BERT

컴퓨터공학과 손수현

BERT

: Pre-training of Deep Bidirectional Transformers for language understanding

&

Attention Is All You Need

CONTENTS

01 기존 모델과의 차이점

BERT

01. ELMo, OpenAI GPT

*** NLP에서 Pre-training model을 사용하는 방법 2가지**

1. Feature based - ELMo

- Task-specific한 architecture
- Pre-train한 representation을 additional feature로 넣는 방법
  (=2개의 network를 붙여서 사용)

2. fine-tuning – OpenAI GPT

- BERT 이전 SOTA를 달성한 model인 GPT가 사용한 방법
- Task-specific한 parameter를 최소화하여 fine-tuning
- bidirectional하려고 노력했으나, 결국은 [단방향 concat 단방향]

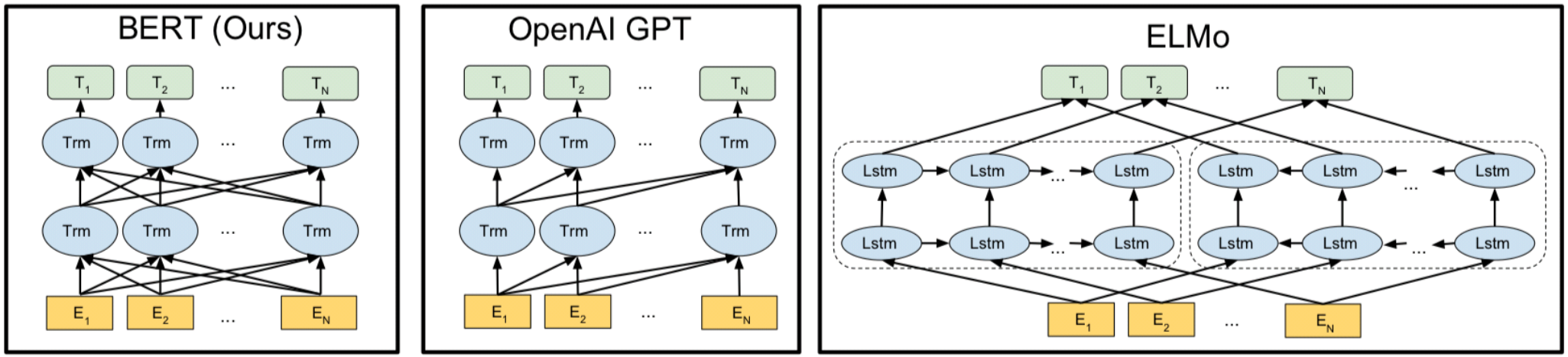
01. BERT

- **BERT**

1) unidirectional이 아닌 bidirectional

2) pretraining의 새로운 방법론 2가지 제시

3) 대규모 dataset으로 pre-training, 적은 양의 labeled dataset에 대해 fine-tuning



01. BERT

1) unidirectional이 아닌 bidirectional

- 기존

: 앞의 단어 n개의 단어를 가지고 뒤의 단어를 예측 → unidirectional

- BERT

: 주변 단어들을 보고 Masked 단어를 예측 → bidirectional

2) BERT의 Pre-training의 새로운 방법론 2가지

1. Masked Language Model

: 주변 단어들을 보고 Masked 단어를 예측

2. Next Sentence Prediction Task

: 문장들 사이의 관계를 학습하기 위해 “다음 문장이라는 Label” 추가

01. BERT

2-1. Masked Language Model

: input 에서 random하게 몇 개의 Token을 Mask

: Mask 처리한 sequence를 Transformer의 Encoder 에 넣어서 주변 단어 Context를 보고 Mask 된 Token을 예측하는 Model

2-2. Next Sentence Prediction Task

: 두 문장을 Pre-training때 함께 넣어서 두 문장이 이어지는 문장인지 아닌지 맞추게 하는 → 문장내의 상관성을 위해

: Pre-training시,

실제로 이어지는 두 문장 : 랜덤한게 추출한 두 문장 = 50 : 50

01. BERT

3) unlabeled data(wiki, book data 등)으로 model을 pre-training한 후 특정 Task를 가진 labeled data로 fine-tuning

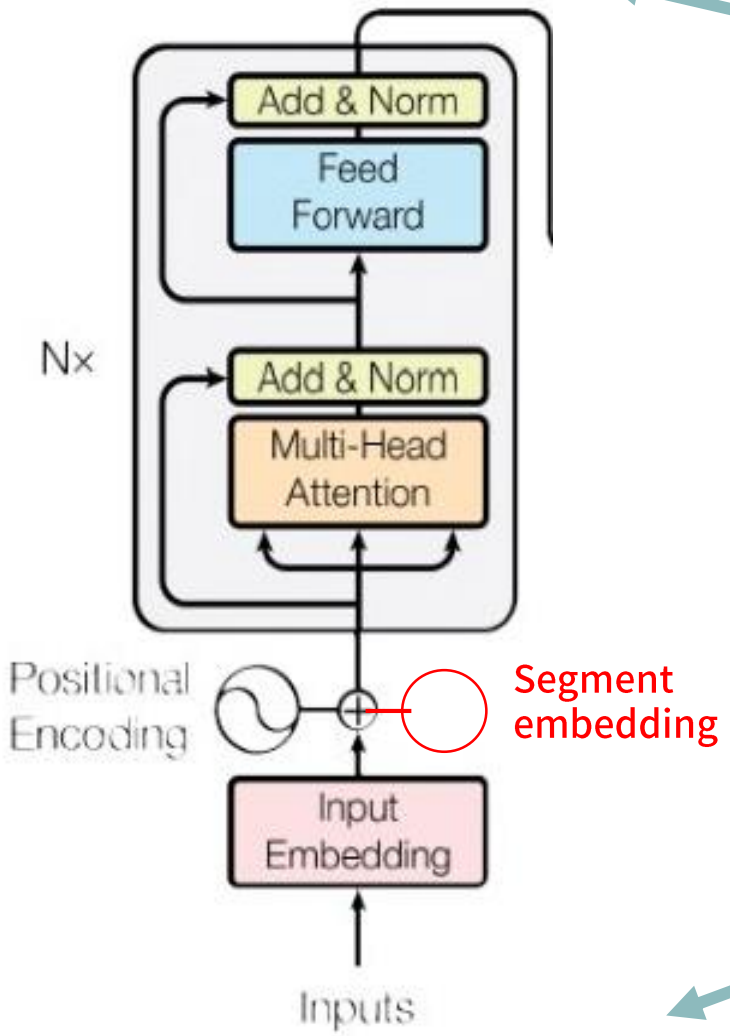
→ 특정 Task를 위한 network 붙일 필요 X

→ pre-trained BERT Model은 1개의 output layer로 fine-tuning할 수 있다

02 BERT 구조

BERT

02. BERT 구조



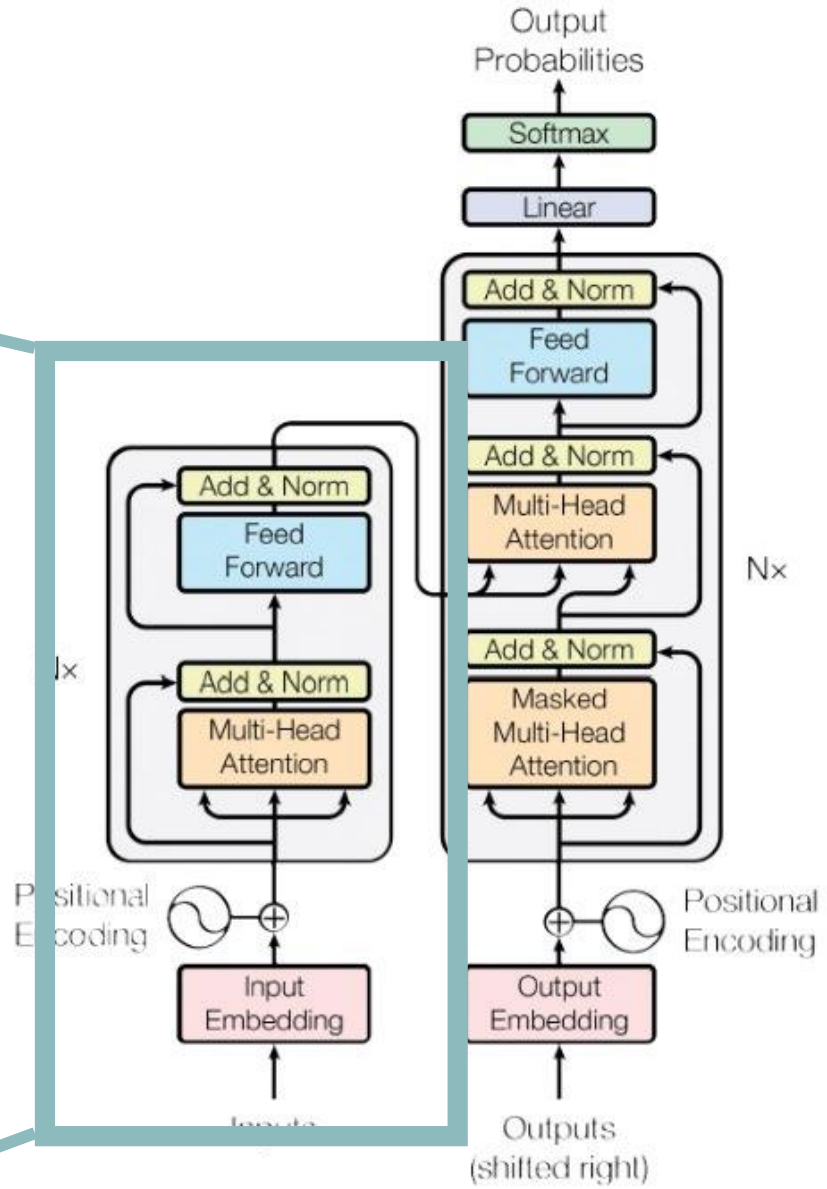
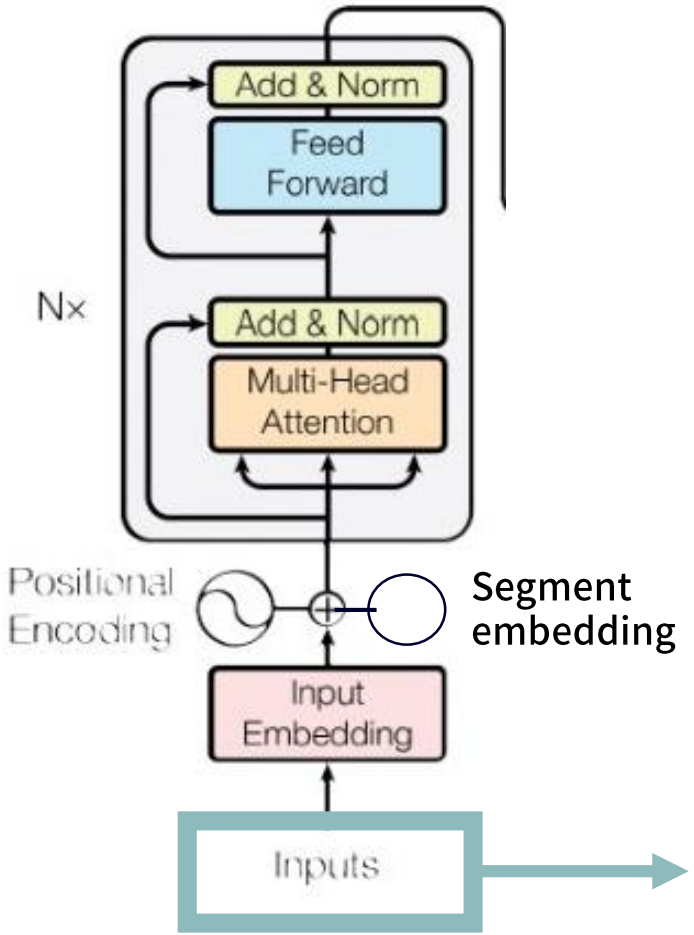


Figure 1: The Transformer - model architecture.

02. BERT 구조



Preprocessing

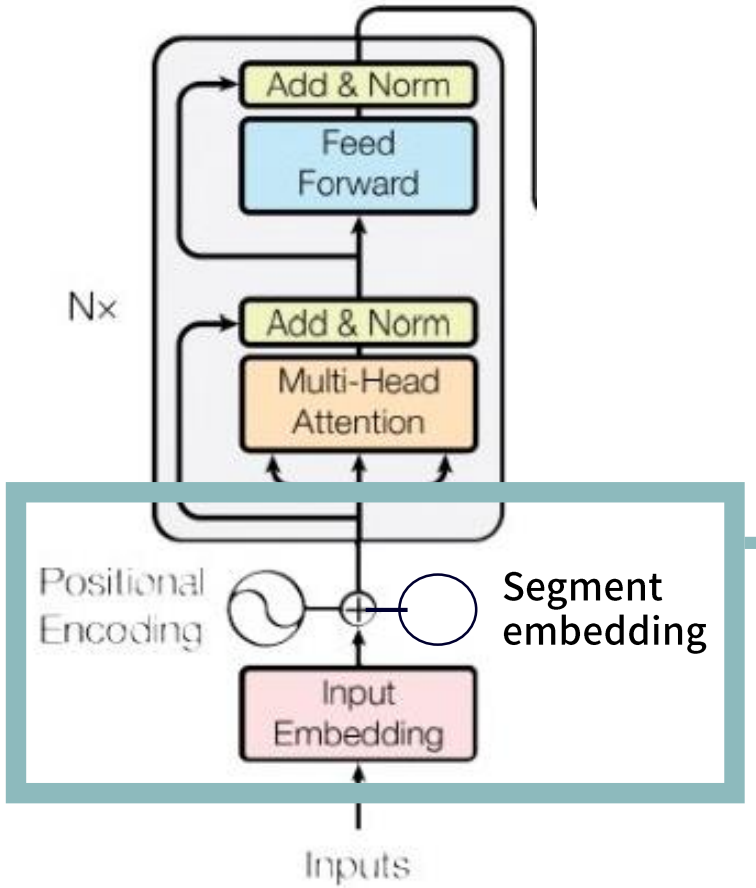
→ [CLS] : Task-specific한 정보를 주기위한 Token

→ [SEP] : Token A,B를 구분하는 Token

→ [SEP] : Token B의 끝을 알리는 Token

→ [MASK] : 예측하는 Target

02. BERT 구조



3가지의 Embedding

1) Input embedding

2) Positional embedding

3) Segment embedding

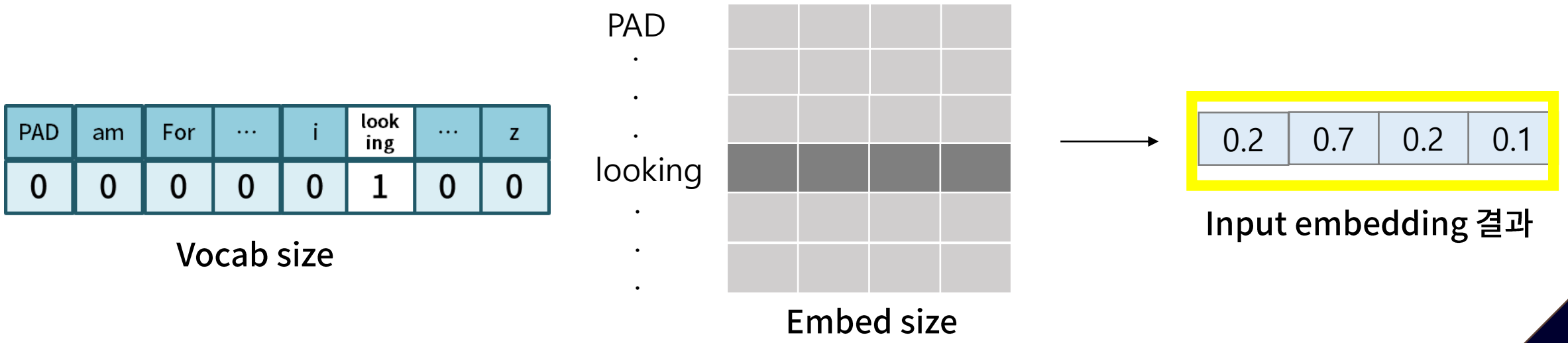
02. BERT 구조

1) Input embedding

:각 token의 one-hot vector를 Weight matrix랑 곱한다.

: 그 결과 → embedding vector

[CLS] I am looking for happiness [SEP] B type sentence [SEP]



02. BERT 구조

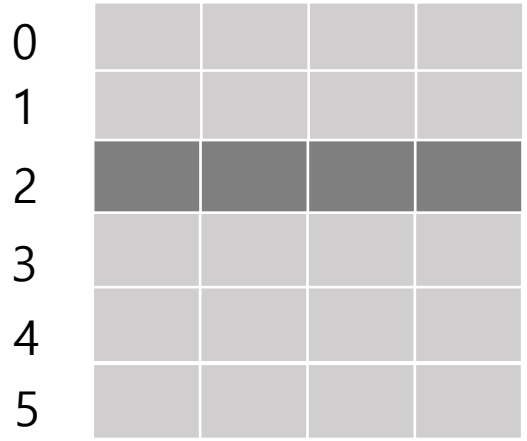
2) Positional encoding

: 각 단어의 위치정보를 저장하기 위해 단어의 embedding vector + 위치정보

: position에 대한 one-hot vector를 weight matrix와 곱한다.

[CLS] I am looking for happiness [SEP] B type sentence [SEP]

0	1	2	3	4	5
0	0	1	0	0	0



0.1	0.1	0.9	0.1

position encoding 결과

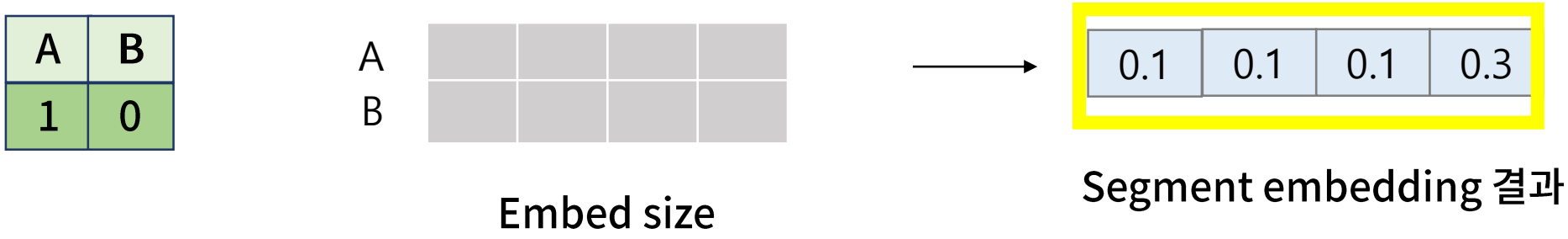
02. BERT 구조

3) Segment embedding (Token embedding)

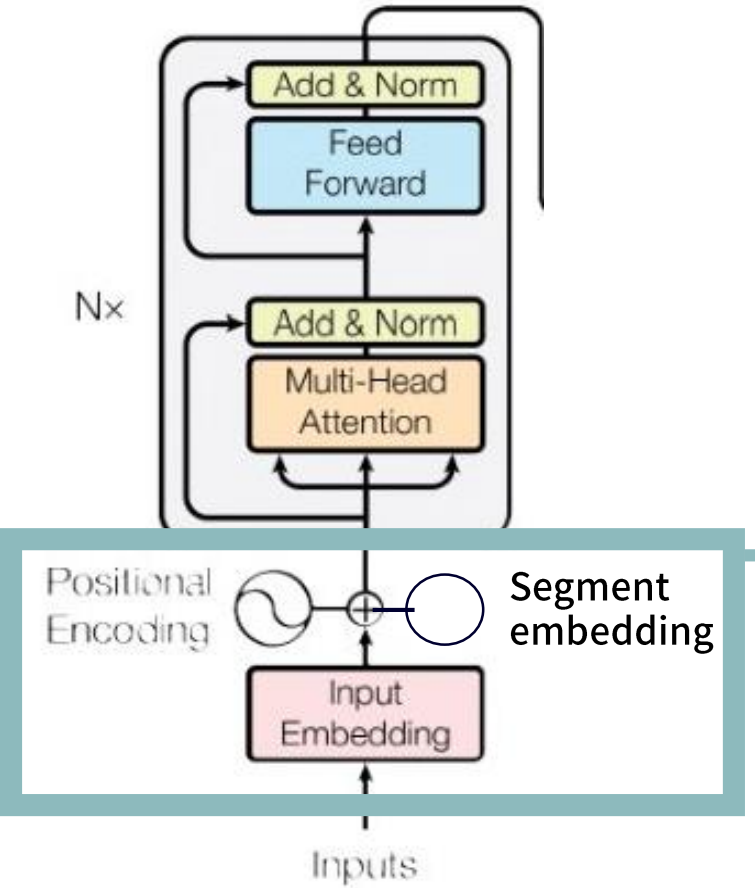
: Token A group, Token B group을 구분하는 정보

: BERT에서 추가된 embedding

[CLS] I am looking for happiness [SEP] B type sentence [SEP]



02. BERT 구조



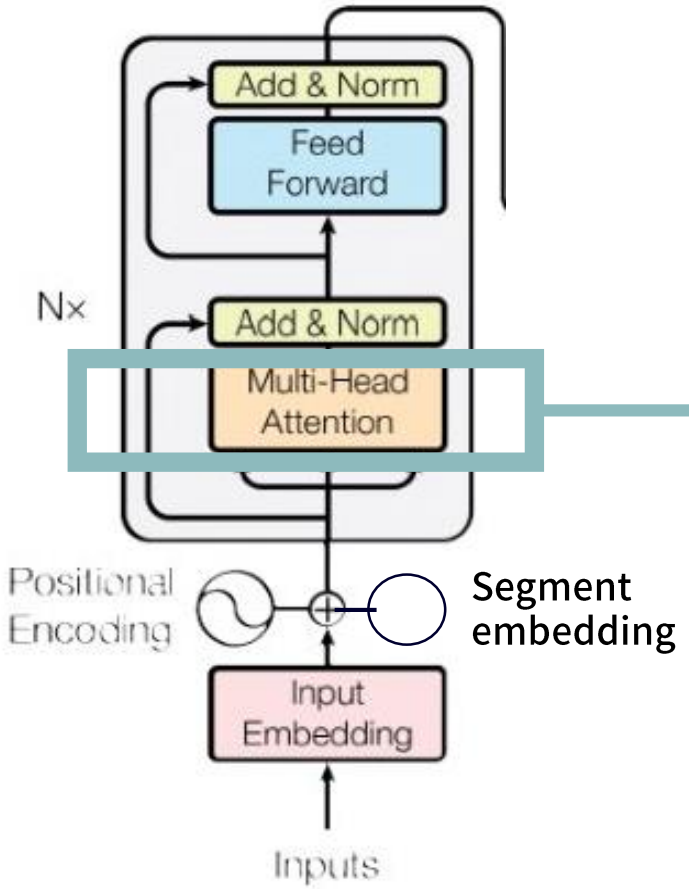
Input embedding 결과

+ Positional encoding 결과

+ Segment embedding 결과

→ Encoder의 입력

02. BERT 구조



* Multi-Head Attention

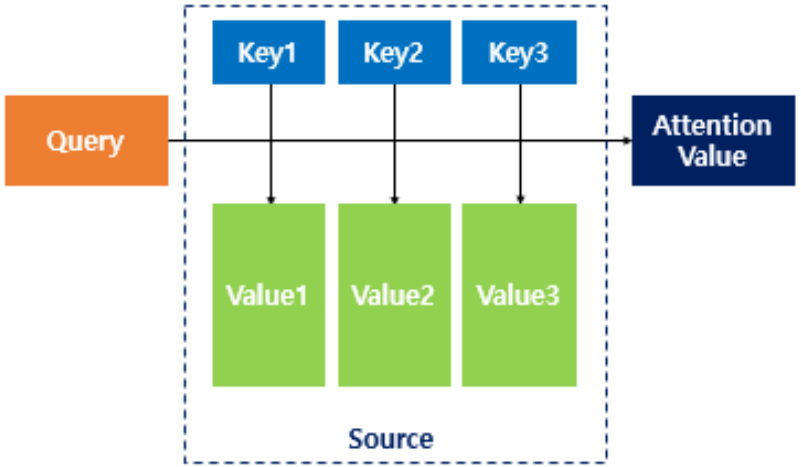
: Attention(Q*K)을 scaling

02. Attention

- **Attention (transformer)**

  : decoder에서 매 시점마다 Encoder의 전체 입력문장을 다시한번 참고하는 것

  : 문장 전체 참고 X → 해당 시점에서 예측해야 할 단어와 연관이 있는 부분만



{Key:Value}
Q: Query→ t시점의 decoder 셀에서의 은닉상태
K: Key → 모든 시점의 encoder 셀의 은닉상태
V: Value → 모든 시점의 encoder 셀의 은닉 상태

: Query에 대한 모든 Key와의 유사도 구함

: 구한 유사도를 Key와 Mapping되어 있는 Value에 반영

: 그 Value를 모두 더한 것 → attention value

02. Attention

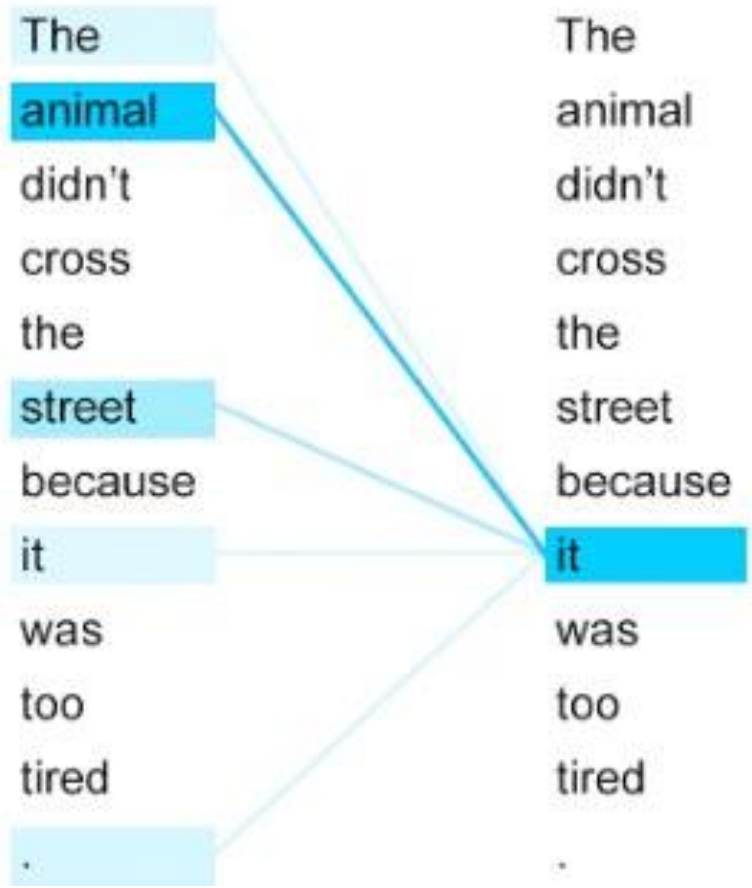
- Input sentence를 입력 받으면 3가지 종류의 차원으로 변환 → query, key, value

- **Self-Attention**

  : attention을 자기 자신에게 수행하는 것

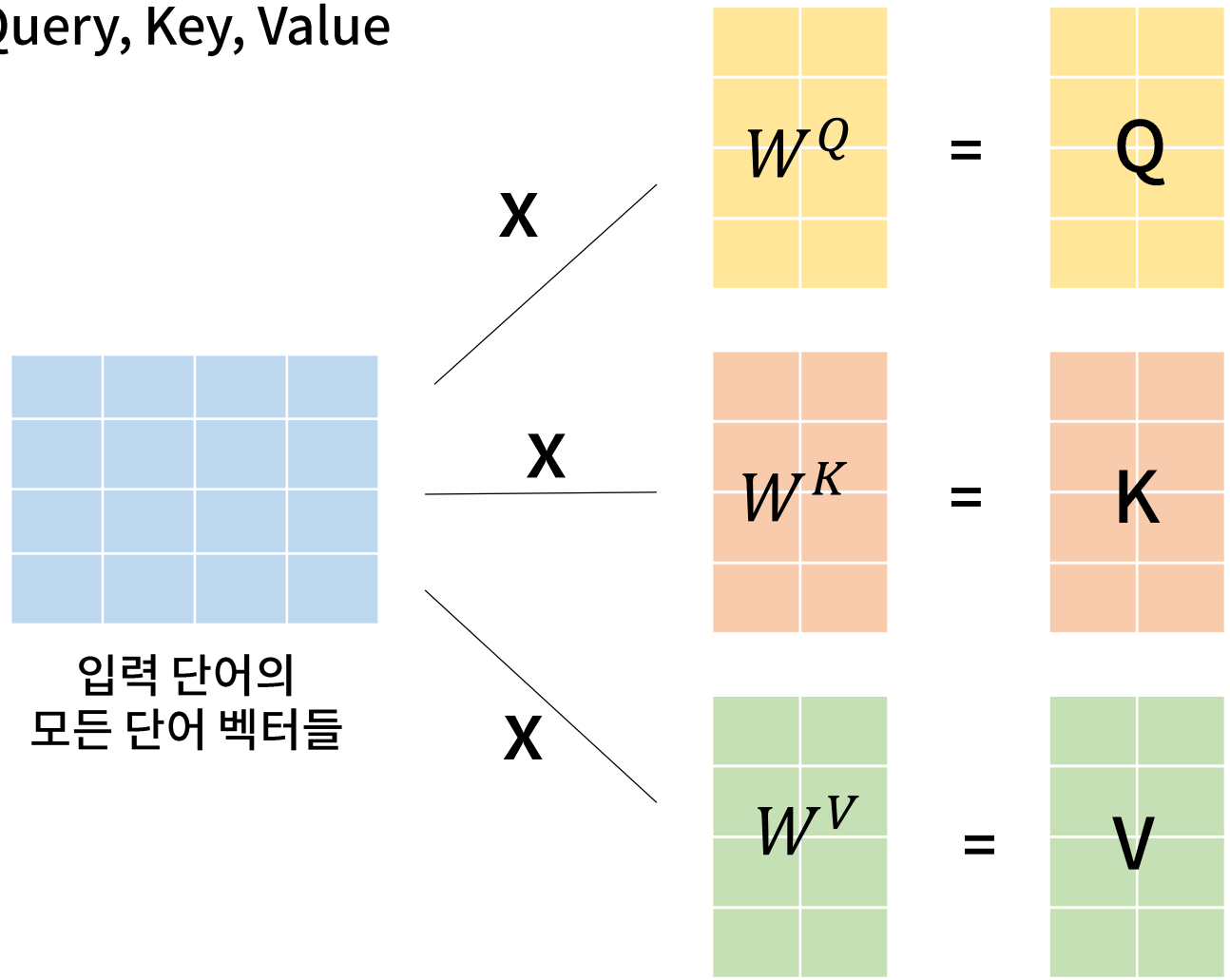
  : 효과

  → self-attention은 입력문장내의 단어들끼리 유사도를 구할 수 있음



02. Attention

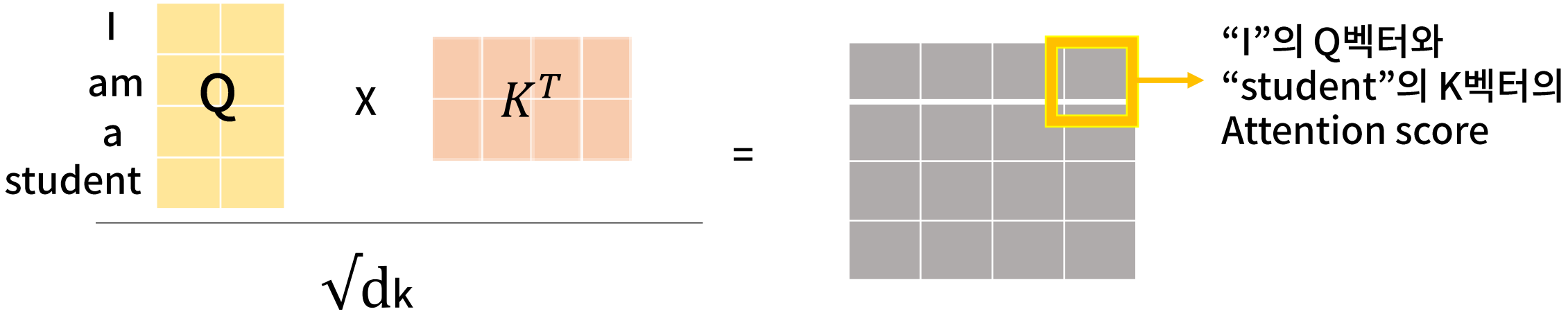
- Query, Key, Value



문장 행렬에 가중치 행렬을 곱해서
Q, K, V 행렬을 구함

02. Attention

1) 각 Q 벡터는 모든 K 벡터에 대한 Attention score 계산



$$\frac{Q \times K^T}{\sqrt{d_k}}$$

: d_k →key vector의 dimension

: d_k로 나누는 이유

→벡터의 차원이 커질수록 내적 값이 커질 가능성이 높고, 여기에 Softmax 를 만들면 극단적인 값들이 만들어지기 때문 (일종의 scaling)

02. Attention

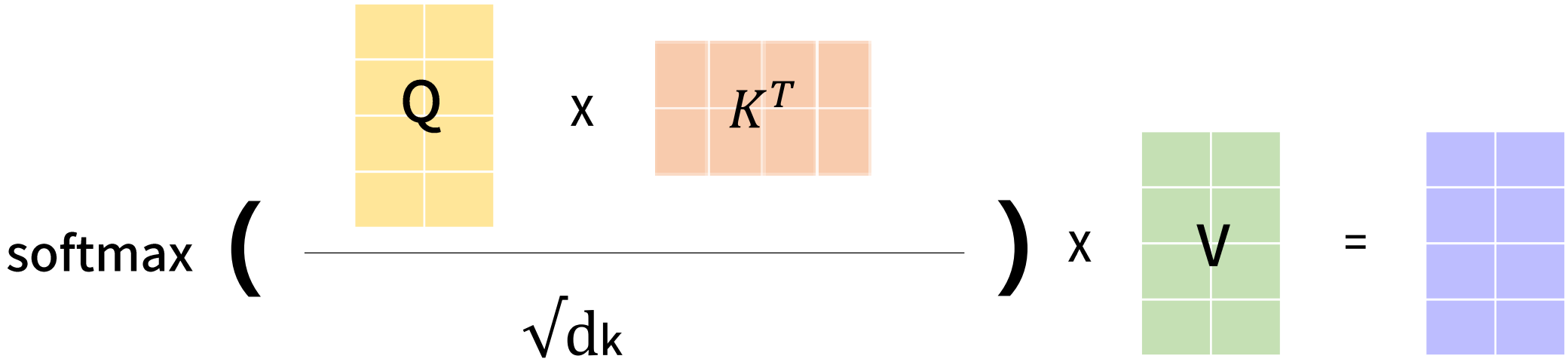
2) Attention distribution 계산

: 앞의 결과를 이용하여 모든 단어에 대한 Attention 값을 구해야 함

: Attention Score에 Softmax를 적용하면 그 결과가 Attention distribution

3) Attention value Matrix 계산

: Attention distribution에 V행렬 을 곱하면 그 결과가 Attention Value Matrix



02. Attention

- Multi-head Attention

: 한번의 Attention보다 여러 번 하는 것이 더 효율적

: 다양한 시각으로 볼 수 있다.

"The animal didn't cross the street because it was too tired"

-첫번째 Attention head: it과 animal의 연관도를 높게

-두번째 Attention head: it과 tired의 연관도를 높게

: d_{model}의 차원을 num_heads개로 나누어 Q, K, V에 대해 num_heads개의 병렬 Attention 수행

: 가중치 행렬 W^Q, W^K, W^V 도 Attention head마다 다 다름

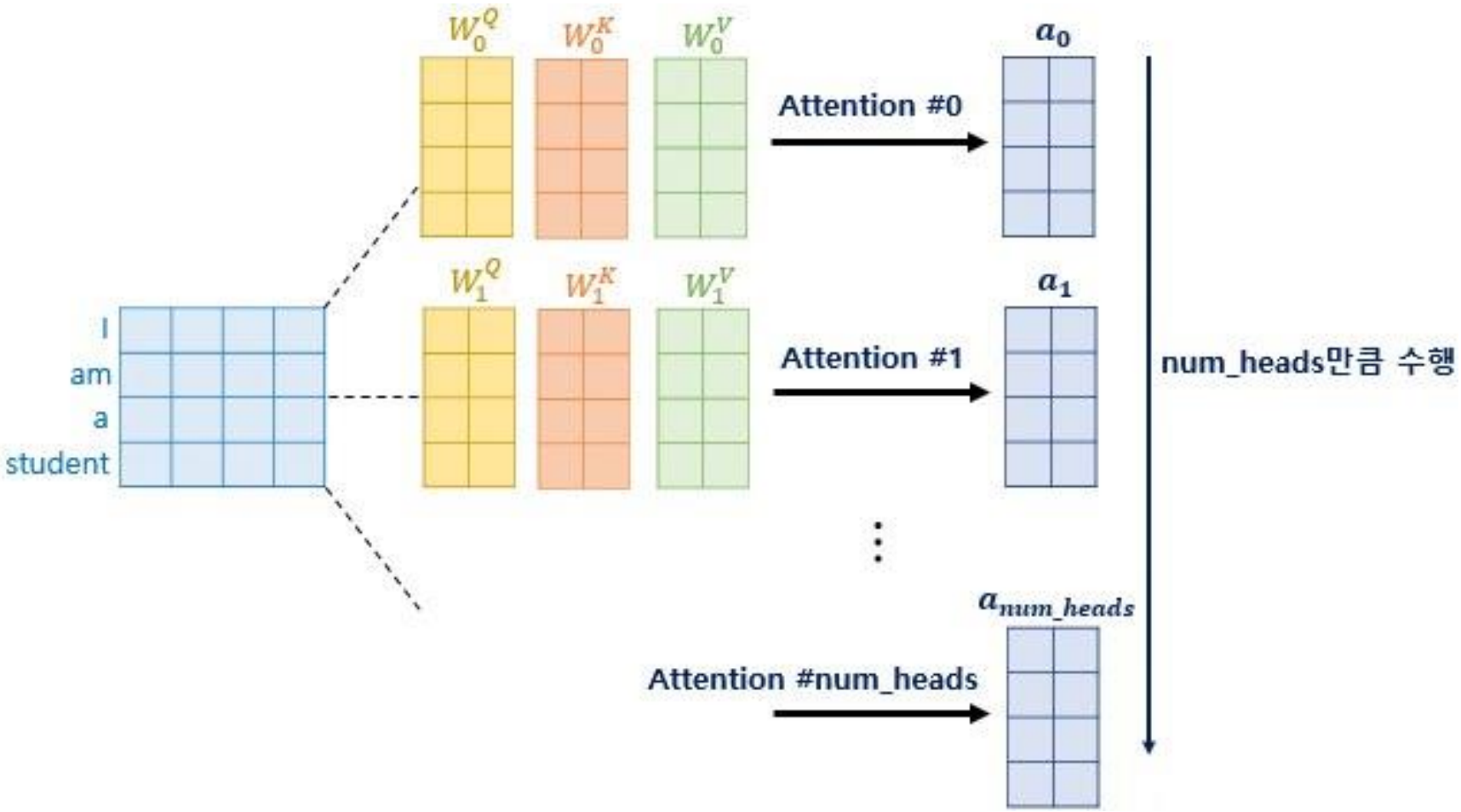
*Attention head

: num_heads가 8이면, 8개의 병렬 Attention

→ 이때 각각의 Attention 값 행렬: Attention head

02. Attention

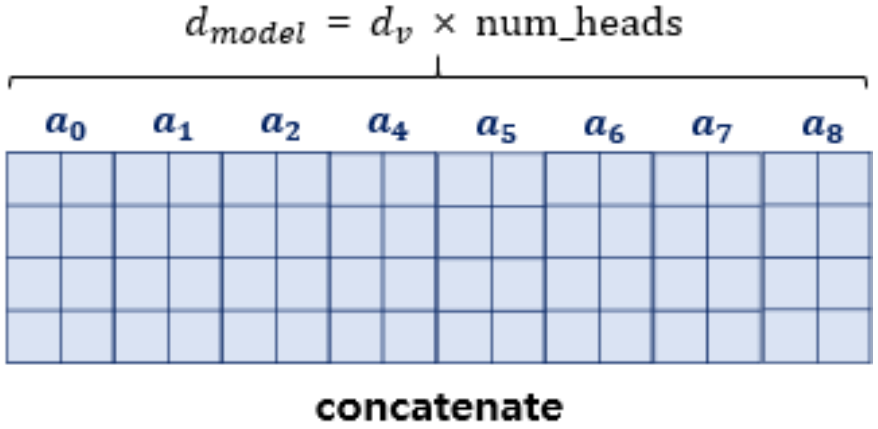
- Multi-head Attention



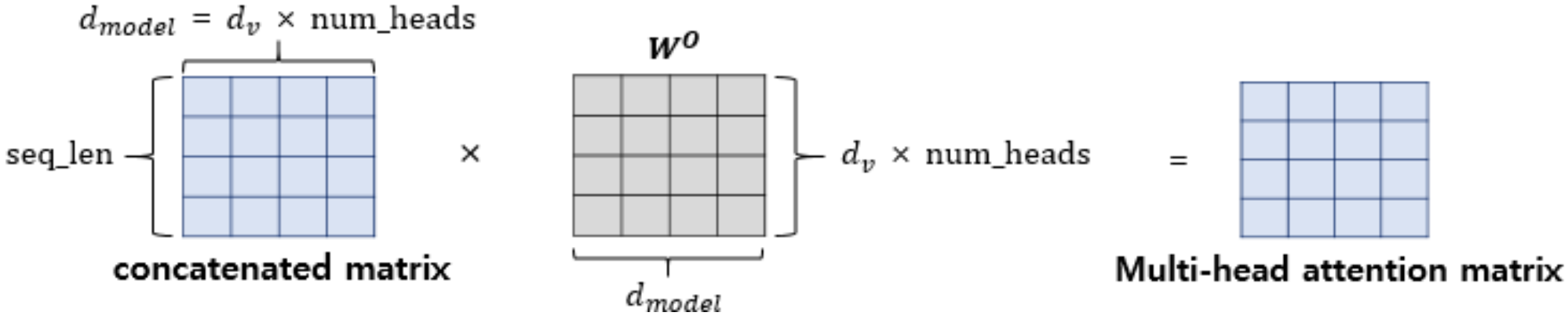
02. Attention

- Multi-head Attention

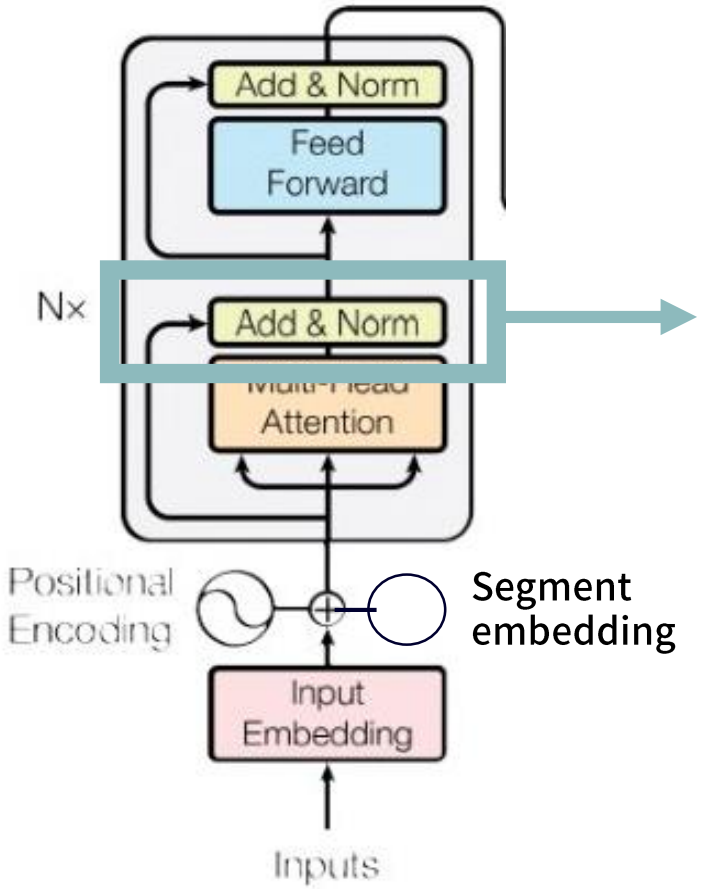
: 병렬 Attention을 모두 수행했으면 모든 Attention head를 concatenate



: concat한 행렬을 또 다른 가중치 W와 곱한다



02. BERT 구조

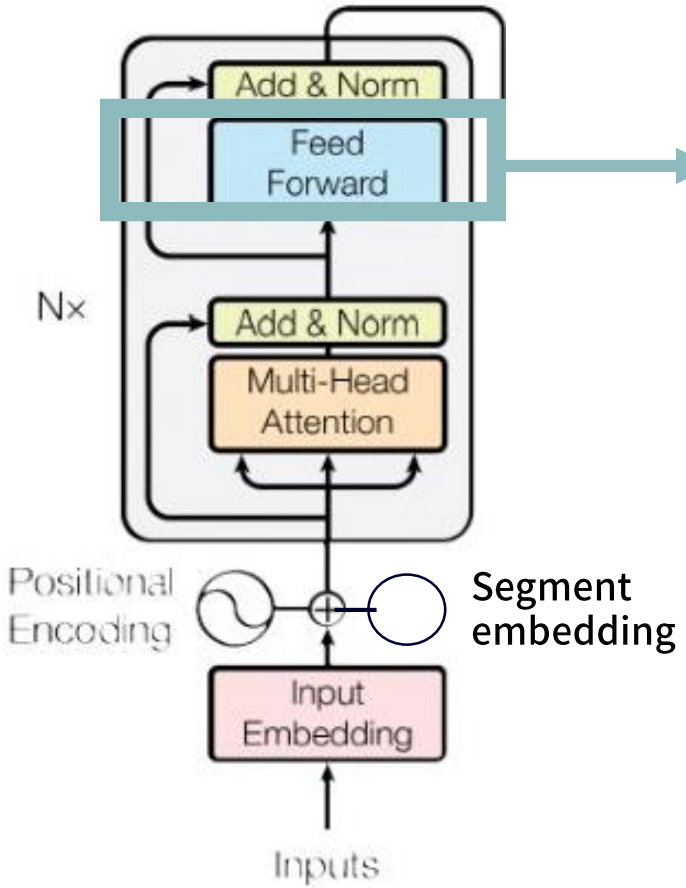


:(Multi-Attention head결과 matrix * linear projection)

+ 원래의 input

: 위의 결과 matrix에 layer_normalization

02. BERT 구조



: Transformer에서는 Relu

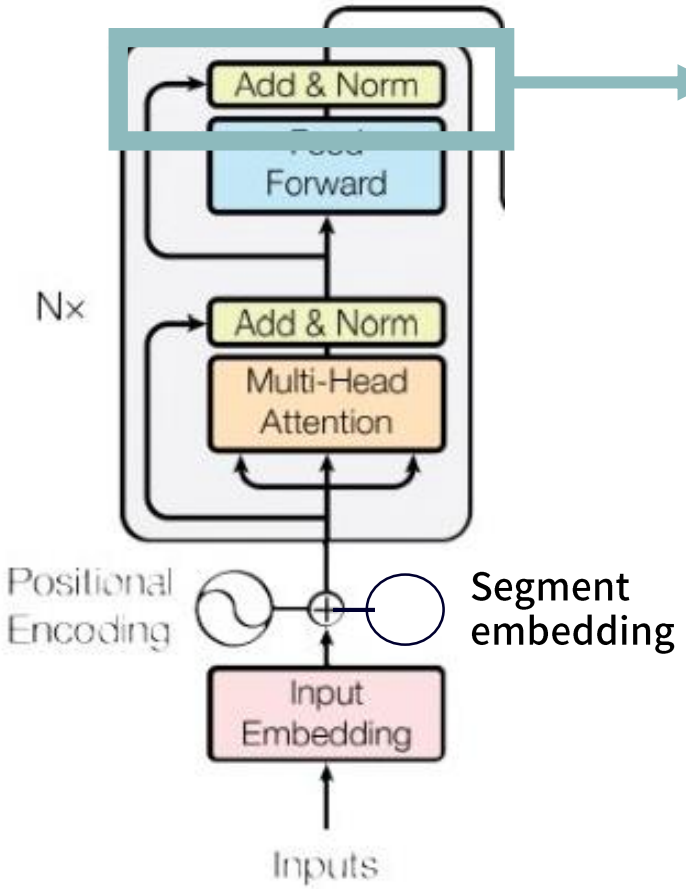
BERT에서는 보다 부드러운 Gelu

: Multi Head Attention에서 각 head가 자신의 관점으로만 문장을 self-Attention

→ 각 head에 따라 Attention이 치우침

: 각 head가 만들어낸 Self-Attention을 치우치지 않게, 균등하게 하는 역할

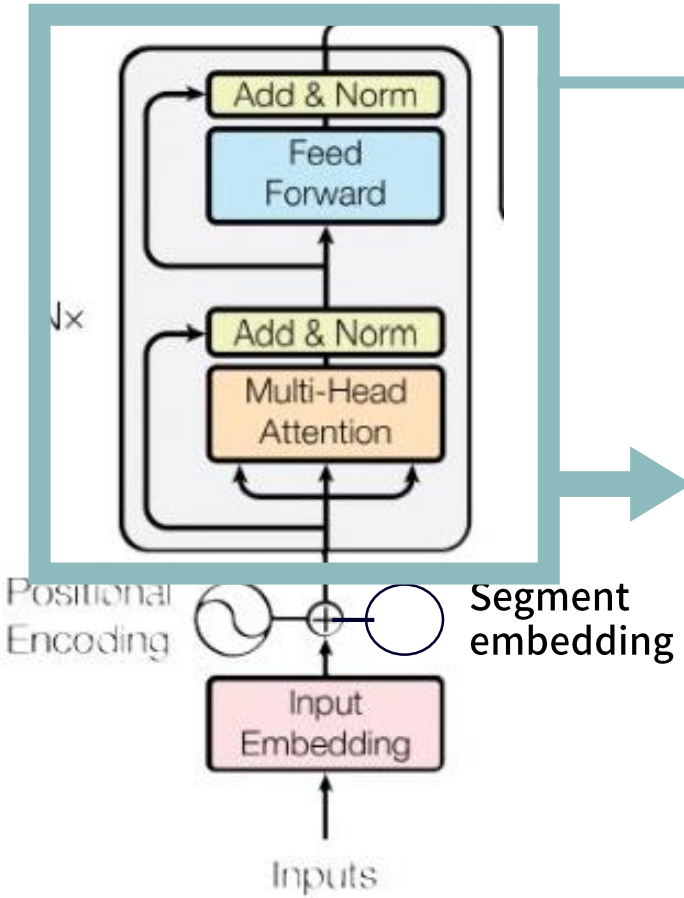
02. BERT 구조



: dropout한 후 이전의 Add&Norm에서 나온 값과 더함

: 더한 결과에 layer_normalization

02. BERT 구조



BERT의 단어 embedding vector

이 과정을
Block 수만큼 반복

03 Experiment

BERT

04. Experiment

- Pre-trained BERT model에, 하나의 output layer로 fine-tuning한 experiment

1) SQuAD dataset

: paragraph와 question pair가 주어지면 정답을 포함하는 text span을 찾는 문제

2) GLUE dataset

: 입력 sentence가 비문인지 아닌지

3) CoNLL-2003 dataset

: bio tagging된 data를 이용한 NER(Named Entity recognition)

: 우리의 과제 → NER

04. Experiment

- NER

  : 이름을 가진 개체(named entity)를 인식하겠다는 것

  : 어떤 이름을 의미하는 단어를 보고는 그 단어가 어떤 유형인지를 인식하는 것

  : [해리포터 보러 메가박스 가자]라는 문장이 있을 때 →

해	B-movie
리	I-movie
포	I-movie
터	I-movie
보	O
러	O
메	B-theater
가	I-theater
박	I-theater
스	I-theater
가	O
자	O

- BERT NER

  : biobert(BIO tagging을 이용한 NER) → NER task로 fine-tuning 진행

  → NER을 위한 output layer 추가하여 fine-tuning

04. Experiment

```
def get_bert_finetuning_model(model):
    inputs = model.inputs[:2] #segment, token 두개의 input
    dense = model.output
    out = keras.layers.Dropout(0.2)(dense)
    output_layer=keras.layers.Dense(7, activation='softmax')(out)
    model = keras.models.Model(inputs, output_layer)
    model.compile(

      optimizer=RAdam(learning_rate=LR, weight_decay=0.001),
      loss="categorical_crossentropy",
      metrics=["categorical_accuracy"])

    return model
```

Thank you