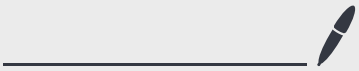
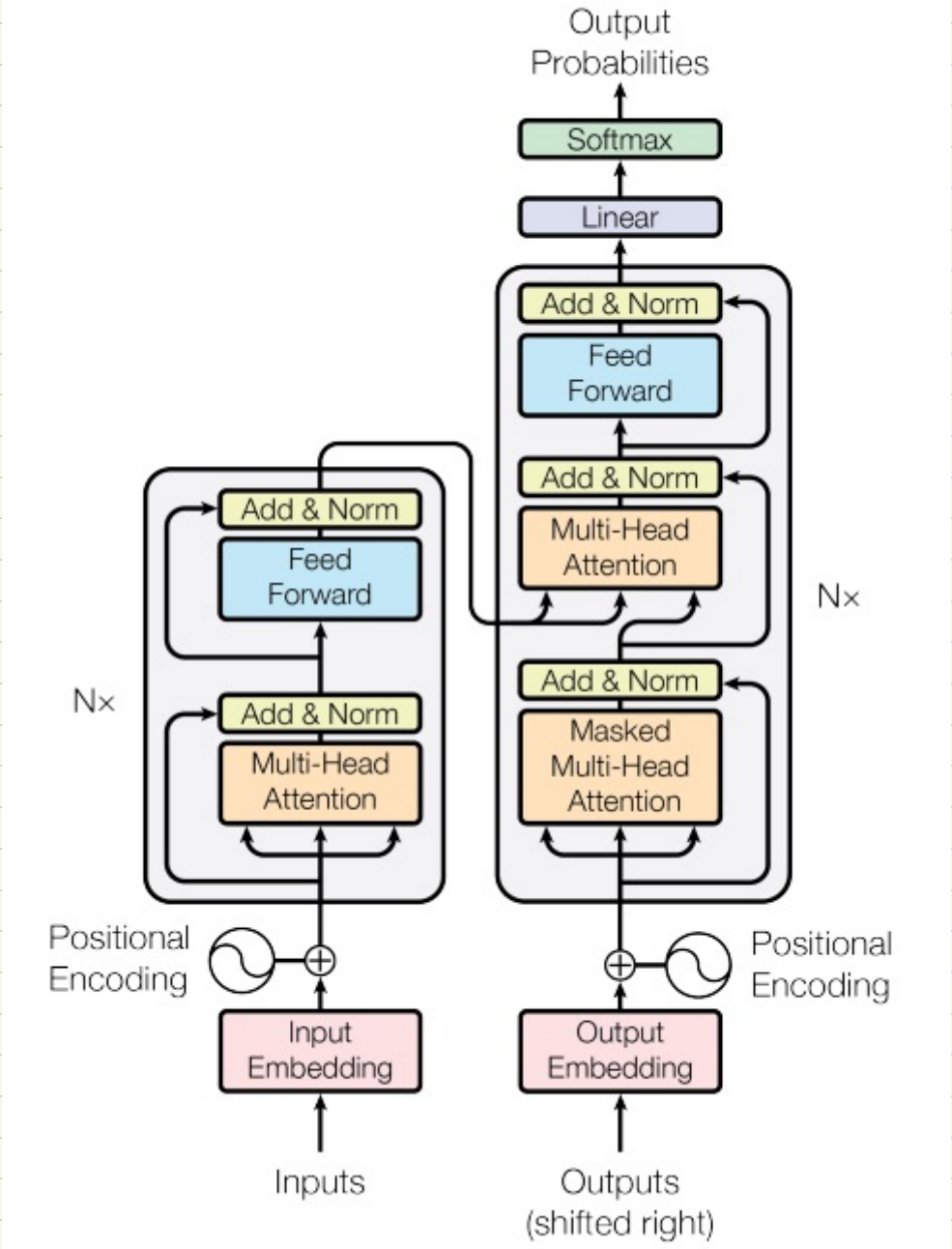



Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

훈련의 목표 : 입력 문장 + 출력 문장을 받아서 출력 문장의 각 위치에서 다음 토큰을 예측

입력 문장

: BOS + I + like + apples + EOS + (PAD)

출력 문장

: BOS + 나는 + 사과를 + 좋아한다 + EOS

model에 input으로 들어감

정답으로 쓸 Label

Encoder

하나의 batch별로 처리됨 B : 배치 크기

초기 입력 : (B, S) S : 배치 내에서 문장의 최대 길이

token index가 저장 ↔ 짧은 애들은 PAD로 채움

1. 임베딩

(B, S, d-model) 각 토큰이 d-model 길이의 embedding vector로 변환

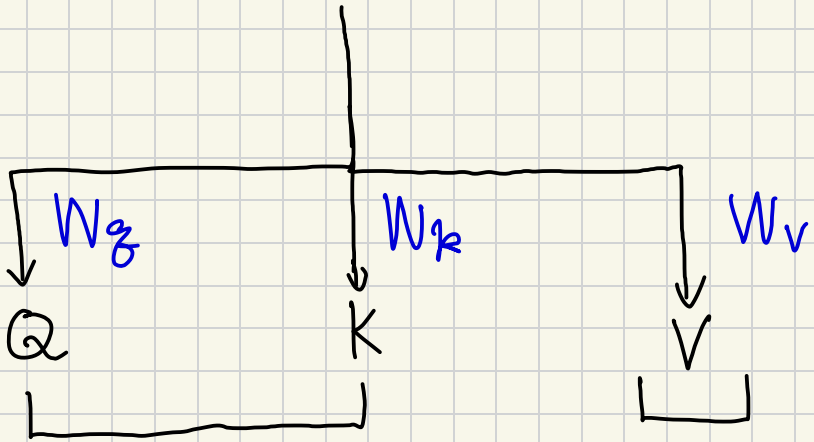
2. Positional Encoding

(B, S, d-model) + (1, S, d-model)

문장에서 token의 위치 pos에 따라 각 token의 embedding vector에 positional vector를 더함

3. Multi-head attention (Enc self-attn)

start (B, S, d_model)



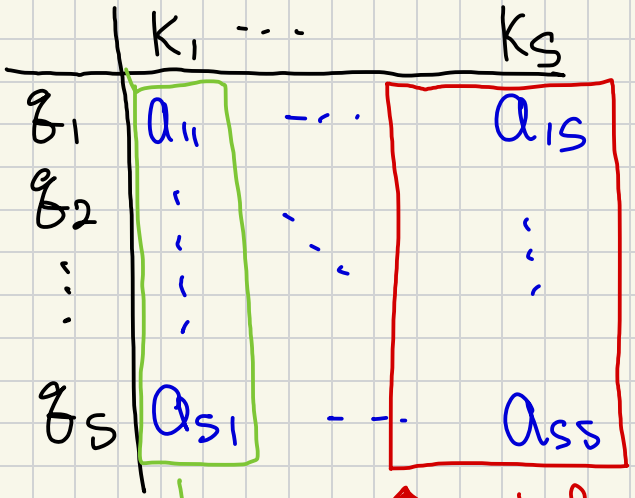
(B, S, n_head * d_k) (B, S, n_head * d_v)

↓ transpose

(B, n_head, S, d_k) (B, n_head, S, d_v)

$$attn = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

앞의 (B, n_head)는 유지된다고 생각



q_i, k_j는 길이 d_k의 vector

QK^T : (B, n_head, S, S)

-inf → (B, 1, 1, S)

mask를 통해 PAD와의 attention은 영향 안 주도록

이 방향으로 Softmax

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

(B, n_head, S, S) (B, n_head, S, dv)

⇓

(B, n_head, S, dv)

$q_1 = a_{11}V_1 + a_{12}V_2 + \cdots$ (PAD 전까지)

$q_2 = a_{21}V_1 + a_{22}V_2 + \cdots$ (〃)

(B, n-head, S, dv)

↓ transpose

(B, S, n-head * dv)

↓ W

(B, S, d-model) end

⇒ 각 token의 새로운 embedding vector

마지막으로 이 결과를 이전의 embedding

과 더해주면 끝 (start + end)

↳ 이 과정이 Residual

정규화는 생략

4. FFN (Feed-Forward Network)

(B, S, d_model)

↓ W_1

(B, S, d_hid)

↓ Relu

(B, S, d_hid)

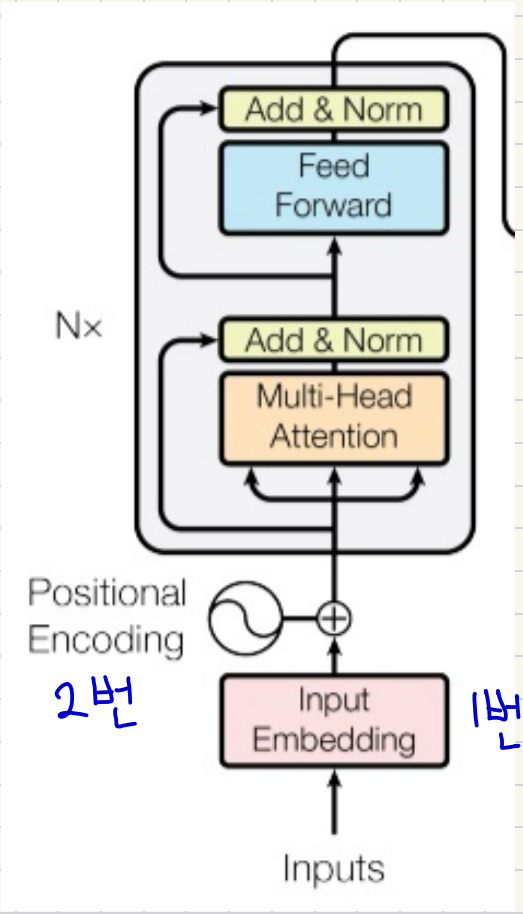
↓ W_2

(B, S, d_model)

FFN의 의의: attention에서 주지 못하는 비선형 학습관계를 제공

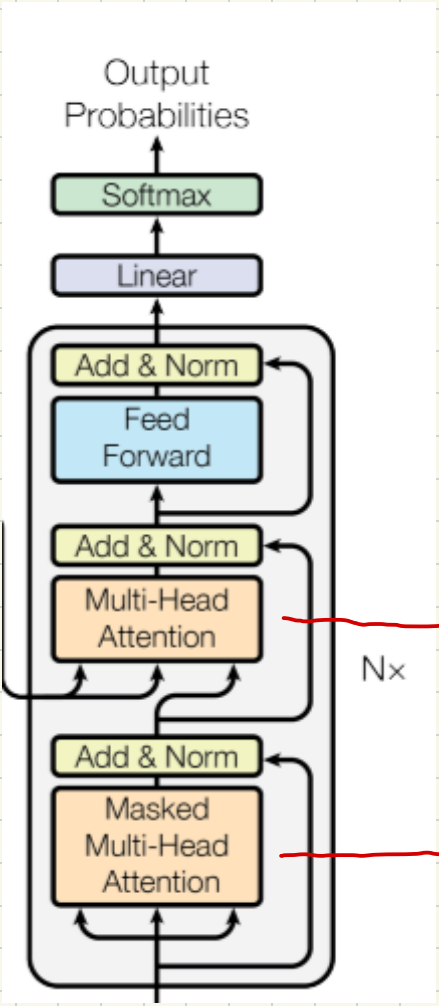
여기에 정규화 + Residual

최종 output : (B, S, d_model)



Decoder

Embedding과 Positional Encoding은 동일하니까 제외



Enc-Dec attention

Dec self-attention

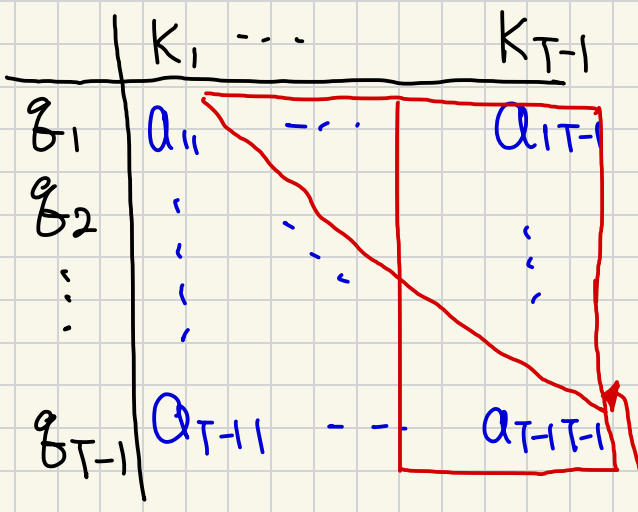
두 attention을 중심적으로 보자

1. Dec-self attention

Q : (B, n_head, T-1, d_k)

K : (B, n_head, T-1, d_k)

V : (B, n_head, T-1, d_v)



q_i, K_j는 길이 d_k의 vector

QK^T

: (B, n_head, T-1, T-1)

- inf

PAD mask + q_i 입장에서 미래 시점의 토큰인 K_j는 참조 불가능하도록 함 ($j > i$)

mask : (B, 1, T-1, T-1)

나머지는 Enc self-attention과 동일함

Output은 역시 (B, T-1, d_model)

2. Enc-Dec attention

Dec embedding
(B, T-1, d_model)

W_q, transpose

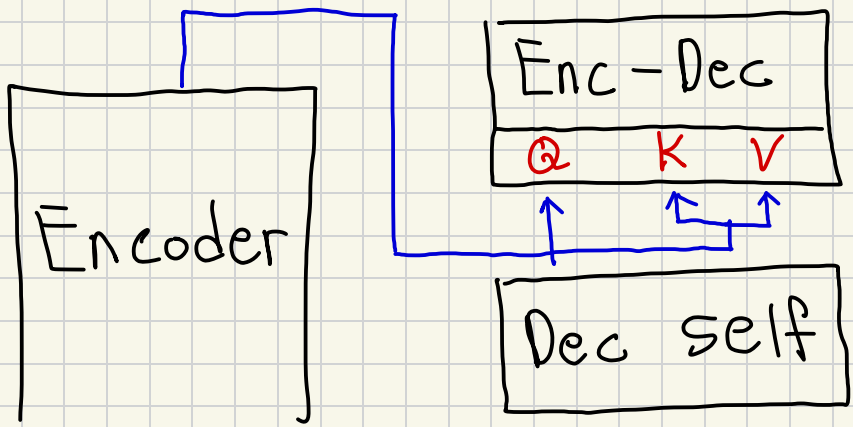
Q

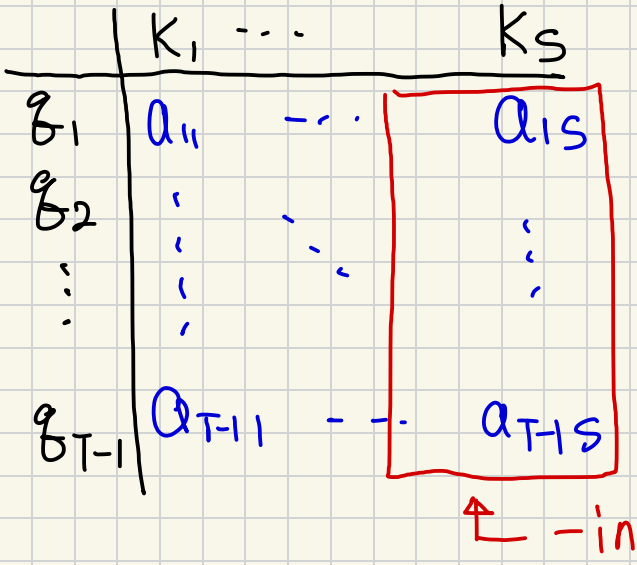
Enc output
(B, S, d_model)

W_k transpose W_v

K V

Q : (B, n_head, T-1, d_k)
K : (B, n_head, S, d_k)
V : (B, n_head, S, d_v)





q_i, k_j는 길이 d_k의 vector

QK^T

: (B, n-head, T-1, S)

마찬가지로 PAD인 key는 attention 계산에 반영하지 않음

(Query쪽을 안 막는 이유는 어쨌피 PAD의 embedding은 쓰이지 않기 때문)

mask: (B, 1, T-1, S)

역시나 Output은 (B, T-1, d-model)

(Decoder쪽 token의 embedding vector를 업데이트하는 과정이므로)

Residual을 쓰는 이유 (뇌피셜)

우리는 Enc-Dec attention에서

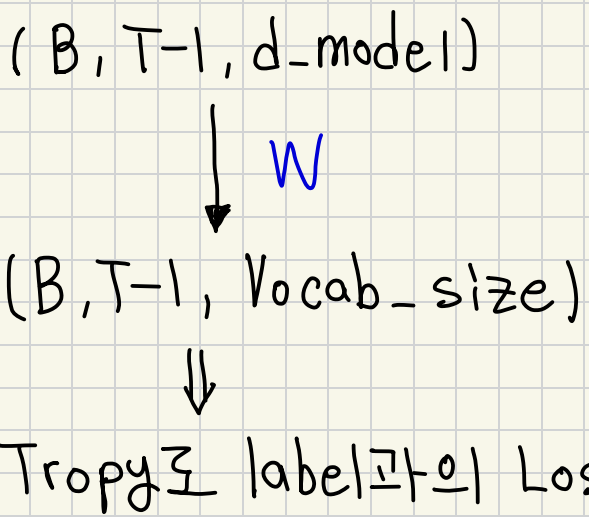
$g_1 = a_{11}V_1 + a_{22}V_2 + \cdots$ 처럼 decoder쪽 token의 embedding을 encoder output으로 만들어진 V_i들의 선형결합으로 바꾸고 있다

=> 원래 g_i이 지니던 decoder token 간의 관계를 나타내던 embedding은 사라지는 느낌 (가중치에만 반영되므로)

=> Residual로 이를 살려준다

최종 Loss 계산

Decoder에서 최종 Output : (B, T-1, d_model)



⇓

Cross-EnTropy로 label과의 Loss를 계산