x

K-Digital Training 스마트 팩토리 3기

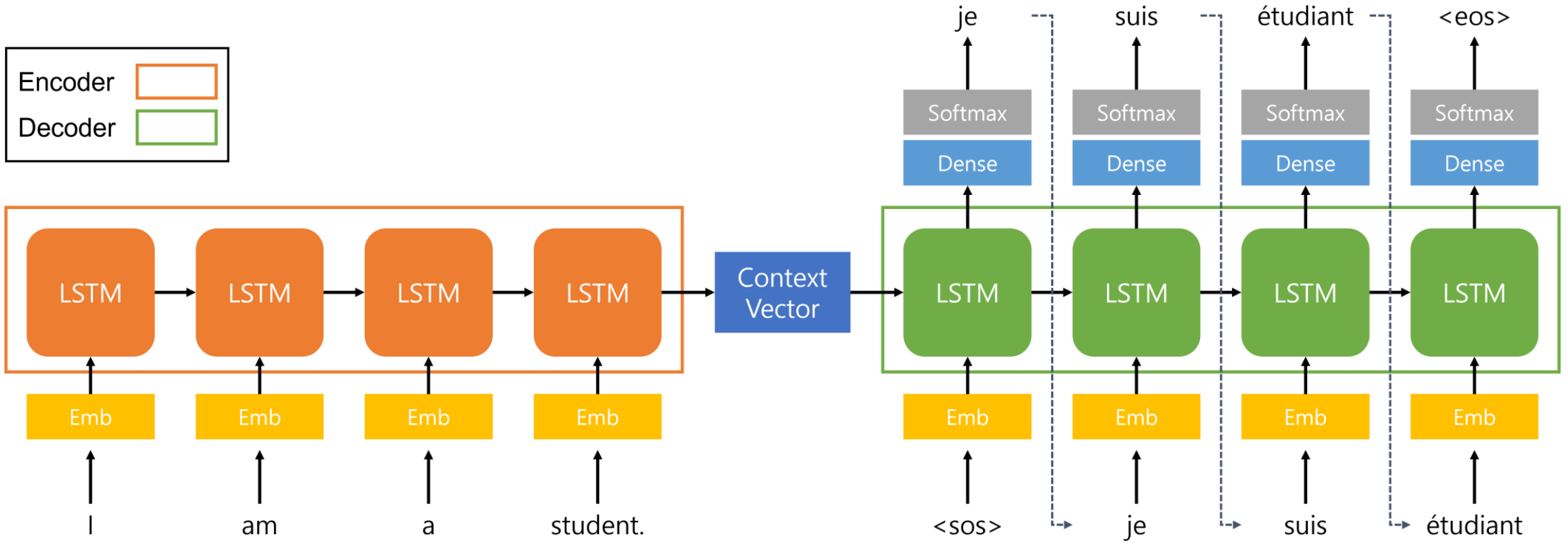
Transformer

Transformer 모델

- 문장 속 단어와 같은 순차 데이터 내의 관계를 추적해 맥락과 의미를 학습하는 신경망
- 구글(Google)의 2017년 논문(Attention Is All You Need)에 처음 등장
- BERT, GPT 등 최근 널리 사용되는 언어 모델의 중추
- 자연어 처리를 위해 개발되었지만, 컴퓨터 비전, 약물 발견 등 전 산업에서 활용되고 있음
- CNN과 RNN을 대체
- 라벨링을 없애고 병렬 프로세싱을 가능하게 한다.

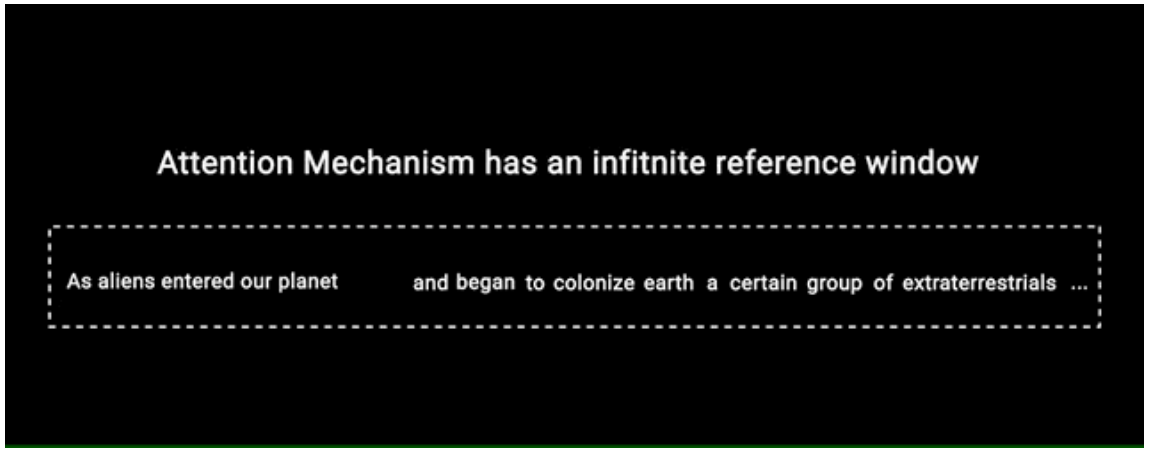
기존 모델

- 인코더는 문장을 하나의 문맥 백터(Context vector)로 저장
- 디코더는 전달받은 문맥 벡터로 초기화
- 매시점에 직전 시점에 출력했던 단어를 입력받아 <eos> 토큰이 나올때까지 수행



기존 모델

- 단점
  - 병렬화 문제
    - 순차적으로 입력을 처리해야 해서 병렬화 불가
    - 대규모 데이터셋의 경우 학습 시간이 매우 길어짐
  - Long Distance Dependency 문제
    - 참조 윈도우의 크기가 고정되어 있었기 때문에 멀리 떨어진 항목들 간의 관계성은 Gradient Vanishing/Exploding 문제로 학습이 되지 않음

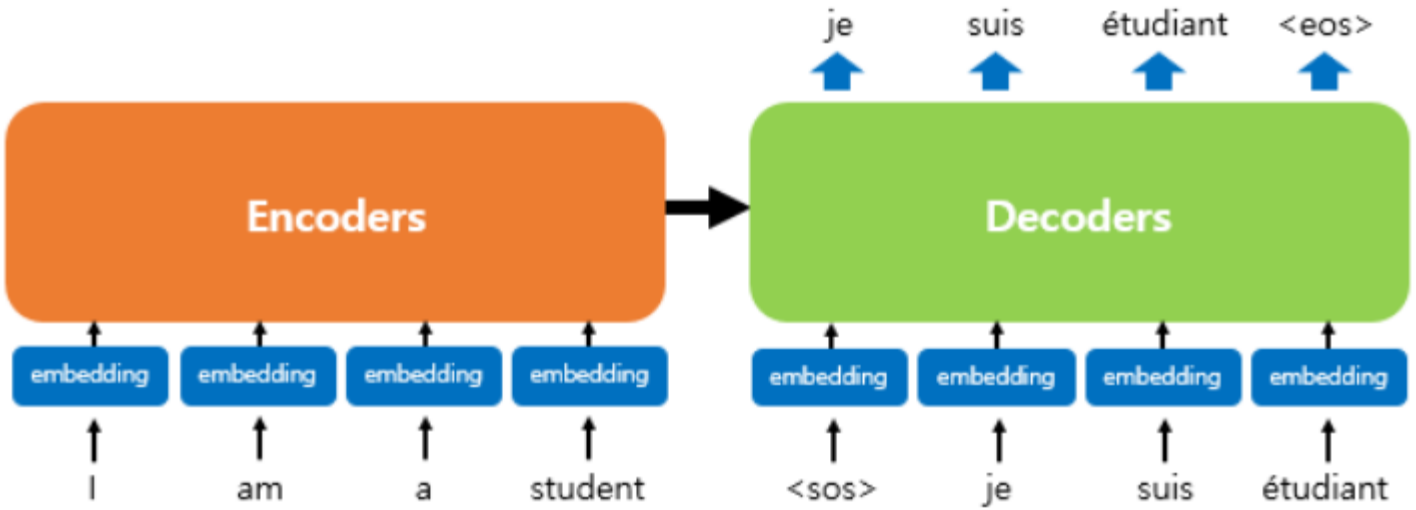


Transformer 모델

- 주요 하이퍼파라미터
  - d_{model} = 512
    - 인코더와 디코더에서의 정해진 입출력 크기
    - 임베딩 벡터의 차원
  - num_layers = 6
    - 인코더와 디코더의 층수
  - num_heads = 8
    - 병렬 어텐션의 개수
  - d_{ff} = 2048
    - 피드 포워드 신경망의 은닉층의 크기

Transformer 모델

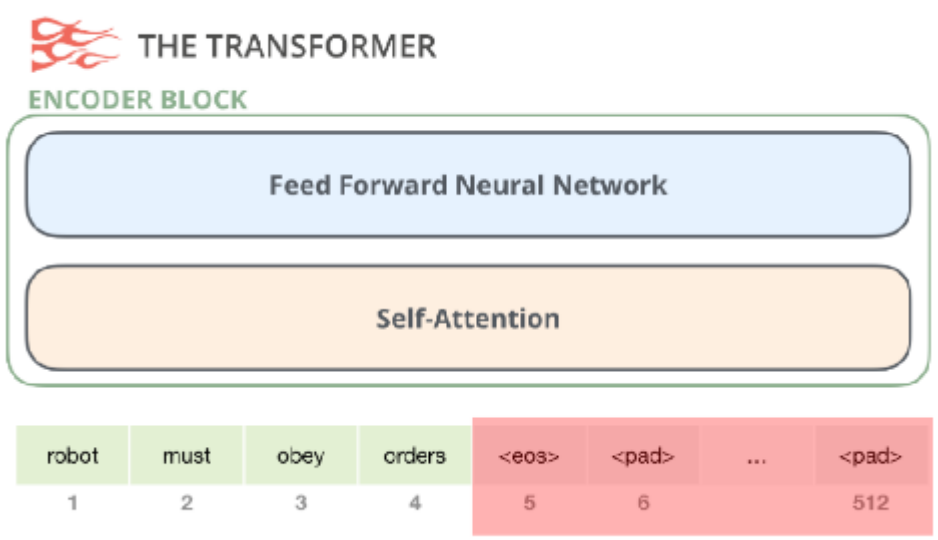
- RNN을 사용하지 않지만 시퀀스 모델처럼 인코더-디코더 구조를 유지. 인코더와 디코더 라는 단위가 N개 존재할 수 있다는 점이 다른 점.
- 시작 심볼 <sos>를 입력 받아서 종료 심볼 <eos> 가 나올 때까지 연산

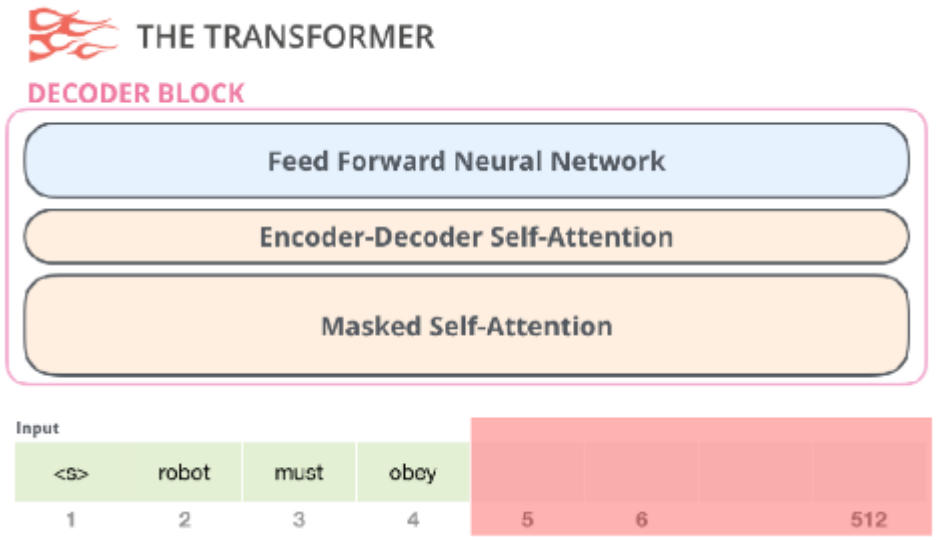


출처: 딥 러닝을 이용한 자연어 처리 입문

Transformer 모델

- 인코딩 블록 vs 디코딩 블록 = Unmasked vs Masked
- 디코딩시에는 앞단어와 연관지어 생성하기 위해서 뒤의 단어를 Masking

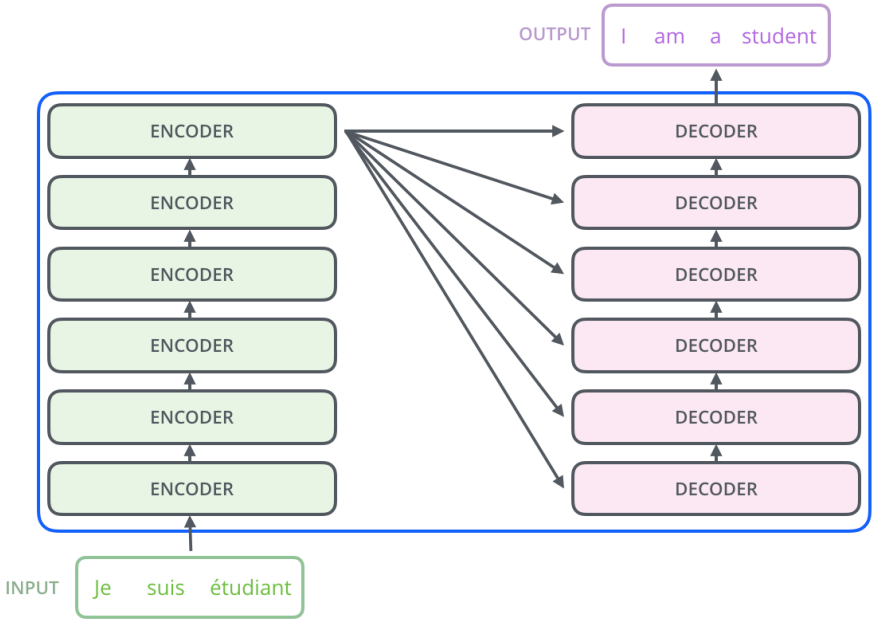




출처:https://github.com/pilsung-kang/Text-Analytics/blob/master/08%20Seq2Seq%20Learning%20and%20Pre-trained%20Models/08-2_Transformer.pdf

Transformer 모델

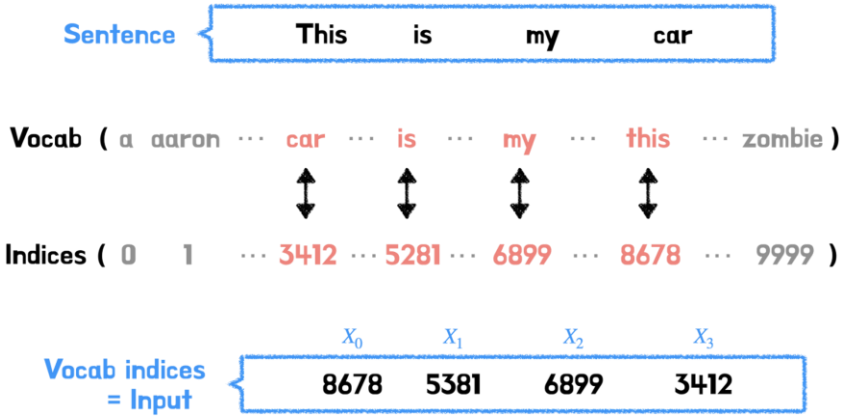
- Encoder
  - 동일한 구조의 인코더를 쌓아서 만듦(각각의 인코더블록의 weight가 공유되지는 않는다)
  - 단어소 각각에 대해서 self-attention과 Fast Forward NN을 거친다(병렬로)



출처:http://jalammar.github.io/illustrated-transformer/

Transformer 모델

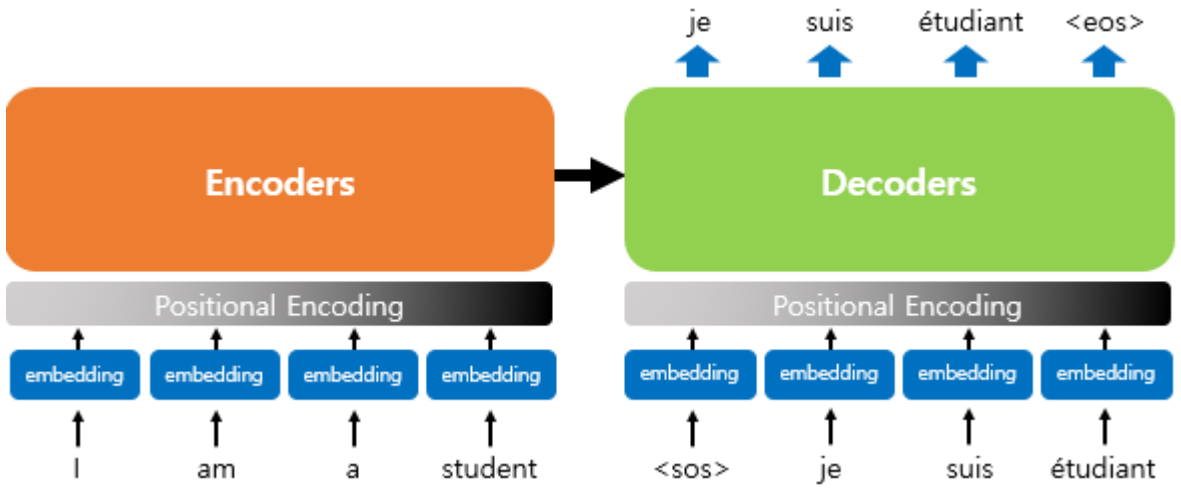
- 임베딩(embedding)
  - 최초 Encoder의 입력시 사용(논문에서는 512차원)
  - 나머지 인코더들의 입력은 이전 인코더의 결과 값 사용
  - 문장을 구성하는 각각의 단어는 그에 상응하는 인덱스 값에 매칭이 되고, 이 인덱스 값들은 Input Embedding에 전달된다.
  - 공유하는 feature값이 존재하고 문맥상 유사도가 높다면, 임베딩 벡터값은 점점 가까워 진다.

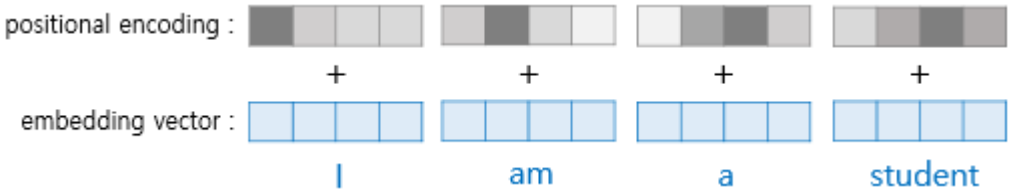


Transformer 모델

- 포지셔널 인코딩(Positional Encoding)
  - RNN이 자연어 처리에서 유용했던 이유는 각 단어의 위치정보를 가질 수 있기 때문
  - “I am a student” 의 경우 student라는 명사는 be동사 뒤에 옴
- 트랜스포머는 단어 입력을 순차적으로 받지 않고 병렬로 받기 때문에 단어의 위치정보를 알려줘야 함
- 각 단어에 임베딩 벡터에 위치 정보들을 더해 모델의 입력으로 사용하는데, 이를 포지셔널 인코딩이라고 함

Transformer 모델





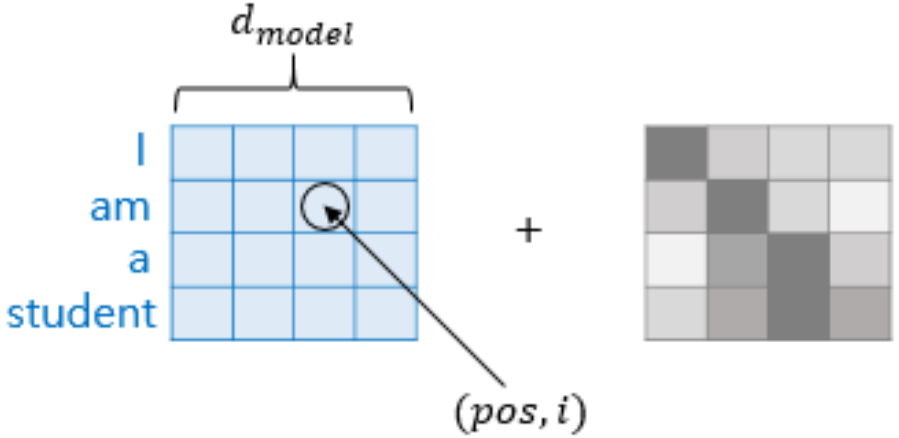
출처: 딥 러닝을 이용한 자연어 처리 입문

Transformer 모델

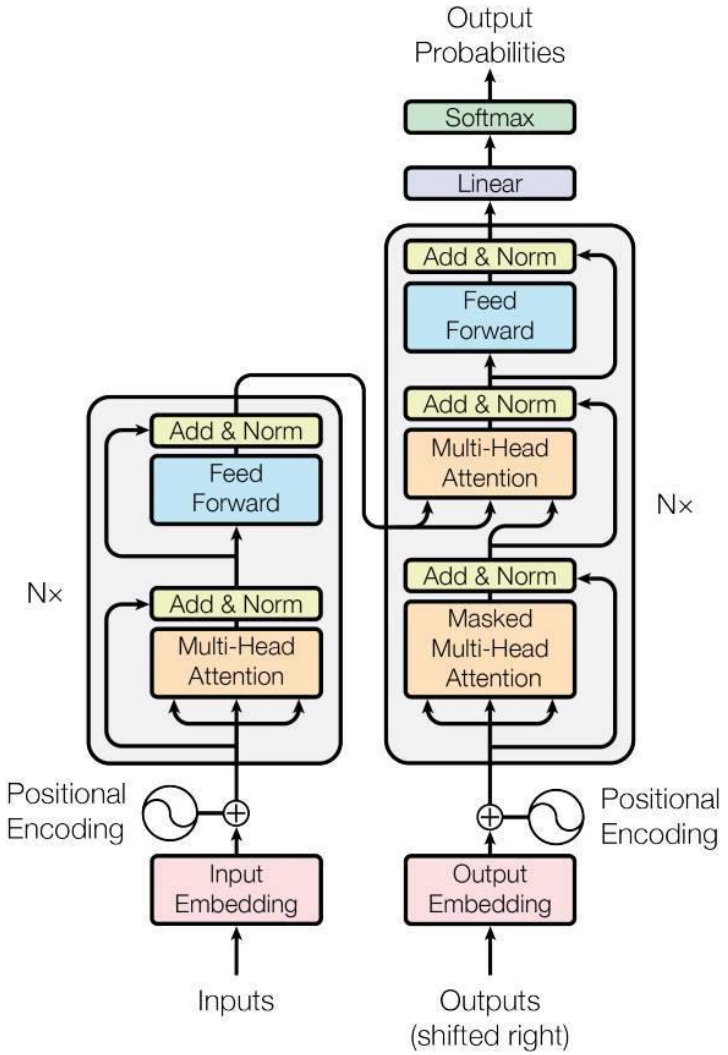
- 포지셔널 인코딩(Positional Encoding)
  - 논문에서 제시한 값

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

    - d_{model} : 모델의 차원
    - pos : 임베딩 벡터의 위치
    - i: 임베딩 벡터내 차원의 인덱스
  - 이외에도 많은 포지셔널 인코딩 값들이 쓰이고 있음

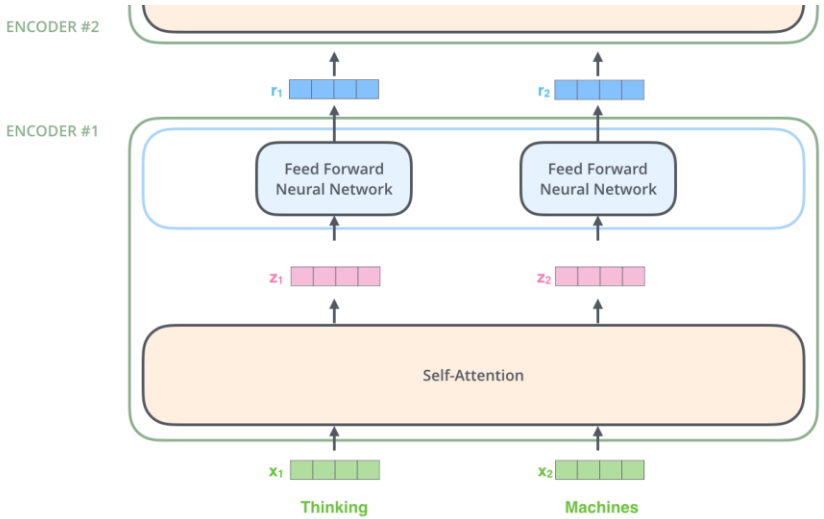


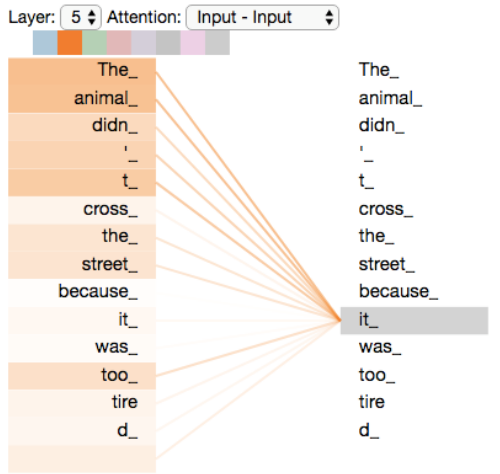
Transformer 모델



Transformer 모델

- Self-attention
  - Self-attention layer 끼리는 dependency 가 있다.
  - Feed forward 사이에는 dependency가 없다. (같은 block내에서 W는 공유)
  - 아래 그림에서 'It' 이라는 단어가 어떤 단어와 연관되어 있는지를 알아야 한다.
  - 같은 문장 내에서, 즉 입력으로 들어온 시퀀스 안에서 단어들 간의 관계를 고려하는 것





Transformer 모델

- Self-attention
  - 각각 input vectors에 대해서 세가지 벡터를 만든다
  - **Query**
    - 현재 보고 있는 단어(token)의 표현, 다른 단어에 score를 매기기 위한 기준
    - 현재 프로세싱 중인 단어만 신경 씀
  - **Key**
    - Label과 같은 역할
    - Query와 비교하는데 사용되는 벡터
  - **Value**
    - 실제 값
    - 가중치를 구하는데 사용되는 벡터

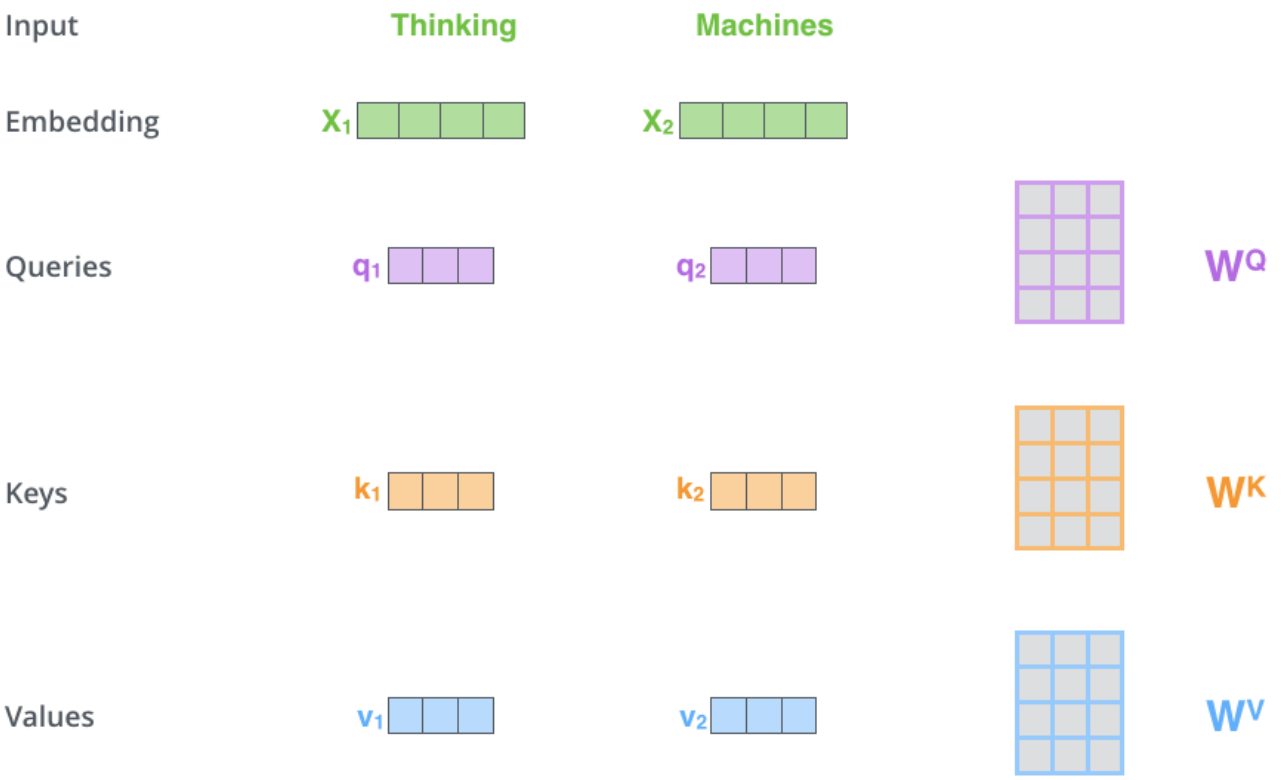
Transformer 모델

- Self-attention
  - 쿼리와 키 사이의 유사성 계산
  - 코사인 유사도 사용(Cosine Similarity)
    - 두 벡터 간의 코사인 각도
    - 두 벡터가 가리키는 방향이 얼마나 유사한가
    - scale = sqrt(dimension)

$$Similarity(Q, K) = \frac{Q . K^T}{scaling}$$

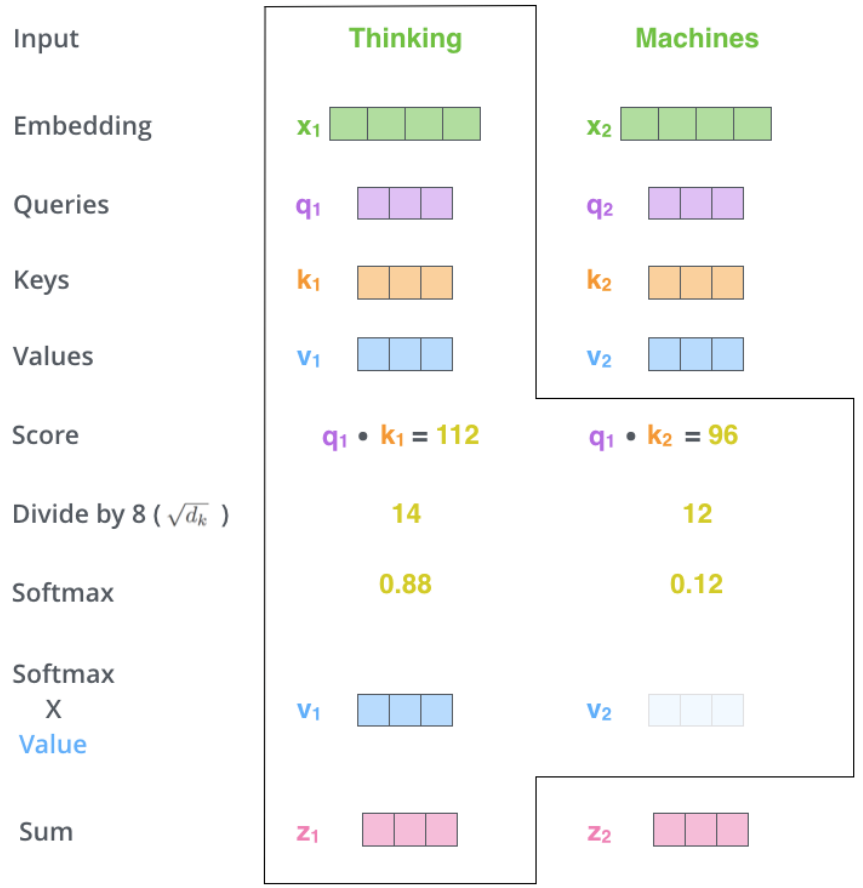
Transformer 모델

- QKV
  - W_q, W_k, W_v 를 찾아내는게 목표
  - 보통 input, output 의 차원보다 Q,K,V의 차원을 작게 만든다(논문에서는 64차원, 512=64*8, 여기서 8은 Multi-head의 숫자)



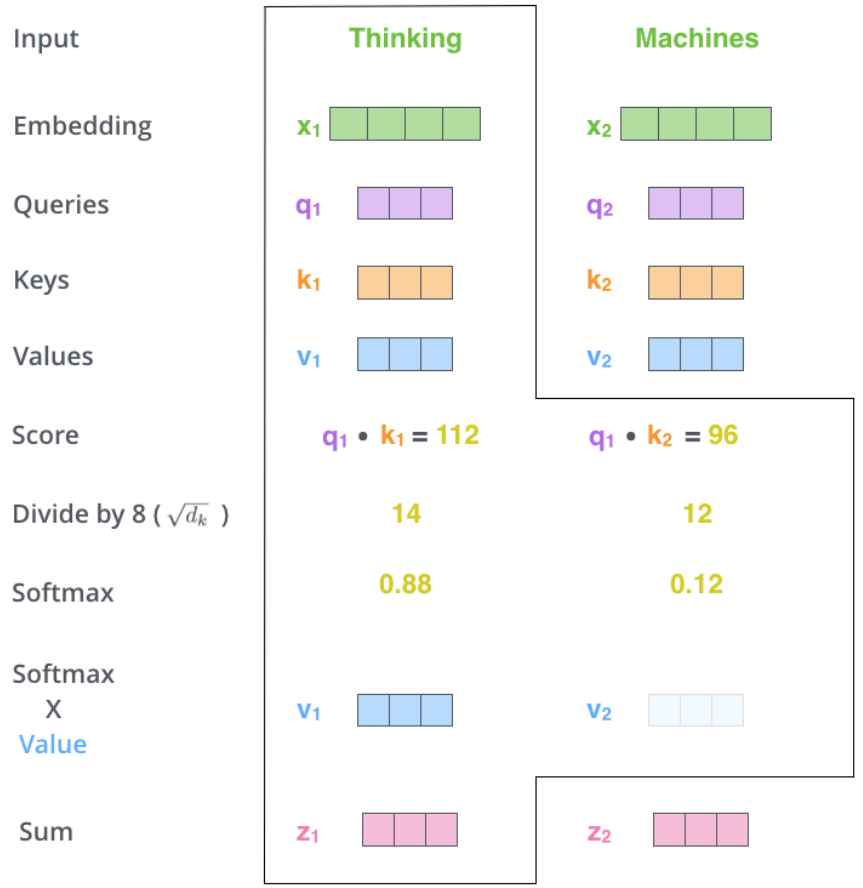
Transformer 모델

1. Thinking에 대해서 점수를 계산
   - q_1과 k_1, q_1과 k_2를 각각 내적하여 score 계산
2. Key의 벡터 사이즈인 64의 제곱근인 8로 각각을 나눠주고 softmax를 취해 줌
   - 이 softmax 점수는 현재 위치에 어떤 단어들이 들어갈지 결정하는데 도움을 줌



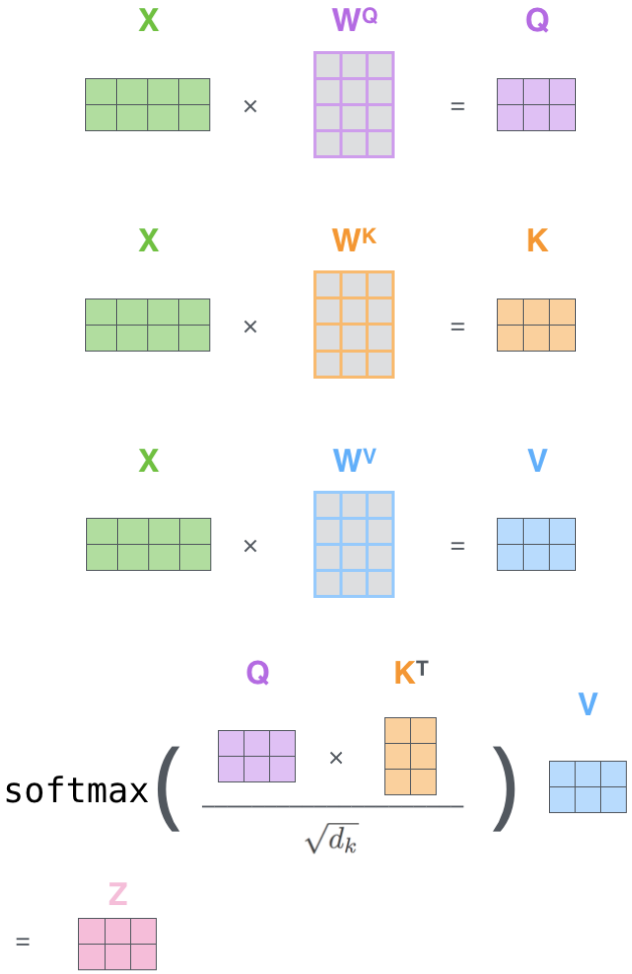
Transformer 모델

3. 각 단어들의 value에 이 점수를 곱함
   - 관련 있는 단어는 남기고 관련 없는 단어는 없애기 위함
4. 이렇게 곱해진 weighted value 벡터들을 다 합하여 출력

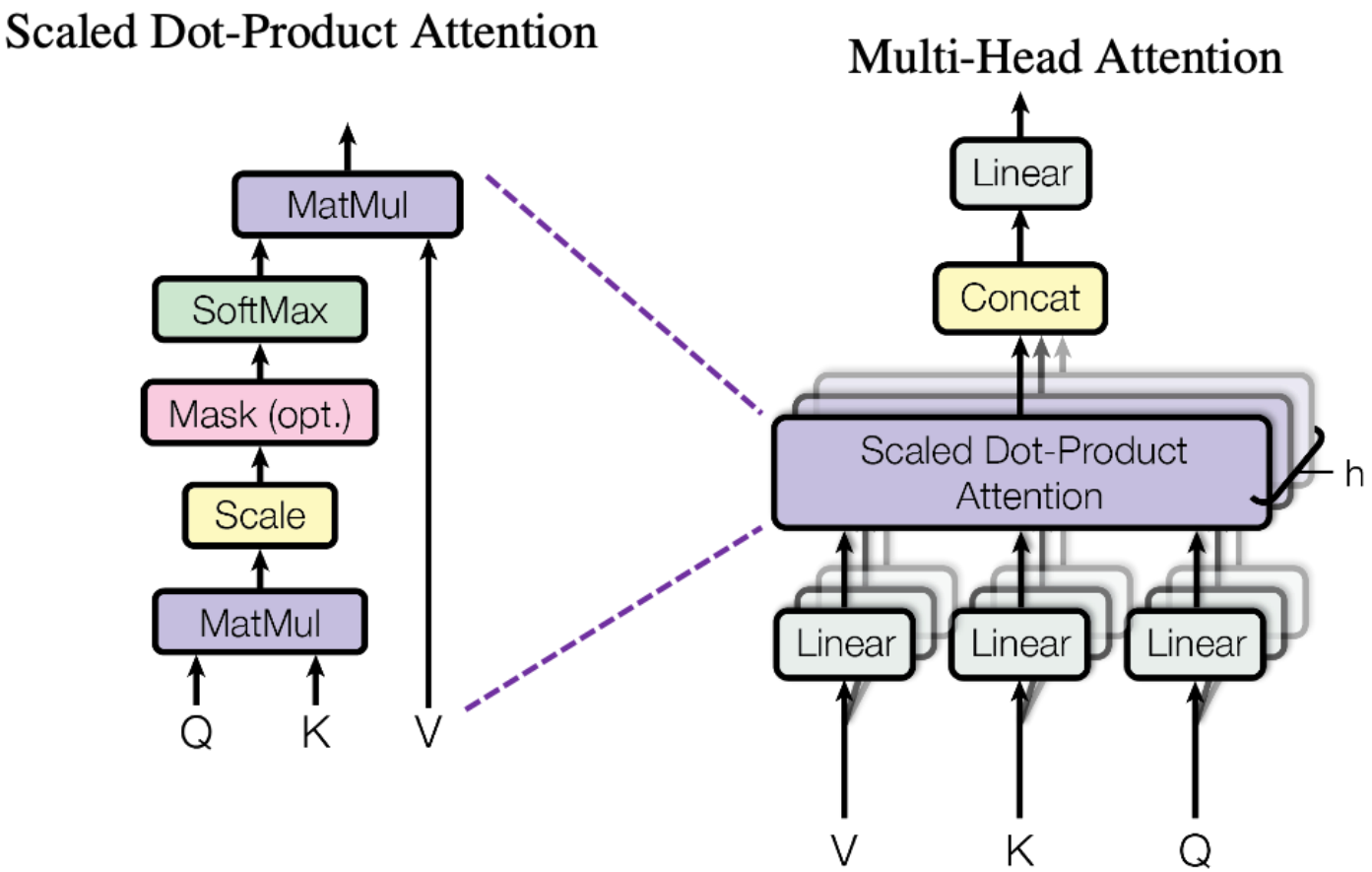


Transformer 모델

- 실제로는 앞서 나온 과정을 단어 하나하나당 하는게 아닌 하나의 행렬 x에 넣고 W_q, W_k, W_v 행렬과 곱하여 한 번에 계산



Transformer 모델

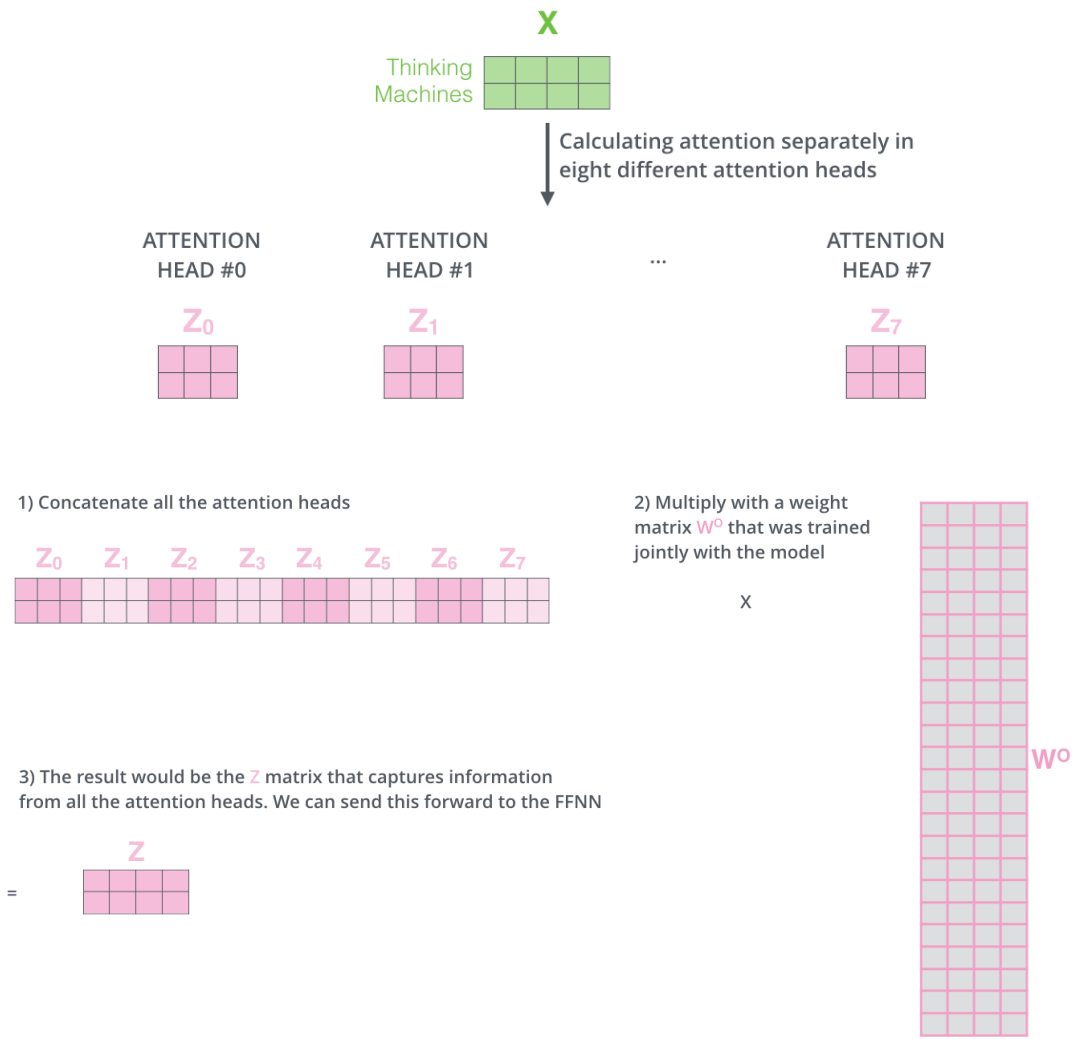


Transformer 모델

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

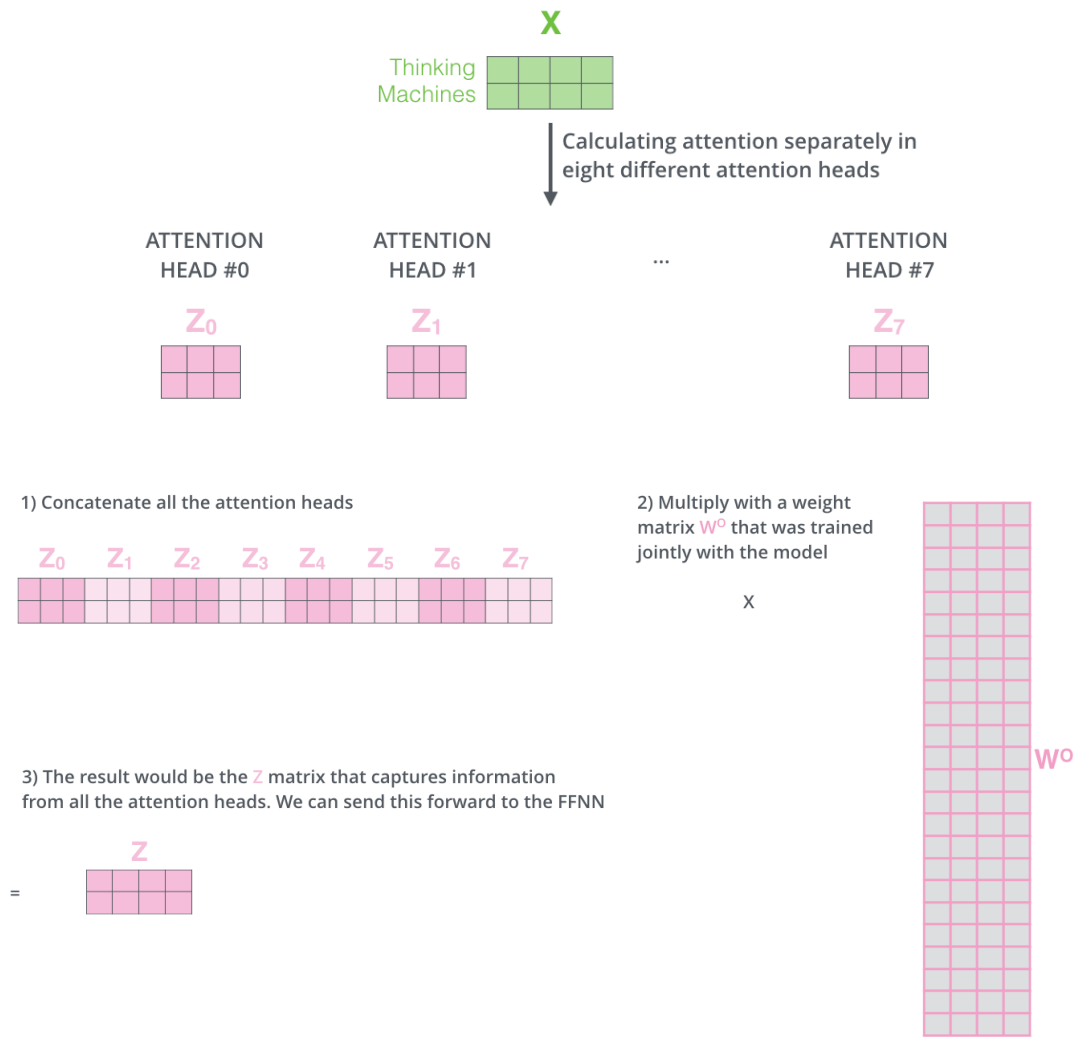
Transformer 모델

- Multi-headed attention
  - Attention layer가 여러 개의 representation 공간을 가지게 해 줌
  - 각 encoder, decoder마다 8개(논문기준)의 QKV weight 행렬 세트를 가지게 됨



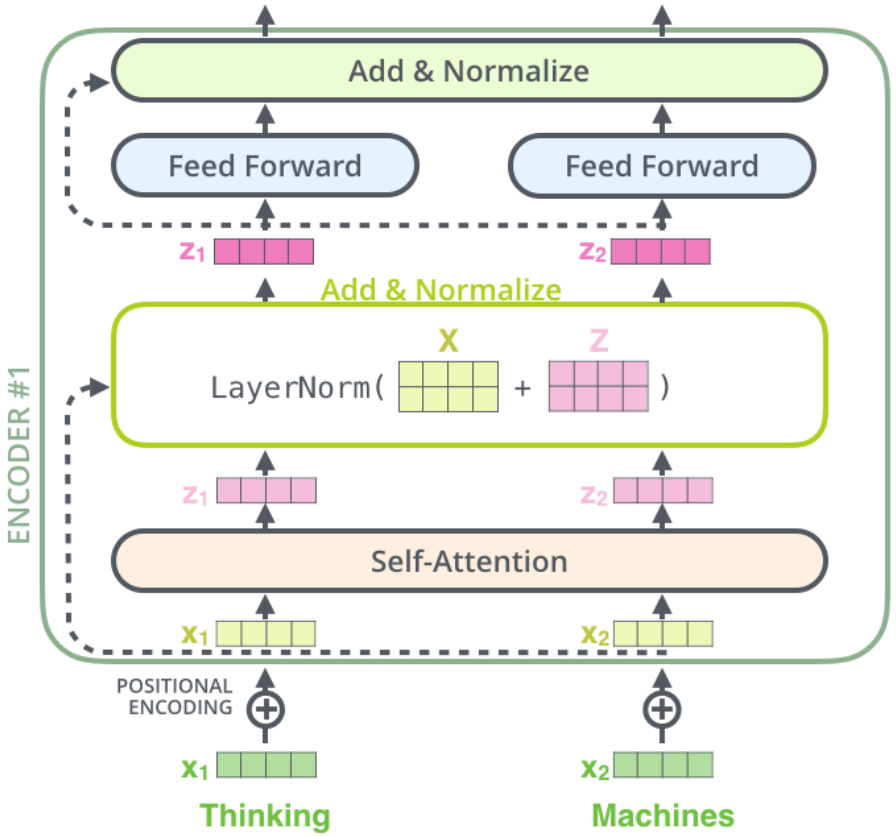
Transformer 모델

- Multi-headed attention
  - 각각의 세트는 랜덤으로 초기화되어서 학습되고 각각 다른 representation 공간으로 표현함
  - 8개의 행렬을 모두 이어서 하나의 행렬로 만들고 또다른 weight 행렬인 Wo와 곱함



Transformer 모델

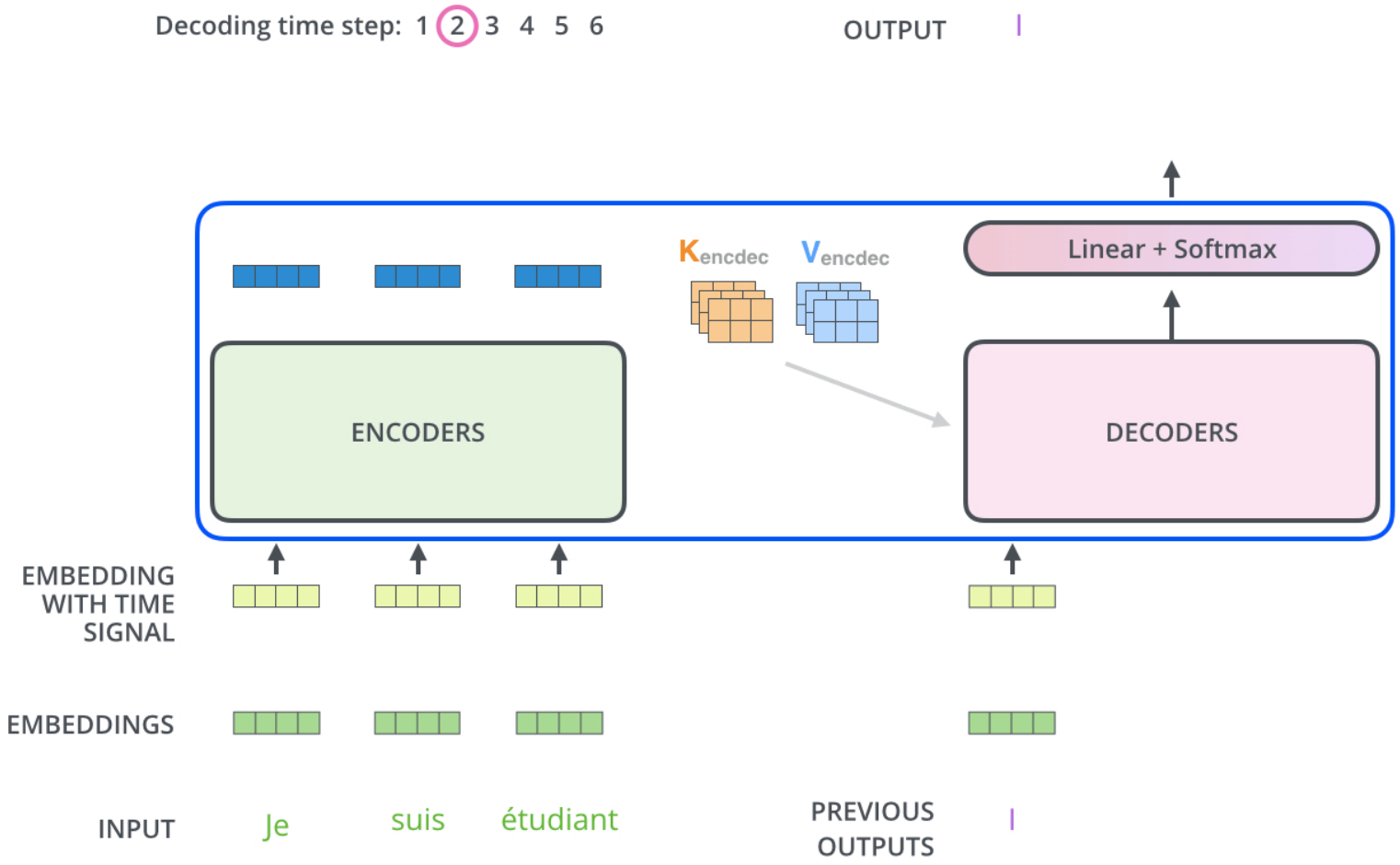
- Self-attention 의 결과는 residual 과 layer-normalization 을 거쳐서 feed forward의 input으로 사용됨
- 디코더에 있는 레이어들도 마찬가지 형태로 적용되어 있음



Transformer 모델

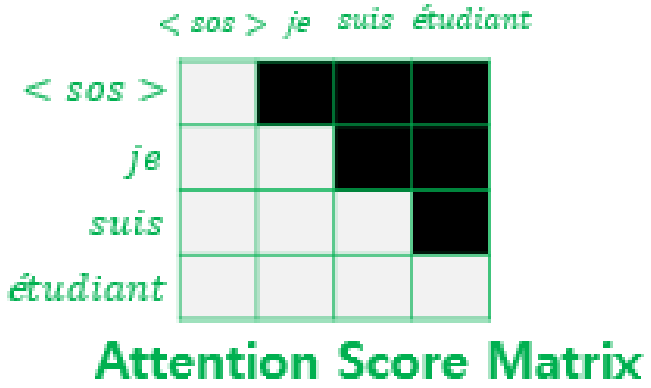
- Decoder side
  - Encoder가 입력 시퀀스를 처리하면 출력이 attention 벡터들인 K와 V로 변형됨
  - Decoding 단계의 각 스텝은 출력 시퀀스의 한 element 를 출력함
  - 이는 decoder가 출력을 완료했다는 기호인 〈eos〉 (end of sentence)를 출력할 때까지 반복

Transformer 모델



Transformer 모델

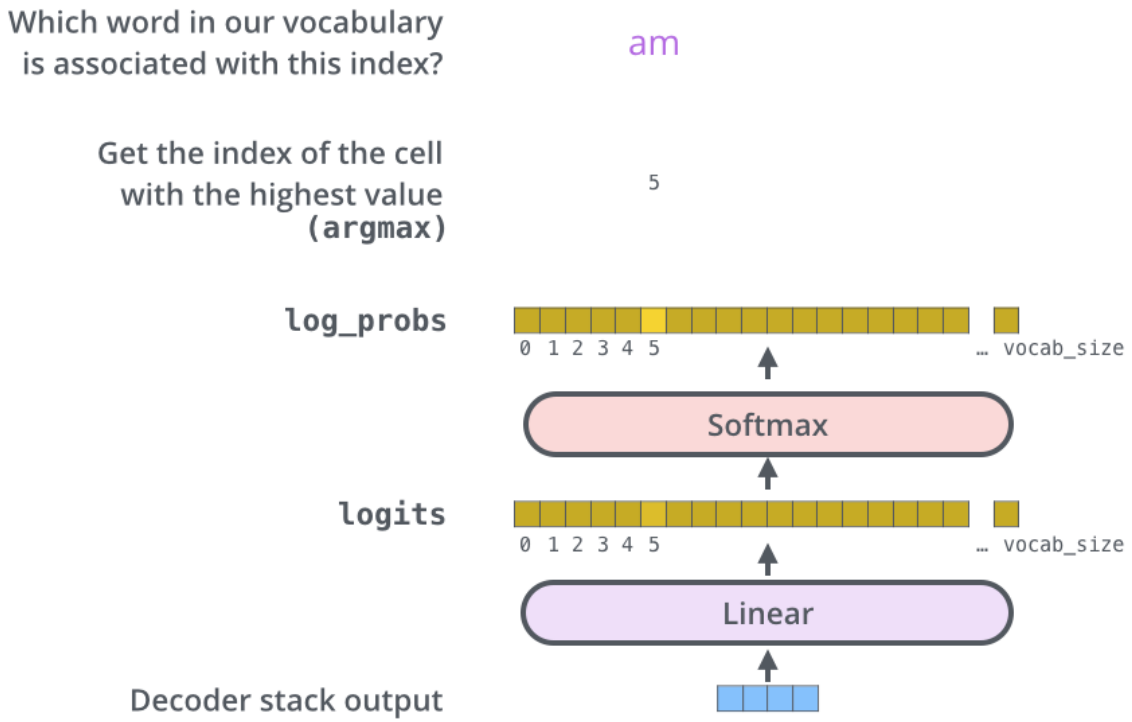
- Decoder side
  - 각 스텝마다 출력된 단어는 다음 스텝의 가장 밑단의 decoder에 들어감
  - Decoder 에서의 self-attention layer는 현재 위치 이전 위치들에 대해서만 attention을 계산
  - Self-attention 계산 과정에서 softmax를 취하기 전에 현재 스텝 이후의 위치들에 대해서 masking(-inf로 치환)해줌으로써 가능



Attention Score Matrix

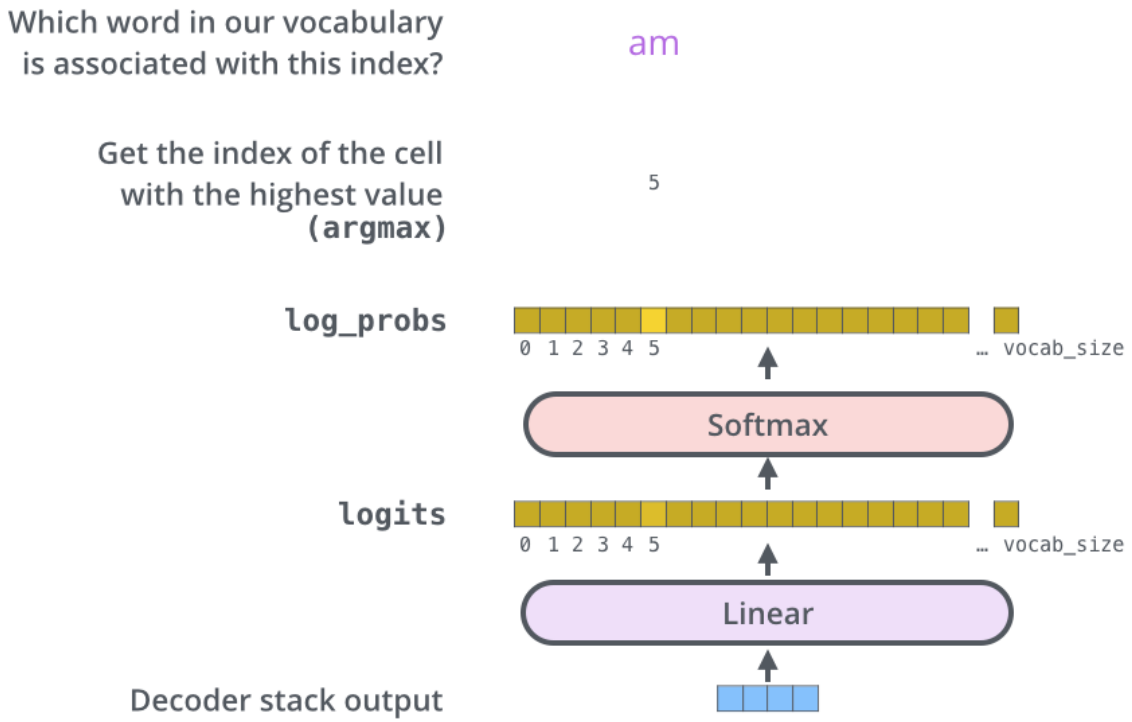
Transformer 모델

- Linear Layer와 Softmax Layer
  - Linear layer는 FC 신경망으로 decoder가 마지막으로 출력한 벡터를 그보다 훨씬 큰 사이즈의 logits 벡터로 투영
  - 모델이 training 데이터에서 총 10,000개의 영단어를 학습하였다고 가정하자(output vocabulary). 이 경우 logits vector의 크기는 10,000이 된다.



Transformer 모델

- Linear Layer와 Softmax Layer
  - 그 다음 softmax layer를 통과하여 가장 높은 확률값을 가지는 셀에 해당하는 단어가 최종 결과물로서 출력



Transformer 모델

- 학습 과정
  - label된 학습 데이터 셋에 대해 학습시키는 중이므로 실제 label된 정답과 비교할 수 있음
  - Output vocabulary는 preprocessing 단계에서 완성
  - Vocabulary의 크기만 한 벡터를 이용하여 각 단어를 표현

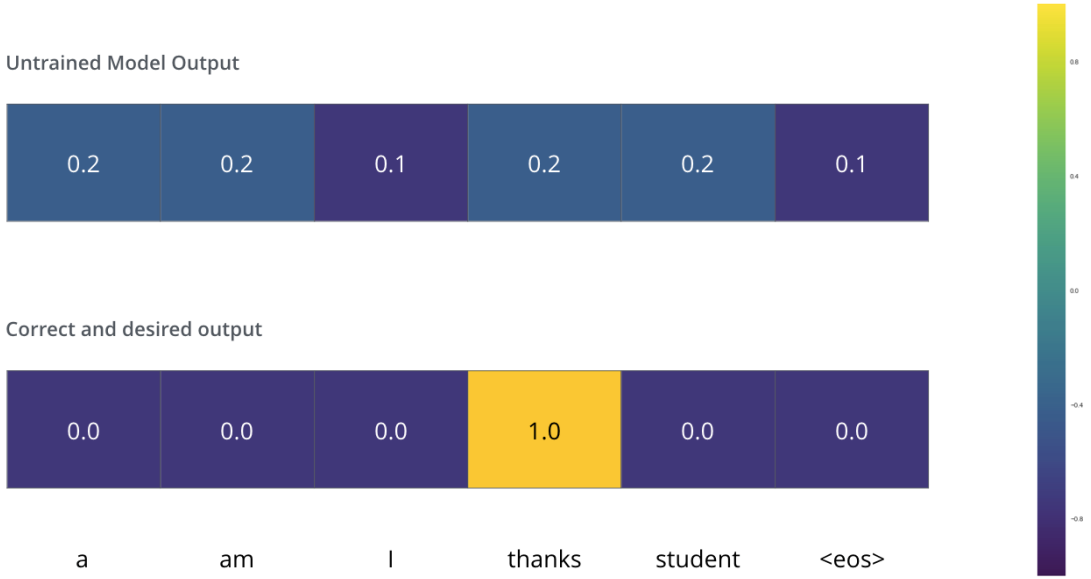
Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

One-hot encoding of the word “am”

Transformer 모델

- Loss function
  - “merci”를 “thanks”로 번역 하는 간단한 예시
  - 우리가 원하는 모델의 출력은 “thanks”라는 단어를 가리키는 확률 벡터
  - 학습이 시작될 때 W들은 랜덤으로 값을 부여하기 때문에 임의의 값을 출력
  - 이 출력된 값을 정답과 비교하여, 역전파를 이용해 현재의 모델들의 weight를 조절



Transformer 모델

- 모델의 출력 값은 모든 단어가 0보다는 조금씩 더 큰 확률을 가짐(softmax layer의 유용한 성질)
- 모델이 가장 높은 확률을 가지는 하나의 단어만 출력 할 수도 있지만 그보다 확률이 낮은 단어를 출력하게 할 수도 있음(실행할 때마다 확률적으로 다른 답이 나오게 됨)