13-2

전반적인 구조

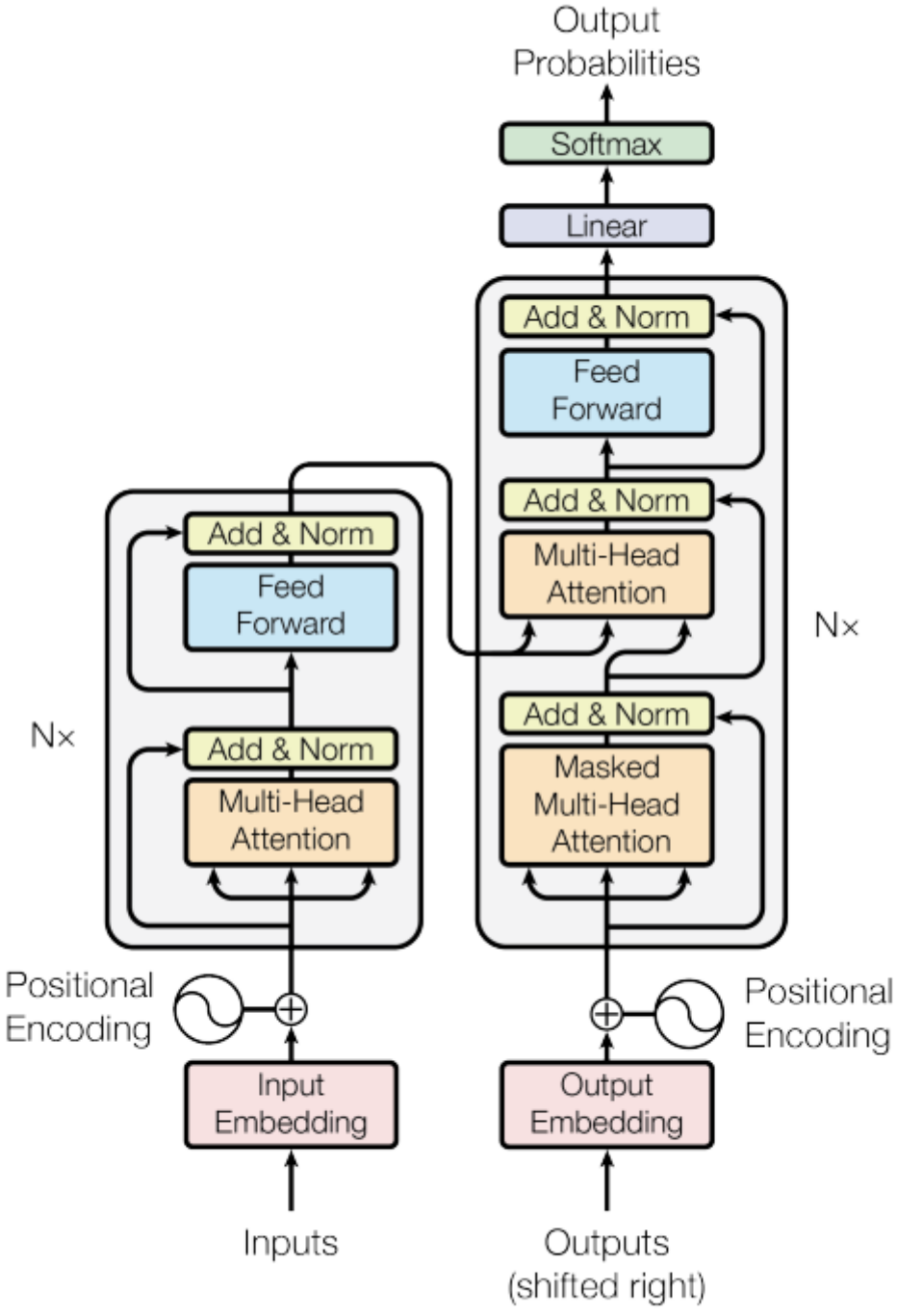
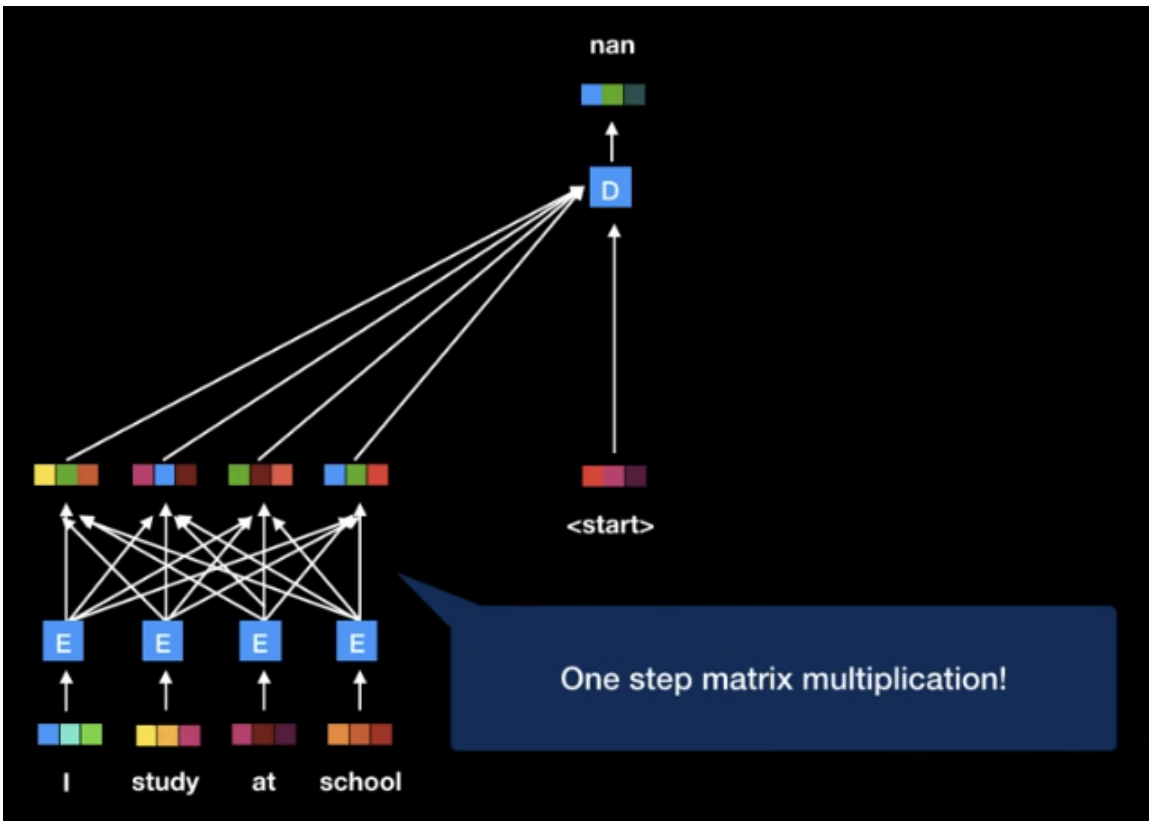
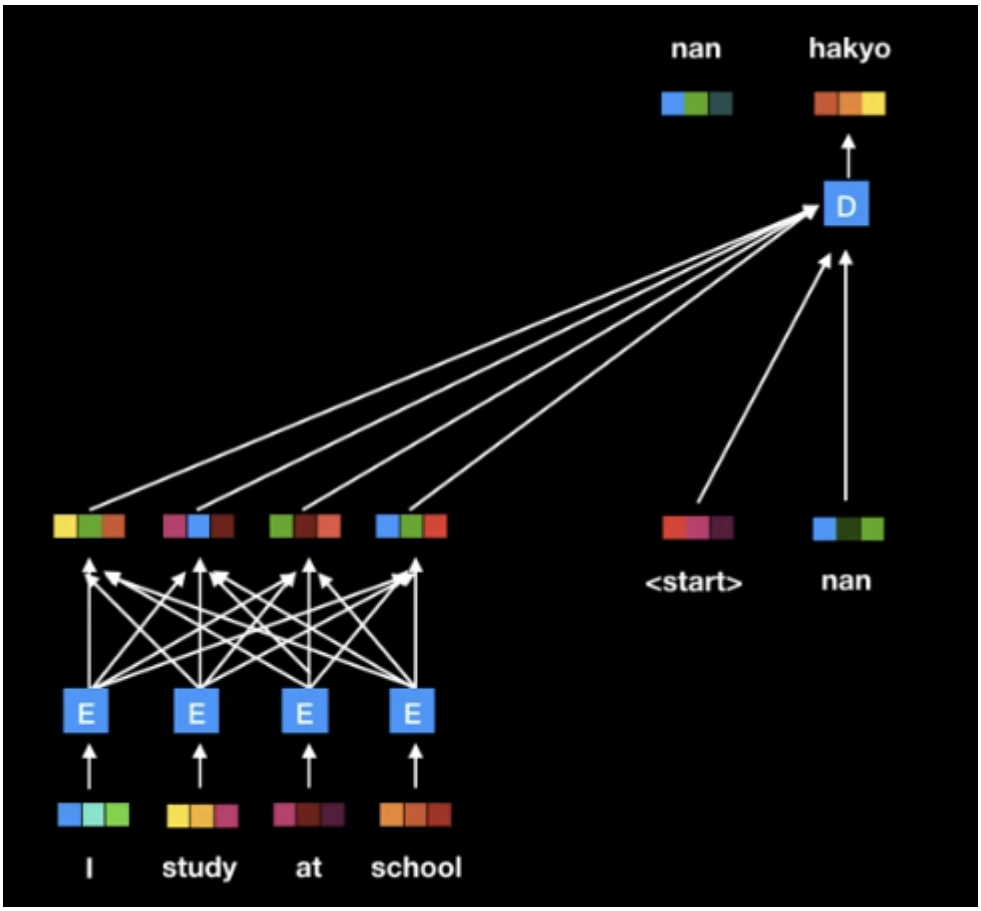
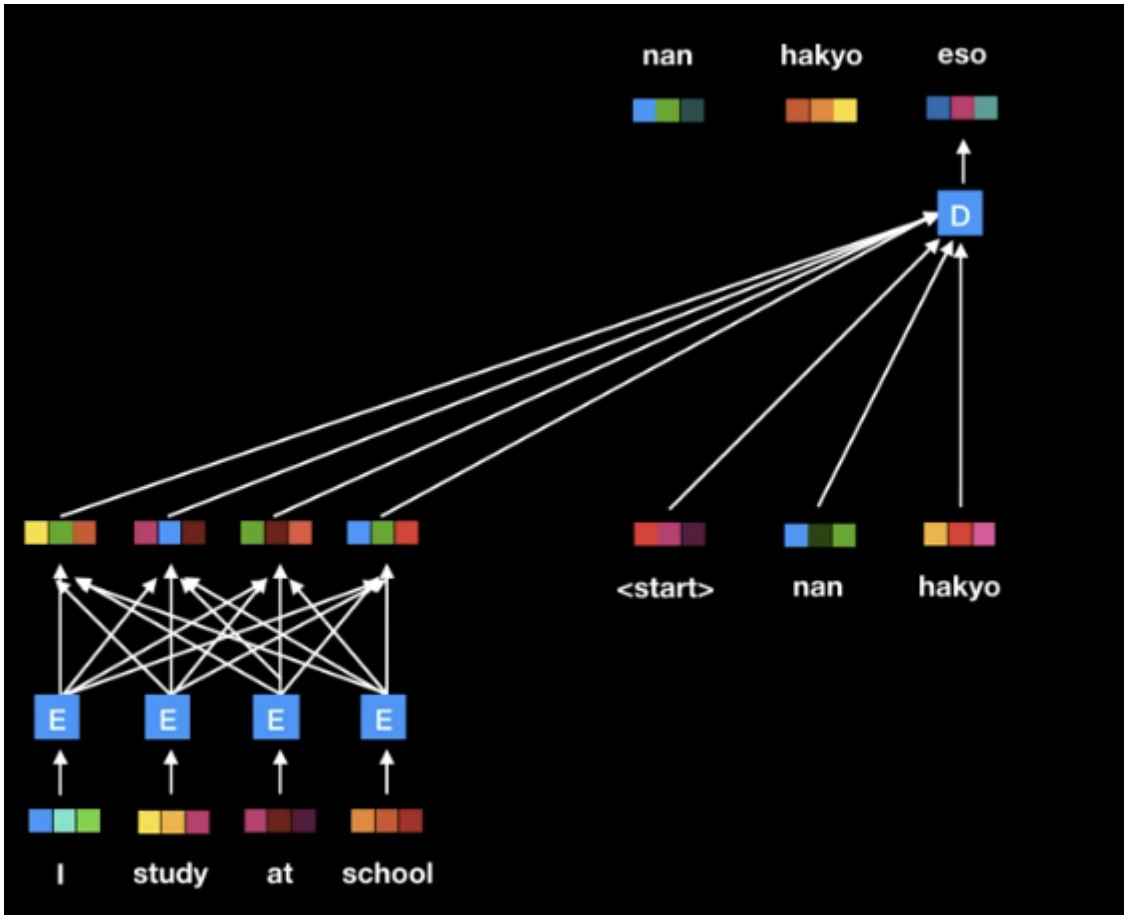


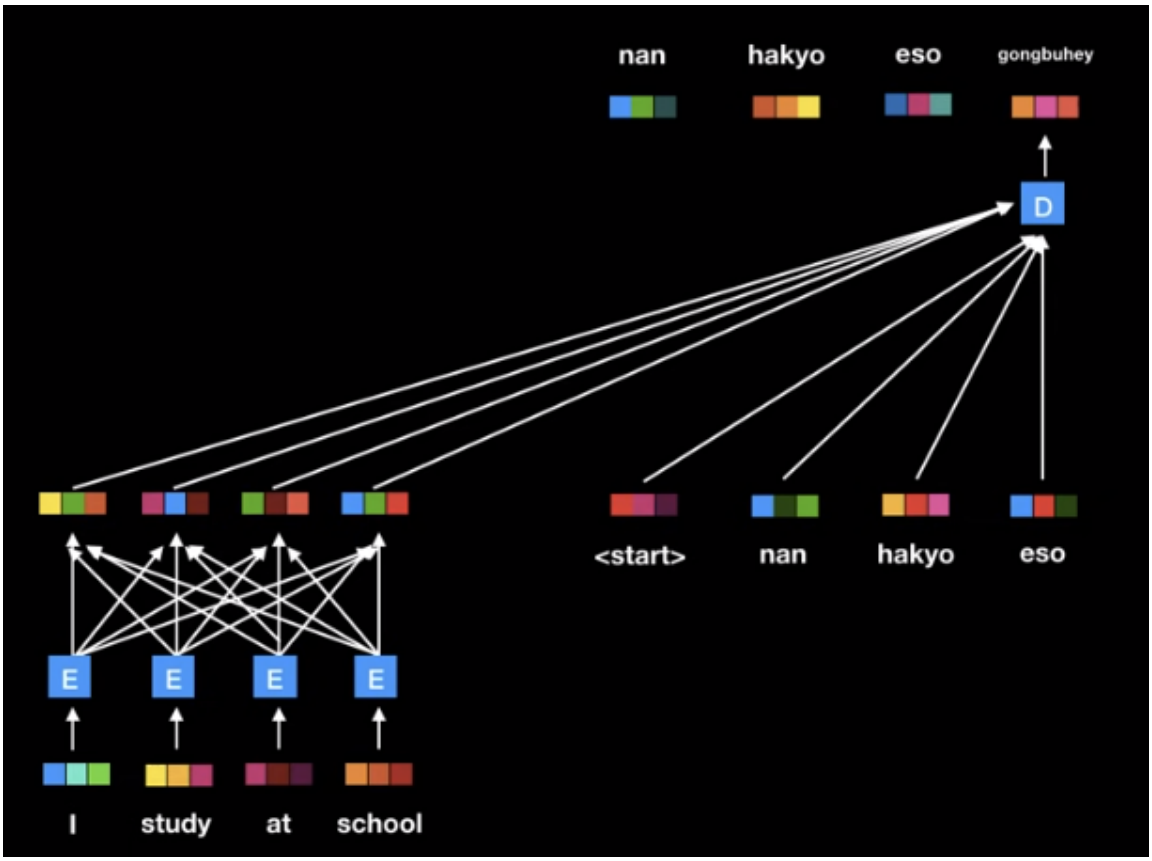
Figure 1: The Transformer - model architecture.

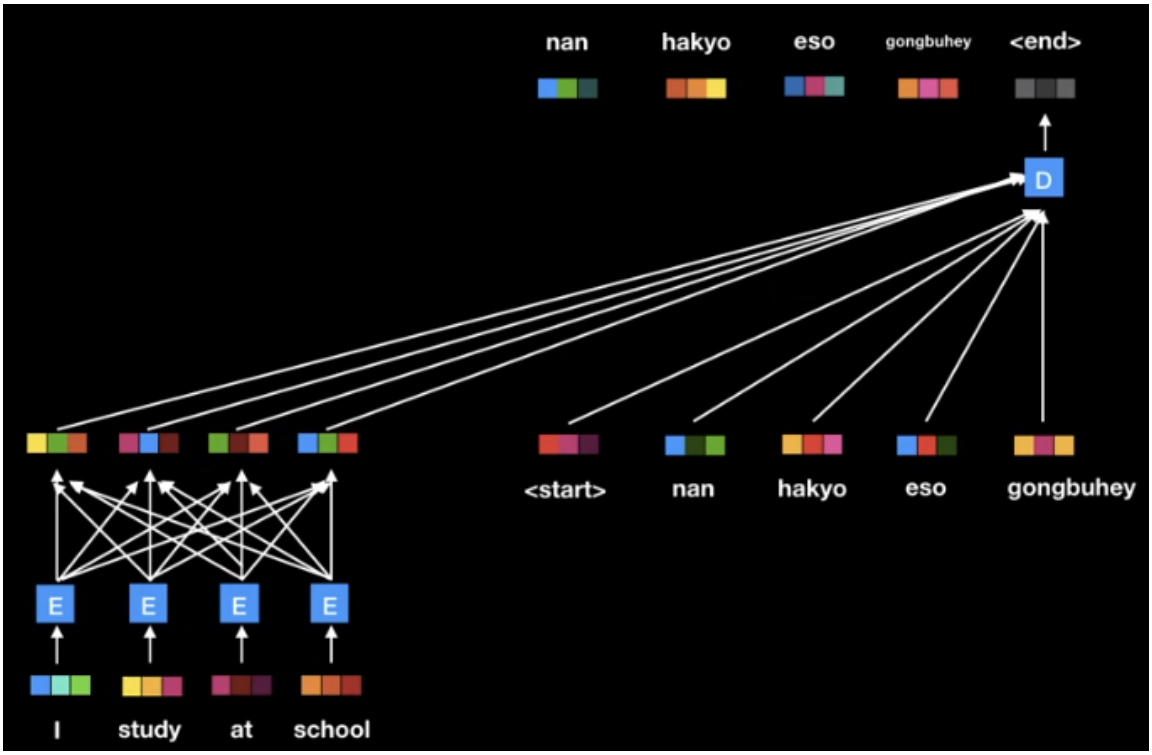
- shifted right의 의미






nan
hakyo
eso
D
<start>
nan
hakyo
E
E
E
E
I
study
at
school


nan
hakyo
eso
gongbuhey
D
<start>
nan
hakyo
eso
E
E
E
E
I
study
at
school



- softmax * value 를 다 더한 벡터 결과
  - 그냥 단어가 아닌 문장 속, 혹은 문맥 속의 단어 벡터

Encoder

input embedding

- 우리가 아는 단어를 컴퓨터가 인식할 수 있게 숫자 혹은 벡터로 표현하는 것
- 가장 쉬운 예시는 one-hot-vector
  - 하지만 문제는 단어가 1000개 있으면, 벡터의 차원수는 1000이 되고
    단어가 10만개 있으면 벡터의 차원수 역시 10만이 된다
  - 대부분이 0으로 이루어진 이 벡터의 특성상 희소 표현(sparse representation)
- 희소 표현이 있으면 밀집 표현(dense representation)
  - 벡터의 요소가 0과 1로 한정된 벡터가 아니기에 적은 차원으로 훨씬 많은 표현 가능
    → 사용자가 벡터의 차원을 정할 수 있음
- 이런 단어 embedding은 사전 데이터로 학습을 한 결과물이라고 함
  - 예를 들어,
    this 와 car가 비슷한 특성을 지니고, 문맥 유사도가 높으면 embedding vector값

이 가까워 지고, 반대로 좀비는 가까워지지 않음

https://oopy.lazyrockets.com/api/v2/notion/image?src=https%3A%2F%2Fs3-us-west-2.amazonaws.com%2Fsecure.notion-static.com%2Fc7d5f597-399a-4084-ab8f-1bd8b340892e%2FUntitled.gif&blockId=c3f57ec7-a058-4179-bc78-4ec07183786f

- 여기서 한 단어를 나타내는 embedding vector의 차원은 512

positional encoding

- positional encoding

3.5 Positional Encoding

Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

In this work, we use sine and cosine functions of different frequencies:

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

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [9] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

  - 해당 부분은 위 내용이 전부
  - 그마저도 왜 이런 함수를 썼는지 자세하게 설명되어 있지 않음, 왜 positional encoding이 필요했는지 설명
    - 기본적으로 seq2seq은 문장을 단어 단위로 순차적으로 입력하기 때문에 입력 자체에 순서 정보가 들어가 있음

        - 근데 transformer는 문장, 혹은 문단을 통째로 input으로 넣고 병렬적으로 처리하기 때문에, 순서 정보가 없음
        - 하지만, 문장의 해석이나 번역에서 단어의 순서 정보는 매우 중요
            - 예를 들어,
        - 그렇기 때문에 위치 정보를 전달 혹은 저장 혹은 포함하고 있는 벡터가 필요, 이게 positional encoding vector의 존재
    - 여기서 알 수 있는 것은, 후에 여기서 나오는 positional encoding vector와 앞서 들어온 input embedding vector와 더할거기 때문에, positional encoding vector 역시 차원이 512이다
    - 그럼 왜 sin cos를 사용했나
        - 일단 positional encoding vector는 몇가지 조건이 필요
            - 같은 위치이면 시퀀스의 길이나 input에 상관없이 같은 값을 반환해야 한다
                - 예를 들면, 2번 위치의 벡터를 A라 하면, 2번 위치에 hello가 오든 world가 오든 항상 벡터 A를 반환해야 한다는 뜻,
                - 어쩌면 매우 당연한 소리, 위치 정보를 식별하고 싶은데, 같은 위치에 다른 위치 벡터가 나오는 건
                - 비슷한 이유로 2번 위치 벡터가 A인데 5번 위치 벡터도 A이면 안됨
            - 벡터의 크기가 너무 크면 안됨
                - 나중에 input vector와 더해 연산을 할 건데, 위치 벡터의 크기가 너무 커지면, 앞서 도출한 단어의 정보를 담고 있는 벡터의 정보가 상대적으로 소실됨
                - 예를 들어, 단어 임베딩 벡터은 0과 1사이의 값으로 정규화를 했는데, 위치 벡터의 크기가 100, 1000이러면 상대적으로 임베딩 벡터의 존재는 흐려짐
        - 몇가지 예를 통해,
            - 위치 정보를 그대로 사용,
                - 예를 들어, 1번째 위치는 모든 벡터의 요소가 1
                  2번째 위치는 모든 벡터의 요소가 2
                - 1번 조건은 만족하지만 2번 조건 만족하지 않음
            - 가장 앞을 0, 가장 뒤를 1로 하고 1/n하여 부여

        - 2번 조건을만족하지만 1번 조건을 만족하지 않음
    - 처음부터 증가폭을 매우 작게 설명하면
        - 너무 제한적이로, 기본적으로 512 차원의 벡터를 모두 같은 숫자로 채우는것 자체가 비합리적
    - 함수를 사용하자
        - 그렇다면 사인 코사인은 어떨까
            - 크기가 최대 1 최소 -1로 정해져 있어 2번 조건은 만족
            - 근데, 주기함수이기 때문에, 2파이 혹은 설정된 주기에 따라 다른 위치에 같은 위치 벡터가 할당될 수도 있음
        - 그렇다고 비슷한 느낌의 sigmoid 함수는?
            - 같은 값을 가질일은 없지만, 문장이 길면, 뒤에 위치한 단어의 위치벡터는 매우 비슷해짐 → 1로 수렴하니
        - 다시 사인 코사인으로 돌아와서
            - 알아야할 점이, 지금 우리는 위치 벡터를 구하는 거지, 위치 스칼라를 구하는게 아님,
            - 즉, 512차원의 벡터를 정하면 되기에, 그냥 서로다른 주기를 가지는 사인 코사인 함수 512개를 사용하여, 각각의 벡터의 차원 혹은 인덱스에 적용을 하면 됨
                - 예를 들어, i번 인덱스는 주기가 1인 사인 코사인을 사용하고, i+1번 인덱스는 주기가 2인 사인 코사인을 사용하고
                - 근데 모두 다 사인을 혹은 모두 다 코사인을 사용하게 되면, 벡터간의 위치 정보 차이가 미미하게 되니, 사인 코사인을 번갈아 가며 사용
    - 이렇게까지 해서 transformer 를 사용하는 이유는 그만큼 transformer가 가지는 병렬성이라는 무기가 강력하기 때문

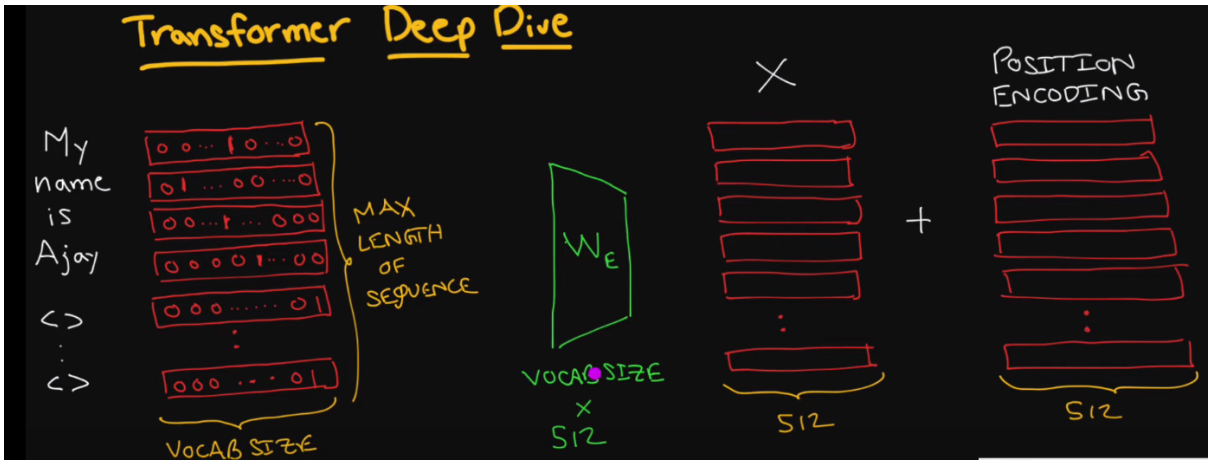
    https://www.blossominkyung.com/deeplearning/transfomer-positional-encoding

- positional embedding
    - positinal embedding은 위치 정보를 표현하기 위해 추가적인 임베딩 층을 사용하여 학습이 이루어 지면서, 이 위치 정보를 담고 있는 임베딩 층 역시 스스로 학습하는 것

- https://heekangpark.github.io/ml-shorts/positional-encoding-vs-positional-embedding#kramdown_positional-embedding

- 결론
  - attention is all you need 논문을 보면 저자들이 둘 다 해봤지만 비슷했다 라고…

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

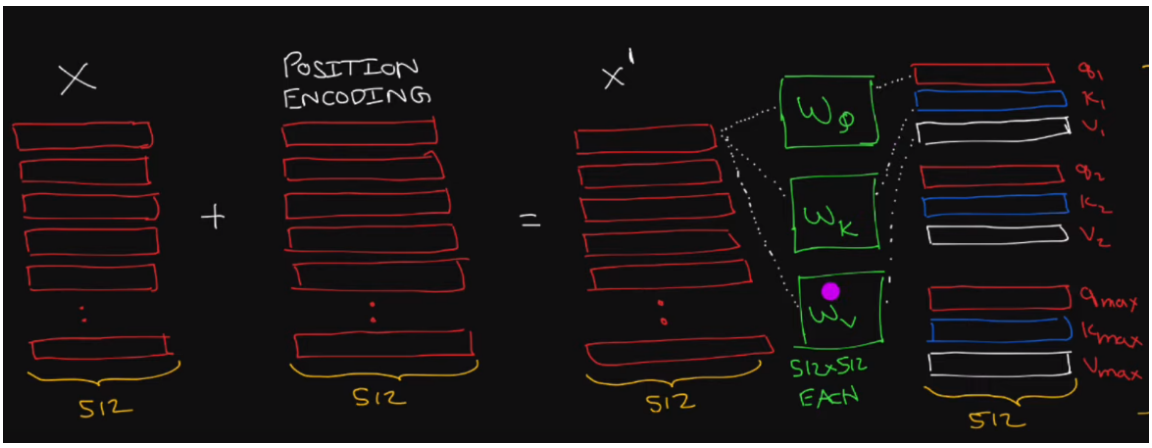


https://youtu.be/ZMxVe-HK174

****

multi head attention

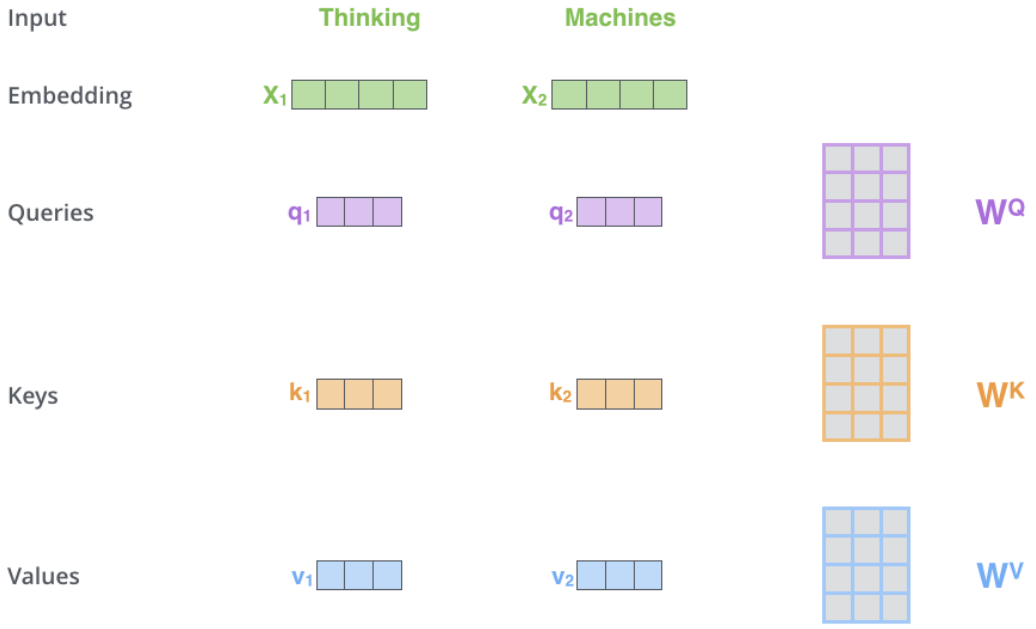
- self attention



https://youtu.be/ZMxVe-HK174

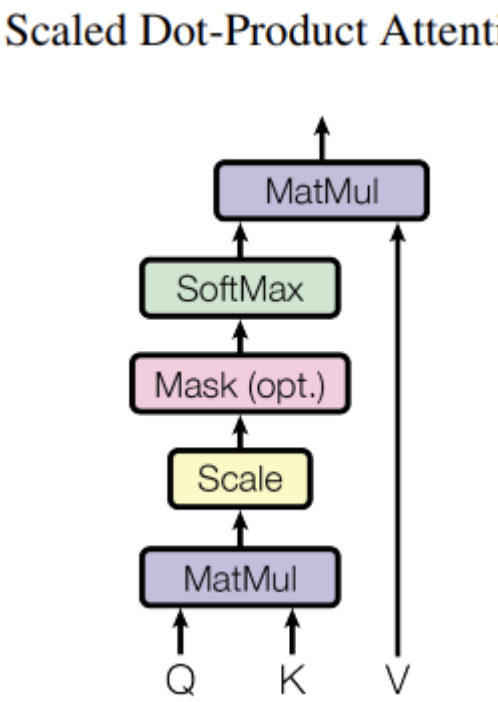
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

  - 임베딩을 마치고 positional vector와 합한 결과인 X을 QKV로 나누는 건

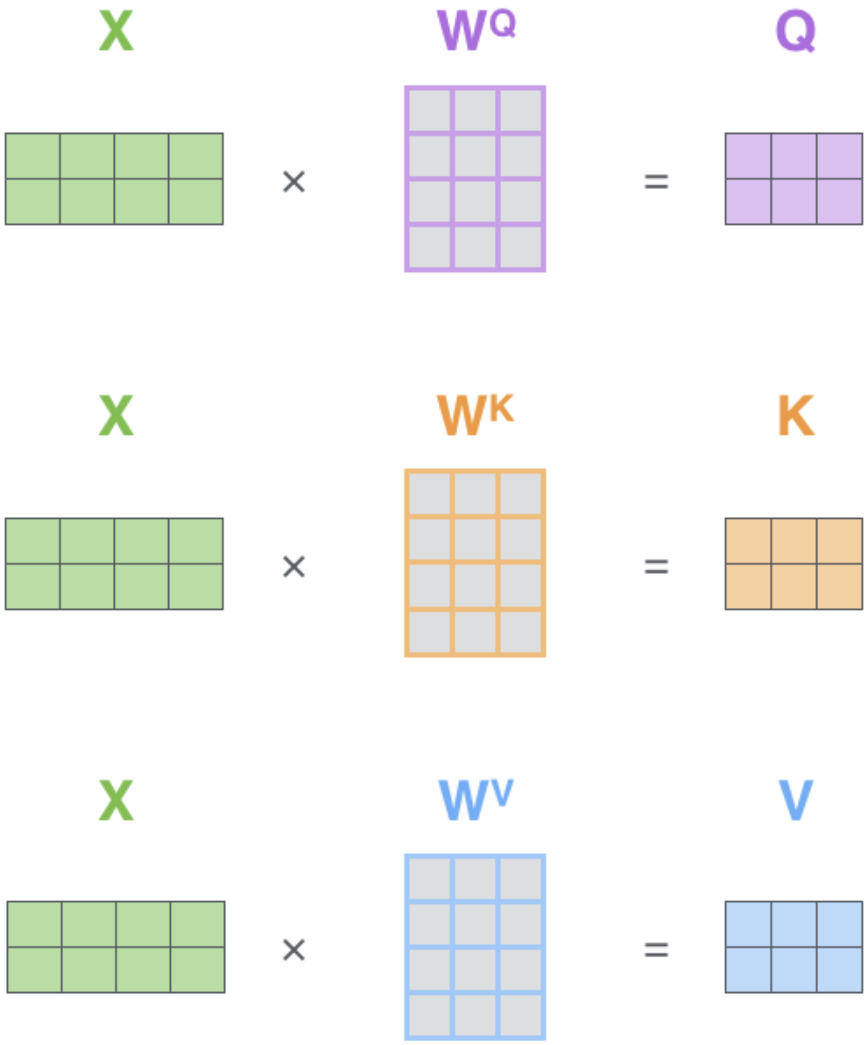


https://jalammar.github.io/illustrated-transformer/

  -



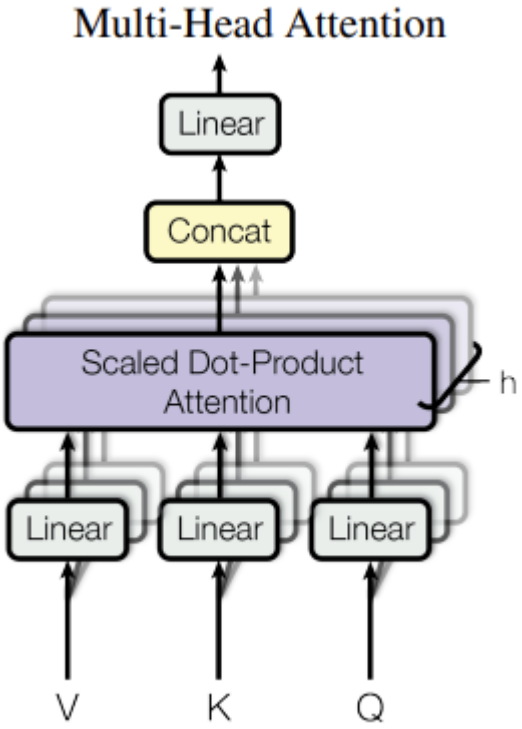
- 위 그림 설명
  - matmul은 행렬 곱
    - 여기서 실제로 일어나는 것은 벡터의 내적인데 행렬 곱이라고 표현한 이유
      -



https://jalammar.github.io/illustrated-transformer/

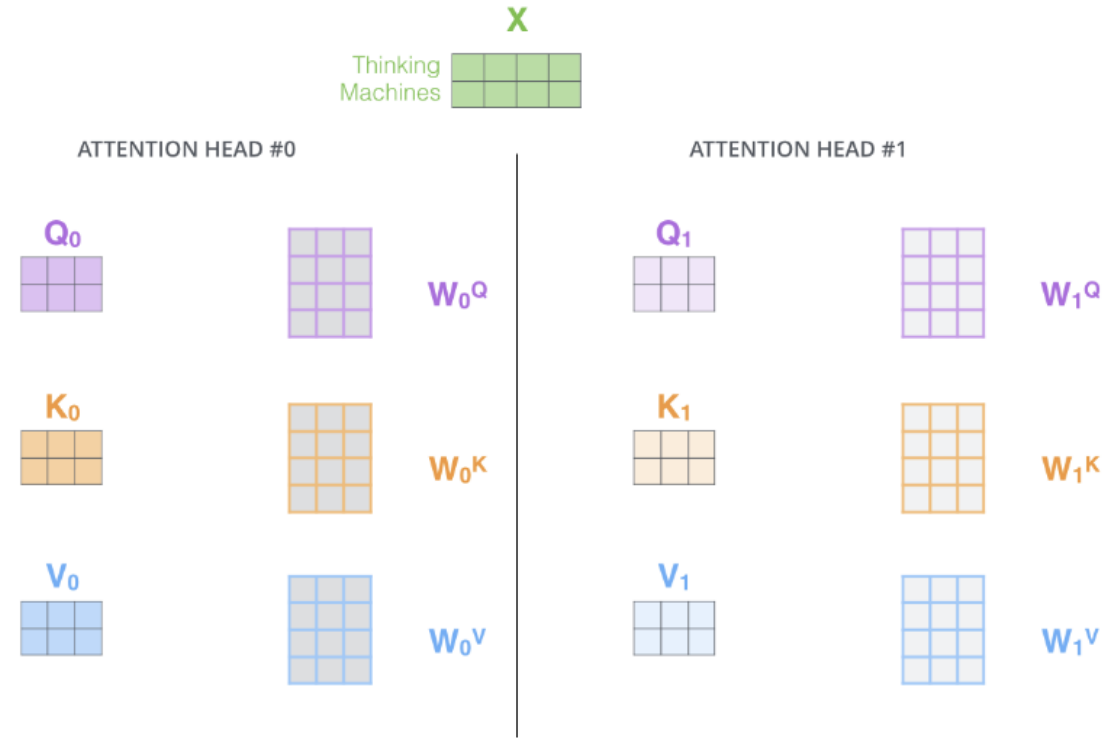
- 모든 단어 벡터, 혹은 토큰으로 불리는 벡터는 하나의 행렬로 concat 되어 연산되기 때문에, 위에서 보는 Q K V는 사실 단일 단어의 벡터가 아니라, 모든 단어, 혹은 토큰 벡터의 합인 행렬이다, 그리고 순차적인 벡터 연산이 아닌, 하나의 행렬 연산이 가능하다는 것이 transformer 의 강력한 무기
- 그리고 이러한 행렬을 곱 연산 하기 때문에, Q행렬의 행, K행렬의 열 단위로 내적이 이루어 지지만, 전체적으로 봤을때는 Q행렬과 K행렬의 행렬 곱
  - 내적을 하는 이유, 벡터의 내적을 보면 이해가 쉬운데, 벡터의 방향이 비슷하면 내적은 큰 값을 가짐, 즉, 큰 값이 나올수록 두 벡터의 방향이 비슷하다는 것 → 유사함의 척도

                - 그러면, 방향이 조금 다르더라도, 하나의 벡터의 크기가 엄청 크게 학습이 되어도, 유사도가 크게 나올거 아니냐
                    - 하지만, 하나의 가중치 행렬을 모든 토큰이 공유하기 때문에, 하나의 단어의 유사도를 위해 가중치 행렬의 특정 요소를 이상치로 만드는 것은, 전반적으로 봤을 때 오히려 LOSS를 높히는 결과로 이어짐
        - scale은 앞서 설명한 루트 dk(즉 8)(dk = 64)로 나누는 작업
            - 이를 통해, 몇몇의 너무 큰 값때문에, 전체적인 그래프가 너무 가파른 기울기를 갖지 않게, 학습을 안정적으로
        - mask 작업은 보시면 opt.라고 되어있는데, 이는 optional 즉 항상 있는 과정은 아님
            - 인코더 부분에서는 일단 mask 작업이 필요없으니, 후에 디코더 설명할 때 자세하게 설명
        - 소프트 맥스
            - 기본적으로 Q와 K와의 내적은 두 단어, 혹은 토큰의 유사도를 측정하는 척도이기에, 여기에 소프트 맥스를 취하면 0과 1사이의 확률값이 되고, 이를 모두 합한 값이 1이 됨
        - 여기에 V행렬을 행렬 곱을 하면 단지 단어 그 자체의 정보만 담고 있던 V행렬이, 이제 문맥속에서의 단어 정보를 담고있는 행렬 V가 됨
- multi head attention



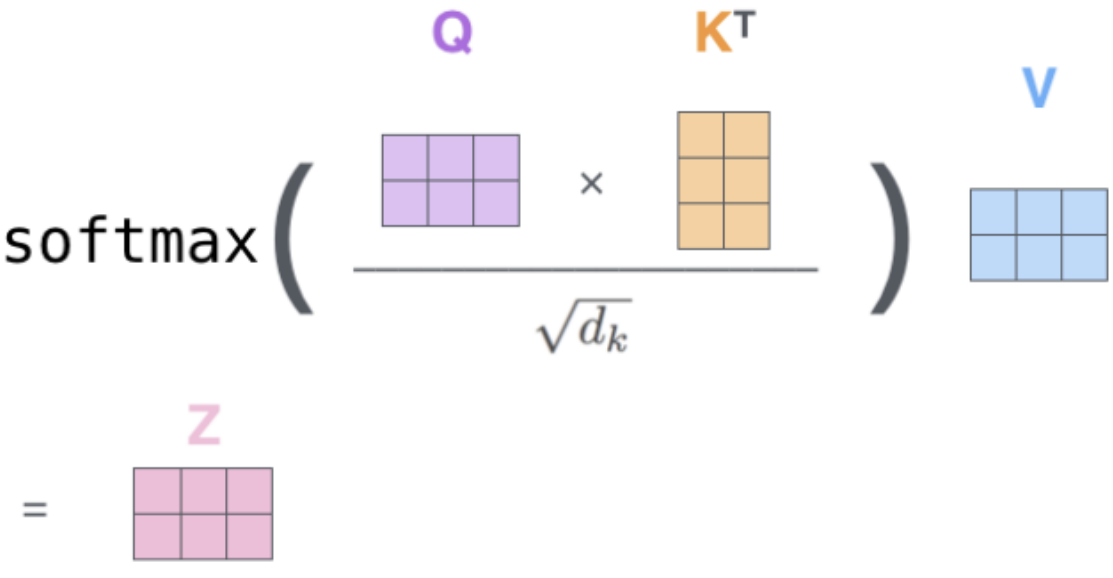
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- 결론부터 말하자면, h=8 즉, 헤드는 8개 이고, 각각의 헤드는 각각의 쿼리 가중치 행렬, 키 가중치 행렬, 밸류 가중치 행렬이 있다. 즉, 하나의 인코더의 하나의 멀티 헤드 어텐션에는 총 3 * 8 = 24개의 가중치 행렬이 있다는 의미
-



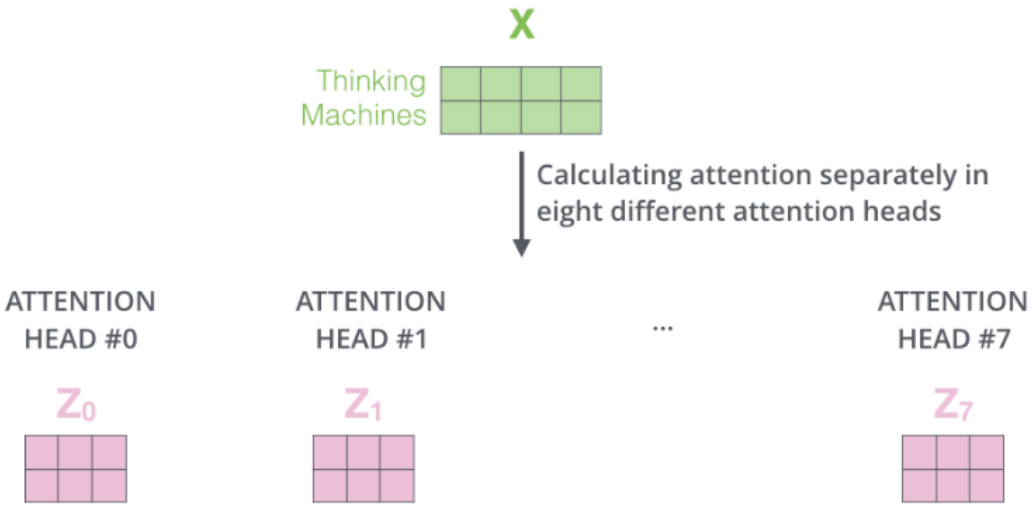
https://jalammar.github.io/illustrated-transformer/\

- 동일한 X에서 각각 다른 가중치와 연산하여 각각 다른 QKV를 도출
- 그러면 이러한 head 하나하나에서



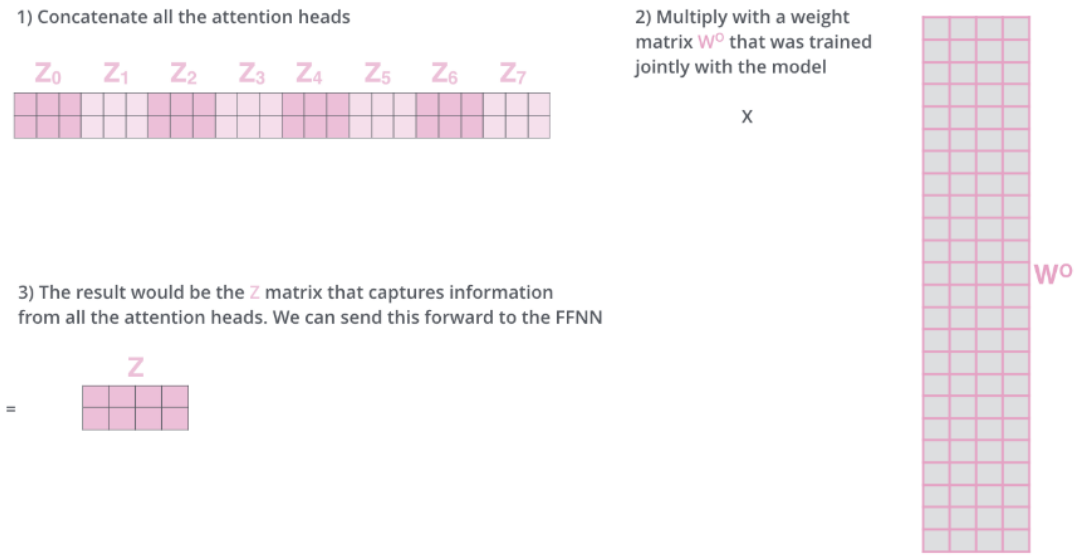
https://jalammar.github.io/illustrated-transformer/

- 위와 같은 연산이 일어나고



https://jalammar.github.io/illustrated-transformer/

- 이렇게 총 8개의 벡터가 나옴을 알 수 있음
- 문제는 이러한 벡터가 바로 그 다음 과정인 피드 포워드 과정으로 넘어갈 수 없음
  - 먼저 8개 각각의 행렬은 모두 문장 혹은 시퀀스 전체의 의미를 담고 있기 때문에, 이 8개의 행렬을 순차적으로 피드포워드에 넣는다는 것은 논리적으로 모순
  - 8개의 행렬은 마찬가지로 제일 처음 input인 X와 모양이 같지 않아 넣을 수 없음
- 이를 위해 이 Z들을 concat 해서 새로운 가중치 행렬 W0에 곱해, 내용적으로도 모든 값을 포함하고, 모양도 맞춰줌



https://jalammar.github.io/illustrated-transformer/

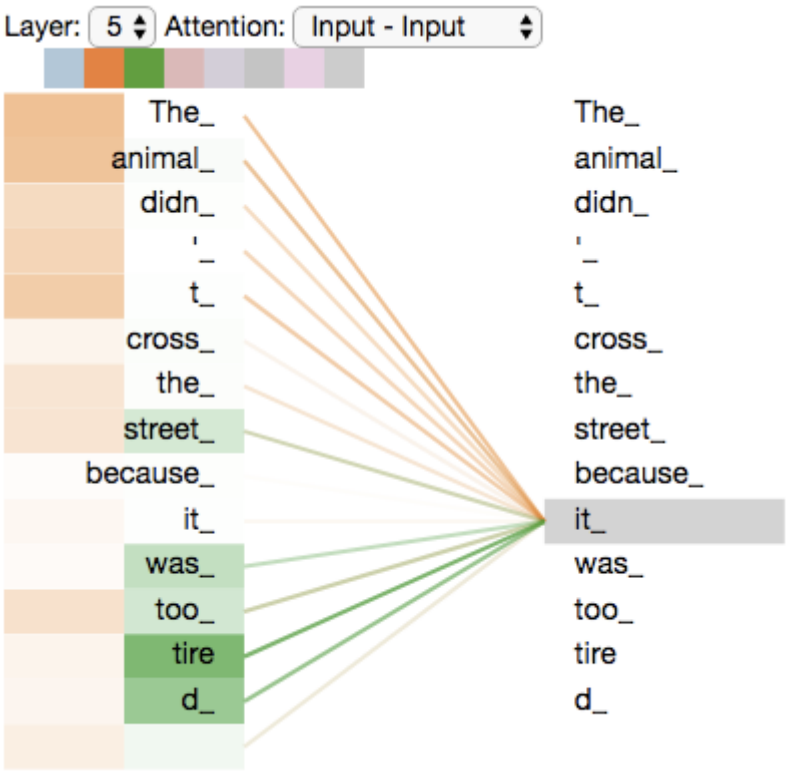
- 이렇게 하는 이유

3.2.2 Multi-Head Attention

Instead of performing a single attention function with d_{model}-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding d_v-dimensional

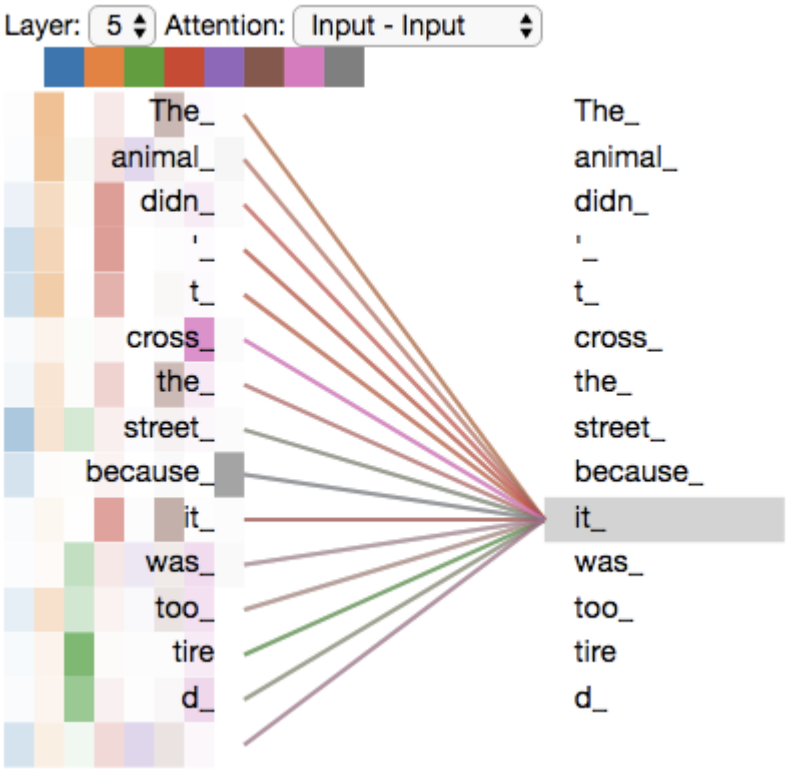
---

- 논문에서는 그냥, 이렇게 병렬로 하는게 좋다라고 함
- 인간으로 치면 8개의 서로 다른 시각, 혹은 관점에서 해석할 수 있음
- 즉, 더 많은 표현 공간을 줌



https://jalammar.github.io/illustrated-transformer/

- 위 그림은 head가 2개일 때, IT이라는 단어를 어디에 가장 연관 있다고 판단하는 지 보여주는 시각화 자료
  - the animal didn’t cross the street because it was too tired
  - 에서 it이 가장 연관있는 단어로, 주황색 헤드는 the animal을 골랐고, 초록색 헤드는 tired를 골랐다.
  - 보면 둘 다 상당히 합리적인 결과라는 생각이 듦



https://jalammar.github.io/illustrated-transformer/

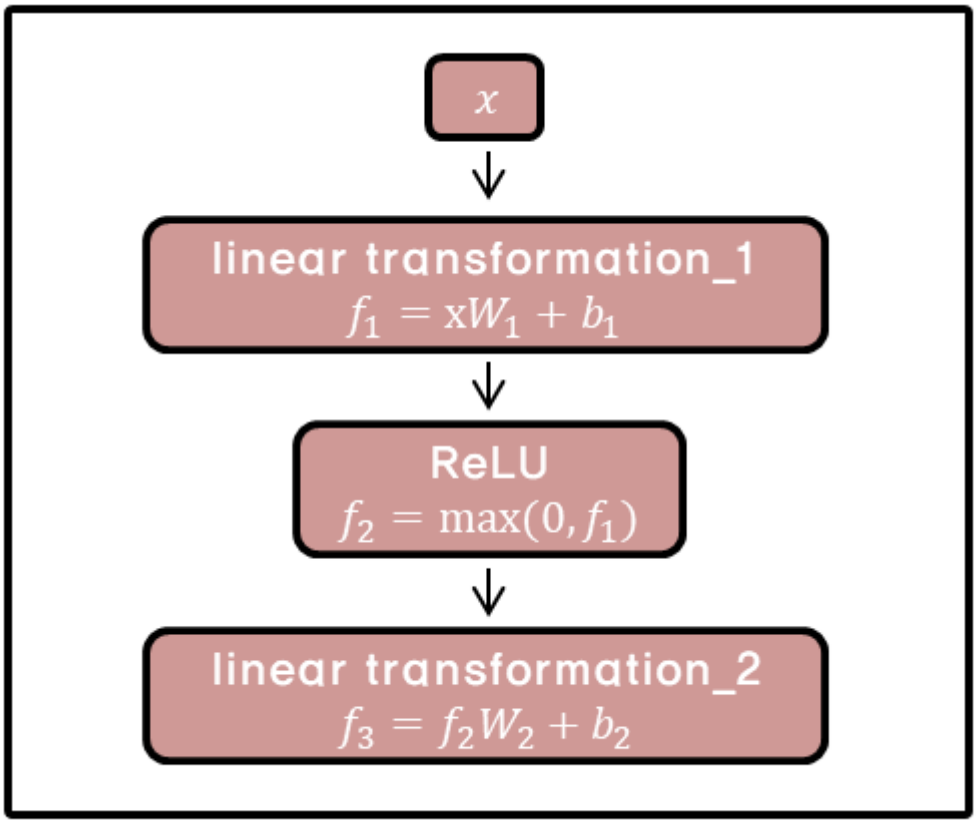
- 위 그림은 헤드가 8개 일때의 같은 내용의 시각화 자료
  - 이제 뭔가 의미를 찾기 어려워짐
  - 그래도 굳이 의미를 부여하자면,.....어느건 street에 어느건 tired에 심지어 어느건 because에 집중하는 것을 볼 수 있음
  - 여기서 나올 수 있는 질문이, 이렇게 너무 많은 부분에 집중하면, 결과적으로 그 어떤 것에도 집중을 하지 않은것 아니냐라는 것
    - 매우 맞는 말,
      - 실제로, 이 헤드의 수는 하이퍼파라미터로 사람이 직접 설정해 줘야함
      - 너무 적으면, 멀티 헤드가 가지는 장점을 줄이고
      - 반대로 너무 많으면, 어탠션이 초점 혹은 논점을 잃고 전반적인 성능의 저하로 이어질 수 도
      - 실제로, 연구자들은 여러개의 헤드수를 가지는 모델로 실험을 한다고

add & norm

- add는 risidual connection으로 multi head attention의 출력값 Z을 multi head attention에 들어가기 직접 값인 X와 더해주는 작업
  - 그 이유는 앞선 단원에서 배운것 처럼, back prop때 정보의 흐름이 잘 유지되게 함이고, 그로 인해 grad vanish 문제를 줄이기 위함
  - 앞서 모든 과정에서 행렬의 shape를 모두 초기 X와 맞췄기 때문에 가능
- norm은 layer normalization으로
  - 각각의 벡터 내부에서 하는거지, 벡터와 벡터끼리의 연산을 하는것은 아님

feed forward (MLP)

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



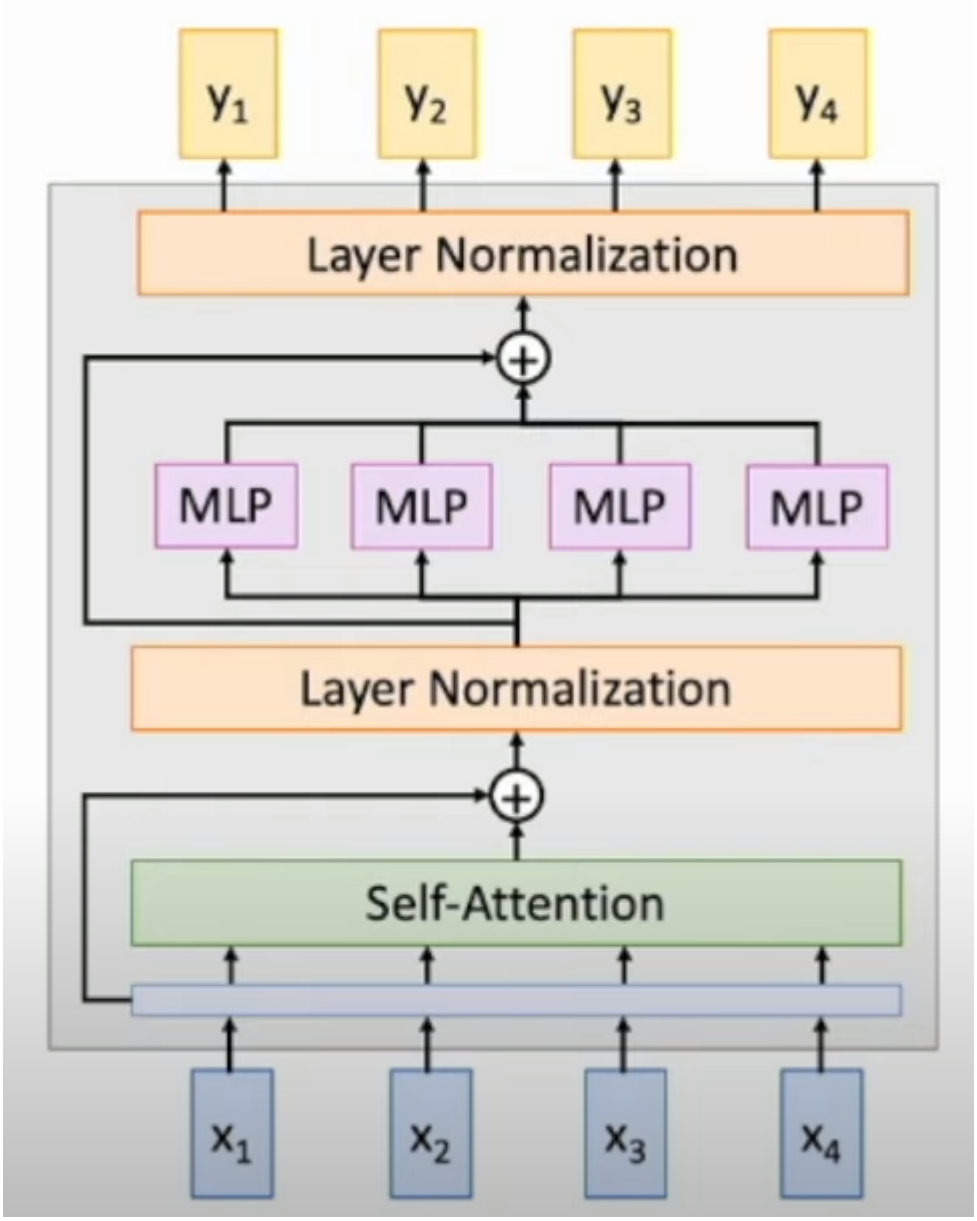
3.3 Position-wise Feed-Forward Networks

In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

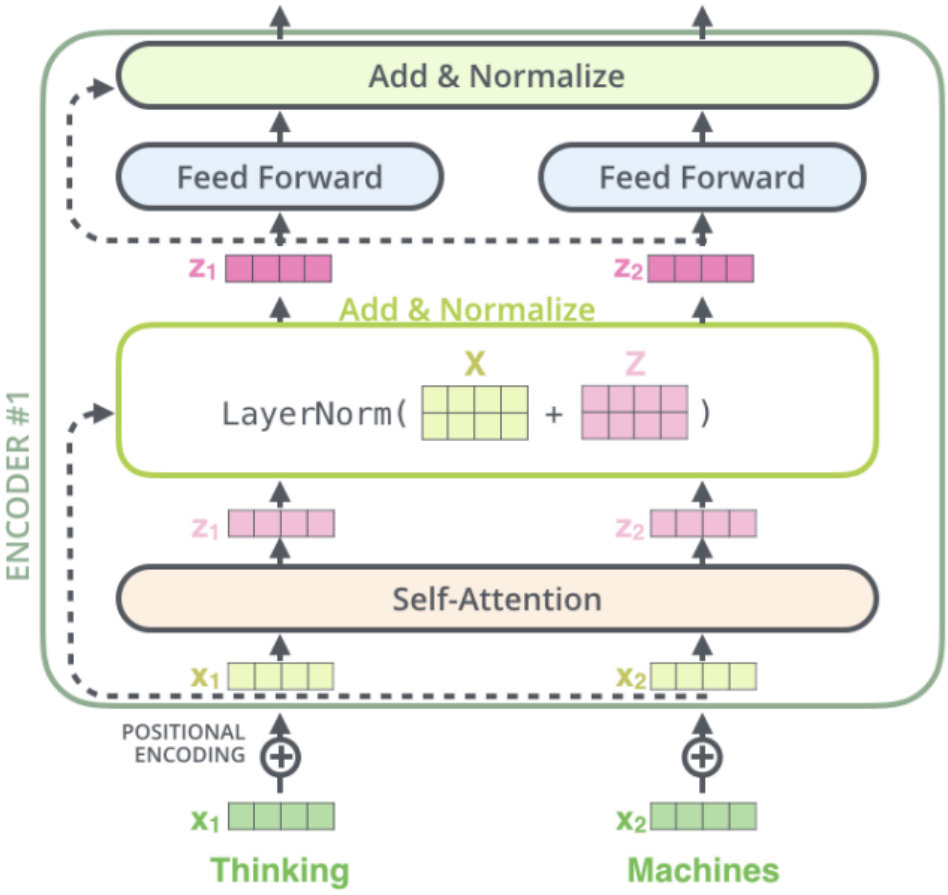
$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{2}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1. The dimensionality of input and output is $d_{\text{model}} = 512$, and the inner-layer has dimensionality $d_{ff} = 2048$.

- 위 부분이 피드 포워드 부분을 설명한 부분인데, 피드 포워드의 필요성에 대해서는 언급을 하지 않고, 단지 어떻게 설정했는지만 설명
    - 우선, 각 포지션 별로 따로, 그리고 동일하게 적용했다



- 이런 그림인데, 보면 입력이 4개이고 MLP도 4개
  - 즉,



- 이렇게 결과인 z를 다시 단어(?), 토큰(?)별로 나눠서 feed forward
- normalization과 마찬가지로 벡터끼리의 상호작용은 없음
- 그말인즉슨, 하나의 블록에서 벡터끼리의 상호작용은 오직 self-attention sub layer에서 발생

◦ feed forward는

- 비 선형성을 추가하여 모델이 더 많은 표현을 가능하게 함
- 멀티 헤드 어탠션에서 나온 다양한 정보들을 합쳐주는 역할도
- 게다가 각각의 요소 하나하나를 따로 처리하기 때문에, 병렬성도 좋음
-

decoder

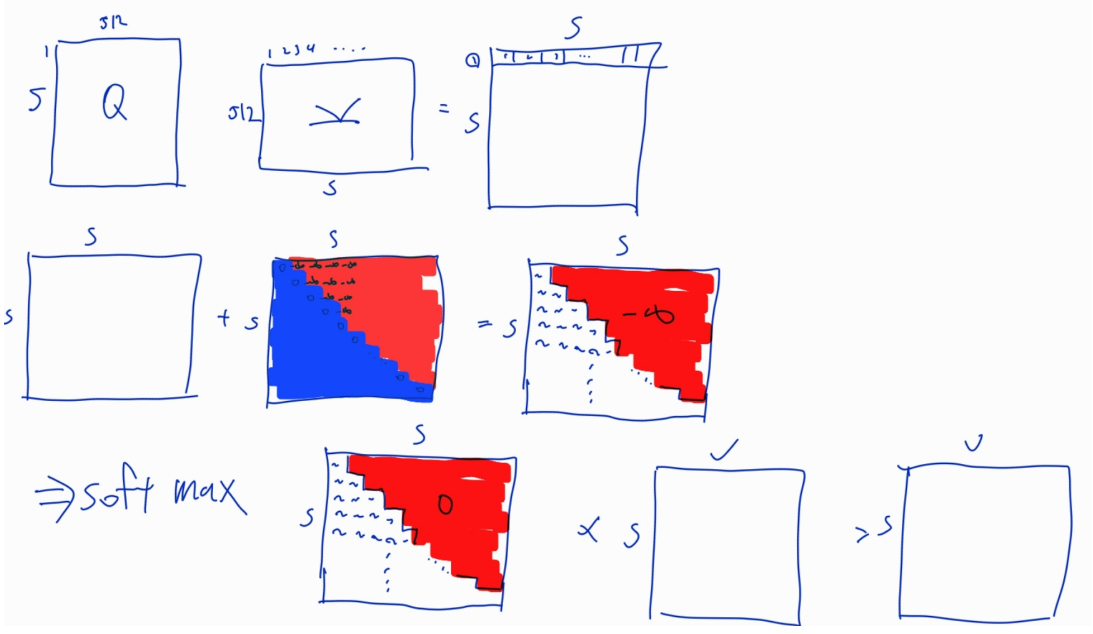
output embedding

masked multi head attention

- 마스크드 멀티 헤드 어탠션은 일반적인 멀티 헤드 어탠션과 같지만, look a head 마스크라는 작업이 추가된 서브 레이어
- 기본적으로, 훈련중의 디코더에서만 작동을 함
  - 디코더는 어차피 시작 토큰 <sos>부터 시작해서, 하나씩 만들어 가면서 다시 입력을 받는데 왜 마스킹이 필요한가
    - 훈련할 때는, 정답 시퀀스를 통 째로 디코더의 입력에 넣기 때문에, 이러한 작업이 필요
    - 즉, 훈련때는, 미래에 올 단어까지 학습을 혹은 주의를 기울이는 우를 범할 수 있기 때문에, 마스킹
      - 이는 아직 오지 않은 부분의 값을 -inf로 하여

> • Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections. See Figure 2.

  - 그리고, 훈련이 아닌, 추론 혹은 예측 작업을 할 때는, 이러한 마스킹 작업이 없다고 함
  - 구체적을 어떻게 마스크하냐

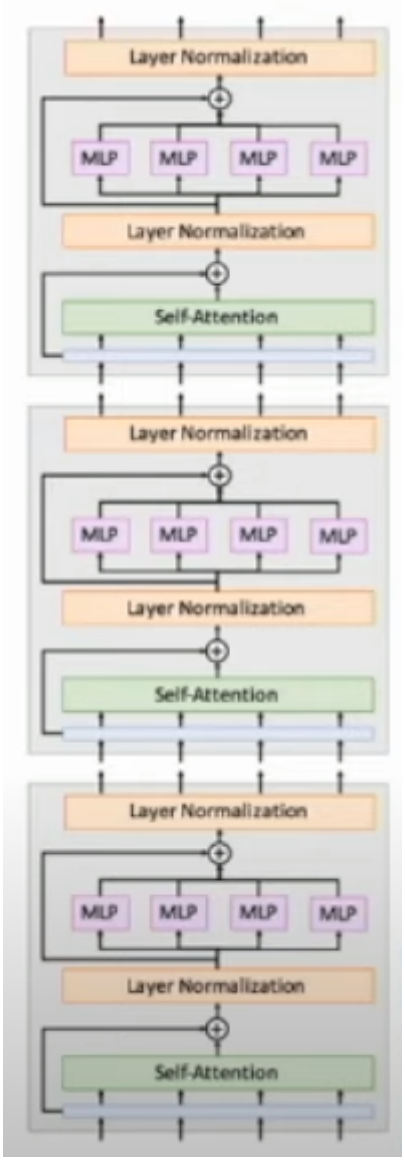


  - 인코더의 attention이나, 디코더의 다른 attention에도 masking 작업은 있음

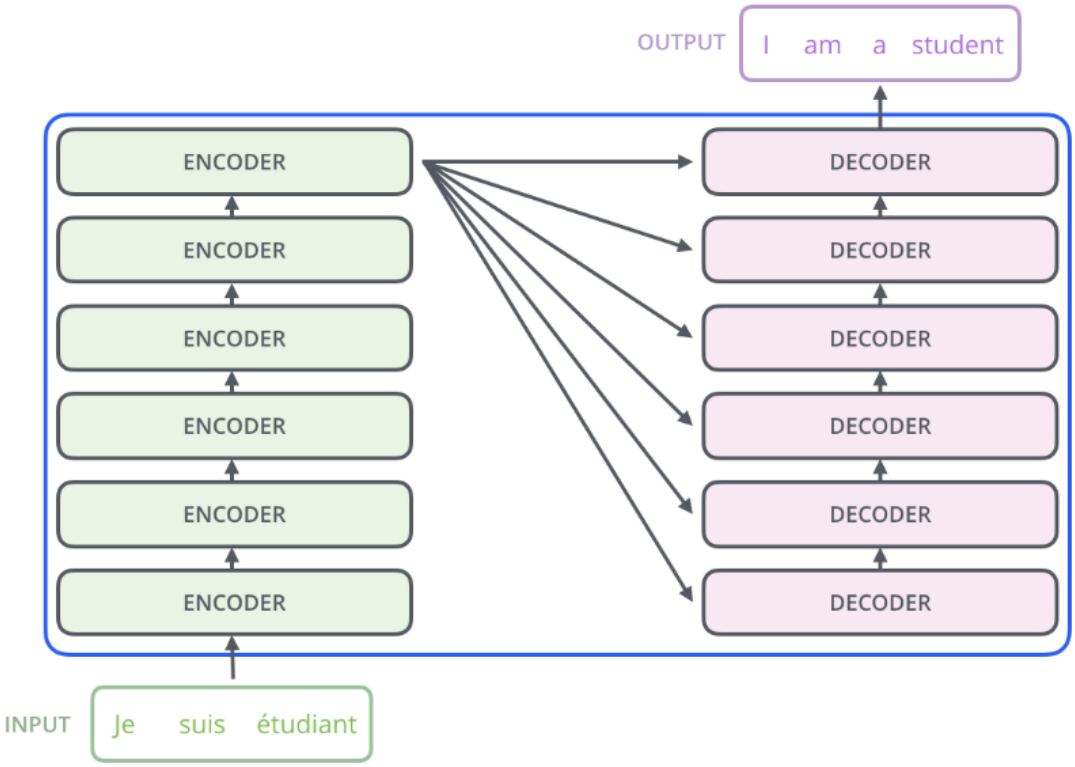
- padding mask라고 하는데,
- 기본적으로 트랜스포머의 입력으로 정해진 시퀀스 행렬을 넣으니, 그 시퀀스 크기보다 작은 길이의 문장을 넣으면, 미처 채우지 못한 빈 공간이 발생,
- 이 빈공간을 패딩 토큰이라는 아무 의미 없지만, 정해진 토큰으로 채움
- 패딩 마스크 역시 비슷한 원리로, Q와 K를 곱한 attention score의 패딩 열을 작은 음수값을 넣어서, 패딩 토큰에 attention하지 않게 조절하는 역할

전반적인 작동 구조

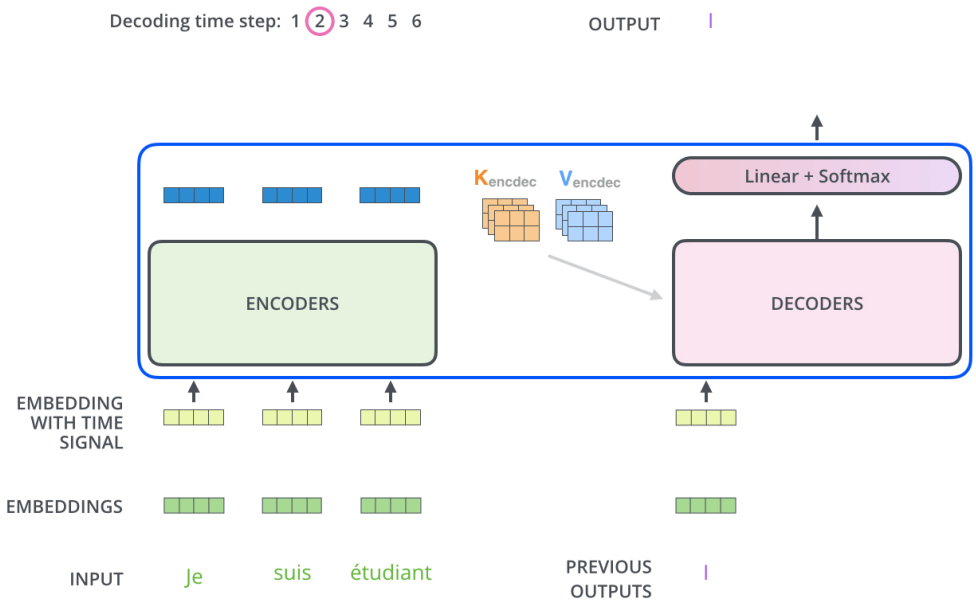
- 논문에는 인코더 6개 디코더 6개를 쌓아 모델을 만들었다고
  - 즉 강의에서 본



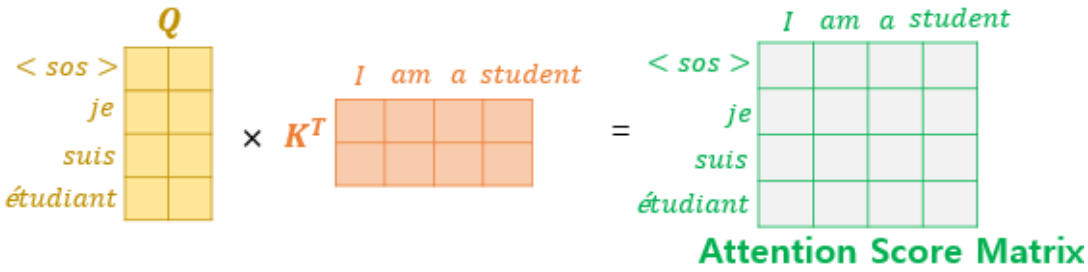
- 위 그림이, 처음에는 트랜스포머의 그림과 너무 달라 이상하게 생각했지만, 위와 같은 구조로 인코더가 6개, 디코더가 6개 있는 것이 트랜스포머
- 즉,



- 이런 식의 구성을 갖게 됨
- 여기서 최종 인코더에서 각각의 디코더로 넘어가는 것이 바로
  최종 인코더의 K값과 V값
- 그리고 K값과 V값을 받는 것은 디코더의 중간에 있는 multi head attention
- 즉,
-



- 디코더의 multi head attention
  - 디코더 중단에 있는 multi head attention은 encoder-decoder cross attention이라고도
  - 셀프 어탠션은 QKV값이 모두 자기 자신한테서 나왔다면
  - cross 어탠션은 QKV값의 원천이 서로 다름
  - 즉, encoder-decoder cross attention은 Q값을 이 전단의 masked multi head attention으로부터 받고, K값이랑 V값은 encoder로부터 받음



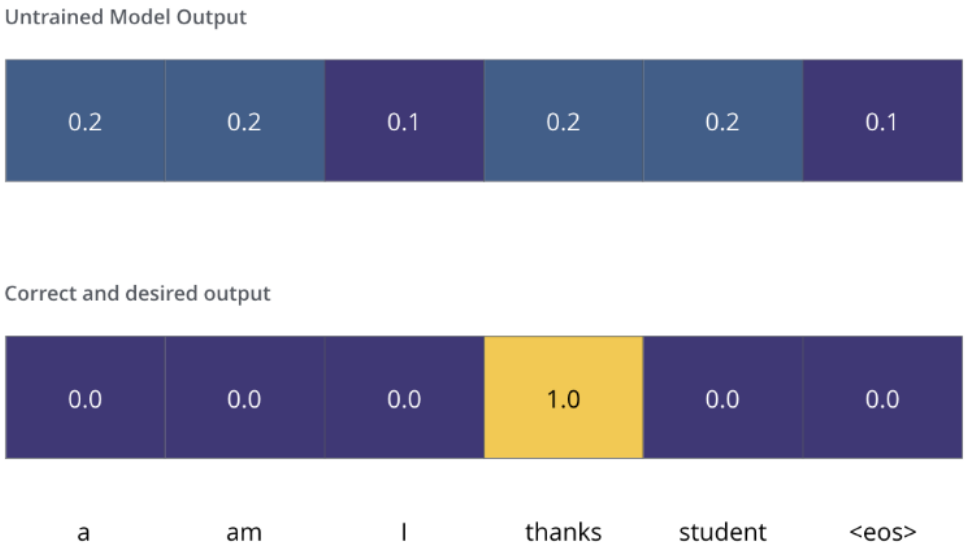
  - 이렇게 하면, 디코더의 Q값은 인코더의 입력 값인 X을 온전히 참조할 수 있게 됨
    - 일단 masked multi head attention의 결과값 그 자체가 이미 전체적인 문장의 의미를 담고있기 때문에, 그대로 Q값으로 사용 가능, 논문에서도 그대로 사용한것처럼보임
    - 근데, 일반적으로 QKV값을 구할때처럼, 추가적인 linear 층을 사용해서 Q값을 새로 도출해 사용하는 것도 유효한 방식

        - 이렇게 Q값 만을 위해서 존재하는 추가적인 선형 계층을 query projection layer라고 함
        - 이렇게 하면, 추가적인 파라미터가 생기고, 모델의 용량이 증가하여, 성능의 향상으로 이어질 수 있지만, 실제로는 데이터나 주어진 작업 환경에 가장 맞는 모델을 실험..
    - 이 cross attention에는 look ahead mask작업이 없음
        - Q값에는 이미 마스킹이 되어 상관없다고 해도 여기에 들어오는 인코더의 K값이랑 V값도 미래의 정보를 담고있는데?
        - 라고 생각할 수 있지만
        - 굳이 예시를 들자면, 인코더에서 오는 K값이랑 V값은 문제에 해당
        - 이 전단에서 오는 디코더의 Q값은 답에 해당
        - ‘나는 집에 가고 싶다’라는 한글 문장과 I want to 라는 영어 문장을 넣어서 go라는 결과를 얻는 것은 자연스럽다.
            - 추가적으로 어순의 문제도 있기 때문에, 인코더에서 오는 KV값은 모두 포함되어야 함
        - 근데, ‘나는 집에 가고 싶다’라는 한글 문장과 I want to go home이라는 영어 문장을 넣어서 I want to 다음에 올 단어는 go라고 추측하는 것은 이상함.
    -
- 역전파
    - 다른 부분의 역전파는 예전에 하는것처럼 하면 되고, 인코더와 디코더 사이의 기울기 전파에 대해
    - 기본적으로 동일한 K랑 V값이 모든 디코더에 전달됨으로, 디코더 하나가 역전파 할 때마다 인코더도 역전파 하는것은 말이 되지 않음
    - 그럼 어떻게 하냐
        - 디코더 하나하나 역전파 할 때마다, K값과 V값의 기울기를 누적해서
          마지막에 그 누적값을 한번에 전달

shift right

최종단의 linear와 softmax

- 최종적으로 출력되는 것은 소수가 담긴 벡터 하나
- 이 벡터 하나를 단어로 바꾸는 부분이 바로 linear와 softmax
- linear
  - 이 리니어 단은, 출력의 차원을 맞추는 역할
  - 마치 cnn에서 마지막의 fc layer 마냥
  - 예를 들어, 학습한 단어의 수가 1만개라면, 리니어 단의 출력 차원은 1만이 되고, 각각의 요소는 해당하는 단어와 연관이 있게 됨
- softmax
  - 위에서 나온 만개의 벡터를 확률로 계산하여, 가장 큰 확률을 가진 단어를 뽑아내는 역할
  -
- 여기서 새로 나오는 개념이
  - label smoothing 과 beam search
  - loss를 구할 때, cross entropy 를 사용
    - 즉, 정답 벡터는 원 핫 벡터라는 뜻



    - 라벨 스무딩은 이렇게 0아니면 1로 이루어진 라벨을
      예를 들면, 0.02 0.02 0.02 0.9 0.02 0.02등으로 바꿔 loss function의 정답 라벨로 사용하는 것

Label Smoothing During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ [36]. This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

- 논문에서는 그냥 이정도 말 밖에 없음
  - 보면 결국에는 결국 정확도와 bleu 점수가 늘었다 정도
- 그래서 따로 찾아보려고 했으나,
  - 찾아 보니 실제로 매우 범용적으로 사용하고 있는 기법임에도, 정확히 어떤 매커니즘으로 성능 향상이 되는지는…
  - 오히려 라벨 스무딩이 왜 성능 향상에 도움이 되는지를 연구한 논문도 여기서 다루기에는 제 역량이 부족해서 그냥 참고할만한 자료만 첨부
  - https://velog.io/@xuio/ML-TIL-Label-Smoothing에-대해-알아보기-feat.-When-Does-Label-Smoothing-Help-논문
  - https://blog.si-analytics.ai/21

- 다음은 beam search
  - 최종단에서 단지 가장 높은 확률의 단어만 출력하는 것이 과연 옳을까?
  - 이전에 알고리즘 스터디에서 한 그리디 알고리즘(즉, 현재 가장 좋은 선택이 전체적으로 봤을 때 가장 좋은 선택인가?)라는 물음을 던졌을 때, 이 번역, 또는 문맥의 추론 영억은 부정적인 대답을 내놓을 것

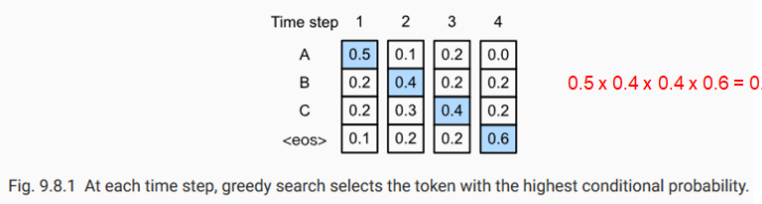


Fig. 9.8.1 At each time step, greedy search selects the token with the highest conditional probability.

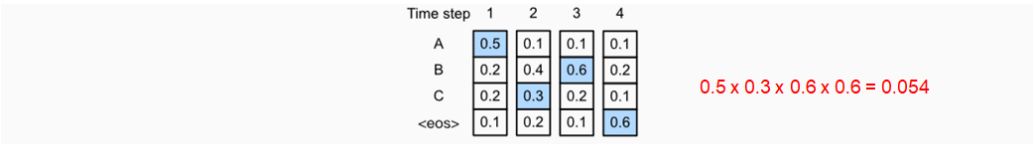
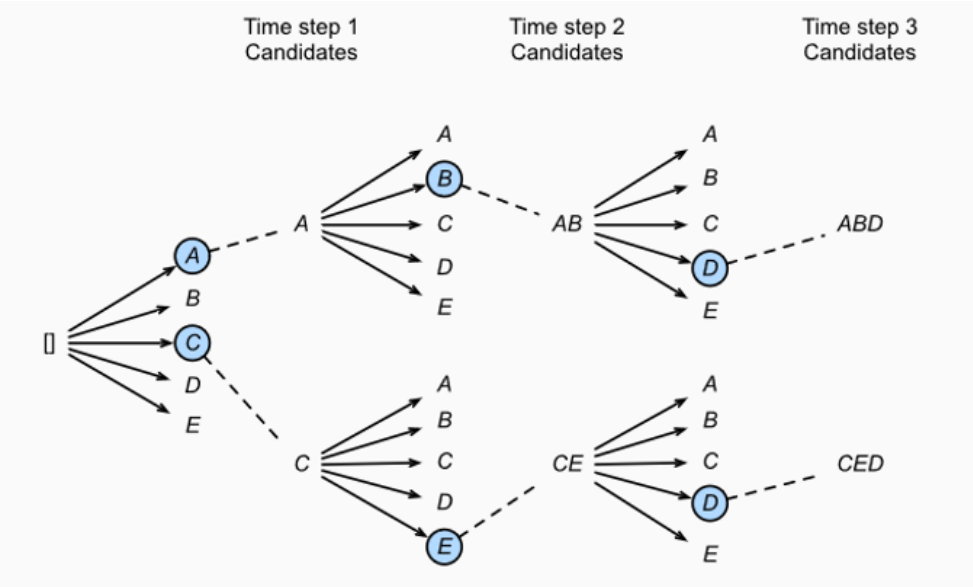


Fig. 9.8.2 The four numbers under each time step represent the conditional probabilities of generating "A", "B", "C", and "<eos>" at that time step. At time step 2, the token "C", which has the second highest conditional probability, is selected.

- 위 그림이 그 예시
  - 첫번째 상황에서는 2번 time step에서 가장 높은 확률을 갖는 B를 선택
  - 두번째 상황에서는 2번 time step에서 두번째로 높은 확률을 갖는 C를 선택

- 이제 다시 디코더의 성질이 나오는데,
  - 디코더는 기본적으로 현재의 선택이 다시 다음 step의 입력으로 들어가기 때문에, 서로 다른 선택을 한 두 상황은 3번째 time step 부터는 다른 확률 벡터를 갖게 됨
  - 그리고 결론적으로 비록 2번 time step에서 가장 높은 확률을 선택하지 않은 두번째 상황이 전체적인 확률은 더 높아짐
- 이러한 문제를 해결하기 위해 나온 방식이 beam search



  - 여기서 beam size K는 하이퍼파라미터
  - 그럼 K가 뭐냐
    - K번째 높은 확률까지는 고려를 하겠다 라는 의미
    - 여기서 주의할 점이, 각각의 가능성을 통해 생성된 사건 모두에 적용하는 것이 아닌, 통합적으로 적용하는 것
    - 예를 들면, 위 그림에서 A와 C를 골랐으면
    - 그 다음 step은 AA, AB, AC, AD, AE … CA, CB, CC, CD, CE 이렇게 총 10개가 나올텐데, 이 10개 중에서 다시 가장 확률이 높은 2개를 고른다는 의미
    -

optimizer

- 옵티마이저는 아담을 사용

regulation

- 드롭 아웃 기법을 사용

강의 내용

https://oopy.lazyrockets.com/api/v2/notion/image?src=https%3A%2F%2Fs3-us-west-2.amazonaws.com%2Fsecure.notion-static.com%2Fc7d5f597-399a-4084-ab8f-1bd8b340892e%2FUntitled.gif&blockId=c3f57ec7-a058-4179-bc78-4ec07183786f

제목 없음