Distinguished Engineering

Transformer 3/5
- Transformer, a new hope
...

BW

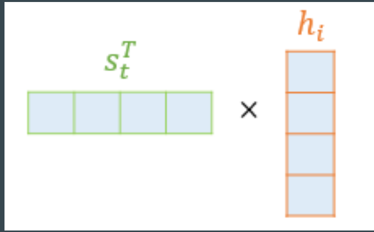
Plan

- Prologue, seq2seq
- **Attention, please**
- Transformer, a new hope
- Transformer, revenge of the fallen
- Transformer, vision

Attention

- Dot product attention

Attention score → Softmax → Context vector → Predict



$$score(s_t,\ h_i) = s_t^T h_i$$

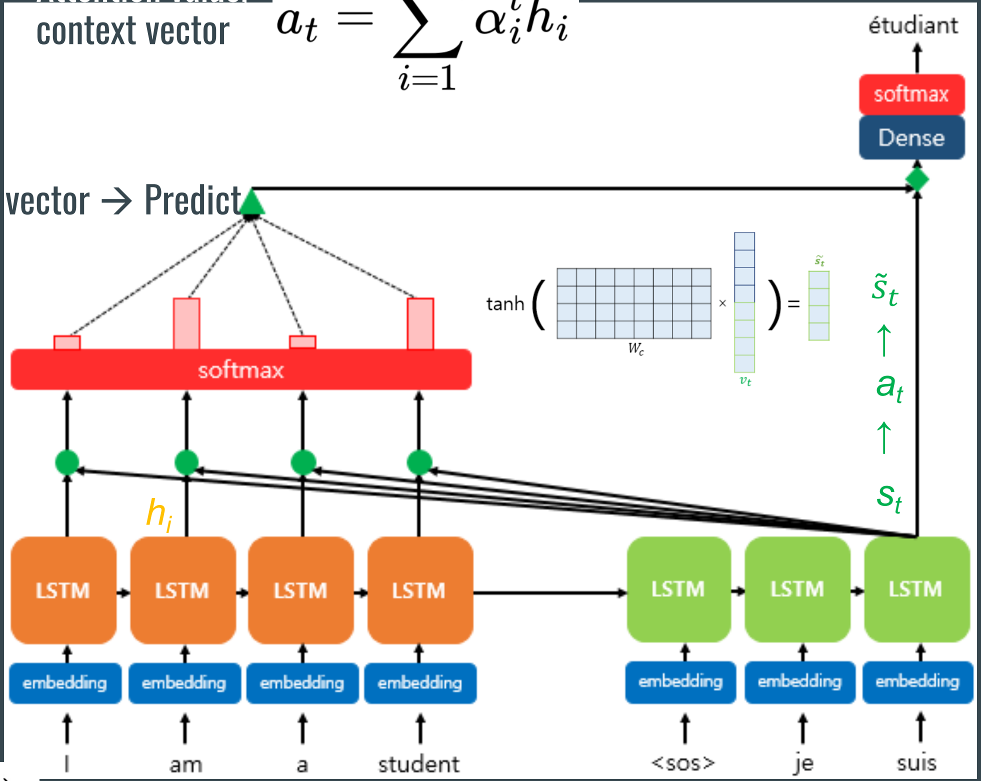
$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

$$\alpha^t = softmax(e^t)$$

$$\tilde{s}_t = \tanh(\mathbf{W_c}[a_t; s_t] + b_c)$$

Attention value, context vector

$$a_t = \sum_{i=1}^{N} \alpha_i^t h_i$$

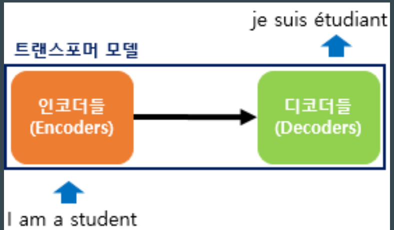


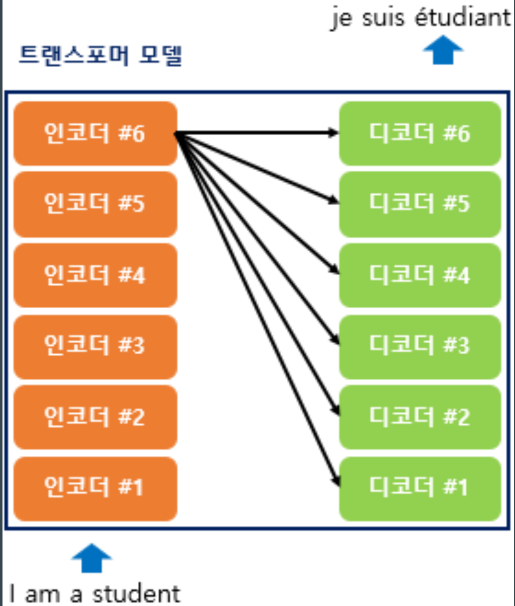
Transformer

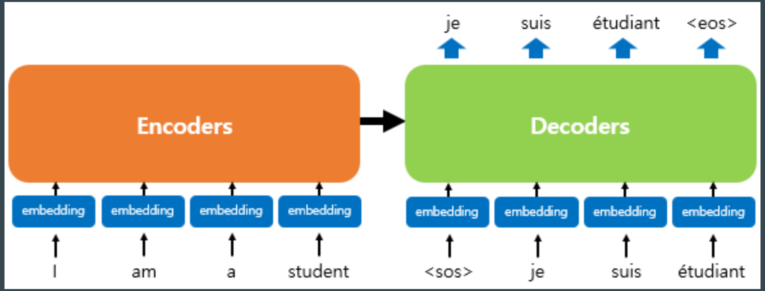
- Hyper parameters
  - 입출력 크기 $d_{model} = 512$
  - Layer 수 $\mathrm{num\_layers} = 6$
  - Head수 $\mathrm{num\_heads} = 8$
  - FFNet hidden dimension $d_{ff} = 2048$

Transformer

- Attention is all you need
  - https://arxiv.org/abs/1706.03762

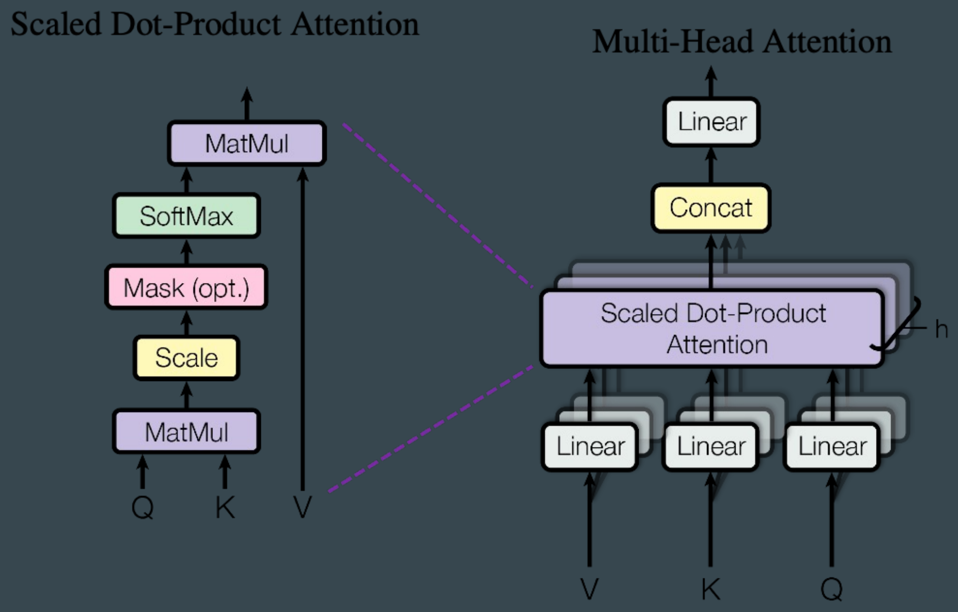


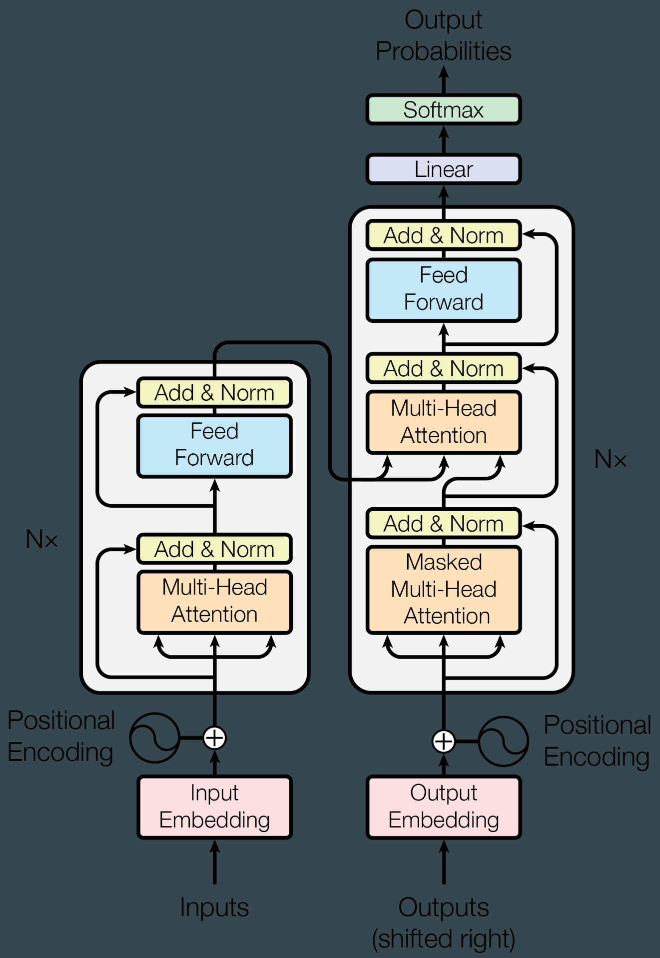




Transformer

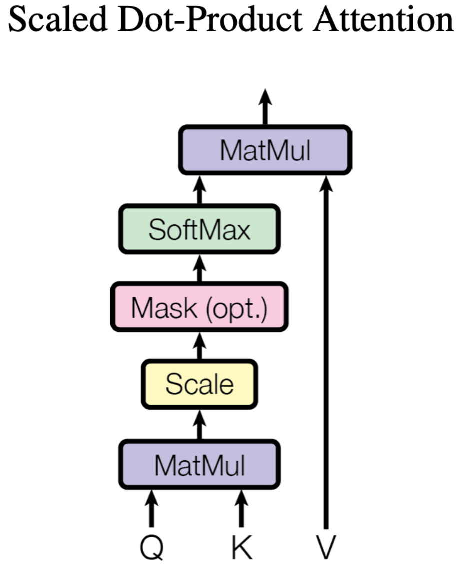
- Attention is all you need
    - https://arxiv.org/abs/1706.03762

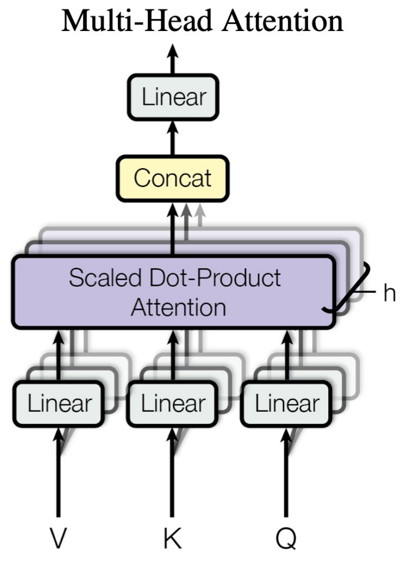


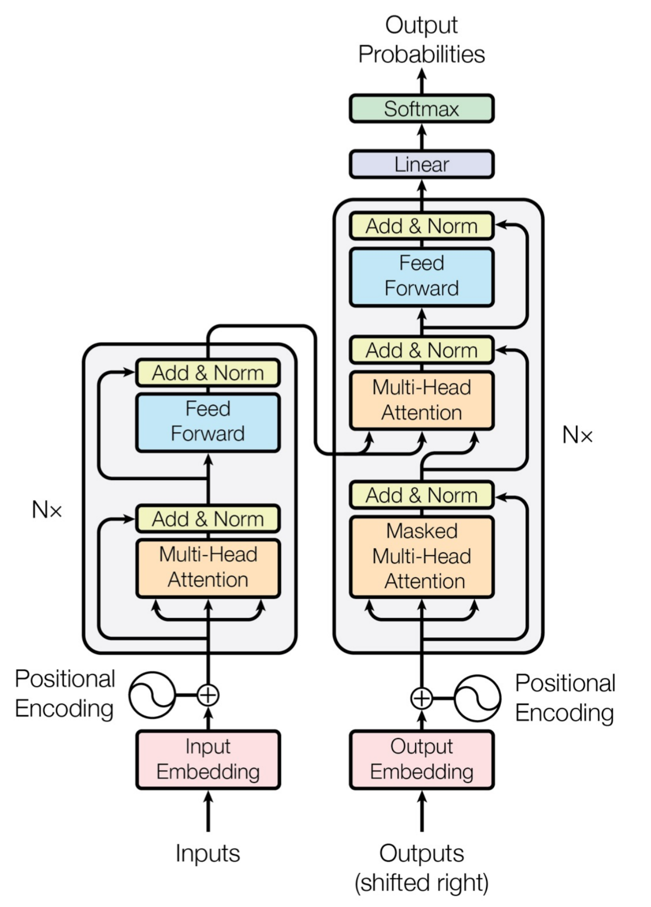


Transformer

- Attention is all you need
    - https://arxiv.org/abs/1706.03762

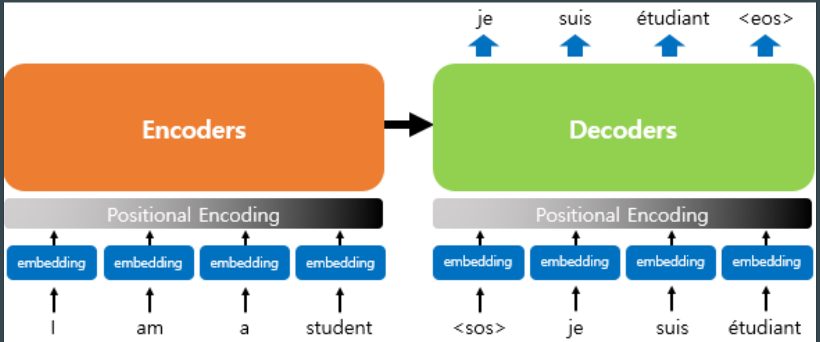






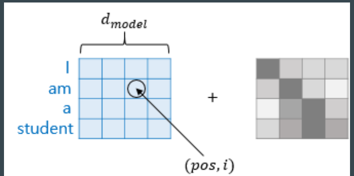
Transformer

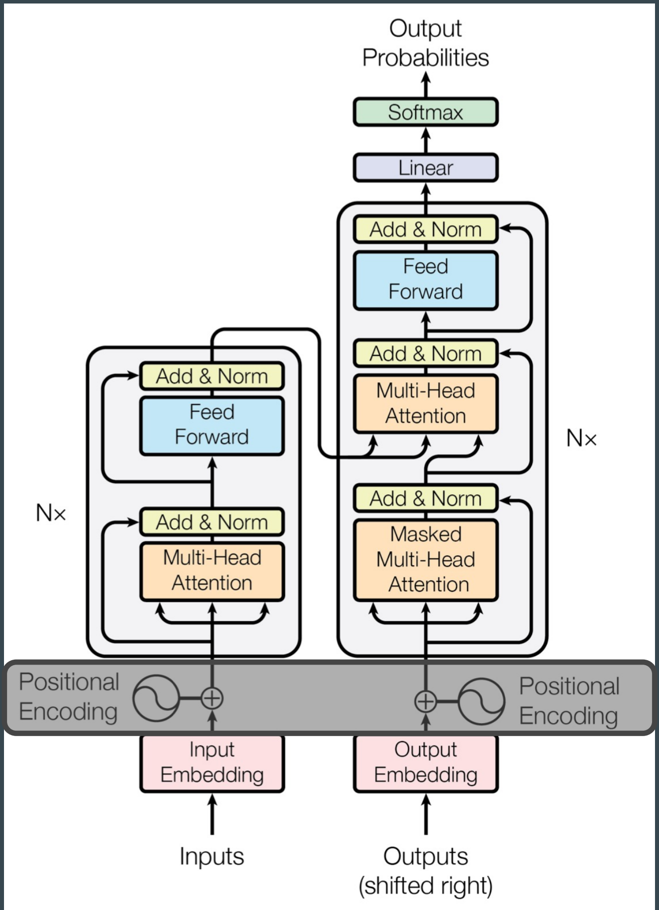
- Positional encoding



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

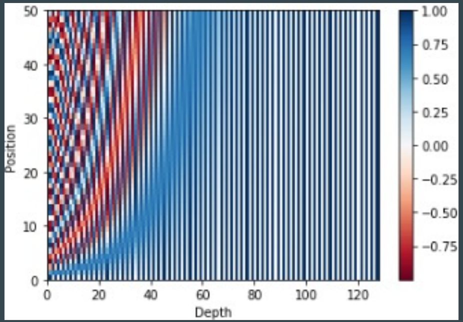
$$PE_{(pos,\, 2i+1)} = cos(pos/10000^{2i/d_{model}})$$

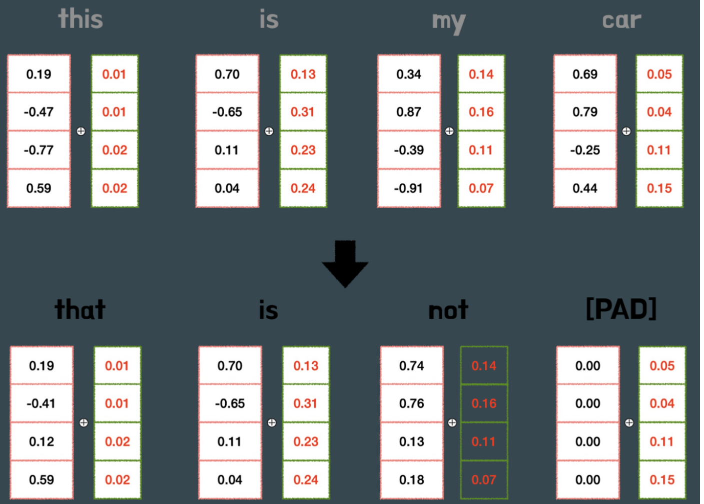




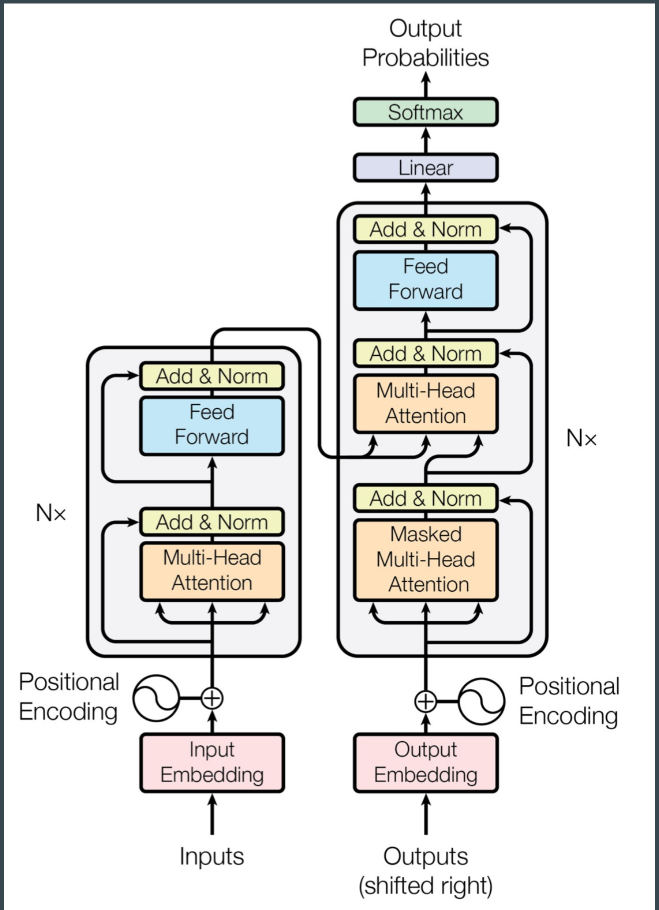
Transformer

- Positional encoding



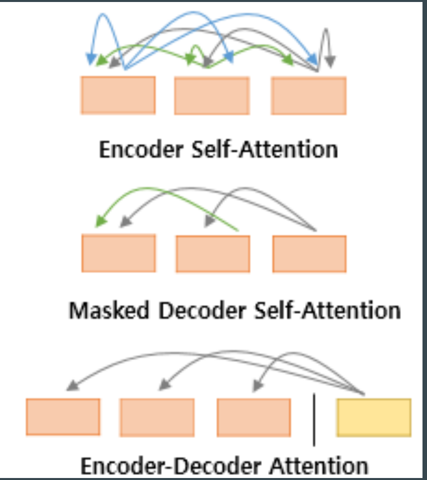


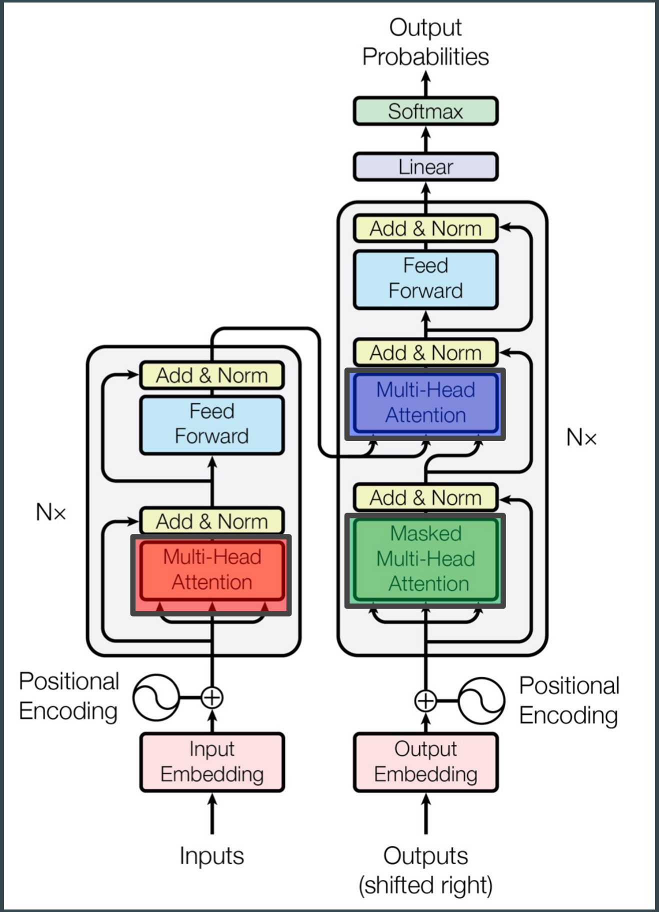
Positional Encoding Value
문장의 길이 50, 임베딩 벡터의 차원 128



Transformer

- Attentions,

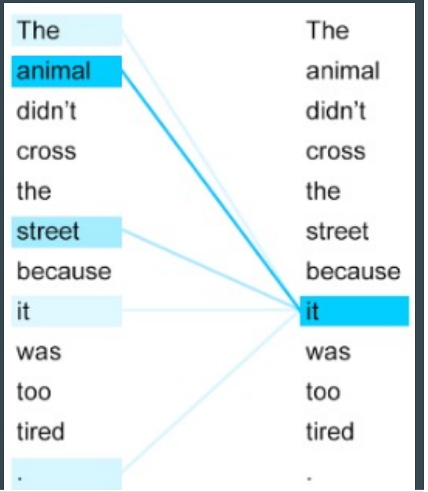




Encoder의 Self-Attention : Query = Key = Value
Decoder의 Masked Self-Attention : Query = Key = Value
Decoder의 Eocoder-Decoder Attention : Query : Decoder Vector /
Key = Value : Encoder Vector

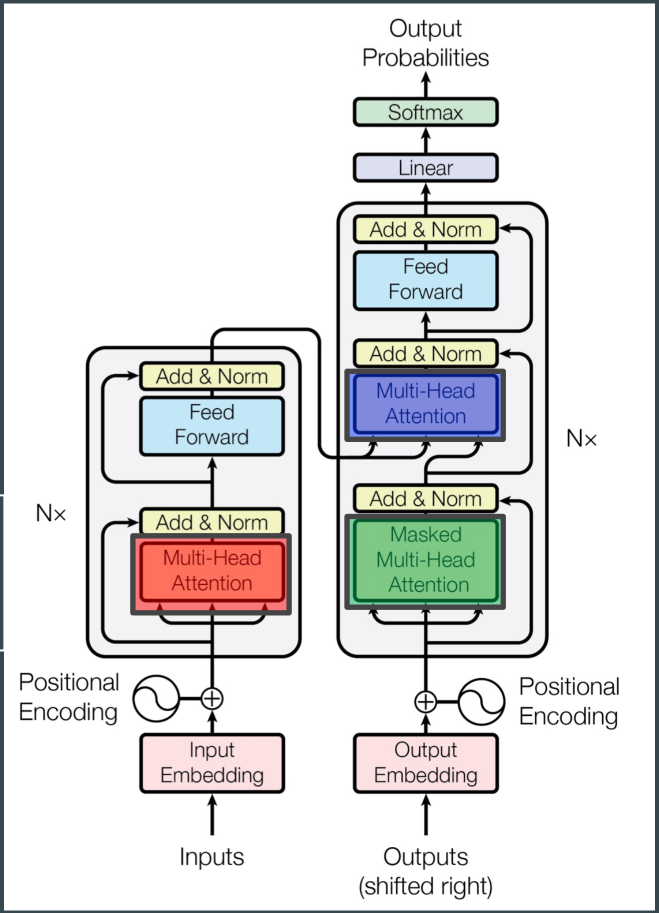
Transformer

- Encoder Self-Attention

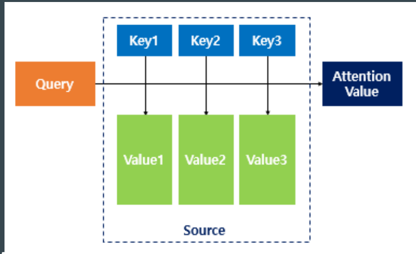


transformer

Q : 입력 문장의 모든 단어 벡터들
K : 입력 문장의 모든 단어 벡터들
V : 입력 문장의 모든 단어 벡터들



vs

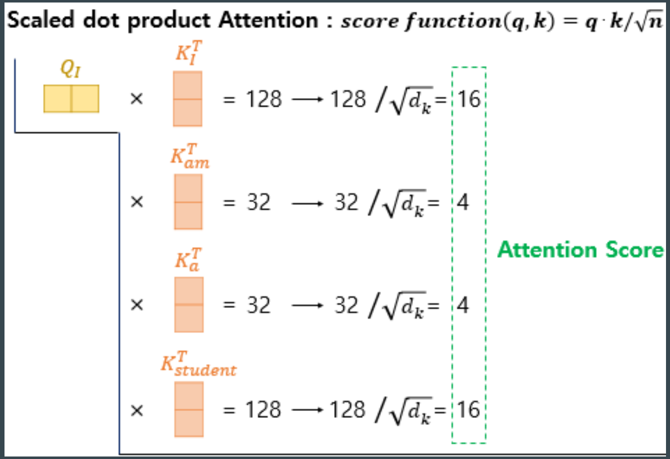


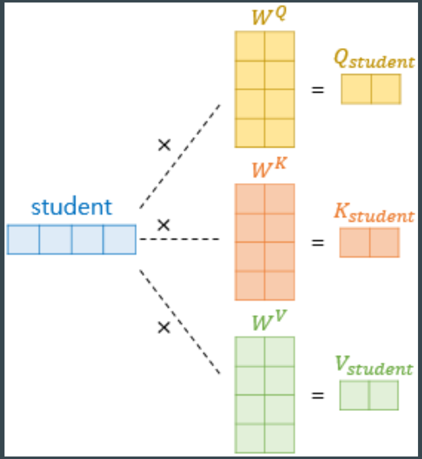
Q = Query : *(모든)* t 시점의 Decoder 셀에서의 은닉 상태
K = Keys : 모든 시점의 Encoder 셀의 은닉 상태들
V = Values : 모든 시점의 Encoder 셀의 은닉 상태들

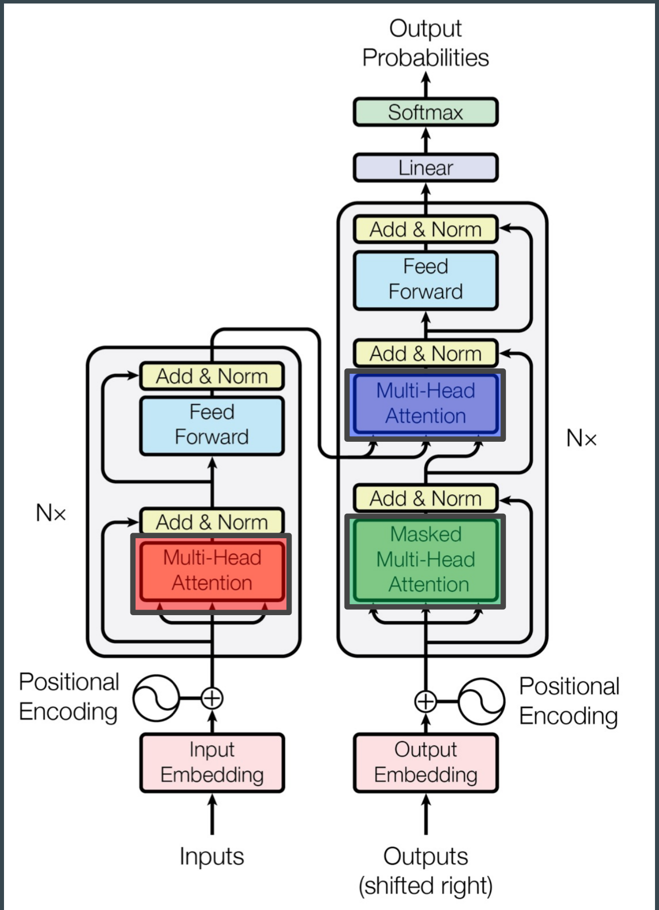
seq2seq

Transformer

- Encoder Self-Attention
  - Scaled dot-product Attention







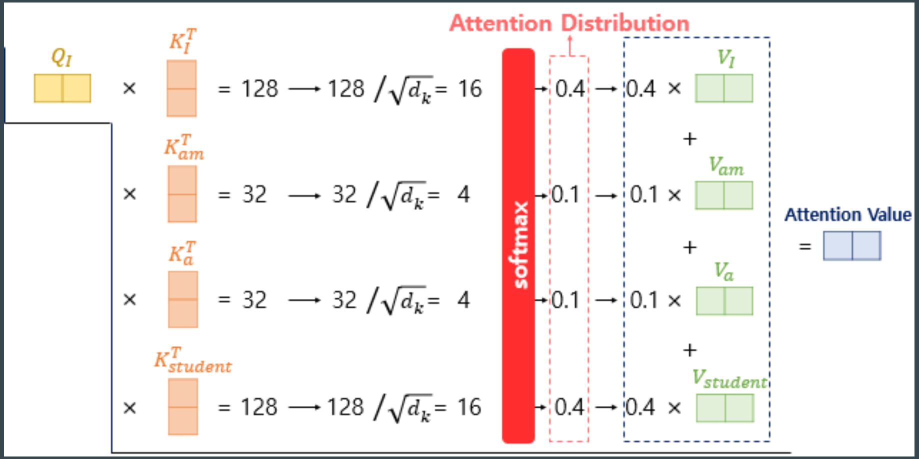
$$W = d_{model} \times \left(\frac{d_{model}}{\text{num\_head}}\right) = 512 \times 64$$

$$d_k = \frac{d_{model}}{\text{num\_head}} = 64$$

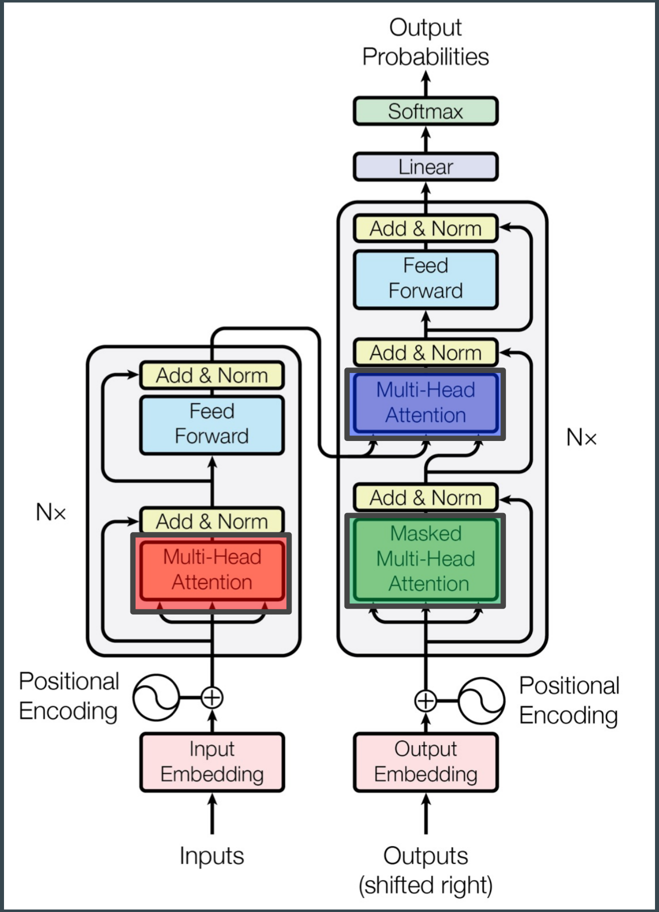
$$d_{model} = 512, \text{num\_head} = 8$$

Transformer

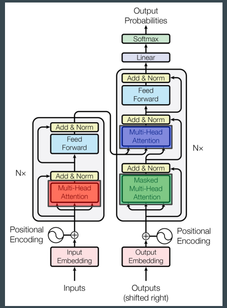
- Encoder Self-Attention
  - Scaled dot-product Attention



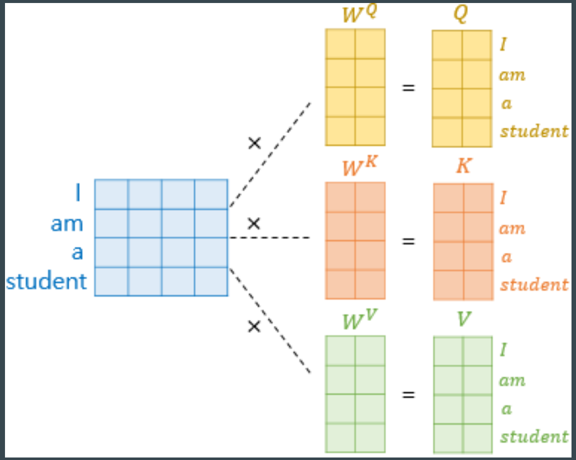
$$d_k = \frac{d_{model}}{\text{num\_head}} = 64 \quad score(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q} \cdot \boldsymbol{k}/\sqrt{d_k}$$

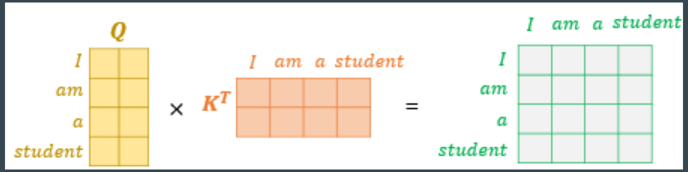


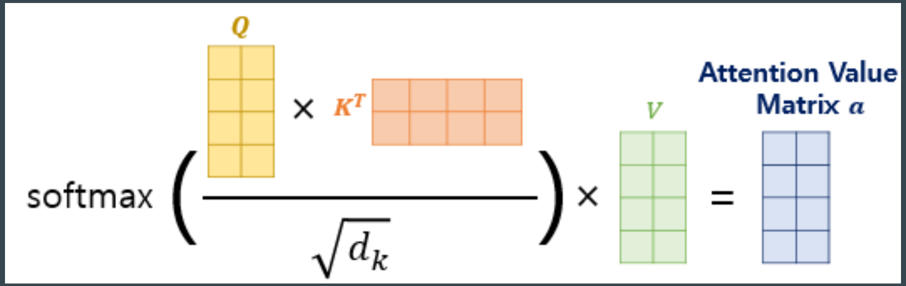
Transformer



- Encoder Self-Attention
    - Scaled dot-product Attention → MATRIX!!!!



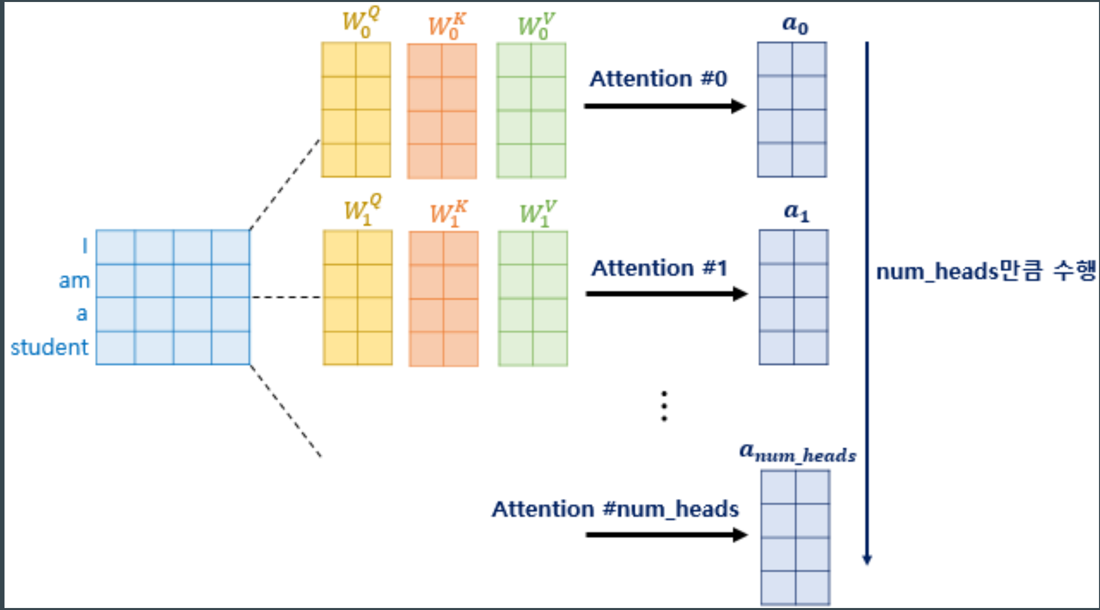


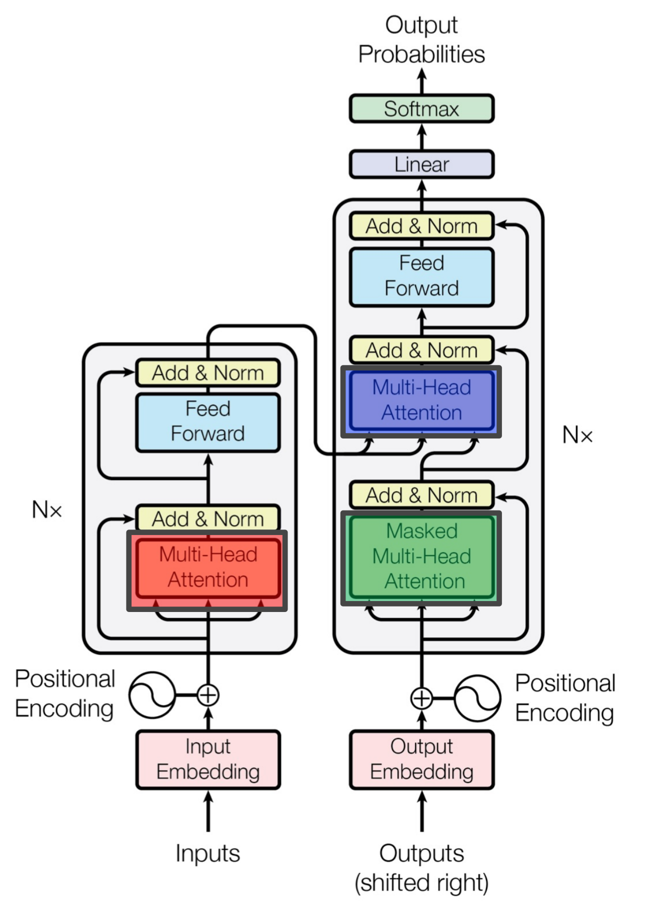


$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Transformer

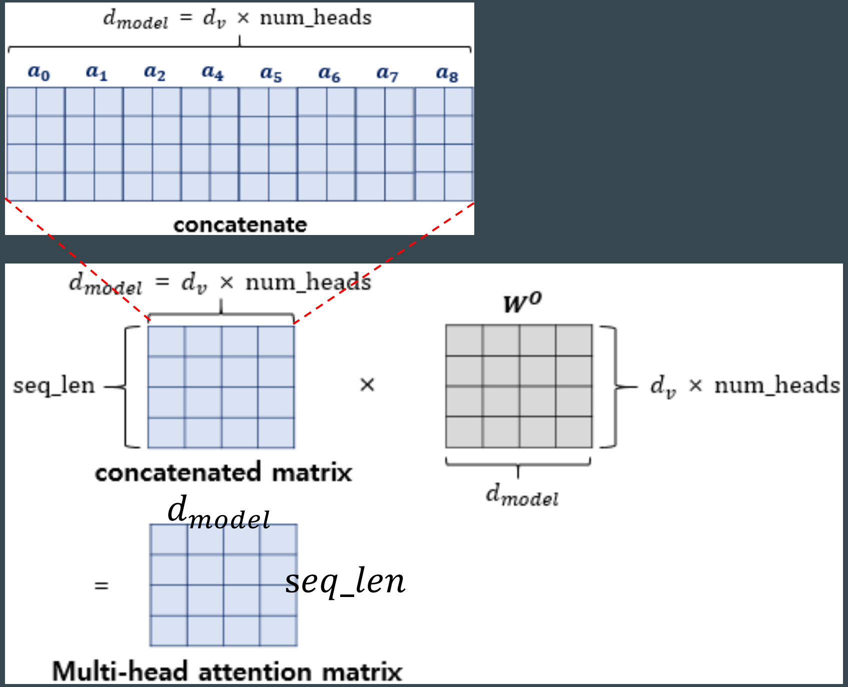
- Multi-head Attention

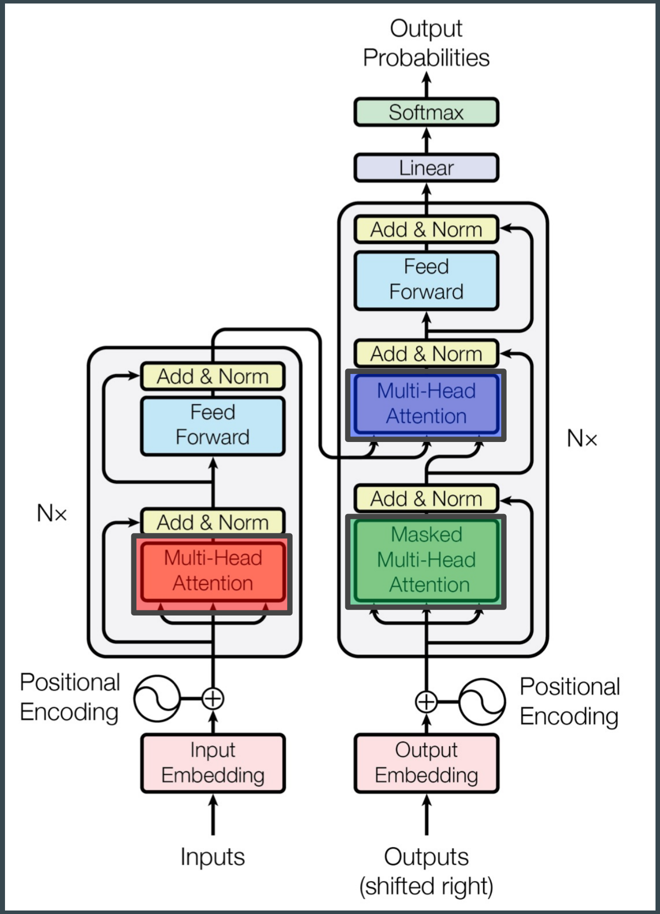




Transformer

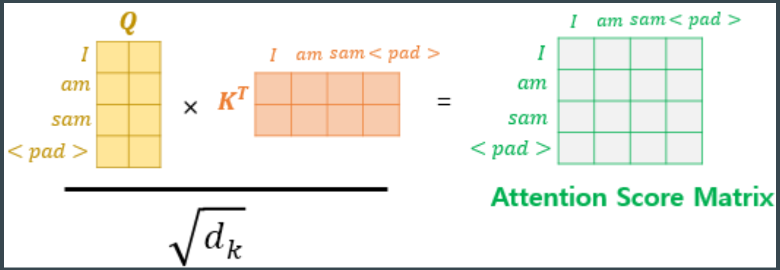
- Multi-head Attention

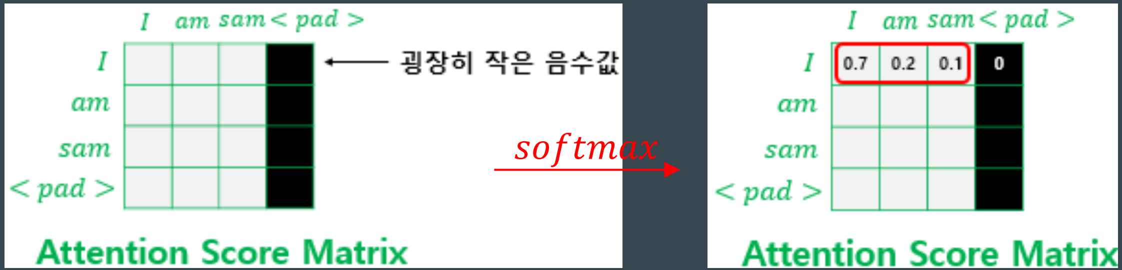


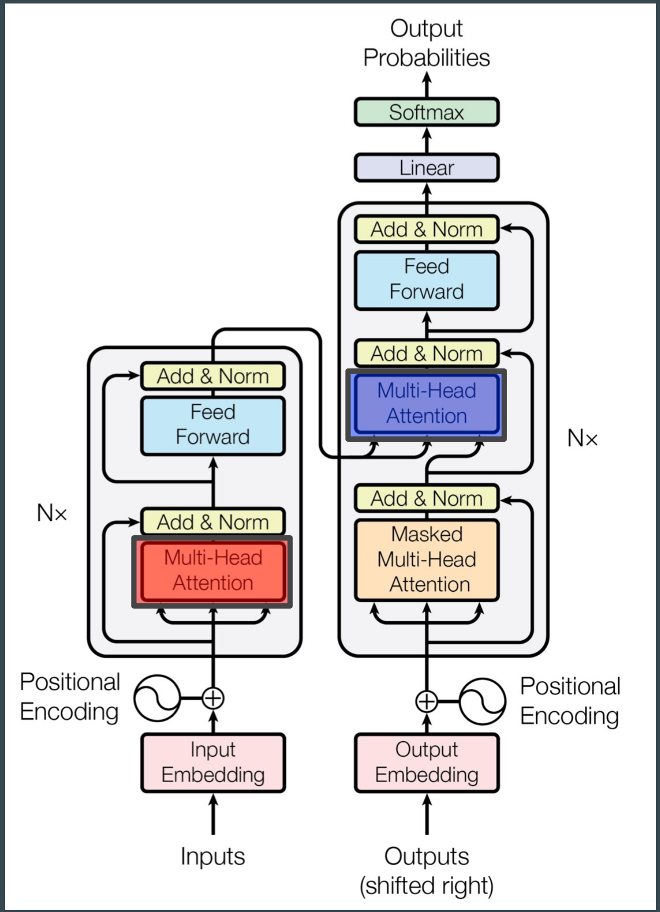


Transformer

- Padding Mask
  - Key의 경우에 <PAD> 토큰이 존재한다면 이에 대해서는 유사도를 구하지 않도록

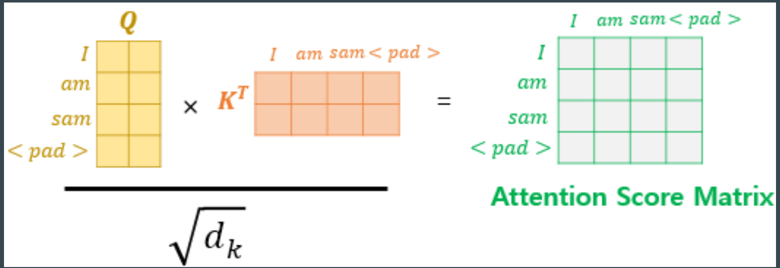


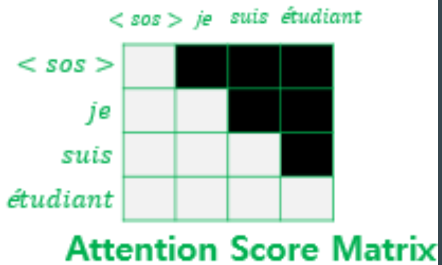


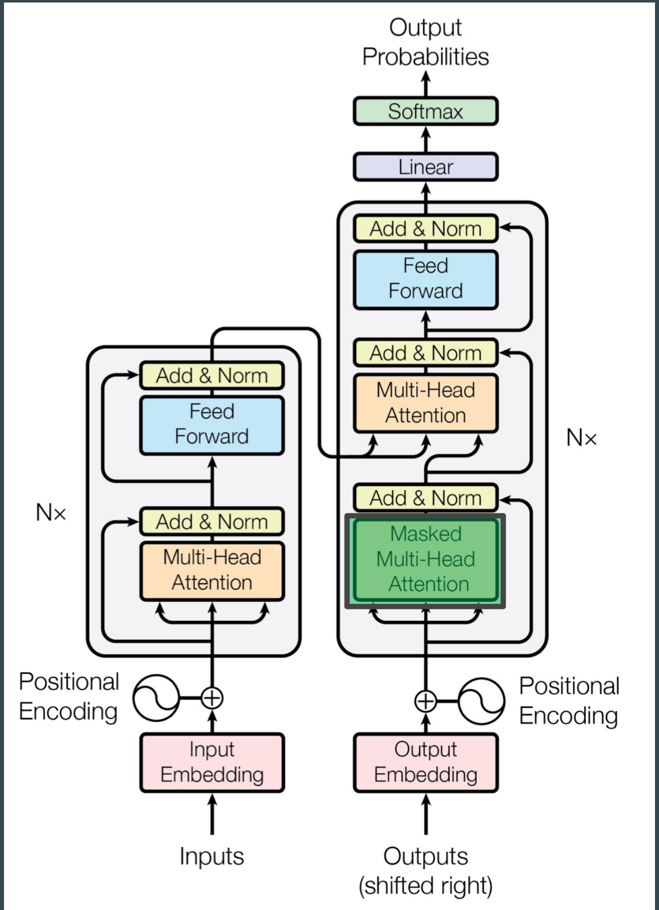


Transformer

- Look-ahead Mask
  - 현재 시점의 예측에서 현재 시점보다 미래에 있는 단어들을 참고하지 못하도록

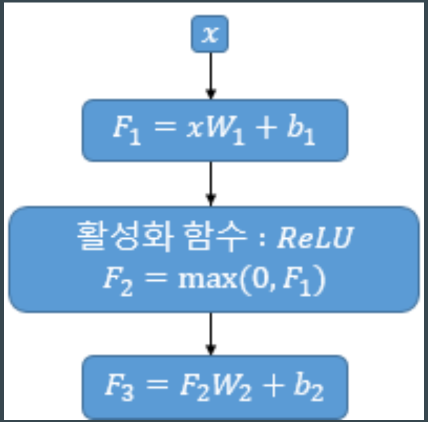




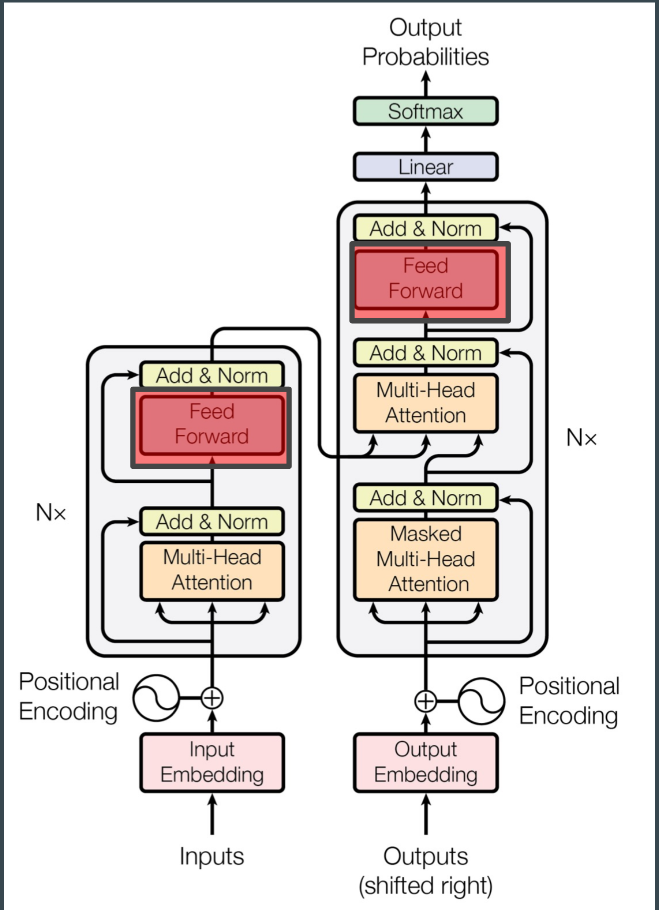


Transformer

- Position-wise FFNN

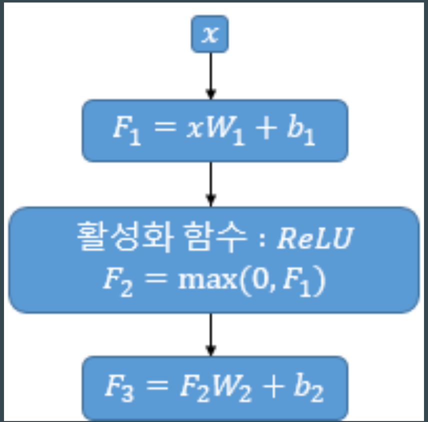


$$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$$



Transformer

- Position-wise FFNN

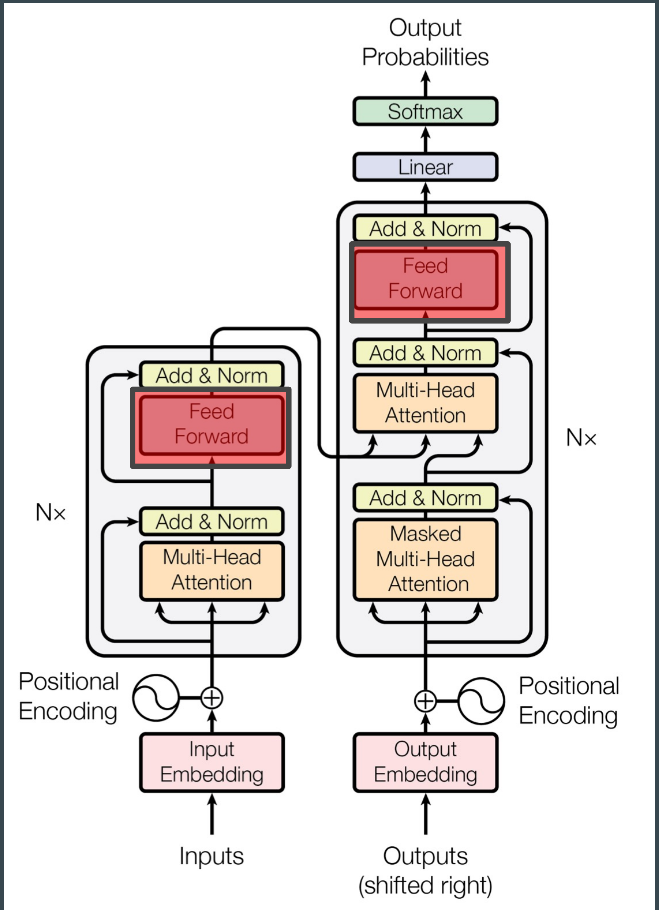


$d_{model} = 512$

$d_{ff} = 2048$

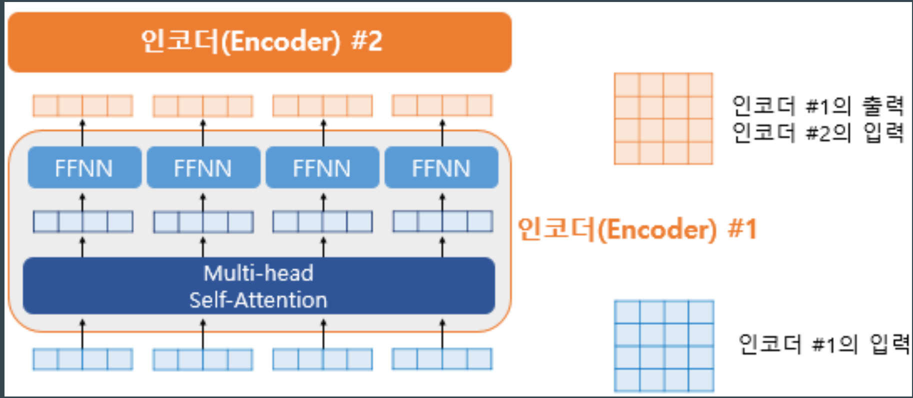
$d_{model} = 512$

$$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$$

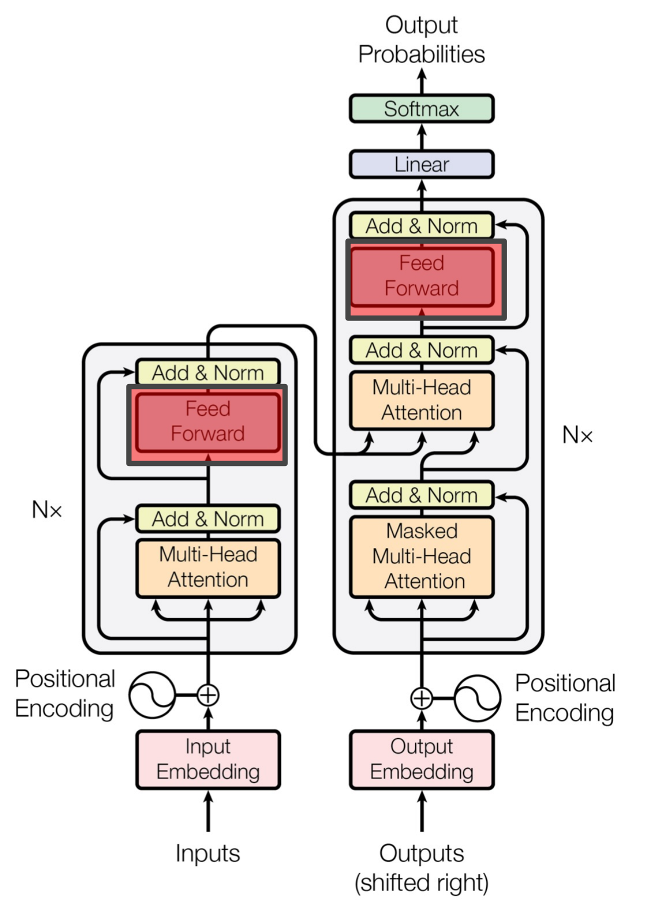


Transformer

- Position-wise FFNN

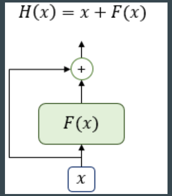


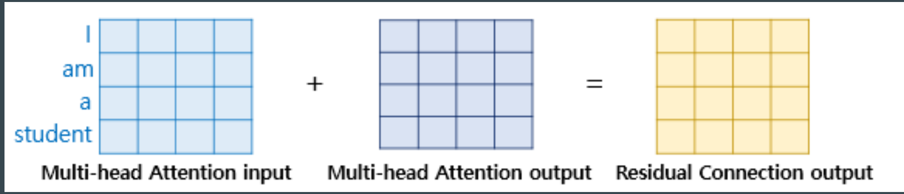

```
outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
outputs = tf.keras.layers.Dense(units=d_model)(outputs)
```



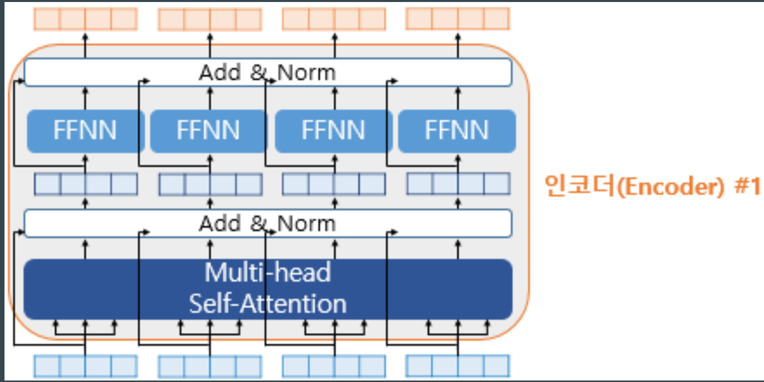
Transformer

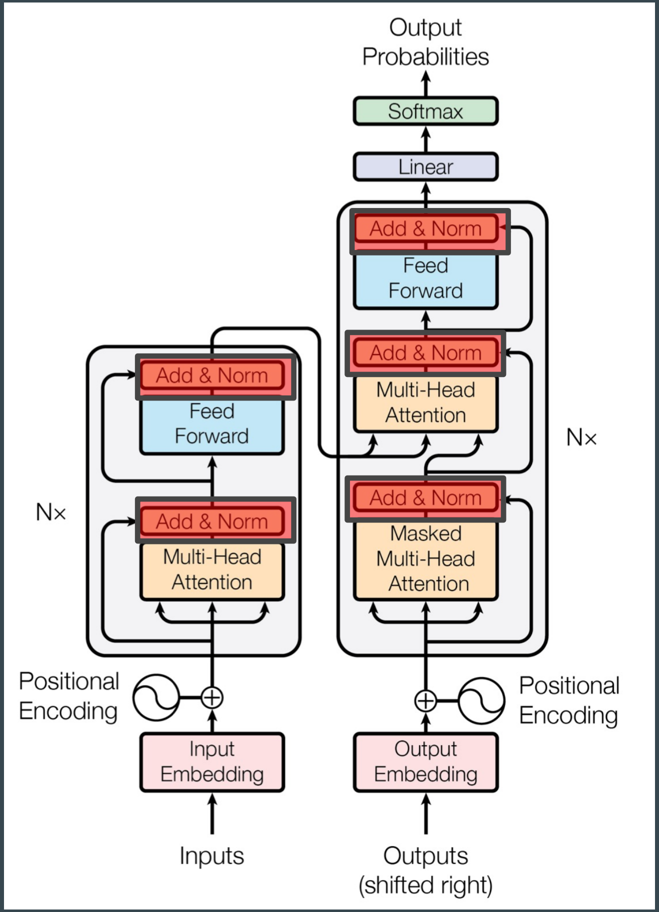
- Residual connection, Layer Normalization





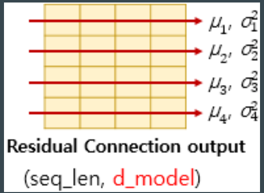
$$H(x) = x + MultiHeadAttention(x)$$





Transformer

- Residual connection, Layer Normalization

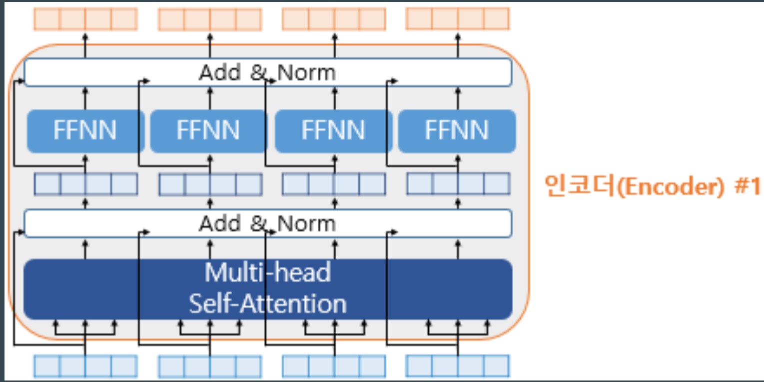


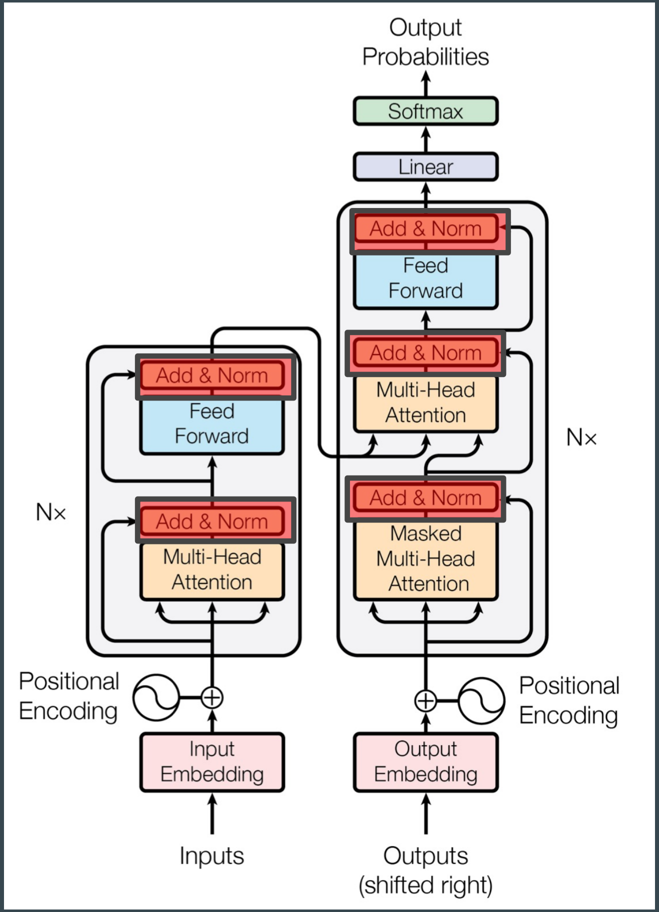
$$ln_i = LayerNorm(x_i)$$

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

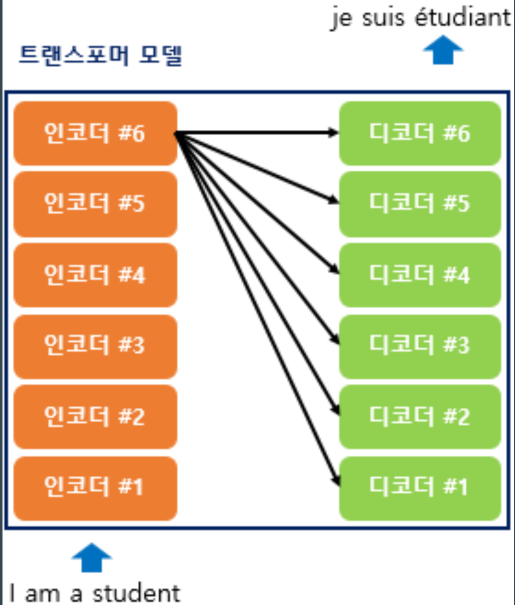
$$ln_i = \gamma \hat{x}_i + \beta = LayerNorm(x_i)$$

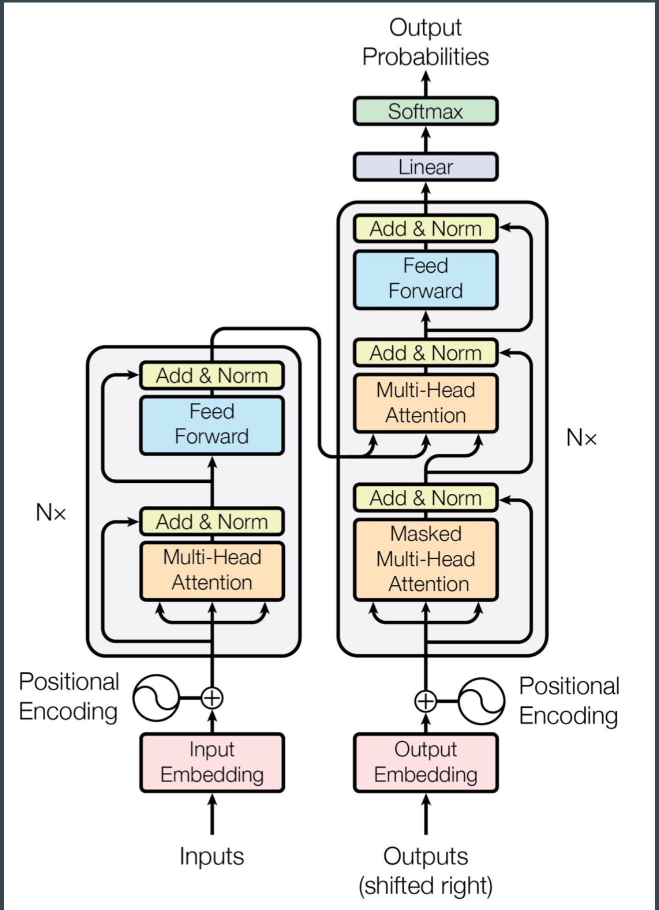
γ	1	1	1	1
β	0	0	0	0





Transformer





Everything can be found in
https://wikidocs.net/book/2155
https://github.com/ukairia777/tensorflow-nlp-tutorial