D&A
Deep Session 9차시

Attention, Tramsformer, BERT

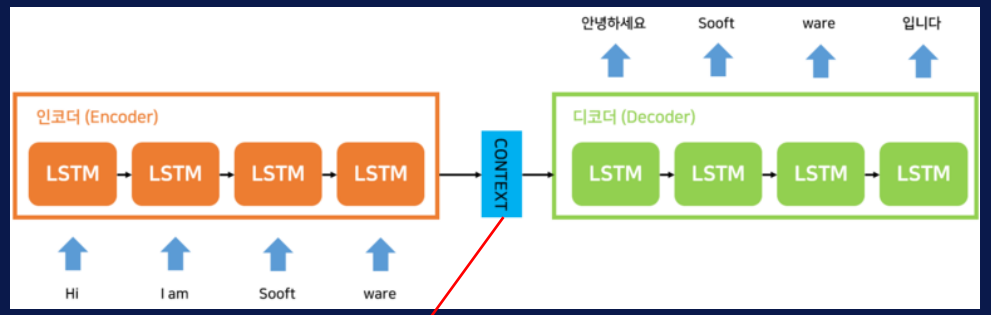
2022 / 05 / 19
D&A 운영진 이경욱

CONTENTS.

01 – Attention

01. Attention

Seq2seq 모델의 문제점



결국 마지막 RNN셀의 hidden state

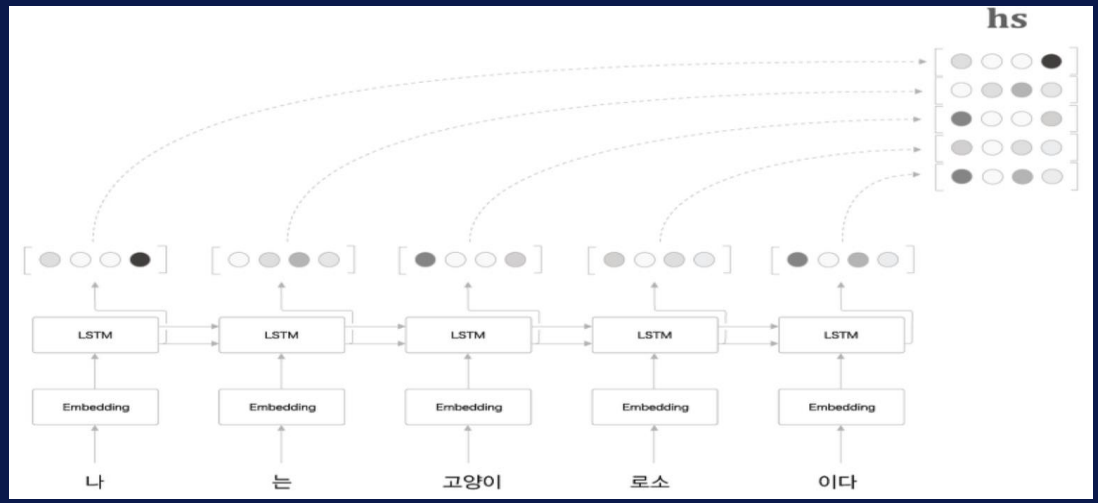
➡ Input의 길이에 상관없이 decoder에 고정된 길이의 벡터를 전달함

➡ Input의 정보가 decoder에 효과적으로 전달될 수 없음

Attention의 핵심 아이디어

➡ Decoder의 모든 예측시점마다 예측해야 할 단어와 연관이 있는 input 부분을 좀 더 집중(attention)해서 보자

➡ 일단 Encoder의 모든 hidden state를 decoder로 보내고, 거기서 중요한거 뽑아서 쓰자!!



➡ Encoder의 모든 hidden state 벡터를 행렬로 쌓아 Decoder의 모든 예측시점에 전달

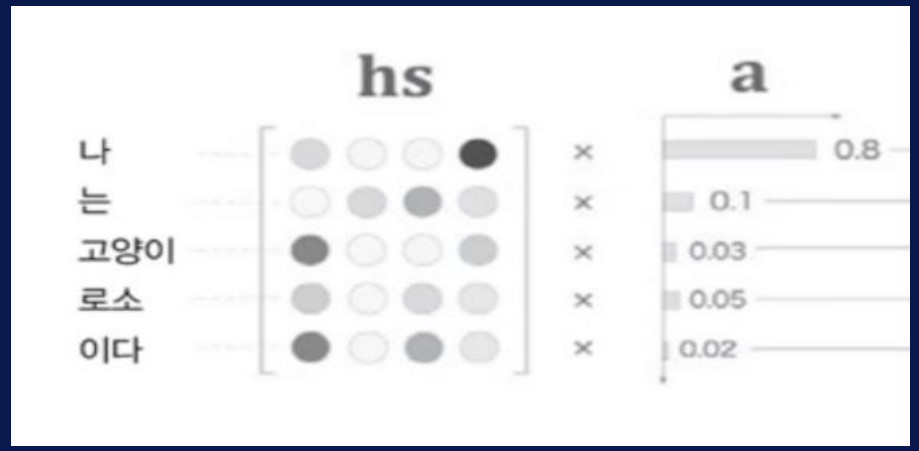
01. Attention

Attention – hs행렬에서 중요한 정보를 어떻게 뽑을건데??

1) Hs의 각 벡터들 중 중요한 벡터를 "선택"할 경우

➡ 단순 선택하는 작업은 미분이 불가능해 오차역전파가 불가능함

2) 미분이 가능한 연산으로 중요한 정보를 뽑아오기

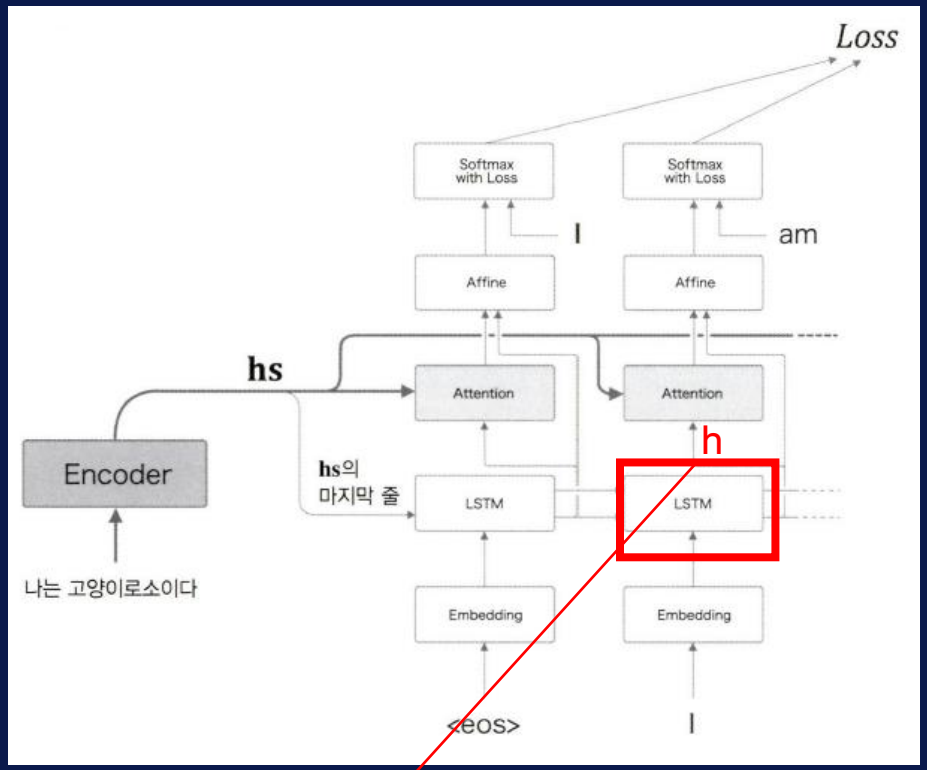


➡ Hs 내 각 벡터의 중요도(a)를 계산하여
가중치 합을 구하면됨

Attention – 중요도 계산하기

Hs에서 각 벡터들의 중요도가 의미하는 바는?

➡ Decoder 각 예측시점의 hidden state값과 hs의 벡터들이 얼마나 유사한가?

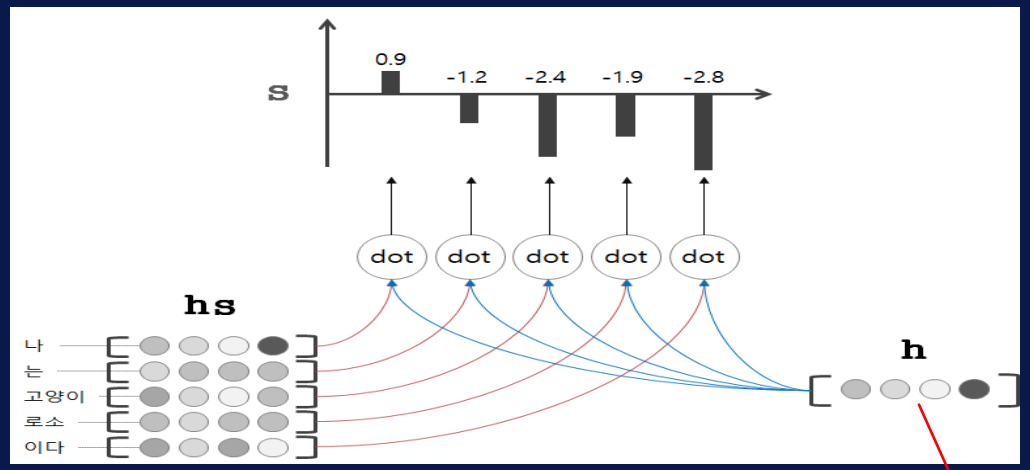


Decoder 특정 시점의 hidden state
-> 이게 hs의 각 벡터들과 얼마나 유사한가?
-> 이것이 중요도

01. Attention

Attention – 중요도 계산하기

1) Hs각 벡터들과 decoder의 hidden state의 유사도 구하기
2) 각 유사도를 중요도(가중치)로 바꾸기 위해 소프트맥스 함수로 정규화



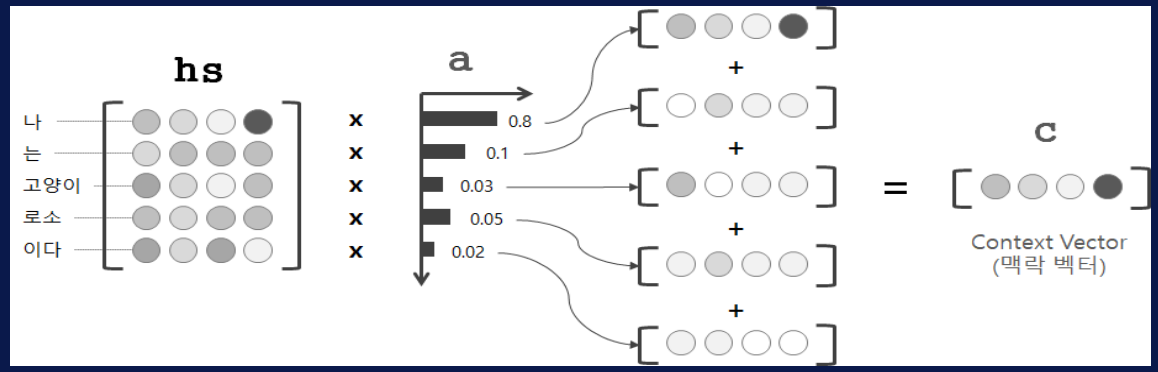
Decoder 예측시점의 hidden state

➡ Hs 각 벡터들과 h 벡터 간 유사도를 구함

➡ 각 값을 소프트맥스함수에 적용해 0~1사이의 값으로 만듦

attention 중요도를 구하는 방법은 여러 방법이 존재

Attention - 맥락벡터c

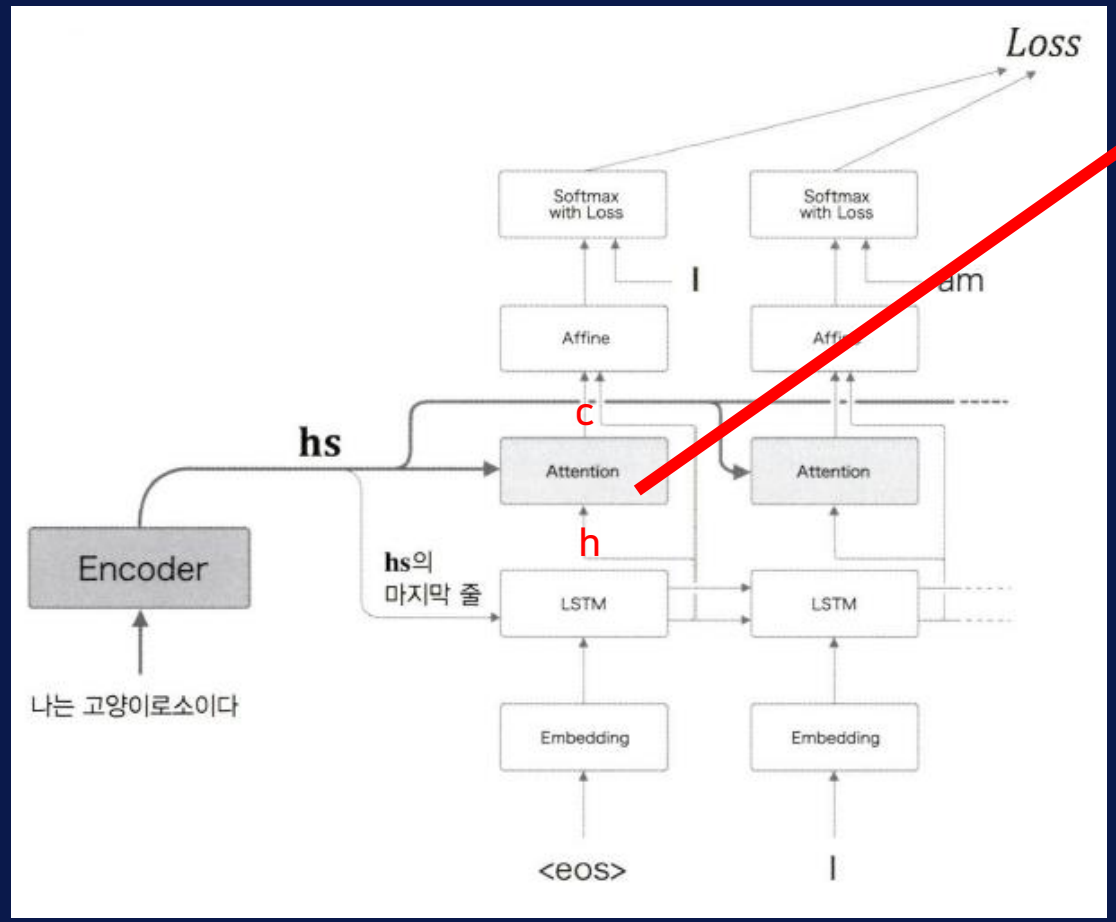


➡ 위 과정이 decoder의 각 시점마다 일어나고 있음

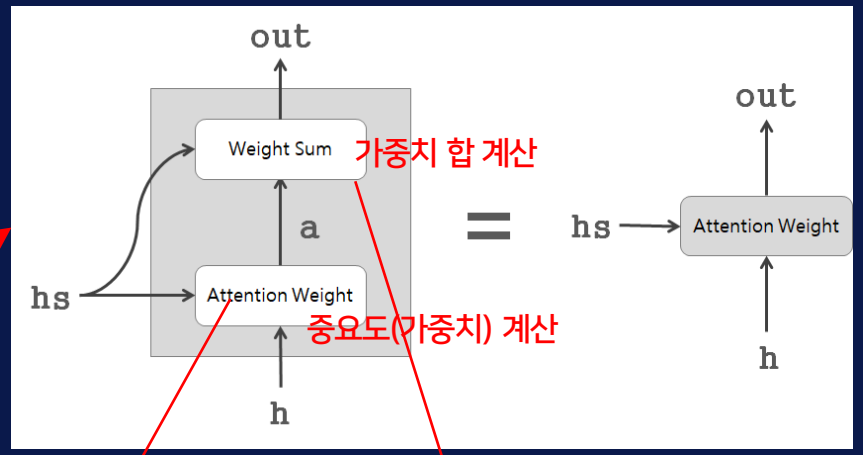
➡ Decoder특정 시점에 만들어진 중요도a를 가중치로 삼아 hs의 가중치 합을 계산 -> 해당 시점의 맥락벡터 c

01. Attention - 전체 맥락

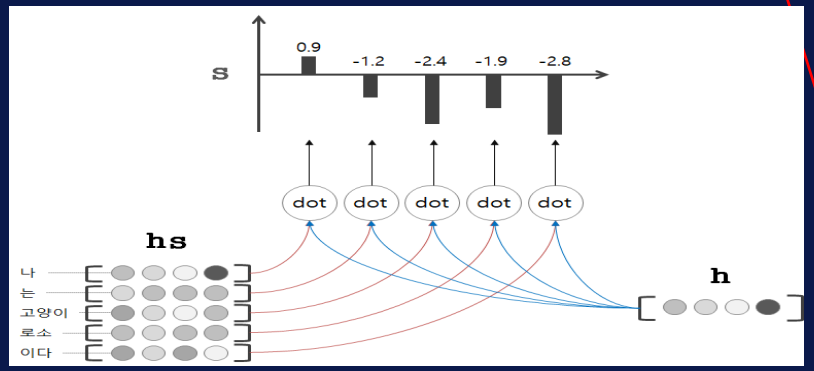
Decoder 전체



Attention 적용



중요도 구하기



맥락벡터 구하기

01. Attention

Attention에서 중요도(attention socre)를 계산하는 방법들

이름	스코어 함수	Defined by
dot	$score(s_t,\ h_i) = s_t^T h_i$	Luong et al. (2015)
$scaled\ dot$	$$score(s_t,\ h_i) = \frac{s_t^T h_i}{\sqrt{n}}$$	Vaswani et al. (2017)
$general$	$score(s_t,\ h_i) = s_t^T W_a h_i$ // 단, W_a는 학습 가능한 가중치 행렬	Luong et al. (2015)
$concat$	$score(s_t,\ h_i) = W_a^T\ tanh(W_b[s_t; h_i])score(s_t,\ h_i) = W_a^T\ tanh(W_b s_t + W_c h_i)$	Bahdanau et al. (2015)
$location - base$	$\alpha_t = softmax(W_a s_t)$ // α_t 산출 시에 s_t만 사용하는 방법.	Luong et al. (2015)

02 – Transformer

02. Transformer – attention is all you need

기존 RNN계열 모델의 문제점

1) input의 정보를 고정된 길이의 벡터로 decoder에 전달해 input 시퀀스의 정보가 효과적으로 전달되지 않음 (마지막 RNN셀의 hidden state만을 전달)

➡ Attention의 출현으로 보정

Attention 메커니즘만을 사용해서 번역 모델을 만들면 어떨까????

"Transformer"

2) Attention 또한 RNN 모델에 의존적임(병렬화를 배제)

3) Attention 메커니즘은 모델이 아니라 RNN을 보정하는 용도임

02. Transformer – attention is all you need

Transformer 구조 전체와 self-attention 개념

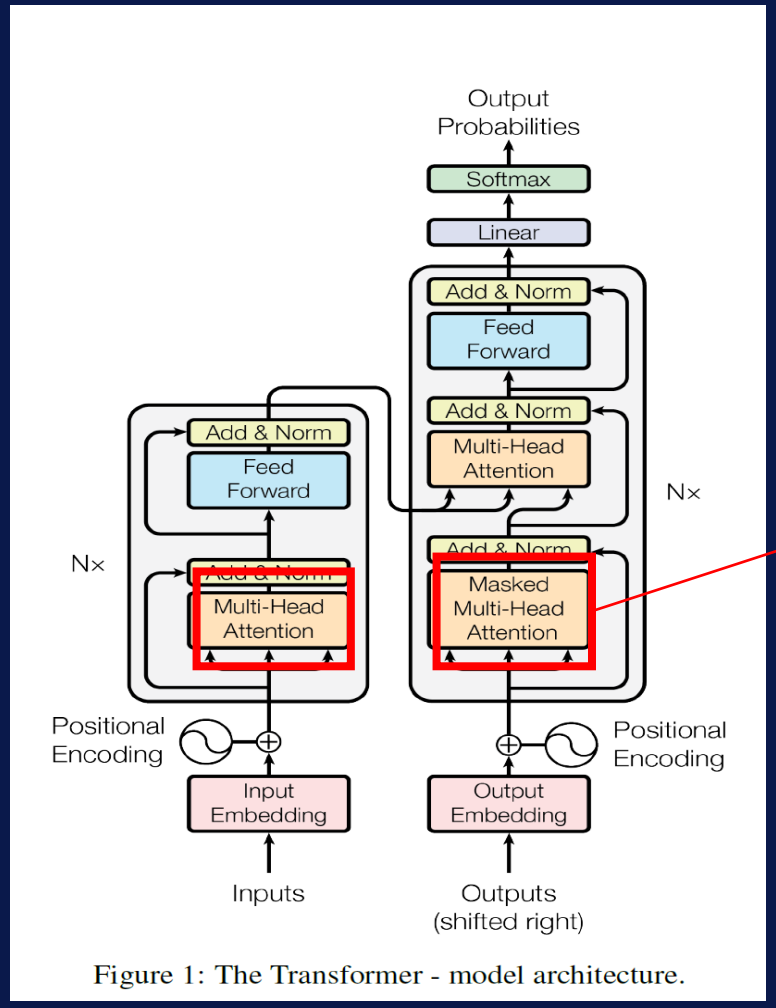


Figure 1: The Transformer - model architecture.

➡ 일단 RNN계층이 안 보인다!

➡ Attention이란 단어는 보이는데 계층 이름이 다르네?

주목할점

Input과 output의 교류가 없었는데 attention이 적용되었다

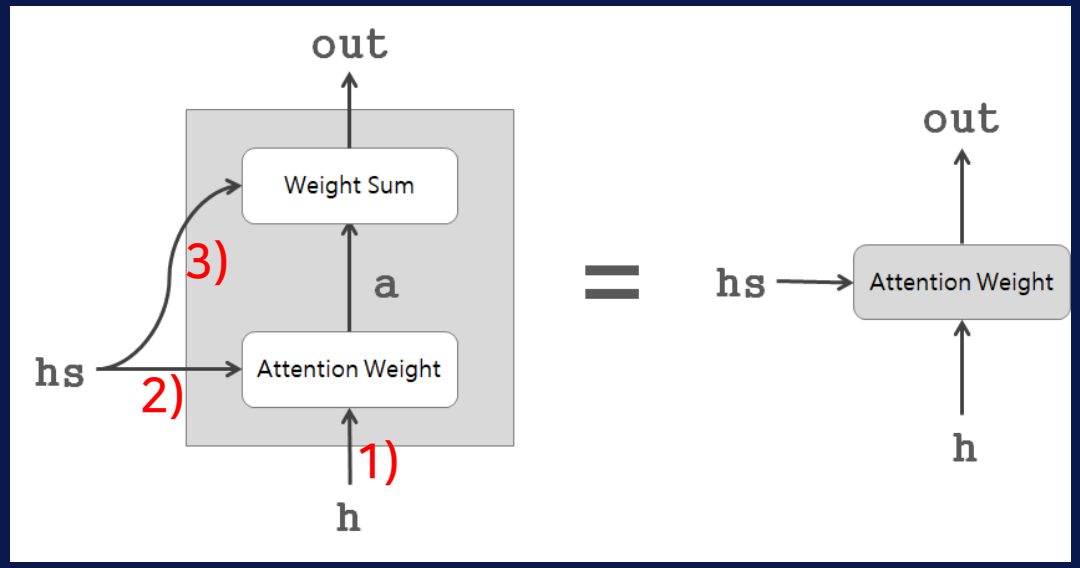
➡ Encoder와 decoder 각자의 input만을 사용해 attention을 적용한다?

Self-attention

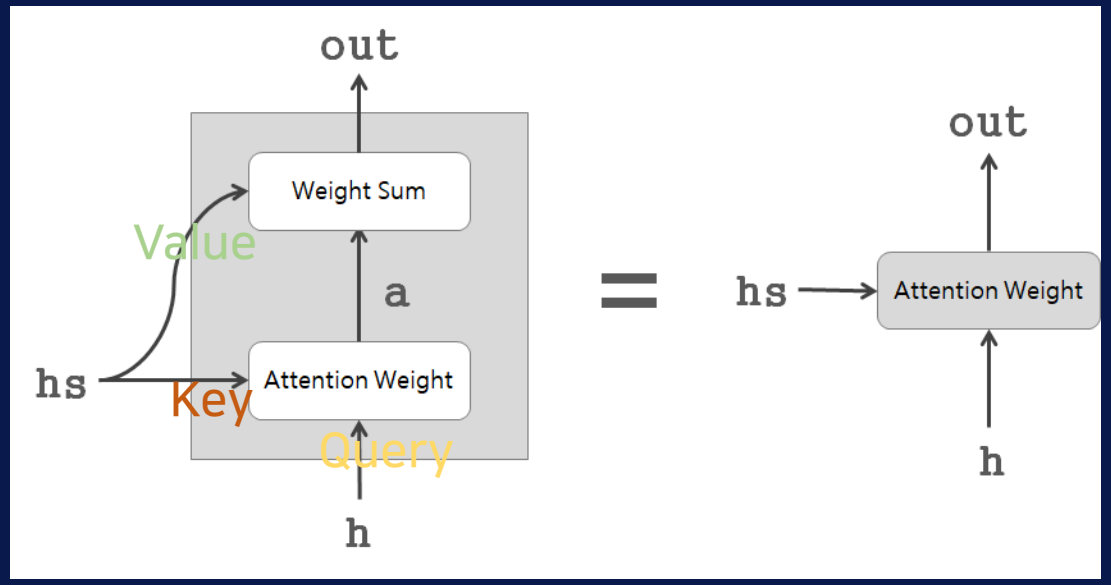
02. Transformer – attention is all you need

Self-attention – 기존 attention계층 다시보기

기존 attention에서 사용된 세 가지 값



여기에 그냥 이름을 지어보자!



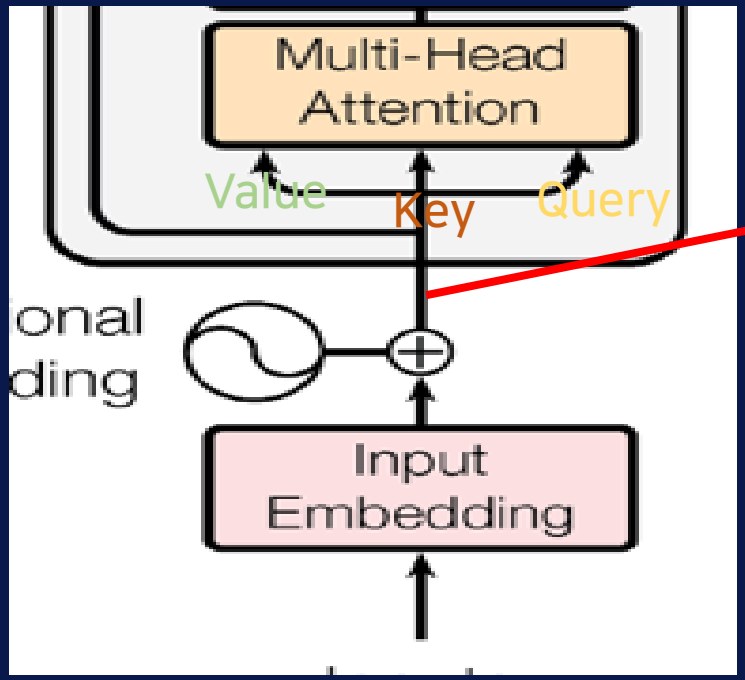
1) : 특정 시점의 decoder의 hidden state(h)
2) : 모든 시점의 encoder의 hidden state(hs)
3) : 모든 시점의 encoder의 hidden state(hs)

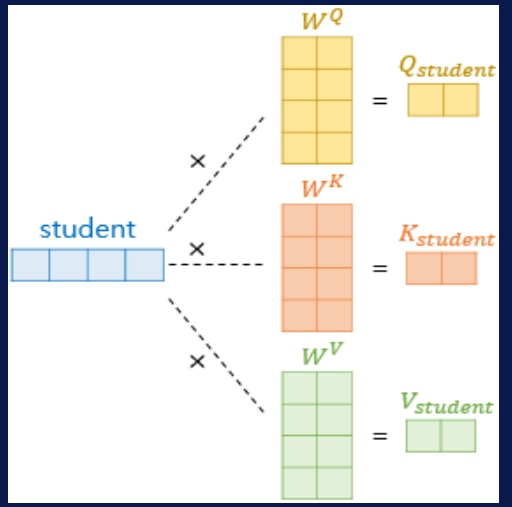
Query : 특정 시점의 decoder의 hidden state(h)
Key : 모든 시점의 encdoer의 hidden state(hs)
Value : 모든 시점의 encoder의 hidden state(hs)

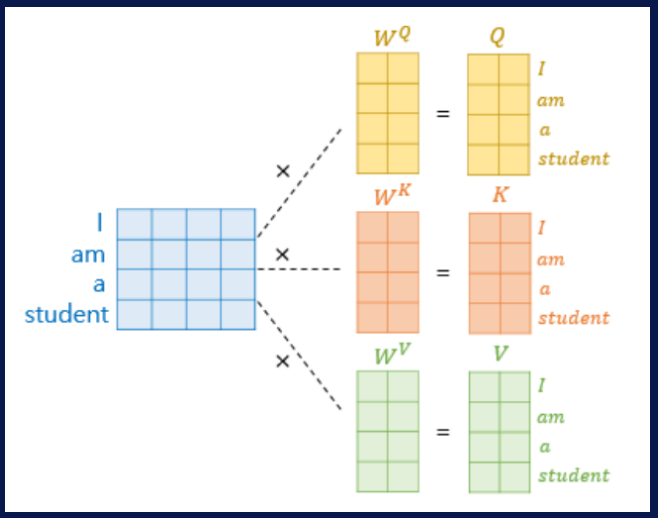
02. Transformer – attention is all you need

Self-attention – 같은 출처의 벡터들로 attention을 적용하기 위한 값 생성하기(feat. query, key, value)

Transformer의 self-attention에서 사용된 세 가지 값







Query : 입력 벡터의 특정 가중치 합의 결과
Key : 입력 벡터의 특정 가중치 합의 결과
Value : 입력 벡터의 특정 가중치 합의 결과

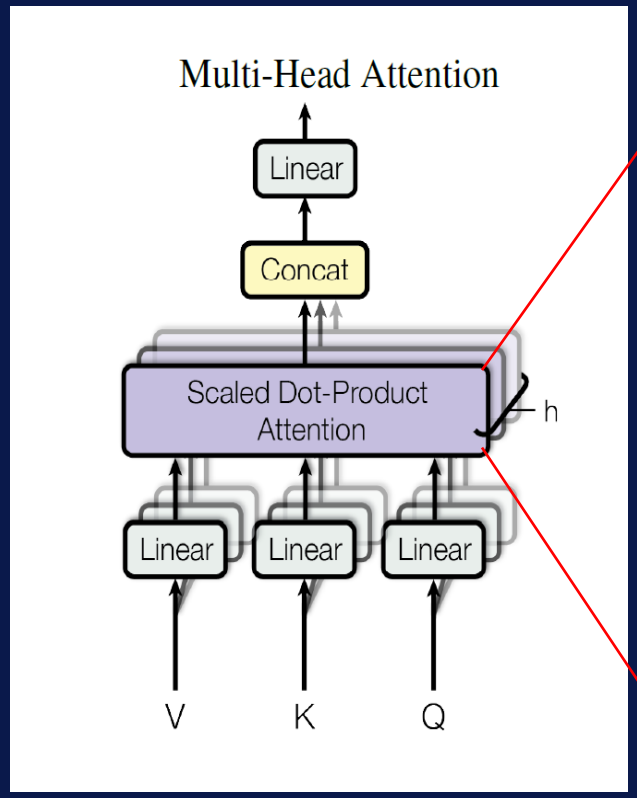
➡ Transformer에서는 같은 입력 단어에 다른 세 가중치를 적용해 세 값을 만듬

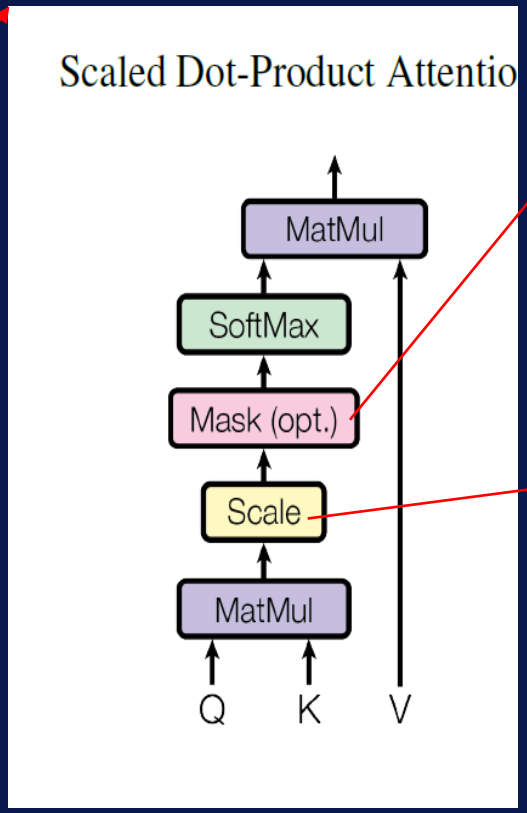
➡ 실제로는 행렬연산으로 일괄처리(오른쪽 그림)

➡ Attention에 적용되는 세 값의 출처가 같음
-> self-attention

02. Transformer

기본적인 Multi-head attention의 Scaled Dot-product





기본 Multi-head attention에는 적용되지 않음(밑의 수식에 없음)

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

d_k ➡ Key값의 차원수

02. Transformer

기본적인 Multi-head attention

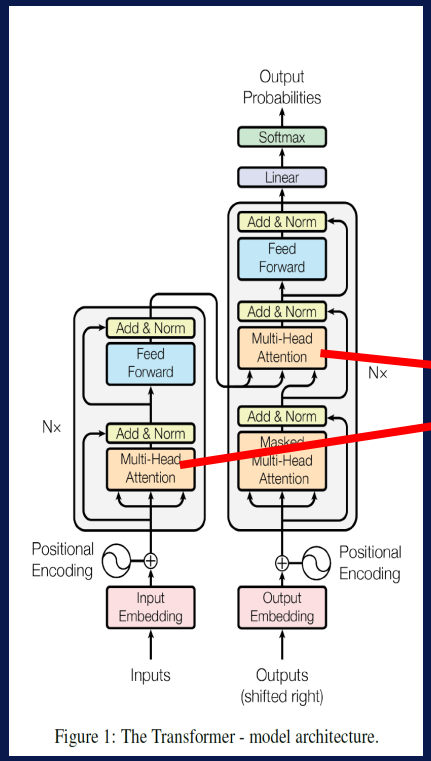
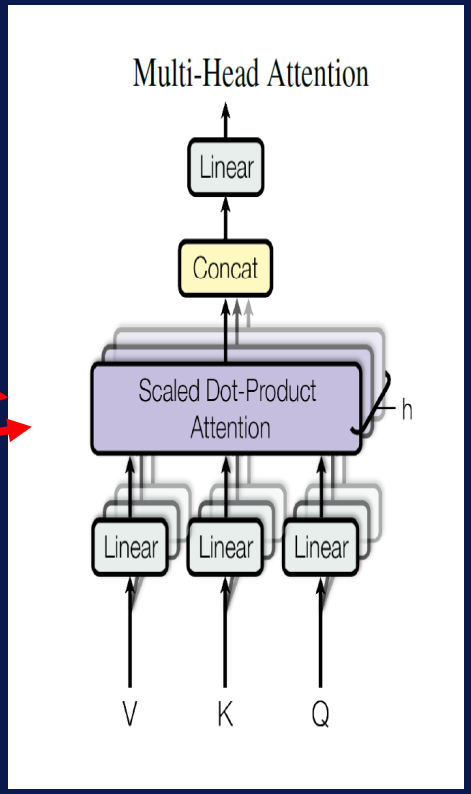


Figure 1: The Transformer - model architecture.



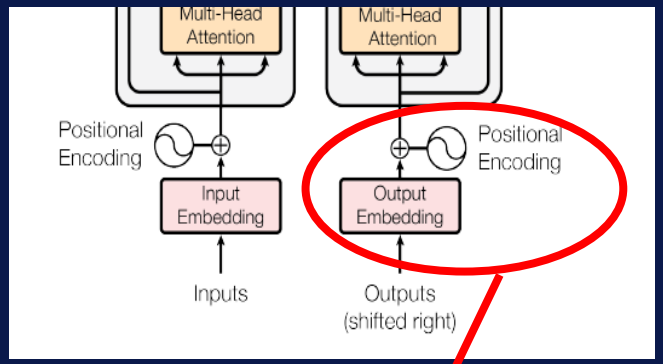
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

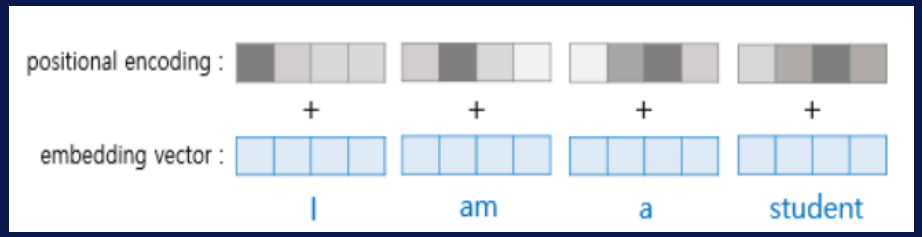
1) (입력값 차원 수 / mulithead 개수)의 차원으로 Q, K, V 생성
2) 각 Q, K, V를 h개의 버전으로 선형 변환
3) Q, K, V를 사용해 여러 버전의 Scaled Dot-Product Attention 진행
4) 각 버전을 concat
5) Concat된 값을 한 번 더 선형 변환

02. Transformer

Positional Encoding

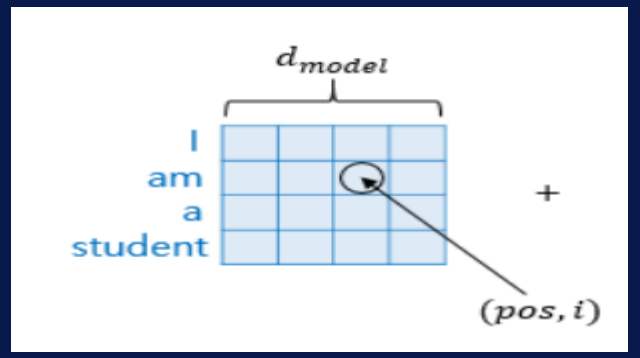
RNN계열의 장점 -> 순차적으로 단어를 입력 받아 단어의 위치 정보를 가질 수 있다.
Transformer -> 단어를 순차적으로 입력 받는 방식이 아니라 위치 정보를 알려줄 필요가 있음





➡ 임베딩 벡터에 positional encoding을 구해 더해줌

Positional Encoding 값 구하기



d_model : 임베딩 계층의 출력 차원
Pos : 입력 문장에서 임베딩 벡터의 위치
I : 임베딩 계층 차원 인덱스

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

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

Sin, cos 함수를 통해
순서정보를 계산해냄

02. Transformer - Feed Forward와 Add&Norm, 잔차 연결

Feed Forward, Add&Norm

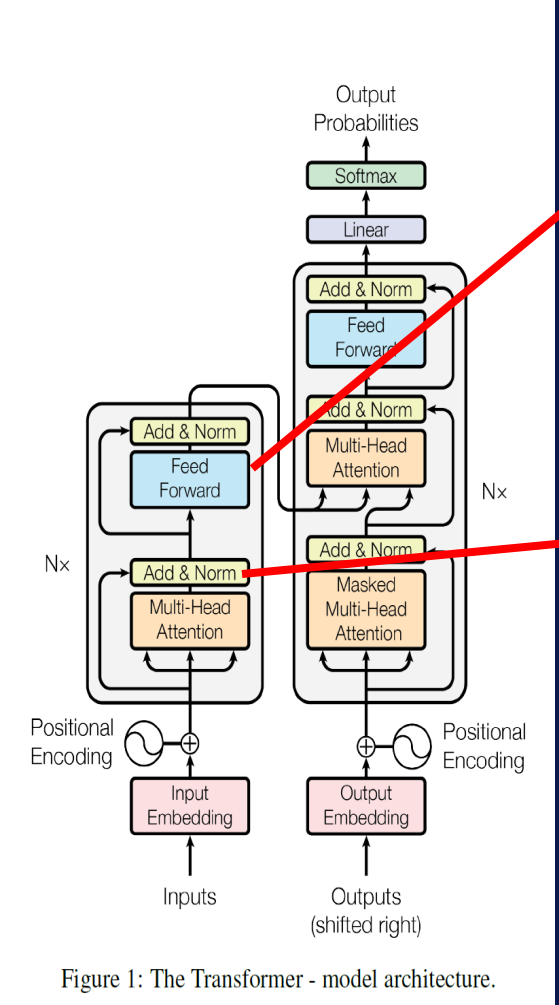


Figure 1: The Transformer - model architecture.

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

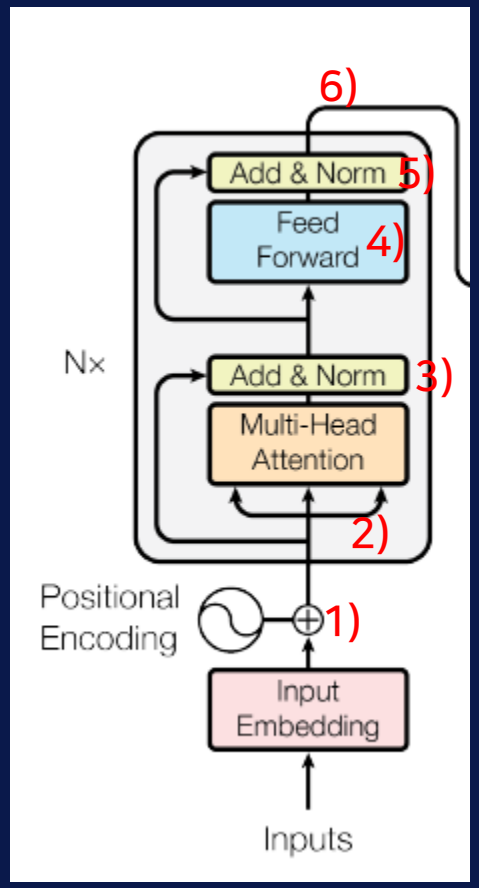
➡ 두 번의 선형 변환 사이에 ReLU함수가 적용됨

$$LN = LayerNorm(x + Sublayer(x))$$

➡ 잔차 연결(Add) 이후 정규화(Norm)

02. Transformer

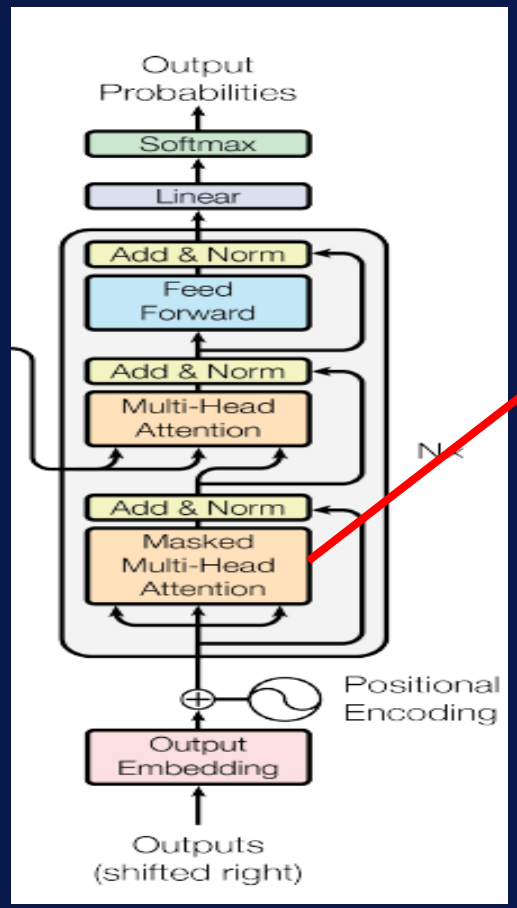
Encoder 정리

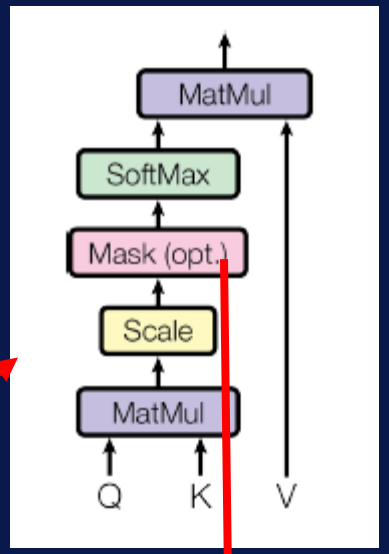


1) 임베딩 벡터에 positional encoding값을 더해 인코더에 입력
2) Input으로 query, key, value를 계산해 Multi-Head attention에 입력
3) Add&Norm에서 잔차 연결과 정규화 진행
4) Feed Forward층에서 선형변환과 활성화 함수 적용
5) Add&Norm에서 잔차 연결과 정규화 진행
6) Encoder의 output배출

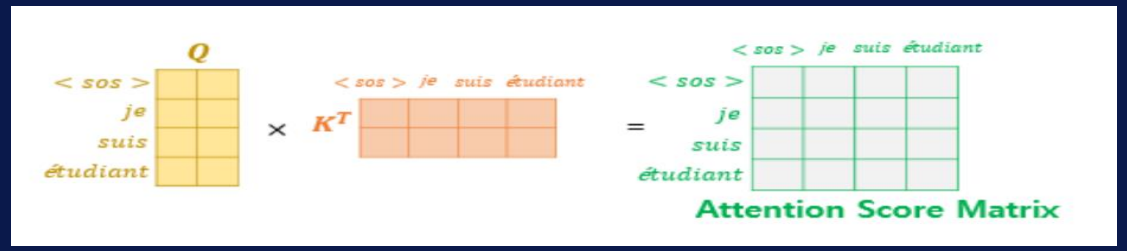
02. Transformer

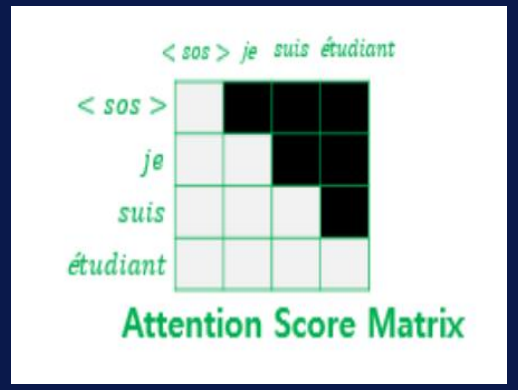
Decoder의 Masked Multi-Head Attention





무엇을 마스킹할까???





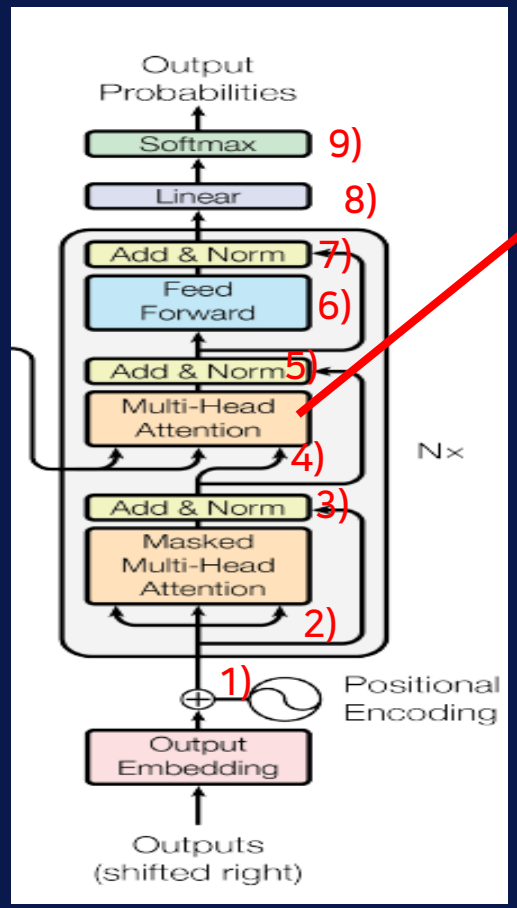
➡ Qeury와 key를 사용해 구한 Attention Score Matrix에서 현재시점보다 미래의 단어들을 가려줌

➡ Masking하는 이유?

Transformer는 문장의 행렬을 입력 받아 모든 시점의 output 시퀀스를 참고할 수 있음. 따라서 현재 시점을 포함한 이전 시점에 입력된 단어들만 참고할 수 있도록 현재 시점 이후의 단어들을 masking해줌

02. Transformer

Decoder정리와 encoder-decoder attention층(self attention X)



Key, value : encoder의 output
Query : decoder의 self-attention계층의 출력

1) 임베딩 벡터에 positional encoding값을 더해 디코더에 입력
2) query, key, value를 계산해 Multi-Head attention에 입력
3) Add&Norm에서 잔차 연결과 정규화 진행
4) 다음 Multi-head Attention에 query전달
5) Add&Norm에서 잔차 연결과 정규화 진행
6) Feed Forward층에서 선형변환과 활성화 함수 적용
7) Add&Norm에서 잔차 연결과 정규화 진행
8) 선형변환
9) Softmax함수 적용

02. Transformer 그 이후

2021년 기준 최신 고성능 모델들은 Transformer 아키텍처를 기반으로 하고있음
- GPT : Transformer의 decoder 아키텍처를 활용
- BERT : Transformer의 encoder 아키텍처를 활용

03 – BERT

2022 / 05 / 19
D&A 운영진 이경욱

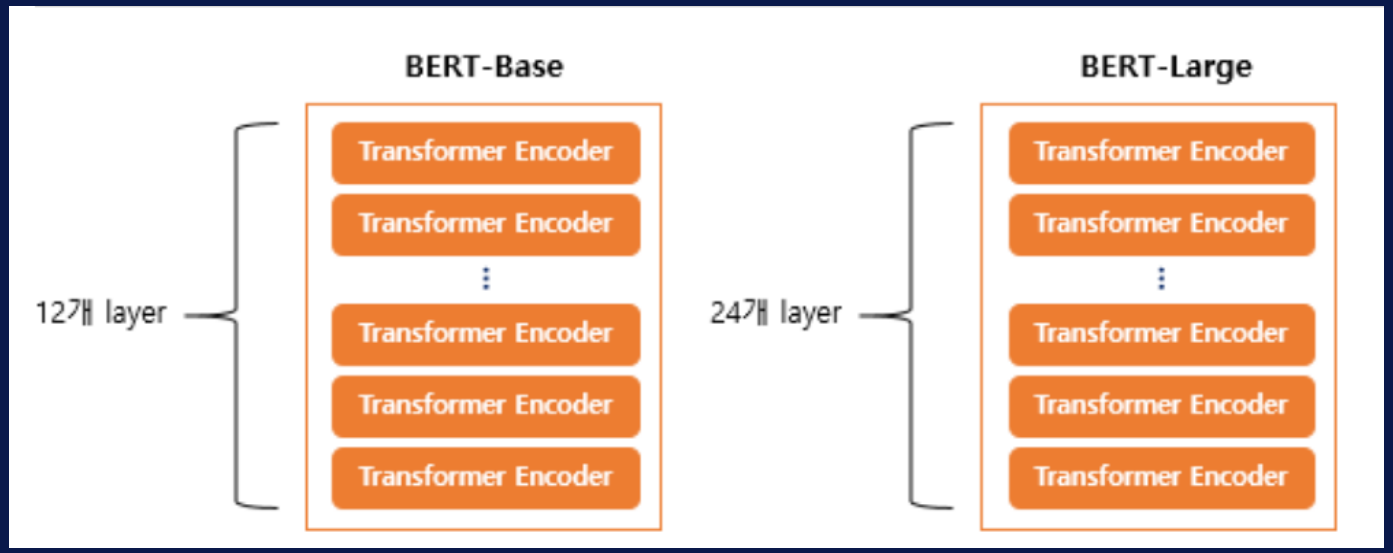
03. BERT

BERT란?

BERT는 Transformer를 이용하여 구현되었으며, 위키피디아(25억 단어)와 BooksCorpus(8억 단어)와 같은 레이블이 없는 텍스트 데이터로 사전 훈련된 언어 모델이다.

특정한 task를 해결하기 위한 모델이 아닌, 언어 전반을 이해하고 이를 벡터로 표현하는데 특화된 모델

BERT의 기본 구조



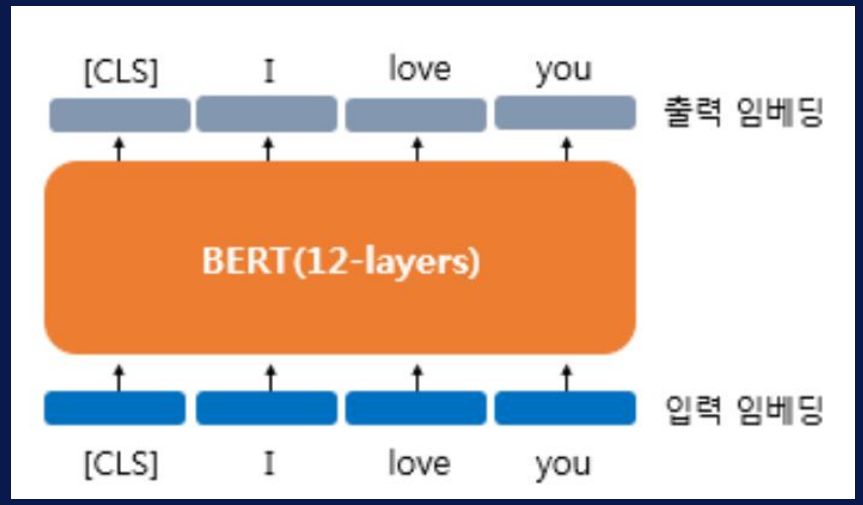
➡ 모델 사이즈에 따라 다른 버전

➡ Transformer의 인코더를 사용

03. BERT

BERT의 두 가지 특별 토큰

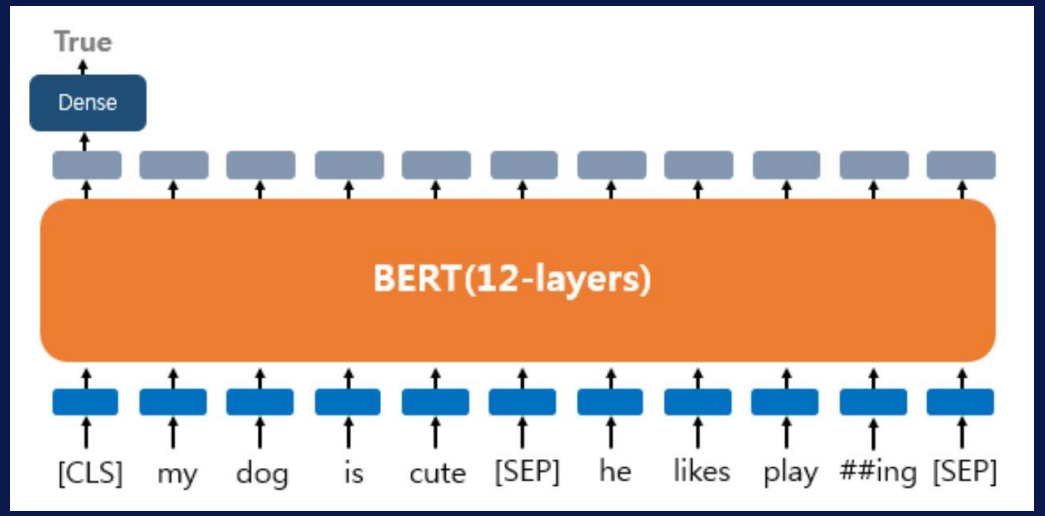
1) CLS



➡ 분류 문제를 풀기 위한 특별 토큰

➡ 문서의 시작 부분에 위치

2) SEP

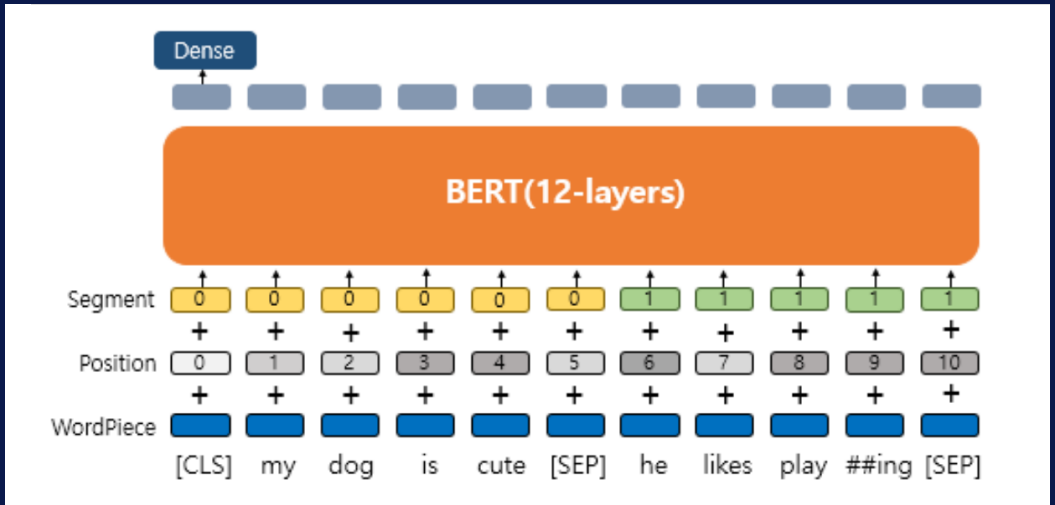


➡ 문장 구별을 위한 토큰

➡ 문장의 끝에 위치

03. BERT

BERT의 임베딩 계층



➡ 3개의 임베딩 계층의 더해 만듬

1) Token Embeddings

WordPiece라는 BPE 유사 알고리즘 토크나이저를
기본 Word Embedding 계층으로 사용

2) Position Embeddings

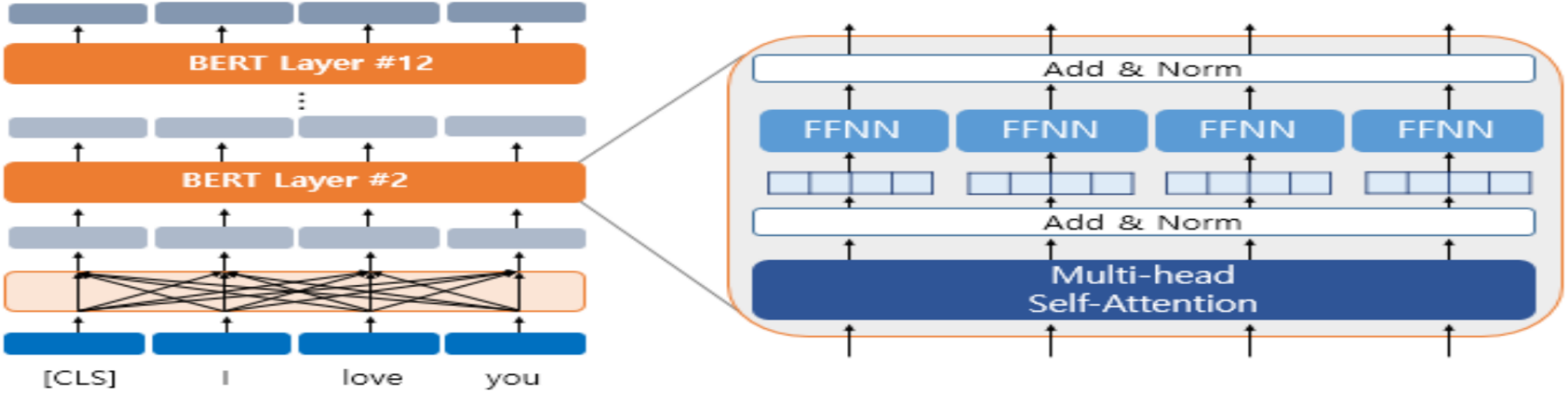
Transformer와 달리 각 단어에 대한 position embedding
vector를 학습으로 구해 더해줌

3) Segment Embeddings

이어지는 두 문장을 구별하기 위한 embeddings
-> 두 문장을 입력받는 task가 아닐 경우 같은 값으로 설정

03. BERT

BERT의 문맥 반영



➡ Self-attention을 사용해 각 단어들은 모든 단어를 참고하게됨

03. BERT

BERT의 Pre_train - Masked Language Model, MLM

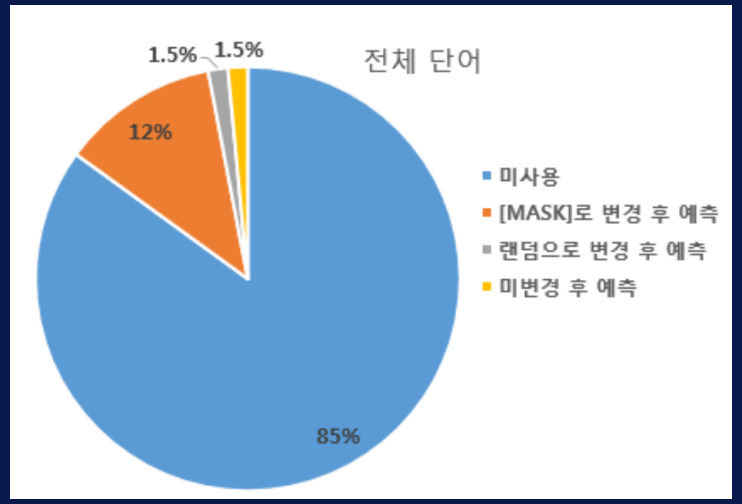
입력 텍스트 중 15%의 단어를 선택한다

⬇ 15%의 단어 중에서!

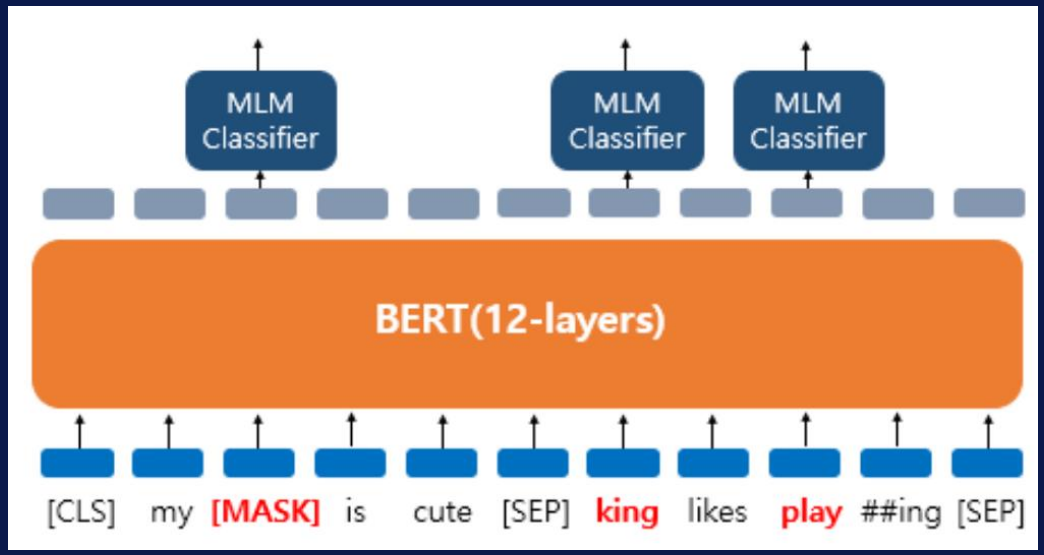
1) 80%의 단어를 [MASK](특별토큰)로 변경한다.

2) 10%의 단어들은 랜덤으로 다른 단어로 변경한다.

3) 10%의 단어들은 동일하게 둔다



My dog is cute. He likes playing이라는 문장을 넣은 경우



➡ dog는 [MASK]로 변경

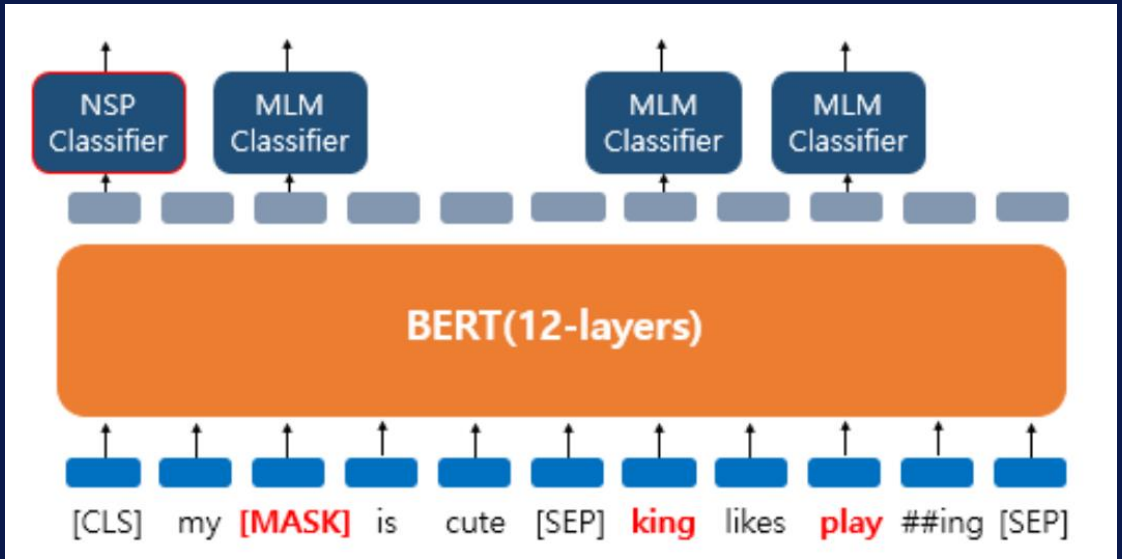
➡ He는 king으로 변경

➡ play는 그대로!

03. BERT

BERT의 Pre_train – Next Sentence Prediction, NSP

다음 문장에 대한 예측



➡ CLS 출력층을 통해 이어지는 문장인지에 대한 여부 분류

➡ MLM과 동시에 훈련 진행

훈련 과정

1) 두 문장이 이어지는 문장의 경우
Label = IsNextSentence

2) 두 문장이 이어지지 않는 경우
Label = NotNextSentece

⬇

위 두 경우에 대해 50:50비율로 훈련

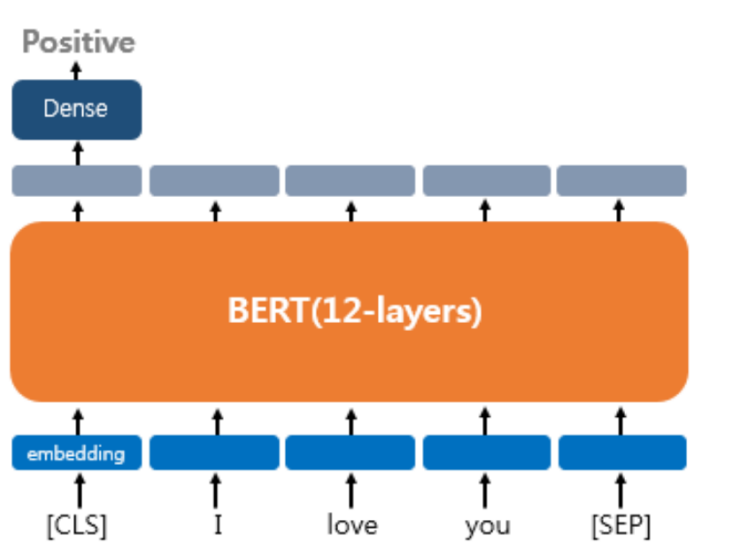
03. BERT

대표적인 language task에서 BERT의 fine-tuning

1) Single Text Classification

-> 하나의 문서에 대한 텍스트 분류 task

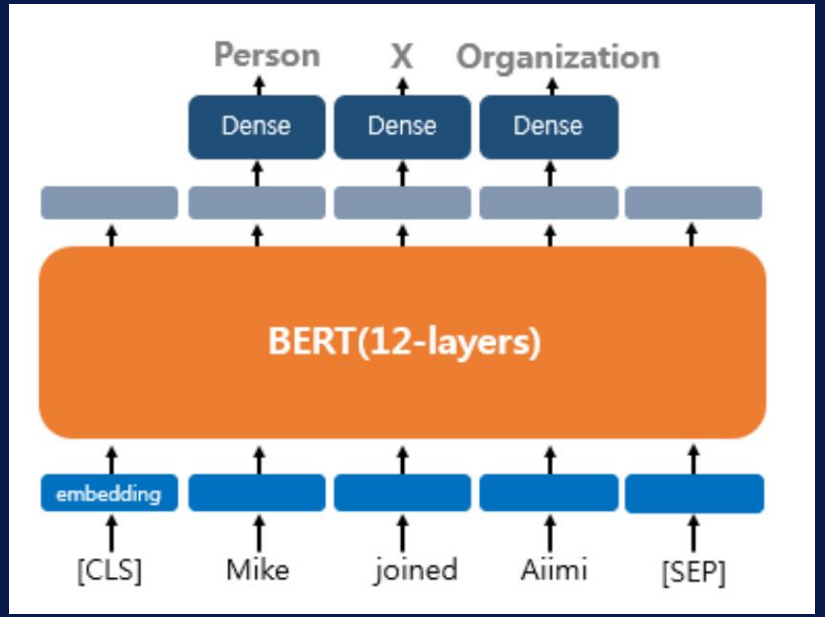
ex) 영화 리뷰 감성 분류, 로이터 뉴스 분류



➡ 분류 문제는 CLS 출력층을 사용

2) Tagging

ex) 각 단어의 품사 tagging, 개체명 인식

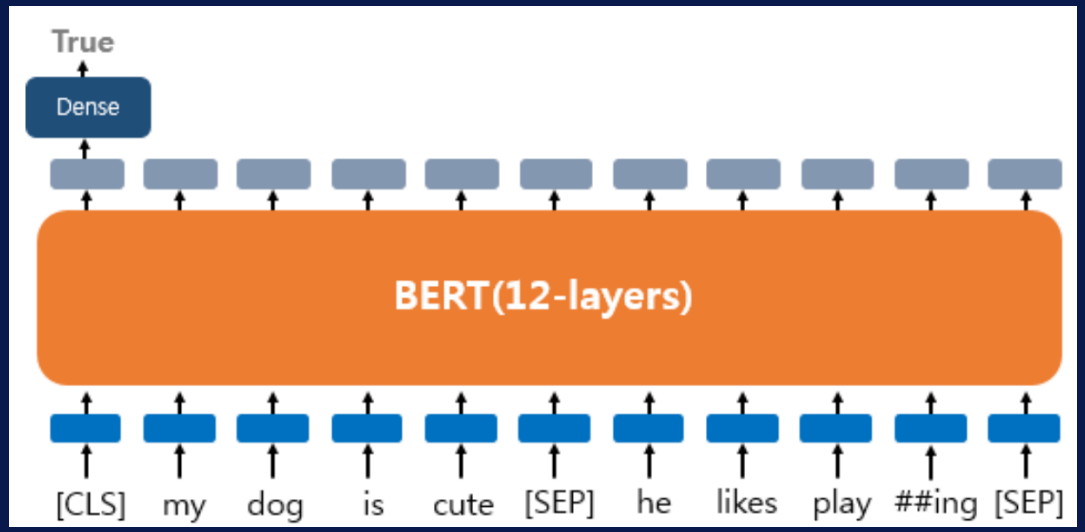


03. BERT

대표적인 language task에서 BERT의 fine-tuning

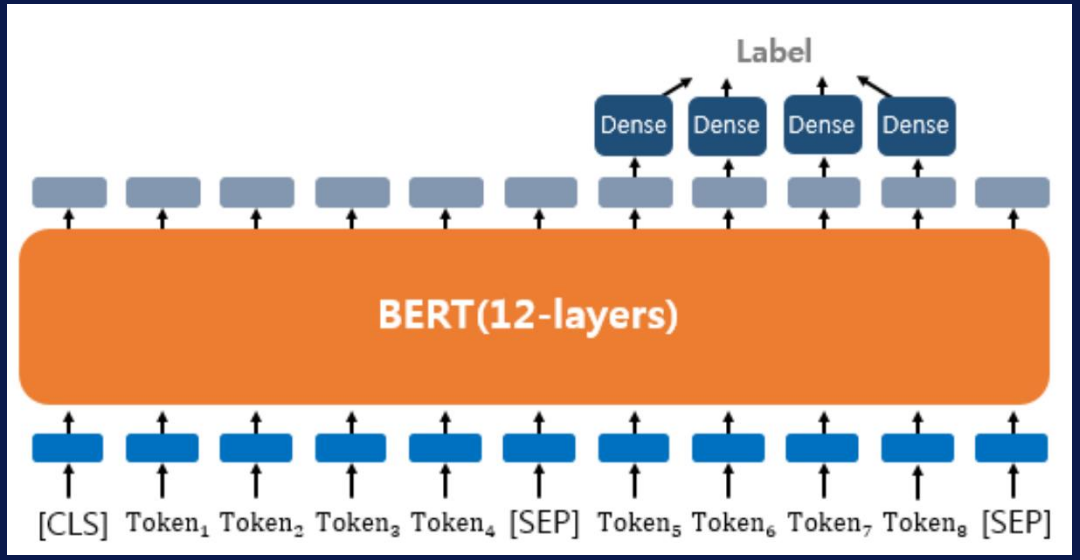
3) Text Pair Classification or Regression

-> 두 문장이 주어졌을 때, 이어지는 문장인가?
-> 두 문장이 어떠한 관계인가?



4) QA

-> 질문에 대한 답을 본문에서 추출해서 답하는 task



첨부자료 출처

BERT
https://wikidocs.net/115055

폰트

네이버 글꼴 모음 _ 나눔 스퀘어 사용
출처 : https://hangeul.naver.com/font

D&A

Deep Session 9차시

Thank You.

2022 / 05 / 19

D&A 운영진 이경욱