Attention Is All You Need

Model Architecture

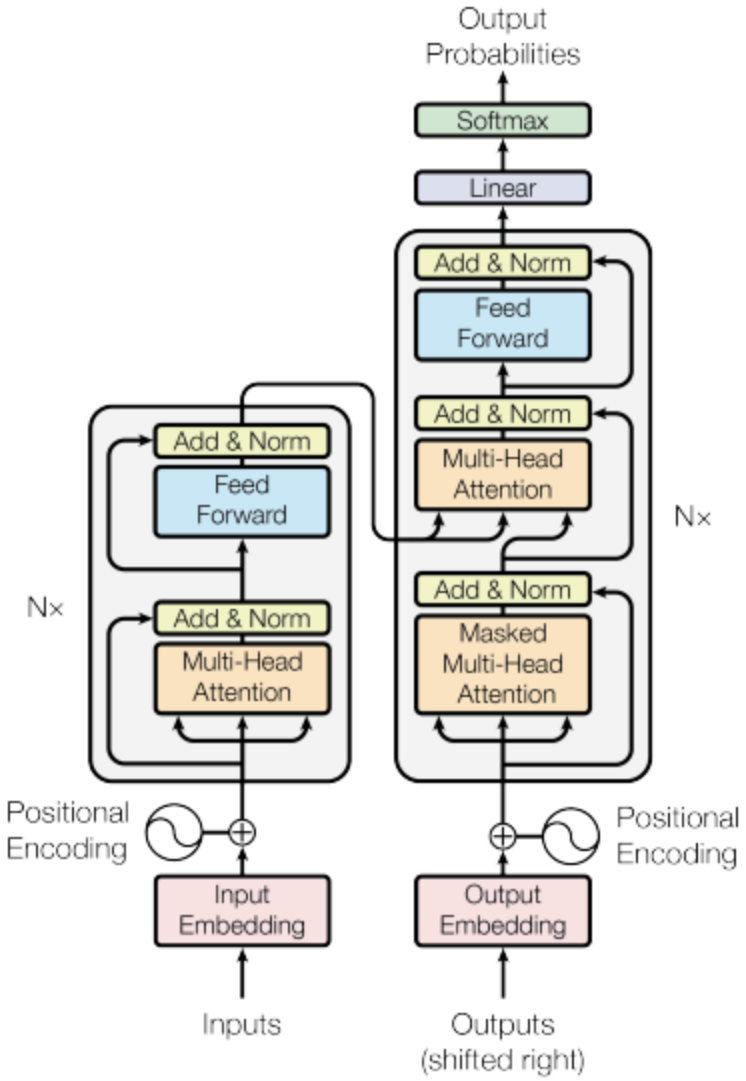


Figure 1: The Transformer - model architecture.

Encoder

- 6개의 동일한 layer stack으로 구성
- 각 layer는 2개의 sub-layer로 구성

1) multi-head self-attention layer

2) position-wise fully connected feed-forward layer

- sub-layer는 layer normalization을 거쳐 residual connection

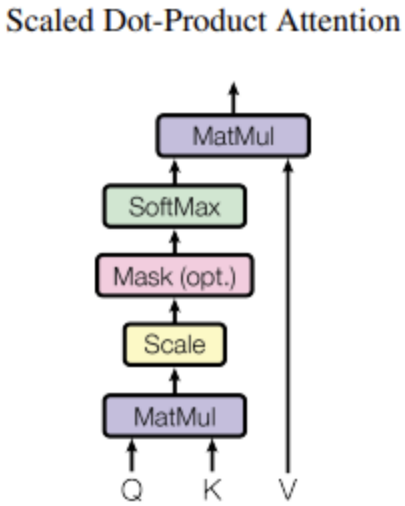
Decoder

- 6개의 동일한 layer stack으로 구성
- Encoder와 동일한 2개의 sub-layer + encoder stack의 출력에 대한 multi-head attention을 수행하는 sub-layer로 구성
- sub-layer는 layer normalization을 거쳐 residual connection

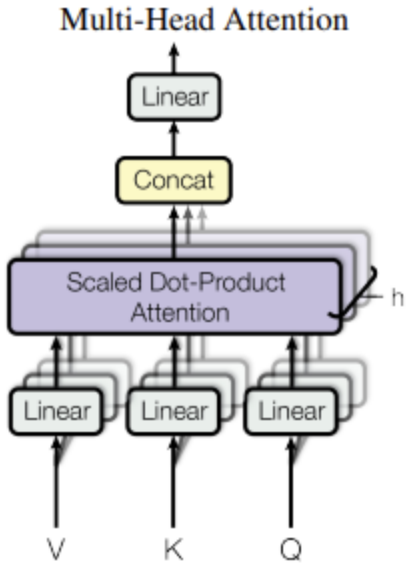
Attention

- query vector, key vector, value vector 쌍을 output으로 mapping하는 과정
- value vector들의 weighted sum으로 output 계산
- query vector와 query의 key vector를 통해 각 가중치 계산

Scaled Dot-Product Attention



Multi-Head Attention



Position-wise Feed-Forward Network

- Each of the layers in encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically
- Consists of two linear transformations with a ReLU activation in between

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

Embeddings and Softmax

- Use learned embeddings to convert the input tokens and output tokens to vectors of dimension d model
- Use the usual learned linear transformation and softmax function to convert the decoder output to predicted next-token probabilities
- Share the same weight matrix between the two embedding layers and the pre-softmax linear transformation

Positioning Encoding

- Added "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks
- The positional encodings have the same dimension dmodel as the embeddings, so that the two can be summed
- Use sine and cosine functions of different frequencies

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