2019 D&A Conference

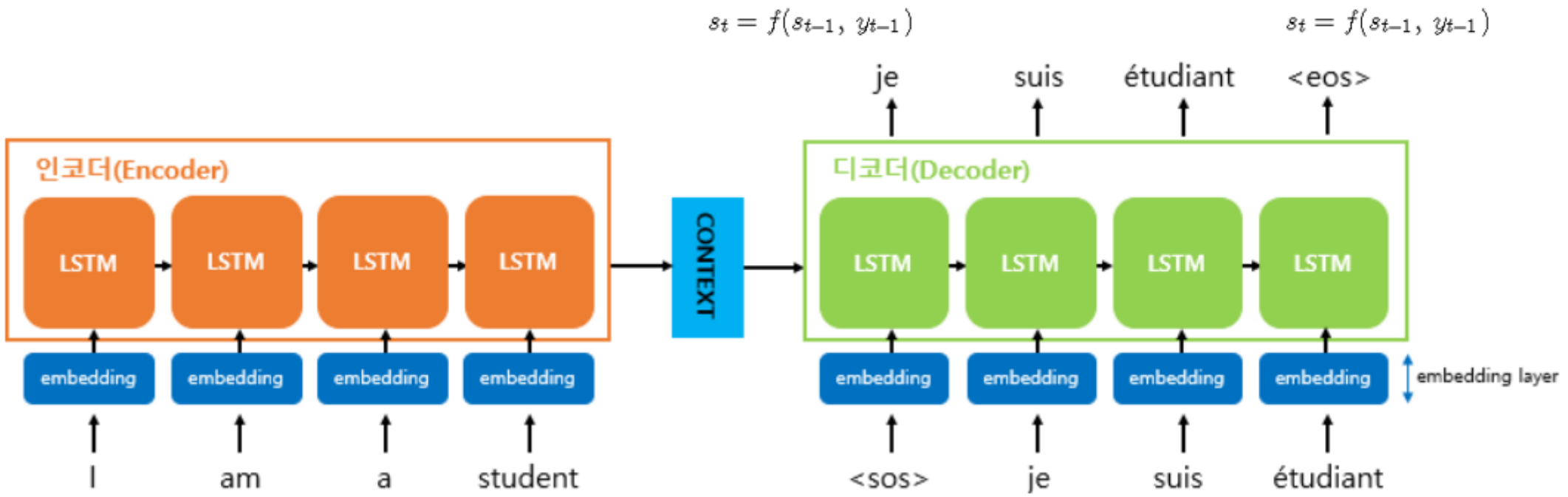
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

최영제

Index

0. Intro

0. Intro : Seq2Seq



- 두개의 RNN 구조를 연결한 Seq2Seq, 주로 번역과 같은 task에 사용
- Encoder를 통해 입력 값을 압축한 context vector를 생성하고 이를 받아서 decoder는 output을 내뱉음
- Decoder의 현재 시점 t에서 출력을 위해 필요한 입력 값은 직전 hidden state s_{t-1}과 output y_{t-1}

$$s_t = f(s_{t-1}, y_{t-1})$$

1. Attention

1. Attention : 배경

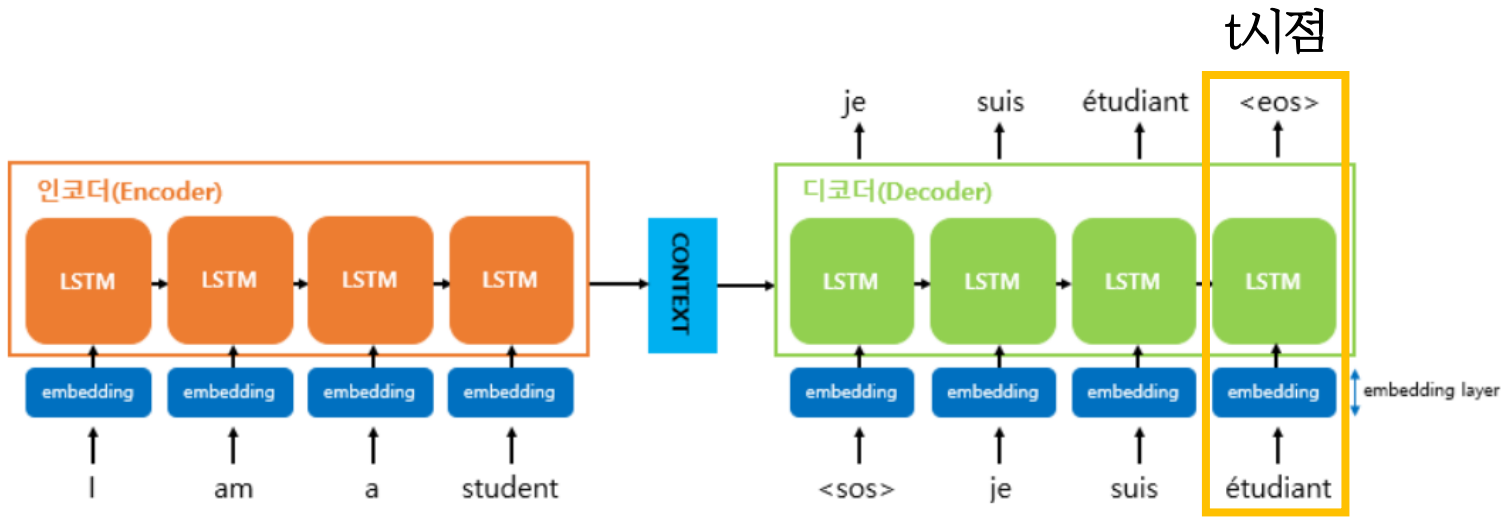
- 하지만 이러한 RNN에 기반한 seq2seq 모델에는 크게 두 가지 문제가 존재

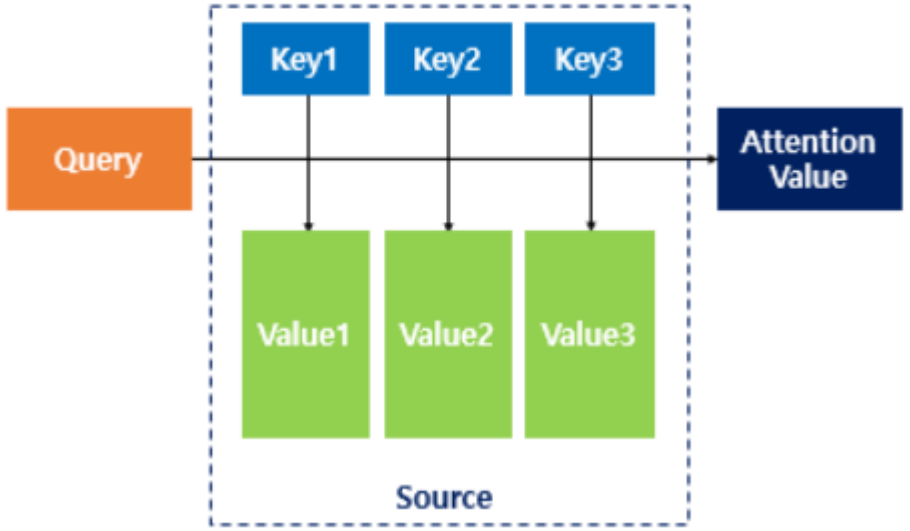
- 첫째, 하나의 고정된 크기의 벡터에 모든 정보를 압축하려고 하니까 정보 손실 발생
- 둘째, RNN의 고질적인 문제인 Vanishing Gradient 문제

- Attention의 기본 아이디어는 Decoder에서 출력 단어를 예측하는 매 시점(time step)마다, Encoder에서의 전체 입력 문장을 다시 한 번 참고하는 것

- 단, 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라 해당 시점에서 예측해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중해서 보는 것

1. Attention : Notation



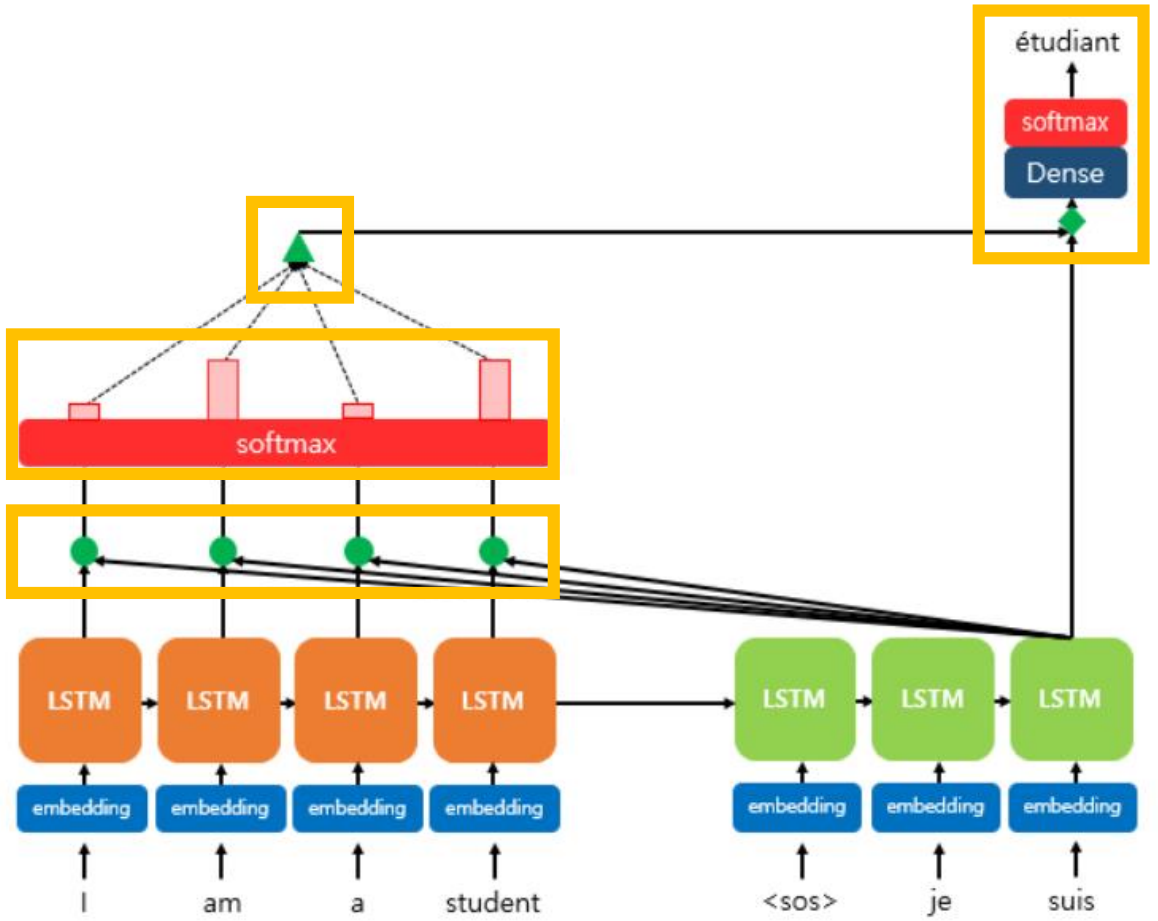


- Query(Q) = t-1 시점의 디코더 셀에서 은닉 상태
- Keys(K) = 모든 시점의 인코더 셀의 은닉 상태들
- Values(V) = 모든 시점의 인코더 셀의 은닉 상태들
- Attention(Q,K,V) = Attention Value

- Attention function은 주어진 'Query'에 대해서 모든 'Key'와의 유사도를 각각 구함
- 그 후 유사도를 키와 맵핑되어있는 각각의 'Value'에 반영
- 그 후 유사도가 반영된 'Value'을 모두 더해서 return. 이를 Attention Value라고 함

1. Attention : Dot_product Attention

1. Attention score를 구함
2. Softmax를 통해 Attention distribution을 구함
3. 각 인코더의 Attention weight와 Hidden state를 weighted sum 하여 Attention Value를 구함
4. Attention Value와 디코더의 t-1 시점hiddenstate를 concate하여 예측



즉 기존 Seq2Seq에서는 시점t에서 디코더 출력을 위해선 입력값 두개를 필요로 했으나
attention 메커니즘에서는 3개가 필요

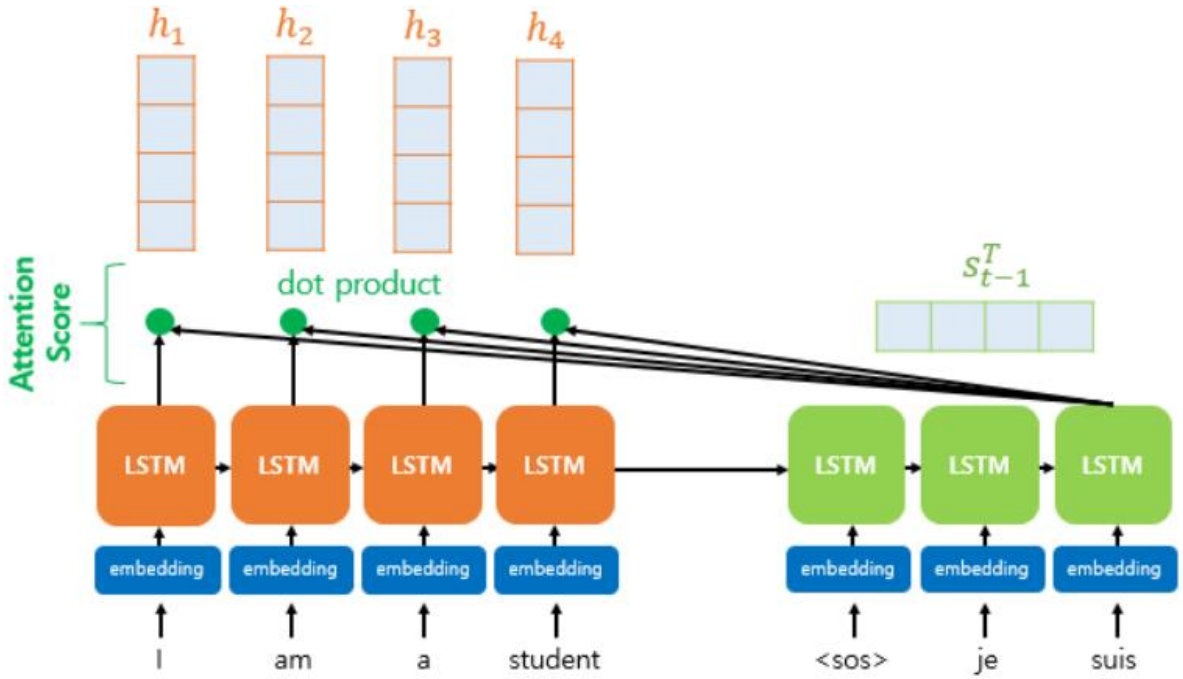
Seq2Seq : $s_t = f(s_{t-1}, y_{t-1})$

Attention : $s_t = f(s_{t-1}, y_{t-1}, a_t)$

1. Attention : Dot_product Attention

1. Attention score를 구함

$score(q, k) = q \cdot k$



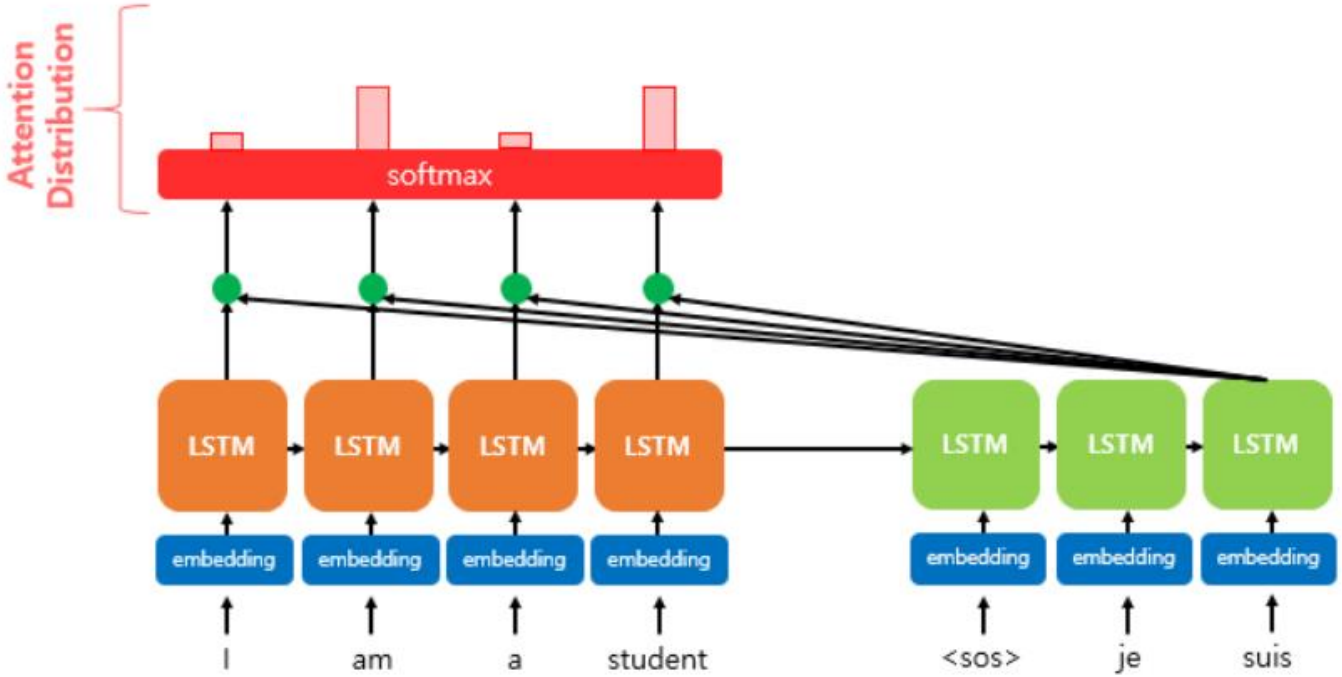
- Dot_product Attention은 내적을 통해 Attention score를 구한다.
- 내적을 통해 인코더의 모든 hidden state(K) 각각이 디코더의 바로 전 시점(Q)의 hidden state와 얼마나 유사한지를 판단하는 score

Attention score function : $score(s_{t-1}, h_i) = s_{t-1}^T h_i$

Attention set : $e^t = [s_{t-1}^T h_1, \ldots, s_{t-1}^T h_N]$

1. Attention : Dot_product Attention

2. Softmax를 통해 Attention distribution을 구함

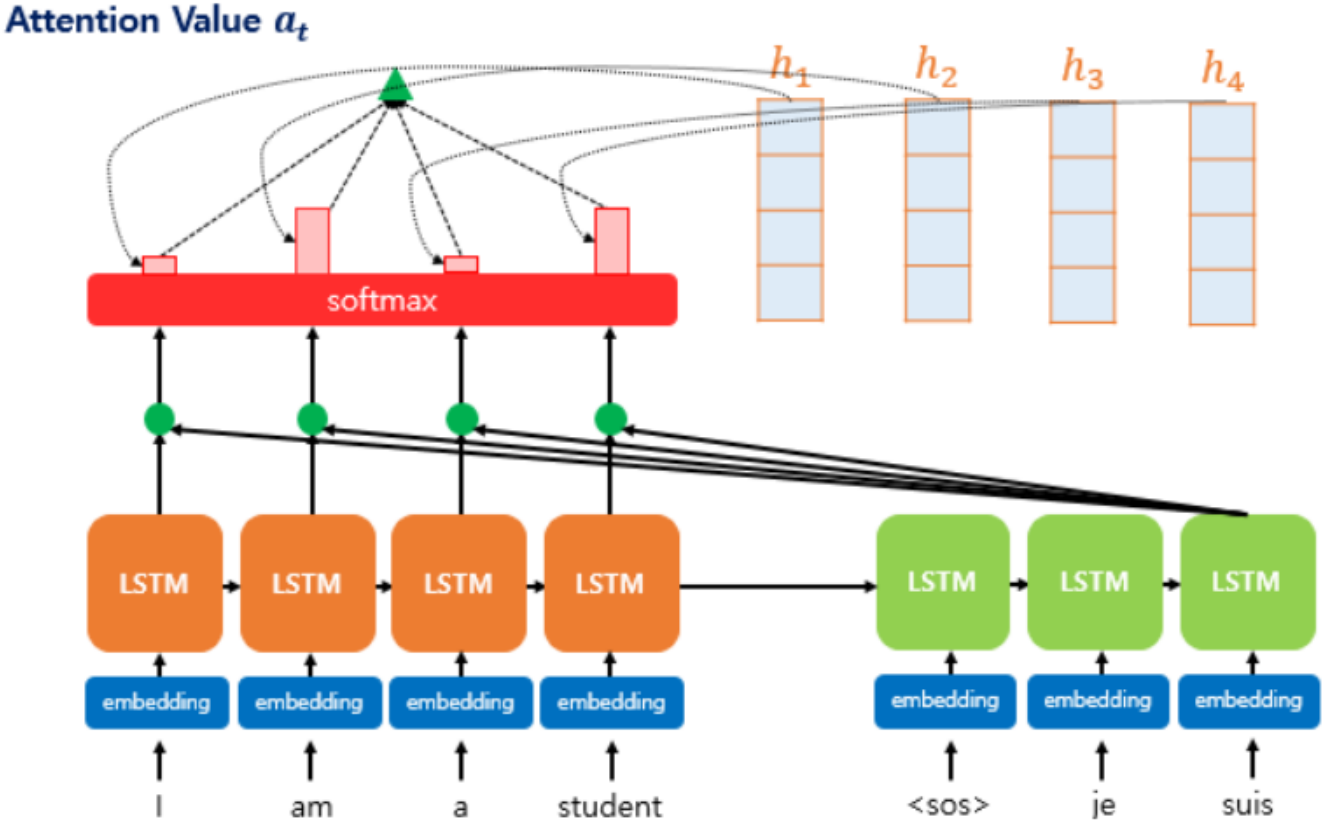


- Attention set에 softmax를 적용하여 모든 값을 합하면 1이 되는 확률 분포를 생성
- 이를 어텐션 분포(Attention Distribution), 각각의 값은 어텐션 가중치(Attention Weight)

Attention distribution : $\alpha^t = softmax(e^t)$

1. Attention : Dot_product Attention

3. 각 인코더의 Attention weight와 Hidden state를 weighted sum 하여 Attention Value를 구함

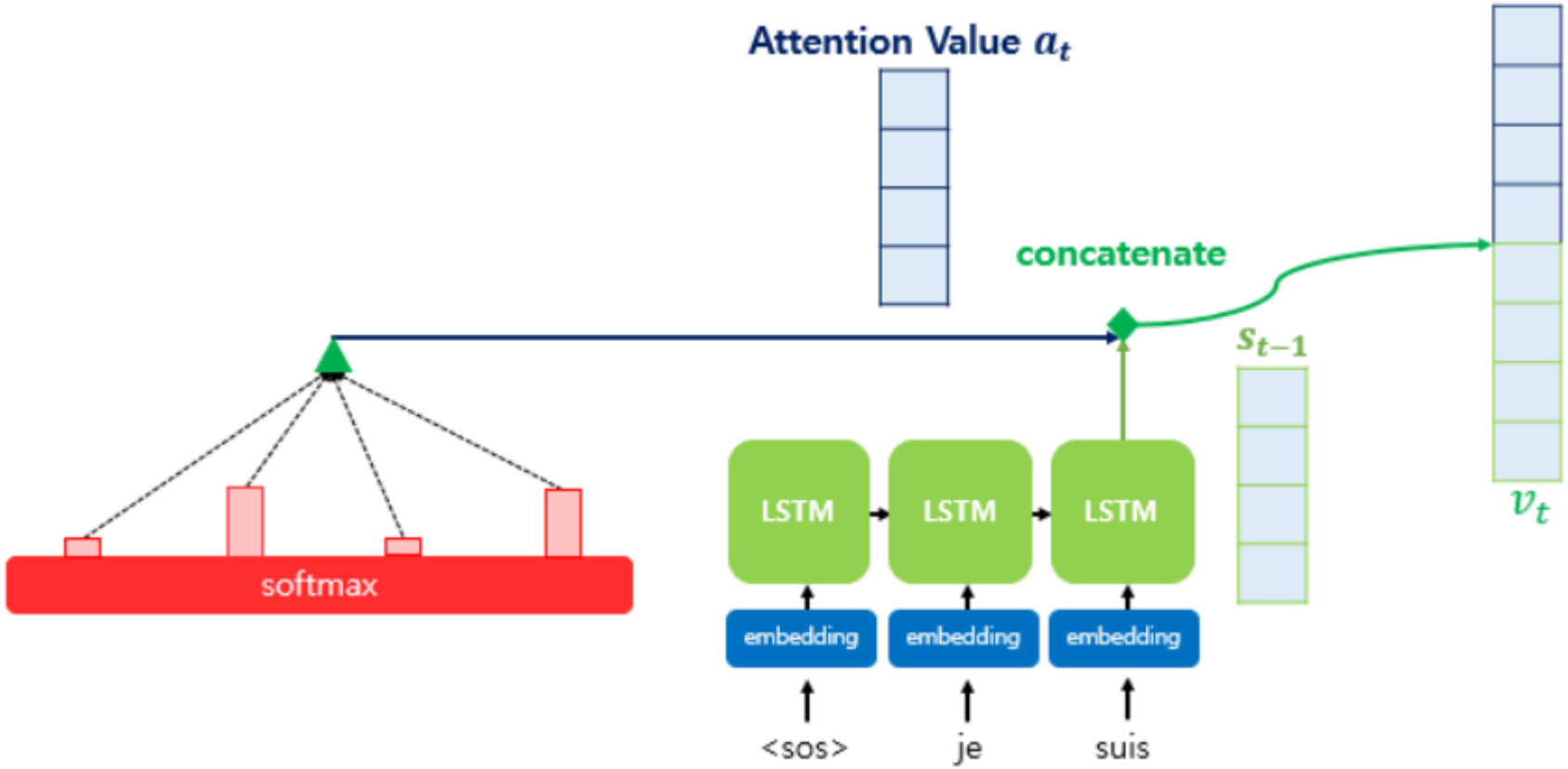


- 어텐션 가중치(Attention Weight)와 인코더의 hidden state 값(V)을 곱하고 이를 모두 더함(Weighted sum)
- 이를 Attention Value라고 함

Attention value : $a_t = \sum_{i=1}^{N} \alpha_i^t h_i$

1. Attention : Dot_product Attention

4. Attention Value와 디코더의 t-1 시점의 hidden state를 concate 하여 예측



- 생성된 Attention Value와 디코더의 t-1시점 hidden state를 concate하여 출력 $s_t = f(v_t, y_{t-1})$

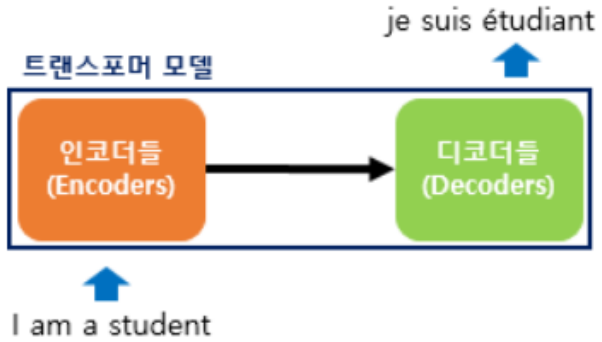
2. Transformer

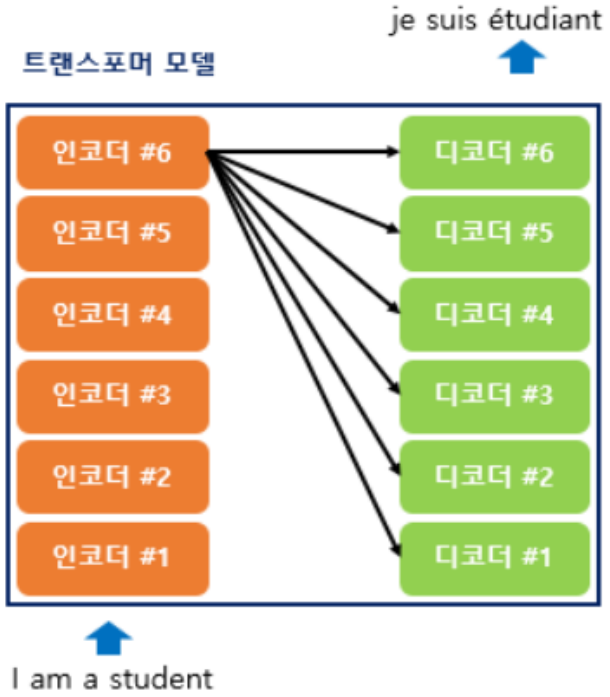
2. Transformer : Background

- 기존의 seq2seq 모델 구조에서 입력 시퀀스의 정보가 일부 손실된다는 단점 발생, 이를 보정하기 위해 Attention 사용

- Attention을 보정을 위한 용도가 아닌 Attention만 사용하여 Encoder와 Decoder를 구축

- 주요 하이퍼 파라미터
  - D_model = 512, 인코더, 디코더의 입출력 크기 = 임베딩 차원
  - Num_layers = 6, 인코더와 디코더 개수
  - Num_head = 8, 병렬적으로 진행할 head 개수





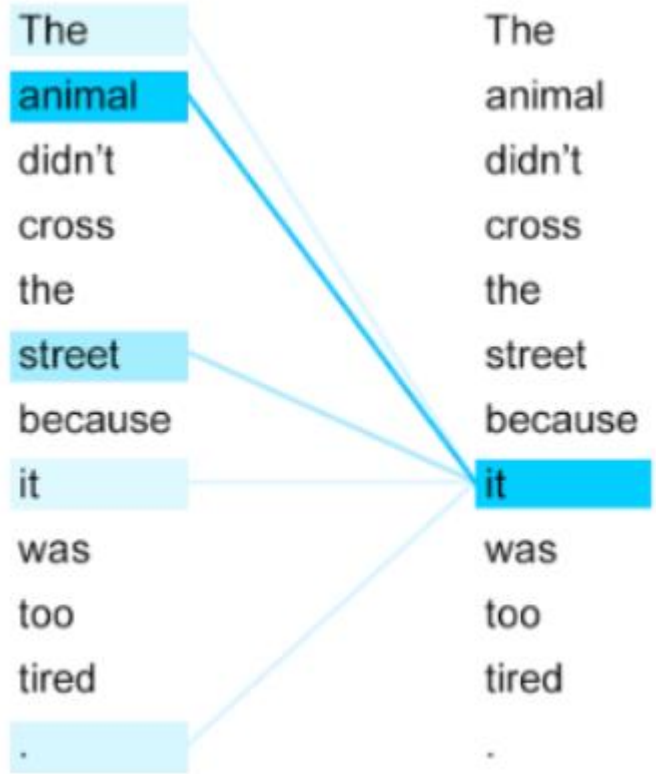
2. Transformer : Self attention

- 기존 Attention에서의 Q,K,V
  Query(Q) = t-1 시점의 디코더 셀에서 은닉 상태
  Keys(K) = 모든 시점의 인코더 셀의 은닉 상태들
  Values(V) = 모든 시점의 인코더 셀의 은닉 상태들

- 셀프 어텐션은 입력 문장 내의 단어들끼리 유사도를 구하므로서, 따라서 self attention 안에서는 Q,K,V가 모두 동일

- Query(Q) = 입력 문장의 모든 단어 벡터들
- Keys(K) = 입력 문장의 모든 단어 벡터들
- Values(V) = 입력 문장의 모든 단어 벡터들

- 우측 그림에서와 같이 같은 문장안에서 it과 animal의 높은 연관성을 찾을 수 있음



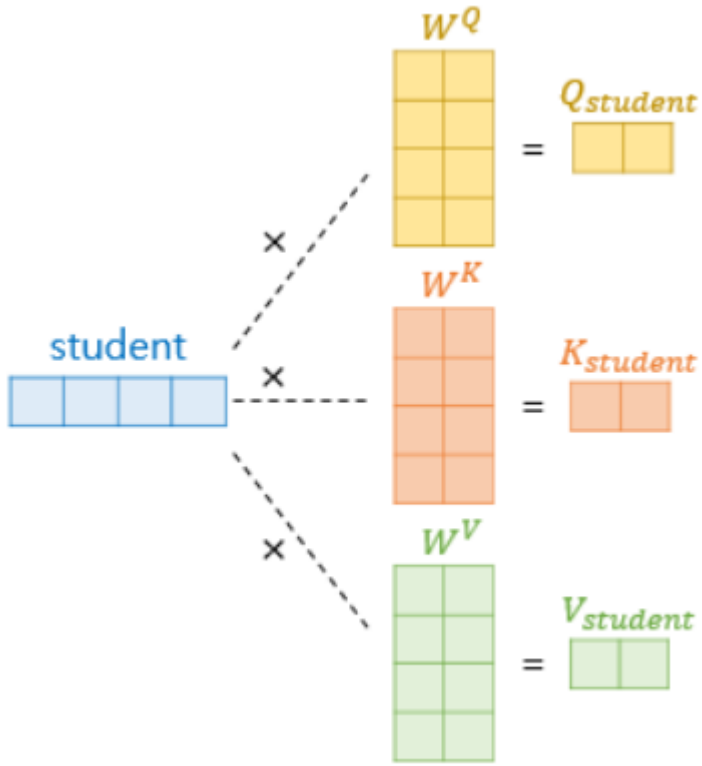
2. Transformer : Self attention

- 셀프 어텐션은 인코더의 초기 입력인 d_model의 차원을 가지는 단어 벡터들을 사용하여 셀프 어텐션을 수행하는 것이 아니라 우선 각 단어 벡터들로부터 Q벡터, K벡터, V벡터를 얻는 작업을 거침

- 이 Q벡터, K벡터, V벡터들은 초기 입력인 d_model의 차원을 가지는 단어 벡터들보다 더 작은 차원을 가지는데, 논문에서는 dmodel=512의 차원을 가졌던 각 단어 벡터들을 64의 차원을 가지는 Q벡터, K벡터, V벡터로 변환

- 64라는 값은 트랜스포머의 또 다른 하이퍼파라미터인 num_heads로 인해 결정되는데, 트랜스포머는 dmodel을 num_heads로 나눈 값을 각 Q벡터, K벡터, V벡터의 차원으로 결정

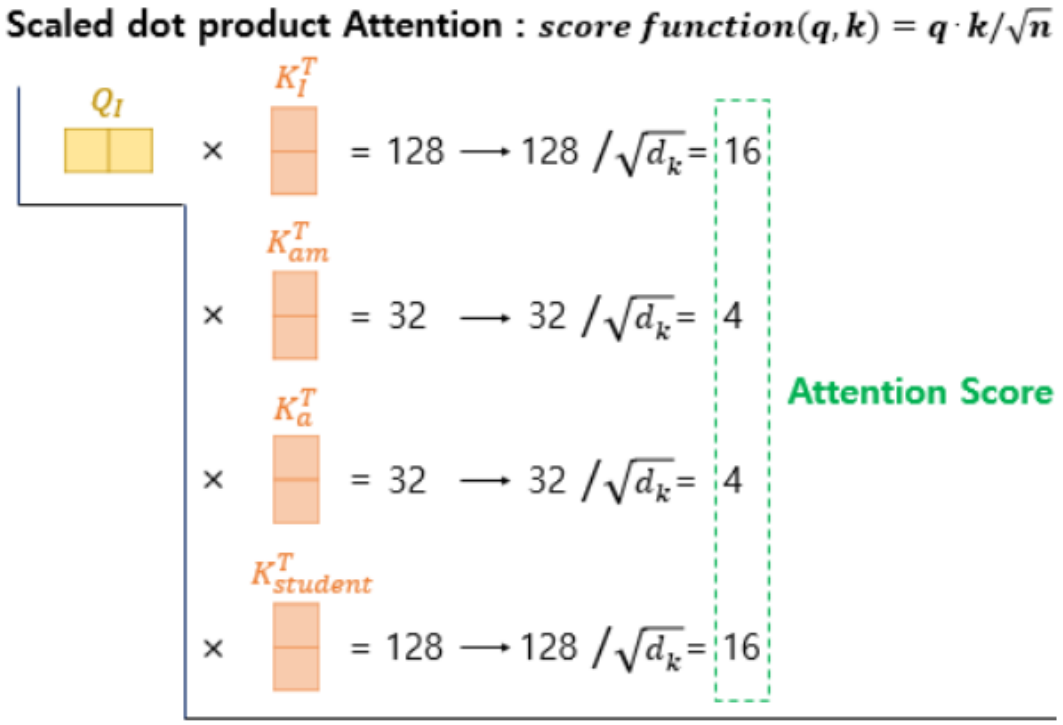
- 기존의 벡터로부터 더 작은 벡터는 가중치 행렬을 곱함으로써 완성되며 training 과정에서 학습이 됨



2. Transformer : Self attention

1. Attention score를 구한다.

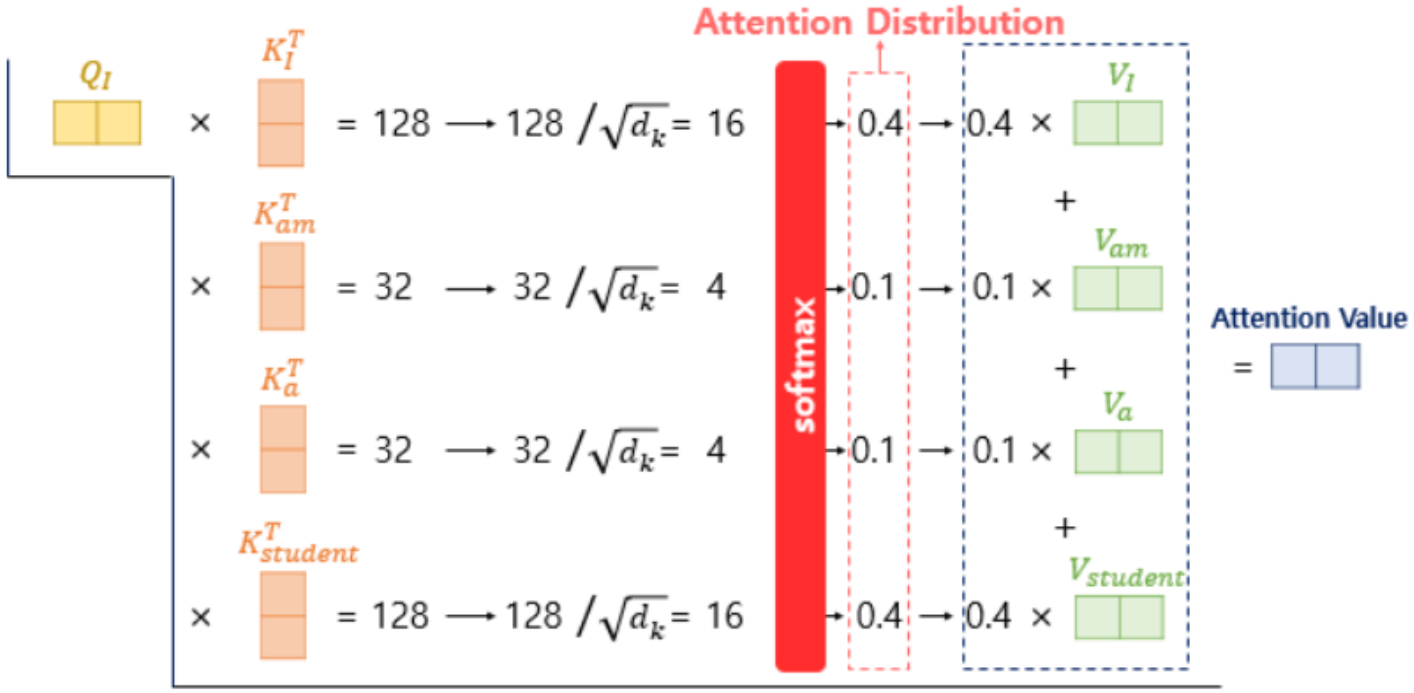
$$score(q,k) = q \cdot k/\sqrt{n}$$



- 기존 Dot_product attentio에서 sacling을 해주어 Scaled dot product attention이라고 함

2. Transformer : Self attention

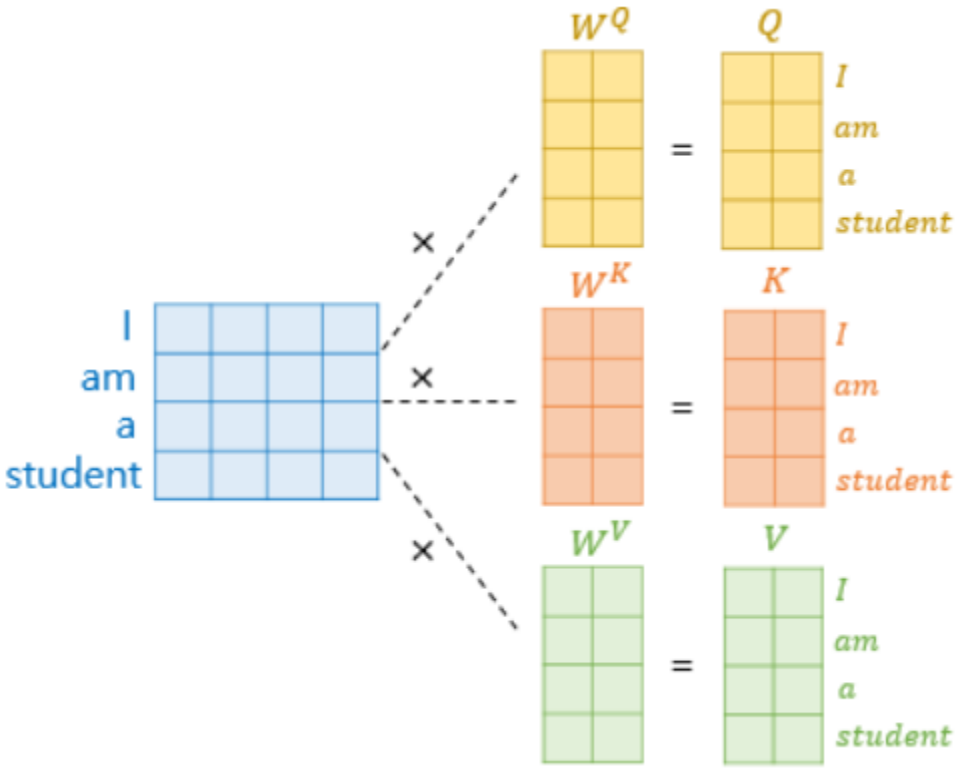
2 ,3 :Attention Distribution 및 Weighted sum을 통한 Attention Value 계산



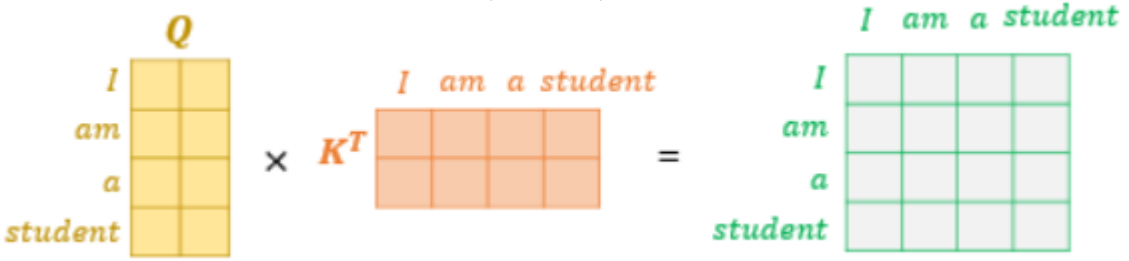
- Attention score에 소프트맥스 함수를 사용하여 Attention Distribution을 구하고, 각 V 벡터와 Weighted sum을 통해 Attention Value을 구함
- 여기서 나온 Attention Value는 i와 student의 속성이 강하게 반영된 값

2. Transformer : Self attention

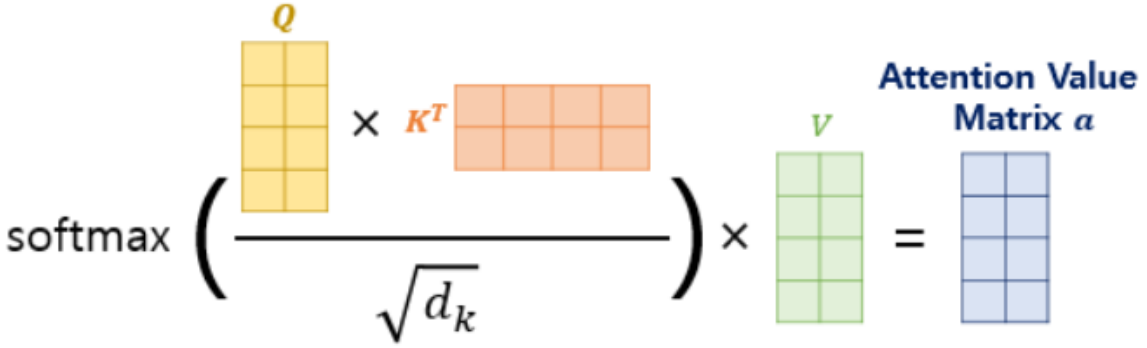
0. 초기 값에서 dense layer를 거친 Q,K,V



1. Attention score를 구한다.



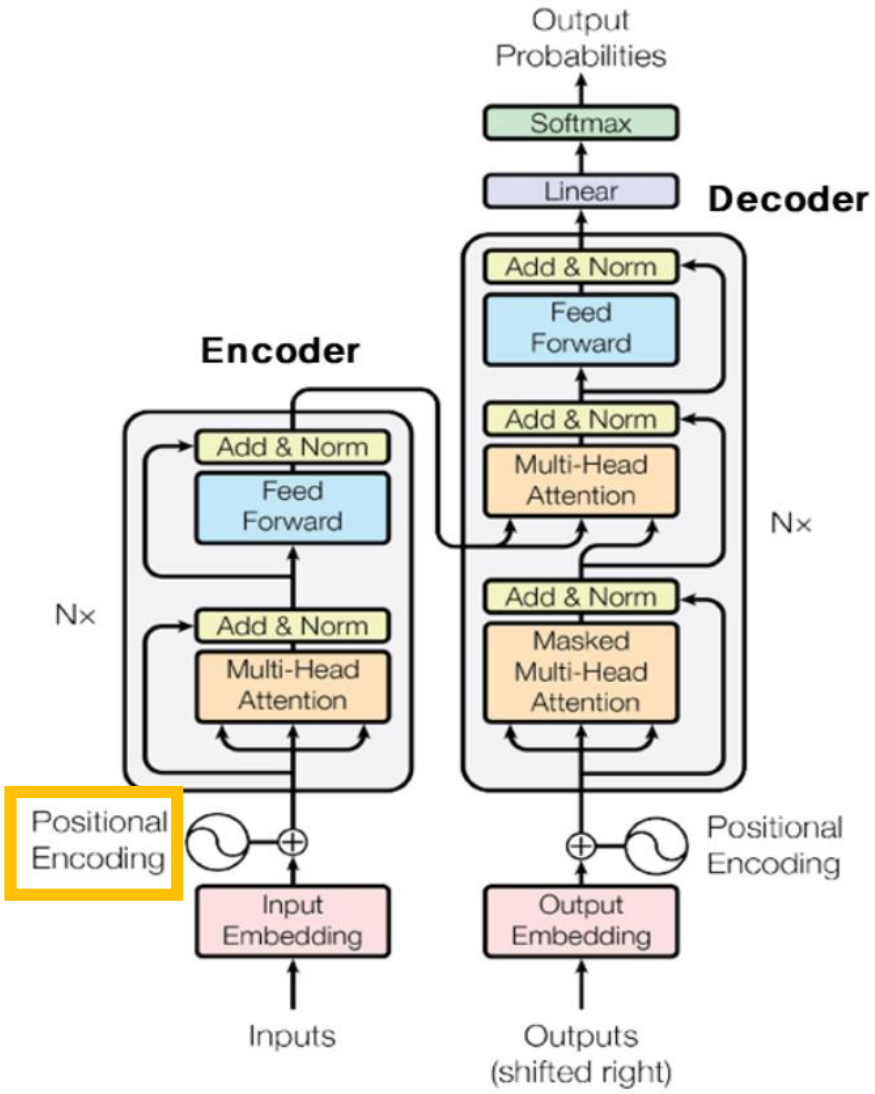
2 – 3 :Attention Distribution 및 Attention Value 계산



- 이 과정은 행렬 연산을 통해 일괄적으로 진행된다

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

2. Transformer：Architecture



2. Transformer : Positional Encoding

- Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. (Vaswani et al., Attention Is All You Need, 2017)

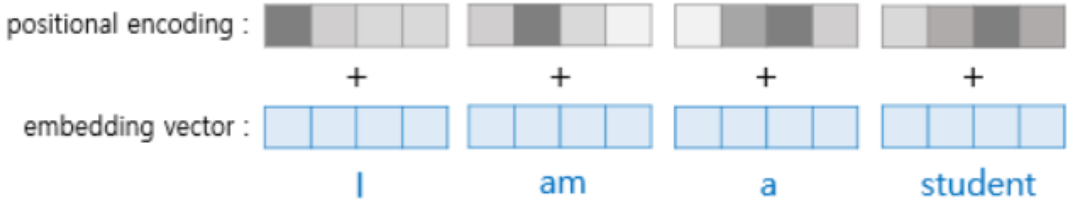
- 토큰의 상대적 또는 절대적 위치에 대한 정보를 주입하는데 이를 Positional Encoding 이라 함

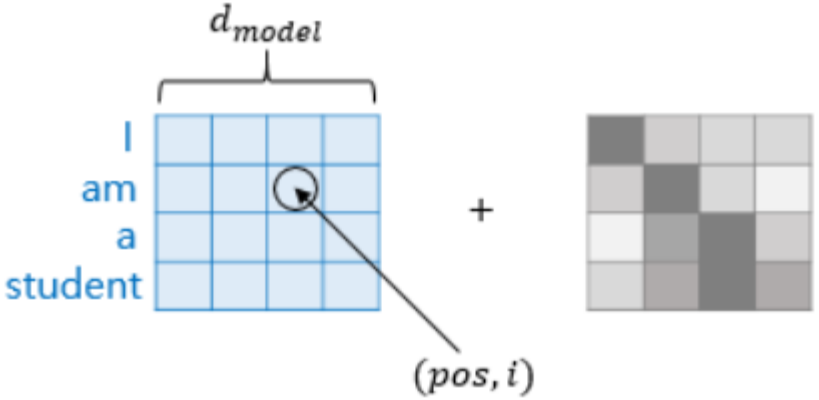
- 사인 함수sinusoid function를 이용한 아래 수식의 결과를 더해 네트워크가 토큰의 상대적 위치와 관련한 정보를 학습할 수 있게 한다.

$$\mathrm{PE}_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$\mathrm{PE}_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

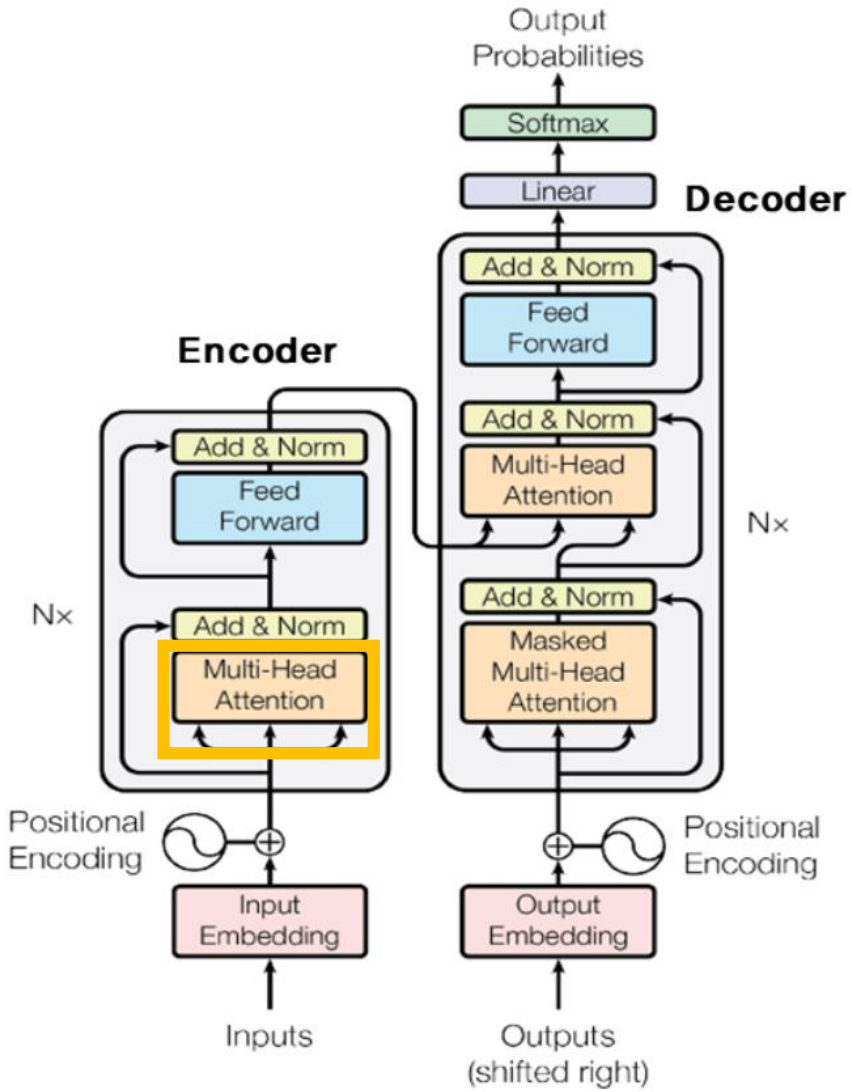
- pos는 position ,i는 dimension을 뜻함

- 각 임베딩 벡터에 포지셔널 인코딩값을 더하면 같은 단어라고 하더라도 문장 내의 위치에 따라서 트랜스포머의 입력으로 들어가는 임베딩 벡터의 값이 달라짐

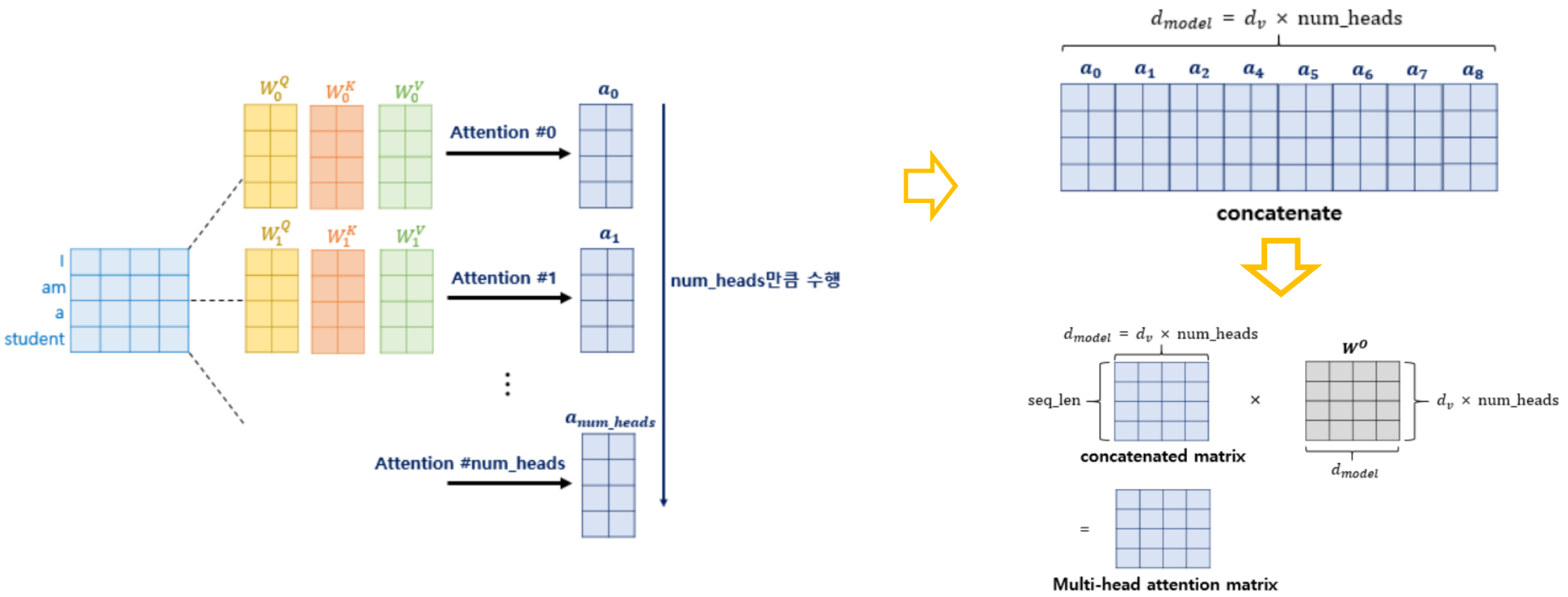




2. Transformer：Architecture

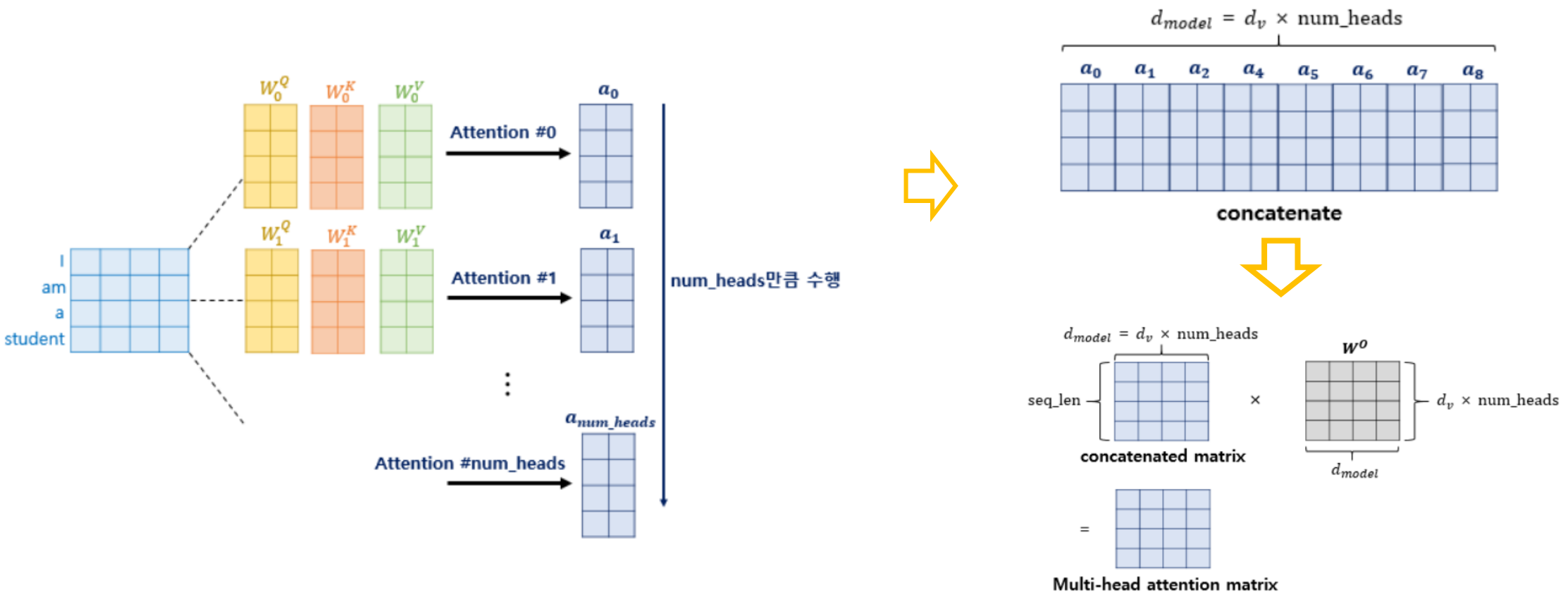


2. Transformer : Multi head self attention



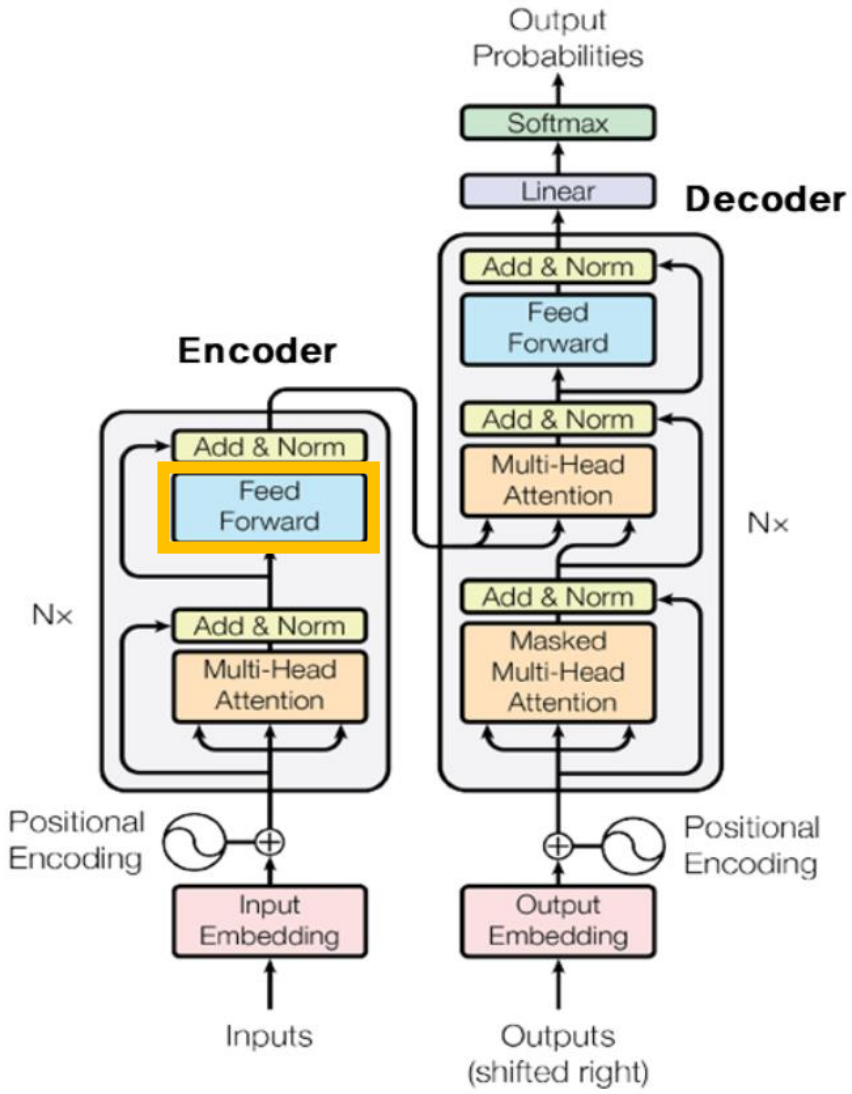
- 트랜스포머 연구진은 한 번의 어텐션을 하는 것보다 여러번의 어텐션을 병렬로 사용하는 것이 더 효과적이라고 판단
- D_model의 차원을 num_heads개로 나누어 dmodel/num_heads의 차원을 가지는 Q, K, V에 대해서 num_heads개의 병렬 어텐션을 수행
- 각각의 어텐션 값 행렬을 어텐션 헤드라고 부르며, 이때 가중치 행렬 W_Q,W_K,W_V의 값은 8개의 어텐션 헤드마다 전부 다름

2. Transformer : Multi head self attention

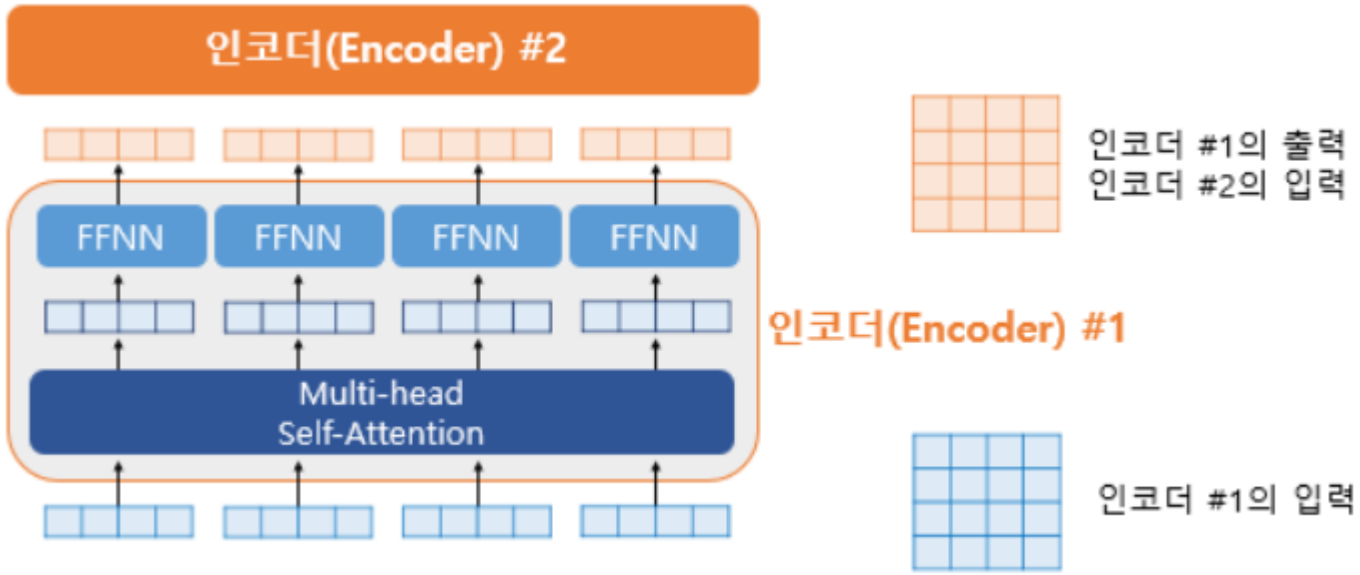


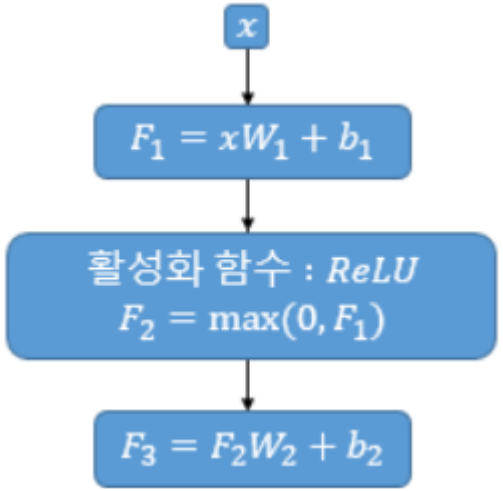
- Why? '그 동물은 길을 건너지 않았다. 왜냐하면 그것은 너무 피곤하였기 때문이다.'
- 단어 it에 대한 Query로 부터 다른 단어와의 연관도를 구하였을 때 n번째 어텐션 헤드는 '동물(animal)'의 연관도를 높게, k번째 어텐션 헤드는 '피곤 (tired)'과의 연관도를 높게 볼 수 있음.
- 각 어텐션 헤드는 전부 다른 시각에서 보고있기 때문
- 이렇게 나온 결과 행렬이 멀티-헤드 어텐션의 최종 결과물

2. Transformer：Architecture



2. Transformer : Position-wise FFNN



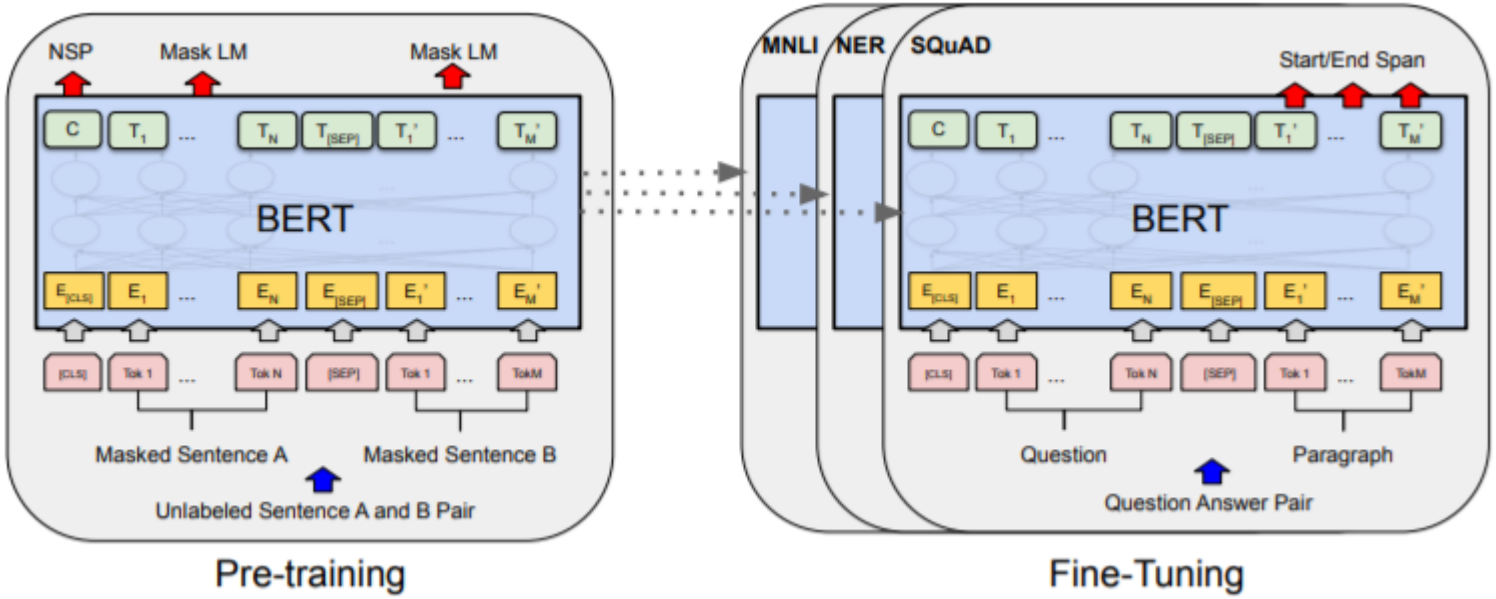



```
def point_wise_feed_forward_network(d_model, dff):
  return tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),  # (batch_size, seq_len, dff)
      # 활성화 함수 relu는 첫번째 층에만 배치한다.
      tf.keras.layers.Dense(d_model)  # (batch_size, seq_len, d_model)
  ])
```

- 두개의 dense layer를 통과시킴, 여기서 node의 개수는 2,048개
- 활성 함수는 Relu를 사용하였으며 여기서 나온 아웃풋이 encoder#n의 아웃풋이며 이는 encoder#n+1의 인풋이 된다.

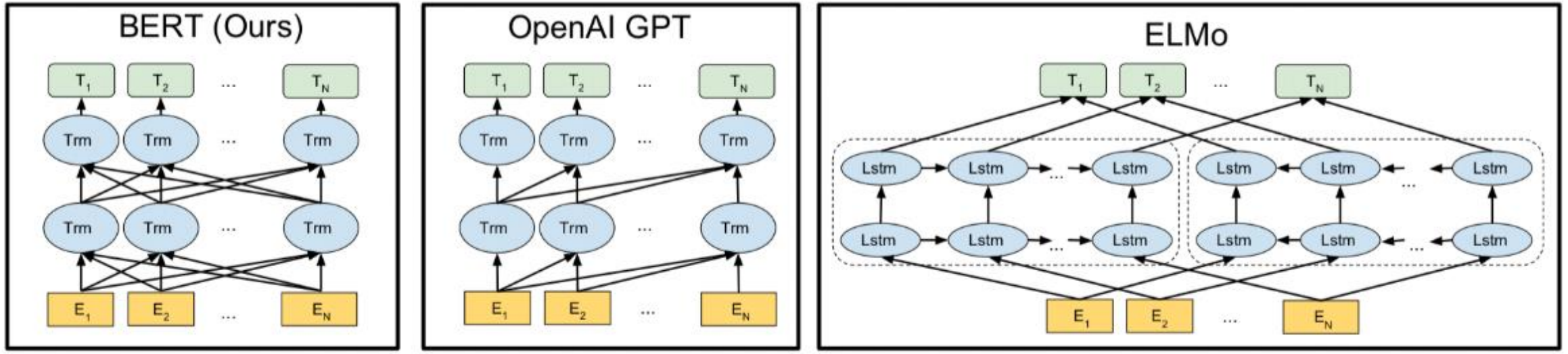
3. BERT

3. BERT : Intro



- There are two steps in our framework: pre-training and fine-tuning
- During pre-training, the model is trained on unlabeled data over different pre-training tasks
- For fine-tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks.

3. BERT : Intro



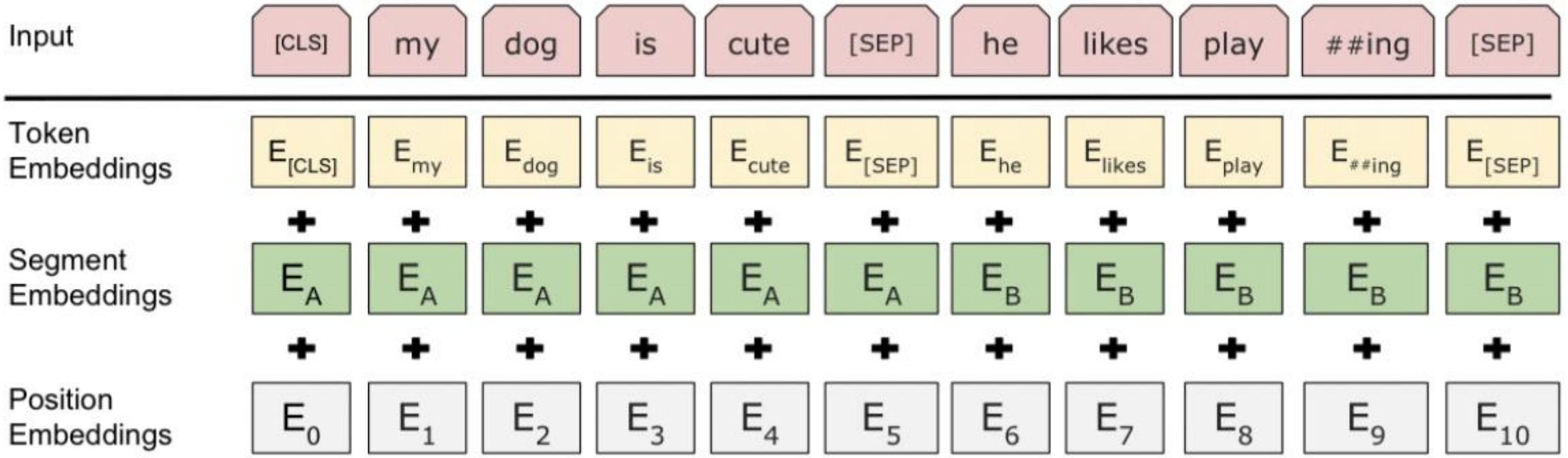
- There are two existing strategies for applying pre-trained language representations to downstream tasks: feature-based and fine-tuning.

- Trm is the abbreviation of Transformation.

* BERT는 transformer에서 encoder 부분만 사용

- OpenAI GPT use a left-toright architecture, where every token can only attend to previous tokens in the self-attention layers of the Transformer (Vaswani et al., 2017). Such restrictions are could be very harmful when applying finetuning based approaches to token-level tasks such as question answering, where it is crucial to incorporate context from both directions

3. BERT : Input Embedding



- A “sequence” refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together
- We use WordPiece embeddings (Wu et al., 2016) with a 30,000 token vocabulary.
- The first token of every sequence is always a special classification token ([CLS]), The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.

3. WordPiece embedding

- Word Piece Model 은 제한적인 vocabulary units, 정확히는 단어를 표현할 수 있는 subwords units 으로 모든 단어를 표현

- bag of words model 는 단어 개수 만큼의 차원을 지닌 벡터 공간을 이용, 단어가 커지면 모델이 무거워지며 제한된 개수만 사용하기에는 OOV 문제가 발생

- 글자 (characters)를 subword units 으로 이용합니다. 영어는 알파벳을 유닛으로 이용합니다.

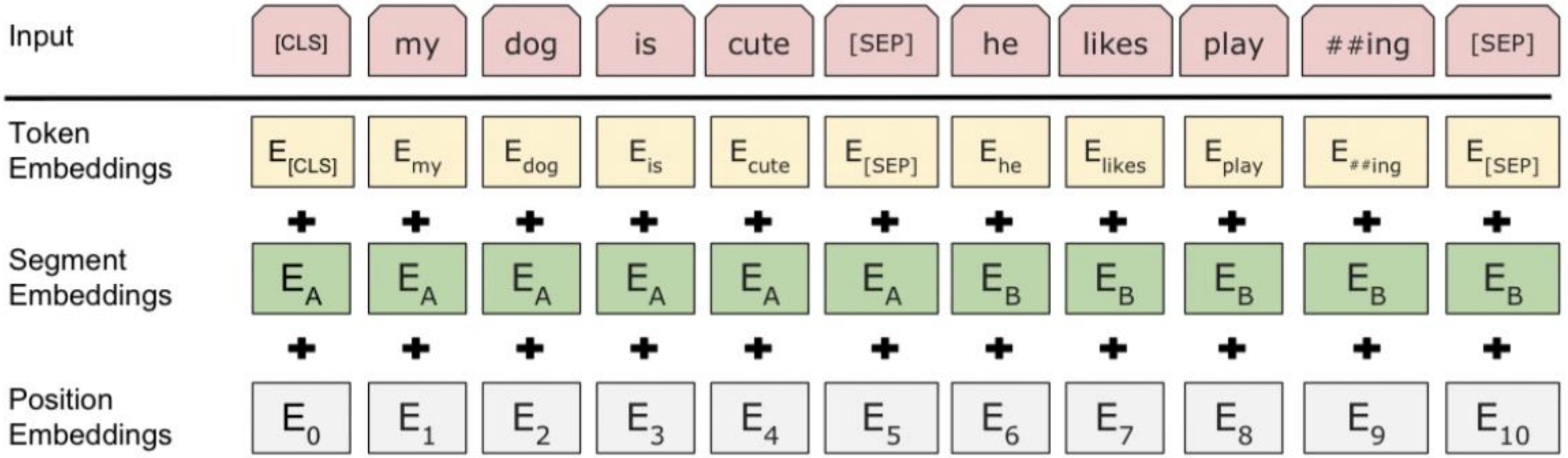
  Ex)

  공연은 끝났어 -> ['공연-' + '-은' + '끝-' + '-났어']
  공연을 끝냈어 -> ['공연-' + '-을' + '끝-' + '-냈어']
  개막을 해냈어 -> ['개막-' + '-을' + '해-' + '-냈어']

  개막공연을 끝냈어 -> ['개막-' + '공연-' + '-을' + '끝-' + '-냈어']

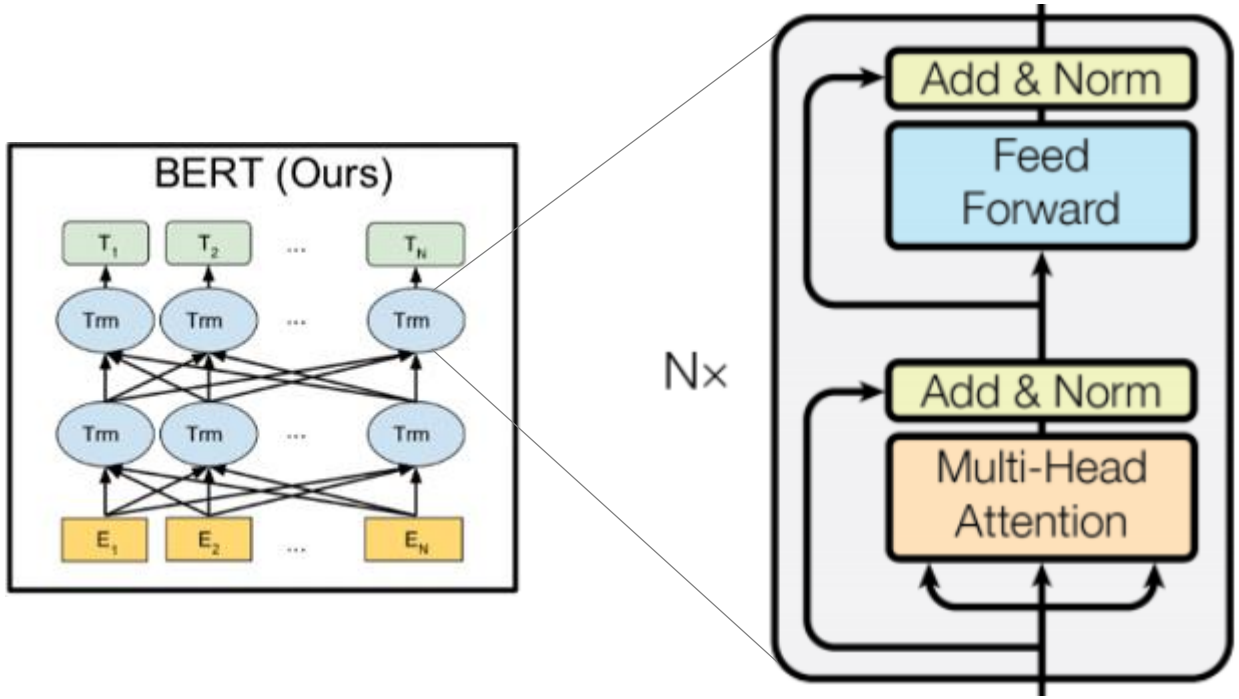
- 개막 공연이라는 독립된 유닛을 생성할 필요가 없음
- 즉 자주 이용되는 단어는 유닛으로 사용하고, 비 빈발 단어는 생성된 유닛의 복합체로써 이용한다

3. BERT : Input Embedding



- BERT는 Positional encoding 대신 Position Embedding으로 위치값을 부여
- pos는 각 토큰의 위치 정보. 위치에 따라 차례대로 값이 부여되는 range(0, max_len)
- seg는 토큰 타입. 입력 문장의 종류에 따라 각각 다른 값을 부여
- 두 문장을 [SEP]로 구분, 첫 번째 문장 위치에는 0, 두 번째 문장 위치에는 1을 부여
- For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings.

3. BERT : Encoder



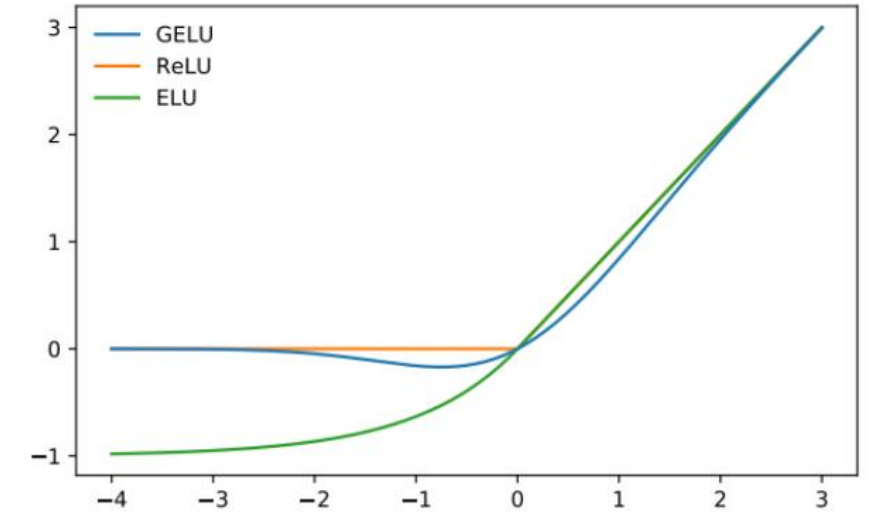


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

- Transfomer의 Encoder와 동일
- 차이점은 FFNN에서 Activation function을 RELU에서 GELU로 변경
- 음수에 대해서도 미분이 가능

3. BERT : Pre-train

- 01. Masked LM

- Intuitively, it is reasonable to believe that a deep bidirectional model is strictly more powerful than either a left-to-right model or the shallow concatenation of a left-toright and a right-to-left model.

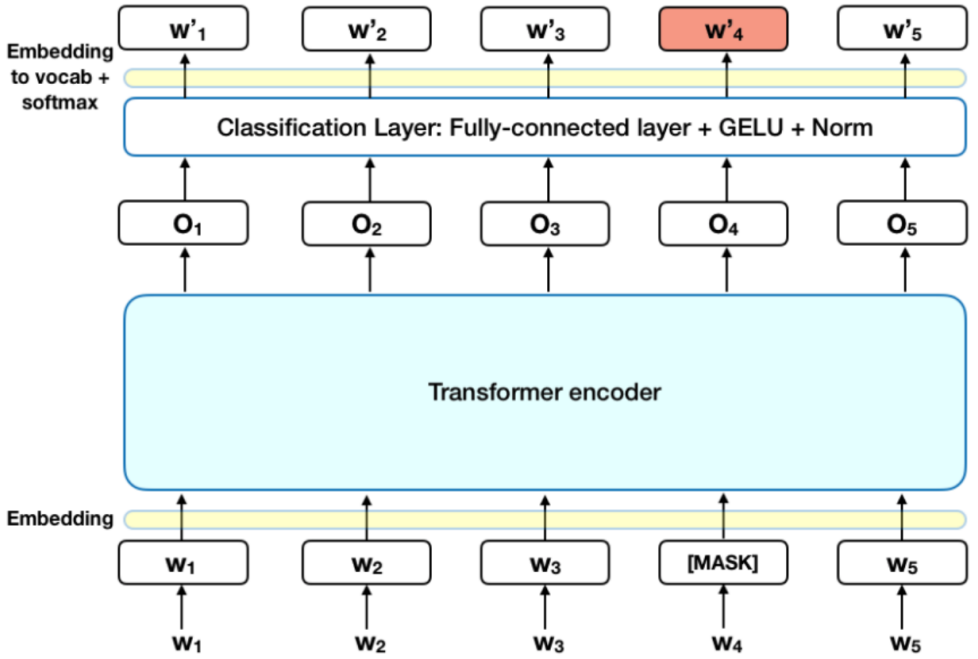
- Unfortunately, standard conditional language models can only be trained left-to-right or right-to-left

- In order to train a deep bidirectional representation, we simply mask some percentage of the input tokens at random, and then predict those masked tokens. We refer to this procedure as a “masked LM” (MLM),

- In this case, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM.

- In all of our experiments, we mask 15% of all WordPiece tokens in each sequence at random.

3. BERT : Pre-train



- Although this allows us to obtain a bidirectional pre-trained model, a downside is that we are creating a mismatch between pre-training and fine-tuning, since the [MASK] token does not appear during fine-tuning.
- The training data generator chooses 15% of the token positions at random for prediction. If the i-th token is chosen, we replace the i-th token with (1) the [MASK] token 80% of the time (2) a random token 10% of the time (3) the unchanged i-th token 10% of the time.
- Then, Ti will be used to predict the original token with cross entropy loss

3. BERT : Pre-train

- 02. Next Sentence Prediction(NSP)

- Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences, which is not directly captured by language modeling.

- In order to train a model that understands sentence relationships, choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext).

```
Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP] LABEL = IsNext

Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP] Label = NotNext
```

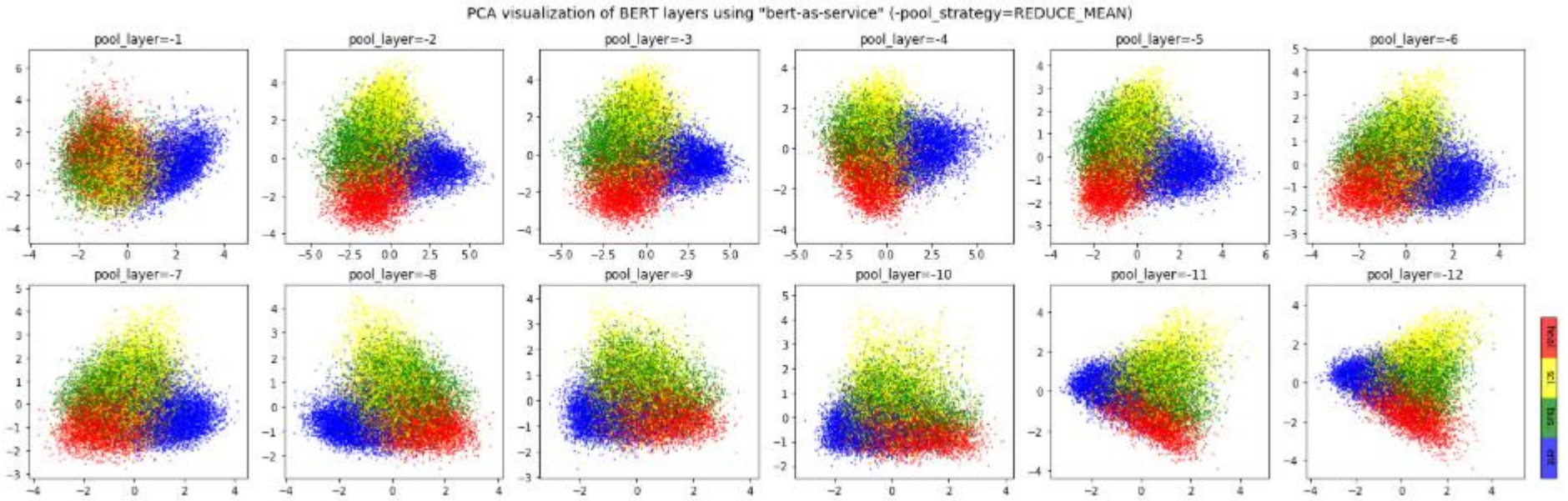
- As we show in Figure 1, [CLS] is used for next sentence prediction (NSP).

- [CLS] 벡터의 Binary Classification 결과를 맞추도록 학습한다

3. BERT : Fine-tuning

- For each task, we simply plug in the task specific inputs and outputs into BERT and fine tune all the parameters end-to-end.
- At the output, the token representations are fed into an output layer for token level tasks, such as sequence tagging or question answering, and the [CLS] representation is fed into an output layer for classification, such as entailment or sentiment analysis.
- Compared to pre-training, fine-tuning is relatively inexpensive.

3. BERT : For embedding

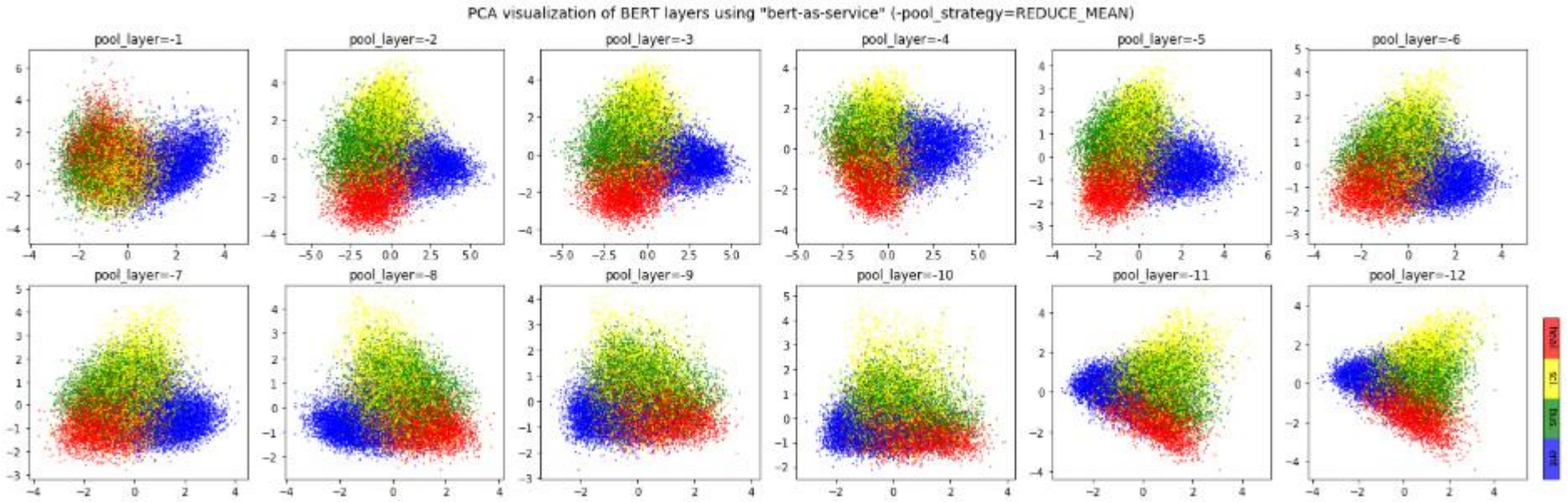


- BERT는 Fine-tuning 하지 않고도 ELMo처럼 fixed feature vectors를 추출하여 Pre-trained Contextual Embeddings를 활용 가능

- 총 12개의 encoder block이 있기 때문에 이를 문장 단위로 mean을 취하여 Sentence Representation으로 간주하는 방법이 있음

- 실제로 이 방식은 최근까지 여러 연구와 논문에서 주로 사용되는 Representation이지만 BERT 논문 저자는 이 방식에 부정적인 견해를 보임

3. BERT : For embedding



- BERT의 경우 Classification을 위한 [CLS] 벡터를 따로 갖고 있으며 분류 문제에서 어떤 레이어가 좋은 Representation인지 dimensionality reduction을 통해 시각화를 진행.
- BERT-Base 모델의 인코더 블럭은 12개이며 각각의 출력 결과를 차례대로 시각화
- pooling_layer=-1은 출력에 가장 가까운 레이어이고, 입력에 가장 가까운 레이어 pooling_layer=-12
- 분류에서는 이를 이용한다면 가장 좋은 성능을 얻을 수 있을 것으로 보인다.

Reference

- Attention
  - https://wikidocs.net/22893
  - https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/10/06/atention/
  - https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transfomer-self-attention-842498fd3225

- Transformer
  - https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transfomer-self-attention-842498fd3225
  - https://pozalabs.github.io/transformer/
  - https://wikidocs.net/31379

- Bert
  - http://docs.likejazz.com/bert/#scaled-dot-product-attention
  - https://tmaxai.github.io/post/BERT/
  - https://mino-park7.github.io/nlp/2018/12/12/bert-%EB%85%BC%EB%AC%B8%EC%A0%95%EB%A6%AC/?fbclid=IwAR3S-8iLWEVG6FGUVxoYdwQyA-zG0GpOUzVEsFBd0ARFg4eFXqCyGLznu7w
  - https://arxiv.org/pdf/1810.04805.pdf

THANK YOU