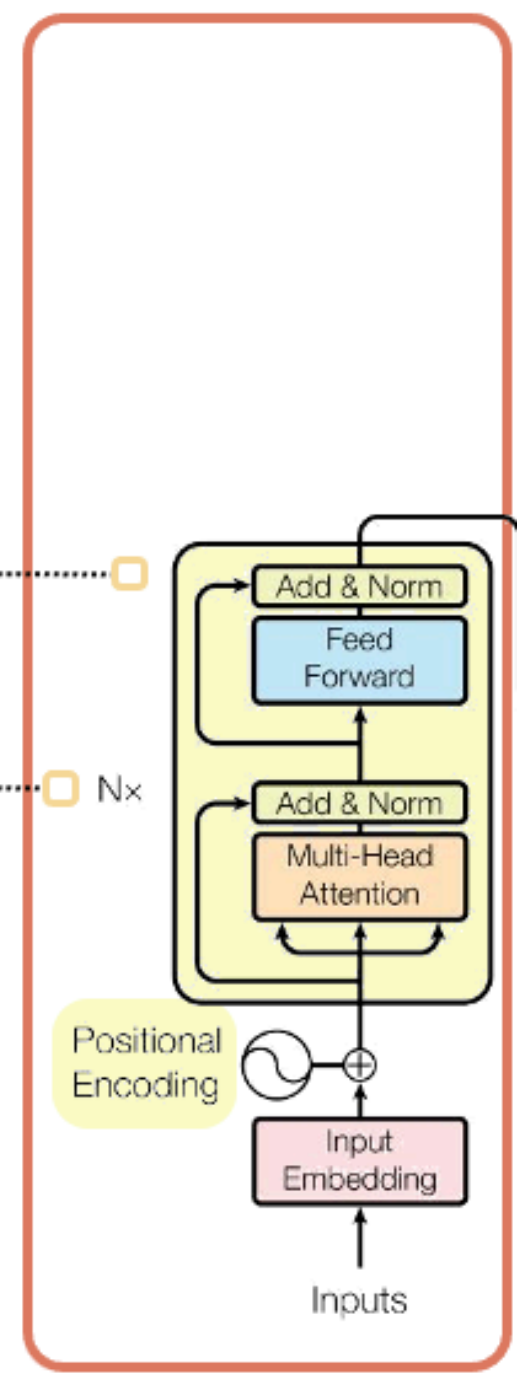
```python
class EncoderLayer(nn.Module):
    ''' Compose with two layers '''

    def __init__(self, d_model, d_inner, n_head, d_k, d_v, dropout=0.1):
        super(EncoderLayer, self).__init__()
        self.slf_attn = MultiHeadAttention(n_head, d_model, d_k, d_v, dropo
        self.pos_ffn = PositionwiseFeedForward(d_model, d_inner, dropout=dr

    def forward(self, enc_input, slf_attn_mask=None):
        enc_output, enc_slf_attn = self.slf_attn(
            enc_input, enc_input, enc_input, mask=slf_attn_mask)
        enc_output = self.pos_ffn(enc_output)
        return enc_output, enc_slf_attn
```

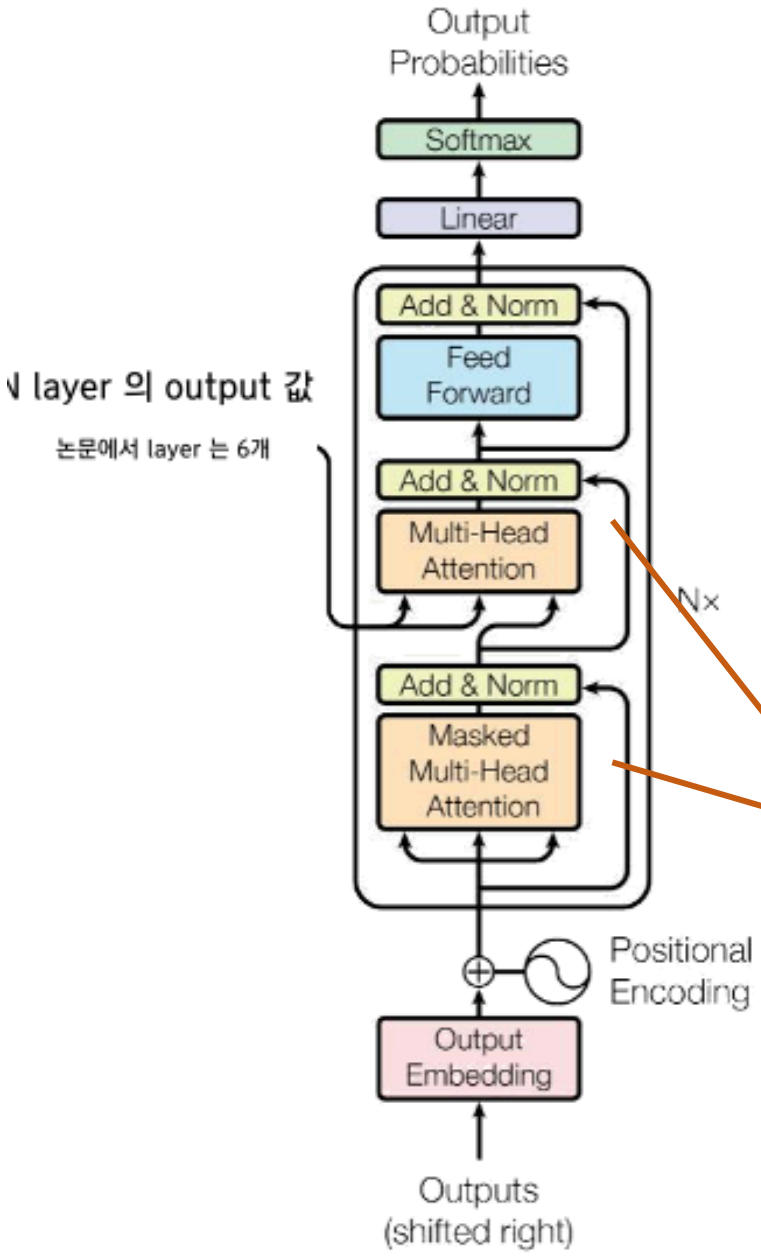
Encoder

Dimension = 512

각각의 layer는 2개의 sub-layers
- 1ST sub-layer : multi-head Attention
- 2nd sub-layer : position-wise fully connected feed-forward network

인코더와 디코더의 Layer는 6개

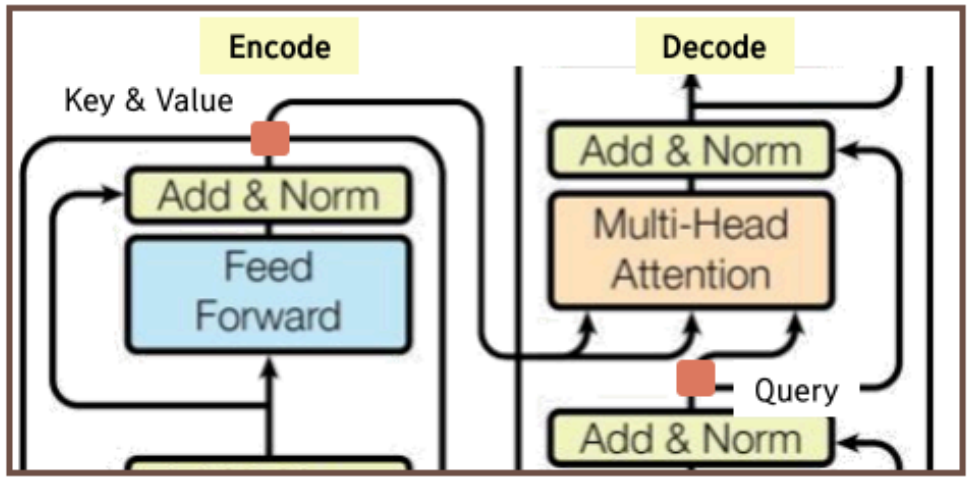




- 1st sub-layer : Masked Multi-Head Attention [Masked Decoder Self - Attention]
- 2nd sub-layer : multi-head Attention [Encoder - Decoder Attention]
- 3rd sub-layer : position-wise fully connected feed-forward network

```
class DecoderLayer(nn.Module):
    ''' Compose with three layers '''

    def __init__(self, d_model, d_inner, n_head, d_k, d_v, dropout=0.1):
        super(DecoderLayer, self).__init__()
        self.slf_attn = MultiHeadAttention(n_head, d_model, d_k, d_v, dropout=dropout)
        self.enc_attn = MultiHeadAttention(n_head, d_model, d_k, d_v, dropout=dropout)
        self.pos_ffn = PositionwiseFeedForward(d_model, d_inner, dropout=dropout)

    def forward(
            self, dec_input, enc_output,
            slf_attn_mask=None, dec_enc_attn_mask=None):
        dec_output, dec_slf_attn = self.slf_attn(
            dec_input, dec_input, dec_input, mask=slf_attn_mask)
        dec_output, dec_enc_attn = self.enc_attn(
            dec_output, enc_output, enc_output, mask=dec_enc_attn_mask)
        dec_output = self.pos_ffn(dec_output)
        return dec_output, dec_slf_attn, dec_enc_attn
```

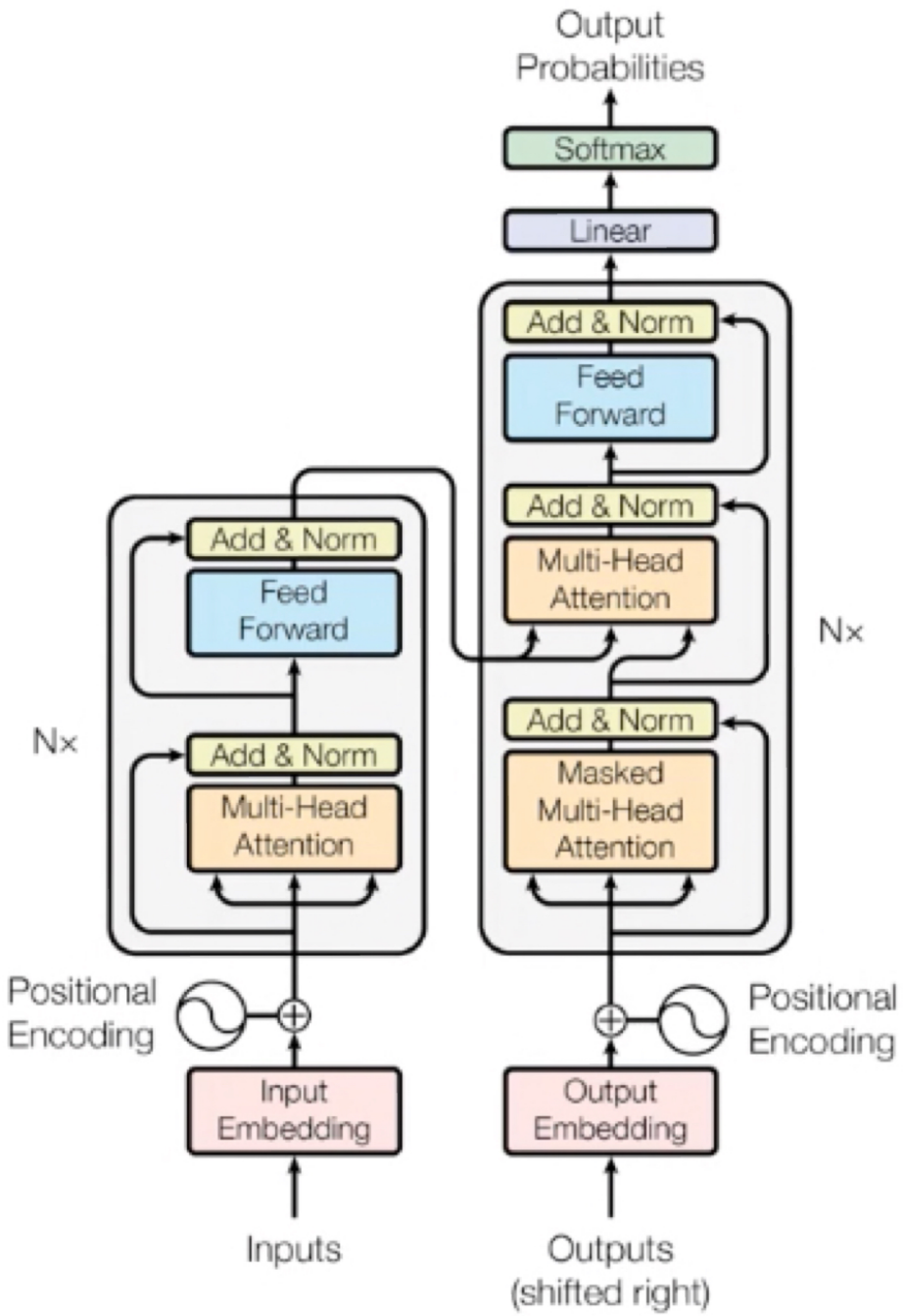
2nd sub-layer : multi-head Attention [Encoder - Decoder Attention]



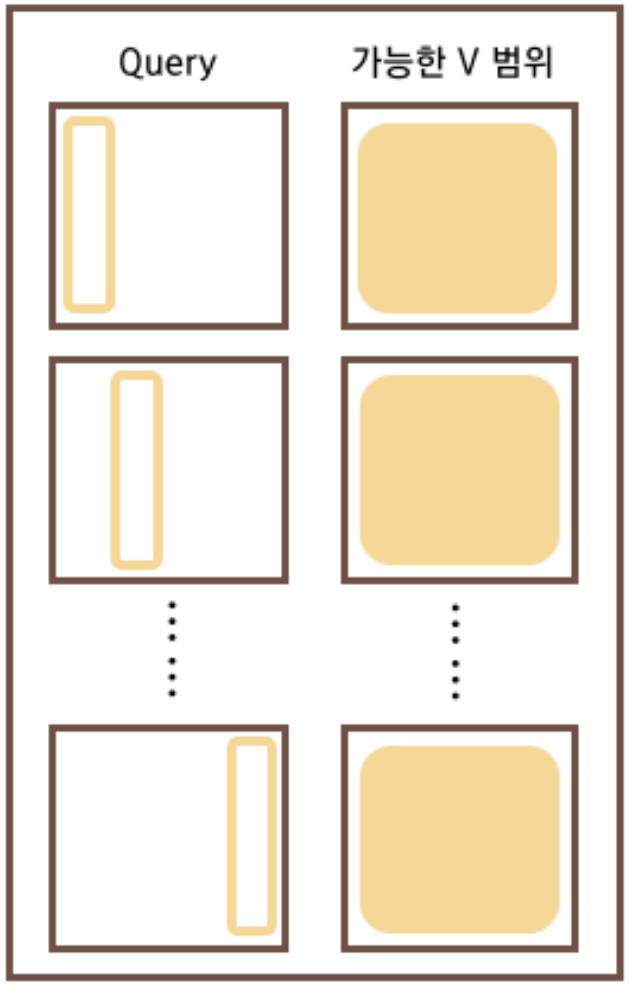
Encoder 의 outpu이 decode의 2nd sub-layer의 key와 value 가 됨
Decoder의 self-attention 의 결과를 2nd sub-layer 의 query로 사용함

```
class DecoderLayer(nn.Module):
    ''' Compose with three layers '''

    def __init__(self, d_model, d_inner, n_head, d_k, d_v, dropout=0.1):
        super(DecoderLayer, self).__init__()
        self.slf_attn = MultiHeadAttention(n_head, d_model, d_k, d_v, dropout=dropout)
        self.enc_attn = MultiHeadAttention(n_head, d_model, d_k, d_v, dropout=dropout)
        self.pos_ffn = PositionwiseFeedForward(d_model, d_inner, dropout=dropout)

    def forward(
            self, dec_input, enc_output,
            slf_attn_mask=None, dec_enc_attn_mask=None):
        dec_output, dec_slf_attn = self.slf_attn(
            dec_input, dec_input, dec_input, mask=slf_attn_mask)
        dec_output, dec_enc_attn = self.enc_attn(
            dec_output, enc_output, enc_output, mask=dec_enc_attn_mask)
        dec_output = self.pos_ffn(dec_output)
        return dec_output, dec_slf_attn, dec_enc_attn
```

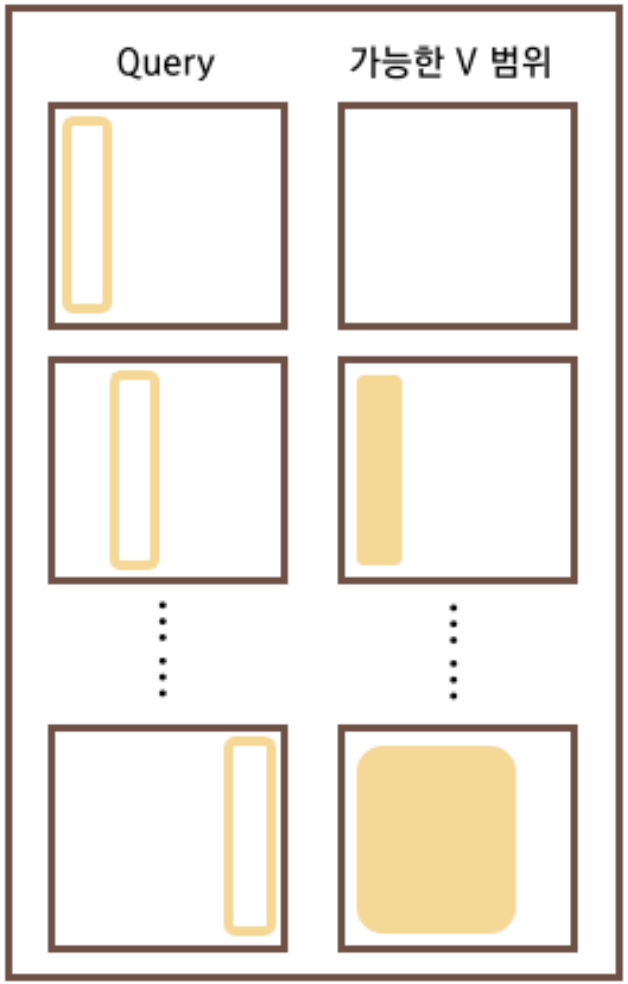

Query
(Decoder)
X
Key^T
(Encoder)
Softmax
Attention Weights
X
Value
(Encoder)
Attention(Q_d, K_e, V_e)



Encode Masking



Decoder Masking



Decoder masking은 자기 자신을 포함한 미래값과는 Attention을 구하지 않음

Encoder / Decoder masking 모두, padding token 의 위치에 대한 making진행

▼ Q&A

Q. 자연어에서 key, query, value가 나눠지는 이유가 무엇인지 알 수 있을까요?? 특히 어텐션 같은 구조에서 자기 자신 임베딩값하나만 가지고도 어텐션을 구할수가 있는데 굳이 qkv로 만들어서 하는 이유가 궁금합니다! 임베딩된 벡터 가 w매트릭스를 거치면서 q, k, v가 되는데 사실, 그렇게 하지 않고, 그 벡터끼리의 유사도(cosine similarity)만 구해서 소프트맥스를 구해도 그 벡터들 끼리의 유사도를 나타내는 매트릭스가 되서 다시 그 값이랑 곱해서 표현할 수 있지 않을까 생각했습니다. 하나의 단어를 각각의 단어에 대해서 linear projection을 통해서 key, query, value를 만든 후, key, query끼리 어텐션을 구하고, value에대해서 weighted sum을 해주는 방식을 취하는데, 그렇게 안해도 벡터들끼리의 유사도를 구해서 소프트맥스를 하면, 비슷한 느낌일거 같은데 나누는 이유를 모르겠네요 ㅜㅜ

A. 아 linear projection 없이 self-attention을 취하는 경우를 이야기 하신거군요

하나는 cross-attention에서 결국 self-attention이 나온거잖아요 ? 그렇다 보니까 기본적으로 cross-attention에 주어지는 query와 context에 대해서 dimension이 달라 projection을 해야 하는 경우가 있을 수 있구요. 두번째로 context에서 key와 value를 모두 추출해야 하는 경우, value는 몰라도 key는 query같은 space를 공유해야 유사도를 구하는게 의미가 있기 때문에 key와 space를 공유하게 만들기 위해 projection layer를 추가하던게 그대로 self-attention에 온게 아닐까 싶습니다

두번째는 self-attention에 만약 projection layer가 없었다면, trainable layer가 아니고, 결국 deterministic 하게 항상 같은 값을 출력으로 내잖아요 ? 그런거 보다는 네트워크 수준에서 좀 더 유동적으로 원하는 값을 선택하고, 추출해서 의미 있는 결과를 만들기 위해 projection layer를 추가한게 아닐까 싶습니다