

## 4. transformer

NLP [ Multi-class Classification-based  
CRF based ]

NLP el transfer learning (기존학습) [ 1) Feature-based : 단어(토큰) 및 대량 품성 베이스를 이용한 학습 ]

특정 task를 위한 단어표현과 이미 학습한 단어표현을 합침으로써 학습하는 방식

2) Fine-tuning : Feature를 추출하는 신경망을 바꿔줌

특정 목적을 위한 parameter 조정을 적용해 방식의 일반적인 대상을 위한 신경망으로

Bert → fine tuning

특정 목적을 위한

[ open fine-tuning ]

: 디버깅하기가 편하다는 점.

: transformer 모델은 self-attention 벡터의 encoder와 decoder를 같은 토큰들이 다른 토큰들에 대한 영향을 반영.

→ Q/A문을 이해하는 힘增强

→ BERT는 이런 문제를 풀 때마다 유방향성 (bidirectional)으로 모델을 적용한다.

[ Bert ]

L : transformer의 block의 개수

[ Word Embedding OR ]

→ 단어를 벡터로 표현하는 것을 말함.

H : hidden layer의 개수

→ 단어를稀疏 표현 (sparse representation) or密集 dense representation으로 표현하는 것.

A : attention의 header 개수

[ transformer의 주요 구성 ]

· Seq2Seq 모델 (기존)의 확장

↳ 인코더-디코더의 구조를 확장하여 적용.

- 인코더 : 입력 sequence를 하나의 벡터 표현으로 압축

↳ 디코더 : 인코더를 통해 나온 벡터 표현을 통해 출력 sequence를 만들거나

→ 인코더에서 얻어진은 결과에서 sequence의 일부가 손실되는 경우 백정하기 위해 어려움이 발생!

( RNN의 농장을 위해 디코더를 사용 )

그럼, 아래 어텐션으로 인코더와 디코더를 만들면?

## [transformer gloss...]

### - transformer의 주요Terminology

**d<sub>model</sub>**: 모델의 디멘션을 정하는 입력과 출력의 크기를 의미.  $\rightarrow$  임베딩 벡터 크기 = d<sub>model</sub>

$\rightarrow$  각 인코더/디코더가 다음 풀의 인코더와 디코더로 분류되는 크기 = d<sub>model</sub>

**num-layers**: 몇개의 인코더-디코더를 한 층으로 생각할 때 transformer model이 총 몇 층으로 구성되었는지 의미.

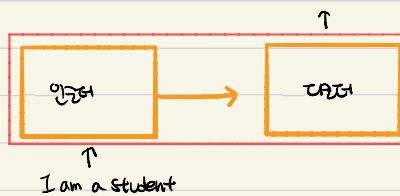
**num-heads**: transformer는 어텐션을 사용한다. 1번 하는 것보다 여러 개로 분리해서 여러 개의 어텐션을 수행하고 결과값을 더해나갈 때, 그때 그 수의 개수.

**dff**: transformer 네트워크는 포드 포트 신경망이 존재하는데, 이는 2개의 풀의 크기 = d<sub>ff</sub>

$\downarrow$  이 신경망의 입력 혹은 출력의 크기 = d<sub>model</sub>

### - transformer Seq2Seq

#### Seq2Seq (RNN)

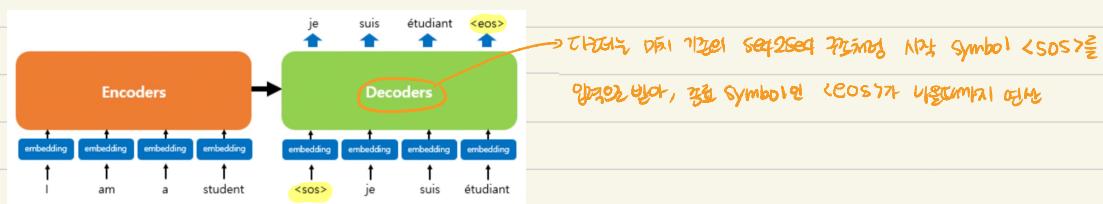


transformer model

$\rightarrow$  트랜스포머는 RNN을 N개 X

But 각각의 Seq2Seq에는 인코더-디코더 구조를 갖지.

$\rightarrow$  (설명): 인코더와 디코더라는 단위가 N개 존재하는 듯



$\rightarrow$  training 때는 encoder는 자기의 position에 따른, 즉 자기의 단어에 대한 attention을 계산해낸다.

decoder에서는 masking 기능을 이용해 병렬처리가 가능해진다.

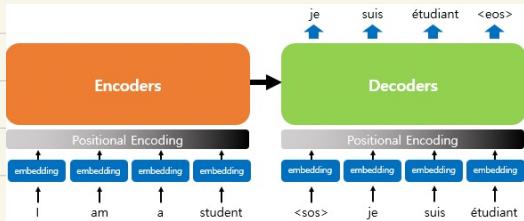
그무는말?

(transformer의 양쪽에 대문)

### Positional Encoding

: RNN을 사용했을 때 → 각 단어의 위치정보를 가질 수 O  $\Rightarrow$  transformer는 RNN 대체 X

: 각 단어의 위치 transformer는 단어의 양쪽에 벡터로 위치정보를 갖다 줄 때 양쪽의 위치의 차 = positional encoding



) 양쪽에 벡터들이 위치정보를 사용하기 위한 positional encoding  
길이 정해짐

### [Transformer의 양쪽에 위치정보]

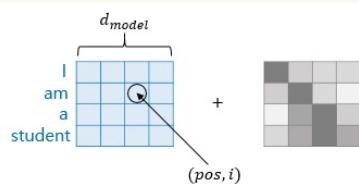
$$\begin{array}{cccc} \text{positional encoding:} & \text{---} & \text{---} & \text{---} \\ \text{embedding vector:} & \text{---} & \text{---} & \text{---} \\ \hline & + & + & + & + \\ & \text{I} & \text{am} & \text{a} & \text{student} \end{array} \rightarrow \text{Transformer는 위치정보를 가진값을 같은 인덱스에 두개씩 합산 사용}$$

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

→ sin과 cos함수의 값을 임베딩 벡터에 대해서도 같은 위치에 같은 값을 더해줌

→ 포지션 인덱스와 임베딩 벡터의 흐름은

포지션 인덱스와 임베딩 벡터가 같은 인덱스에 있을 때만 임베딩 벡터의 행렬의 해당 열을 통한 이어짐



$\rightarrow pos$  : 양쪽에 위치 정보의 위치  $\rightarrow i$ 가 짝수이면 sin함수

$\rightarrow i$  : 양쪽에 위치 정보의 위치의 인덱스  $\rightarrow i$ 가 홀수이면 cos함수

$\rightarrow d_{\text{model}}$  : 드롭아웃과 같은 충돌하는 흐름 차점을 위하여 드롭아웃 하이퍼파라미터

### 하이퍼파라미터란

: parameter  $\neq$  hyper parameter

#### 1) parameter

: 모델 변수, 예상 내보내는 변수

: 데이터셋 내보내는 변수

: 예상치와 실제 값의 차

#### 2) hyper parameter

: modeling이나 예상 내보내는 값

: 하이퍼파라미터

transformer: recurrence(순환)을 사용하지 않고 대신 attention mechanism만을 사용해 input/output dependency를 표현함.

## [Transformer의 특징]

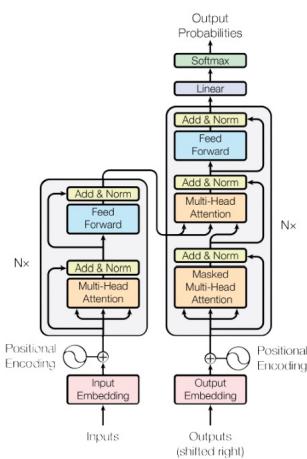


Figure 1: The Transformer - model architecture.

→ Attention 기능을 이용해 다양한 대입을 전부 수행한다.

→ 멀티어 대입을 위한 브로드온을 크게 늘려

[스포크언어]: 어떤 대입이 충돌해 다른 대입의 영향 누적

- 어텐션: 같은 Seq와 Seq와 같이 같은 대입의 결과에 다른 어텐션 영향수락

피드포워드 계층: 어텐션 계층을 거쳐 같은 결과물을 추출적으로 정리.

→ 인코더

: 각각의 스포크언어별 복잡한 피드포워드 계층으로 이루어져있음.

→ 디코더

: 다음의 스포크언어보다 일반언어언어가 번갈아가며 나타나고, 피드포워드계층이 나타나는 대입.

- **인코더**
- 토큰포머는 기본적으로 **multi-layer** 인코더를 가짐
- 2개의 sublayer
  - 1) Multi-head Self-Attention
  - 2) Feed Forward Network

### 1) Multi-head Self-Attention

#### 1-1) Self-Attention

: 어텐션을 각각 차례로 수행하는 것.

: 각각 Attention의 경우  $\rightarrow Q, K, V$  존재  $Q: \text{Query} = \text{단어의 단어 벡터 묶임의 원래 상태}$

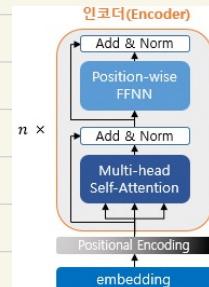
$Q$ 와  $K$ 가 대입,  $V$ 는  $Q$ 와 같은 단어 벡터를

$K: \text{key} = \text{단어 묶임의 단어 벡터 묶임의 원래 상태}$

$V: \text{values} = \text{단어 묶임의 단어 벡터 묶임의 원래 상태}$

: Self-Attention의 경우  $\rightarrow Q, K, V$ 가 모두 같은 값.

$Q, K, V = \text{입력단어의 모든 단어 벡터들}$

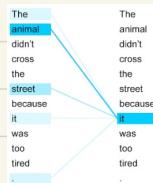


→ 각 계층 별 대입은 서로 다른 영향으로, 같은 대입은 다른

O

### (1-1) Q&A)

#### : Self-Attention의 효과



→ Self-Attention은 임베딩내의 단어들끼리 유사도를 구함으로 "it"이 "animal"과 유사성이 높다는 것을 찾아냄  
 ↳ 같은 Sequence는 앞단어의 문장, 다음단어의 문장 → 이 시퀀스는 두 문장의 연관성을 찾아냄 때문이겠죠 × 정보!

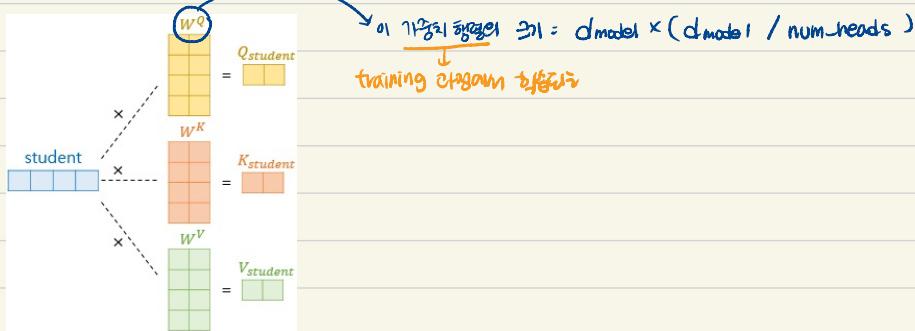
### (1-2) Q, K, V 벡터의 역할

- : Self-Attention은 임베딩의 초기 입력인  $d_{model}$  차원을 가지는 단어ベクトル들을 사용하여 Self-Attention 수행 X
- : 우선, 각 단어를 블록 ① 벡터, ② 벡터, ③ 벡터를 얻는 작업을 거침.

↳ 초기입력인  $d_{model}$  차원보다 더 작은 차원

↳ num\_heads로 인해 가능해짐.

↳  $d_{model}$ 을 num\_heads로 나눈 값이 각 Q, K, V 벡터의 차원이 됨 !

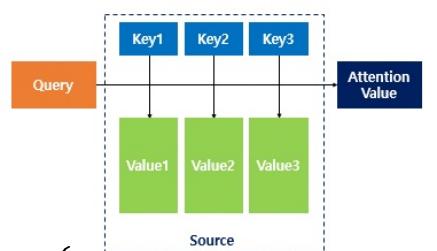


### (1-3) scaled dot-product Attention

- : 각 Q 벡터는 모든 K 벡터에 대한 Attention Score를 구하고 Attention distribution를 계산. 이를 이용하여 모든 V 벡터를

활용함하여 Attention &/ Context vector를 계산

↳ Attention Score를 구함, 텐서를 더한 후 풀기



↳ dot product

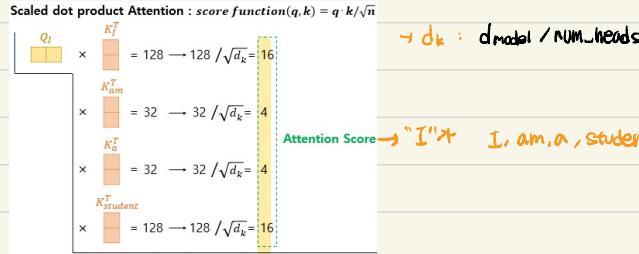
$$\text{Score}(q, k) = q \cdot k / \sqrt{n}$$

↳  $n$ 은 풀기되는 벡터의 차원

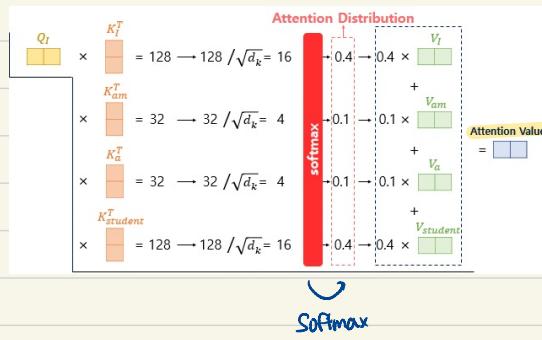
↳ 이런 방식을 쓰는 어텐션을 "scaled dot-product Attention"이라고 함!

## (scaled dot-product Attention)

"I" on air



$\rightarrow d_k : \text{d}_{\text{model}} / \text{num\_heads}$



→ I가 am, a, student 각각에 대한 확률은?

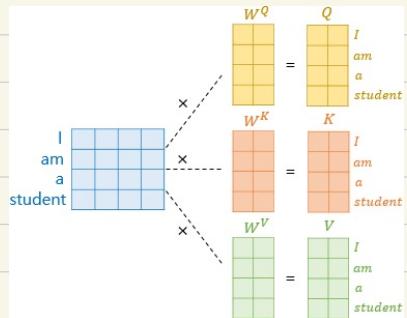
Q. 글이 다 뜨고 해야겠다?

A. 벡터 연산으로 알면 된다

## 1-4) 행렬 연산으로 알았을 때

- ④ 각 단어에 대해  $Q, K, V$  벡터를 구해 Scaled dot-product attention 계산 → 벡터 연산

→ 벡터 연산이 아니라 행렬 연산으로 하면 동시에 계산 가능!



→ 각 단어 벡터마다 일일히 가중치 행렬곱 X

→ 문장행렬이 가중치 행렬을 포함해  $Q, K, V$  행렬구성!

(1-4 Q101M)

$$\begin{matrix} Q \\ I \\ am \\ a \\ student \end{matrix} \times K^T = \begin{matrix} I \\ am \\ a \\ student \end{matrix} \rightarrow \text{Attention score distribution}$$

일단,  $Q$ 행렬  $\times K^T$  행렬 하면  
각 단어의  $Q$ 행렬과  $K^T$ 행렬의 내적과 각 행렬의 유사도가 되는 행렬

보이  $Q \cdot K / \sqrt{d_k}$  일정으로  $\rightarrow$  이 행렬의 값이 일정으로  $\sqrt{d_k}$ 를 나누어주면 각 행렬의 Attention score를 가지게 됨.

$$\begin{matrix} I \\ am \\ a \\ student \end{matrix} \times \begin{matrix} I \\ am \\ a \\ student \end{matrix} \rightarrow I \text{의 } Q \text{행렬对学生 } K^T \text{행렬 Attention score}$$

$\rightarrow$  Attention score 행렬

$\rightarrow$  open Attention distribution을 구하고, 이를 이용해 모든 단어에 대한 Attention 값을 구해야함.

$\hookrightarrow$  원래 행렬의 가로를 곱하고 대각선 부분으로 구해요.

$$\text{softmax} \left( \frac{Q \cdot K^T}{\sqrt{d_k}} \right) \times V = \text{Attention Value Matrix } a$$

Attention distribution

$V$ 는 원래행의 행렬이므로 곱하면 Attention Value Matrix가 나온다.

하지만 보통은  $V$ 행렬에 가로로 수식화 일지!

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

: 입력행렬의 크기가 seq\_len이면 문장행렬의 크기: (seq\_len, d\_model)  $\rightarrow$  예제: 가공자 행렬을 끝나면  $Q, K, V$

행렬을 차운다!

$Q$ 행렬의 크기를  $d_k$ ,  $V$ 행렬의 크기,  $d_v \Rightarrow Q/K$ 행렬의 크기: (seq\_len,  $d_k$ ),  $V$ 행렬의 크기: (seq\_len,  $d_v$ )

그럼 가공자 행렬의 크기는!  $W^Q, W^K$ 의 크기: ( $d_{model}, d_k$ ),  $W^V$ 행렬의 크기: ( $d_{model}, d_v$ )

근데!  $d_k = d_v = d_{model} / \text{num\_heads}$  이다.

결과적으로 이 수를 적용하여 나온 attention 각 행렬의 크기는 (seq\_len,  $d_v$ )

1-5) 구조와 용법...

1-6) 멀티 헤드 어텐션

- 초기화된 attention : d<sub>model</sub> 차원의 벡터를 num\_heads로 나눈 차원을 가지는 Q, K, V 벡터를 바탕으로, 어텐션 수행

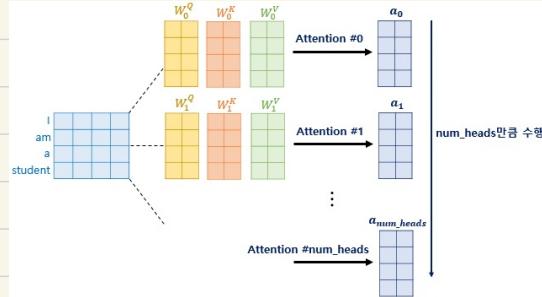
Q. num\_heads의 의미는? 각 d<sub>model</sub> 차원의 단일 벡터를 최소화시켜 어텐션을 수행하는지?

A. 단일의 어텐션보다 예전에는 더 효율적이라 평가!

그럼, d<sub>model</sub>의 차원을 num\_heads 개수 나눈 d<sub>model</sub> / num\_heads 차원을 가지는 Q, K, V 차원에 대해 num\_heads 개의

병렬 어텐션을 수행! → 병렬에는 num\_heads를 8로 지정, 8개의 병렬 어텐션 → 각각의 어텐션 값 합계 = 어텐션 헤드

→ 가중치 행렬  $W^Q, W^K, W^V$ 은 어텐션 헤드마다 다 다르다  $\Rightarrow$  8개의 어텐션에 병렬로 이루어짐.



→ 228 병렬 어텐션으로 알 수 있는 효과는?

= 어텐션을 병렬로 수행하여 다른 시각으로 정보들을 수집했습니다!

Ex) 그 동물은 길을 건너지 않았다. 왜냐하면 그것은 너무 피곤하였기 때문이다

→ 여기서 그것(it)이 주제라면 → 즉, '나에 대한 Q 벡터는 다른 단어의 연결성을 구현합니다'

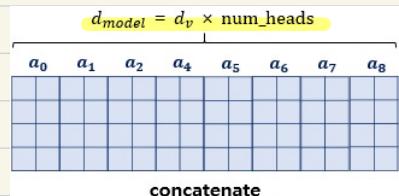
첫번째 어텐션 헤드: 그것(it)과 동물(animal)의 연결성을 높여줍니다

두번째 "": 그것(it)과 피곤한것이 때문입니다(tired)를 연결해 높여줍니다! 보고하기 때문에!

각 어텐션 헤드는 전부 다른 시각으로

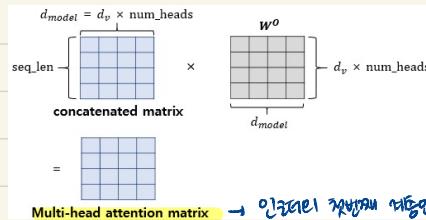
→ 병렬 어텐션을 모두 수행했으면 모든 어텐션 헤드를 연결(concatenate)

: 모든 어텐션 헤드 흡수의 크기는 (Seq\_len, d<sub>model</sub>)



→ 모델은 행렬을 모두 단일한 행렬을 또 다른 가중치  $W^o$  와 곱해짐.

⇒ 그 결과 행렬이 멀티 어텐션의 최종 결과다.



→ 인코더 내부에 적용한 Multi-head Attention 단계  $\Rightarrow$  인코더의 입력으로 들어온 행렬의 크기 유지!

인코더, 디코더에만 적용되는 가지 않고 있는 Sub-layer.

## 2) Position - wise FFNN

평지적 - 터미스 피드 포워드 신경망

Fully-connected FFNN

$$\text{FFNN}(x) = \text{MAX}(0, xW_1 + b_1)W_2 + b_2$$

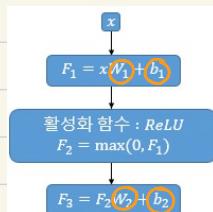
그림으로

$x$ : 각각 multi-head attention의 결과를 나온  
(Seq\_len,  $d_{model}$ )의 크기를 가지는 행렬

$W_1$ : ( $d_{model}$ ,  $d_{ff}$ )의 크기

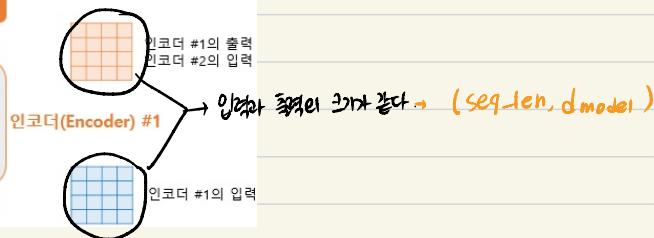
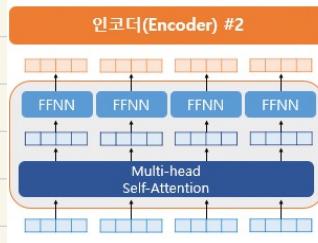
$W_2$ : ( $d_{ff}$ ,  $d_{model}$ )의 크기

은닉층의 크기



이제까지  $W_1, b_1, W_2, b_2$

한 인코더내부에는 같은 문장/단어마다 다른 값이!  
but, 인코더 층끼리는 다른 값을 가짐!



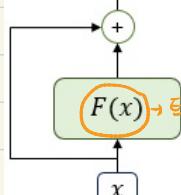


## • 잔차연결 (Residual connection)과 층 정규화 (Layer Normalization)

### 1) 잔차연결

: 이전 계층이나 위치  $H(x)$ 를 더하는 방식입니다!

$$H(x) = x + F(x)$$



.. 잔차연결 = 시브층의 입력 + 출력

( 입력과 출력은 차원이 같으면 가능! )

트랜스포머에서는 시브층에 적용됩니다.

$$\text{S} = x + \text{Sublayer}(x)$$

### 2) 층 정규화 (Layer Normalization)

- LayerNorm ( $x + \text{Sublayer}(x)$ )