Vison Transformer



Presented By
백창우

Outline

- Background
- Transformer
- Vison transformer

Background

Keyword

- 자연어 처리
- RNN
- Gradient vanishing
- Seq2seq(sequence-to-sequence)
- Attention

Background

- **Sequential Data** VS **Non-sequential Data**

- Sequential Data : 어떤 순서로 오느냐에 따라서 단위의 의미가 달라지는 데이터

  ex) 자연어, 시계열 데이터

- Non-sequential Data : 순서에 따라 단위의 의미가 달라지지 않는 데이터

  ex) 일반적인 사진

CVSP

Sequential Data

- 자연어 예시

  I love you =/= You love I

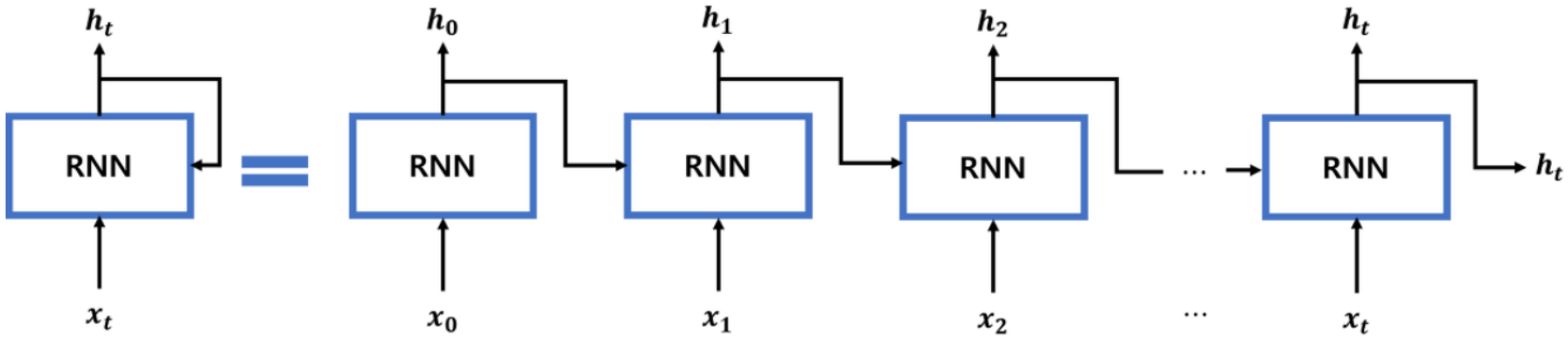
  I am ground =/= ground am I

- 시계열 데이터

  주식 데이터 : 하락 —> 하락 —> 상승 =/= 상승 —> 하락 —> 하락

Background

- 자연어 처리에서 널리 쓰였던 RNN 구조

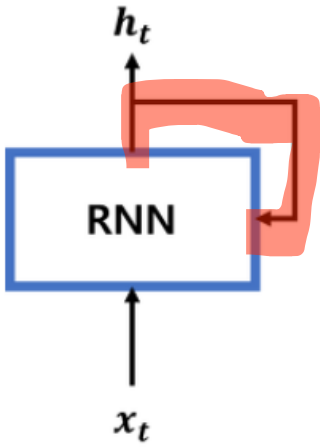


[그림 1] RNN의 순환구조

Background

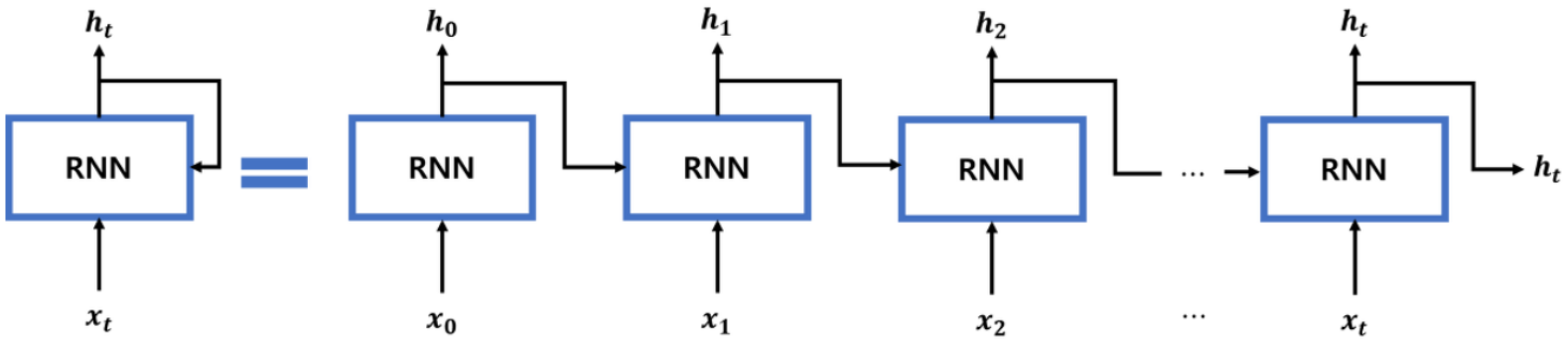
- X : Input vector

- h : hidden state
  - Hidden layer의 출력



Recurrent

Background

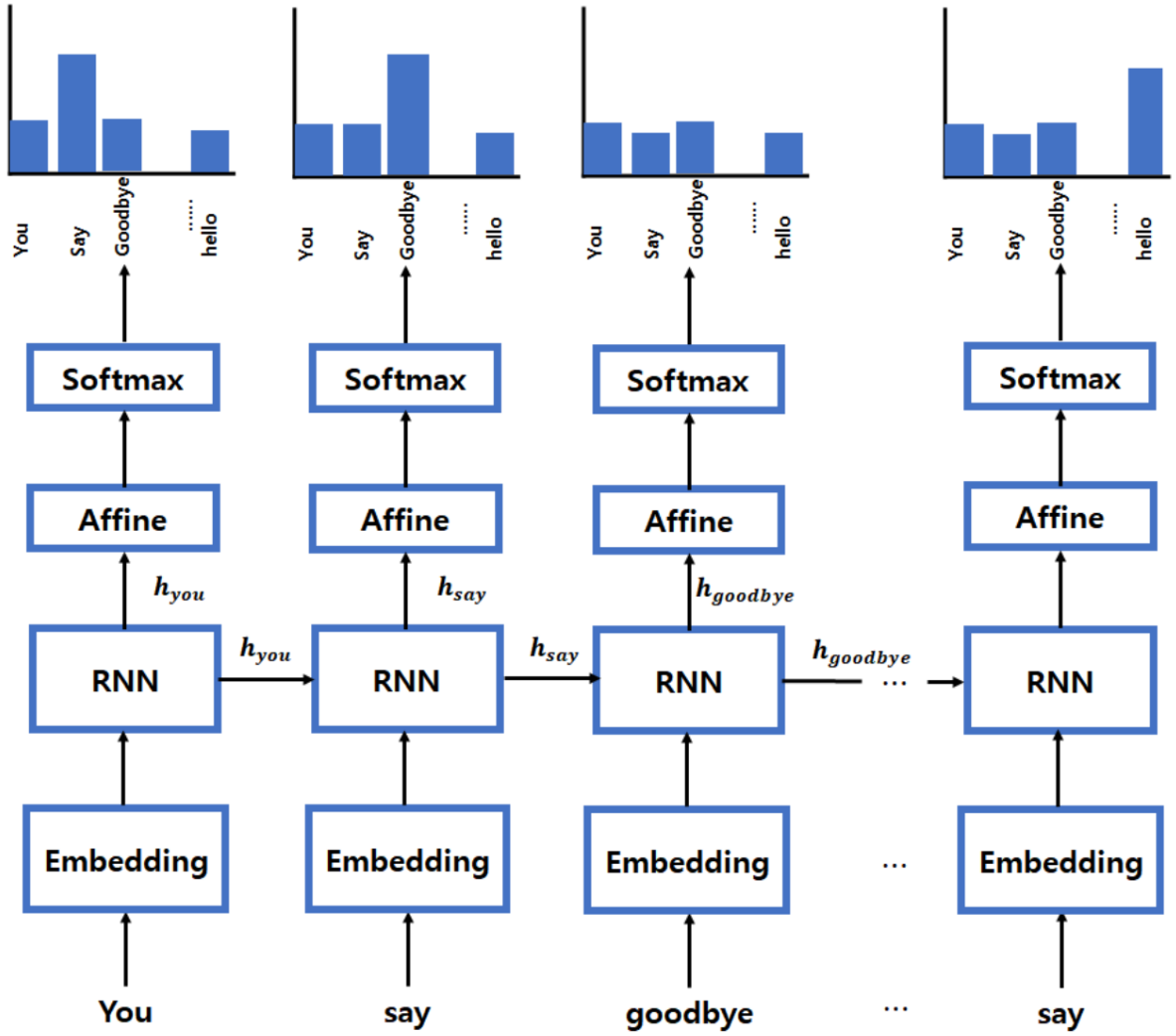


[그림 1] RNN의 순환구조

- 이전 시점의 hidden state 가 다음 시점의 은닉층의 input으로 들어감(Recurrent)

Background

- Ex) You say goodbye and I say hello



[그림 2] RNN의 언어 모델

Background

$$h_{hello} = \tanh(h_{say}W_h + x_{hello}W_x + b)$$

$$y_{hello} = \mathrm{f}\,(w_y \times h_{hello} + b_y)$$

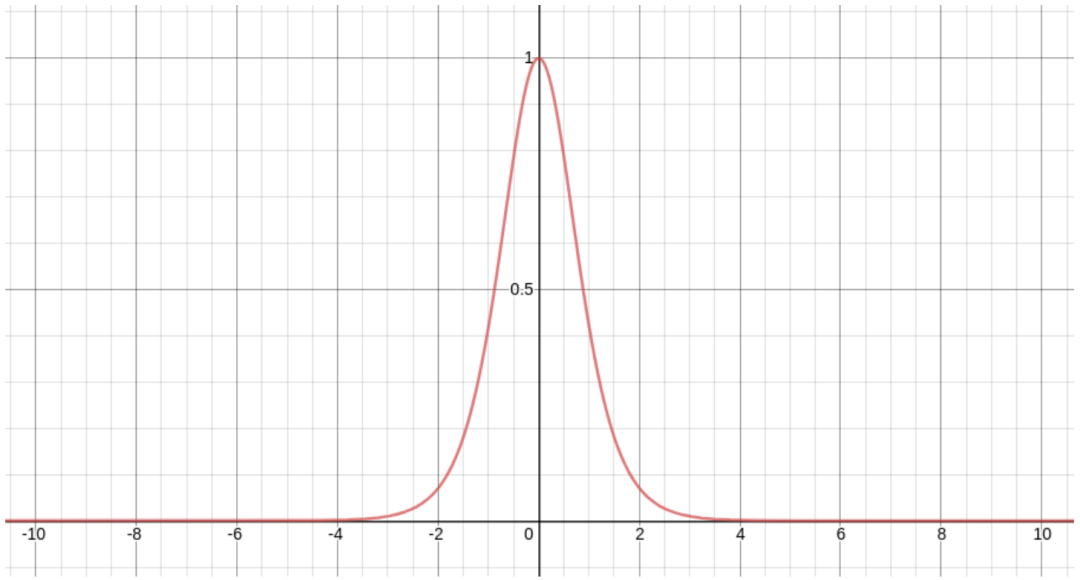
Background

- RNN Problem?

- Gradient vanishing
  - Layer 수가 많아 질수록 Back propagation이 진행되면서 기울기가 소실

Background

- Act function : tanh 미분



- 최댓값이 1이고 (-4, 4)이외의 범위에서는 거의 0에 가까운 값
- Back propagation 과정에서 이 값을 반복해서 곱해주어야 함
- recurrent가 10회, 100회 반복된다고 보면, 이 값의 10제곱, 100제곱이 식 내부로 들어가여 gradient vanishing

Background

- Gradient vanishing 을 해결하기 위해 LSTM , GRU등 여러 variation 등장

Check!

18. 기본 순환 신경망(RNN, Recurrent Neural Network)과 이의 변형 기법에 대한 설명으로 옳지 않은 것은?

① RNN은 계층이 깊어질수록 그레디언트 소멸 문제(vanishing gradient problem)가 발생하기 쉽다.

② LSTM(Long Short Term Memory)의 출력은 이전 시점의 입력값과 은닉층의 값뿐만 아니라 이후 시점의 입력값과 은닉층의 값에도 영향을 받는다.

③ RNN에서 순환 드롭아웃(recurrent dropout)은 과적합(overfitting)을 방지하기 위해 사용된다.

④ GRU(Gated Recurrent Unit)는 LSTM과 유사한 원리로 동작하지만, 게이트 수를 줄여 더 간결한 구조를 갖는다.

Check!

20. 그래디언트 소멸 문제(vanishing gradient problem)에 대한 설명으로 옳지 않은 것은?

① 특정 활성화 함수를 사용하는 계층이 신경망에 많이 추가되면 학습이 되지 않는 현상을 말한다.

② 0이나 0에 가까운 수가 시그모이드(sigmoid) 함수에 들어오면 그래디언트가 0에 가까워 학습이 되지 않는다.

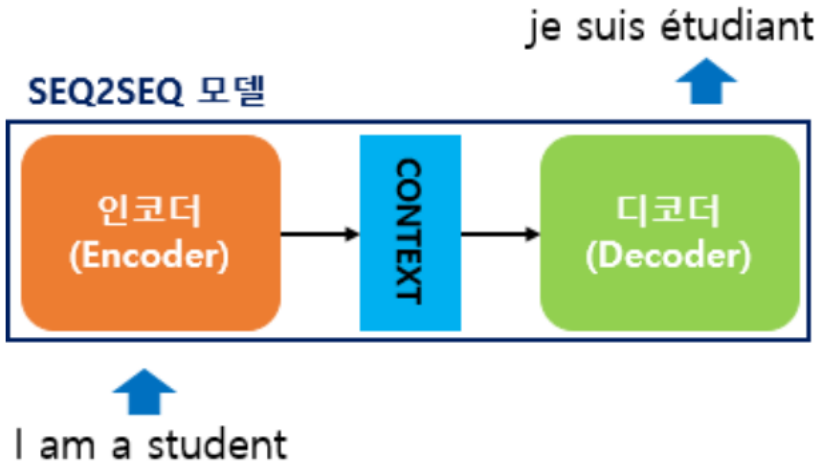
③ 그래디언트의 절댓값이 너무 작아지면 학습이 되지 않아 가중치(weight)와 편향(bias) 값들이 효과적으로 업데이트 되지 못한다.

④ 그래디언트 소멸 문제를 해결하기 위해 DNN(Deep Neural Network)에서는 활성화 함수로 ReLU 함수를 사용한다.

Seq2seq

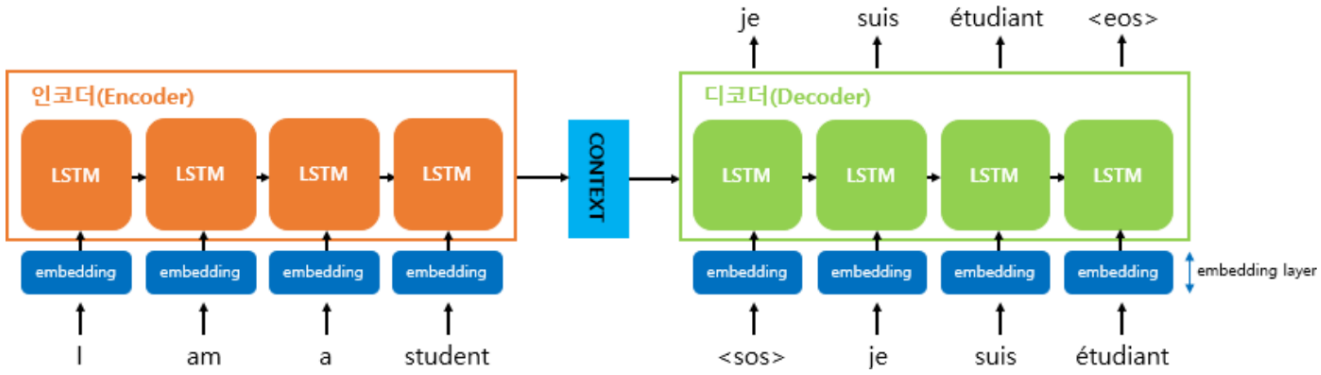
- Seq2seq
  - 번역기에서 대표적으로 사용되는 모델
  - Encoder와 Decoder로 구성

- Encoder
  - 부호화 하는 장치
- Decoder
  - 코드를 해독하는 장치



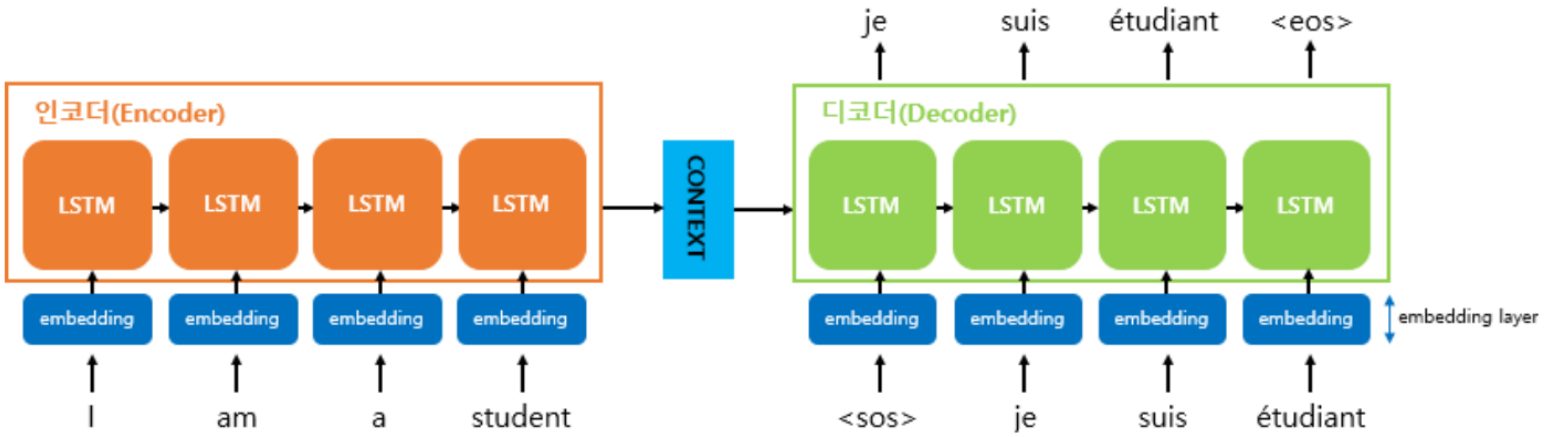
Seq2seq

- Encoder mechanism
  - 입력 문장은 단어 토큰화를 통해 단어 단위로 쪼개짐
  - 모든 단어들을 순차적으로 Encoder에 입력
  - 이전 LSTM 셀의 hidden state가 다음 셀에 입력
  - 모든 단어를 입력 받은 뒤 마지막 hidden state가 Decoder로 전송
  - Context Vector : 마지막 hidden state이자 모든 단어 정보들이 압축된 Vector



Seq2seq

- Decoder mechanism (Test)
  - 문장의 시작을 의미하는 심볼 <sos> 초기 입력
  - <sos>가 입력 되면 다음에 등장할 확률이 높은 단어 예측(softmax) ex) je
  - 이전 셀의 hidden state가 다음 셀의 입력
  - 문장의 끝을 의미하는 심볼 <eos>가 다음 단어로 예측 될때까지 반복



Seq2seq

- Decoder mechanism (Train)
  - Encoder에서 나온 Context Vector와 <sos> je suis etudiant를 입력 받았을때 Je suis etudiant <eos>가 나와야 된다고 정답을 알려주면서 훈련

- Test Train mechanuism difference?
  - Train : Context Vector와 <sos> je suis etudiant 전체 문장 입력
  - Test : Context Vector와 <sos> 심볼만 입력

Seq2seq

- Embedding
  - 기계는 텍스트, 이미지보다 숫자를 잘 처리
  - 단어나 이미지와 같은 데이터를 숫자로 처리된 Vector로 바꾼 결과 혹은 그 일련의 과정을 의미



Attention

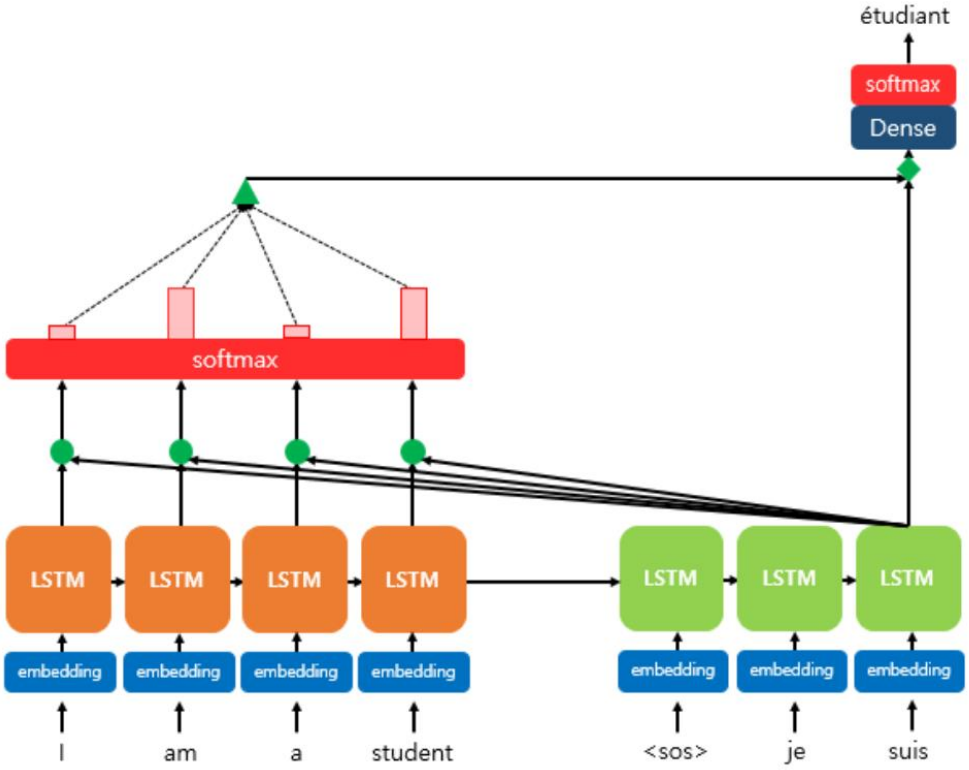
- 기존 RNN 기반 seq2seq의 단점
  - 모든 단어 정보를 하나로 압축하려고 하니 정보 손실 및 불균형 압축이 일어남
  - RNN 기반 아키텍쳐의 고질적인 문제인 Gradient Vanishing

- So Attention
  - 기존 Context Vector 는 Encoder에서 마지막 단어가 지배적인 hidden state로 고정
  - Attention mechanism에서는 매 시점마다 Encoder에서 전체 입력 문장을 다시 한번 참고
    즉 Context Vector가 시점마다 달라짐
  - 입력 문장의 단어를 참고할 때 다 동일한 비율이 아닌 해당 시점에서 예측 해야할 단어와 가장 연관있는 입력 단어부분을 더 attention 즉 단어마다 가중치가 다름

Attention

- Dot-Product Attention
  - 가장 수식적으로 이해하기 쉬운 attention
  - 유사도의 기준이 Dot-Product

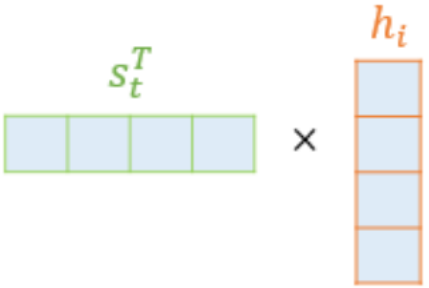
- Notation
  - Decoder의 hidden state : s_t
  - Attention Value : a_t
    - Attention 에서의 Context Vector



Attention

- Find Attention Value
  - First find Attention score with dot-product

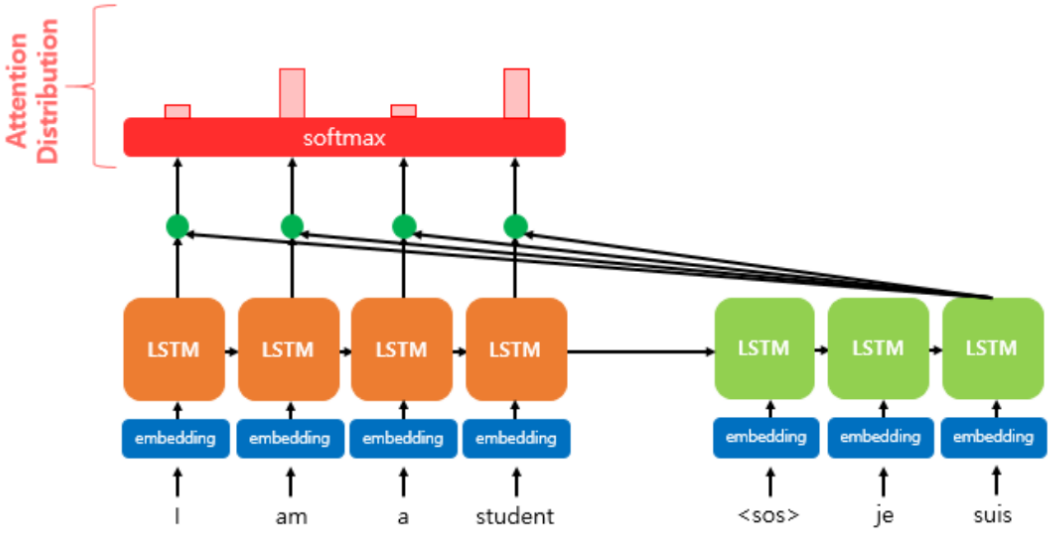
$$score(s_t,\ h_i) = s_t^T h_i$$



  - Attention score의 집합 : $e^t = [s_t^T h_1, \ldots, s_t^T h_N]$

Attention

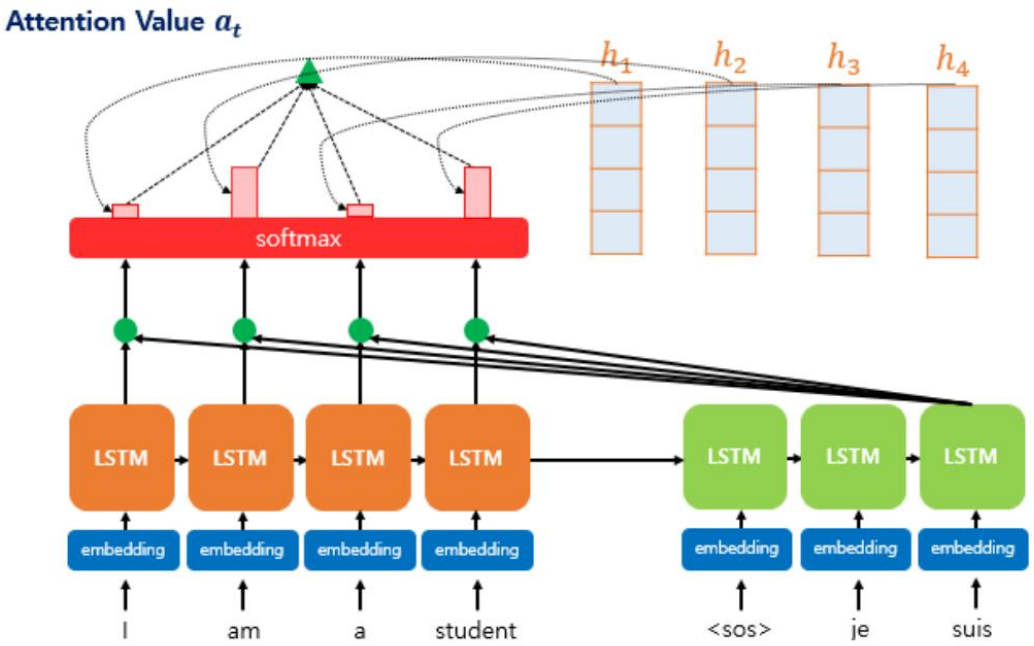
- Find Attention Value
  - second find Attention Distribution with softmax : $\alpha^t = softmax(e^t)$
    - 분포들의 합 =1



Attention

- Find Attention Value
  - Third Weighted Sum

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

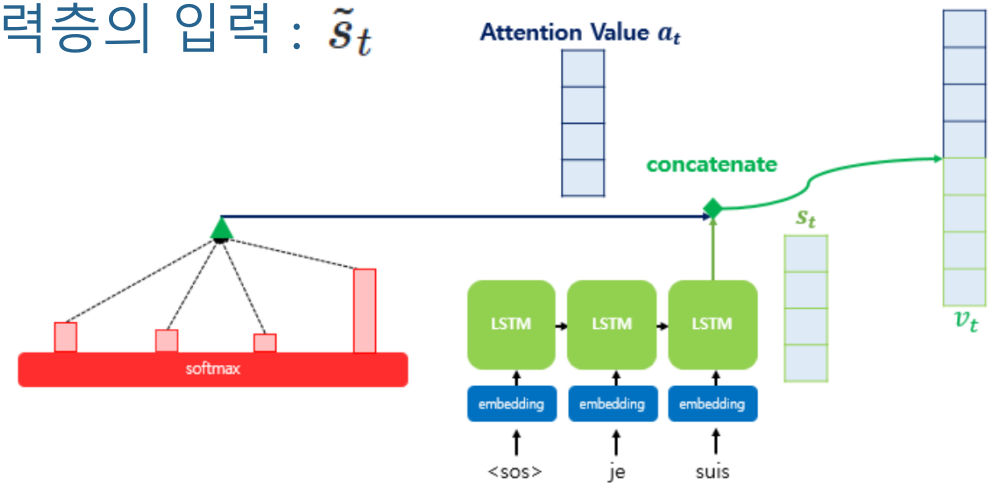


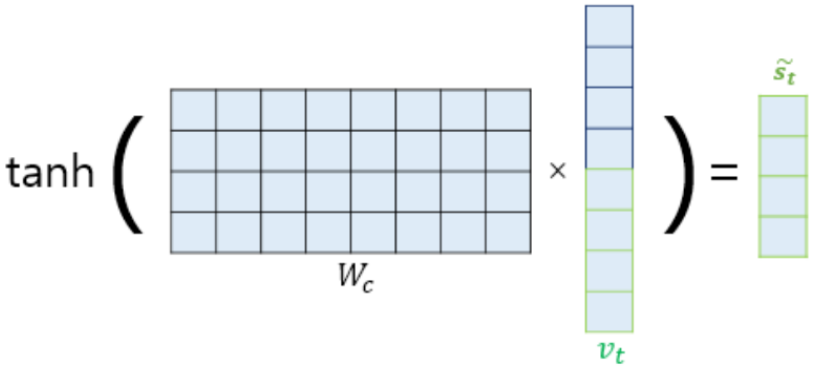
Attention

- Concatenate
  - a_t 를 s_t 와 결합(Concatenate)하여 하나의 벡터로 만듬 : v_t =$[a_t; s_t]$
  - Attention mechanism 에서 출력층의 입력 : $\tilde{s}_t$

$$\tilde{s}_t = \tanh(\mathbf{W_c}[a_t; s_t] + b_c)$$

$$\hat{y}_t = \text{Softmax}\,(W_y \tilde{s}_t + b_y)$$





Attention

- Summary

$$score(s_t,\ h_i) = s_t^T h_i$$

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

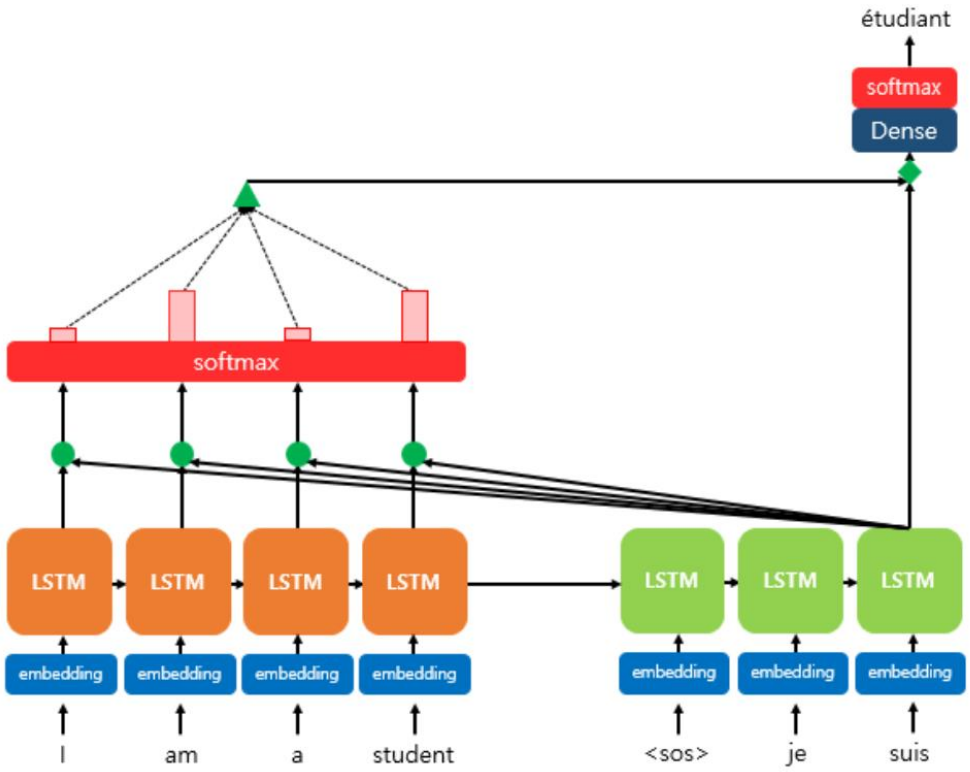
$$\alpha^t = softmax(e^t)$$

★ $$a_t = \sum_{i=1}^{N} \alpha_i^t h_i$$

$$v_t = [a_t; s_t]$$

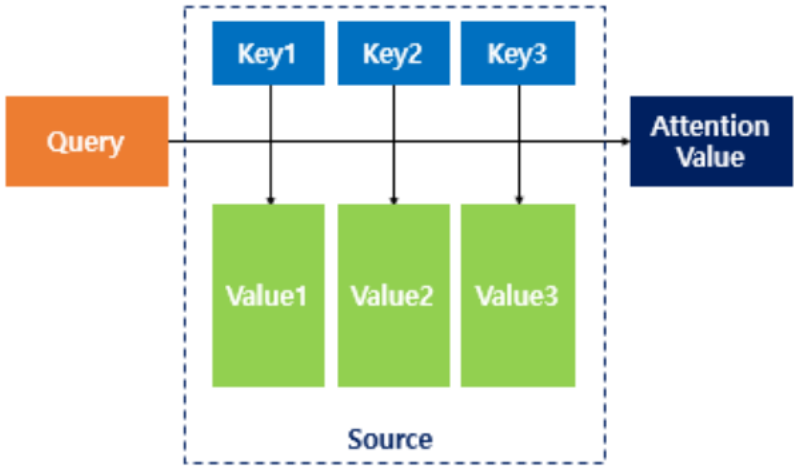
$$\tilde{s}_t = \tanh(\mathbf{W_c}[a_t; s_t] + b_c)$$

$$\hat{y}_t = \text{Softmax}\,(W_y \tilde{s}_t + b_y)$$



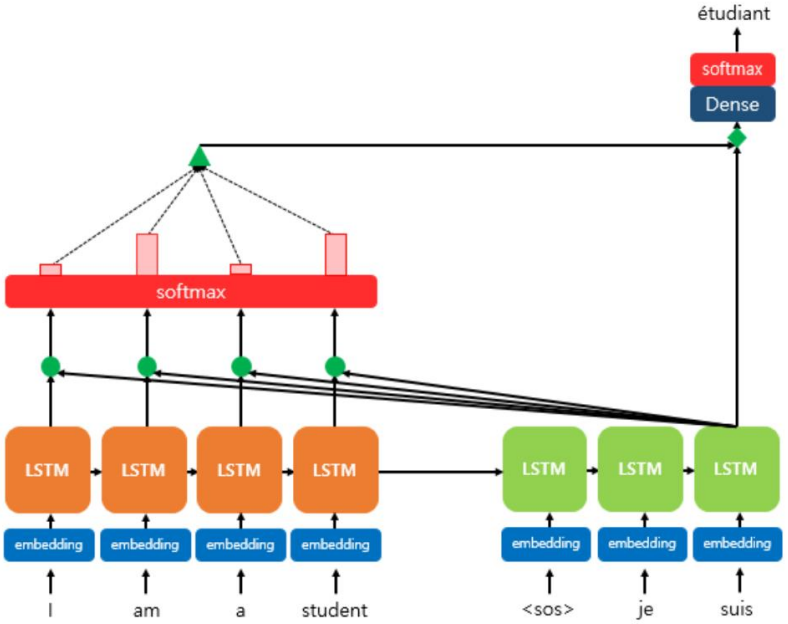
Attention

- Attention(Q,K,V) = Attention value
  - 주어진 Query에 대해서 모든 key와의 유사도를 각각 구함
  - 구해낸 이 유사도로 가중치로 하여 키와 맵핑되어있는 각각의 value에 반영
  - Value를 weight sum하여 Attention Value를 구함



Transformer

- History
  - RNN 구조가 나오고 이 구조의 문제(Gradient Vanishing)해결하기 위해
  - GRU, LSTM 등 여러 변형 구조가 나옴
  - RNN 기반 seq2seq 구조에서 여러 문제를 해결하기위해 attention 도입
  - 그래도 Recurunt 구조에서는 벗어 날수 없는 여전한 문제점 존재



Transformer

- **Attention Is All You Need** (NIPS 2017) Ashish Vaswani, et al.

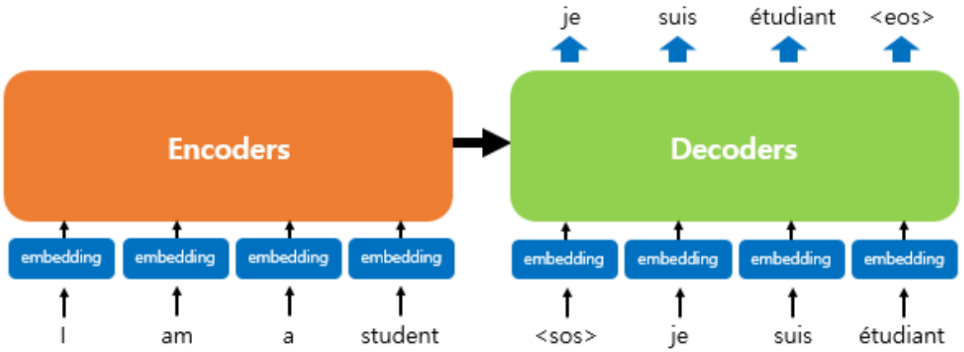
- 트랜스포머 구조를 처음 발표한 구글 브레인의 논문

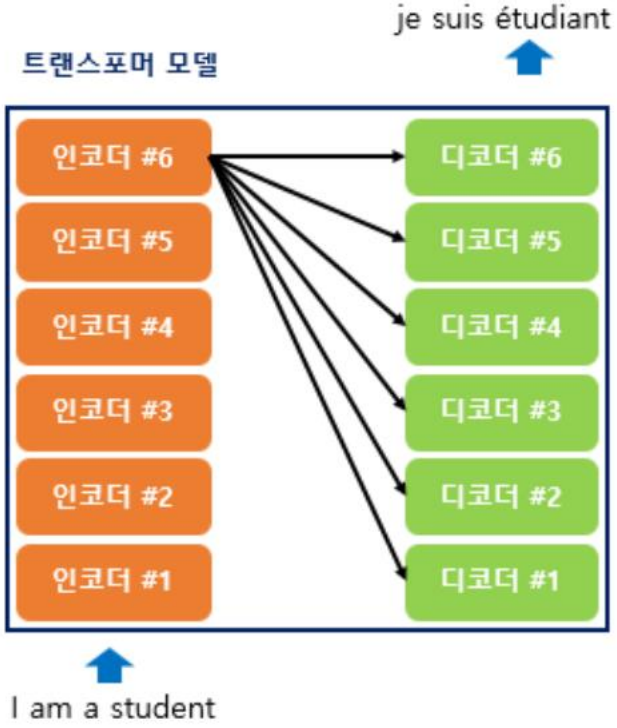
- RNN과 LSTM, GRU 등 여러 RNN 변형 네트워크의 문제를 해결하기 위해 나온 구조로

  attention만을 이용해 인코더 디코더를 만듬

Transformer

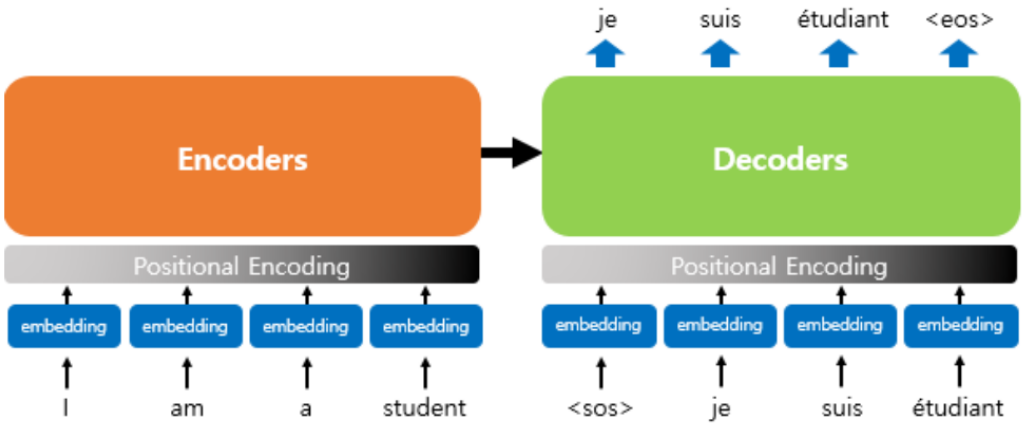
- Transformer
  - RNN이 t개의 시점을 가지는 구조였다면
    인코더,디코더의 개수가 N개로 구성된 구조

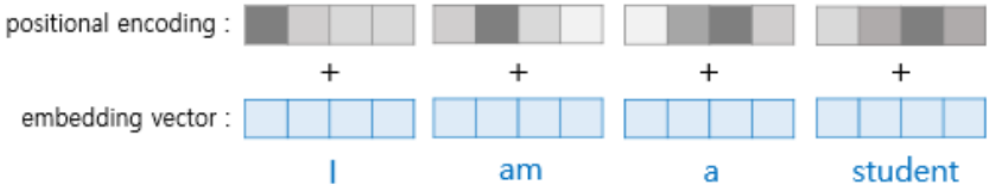




Transformer

- Transformer
  - 단어 입력을 순차적으로 받는 방식이 아니므로 단어의 위치정보를 알려주어야함
  - 임베딩 벡터에 위치정보(Positional Encoding)를 더하여 모델의 입력으로 입력





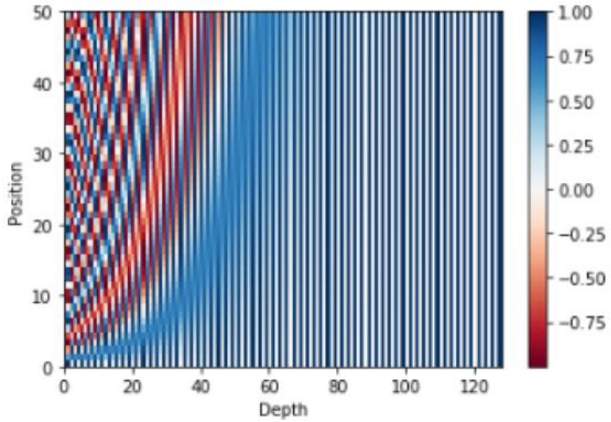
Transformer

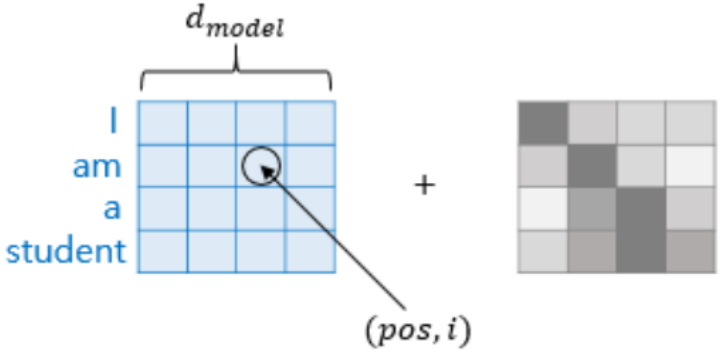
- Positional Encoding
  - pos : 입력 문장에서의 임베딩 벡터의 위치
  - i : 임베딩 벡터 내의 차원의 인덱스
  - 인덱스 i 가 짝수일때 sin함수
    홀수일때 cos함수

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

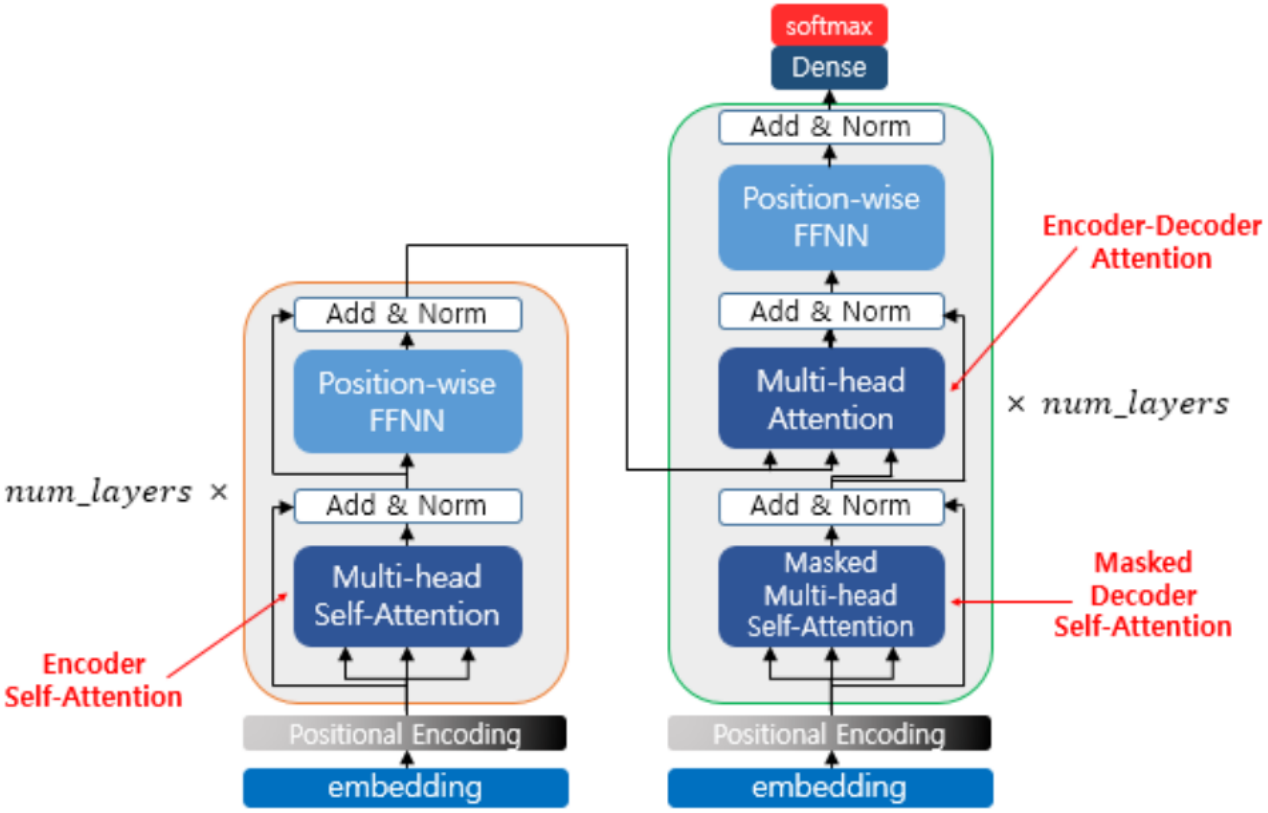
$$PE_{(pos,\,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

  - d_{model} : Transformer의 모든 층의 출력

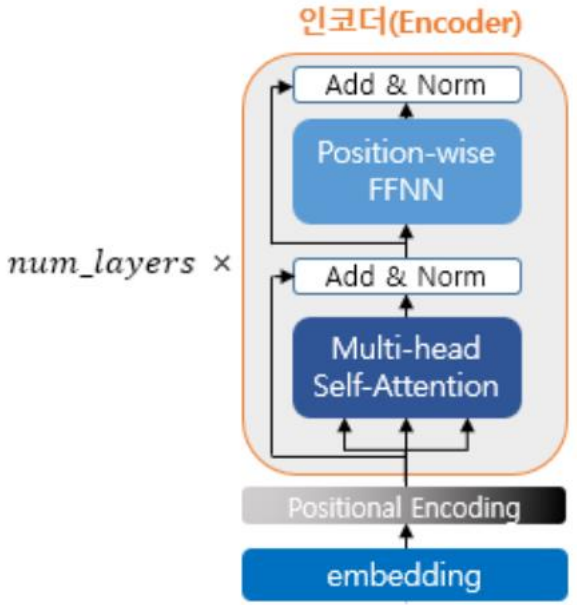




- Transformer

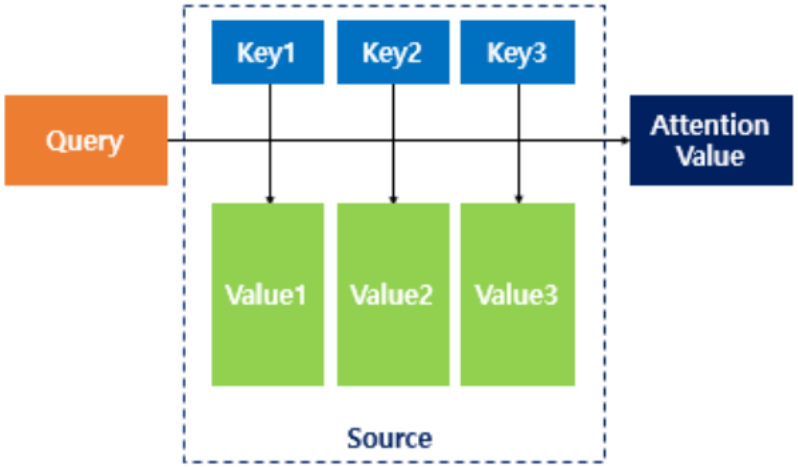


- Encoders
  - num_layers 개수만 큼 인코더 층을 쌓음
  - 하나의 인코더 layer에 2개의 sublayer로 구성
  - 셀프 어텐션 layer와 Poisition-wise FFNN
  - 멀티 헤드 셀프 어텐션 layer
    - 셀프 어텐션을 병렬적으로 사용했다는 의미
  - Poisition-wise FFNN
    - 일반적인 피드 포워드 신경망 (MLP 구조)



Transformer

- Self – Attention
  - Attention을 자기 자신에게 수행
  - Self – Attention은 입력 문장 내의 단어들끼리 유사도를 구함

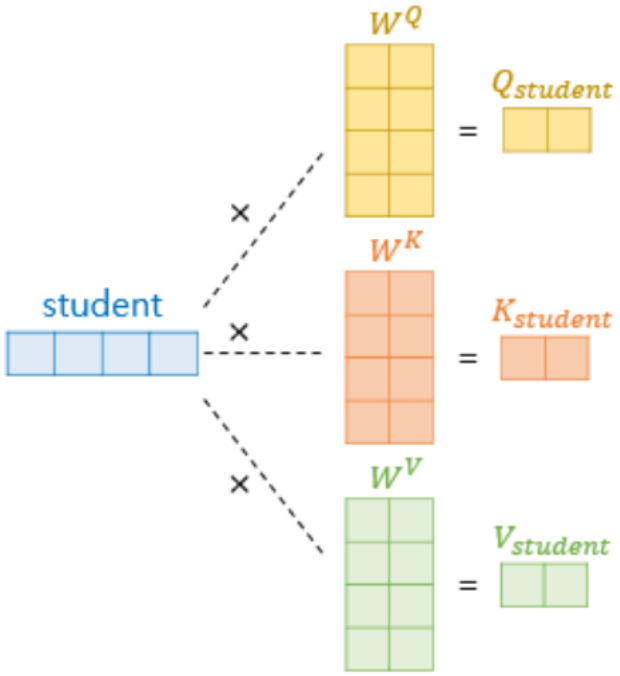


Transformer

- Self – Attention mechanism
  - 각 단어 벡터들로부터 Q,K,V vector 구함
  - 논문에서의 차원 : d_{model}=512

$$\text{num_heads} = 8$$

  - Q,K,V vector의 차원: $\frac{d_{model}}{\text{num_heads}} = 64$
  - 각 W의 차원 : $d_{model} \times (d_{model}/\text{num_heads})$



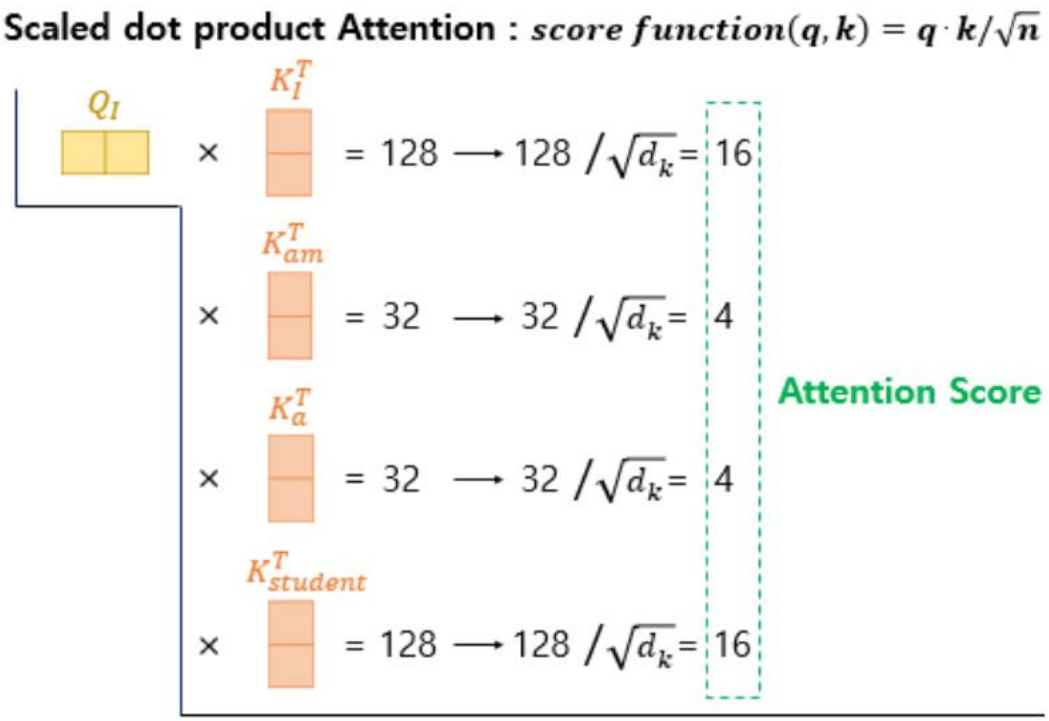
Transformer

- Self – Attention mechanism
  - 기존 mechanism 처럼 각 Q vector는 모든 K vector에 대해 attention score, distribution를 구하고 이를 모든 V vector와 Weight sum하여 attention value(context vector)를 구함

- Scaled dot product Attention
  - 기존의 dot product에 특정값으로 나눔 $score(q, k) = q \cdot k / \sqrt{n}$

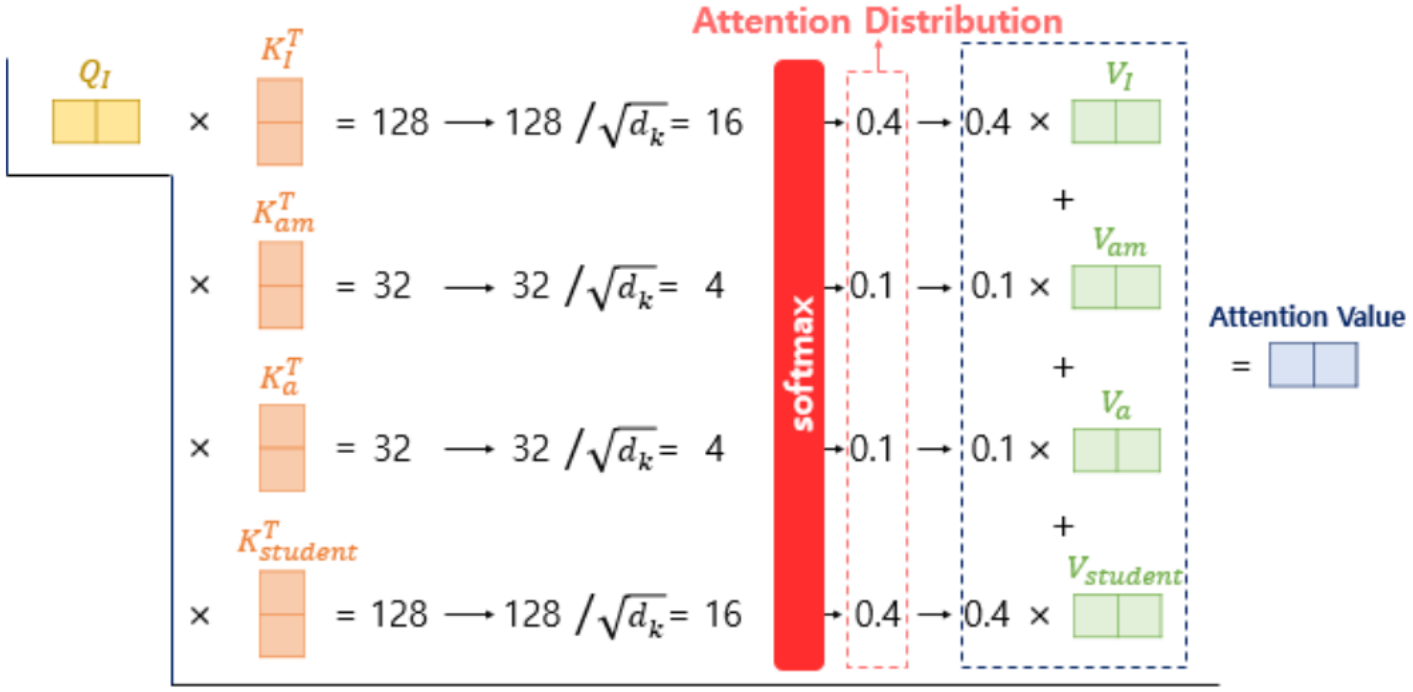
Transformer

- Scaled dot product Attention
  - 단어 " I "에 대해 Q,K,V
  - Q벡터에 K벡터의 Transpose를 곱하고 $\sqrt{d_k}$ 로 나누어 Attention score 구함
  - d_k : $d_{model}/\text{num_heads}$



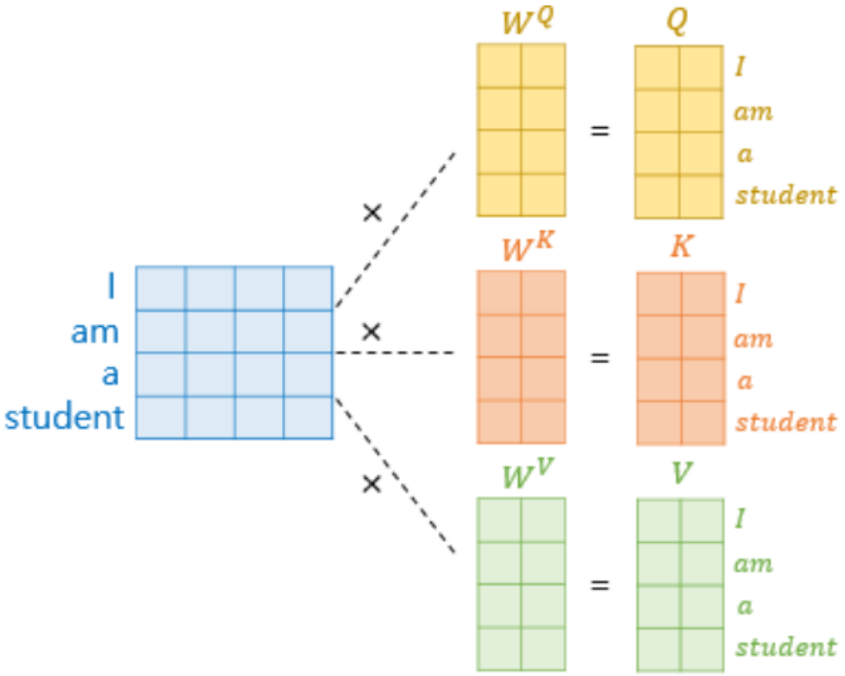
Transformer

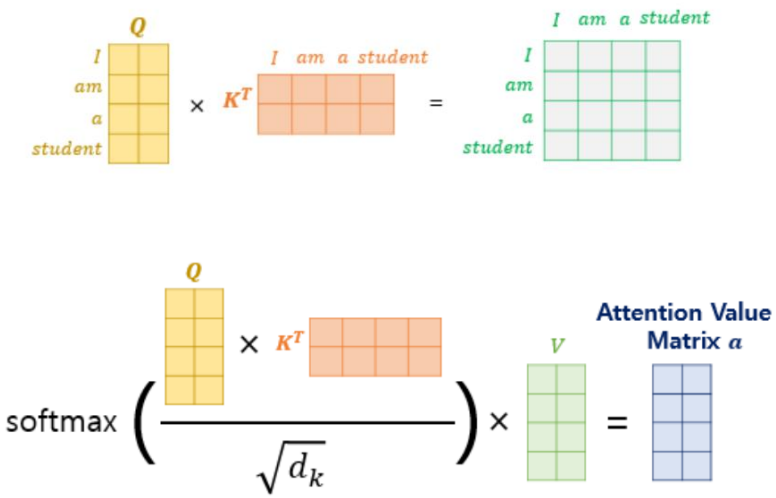
- Attention Value
  - Softmax를 취해 Attention Distribution을 구하여 V 벡터와 Weight sum



Transformer

- 행렬 연산으로 일괄 처리
  - 단어 하나하나 마다 구할 필요 없이 행렬로 연산

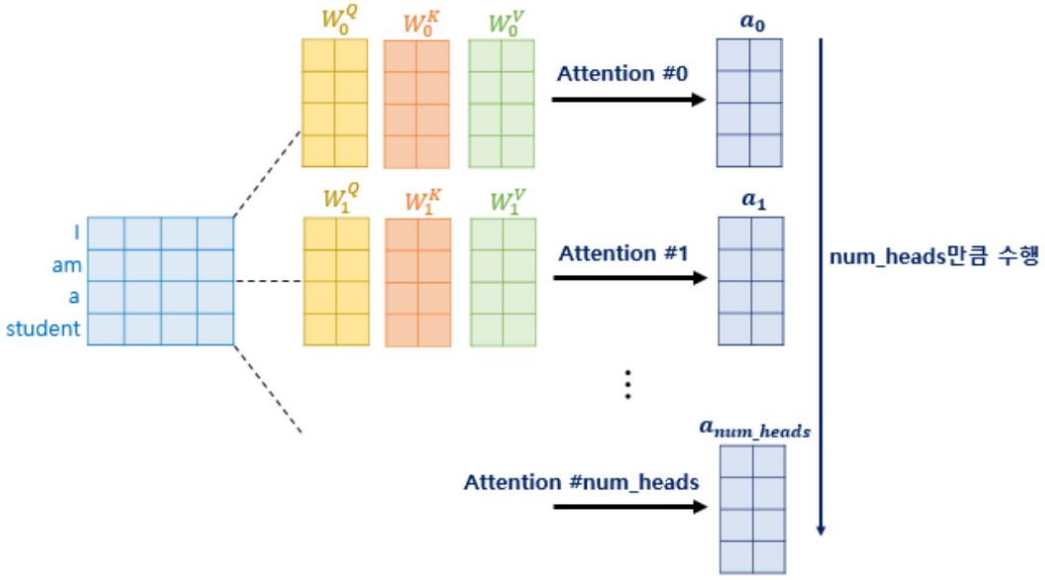




$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

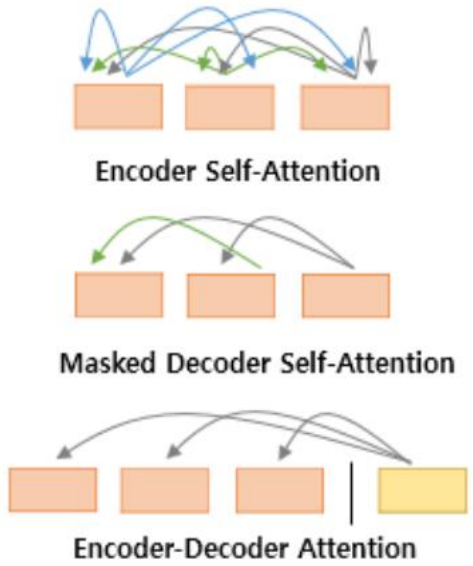
Transformer

- Multi-head Attention
  - 트랜스포머 연구진은 한 번의 어텐션을 하는 것보다 여러 번의 어텐션을 병렬로 사용하는것이 효과적으로 판단
  - 병렬로 수행하여 다른 시각으로 정보를 수집



Transformer

- Attentions



Break

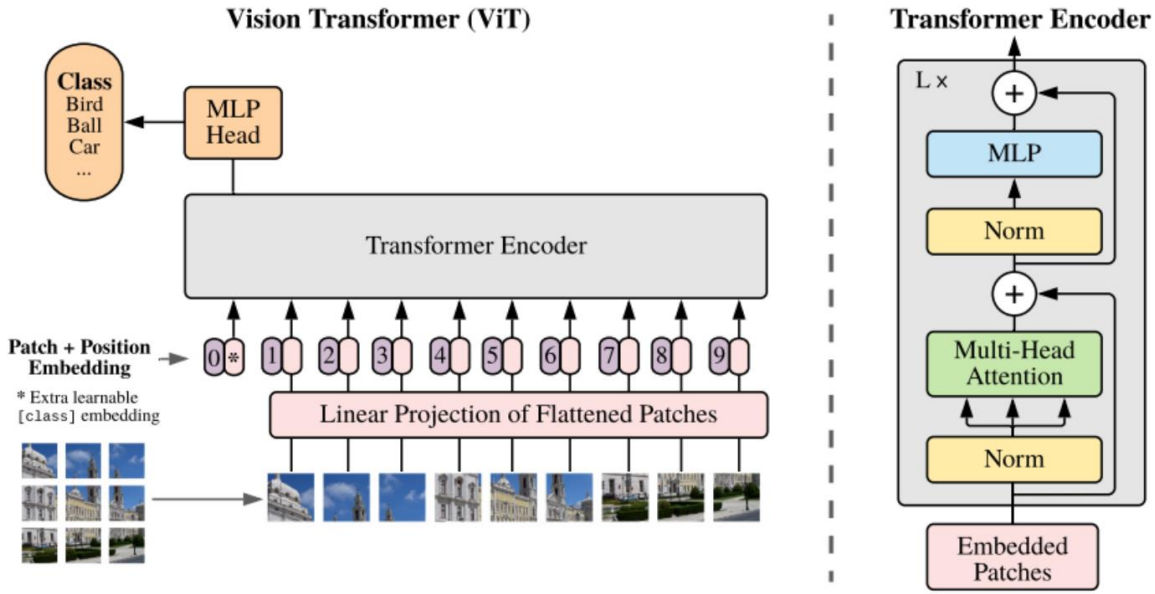
- 이건희 <1942 ~ 2020>
  - “ 일본을 보면 한국을 미래가 보인다 "



CVSP

Vison Transformer

- Vison Transformer(2021) Alexey Dosovitskiy, Google Research
  - 자연어 처리에서 쓰인 트랜스포머 구조를 컴퓨터 비전에 적용
  - CNN 만 이용한 네트워크보다 좋은 성능



$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; \cdots; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{1}$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1 \ldots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \qquad \ell = 1 \ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

Vison Transformer

- REMIND **Non-sequential Data**
  - 이미지는 **Non-sequential Data** 자름으로써 **sequential Data!!!**
    원래 위치(순서)를 알아야 함



Vison Transformer

- VIT mechanism
  - Image Patch
  - Patch Embedding
  - Class token
  - Position Embedding
  - Layer Normalization
  - Mul

Vison Transformer

- Image Patch
  - 트랜스포머는 NLP 분야에서 출발
  - 1D 임베딩 필요, Image 3D
  - [h,w,c] 분해
    - Ex [300,300,3] 9장으로 분해
    - => [100,100,3] x 9



Vison Transformer

- Patch Embedding
  - Patch image를 1D로 변경

$x \in R^{H \times W \times C}$ 원래 이미지 size

$x_p \in R^{N \times (P^2 \times C)}$ Flatten한 뒤, ViT에 입력 : 벡터로 처리하여 입력

$N = \frac{HW}{P^2}$ N : Transformer의 sequence 길이 (패치 수)

P : 패치의 크기, 패치는 정사각형

where 원본 이미지 해상도 : (H,W), 각 이미지 패치 해상도 : (P,P)

Vison Transformer

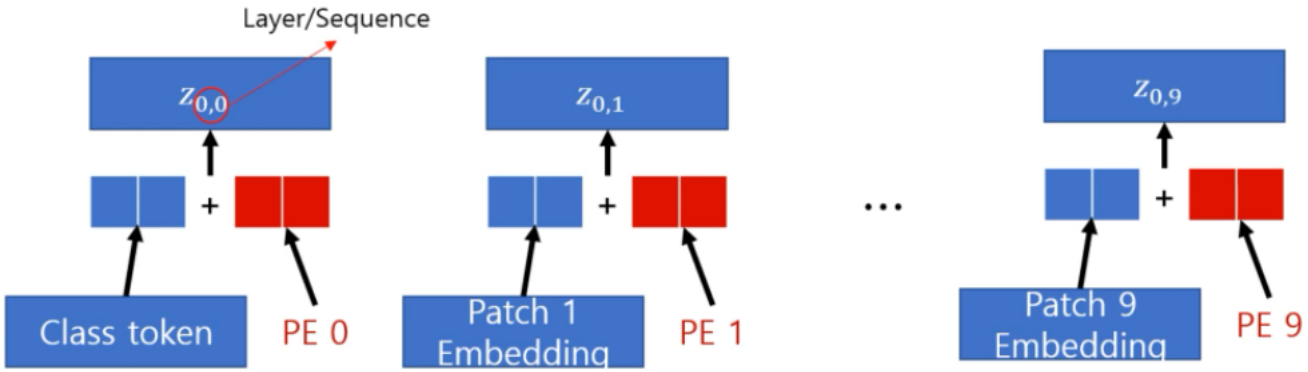
- Class token
  - 임베딩된 patch 의 맨앞에 하나의 학습 가능한 Class token 추가
  - NLP에서 문장의 시작을 나타내기 위해 사용된것 과 유사하며
  - 입력 이미지의 전반적인 정보를 요약하는 역할
  - VIT의 최종 출력으로 나왔을때 이미지에 대한 1차원 representation vector로써의 역할을 수행

- 수식
  - P : Patch size
  - D : Embeding vector의 차원

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \mathbf{x}_p^2\mathbf{E};\ \cdots\ ;\ \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D},\ \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

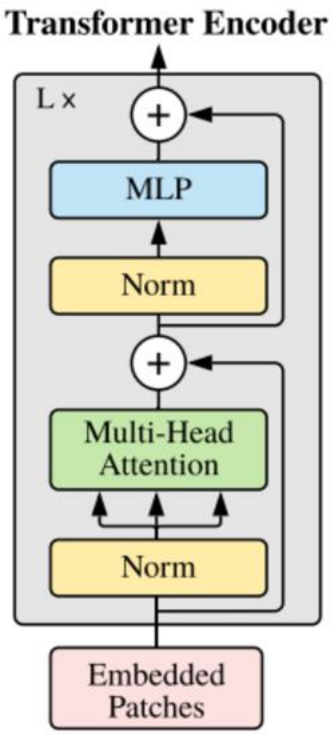
Vison Transformer

- Position Embedding
  - NLP 트랜스포머와 같이 임베딩 벡터에 위치정보를 더함



Vison Transformer

- Transformer Encoder
  - NLP에서의 Transformer 와 똑같이 진행



Vison Transformer

- MLP Classfier
  - CLS 토큰만 MLP를 통과해 classification

