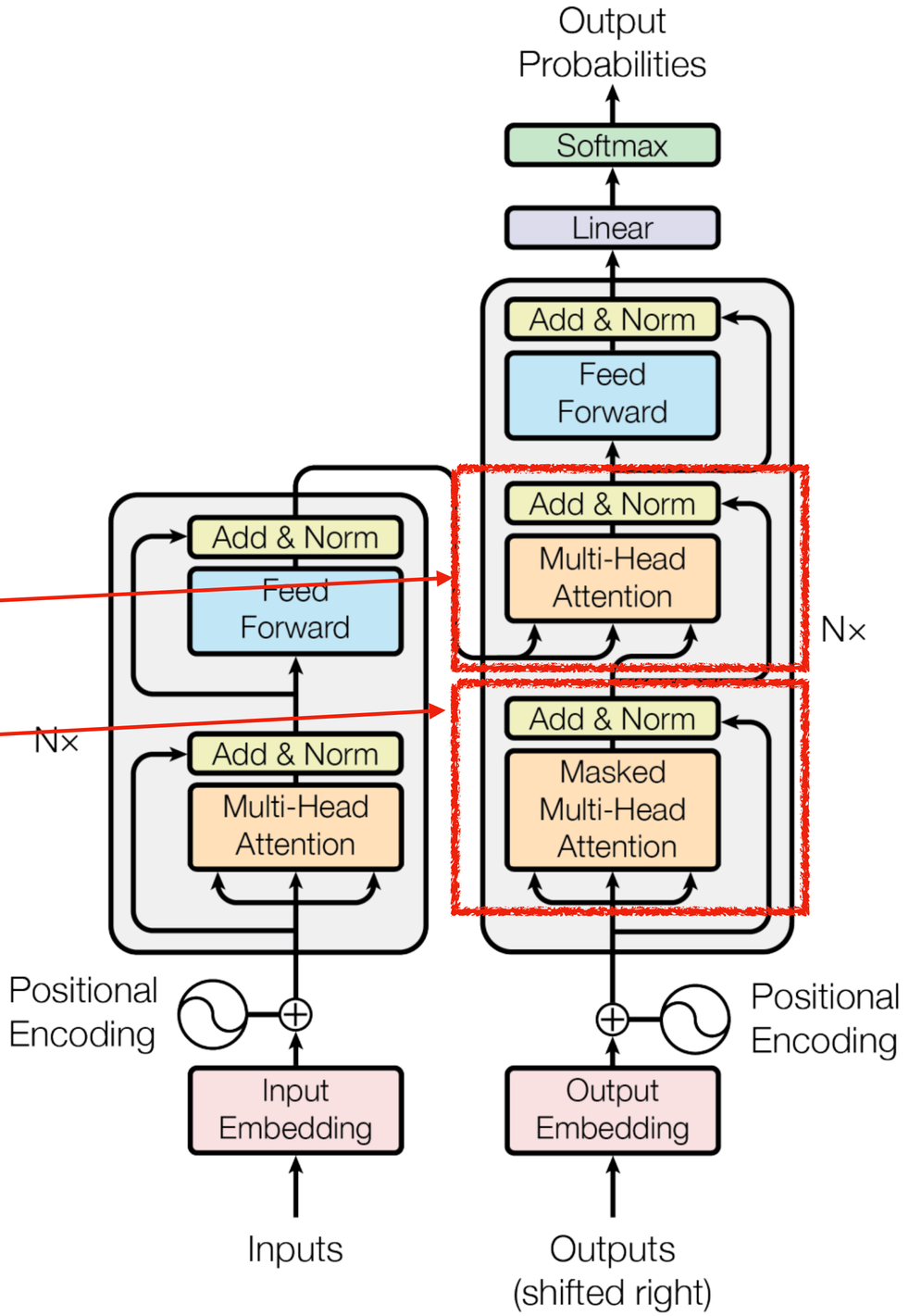


ACADENTIAL

Attention

Transformer Decoder

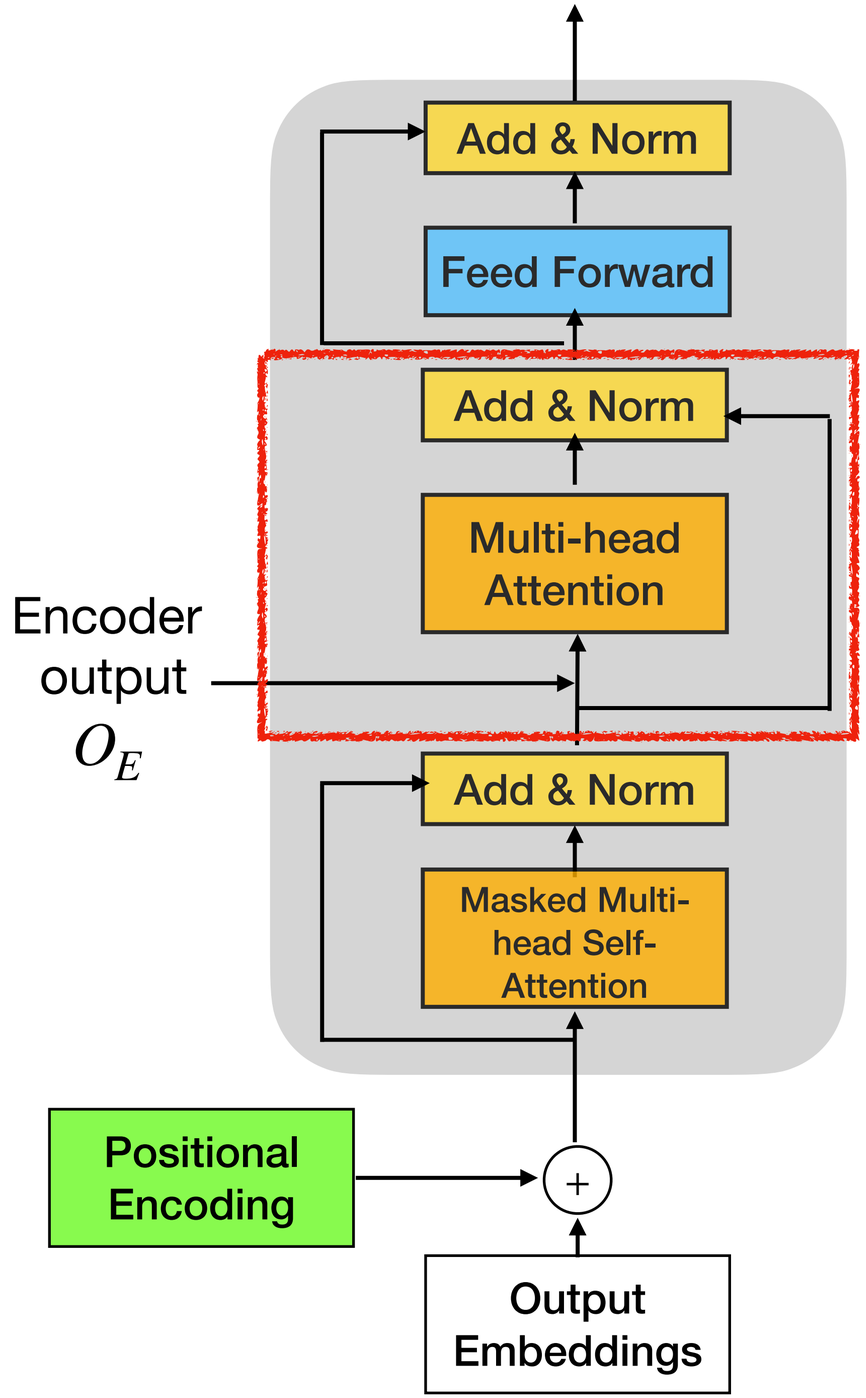
- Decoder은 다음 두 가지 부분에서 Encoder와 차이를 가진다:
  - Multi-head Attention
  - Masked Multi-head (Self) Attention



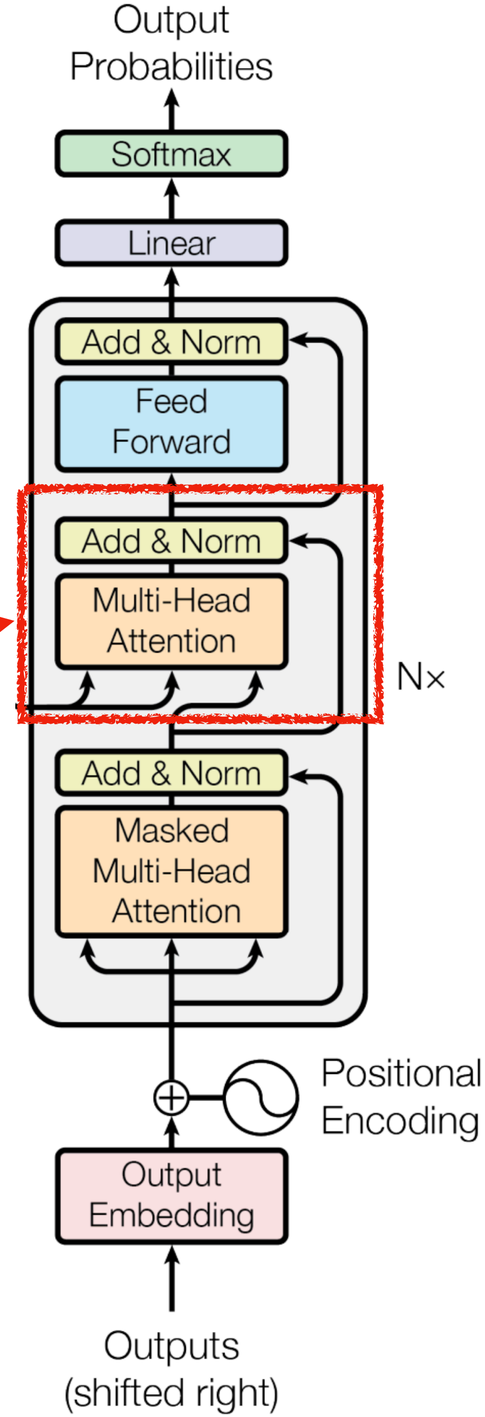
ACADENTIAL

Transformer Decoder

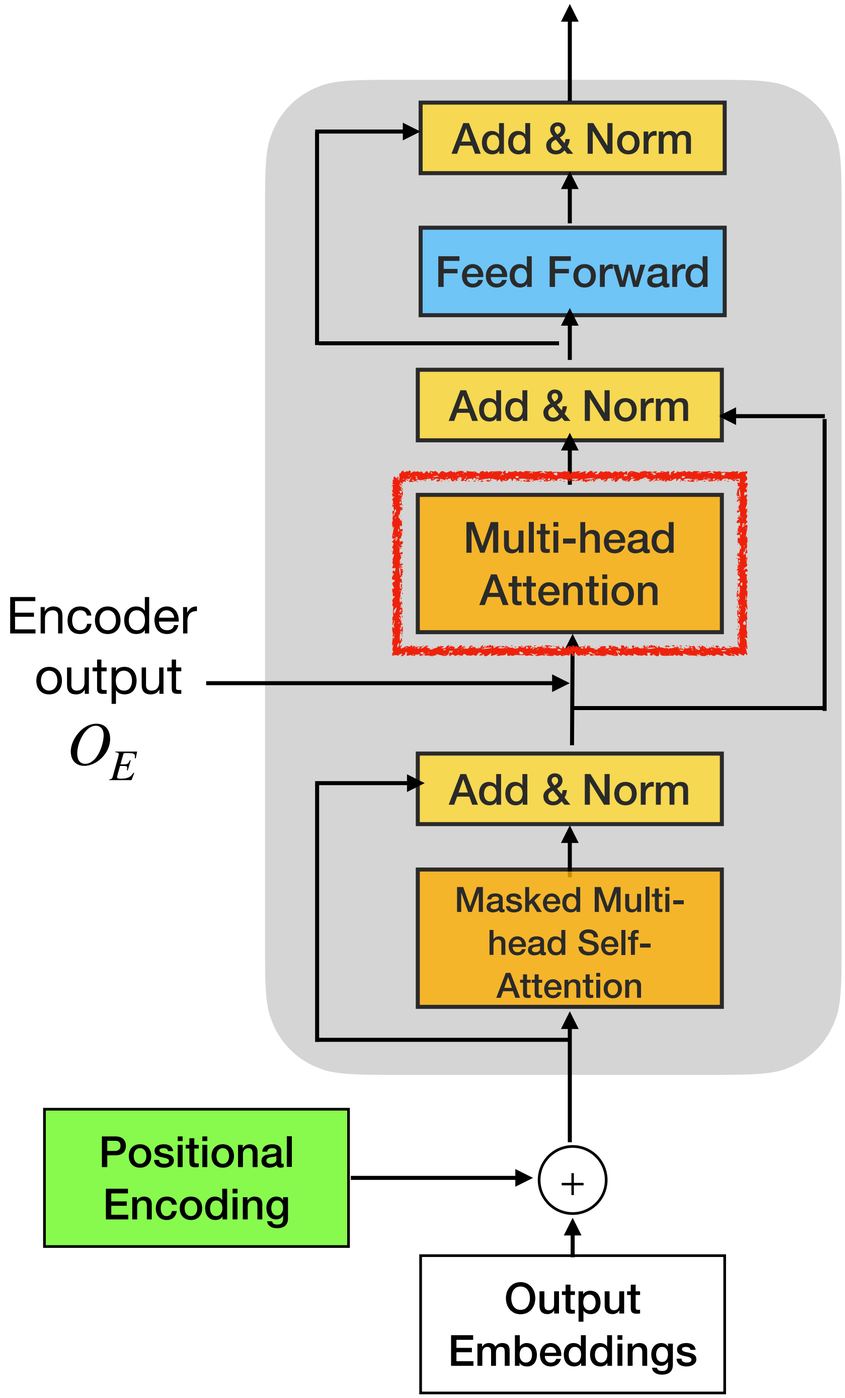
Transformer
Decoder Block



먼저 Multi-head attention을
자세히 살펴보자!



ACADENTIAL



- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T$$

$$K = O_E(W_k)^T$$

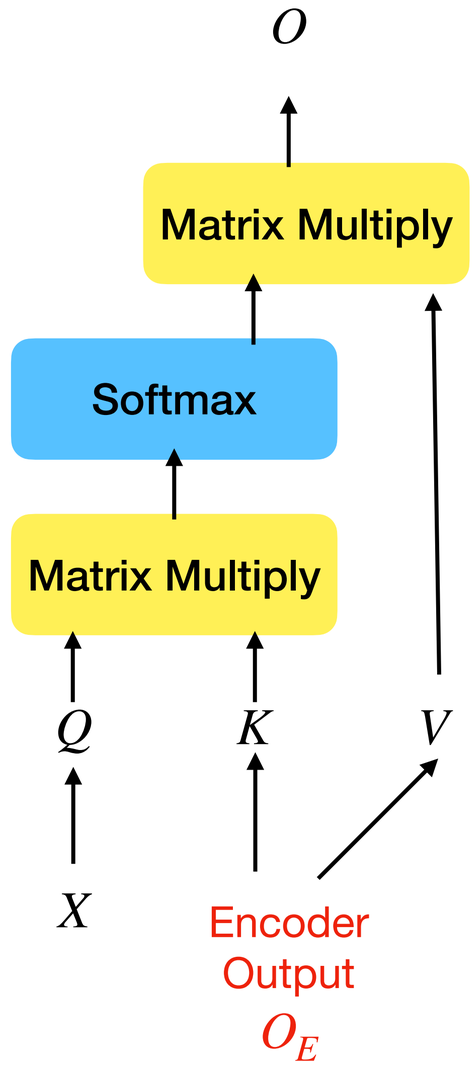
$$V = O_E(W_v)^T$$

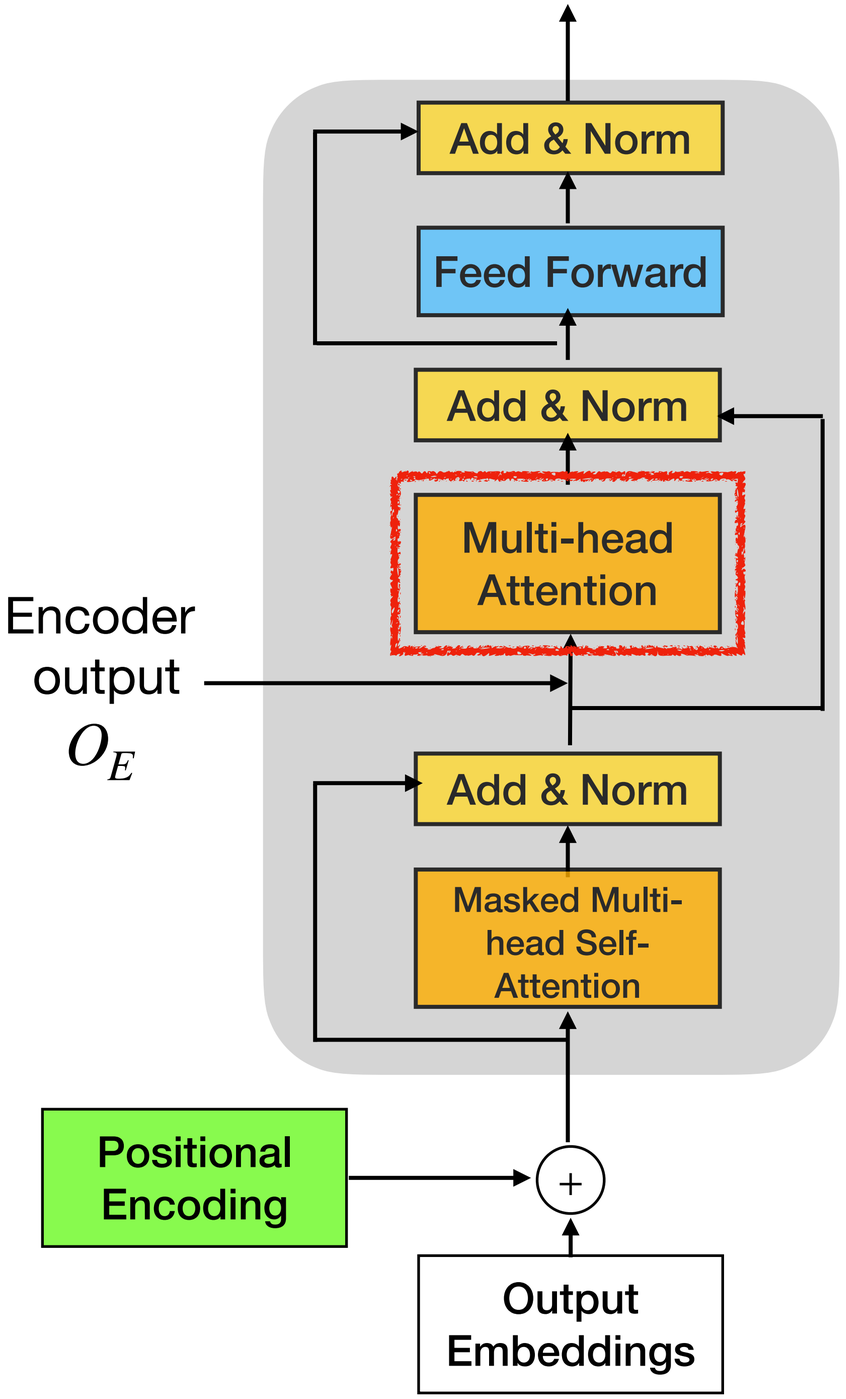
- 참고로, Encoder의 self-attention의 Query, Key, Value은

$$Q = X(W_q)^T$$

$$K = X(W_k)^T$$

$$V = X(W_v)^T$$





참고로 다른 layer의 출력값으로 **Key, Value**을 구하기 때문에

→ **Cross Attention**으로도 불린다!

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T$$

$$K = O_E(W_k)^T$$

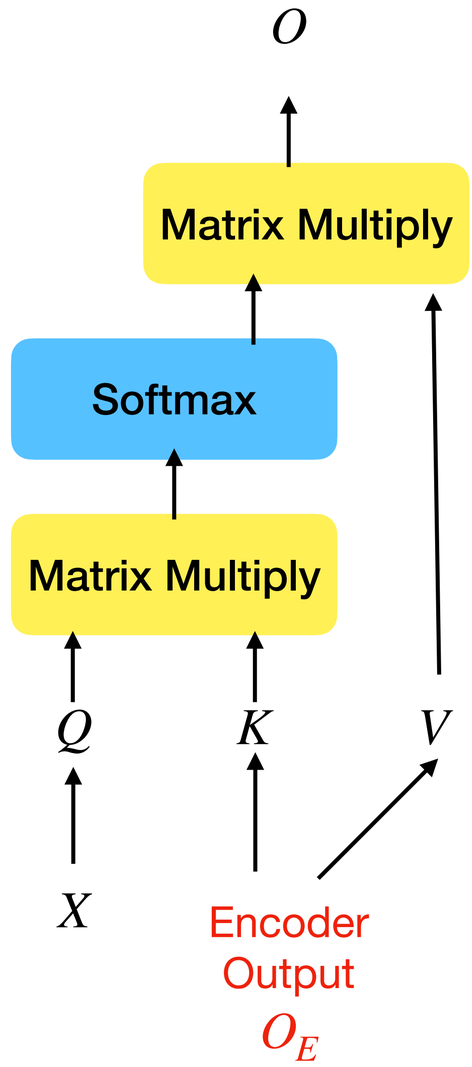
$$V = O_E(W_v)^T$$

- 참고로, Encoder의 self-attention의 Query, Key, Value은

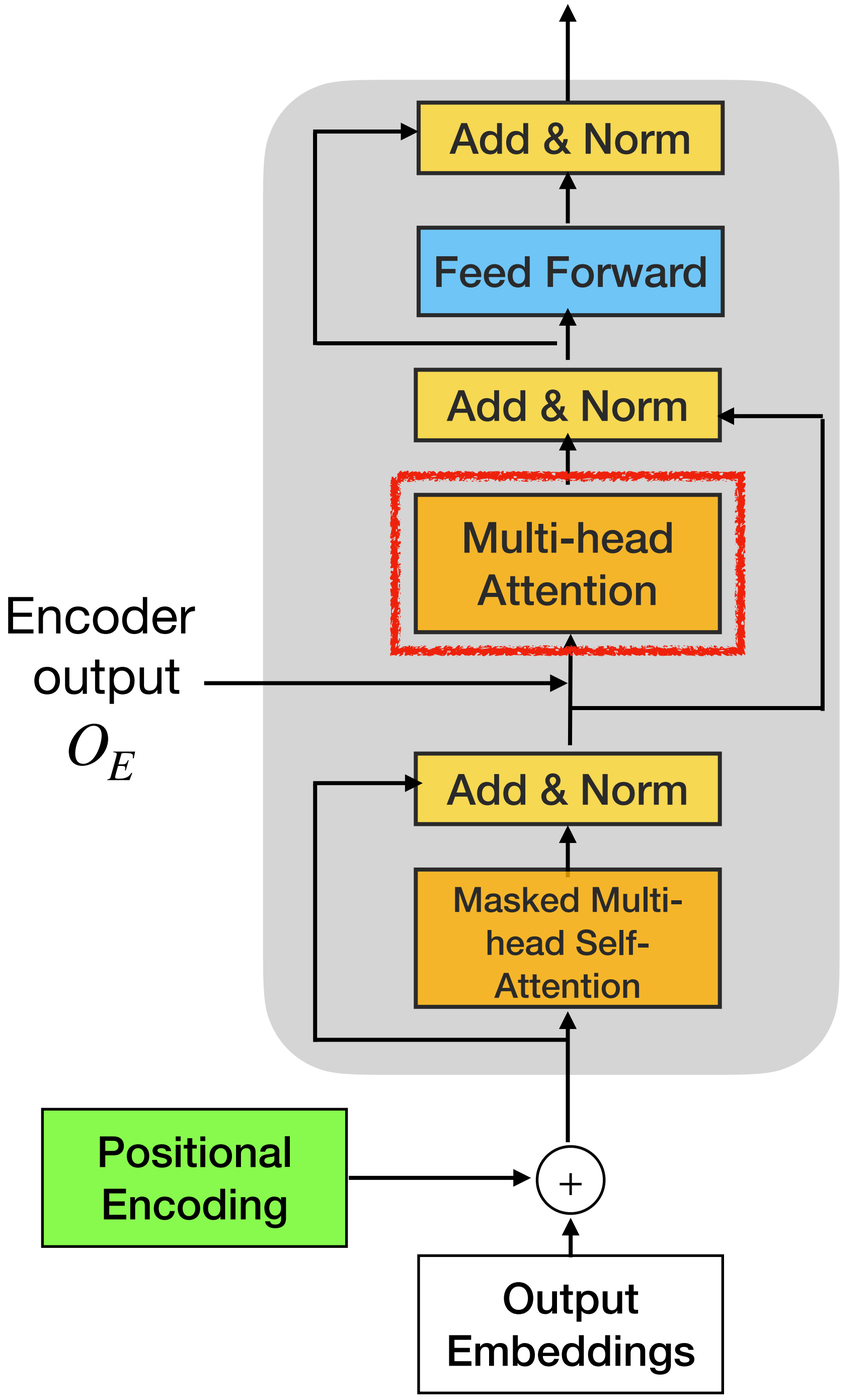
$$Q = X(W_q)^T$$

$$K = X(W_k)^T$$

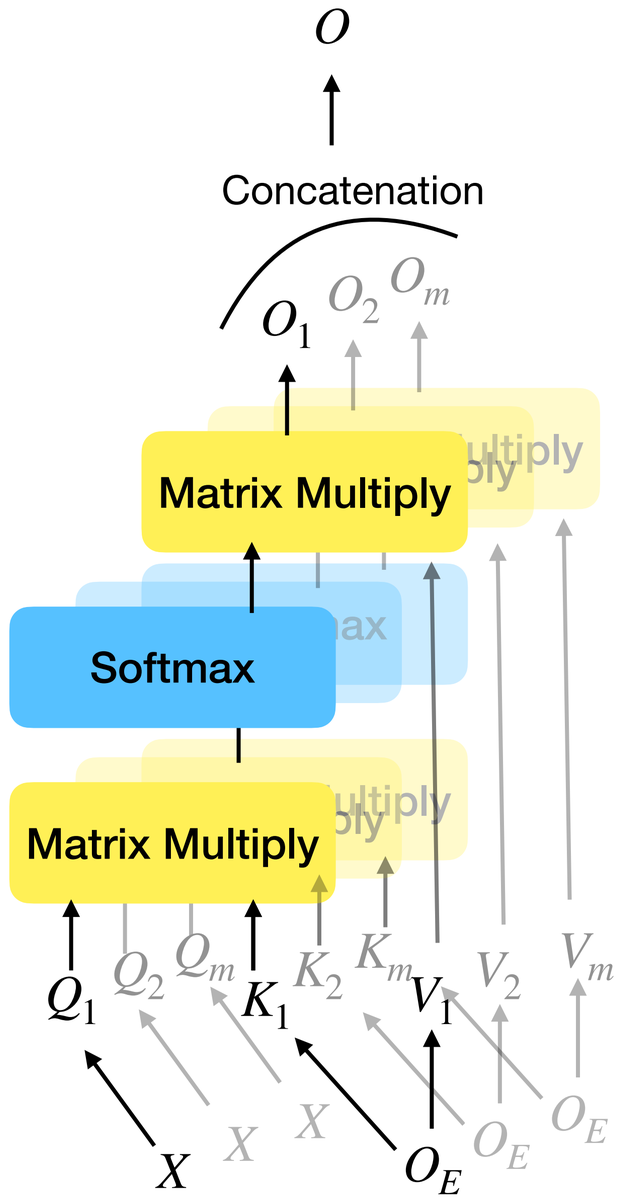
$$V = X(W_v)^T$$



ACADENTIAL



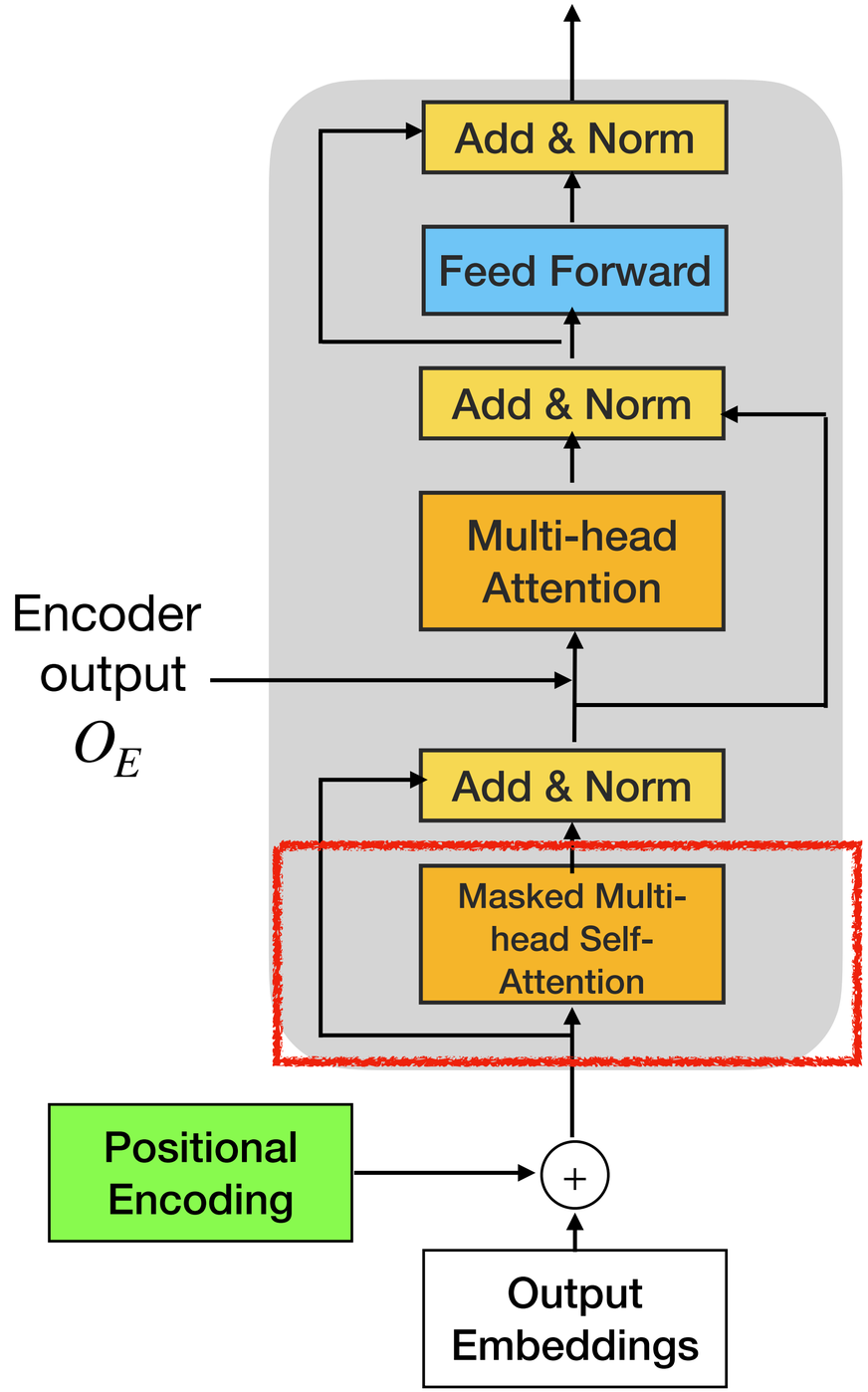
- 각 Head의 output O_m을 concatenate해서 최종 출력값을 구한다!



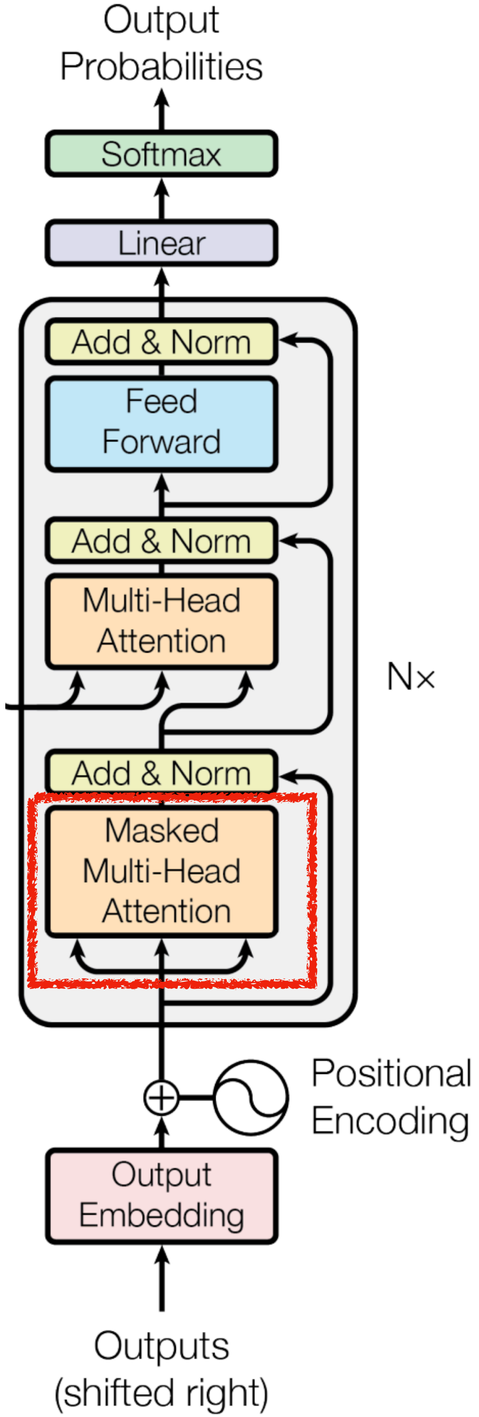
ACADENTIAL

Transformer Decoder

Transformer
Decoder Block

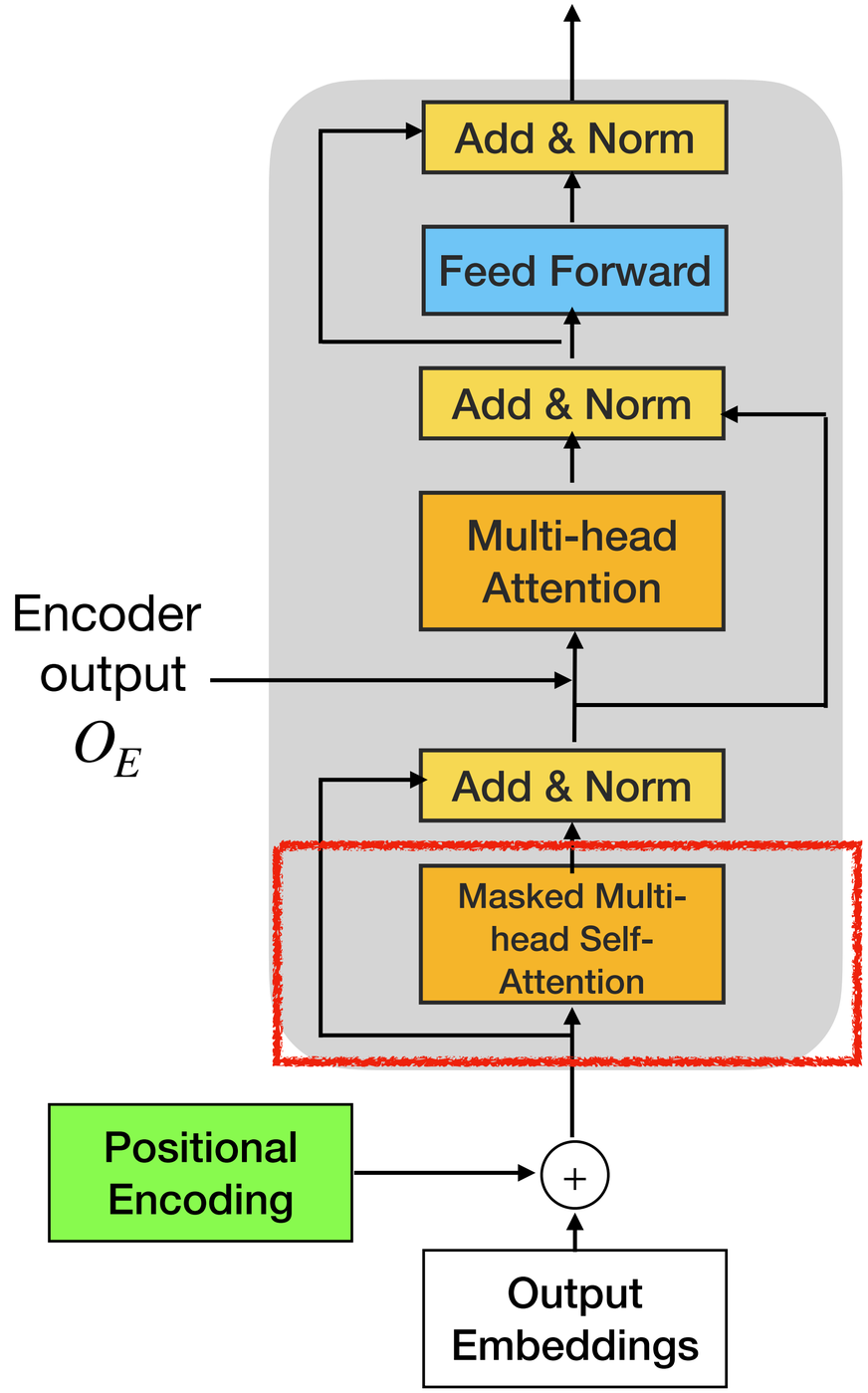


이어서 Masked Multi-head self-attention을 살펴보자



ACADENTIAL

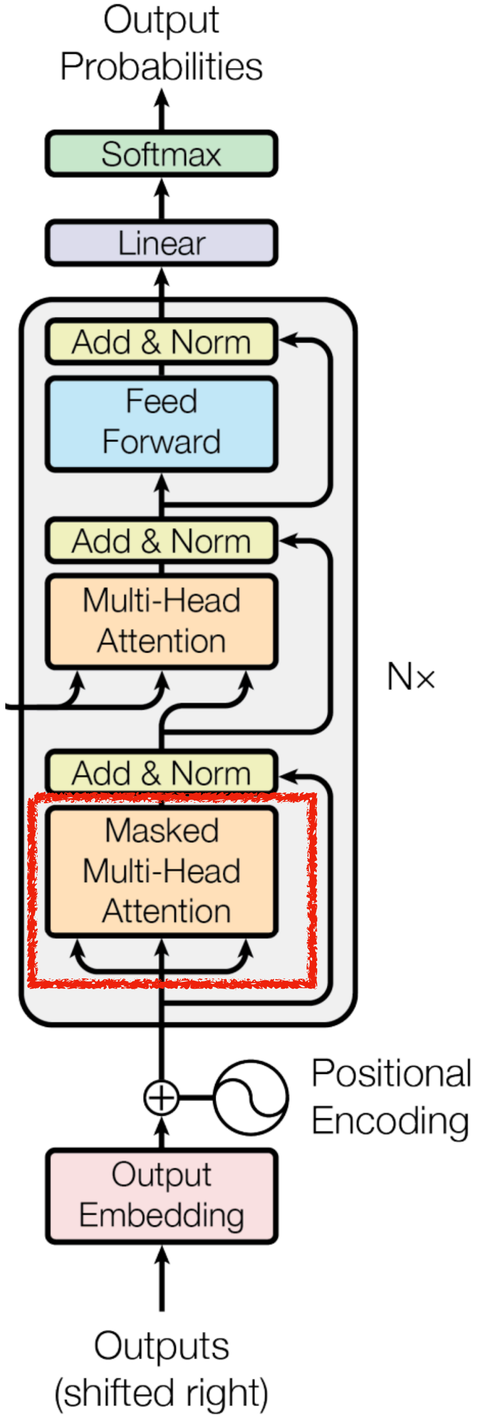
Transformer Decoder



Transformer Decoder Block

하지만, 이것에 앞서서 **Transformer 모델의 forward pass**을 먼저 살펴보자.

(Transformer 모델이 Machine Translation 문제를 어떻게 풀었는지)



ACADENTIAL

ACADENTIAL

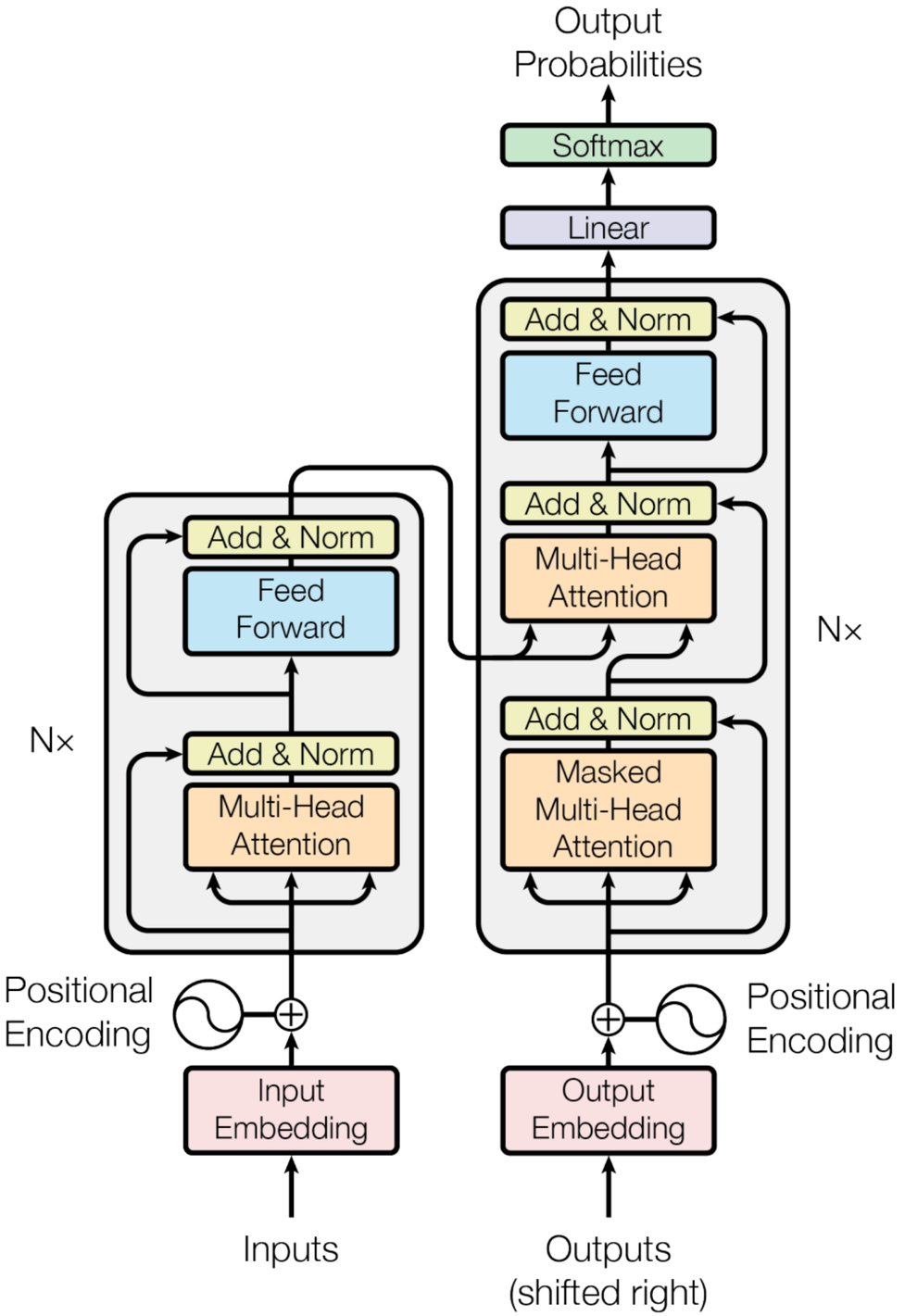
16-8. Teacher Forcing

Attention

Transformer - Forward pass

Recap:

- Transformer은 Machine Translation (번역)을 풀고자 함.
  - Input = source 언어
  - Output = target 언어



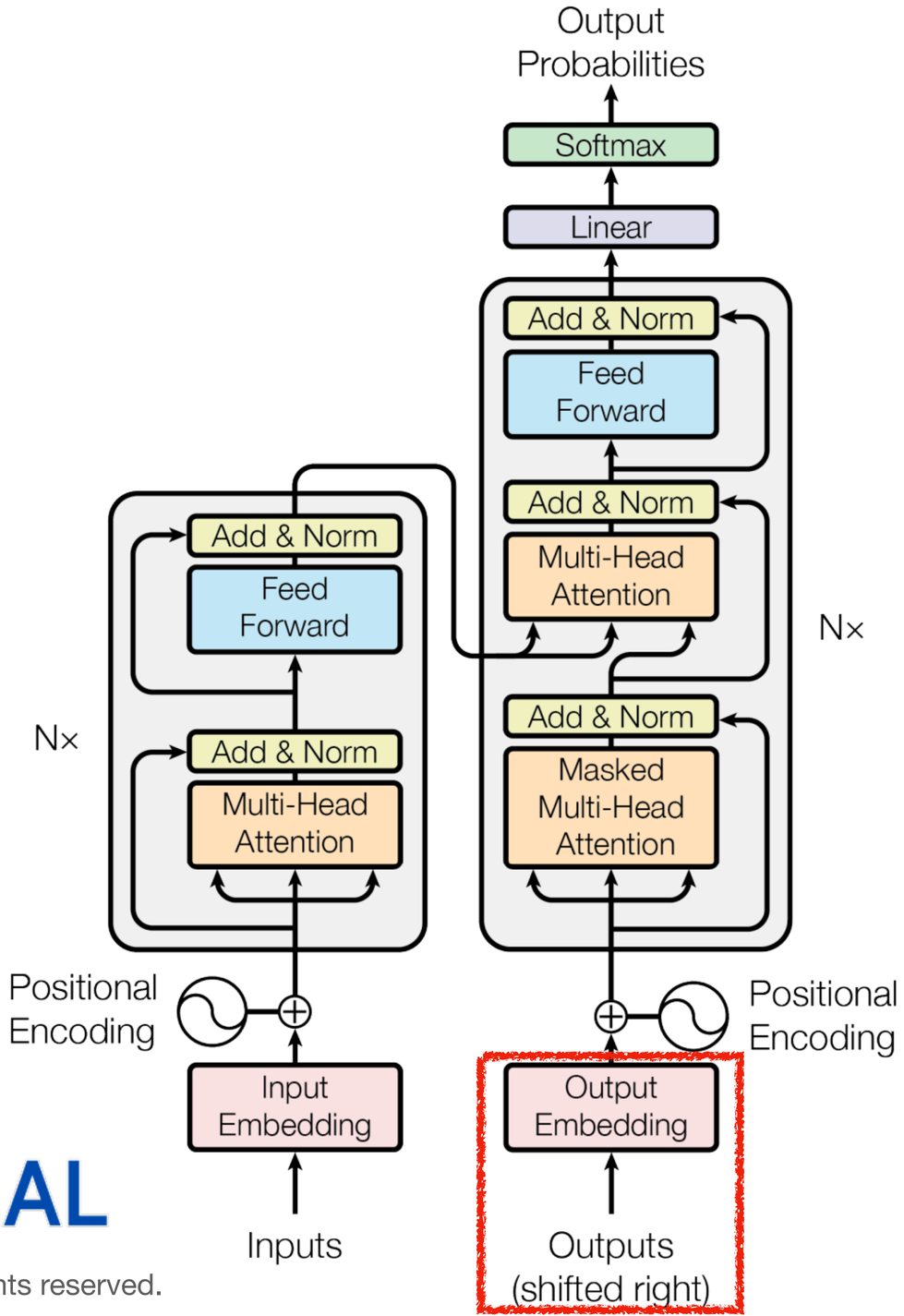
ACADENTIAL

Attention

Transformer - Forward pass

Recap:

- Transformer은 Machine Translation (번역)을 풀고자 함.
  - Input = source 언어
  - Output = target 언어
- 여기서 의아한 점:
  - 왜 Output이 Decoder의 input으로 입력되는걸까?



ACADENTIAL

Attention

Teacher forcing

- 여기서 의아한 점:
  - 왜 Output이 Decoder의 input으로 입력되는걸까?
- 이유:
  - 학습할때 Teacher forcing을 사용해서!
  - 추론할때 예측한 단어들을 다시 입력값으로 사용해서!
- Teacher forcing은 뭘까?

Attention

Teacher forcing

- Teacher forcing에서는
  - 모델이 Ground Truth (정답값)에 해당되는 이전 $t = (1, \cdots, i-1)$의 단어들에 의거해서 $t = i$ 번째 target 단어를 예측한다.
  - 예를 들어, $1 \leq t < i$ 의 단어를 모델이 틀리게 예측했어도
  - $t = i$ 번째 target 단어 예측에 필요한
  - 이전 target token들은Ground Truth 값을 그대로 사용한다는 의미.

Attention

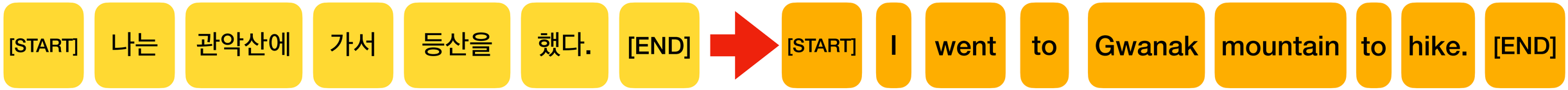
Teacher forcing

- Teacher forcing에서는
  - 모델이 Ground Truth (정답값)에 해당되는 이전 $t = (1, \cdots, i-1)$의 단어들에 의거해서 $t = i$ 번째 target 단어를 예측한다.
  - 예를 들어, $1 \leq t < i$ 의 단어를 모델이 틀리게 예측했어도
  - $t = i$ 번째 target 단어 예측에 필요한
  - 이전 target token들은Ground Truth 값을 그대로 사용한다는 의미.
  - (이게 무슨 말일까… ㅎㅎ)

ACADENTIAL

Attention

Teacher forcing



Input
(source sentence)

Output
(target sentence)
Ground Truth

위 경우를 살펴보자.

ACADENTIAL

Attention

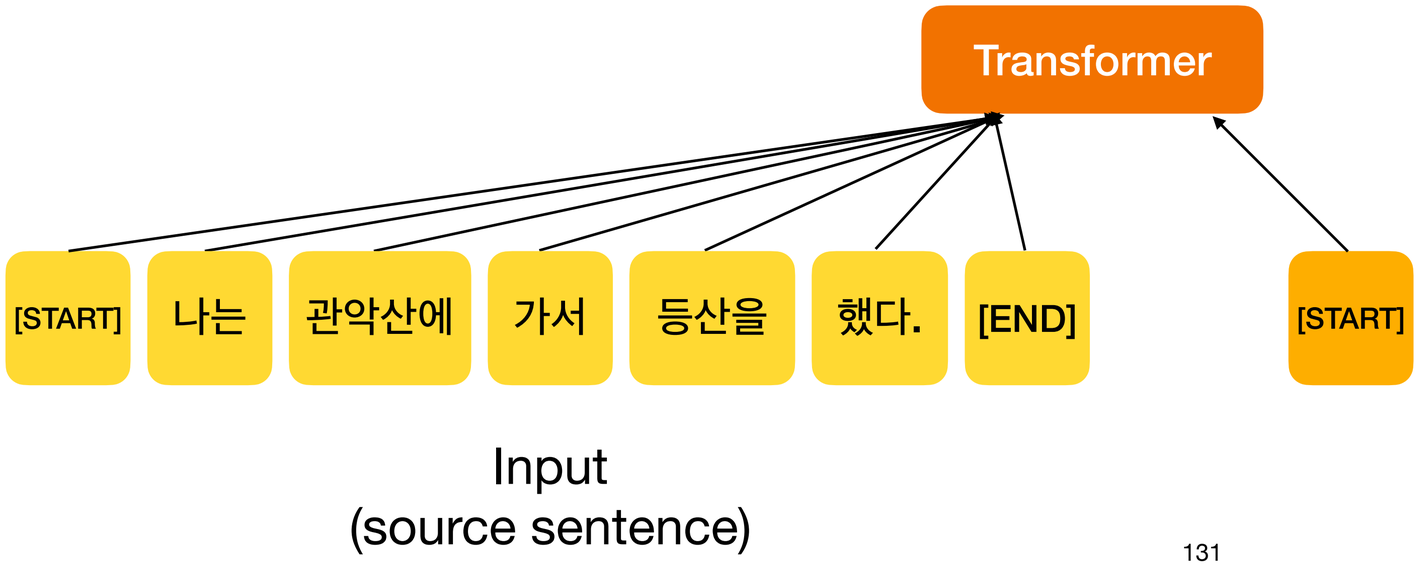
Teacher forcing

Ground Truth
(target sentence)

I | went | to | Gwanak | mountain | to | hike. | [END]

Time step:

$t = 1$



Input
(source sentence)

Attention

Teacher forcing

Ground Truth
(target sentence)

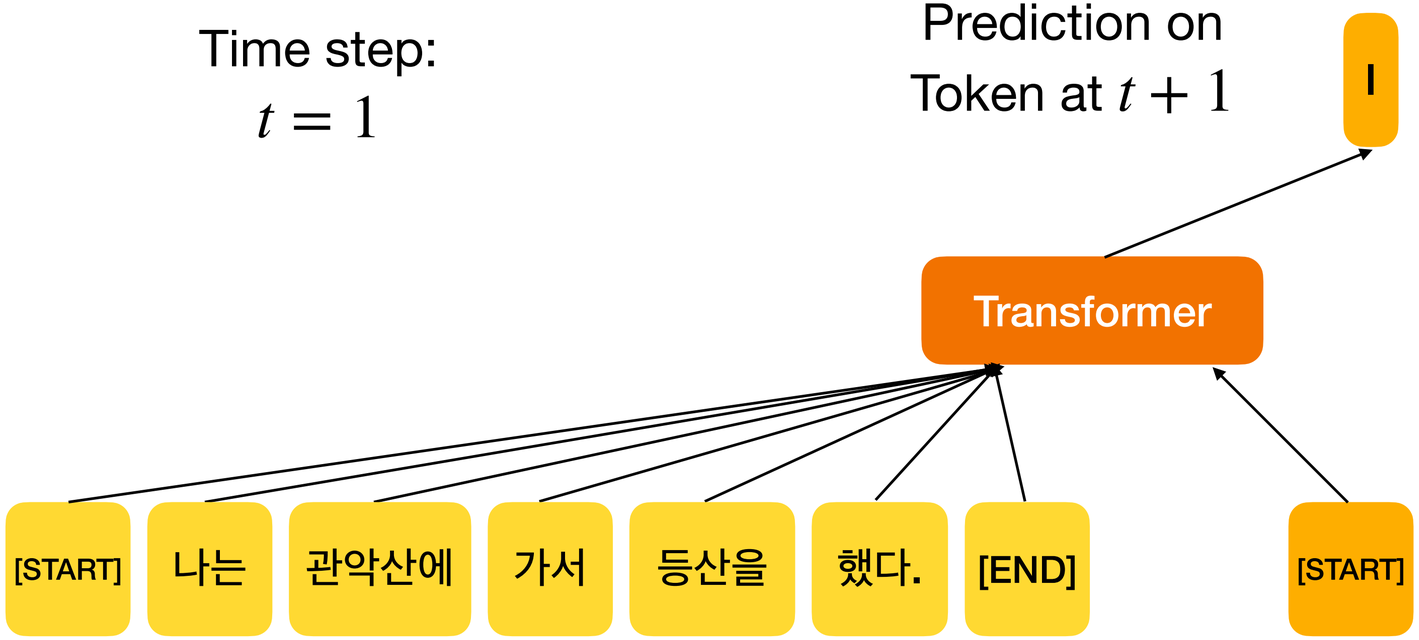
I | went | to | Gwanak | mountain | to | hike. | [END]

Time step:
$t = 1$

Prediction on
Token at $t + 1$

I

Correct
Prediction



Input
(source sentence)

이전 time step ($\leq t$) 에 대한 GT output

Attention

Teacher forcing

Ground Truth
(target sentence)

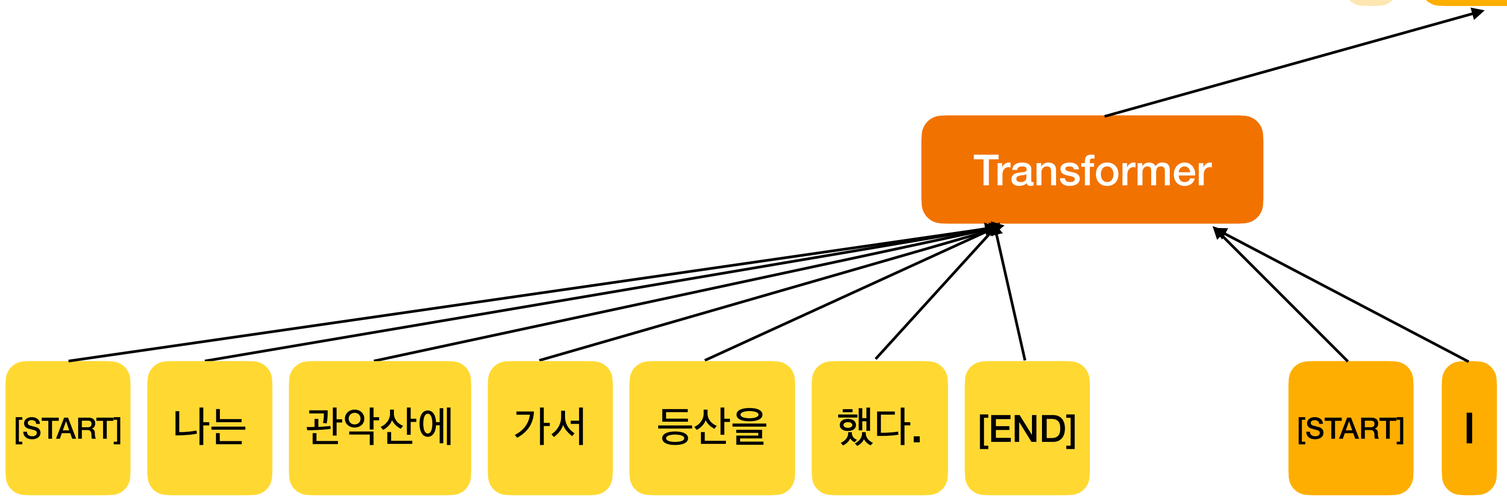
I | went | to | Gwanak | mountain | to | hike. | [END]

Time step:
$t = 2$

Prediction on
Token at $t + 1$

I | moved

Wrong
Prediction

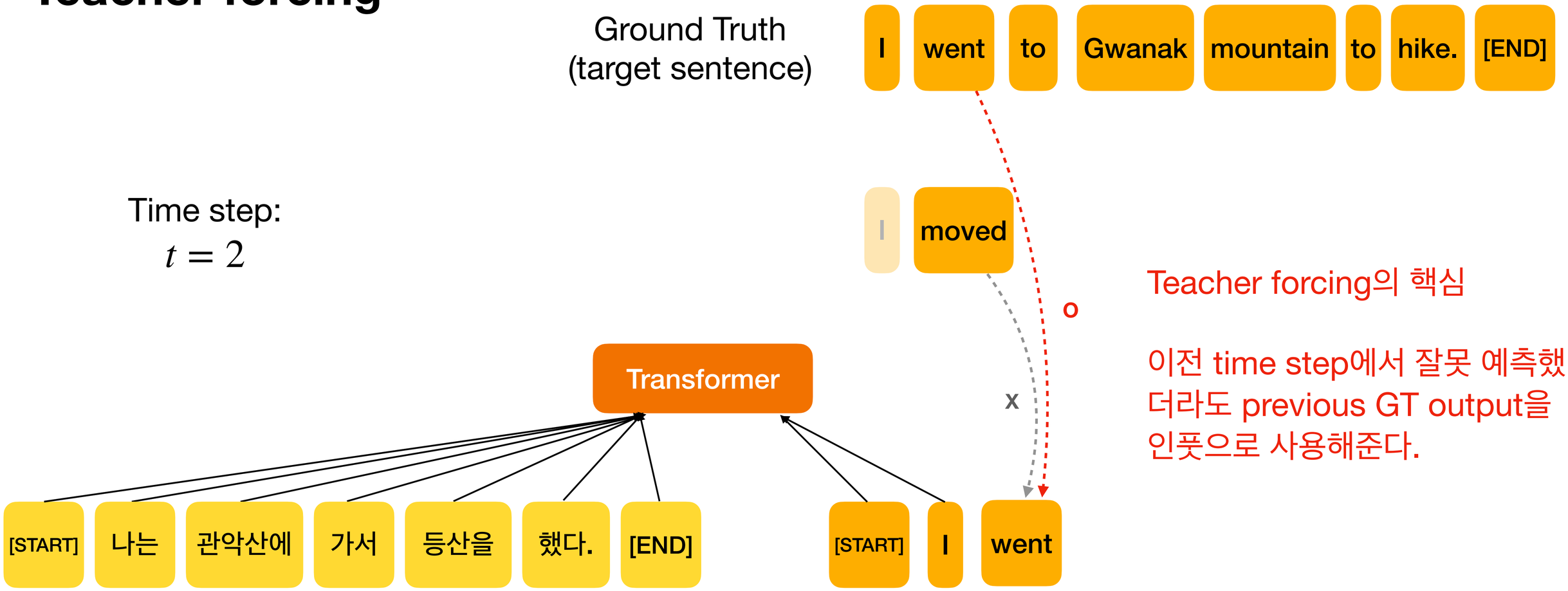


Input
(source sentence)

이전 time step ($\leq t$) 에 대한 GT output

Attention

Teacher forcing



Teacher forcing의 핵심

이전 time step에서 잘못 예측했더라도 previous GT output을 인풋으로 사용해준다.

Attention

Teacher forcing

Ground Truth
(target sentence)

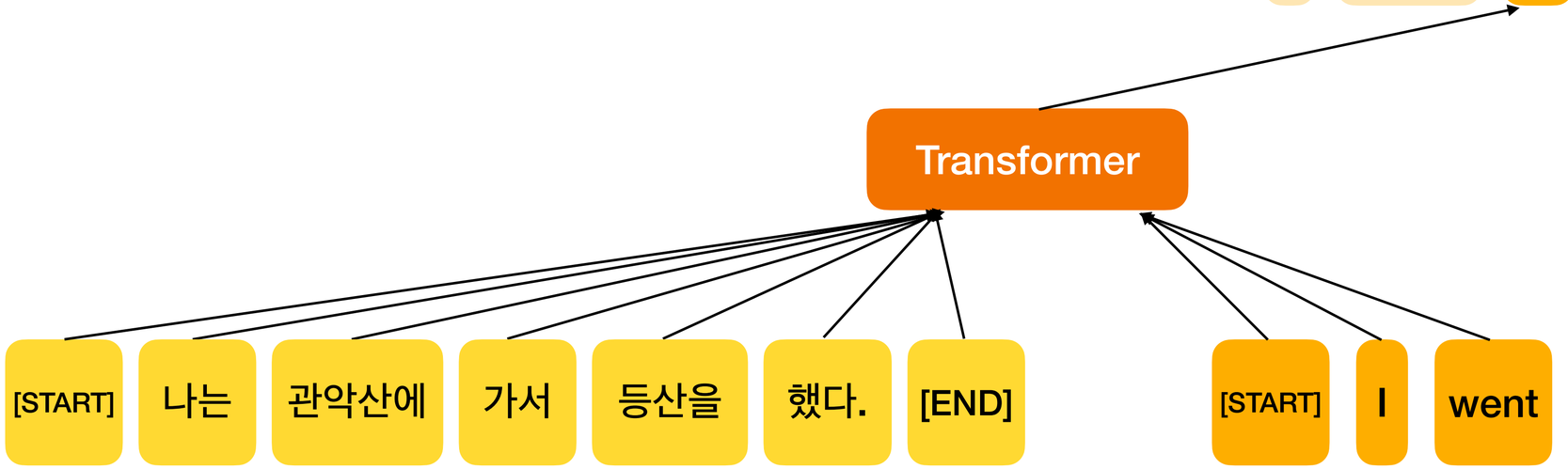
I | went | to | Gwanak | mountain | to | hike. | [END]

Time step:
$t = 3$

Prediction on
Token at $t + 1$

I | moved | to

Correct
Prediction



Input
(source sentence)

이전 time step ($\leq t$) 에 대한 GT output

Attention

Teacher forcing

Ground Truth
(target sentence)

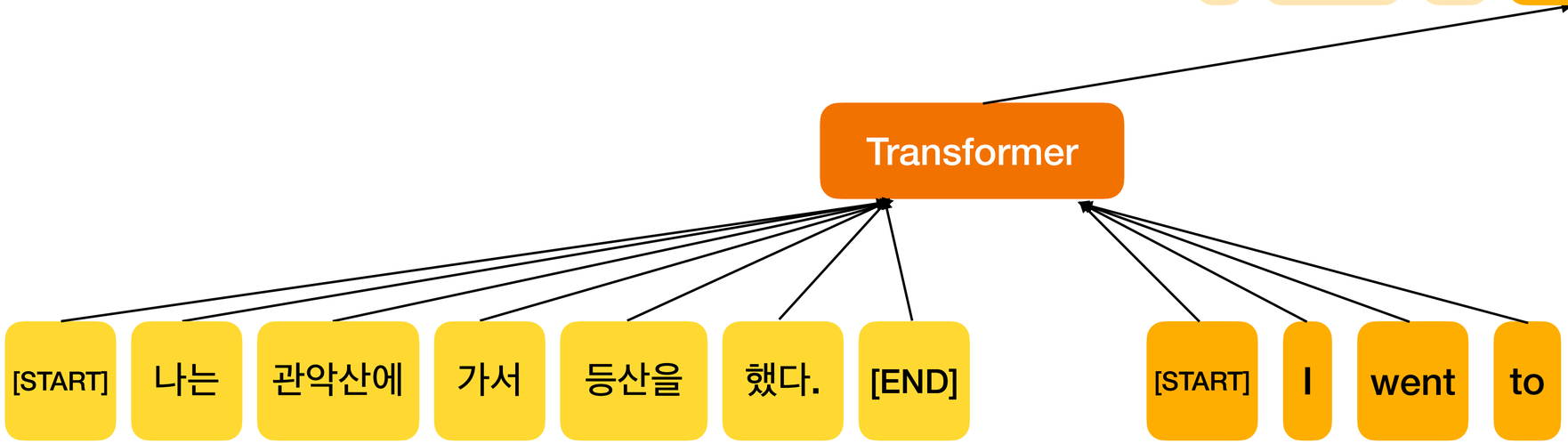
I | went | to | Gwanak | mountain | to | hike. | [END]

Time step:
$t = 4$

Prediction on
Token at $t + 1$

I | moved | to | Bukhan

Wrong
Prediction

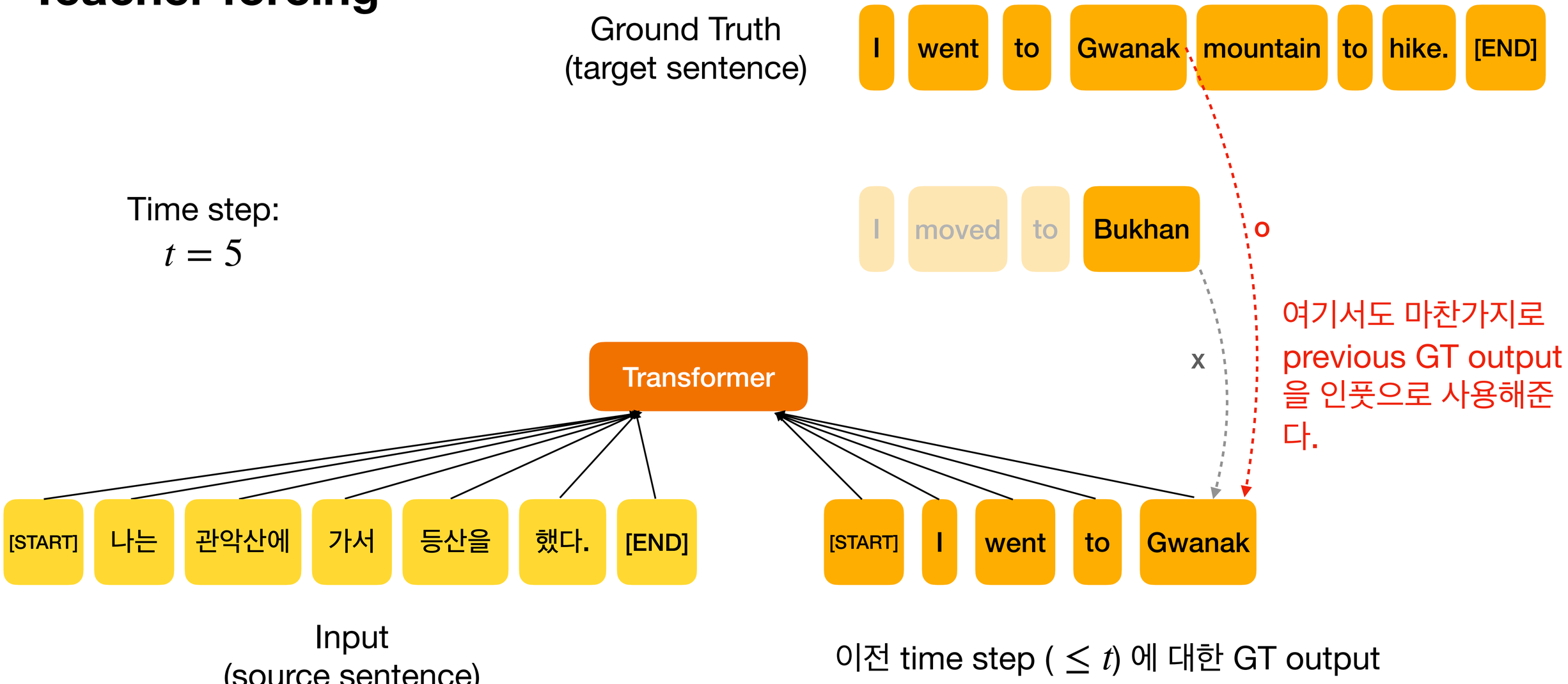


Input
(source sentence)

이전 time step ($\leq t$) 에 대한 GT output

Attention

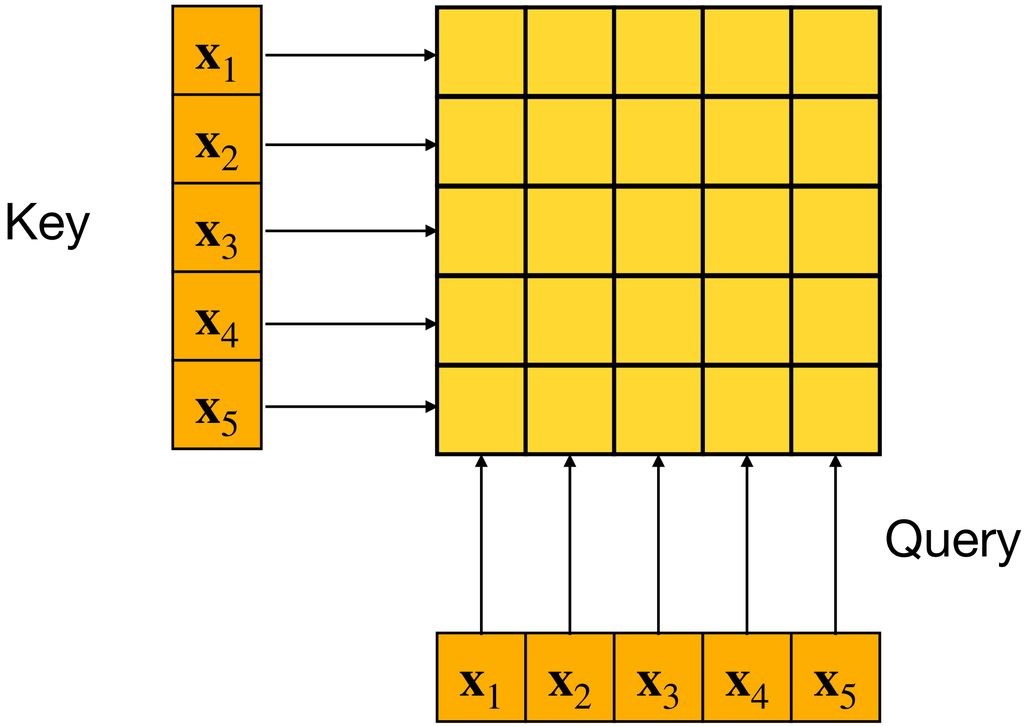
Teacher forcing

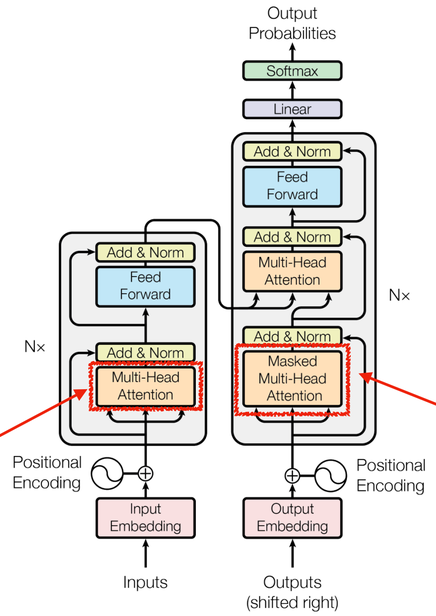


ACADENTIAL

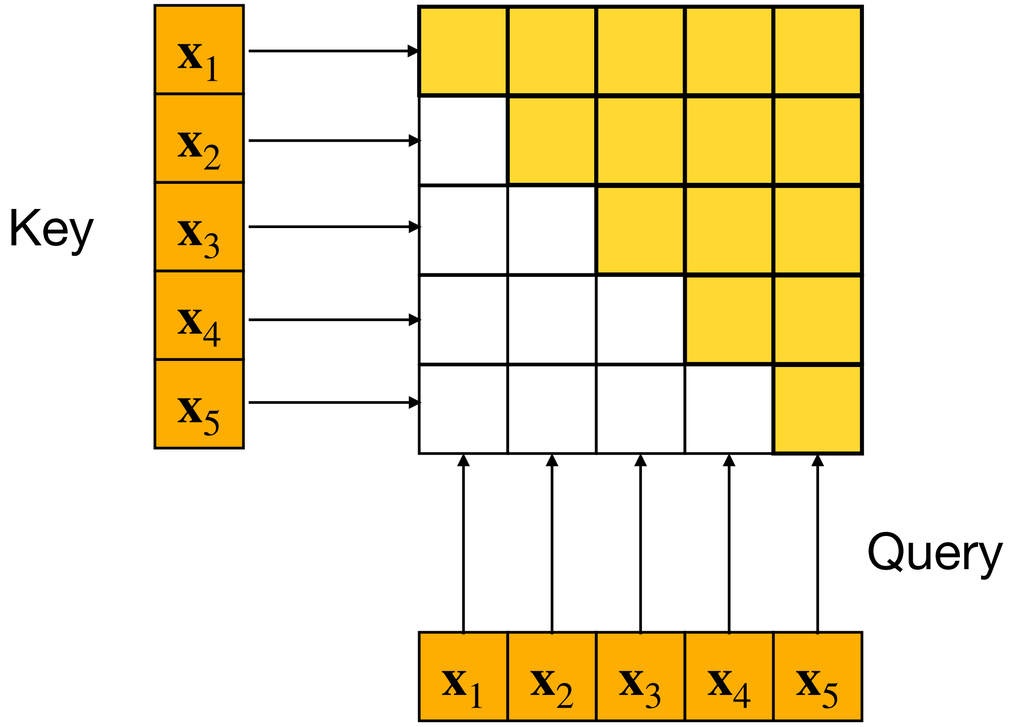
16-9. Masked Attention

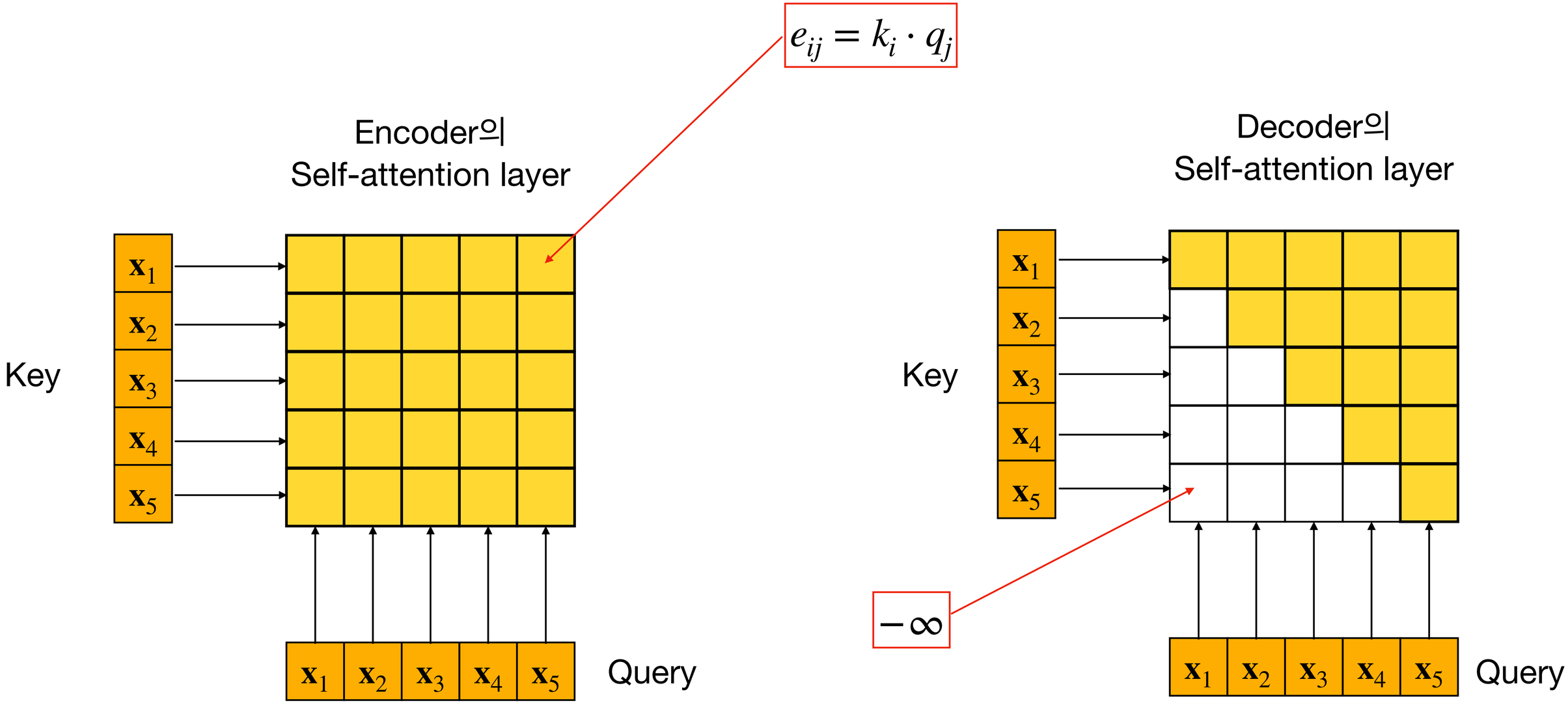
Encoder의
Self-attention layer



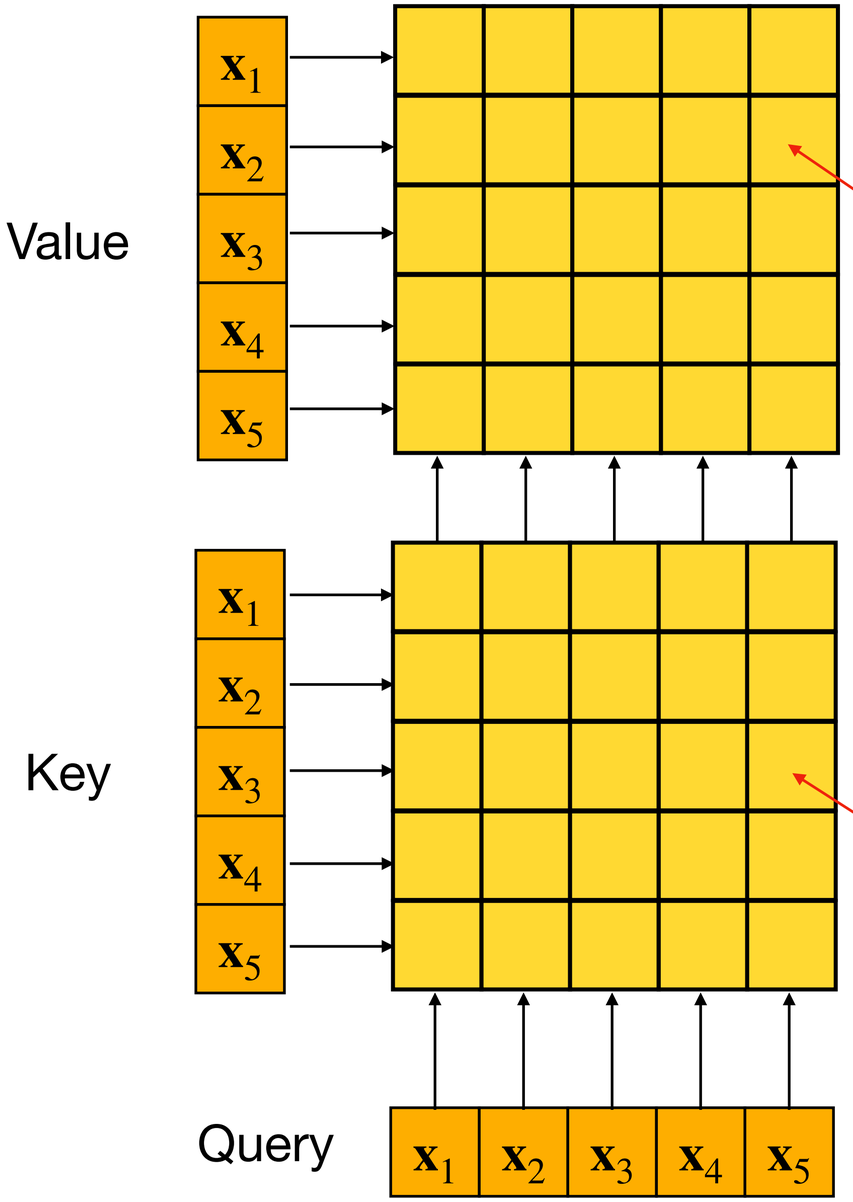


Decoder의
Self-attention layer





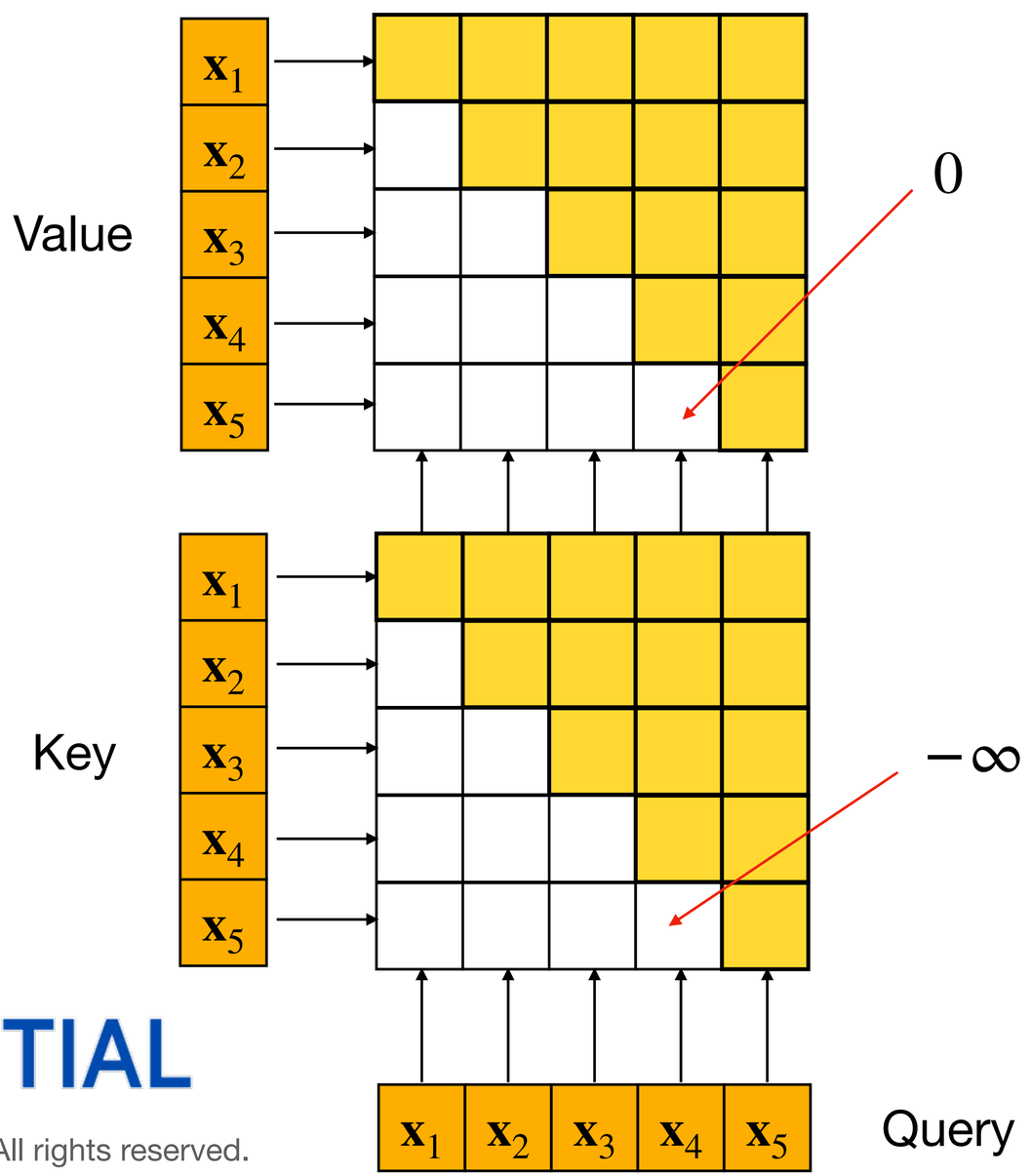
Encoder의
Self-attention layer



$$a_{ij}v_i = \frac{\exp(e_{ij})}{\sum_{j'} e_{ij}} v_i$$

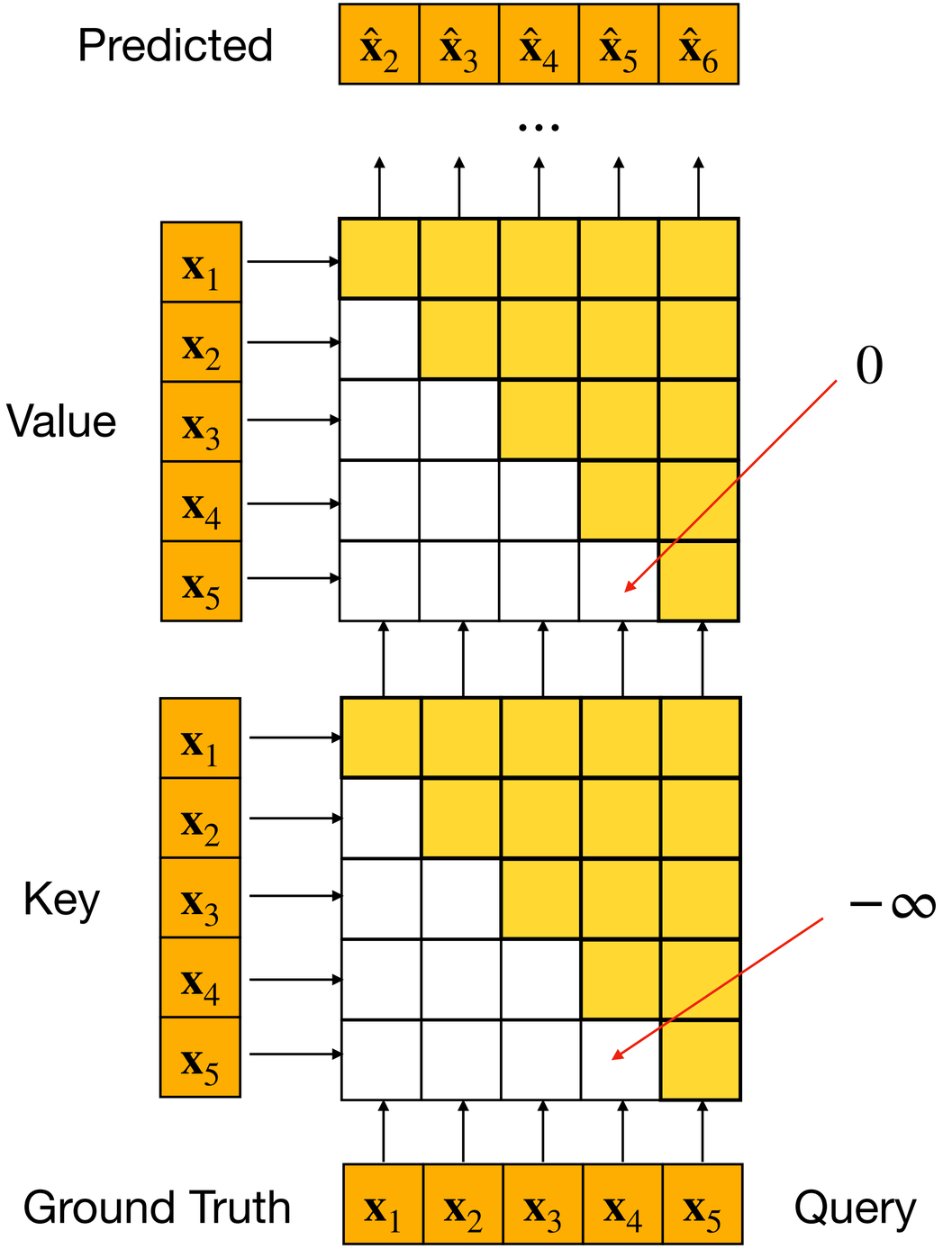
$$e_{ij} = k_i \cdot q_j$$

Decoder의
Self-attention layer



ACADENTIAL

왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

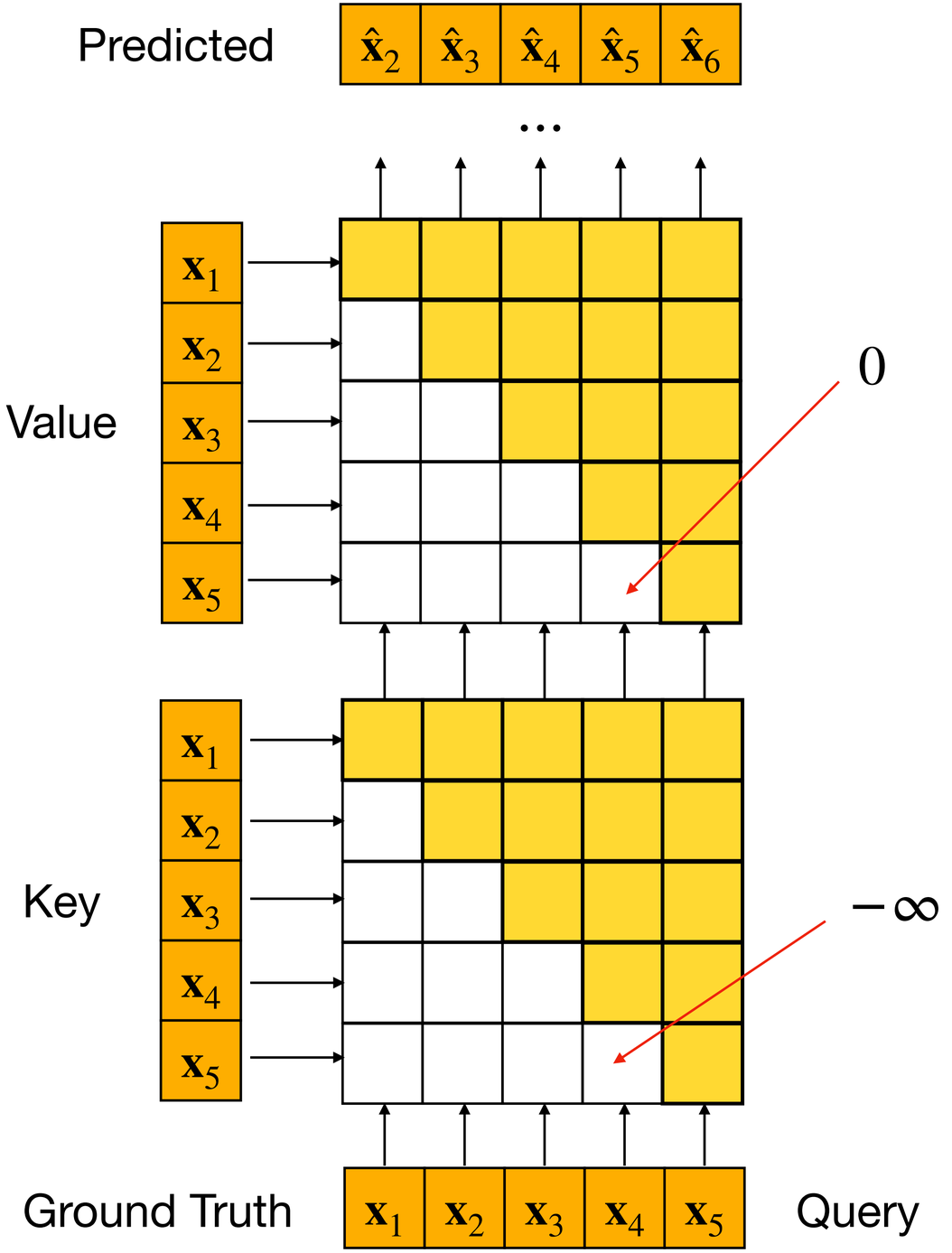


ACADENTIAL

왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

왜냐하면, 예측하고자 하는 token 혹은 아직 예측하지 않은 token의 값을 참고하는 것을 피하기 위해서다!

예를 들어서 $t = 1$의 경우를 살펴보자.

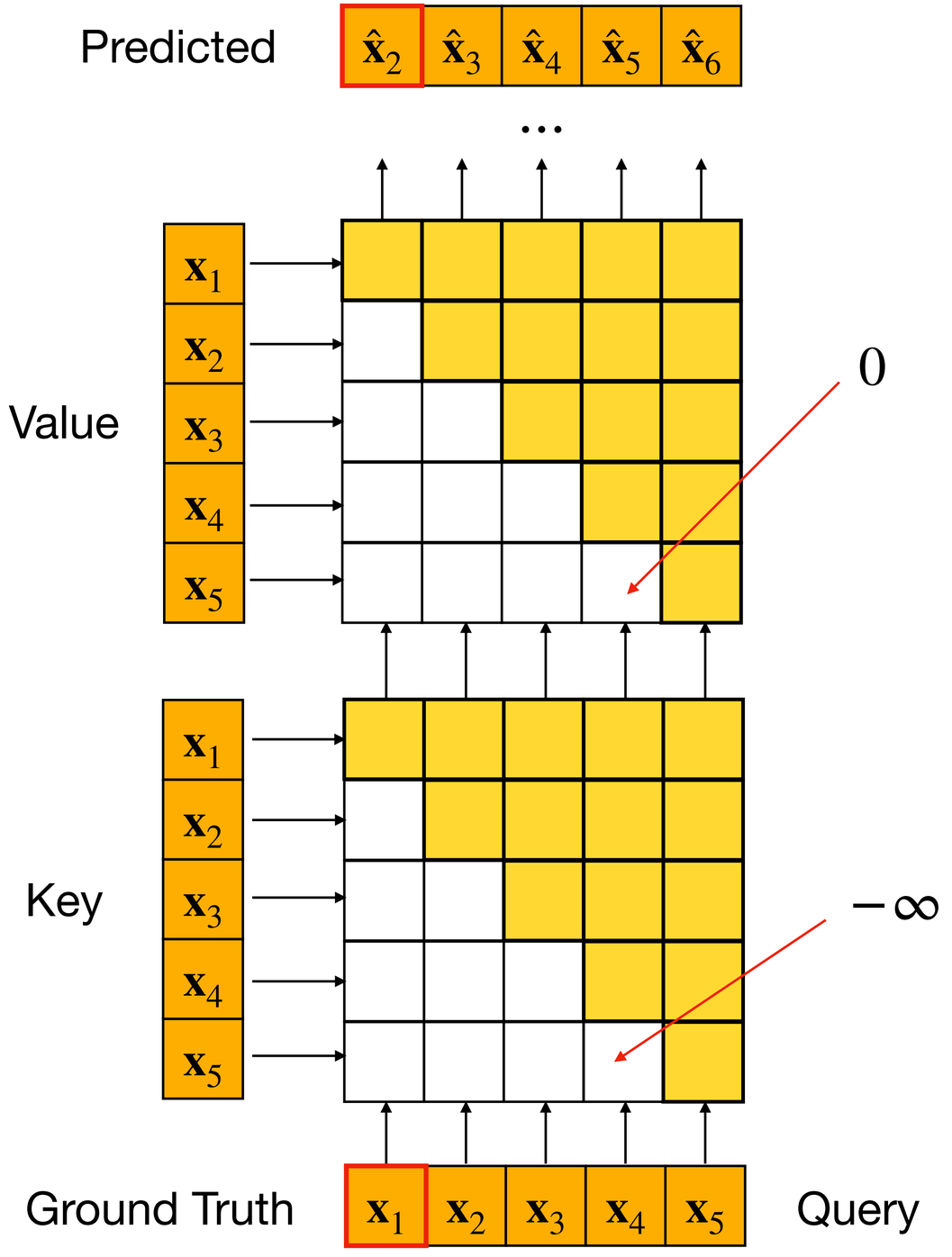


ACADENTIAL

왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

예를 들어서 $t = 1$의 경우를 살펴보자.

- 예측하고자 하는 값: $\mathbf{x}_2$
- 인풋으로 주어진 값: $\mathbf{x}_1$



ACADENTIAL

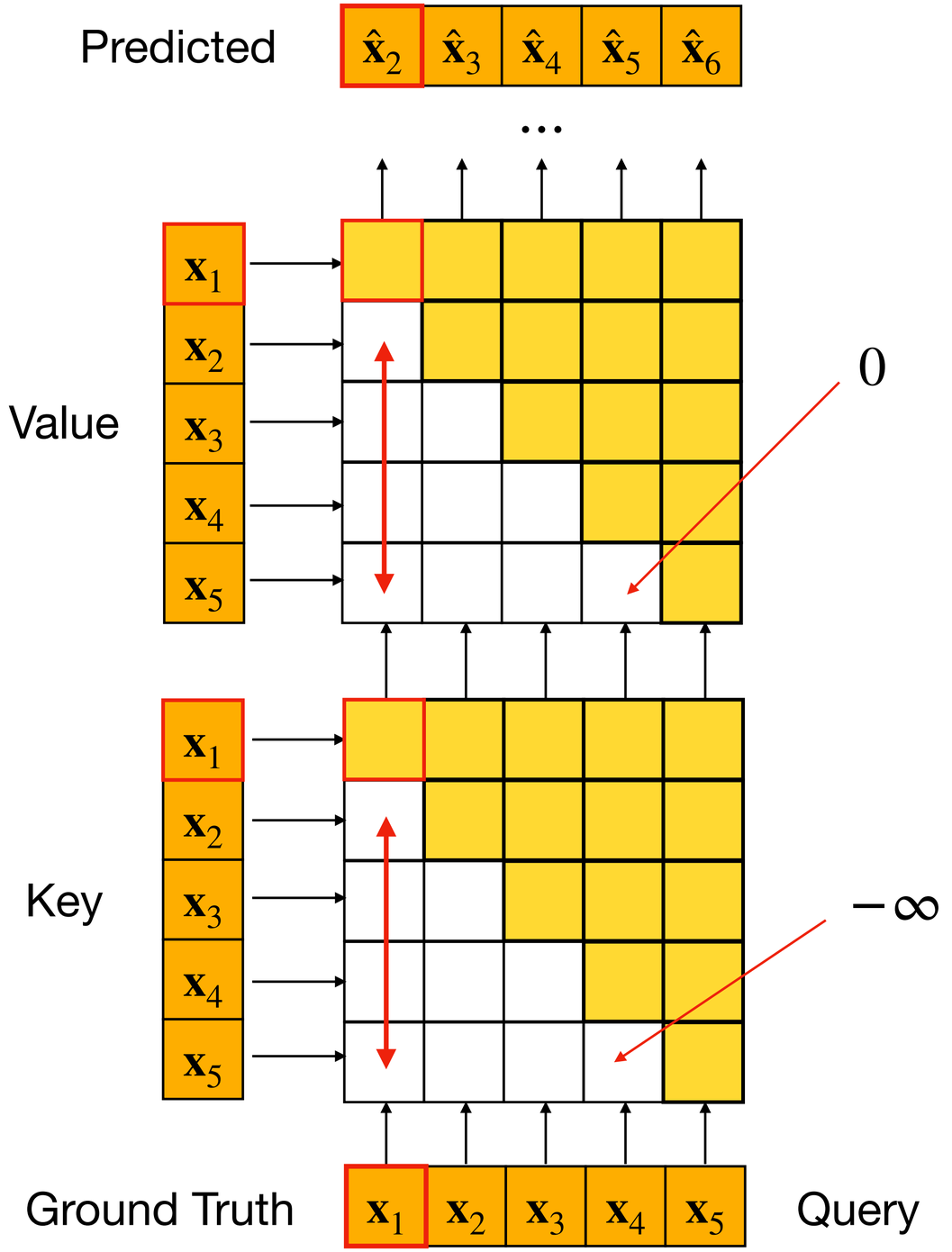
왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

예를 들어서 $t = 1$의 경우를 살펴보자.

- 예측하고자 하는 값: $\mathbf{x}_2$
- 인풋으로 주어진 값: $\mathbf{x}_1$

$\mathbf{x}_2$을 예측하는데 $\mathbf{x}_2, \mathbf{x}_3, \cdots$ ($t > 1$)을 사용하면 안된다!

(즉, $\mathbf{x}_2, \mathbf{x}_3, \cdots$ 로 부터 projection된 value와 key 값들.)
↔ 로 표시됨.



ACADENTIAL

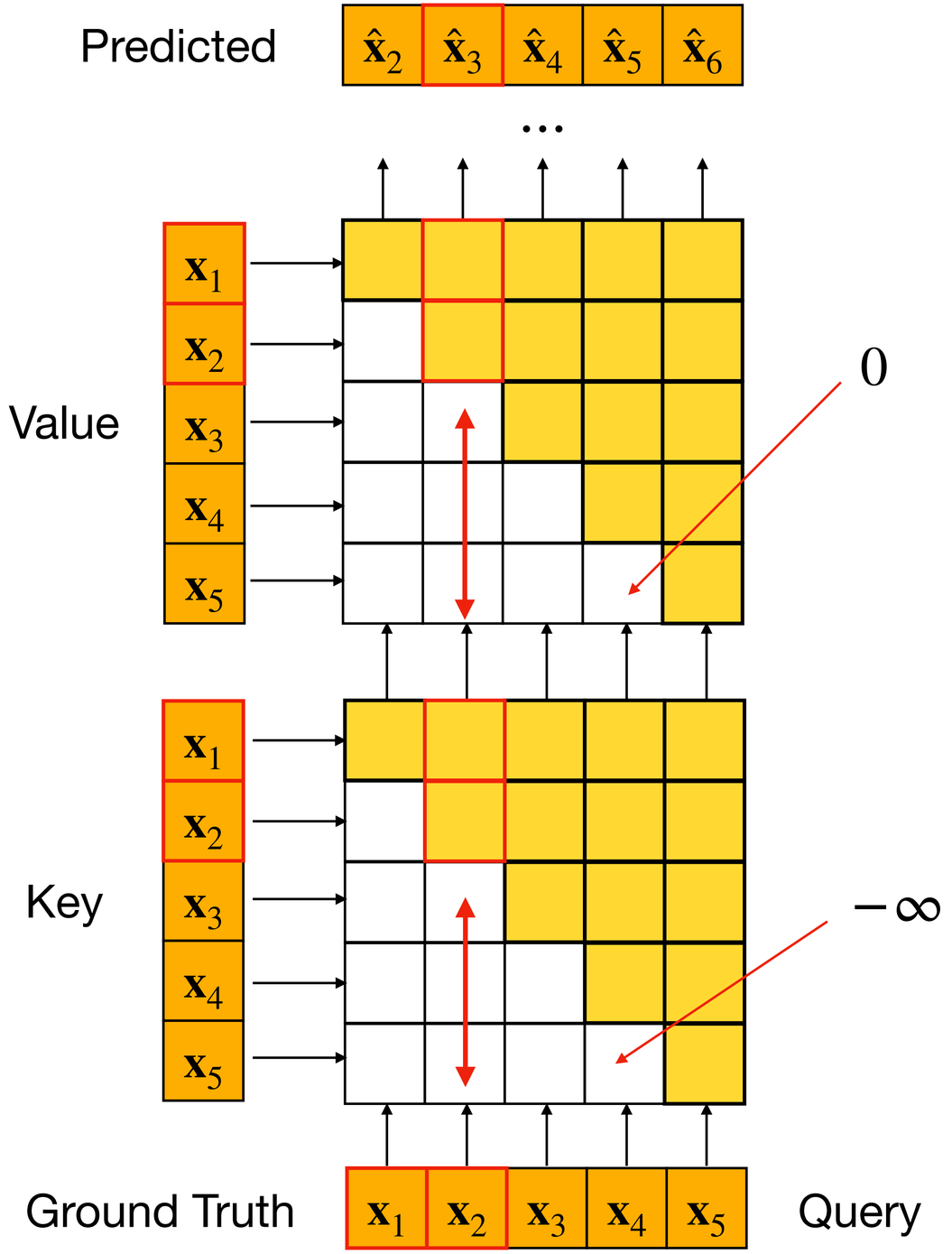
왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

$t = 2$의 경우를 살펴보자.

- 예측하고자 하는 값: $\mathbf{x}_3$
- 인풋으로 주어진 값: $\mathbf{x}_1, \mathbf{x}_2$

$\mathbf{x}_3$을 예측하는데 $\mathbf{x}_3, \mathbf{x}_4, \cdots$ ($t > 2$) 을 사용하면 안된다!

(즉, $\mathbf{x}_3, \mathbf{x}_4, \cdots$ 로 부터 projection된 value와 key 값들.)
↔ 로 표시됨



ACADENTIAL

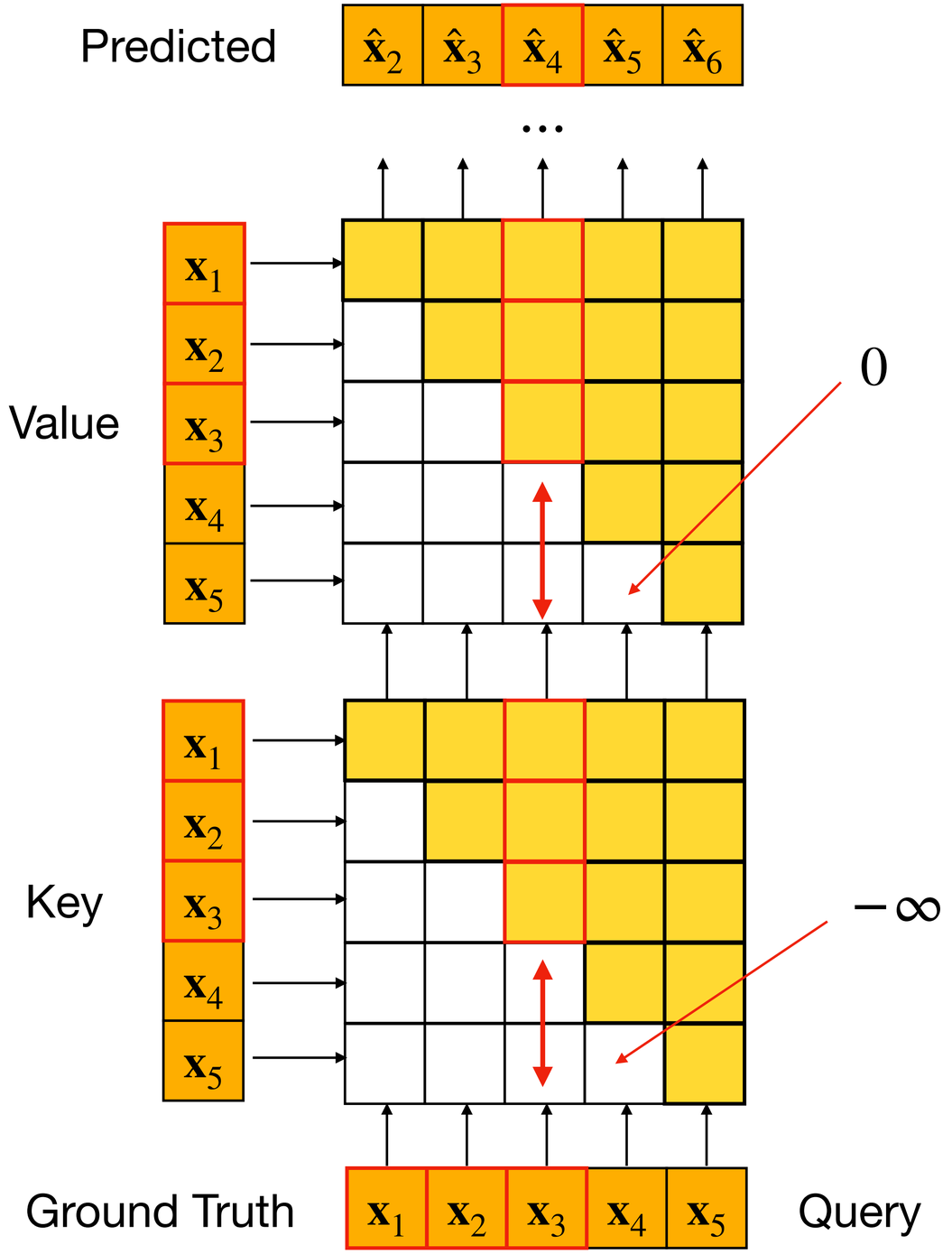
왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

$t = 3$의 경우를 살펴보자.

- 예측하고자 하는 값: $\mathbf{x}_4$
- 인풋으로 주어진 값: $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$

$\mathbf{x}_4$을 예측하는데 $\mathbf{x}_4, \mathbf{x}_5$ ($t > 2$) 을 사용하면 안된다!

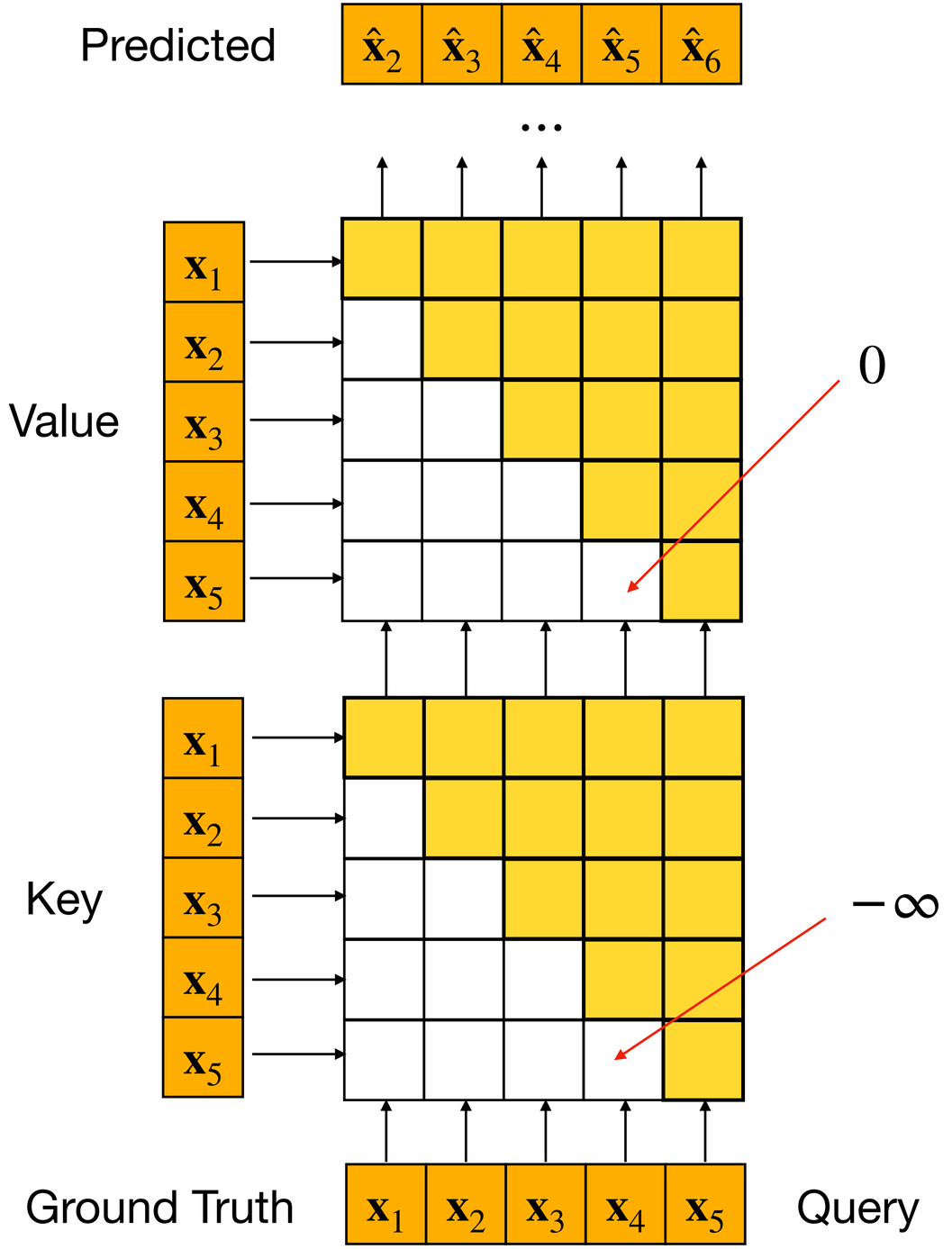
(즉, $\mathbf{x}_4, \mathbf{x}_5, \cdots$ 로 부터 projection된 value와 key 값들.)
↔ 로 표시됨



ACADENTIAL

왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

이렇게 예측할 token, 혹은 아직 예측하지 않은 (앞서는) token에 해당되는 attention 값을 mask처리하는 것

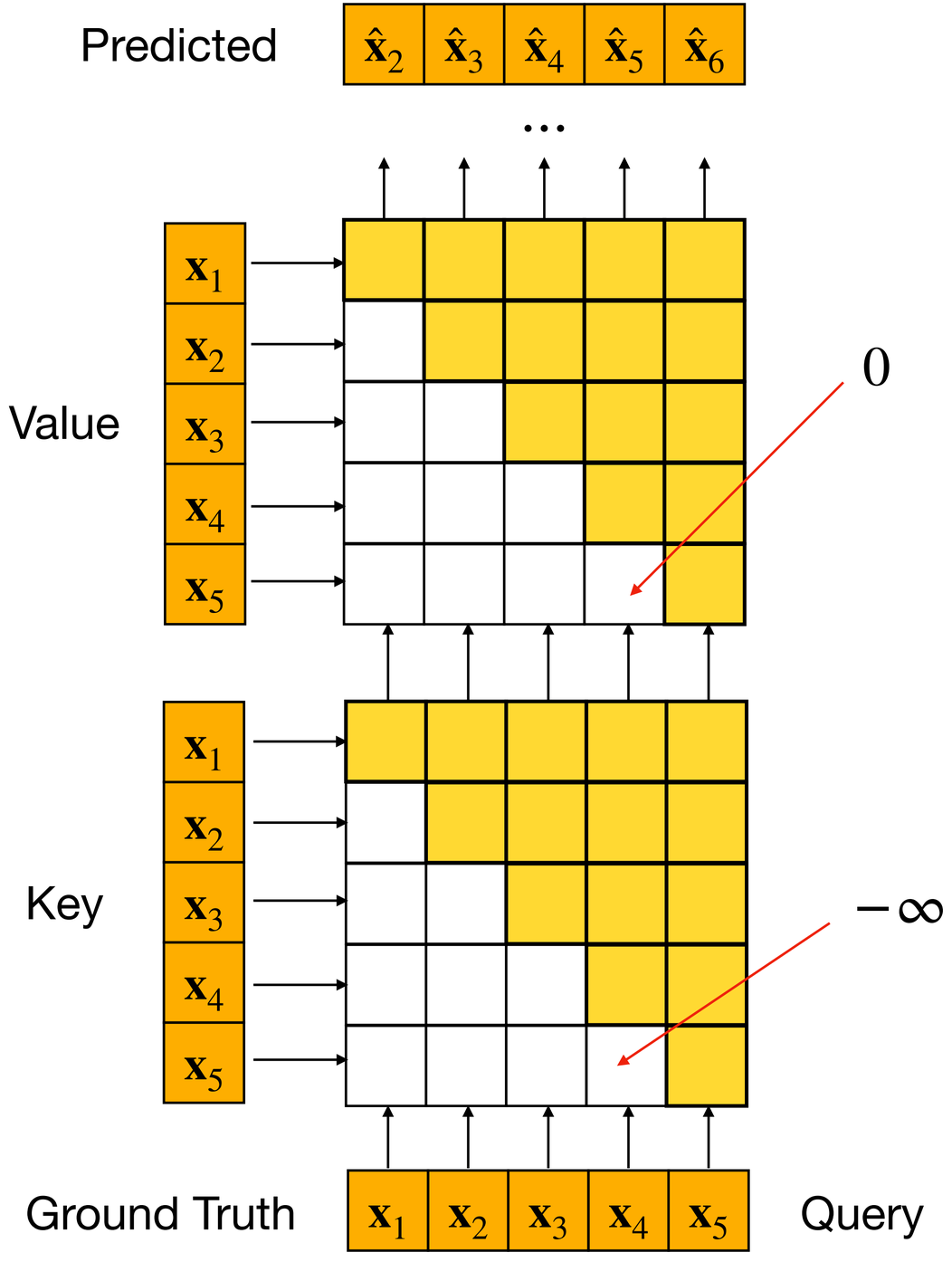


ACADENTIAL

왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

이렇게 예측할 token, 혹은 아직 예측하지 않은 (앞서는) token에 해당되는 attention 값을 mask처리하는 것

→ Causal Masking (Decoder의 self-attention에서 사용됨!)



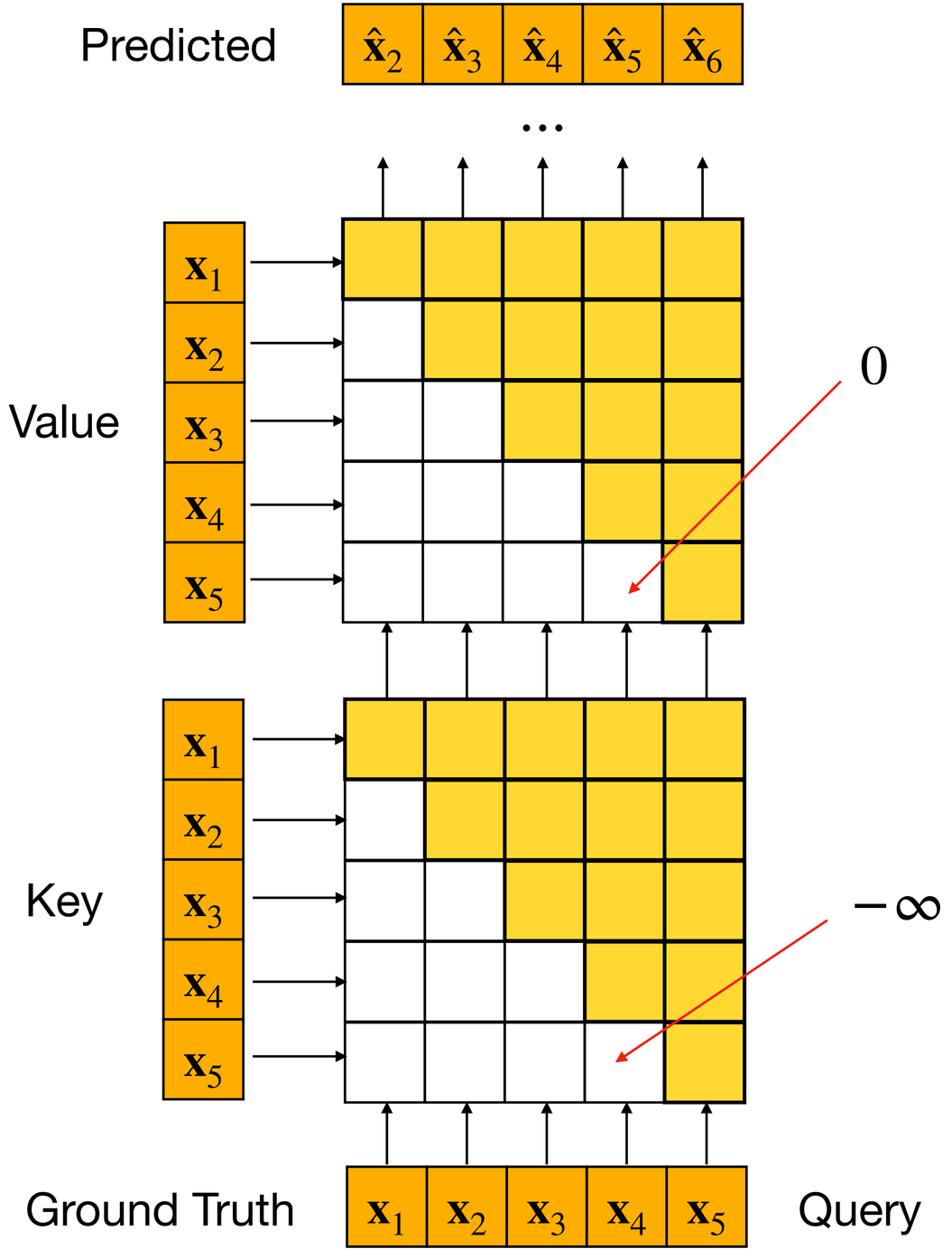
ACADENTIAL

왜 “하얀색”으로 색칠된 부분 ($j < i$)은 Attention 값을 0으로 설정할까?

이렇게 예측할 token, 혹은 아직 예측하지 않은 (앞서는) token에 해당되는 attention 값을 mask처리하는 것

→ Causal Masking (Decoder의 self-attention에서 사용됨!)

이를 통해 **Teacher Forcing**을 **적용**할 수 있고, **한 번의 forward pass만**으로 가능.

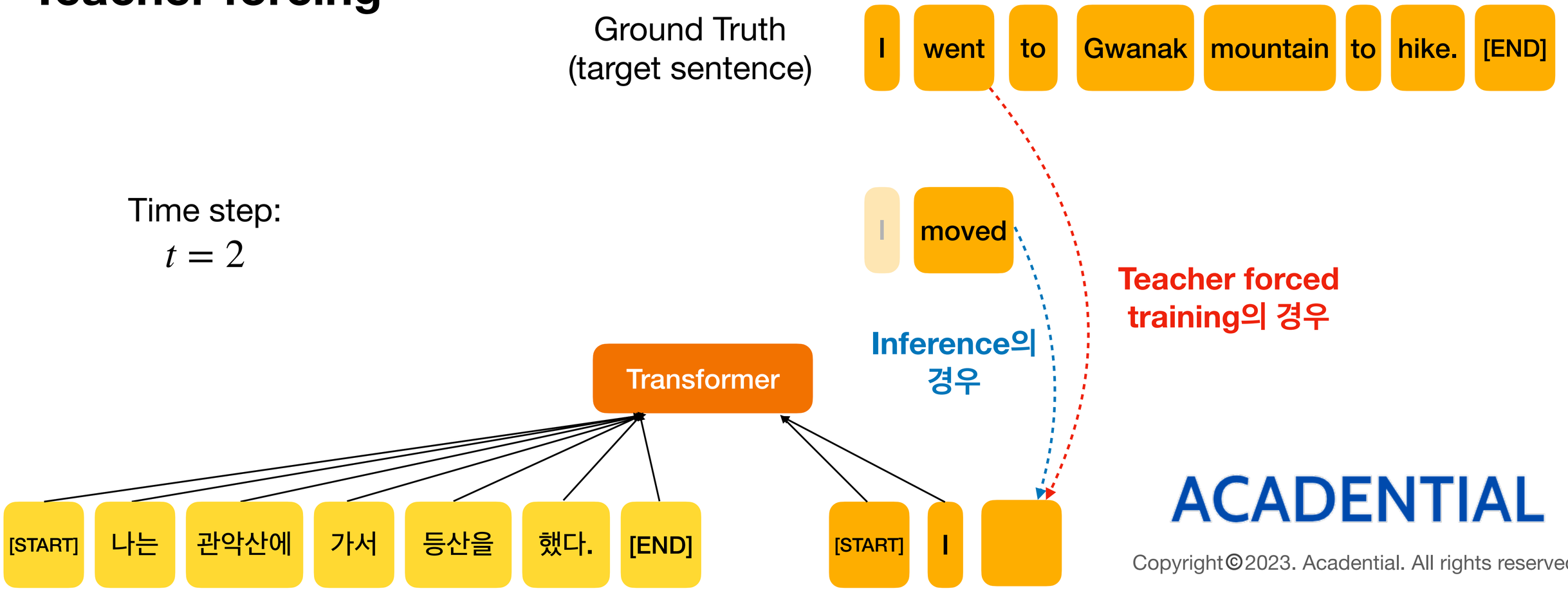


ACADENTIAL

Attention

Teacher forcing

물론 **inference하는 과정**에서는 **하나 하나씩 차례대로 생성**해야 한다.



ACADENTIAL

ACADENTIAL

16-11. Computational Complexity

ACADENTIAL

Time Complexity

- 흔히, “Attention의 Time Complexity은 N^2”이라고 한다.
- 이것의 의미가 뭘까?
- 먼저 Time Complexity가 뭔지 살펴보자!

ACADENTIAL

Time Complexity

- **Time Complexity (시간 복잡도)**

  “어떤 알고리즘을 수행하기 위해서 필요한 작업 (Operation)의 시행 횟수”

- 예를 들어서,

$$x + y \rightarrow \mathcal{O}(1)$$

$$x \cdot y \rightarrow \mathcal{O}(1)$$

$$\mathbf{x} \cdot \mathbf{y} \rightarrow \mathcal{O}(n) \qquad \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$$

$$\mathbf{XY} \rightarrow \mathcal{O}(NML) \qquad \mathbf{X} \in \mathbb{R}^{M \times N}, \mathbf{Y} \in \mathbb{R}^{N \times L}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

ACADENTIAL

먼저 Attention을 계산하는 과정을 recap해보자면:

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T$$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV$$

- 즉,
  $O = \text{softmax}(QK^T)V$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T \longrightarrow \mathcal{O}(NDH)$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV$$

- 즉,
  $O = \text{softmax}(QK^T)V$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T \rightarrow \mathcal{O}(NDH)$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T \rightarrow \mathcal{O}(N^2H)$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV$$

- 즉,
  $O = \text{softmax}(QK^T)V$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

ACADENTIAL

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T \rightarrow \mathcal{O}(NDH)$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T \rightarrow \mathcal{O}(N^2H)$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV \rightarrow \mathcal{O}(N^2H)$$

- 즉,
  $O = \text{softmax}(QK^T)V$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

먼저 Attention을 계산하는 과정을 recap해보자면:

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T \rightarrow \mathcal{O}(NDH)$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T \rightarrow \mathcal{O}(N^2H)$$

- Softmax을 취해서 attention을 구한다:

$$A = \text{softmax}(E)$$

- Attention으로 Value에 대한 weighted sum을 구한다:

$$O = AV \rightarrow \mathcal{O}(N^2H)$$

- 즉,
  $O = \text{softmax}(QK^T)V$

Total complexity = $\mathcal{O}(N^2H + NDH)$

where:

$$X \in \mathbb{R}^{N \times D}$$

$$W_q, W_k, W_v \in \mathbb{R}^{H \times D}$$

$$Q, K, V \in \mathbb{R}^{N \times H}$$

Time Complexity

$$\text{Total complexity} = \mathcal{O}(N^2H + NDH)$$

일반적으로 $X \in \mathbb{R}^{N \times D}$의 embedding 크기 D를 Attention layer의 Query, Key, Value의 hidden feature 크기 H와 동일하게 둔다. (실습편)

$D = H$인 경우,

$$\mathcal{O}(N^2H + NH^2)$$

ACADENTIAL

16-12. PyTorch로 구현된 BERT 모델 뜯어보기

ACADENTIAL

16-13. BERT 모델을 활용한 NLP 프로젝트

ACADENTIAL

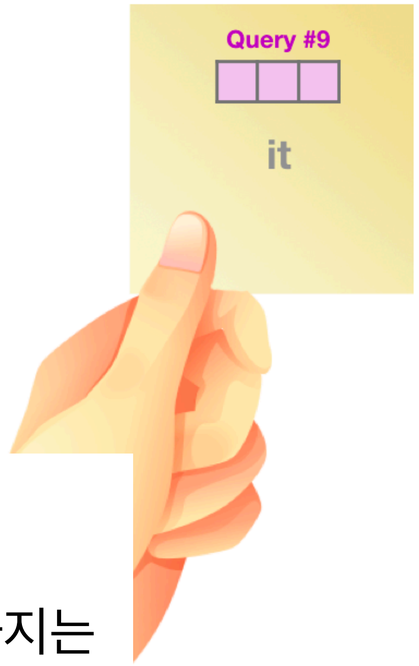
16-14. Section 16 요약

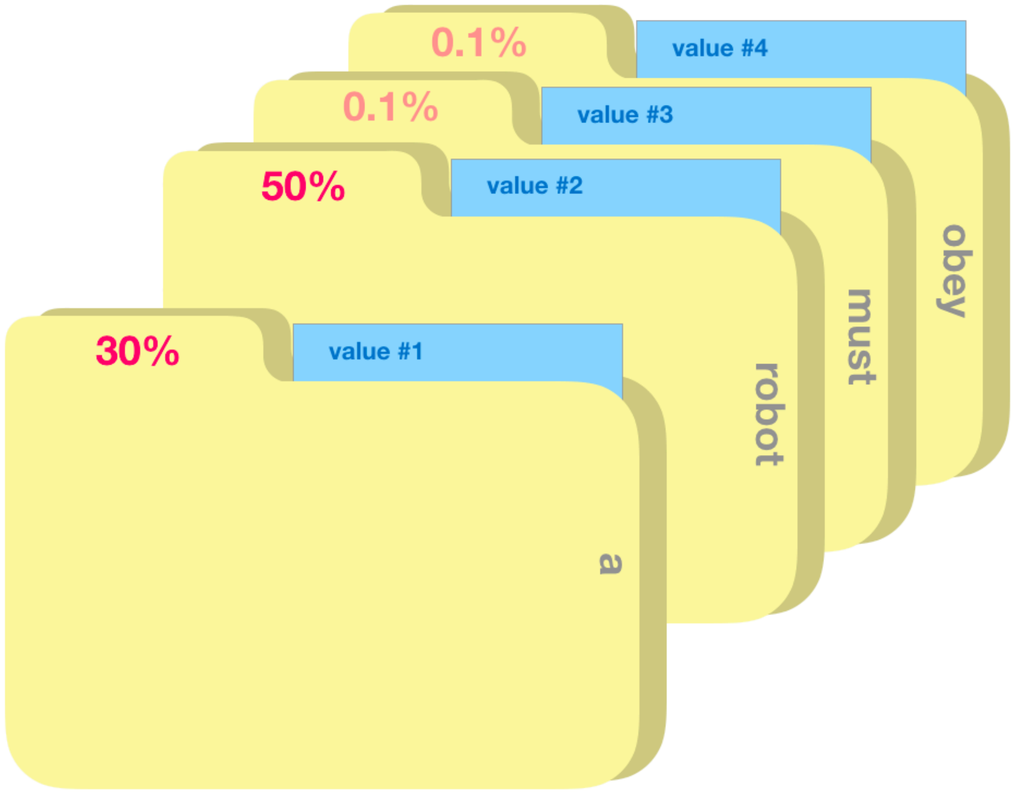
ACADENTIAL

Section Summary

Attention의 기본 개념

- Query Q: 쿼리 (질문)
- Key K: 키 (답안)
- Value V: 값 (내용)





1. “it”을 표현하는 “Query #9”
2. **Query**을 각 **Key**와 비교
3. **Query**와 **Key**가 가장 높은 연관성을 가지는 것을 고른다.
4. 해당 **Key**에 해당되는 **Value**을 출력한다.

출처: https://jalammar.github.io/illustrated-gpt2/

ACADENTIAL

Section Summary

Attention의 기본 개념

- Query, Key, Value 값들을 구한다:

$$Q = X(W_q)^T, K = X(W_k)^T, V = X(W_v)^T$$

- Query와 Key간의 attention score을 구한다:

$$E = QK^T$$

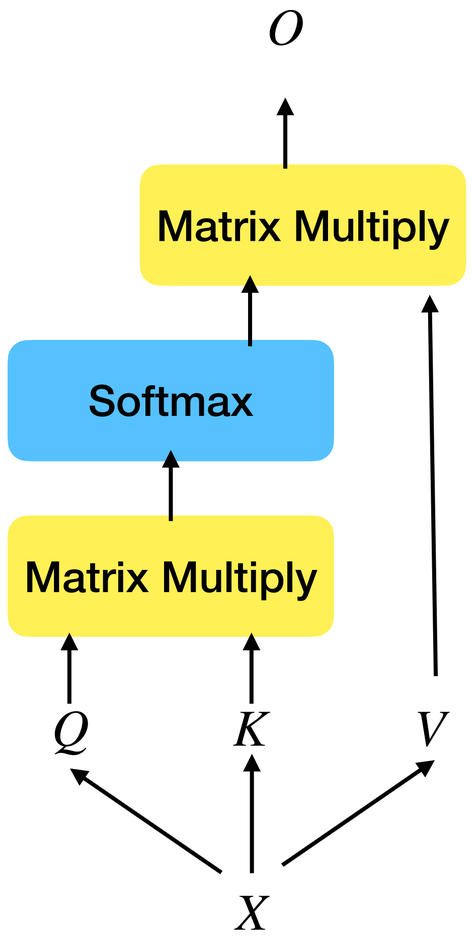
- Softmax을 취해서 attention weight을 구한다:

$$A = \text{softmax}(E)$$

- Attention weight에 가중치 averaged된 value을 구한다:

$$O = AV$$

$$O = \text{softmax}(QK^T)V$$



ACADENTIAL

Section Summary

Attention의 계보 - (Bahnadanau et al 2015)

- Attention이 가장 처음으로 제안된 논문은:

  “Neural Machine Translation by Jointly Learning to Align and Translate”

  (Bahdanau et al. ICLR 2015)

- 특징은
  - RNN Encoder-Decoder 구조에서 Attention이 Decoder에 적용된 것
  - “Additive” attention이 사용됨
  - Machine Translation task에 적용됨.

Section Summary

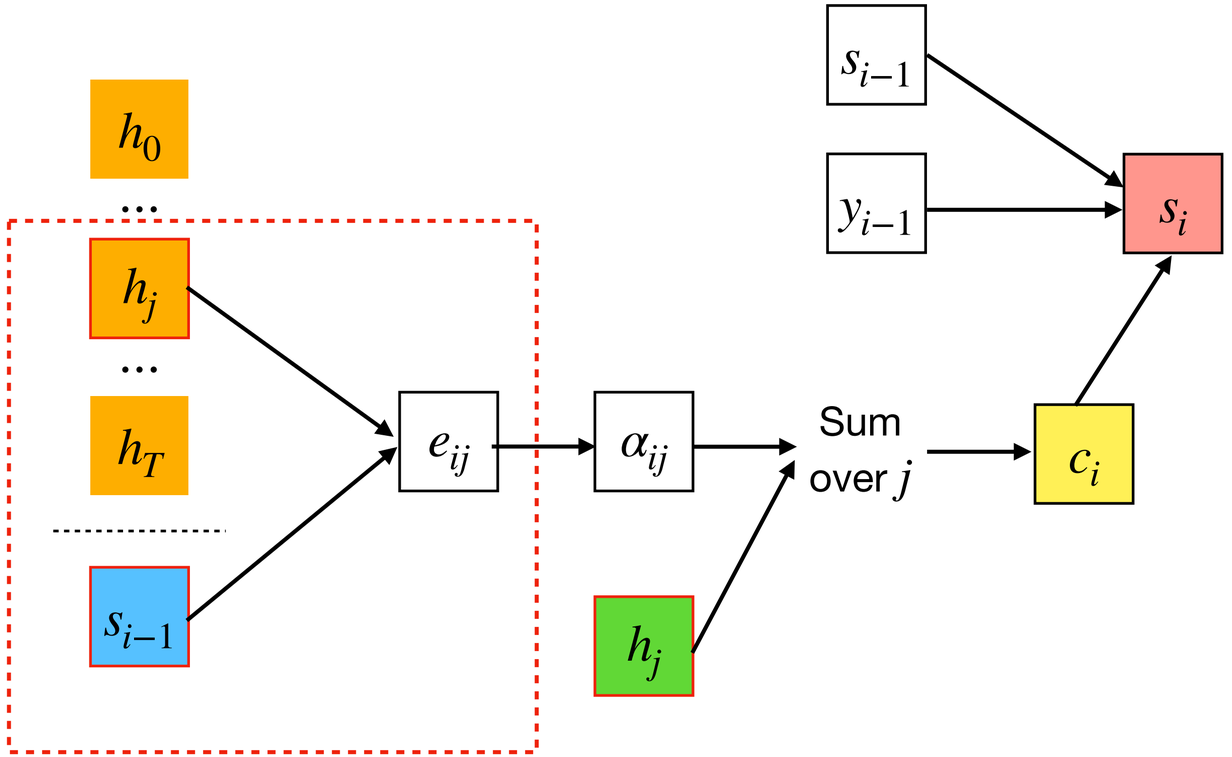
Attention의 계보 - (Bahnadanau et al 2015)

Attention score은 다음과 같다:

$$e_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

참고로, 여기서는 ***additive attention***을 사용한다!

그리고 (query = s_{i-1}) 와 (key = h_j) 이다.



ACADENTIAL

Section Summary

Attention의 계보 - Attention is all you need

특징:

- Transformer 모델을 제안함!
- 기존에는 attention이 CNN 혹은 RNN과 함께 사용되었다.
- Transformer 모델은 attention으로만 구성된 모델이다!
- Encoder와 Decoder로 구성되어 있다.

ACADENTIAL

Section Summary

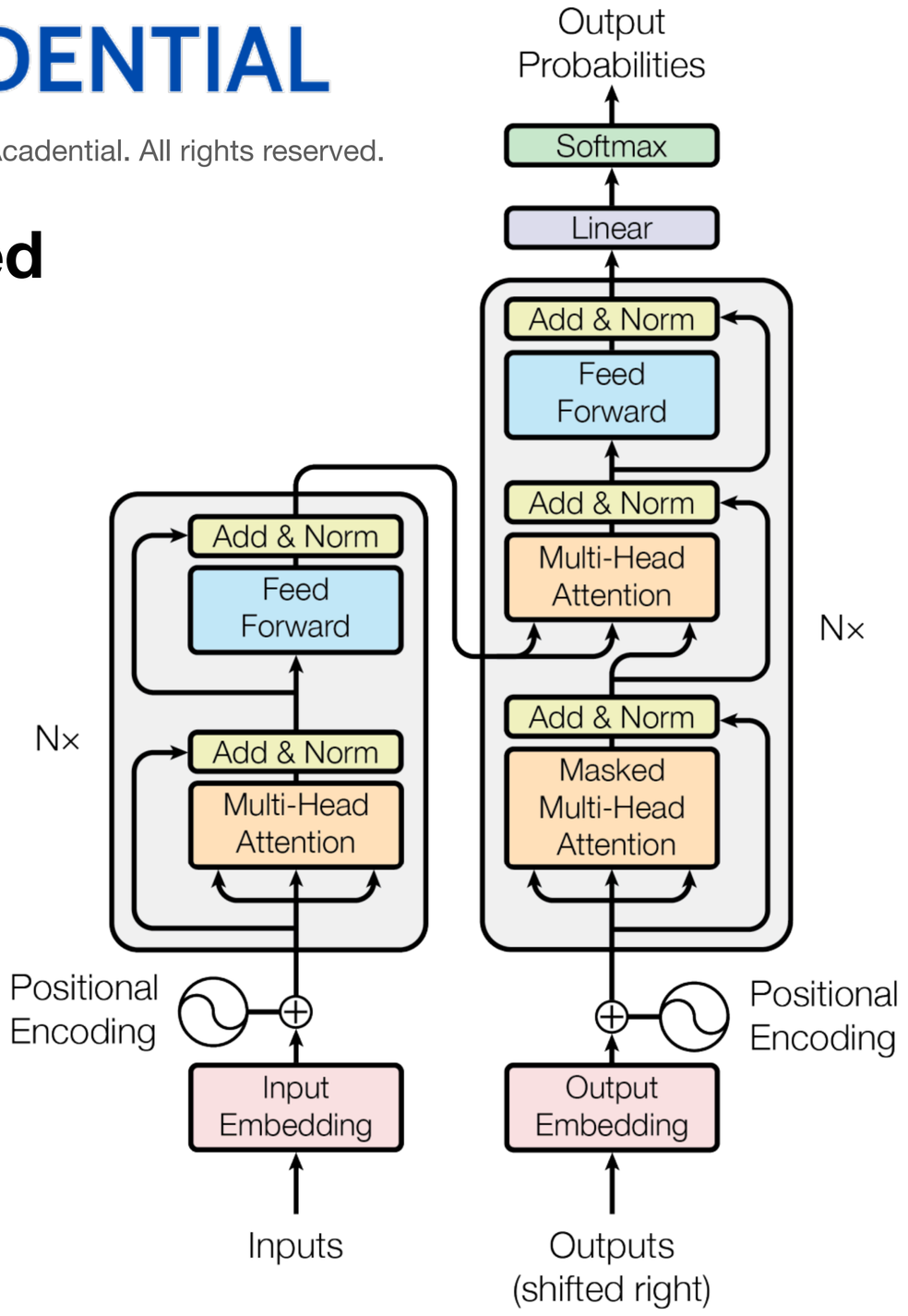
Attention의 계보 - Attention is all you need

특징:

- 다음 구성 요소를 가진다:
  - Multi-head self-attention
  - FC layer
  - Residual connection
  - Layer norm
  - Positional encoding

Encoder와 Decoder의 공통적 구성 요소

  - Multi-head attention (Decoder에서 추가됨)
  - Masked Multi-head self-attention



ACADENTIAL

Section Summary

Attention의 계보 - BERT

특징:

- **BERT** (Bidirectional **Encoder** representations from **Transformers**) 모델을 제안함!
- 즉, **BERT**은 **Transformer의 Encoder만 떼어낸 것**이다!
- **Masked Language Modeling pre-training** (MLM 사전 학습 방법)을 처음으로 제안함!
- MLM pre-training → Fine-tuning
- **Fine-tuning**하는 과정에서는 **마지막 layer만 교체.** (“Unified architecture for NLP”)

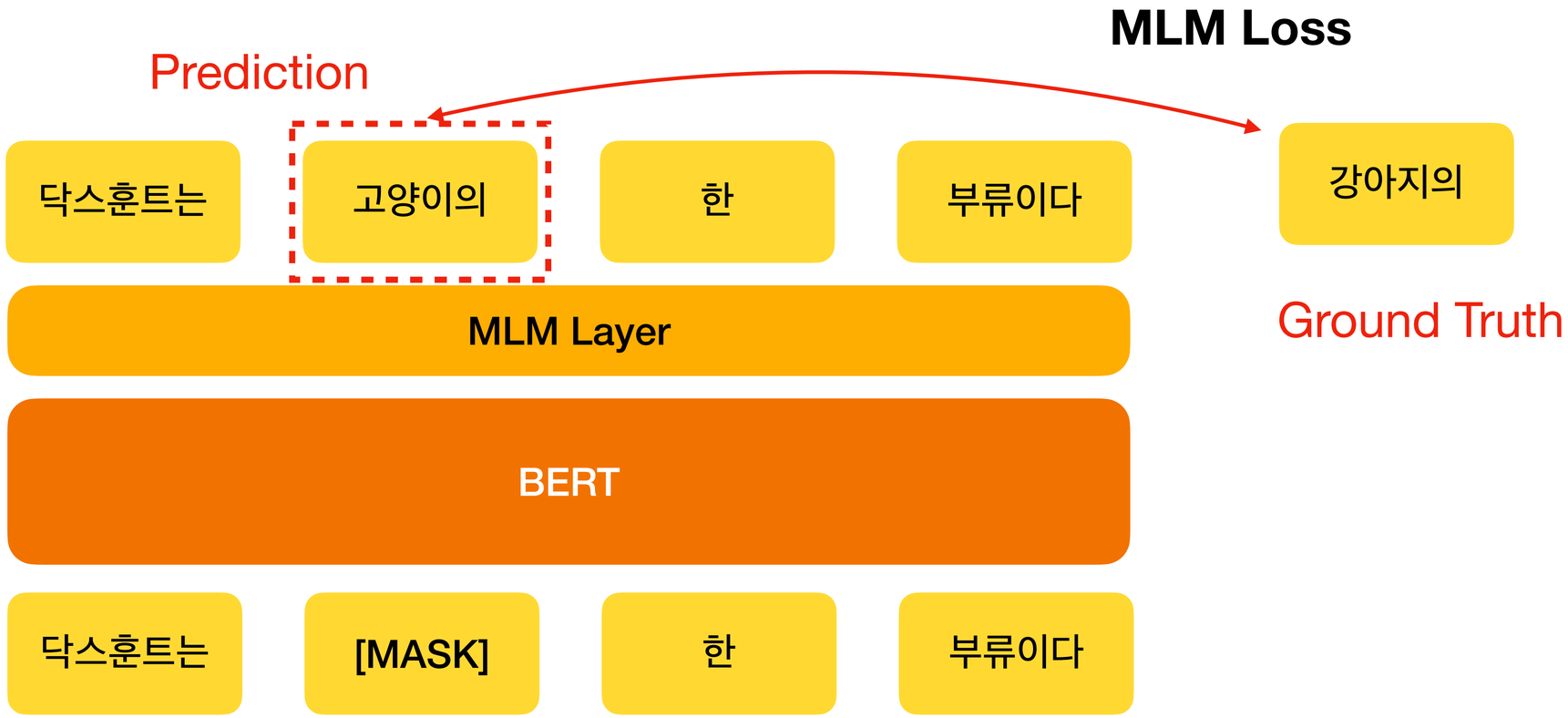
ACADENTIAL

Section Summary

Attention의 계보 - BERT

“Masked Language Modeling (MLM)”은 무엇인가?

“[MASK]”된 토큰을 맞추도록 모델을 학습시킨다!

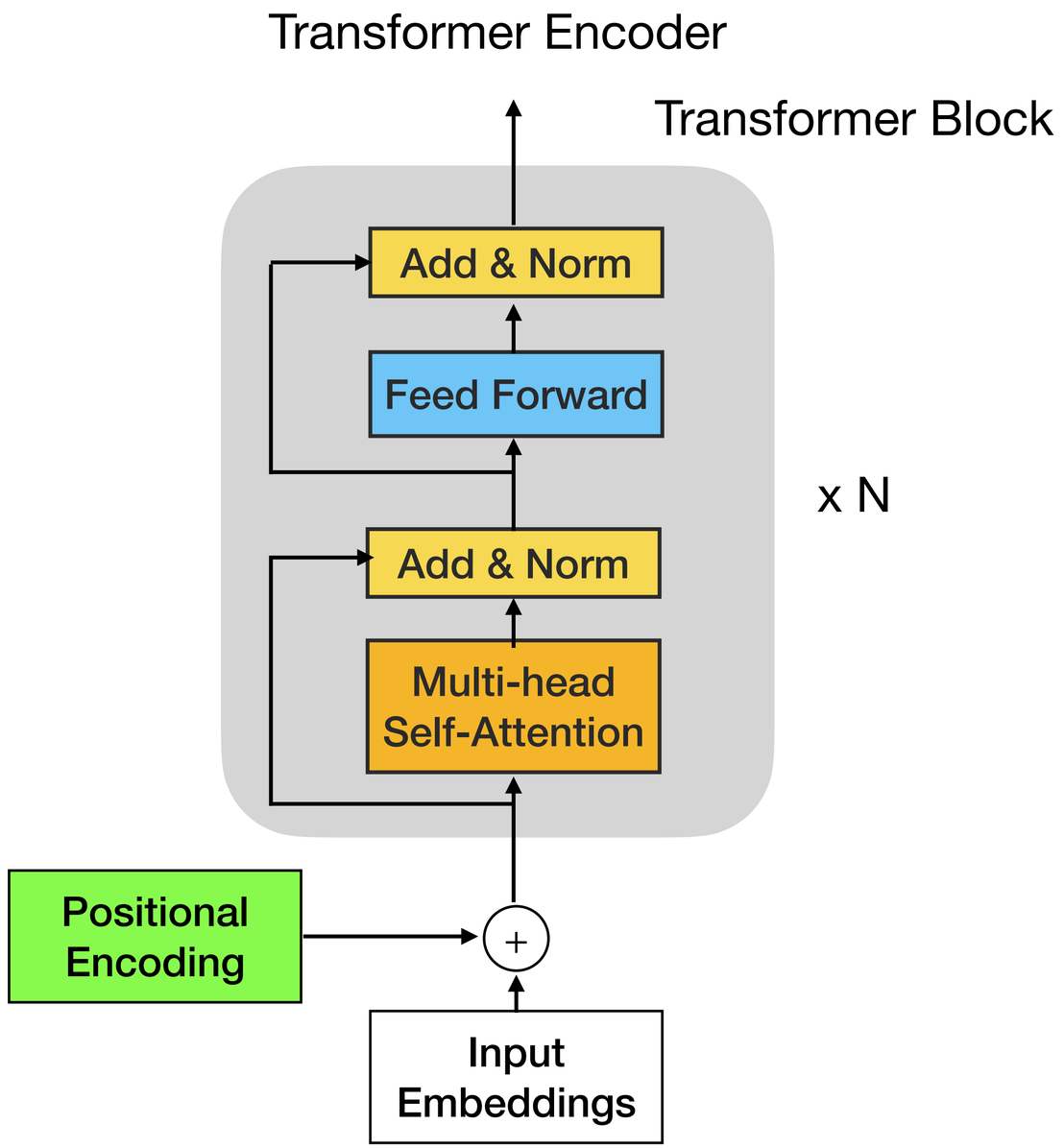


Section Summary

Transformer Encoder

Transformer의 architecture에 대해서 자세히 살펴보았다

- Feed forward layer
- Layer Norm
- Residual connection
- Scaled dot product
- Positional Embedding
- Multi-head attention

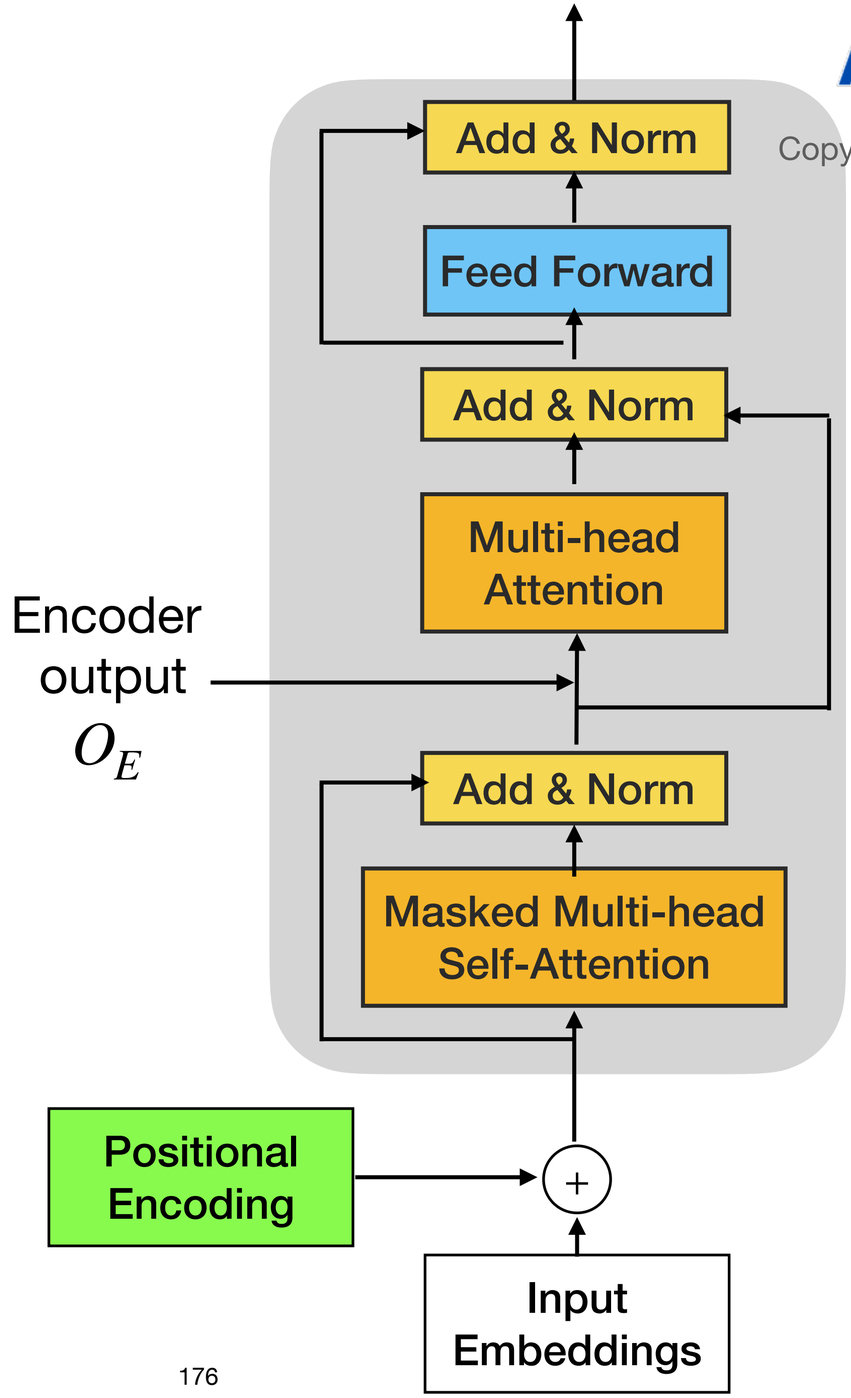


Section Summary

Transformer Decoder

특징:

- Transformer Decoder에서는 **Multi-head attention** layer가 추가되어 있다.

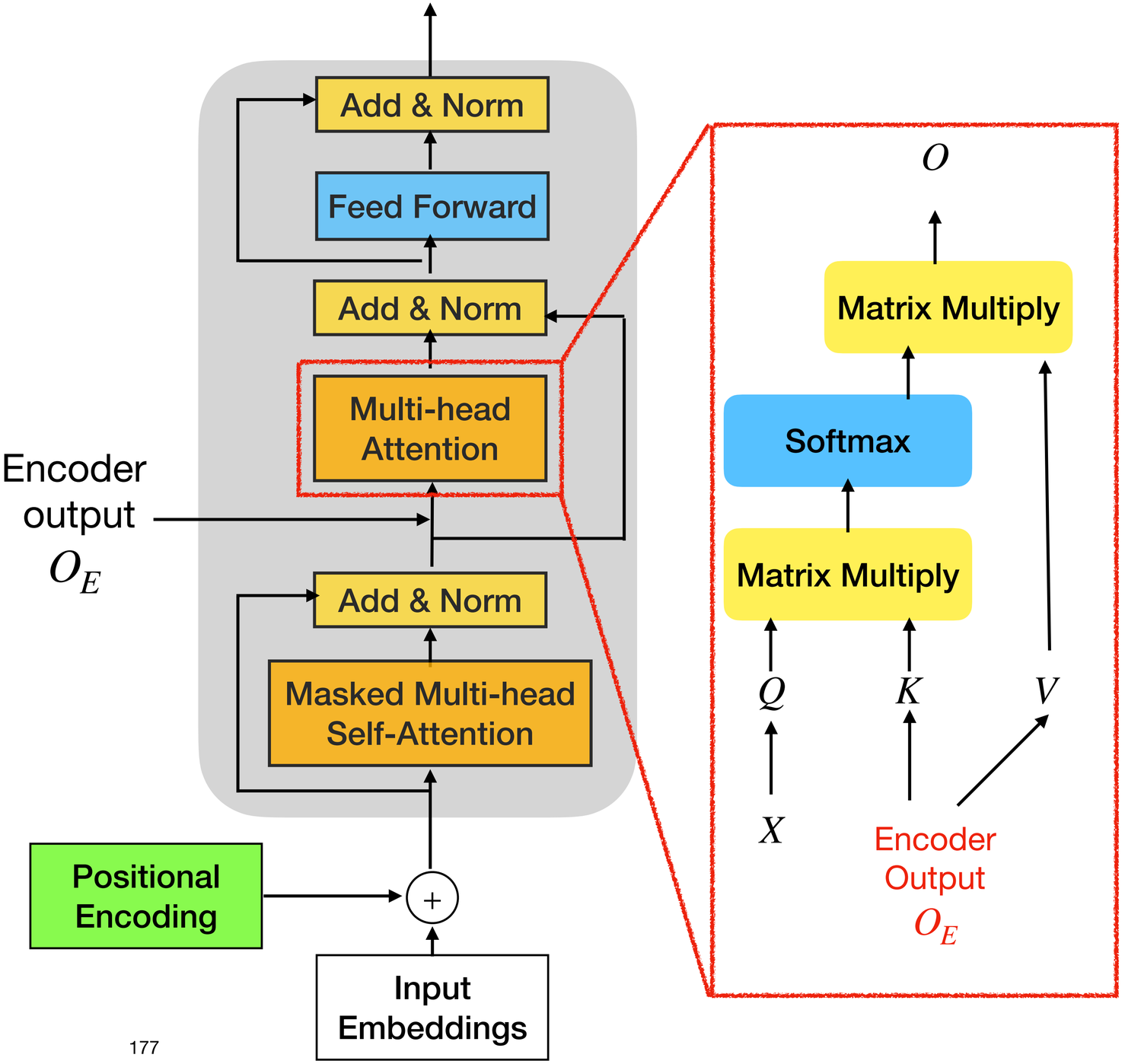


Section Summary

Transformer Decoder

특징:

- Transformer Decoder에서는 **Multi-head attention** layer가 추가되어 있다.



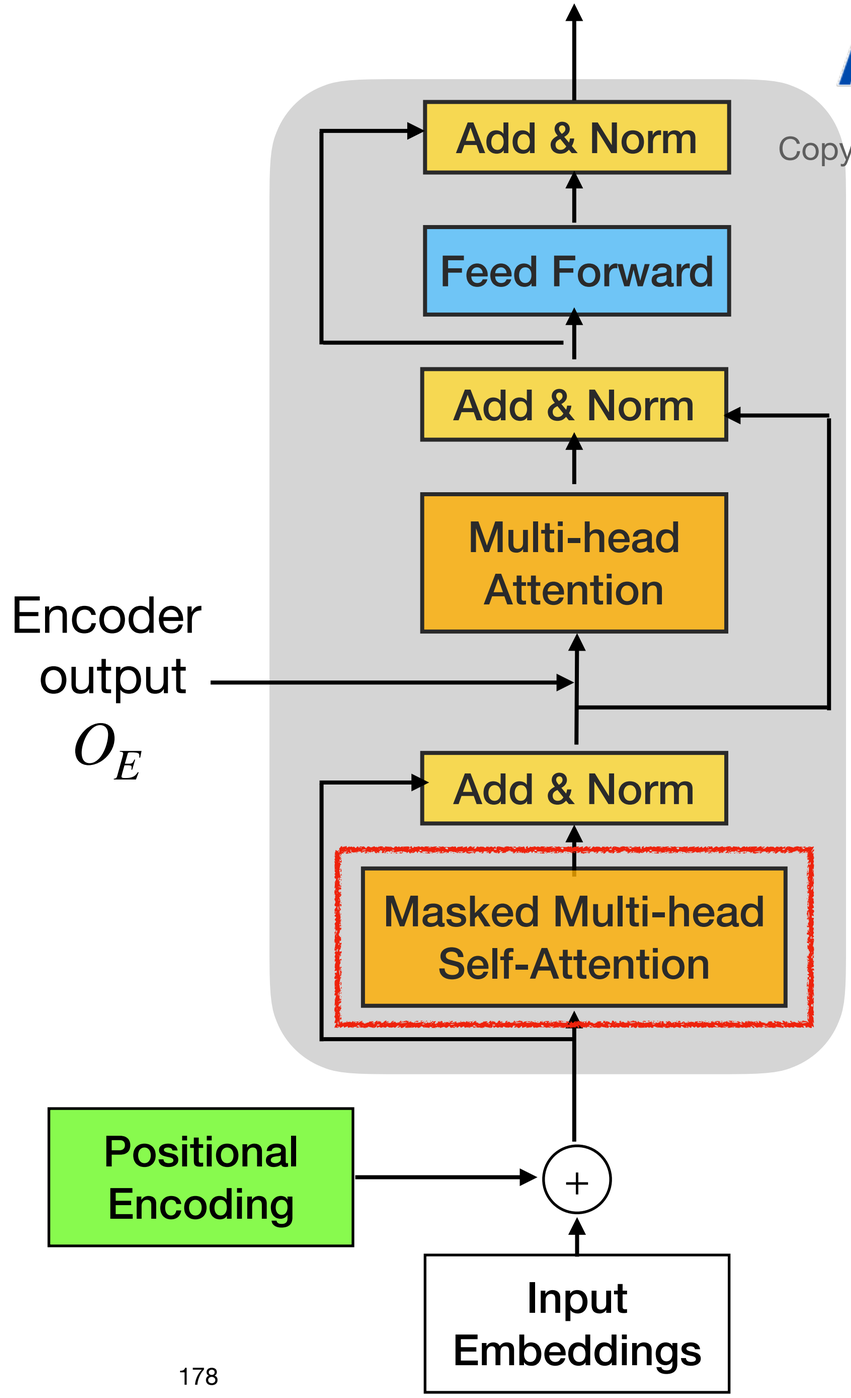
ACADENTIAL

Section Summary

Transformer Decoder

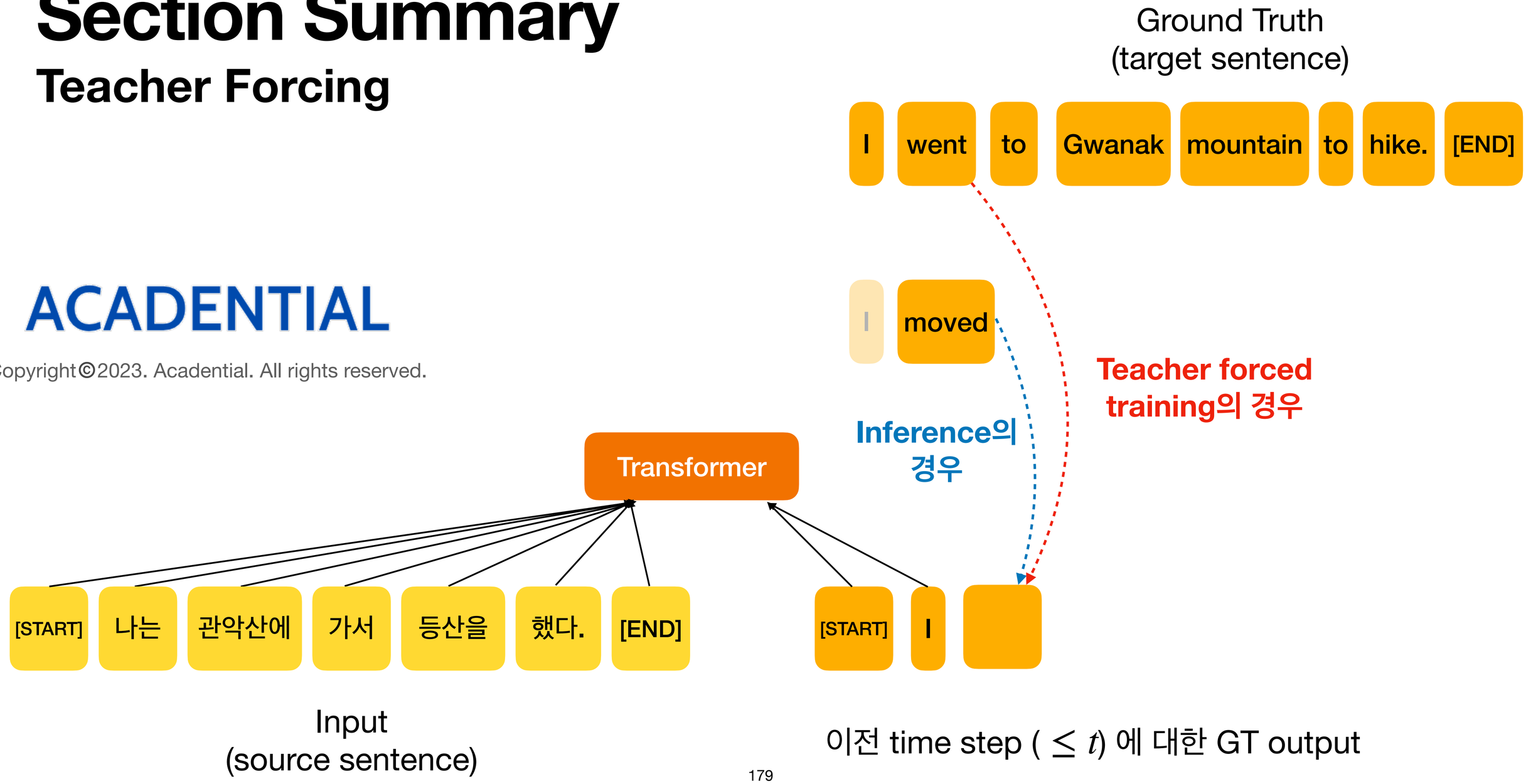
특징:

- Transformer Decoder에서는 Multi-head self-attention 대신에 **Masked Multi-head Self-attention**을 사용한다.



Section Summary

Teacher Forcing



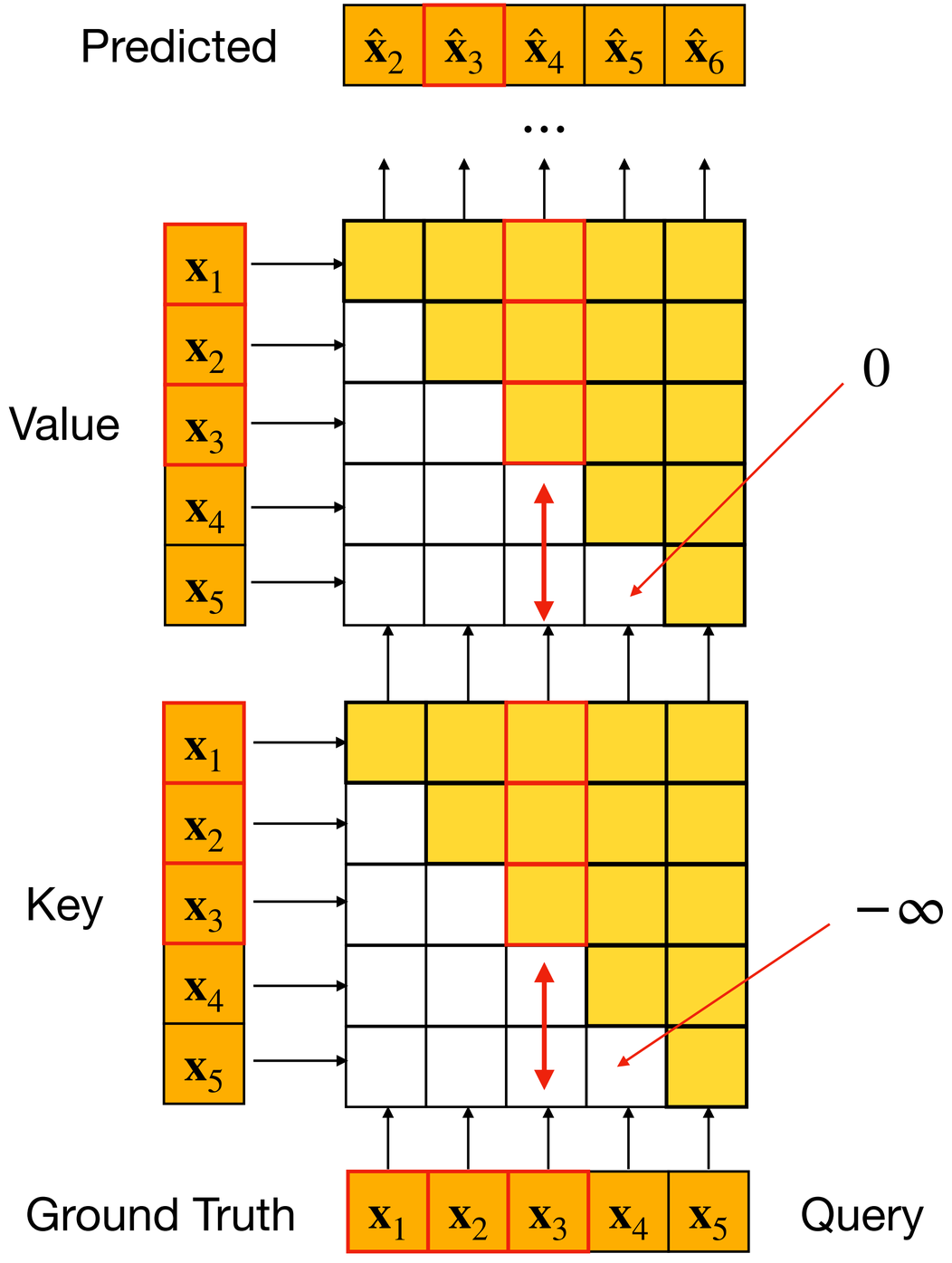
Section Summary

Causal Masking of decoder

Causal Masking

예측할 token, 혹은 아직 예측하지 않은 (앞서는) token에 해당되는 attention 값을 mask처리하는 것.

이를 통해 **Teacher Forcing**을 **적용**할 수 있고, **한 번의 forward pass만**으로도 **entire sequence**을 **단 한번에 예측**할 수 있다.



ACADENTIAL