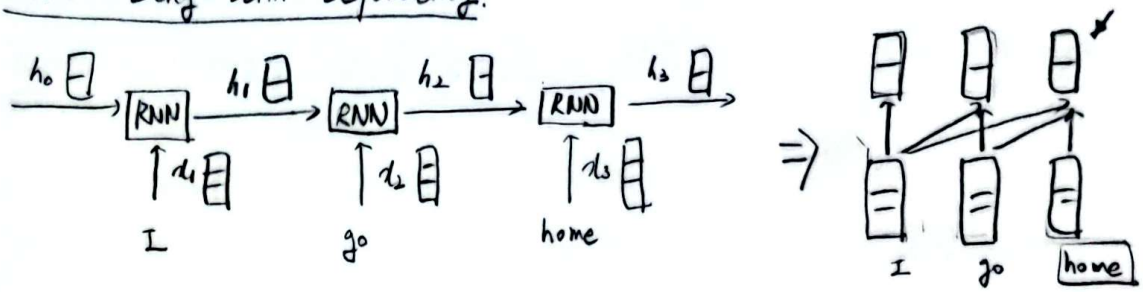
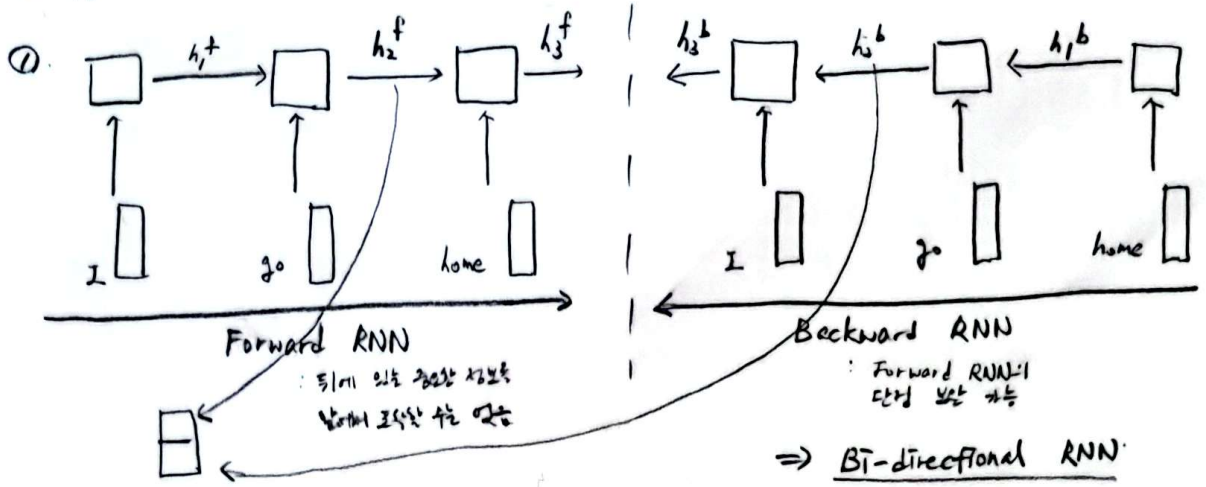
RNN: Long-term Dependency.



문장이 길어지면 먼 곳의 step까지 전달되는 과정에서 정보 유실 발생 가능성 있음

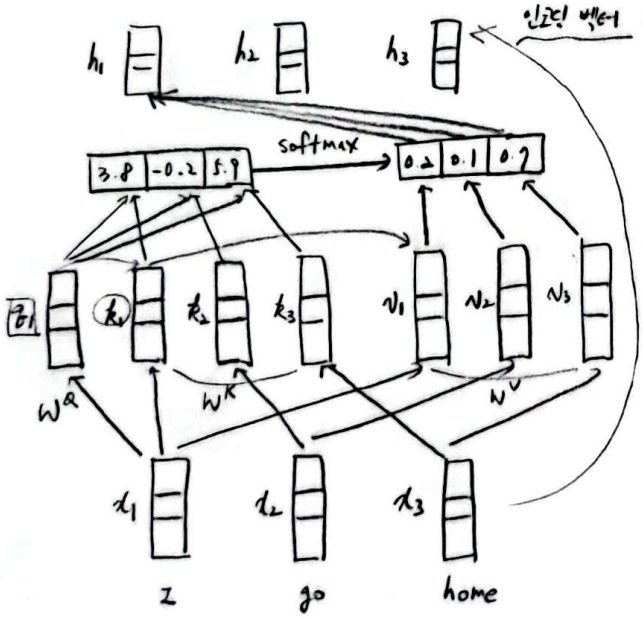
해결책)

① Bi-directional RNN



hidden state를 RNN마다 각각 가져다서 concat을 해서 인코딩 벡터로 쓴다.

② Transformer → sequence 길이가 길어도 time step이 멀리 있는 정보도 손실없이 가져올 수 있음



Decoder hidden state vector 역할

각각의 vector 사이의 유사도를 구하는 데에 사용.

기준 벡터 → Query

(Self-attention으로 Encoder hidden state vector도 동일하게 사용)

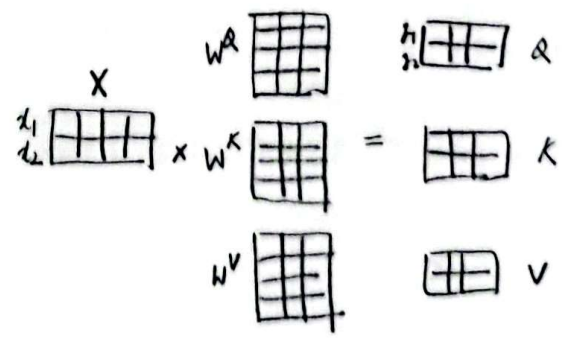
Input	단어1	단어2
Embedding	x_1	x_2
Queries	q_1	q_2
Keys	k_1	k_2

기준 벡터

Input Encoder hidden state vector, Query 벡터와 내적(유사성)을 하는 각각의 재료 벡터

Values	v_1	v_2

각각의 유사도에 softmax 해 나온 가중치를 곱해서 최종 인코딩 벡터에 사용되는 재료 벡터. (Key 벡터와 동일한 개념)



ex)

$x_1 \xrightarrow{W^Q} q_1 \xrightarrow{x_1, x_2, x_3 \text{와 내적, softmax, } v_1, v_2, v_3 \text{와 가중합}} h_1$

Scaled Dot-Product Attention

Inputs: q: query, (k, v): set of key-value pairs (q, k, v는 모두 vector)

Output은 Values의 가중합

각 Value의 가중치는 query와 각 value에 대응하는 key의 내적으로 계산

Query와 Key는 같은 차원 d_k, Value는 d_v (반드시 d_k와 같아야 하는 건 아님)

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k_i)}{\sum_j \exp(q \cdot k_j)} v_i$$

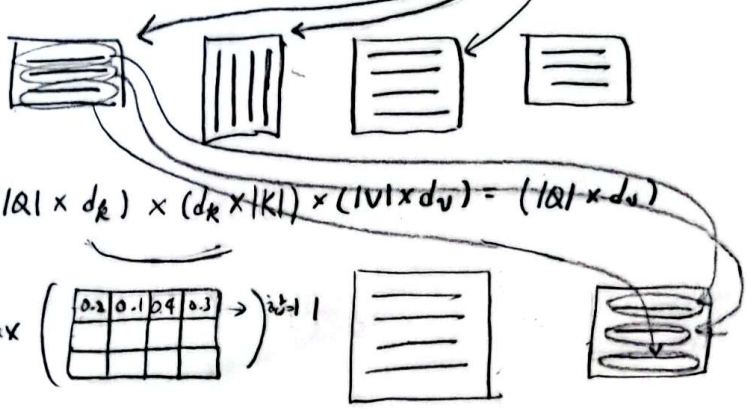
Attention module

softmax

각 원소로 row vector의 내적 연산

만약 query가 여러 개일 때, $A(Q, K, V) = \text{softmax}(QK^T)V$

$K =$

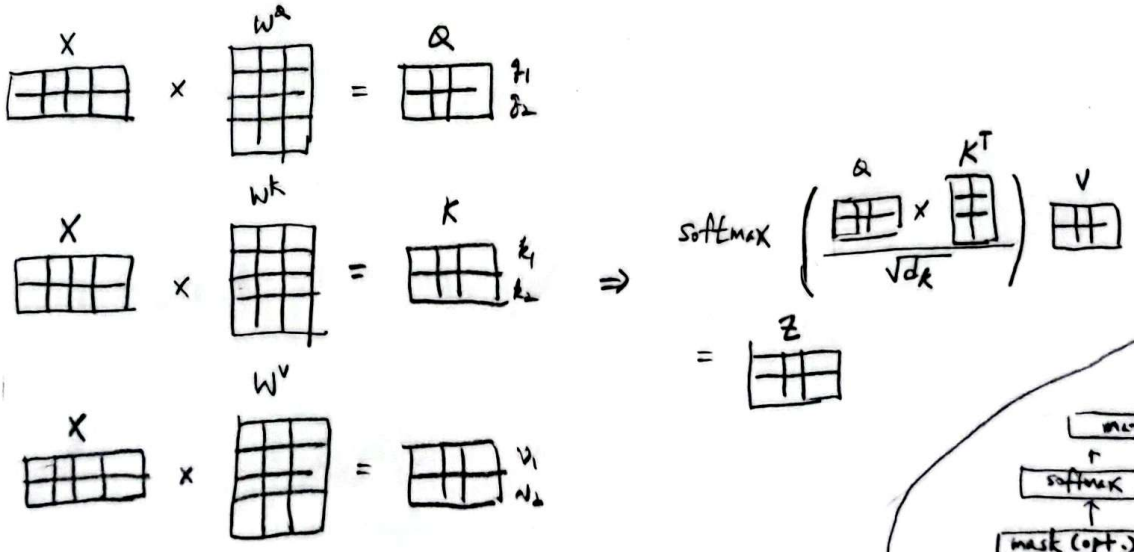


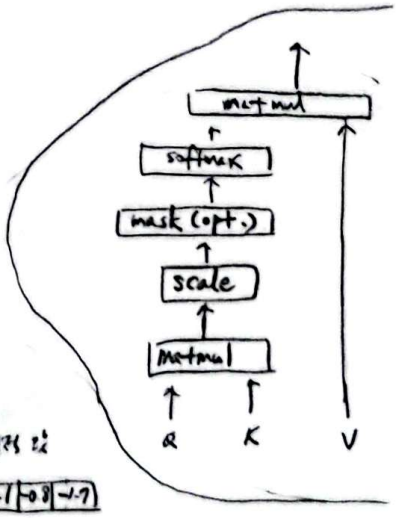
$(|Q| \times d_k) \times (d_k \times |K|) \times (|V| \times d_v) = (|Q| \times d_v)$

(row-wise) softmax $\left(\begin{array}{|c|c|c|c|}\hline 0.2 & 0.1 & 0.4 & 0.3 \\ \hline \end{array} \rightarrow\right)$ 합이 1

상세 Transformer 구현 상으로는 동일한 shape로 mapping된 Q, K, V를 사용하여 각 matrix의 shape는 모두 동일

ex)

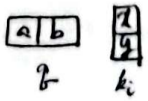




문제점)

d_k가 증가하면 QK^T의 분산이 증가

일부 값이 커질 수 있으며, softmax 값이 한 쪽으로 몰릴 수 있고, 학습 시에 gradient 값이 작아져 가중치 소멸 가능성이 커진다



$= ax + by$

a, b, x, y를 평균: 0, 분산: 1인 확률변수라 할 때 (서로 독립)

$ax + by$의 평균: 0, ax의 분산: 1, by의 분산: 1

$\rightarrow ax + by$의 분산: 2 $\rightarrow$ 내적 값 | 1.1 | -0.8 | -1.7 |

만약 각각 100차원이라면 평균: 0, 분산: 100

$\rightarrow$ 표준편차: 10. $\rightarrow$ | 8 | -11 | 7 |

최종적으로)

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$ $\rightarrow$ 분산을 일정하게 유지

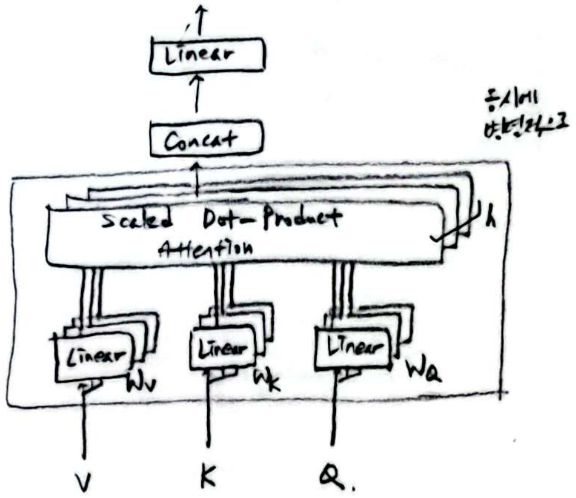
Multi-Head Attention

동일한 sequence가 주어져도 특정한 query word에 대해 서로 다른 기준으로 여러 측면에서의 정보를 추출해야 하는 경우가 있을 수 있다.

ex) 여러 문장으로 이루어져 있지만, 여기선 주어진 하나의 sequence로서 볼 수 있는

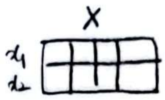
- I went to school
- I studied hard.
- I came back home.
- I took the rest.

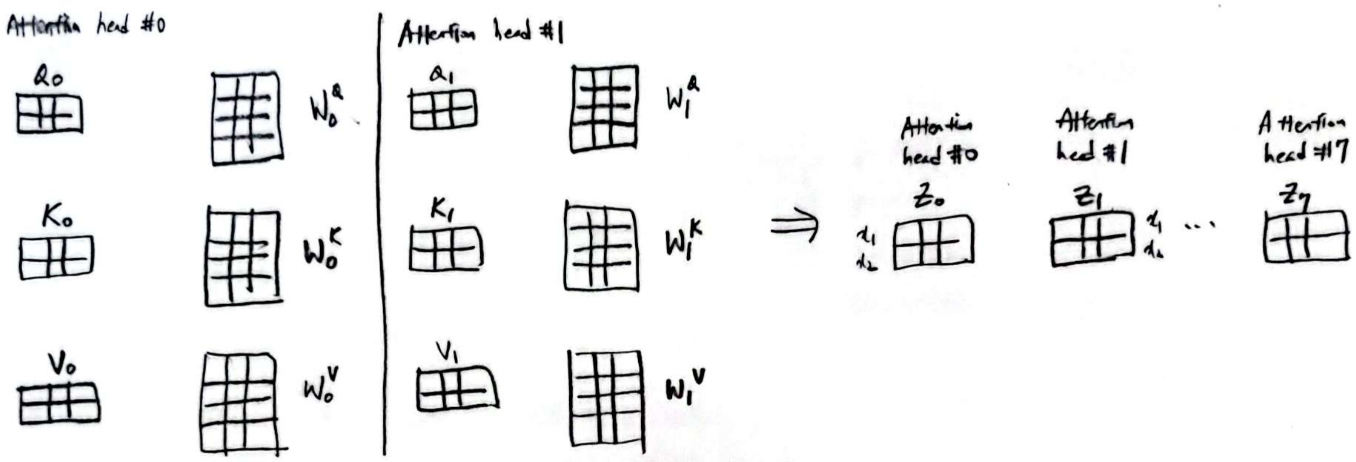
여기서는 'I' query word에 대한 인코딩을 수행하기 위해서는 ① 'I'가 한 행동을 중심으로 ② 'I'가 있던 장소의 변화를 뽑을 수 있다.

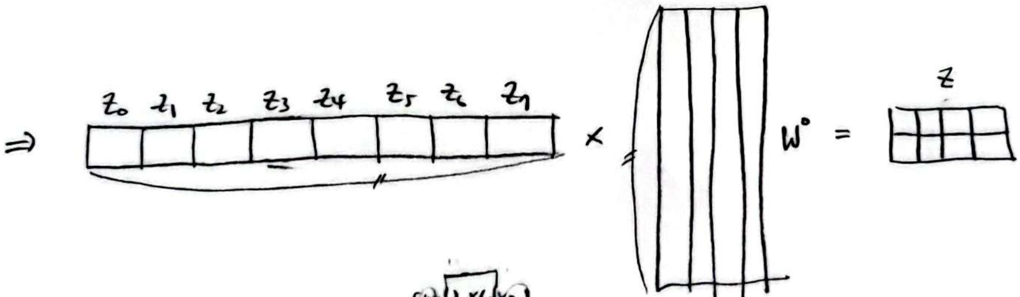


$$\text{Multihead}(Q, K, V) = \text{Concat}(\text{head}_1, \cdots, \text{head}_h) W^O$$

단, $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^W)$







$(n \times d) \times (d \times n)$

$\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$

long-term dependency와 직접적 관련

	Complexity per layer	Sequential operations	Maximum path length
Self-attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$

long-term dependency 해결.

- n: sequence 길이
- d: vector의 차원 (각각의 query, key의 dim)
- k: convolution의 kernel size
- r: restricted self-attention에서 neighborhood의 크기

n: 제한 수 없는 가변적 값

d: 정할 수 있음

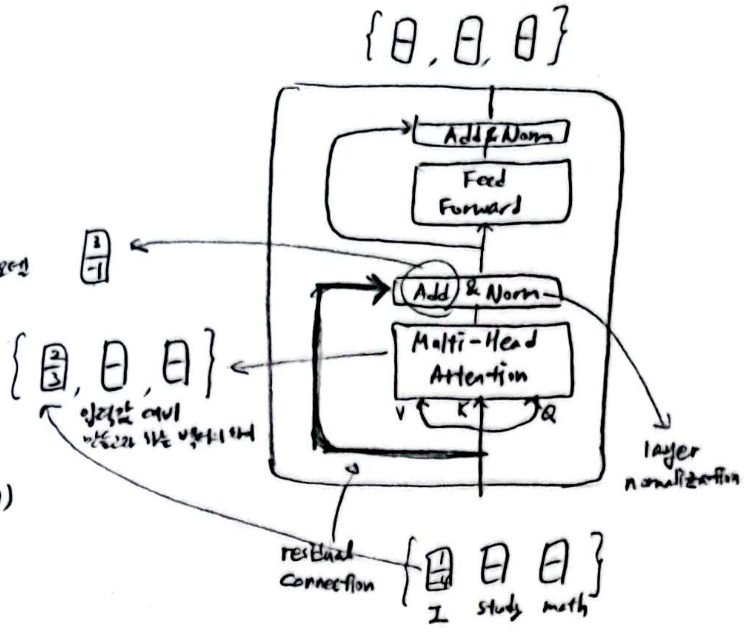
self-attention은 Recurrent에 비해 계산량이 많고 메모리 요구량이 크다.
self-attention은 gpu 코어만 충분하면 병렬처리가 가능하지만
recurrent는 무조건 순차적으로 진행해야 한다 (병렬화 불가)

Block-Based Model

Residual connection

→ CV 분야에서 깊은 layer의 NN(Neural Net)을 만들 때, Gradient Vanishing 문제를 해결하여 학습을 안정화시키면서 layer를 쌓아감에 따라 더 좋은 성능을 보일 수 있도록 하는 모델의 보편

입력 벡터와 Encoding의 output에 대응하는 출력벡터의 차원이 동일해야 적용 가능



각 Block는 2개의 sub-layer
- Multi-head attention
- 2-layer feed-forward NN (with ReLU)

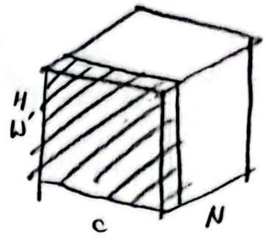
각각은 추가로 2개의 step
- Residual connection, Layer normalization
- LayerNorm(x + sublayer(x))

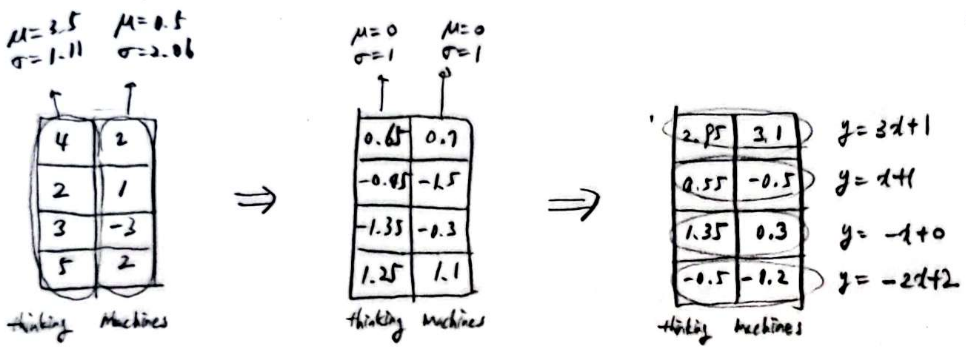
을 가진다.

Layer Normalization

→ 각 layer에서 input의 평균: 0, 분산: 1이 되도록 한다

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l, \quad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}, \quad h_i = f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)$$





① 각 word vector를 평균 0, 분산 1이 되도록 normalization

② 각 sequence vector에서 학습 가능한 parameter에 affine transformation 수행 (원하는 평균, 분산을 주입하기 위함) ← $y = ax + b$

Positional Encoding

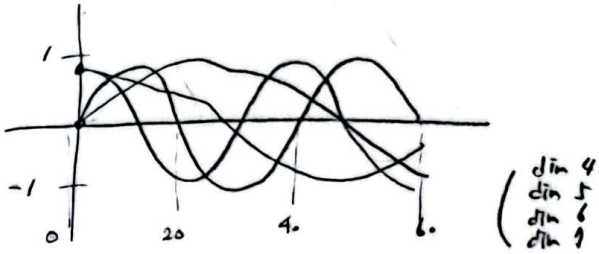
RNN과 달리 self-attention은 고정벡터의 집합으로 순서를 무시한다
즉, sequence를 고려해서 인코딩해 문장 내의 집합처럼 인코딩하게 된다
이를 보완하기 위해 positional encoding이 나왔다

→ 서로 다른 순서 간에 구별이 되는 특정한 상수벡터를 입력 벡터에 더한다
sin, cos 등의 주기함수를 사용하고, 주기를 다르게 해서 여러 순열을 만든다

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

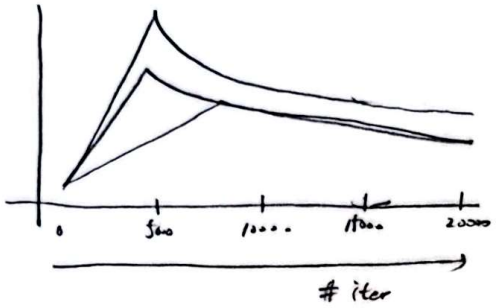
$$PE_{(pos,\,2i+1)} = \cos\left(pos / 10000^{2i/d_{model}}\right)$$

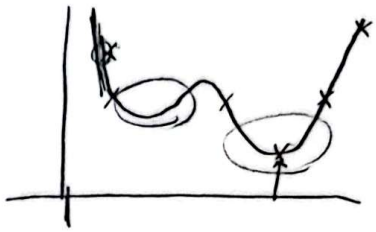
→ dim에 따라 주기가 달라진다



Learning Rate Scheduler

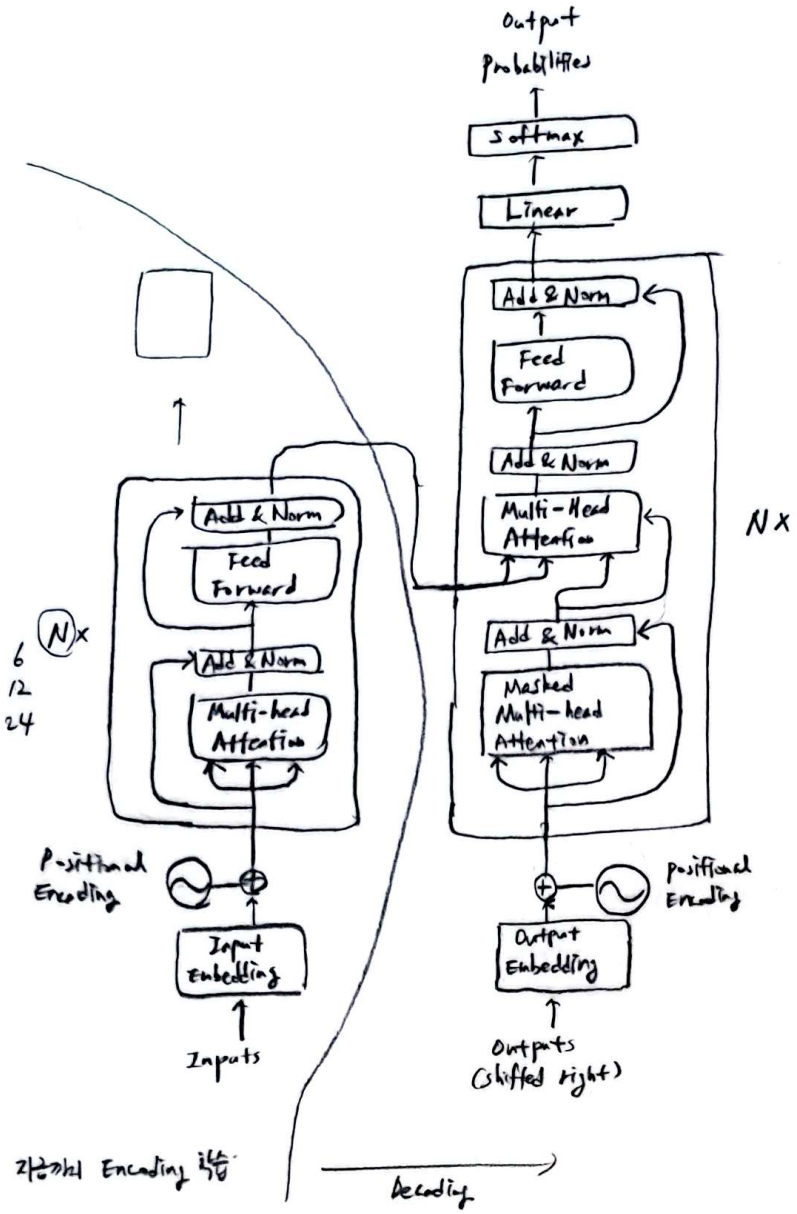
learning rate를 학습 중에 적절히 변경하는 것 임





$$learning\ rate = d_{model}^{-0.5} \cdot \min\left(\#step^{-0.5},\ \#step \cdot warmup\_steps^{-1.5}\right)$$

초반에는 너무 많은 보폭이 발생해서 강도록 lr을 작게 한다
어느 정도 step이 진행되어 안정화되면 실제 모델에서 하는 학습에서 아직 먼 수 있어서
동일을 주기 위해 lr을 iteration 수에 비례해서 올라가게 된다
그러다 학습이 충분히 모델하면 (loss가 크면 수렴하지 않고 멀 수 있으므로) lr을 점차 줄여준다


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-head Attention
Nx
6
12
24
Positional Encoding
Input Embedding
Inputs
Positional Encoding
Output Embedding
Outputs (shifted right)
각각의 Encoding 층
Decoding