Transformer

- Sequential Model 한계점

  완벽한 구성성분을 가진 특정 데이터가 아니면 학습하기 어려움

  - 문장마다 길이가 다름
  - 문장성분 누락 가능
  - 성분 순서가 바뀔 수 있음
- **Transformer**
  - RNN: 재귀적으로 input 들어감

  attention만 사용, sequence 해석

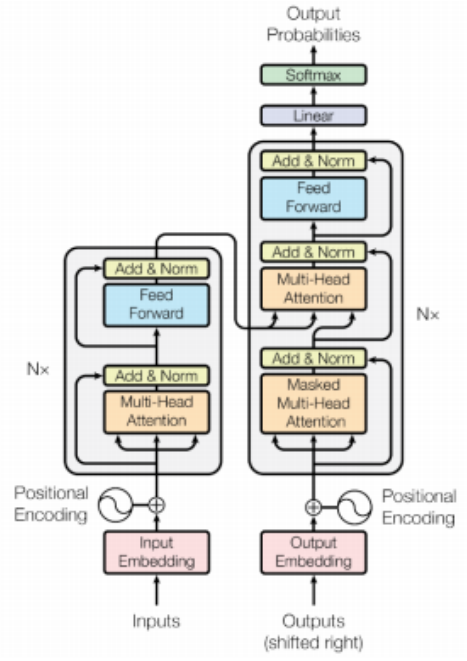
  최초에는 신겨망기계번역(Neural Machine Translation, NMT) 문제에 사용

  시퀀스 데이터 처리하고 인코딩하는 방법 => 여러 분야 사용 가능

  이미지 분류, detection, visual translation 등 활용도 넓음

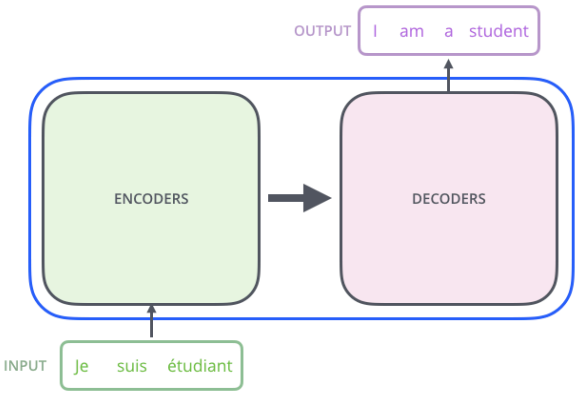
  - RNN: 길이가 긴 sequence 약점

    => transformer: Multi - Head Attention으로 극복

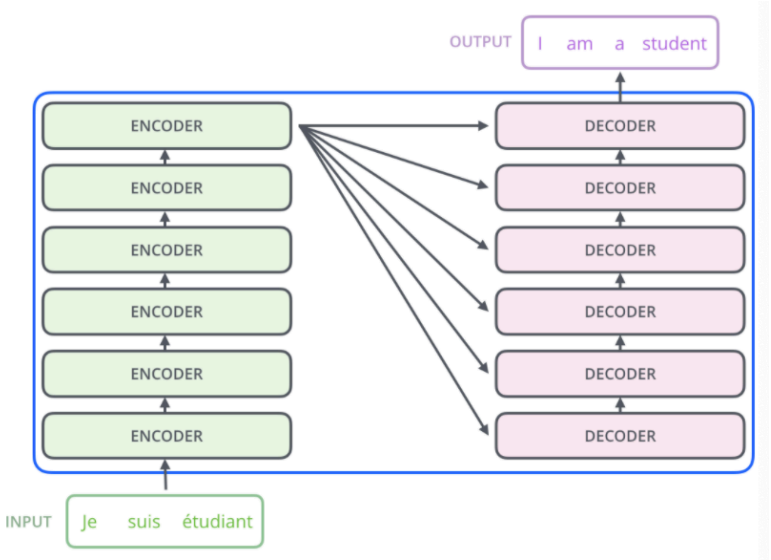


살펴볼 것

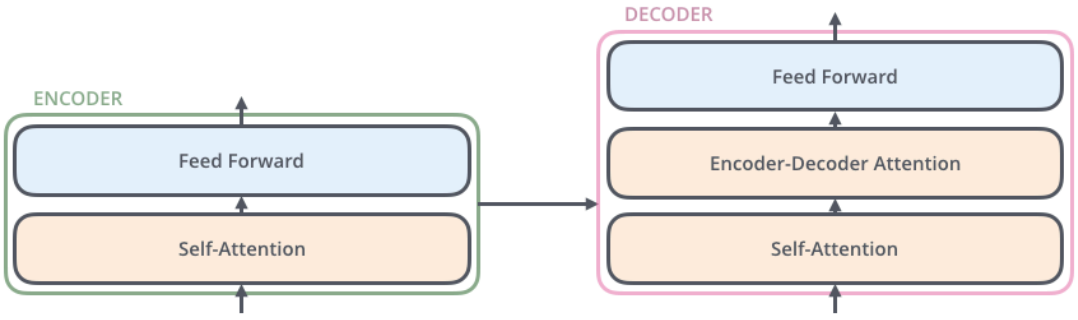
1. **어떻게 N개의 단어를 한 번에 처리하는지**
2. 인코더, 디코더 간에 어떤 정보를 주고받는지
3. 디코더가 어떻게 generation 할 수 있는지



- **encoding**: 여러 개의 encoding 쌓아 올려 만든 것
- **decoding**: encoding 부분과 동일한 개수만큼의 decoder 쌓는 것



- encoder 모두 정확히 똑같은 구조, but 같은 weight 공유 X
- encoder 나누면, 2개의 sub-layer로 구성



- **Encoder**

  모든 단어 벡터를 한번에 입력으로 받음, 출력값을 바로 다음 인코더로 전달

- **Self-Attention**

  Attention 매커니즘의 핵심

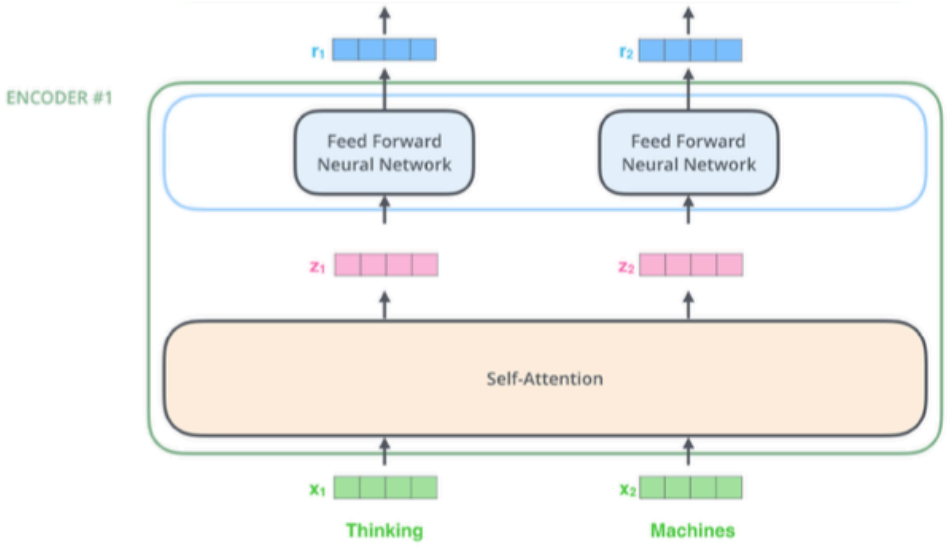
  encoder가 특정한 단어를 encode하기 위해서, 탄정 단어가 아닌 입력 내의 **모든 다른 단어들과 관계를 살펴봄**

  => output을 내보낼 때, 모든 단어들과의 dependancy 생김

- **Feed Forward Neural Network**

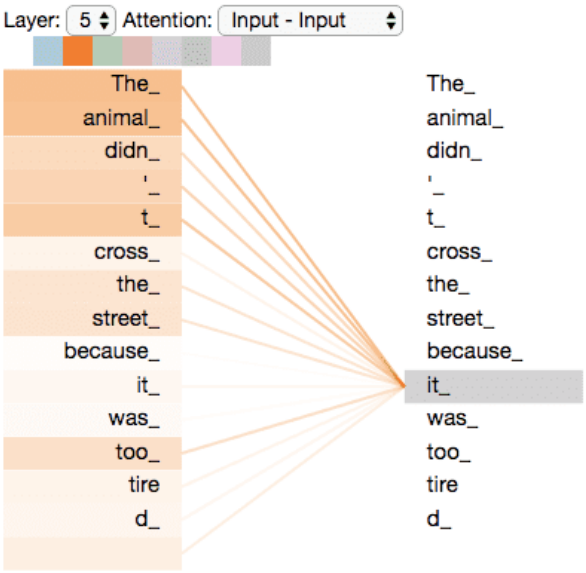
각 위치의 단어마다 독립적으로 적용되어 출력 만듦 => MLP와 별 다른 차이 없음

input된 단어들에 대해 독립적/병렬적 수행 => dependacy X



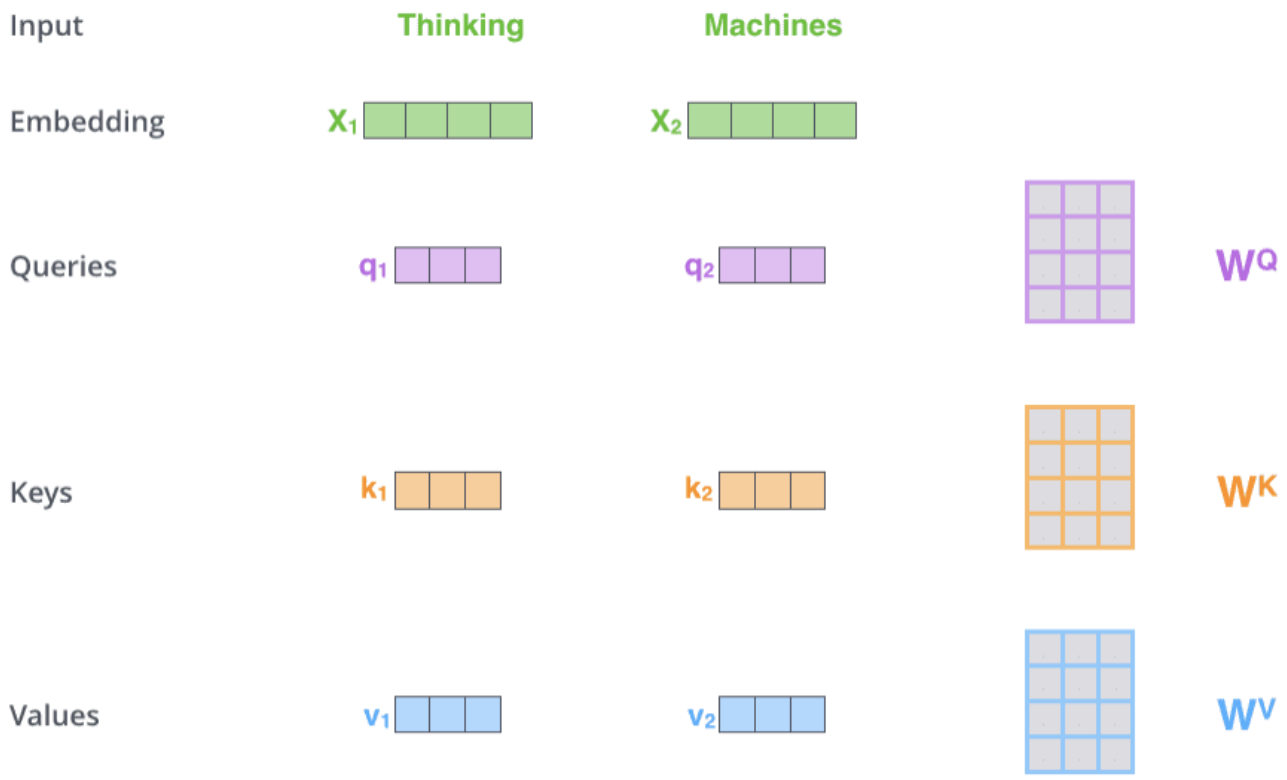
- ex) The animal didn't cross the street beacause it was too tired.

  it이 의미하는 것은?



Self Attention 사용 시, **문장 내에서 it에 대응하는 단어들을 모두 고려**, 가능성이 가장 높은 'animal' 학습

=> 기계가 문장 내부에서 단어의 의미를 비교적 더 잘 이해할 수 있음

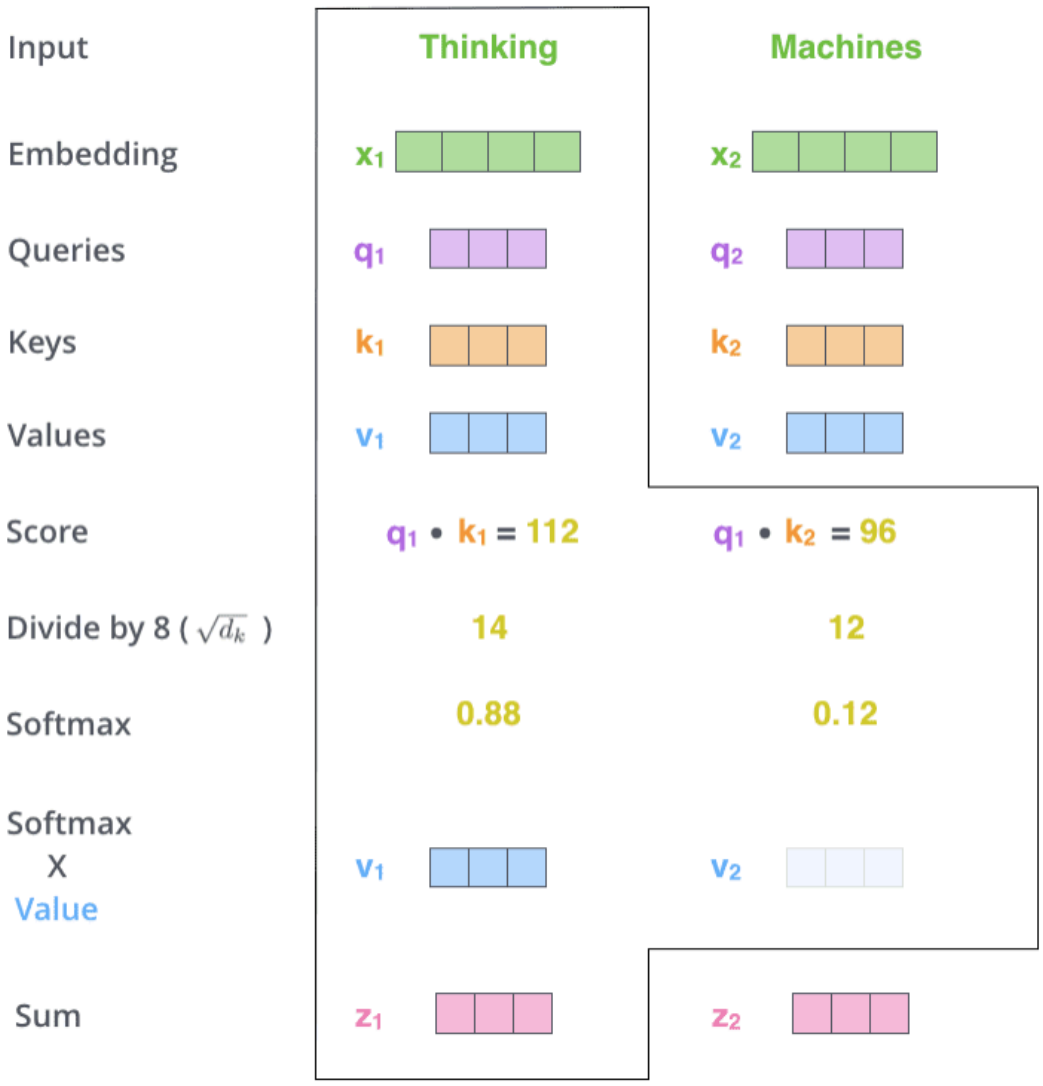


- Self Attention: 3개의 신경망처럼 작동

  하나의 입력(단어 임베딩(Word Embedding))이 주어질 때마다, **각 신경망을 통해 3개의 벡터 만듦**

  => Queary, Key, Value

  3개의 벡터를 이용해 입력받은 단어 X를 새로운 벡터로 바꿔줌



- 단어 주어지면 => 해당 단어에 대한 Query 벡터, 나머지 모든 단어에 대한 key 벡터 구하고, 내적해서 Score 백터 구함
- **Score**: i번째 단어가 다른 단어와 얼마나 관계(유사도)가 있는지 계산하는 역할

  => **해당 단어와 나머지 단어 사이에 얼머나 interaction이 일어나야하는지 학습 가능**

  => 어떤 단어를 더 주의(attention) 깊게 살필지 결정하는 것 => attention 이름 붙여짐
- 계산된 Score 벡터를 normalize함
  - 나눠주는 '8'은 key 벡터의 dimension과 관련 있음

    key 벡터를 몇 차원(d_k)으로 만드는가를 직접 정해주는 하이퍼파라미터

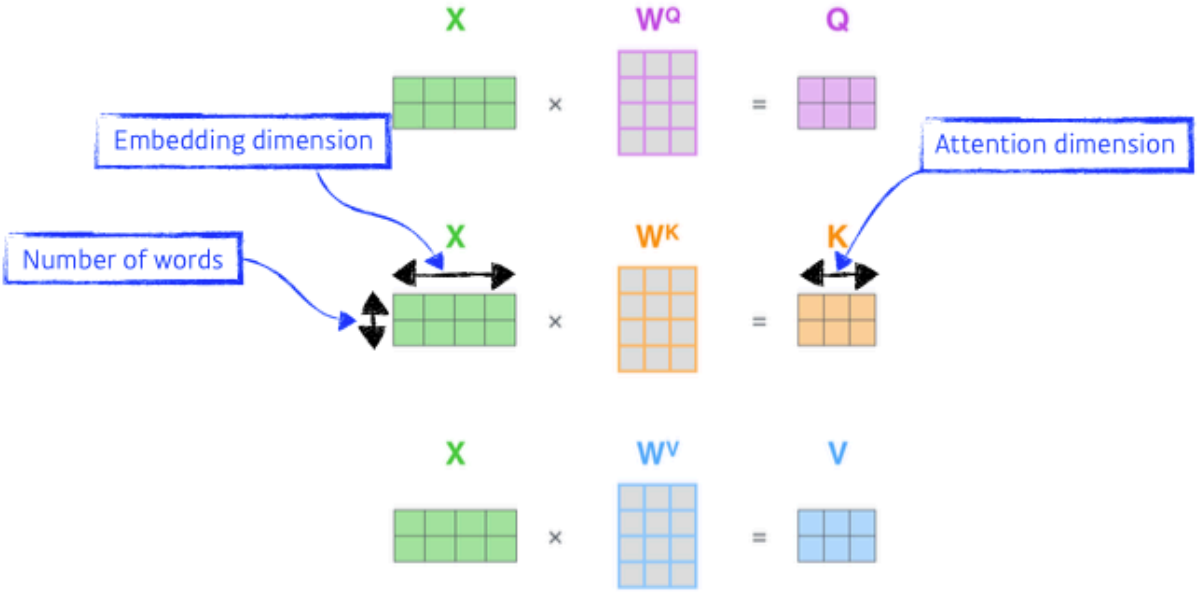
    Normalize 시, $\sqrt{d_k}$를 나눠줘 score의 range 조정
- Normalize된 Score을 Softmax 취해 attention rate으로 바꿔줌

  => **attention rate와 각각 단어에서 나오는 Value 벡터들의 weighted sum(가중합)**이 최종적으로 해당 단어의 **인코딩한 벡터(encoded vector)**가 됨
- 유의할 점
  - Score 벡터 구하기 위해, Query 벡터와 Key 벡터를 내적

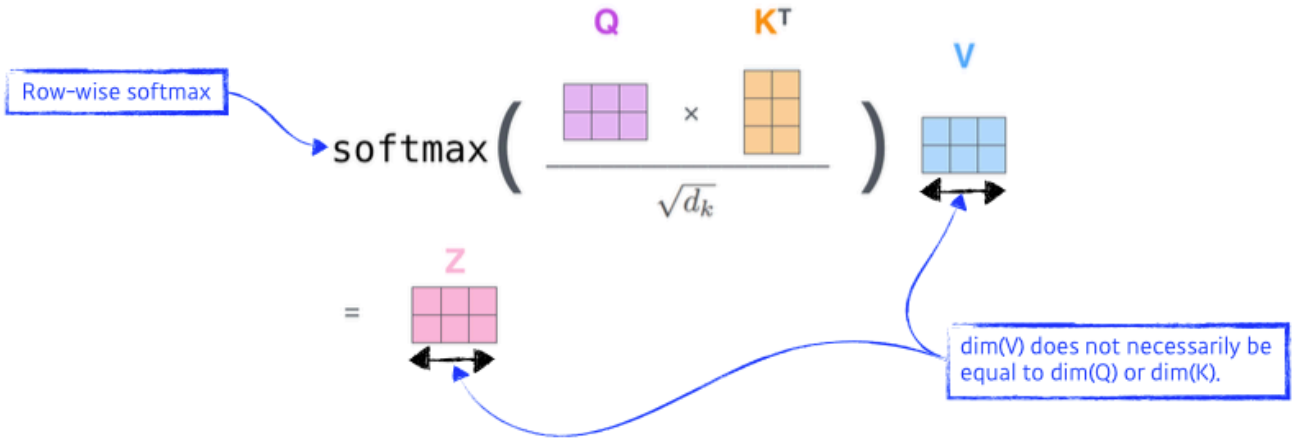
    => **Query, Key 벡터 차원 같아야함**
  - Value 벡터는 attention rate와 가중합 계산만 하면 됨

    => **Value 벡터 차원은 달라도 상관 없음**

    - 최종적으로 산출되는 특정 단어의 **인코딩된 벡터는 Value 벡터 차원과 동일**
- 행렬로 표현



    - X의 각 row: 하나의 단어 / column: 단어의 임베딩
    - Q: Query 벡터 / K: Key 벡터 / V: Value 벡터



- matrix 간의 계산

  Softmax는 Row-wise하게 적용

  $di(V)$(사진 속에서는 3)와 $dim(Z)$가 같고, $dim(Q)$와 $dim(K)$와는 다를 수 있다
- 이미지 주어짐, MLP, CNN 넣으면 입력 fix되어 있으므로 출력 고정

  but, Transformer는 네트워크가 fix되어있어도, 인코딩하려는 단어와 그 옆의 단어들에 따라 인코딩 결과값 달라짐

  => **MLP보다 더 유연한 모델**
    - **입력된 단어에 따라 출력 고정 X, 옆에 주어진 다른 입력(단어)들 모두 고려, 출력 변화할 여지 존재**

  더 많은 것 고려, 더 많은 연산 필요(n^2)

  ex) 1000개의 단어 주어짐 => 한번에 1000^2의 입력 처리해야함

  but, RNN은 1000개의 단어 주어짐 => 1000번 연산 수행하면 됨

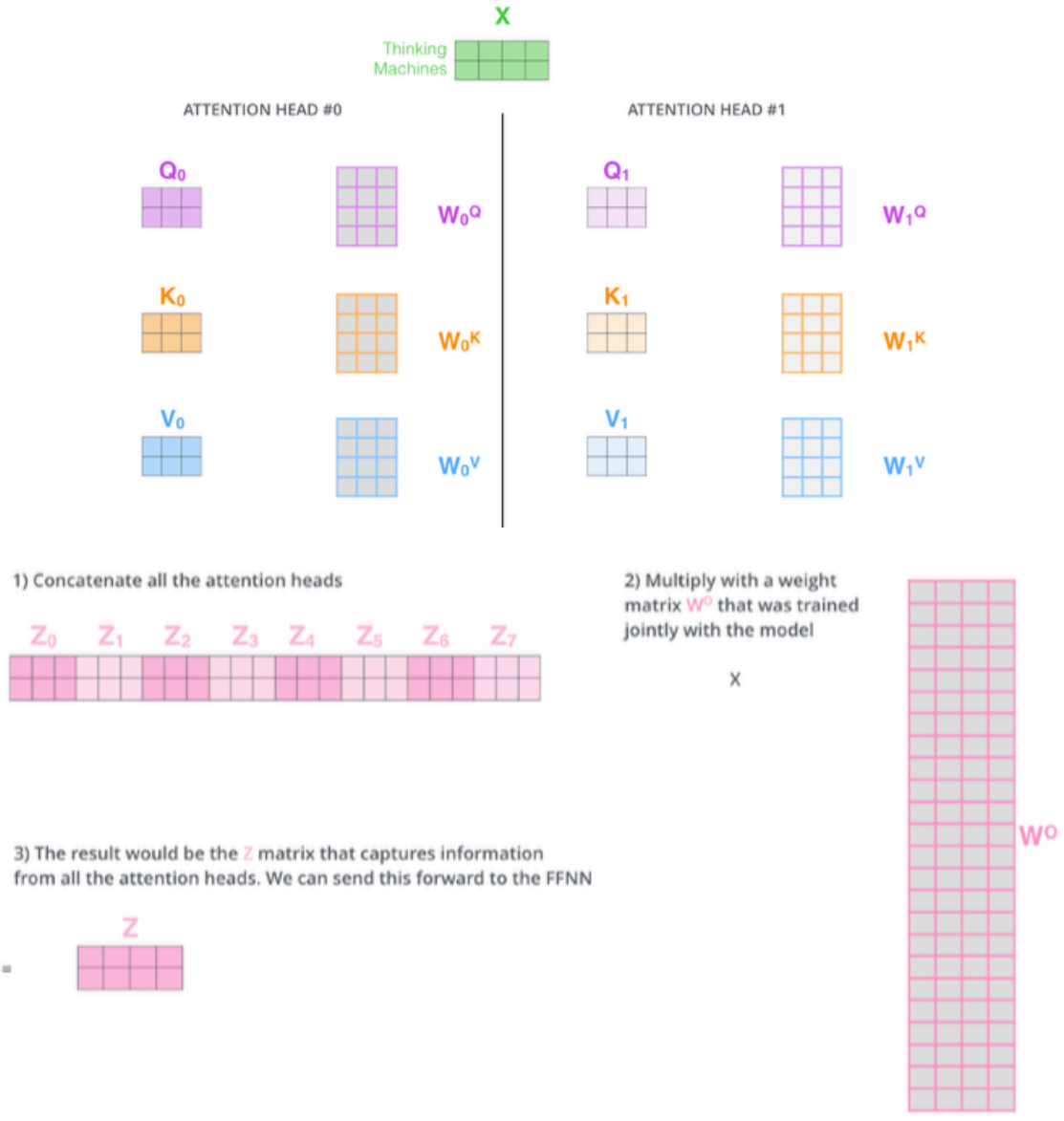
  => **너무 긴 입력이 주어지면 메모리 부족으로 한 번에 계산 X 한계 가짐**
- **기존의 RNN에 비해 훨씬 더 많은 문맥정보 파악 가능**

- **연산 자원이 많이 듬(n^2)**
- **Multi-headed attention(MHA)**

  attention 과정 여러번 수행

  하나의 임베딩 벡터(입력)에 대해 벡터 셋(Query, Key, Value)을 여러 개 만듦

  => multi-headed(머리 여러 개)



  - MHA로 n개의 헤드 사용 => n개의 인코딩된 벡터 나옴

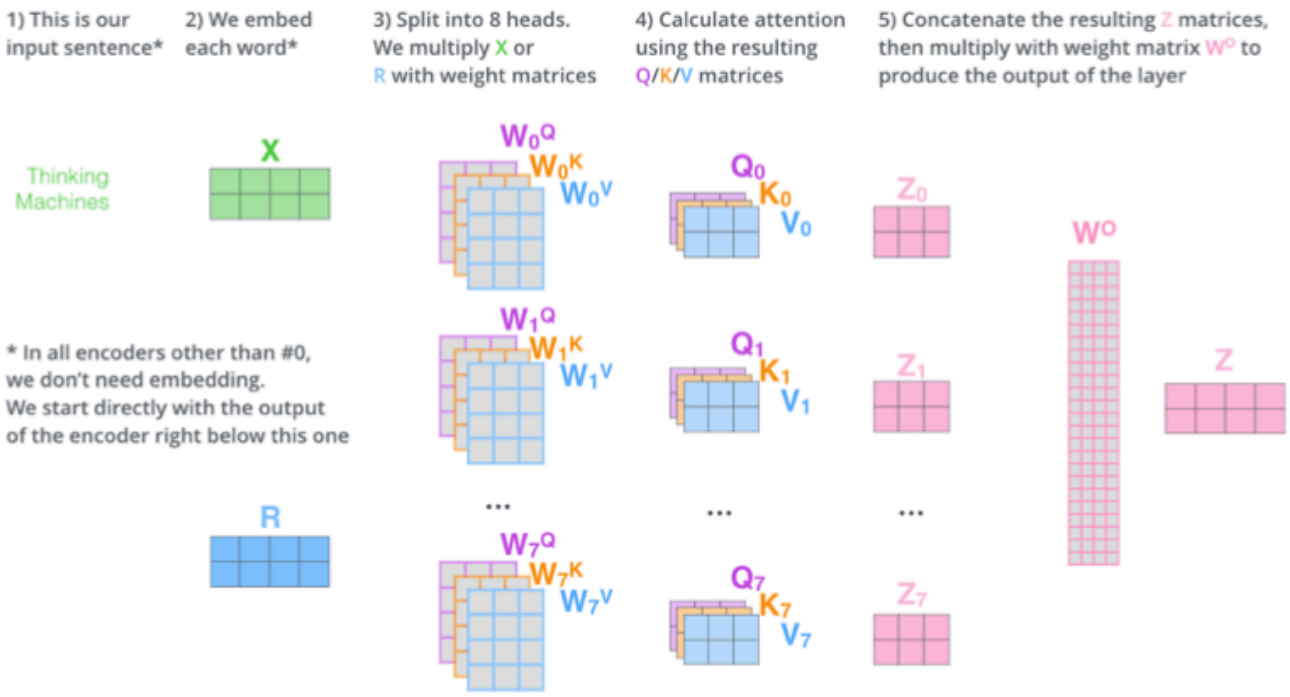
    => 값을 conatenate하여 다음 인코더로 넘겨줌

    - 이 때, 여러 번 인코더 탐 => 인코딩된 벡터(출력값)의 차원을 인코딩하기 전의 입력값 차원과 동일하게 맞출 필요 있음
    - MHA된, 즉 concatenate된 출력 벡터: 기존의 입력값에 비해 n배의 차원 됨

      => 적절한 행렬 W^o와의 행렬곱을 통해 기존의 크기 변환

  - **실제 구현은 위처럼 방식 X**

    원래 주어진 임베딩 단어 차원: 100차원, 10개의 head 사용 => 10개로 나눔, 즉 10차원 입력만을 가지고 Query, Key, Value 벡터 만듦

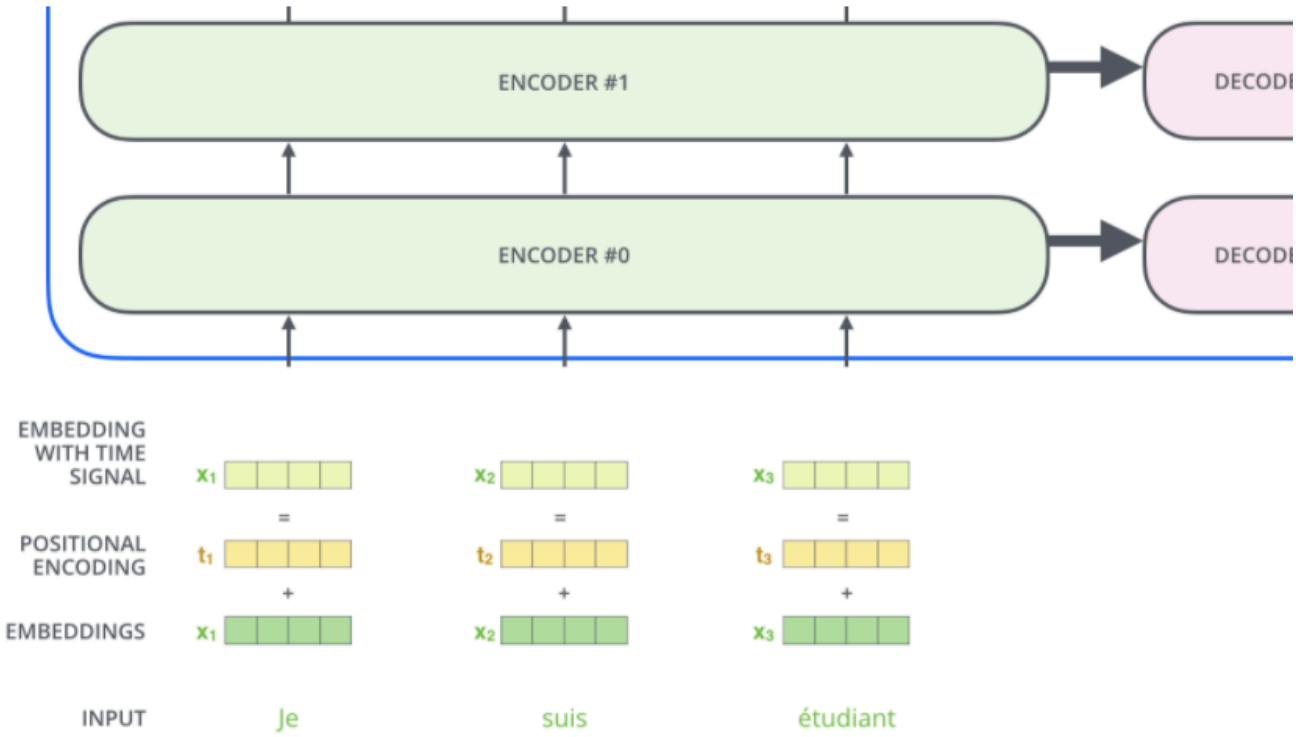


- **Positional Encoding**

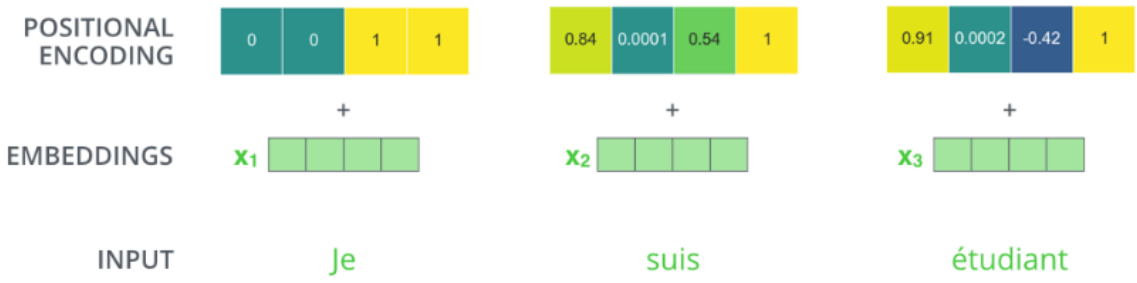
  sequential 정보 포함 X 때문에

  순서에 독립적(order-independent), But 실제 문장에서 순서 중요

  => 주어진 입력에 대해 위치 별로 특정한 패턴을 따르는 positional encoding 벡터 추가

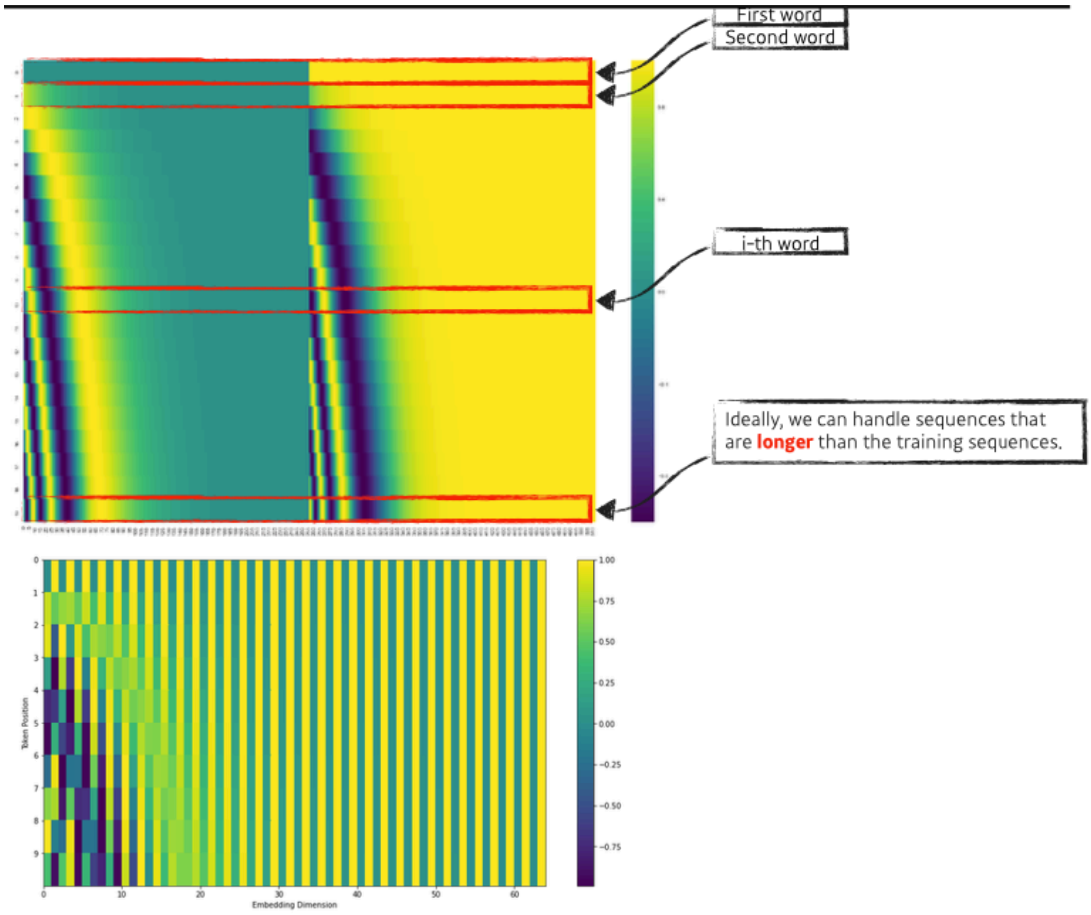


  - ex) 크기 4인 embedding의 Positional encdoing 예시



- 셀들의 값에 대해 색깔 다르게 나타냄

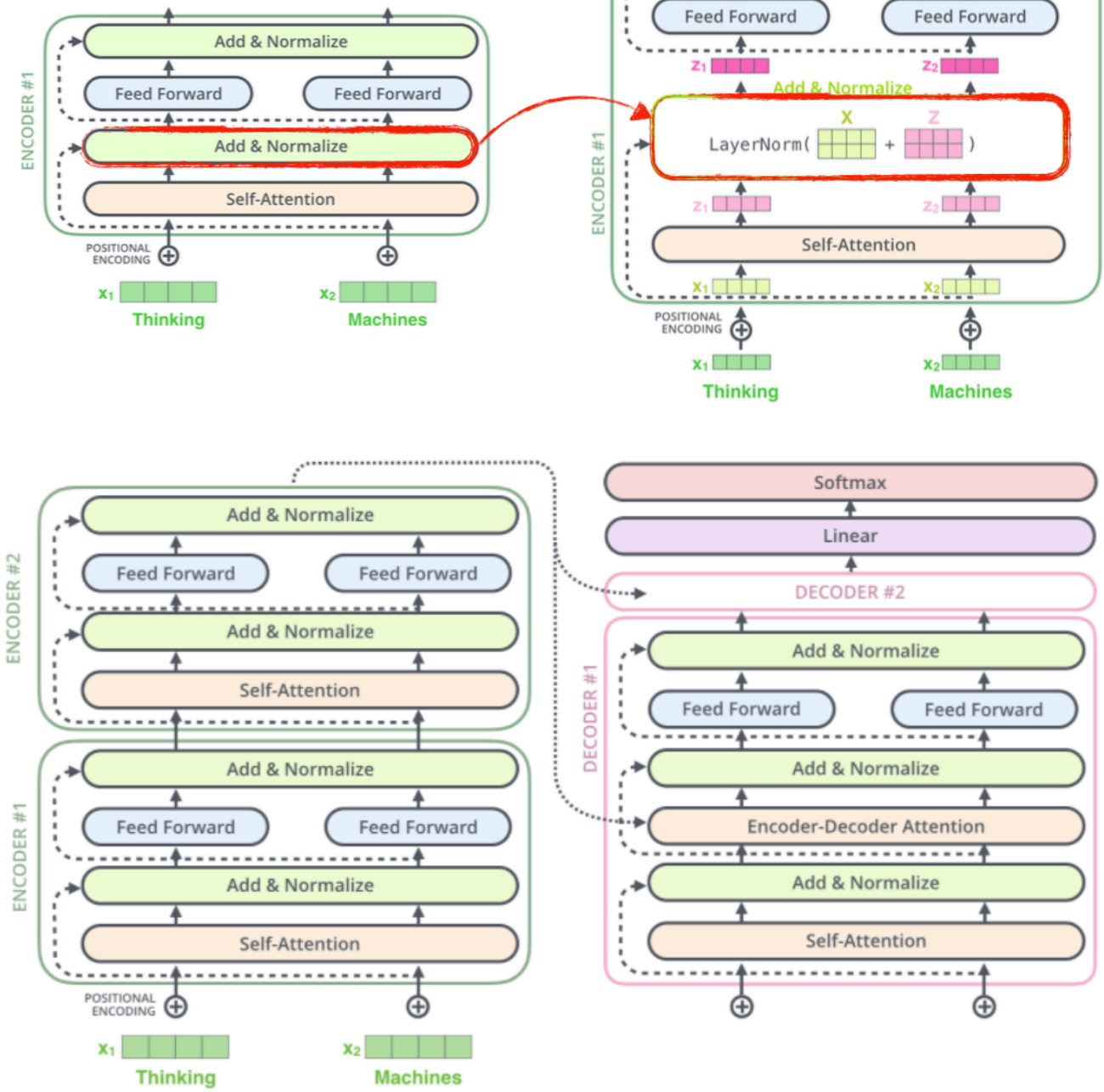
  => positional encdoing 벡터들이 가지는 패턴 시각화



- **The Residuals**

  enconder 내의 sub-layer가 residual connection으로 연결

  그 후에는 layer-normalization 과정 거침



- **Decoder**

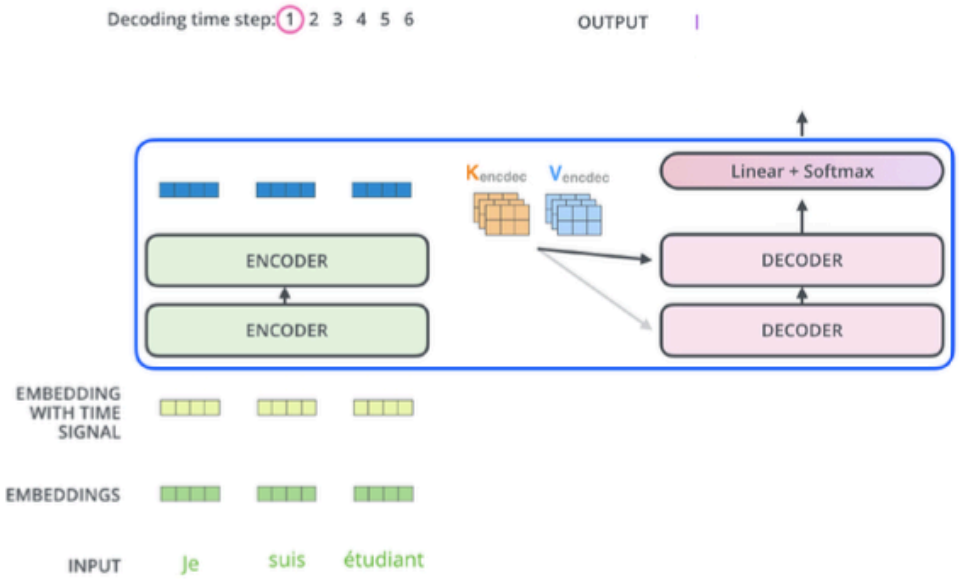
  input 값들을 출력하고자 하는 단어들에 매칭하는 attention map을 만드려면, key 벡터와 value 벡터 필요

  마지막 인코더 층(가장 상위 인코더) 출력값은 여러 다나어들의 [Key, Value] 벡터 시리즈로 제공

  => 각 Decoder 내부의 Encoder-Decoder attention 계층에서 해석하여 입력 시퀀스의 적절한 위치 잡아줌

  - MHA와 동일한 방식으로 동작

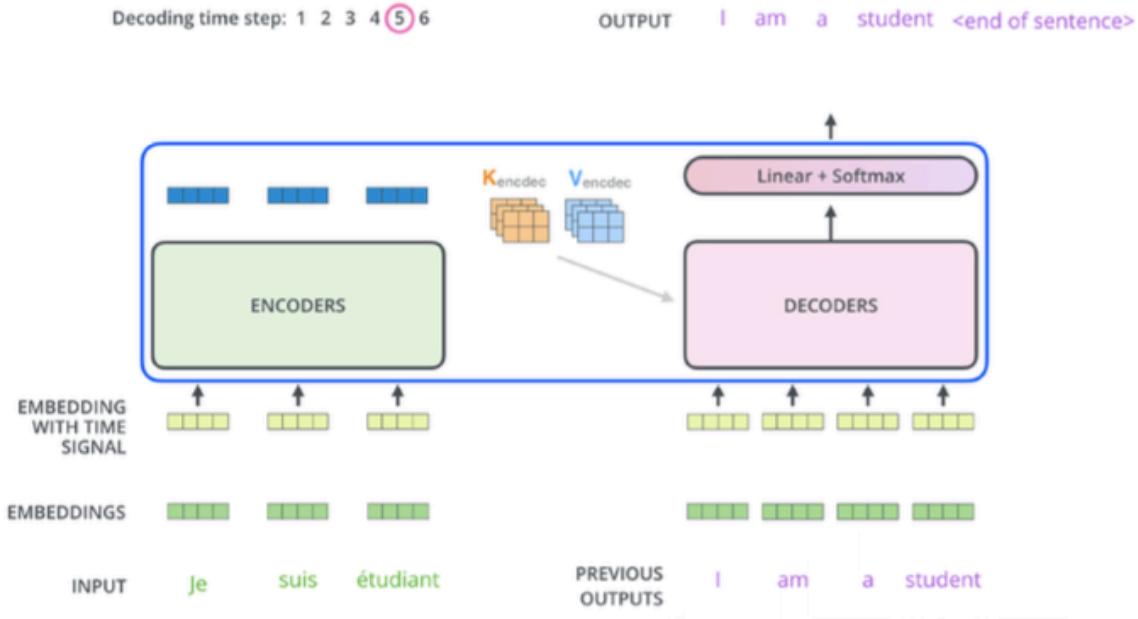
    Query 벡터를 이전 Decoder에서 받아오고 Key, Value 벡터를 인코더 스택에서 받아오는 차이점



- 디코더 마지막 층: **디코더 스택의 출력물을 단어들의 분포로 만들어 내보냄**

  매번 샘플링하는 형식으로 모델 출력 만들어짐

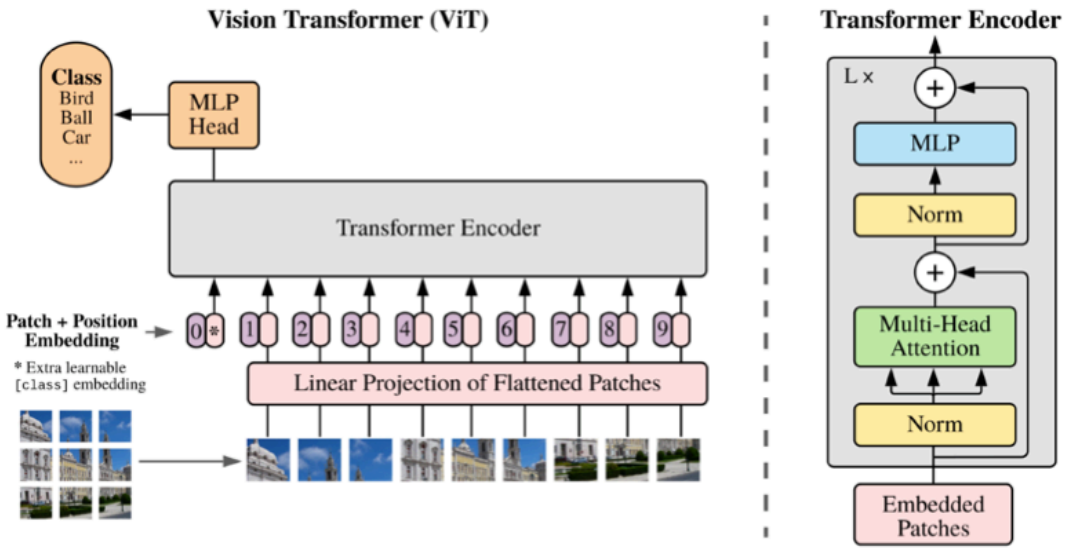
  모든 출력 끝 => EOS(End of Setence) 토큰 나오면서 종료



- **Transformer의 활용**

  시퀀스 데이터에서 머무지 않고, 이미지 등 다양한 분양에서 활용

  - **Vision Transformer**


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches
Transformer Encoder
L x
MLP
Norm
Multi-Head Attention
Norm
Embedded Patches

- **DALL-E**