Chapter 04. 자연어처리 (Natural Language Processing)

자연어처리 대세 Transformer

Attention is all you need

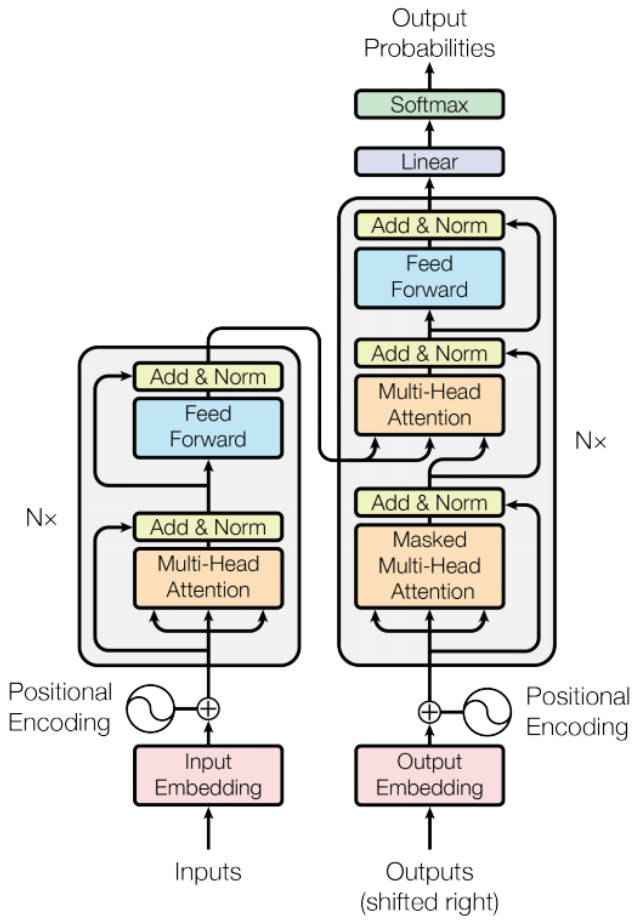


Figure 1: The Transformer - model architecture.

번역 문제에 RNN과 CNN을 사용하지 않고, Attention만을 이용하여 State-of-the-art 성능을 끌어낸 연구

네트워크의 특성

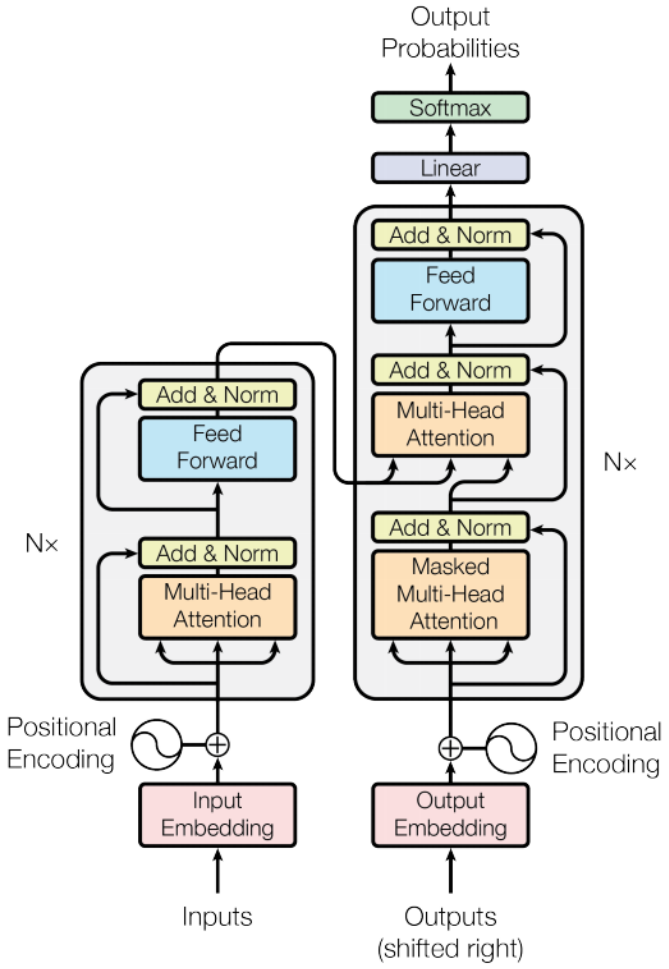


Figure 1: The Transformer - model architecture.

- Seq2seq와 유사한 Transformer 구조 사용
- 제안하는 Scaled Dot-Product Attention과, 이를 병렬로 나열한 Multi-Head Attention 블록이 알고리즘의 핵심
- RNN의 BPTT와 같은 과정이 없으므로 병렬 계산 가능
- 입력된 단어의 위치를 표현하기 위해 Positional Encoding 사용

Transformer vs. Seq2seq

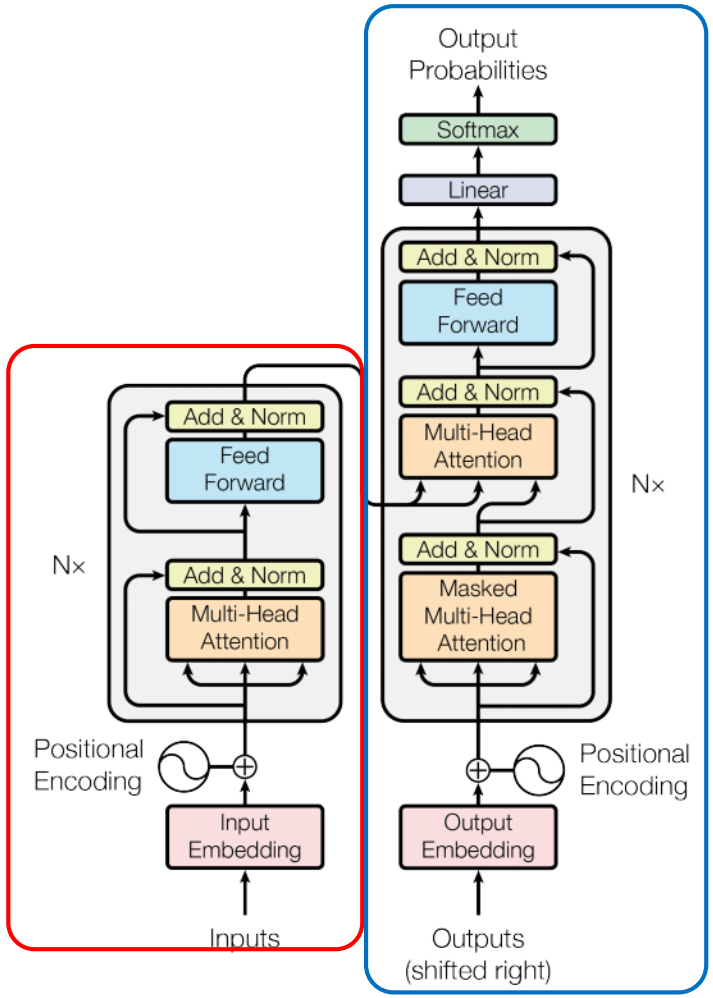
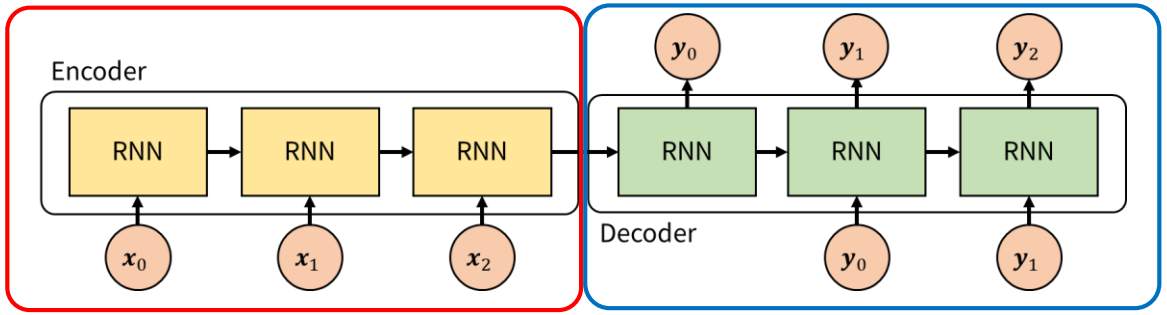


Figure 1: The Transformer - model architecture.



Inputs & Output

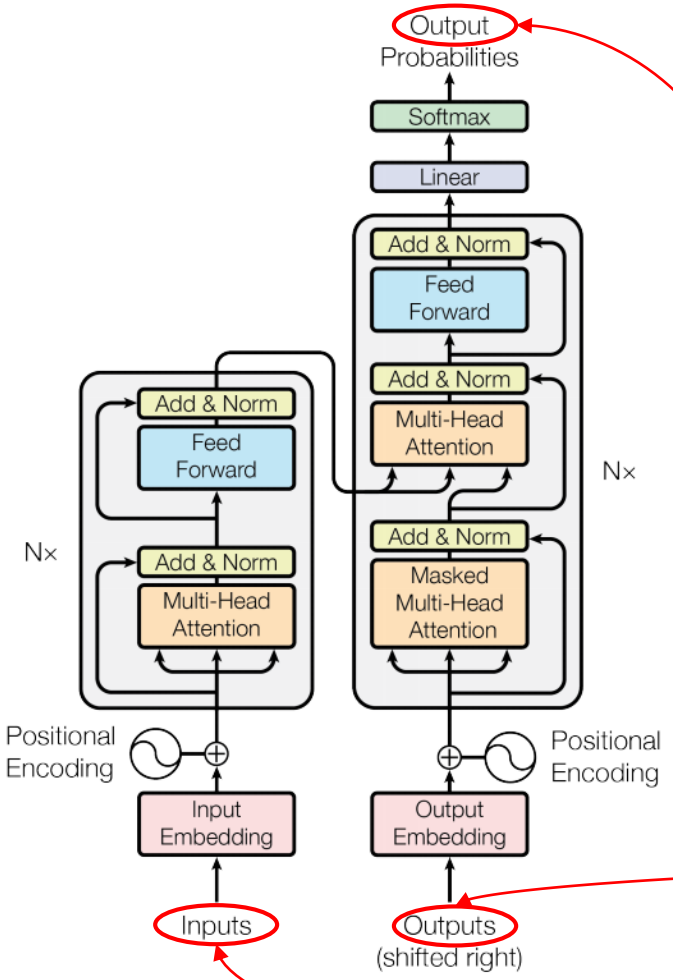


Figure 1: The Transformer - model architecture.

$Y = [y_0, y_1, \dots y_m]$

출력 Sequence 길이 m

출력 단어의 가짓수

0 0 1 0 ... 0

One-hot encoding

$X = [x_0, x_1, \dots x_n]$

입력 Sequence 길이 n

입력 단어의 가짓수

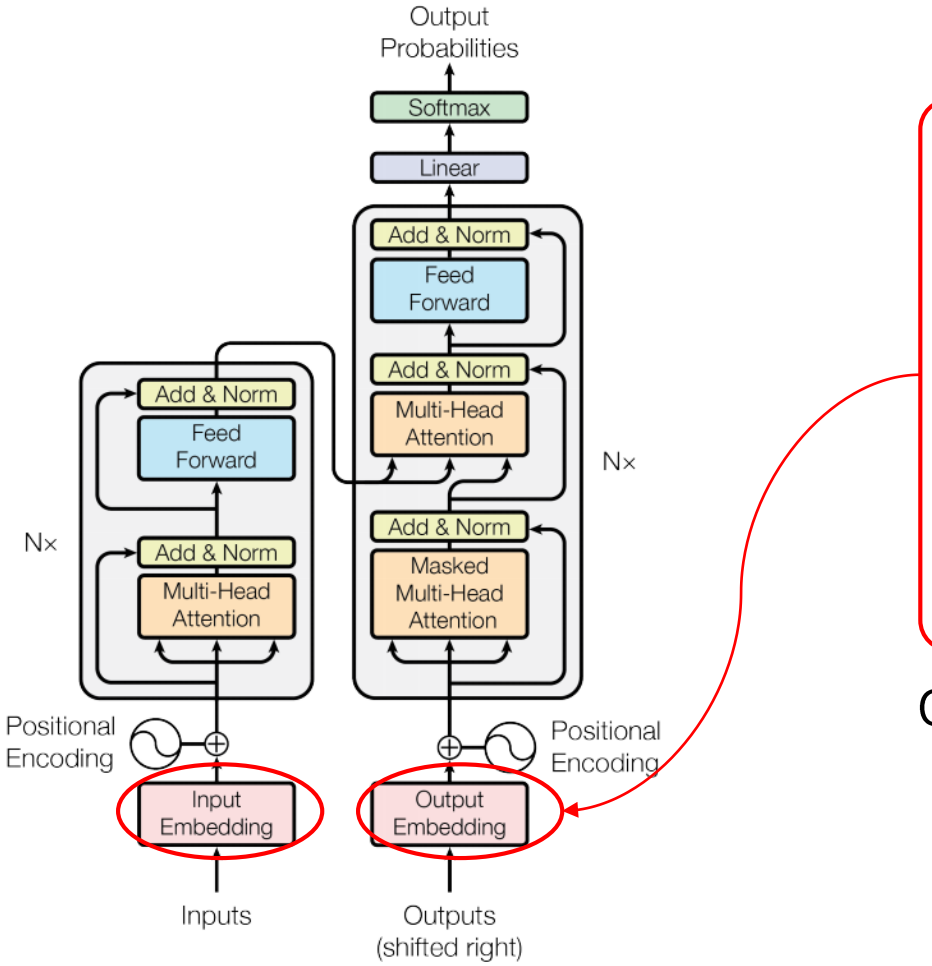
$\hat{Y} = [\mathbf{0}, y_0, \dots y_{m-1}]$

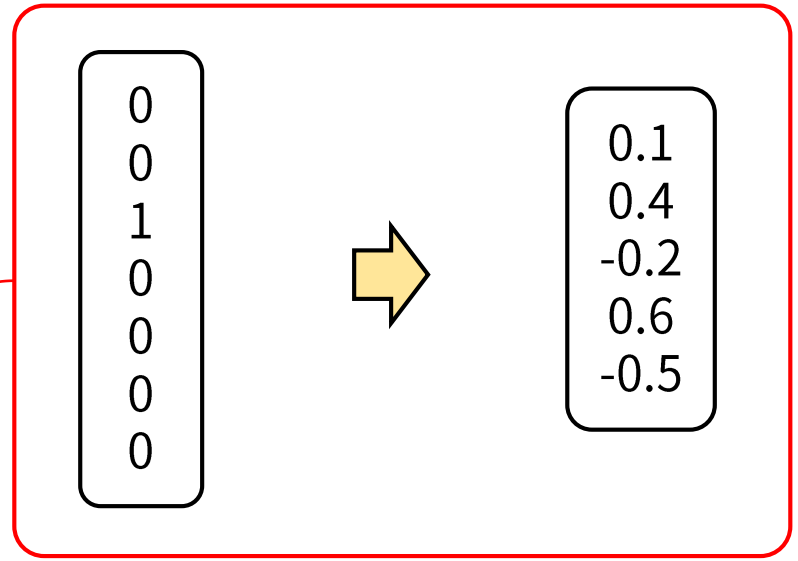
출력 Sequence 길이 m

출력 단어의 가짓수

<SOS>

Word Embedding

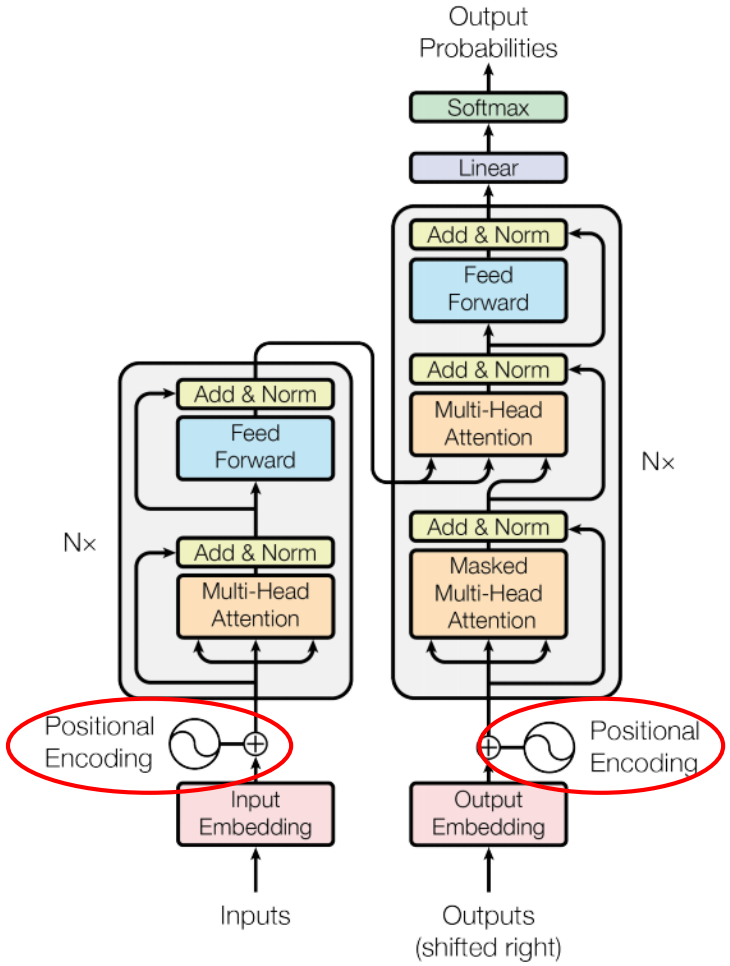




One-Hot Encoding → Embedding

One-Hot Encoding된 단어를 실수 형태로 변경하면서 차원의 수를 줄이는 방법

Positional Encoding



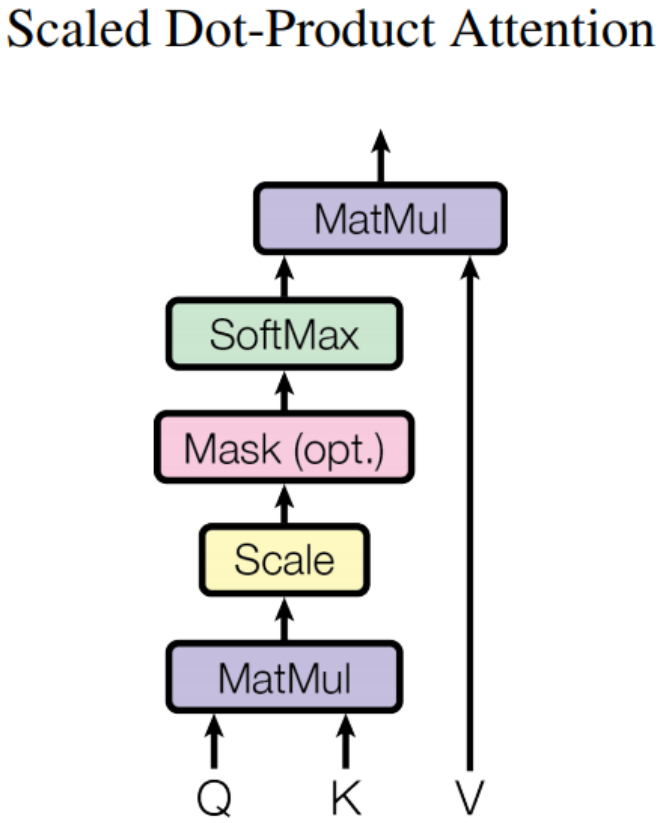
- 시간적 위치별로 고유의 Code를 생성하여 더하는 방식
- 전체 Sequence의 길이 중 상대적 위치에 따라 고유의 벡터를 생성하여 Embedding된 벡터에 더해줌

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

Position별로 구분되는 Encoding을 얻게 됨

pos: 상대적 위치, i: 벡터의 element 인덱스

Scaled Dot-Product Attention



- Query, Key-Value의 구조를 띄고 있음
- Q와 K의 비교 함수는 Dot-Product와 Scale로 이루어짐
- Mask를 이용해 Illegal connection의 attention을 금지
- Softmax로 유사도를 0 ~ 1의 값으로 Normalize
- 유사도와 V를 결합해 Attention value 계산

$$Q = [\boldsymbol{q}_0, \boldsymbol{q}_1, \dots, \boldsymbol{q}_n]$$

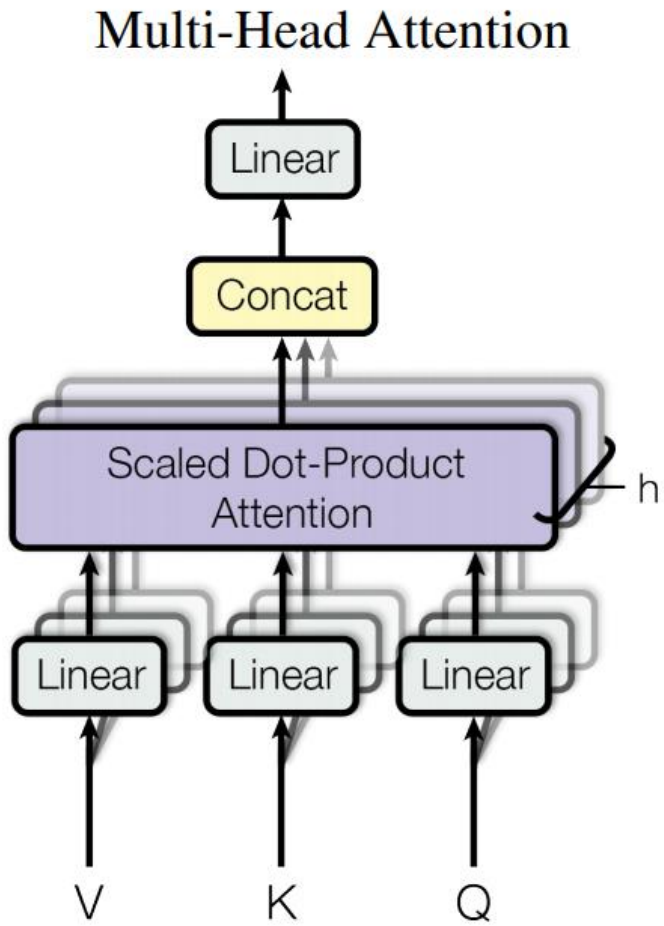
$$K = [\boldsymbol{k}_0, \boldsymbol{k}_1, \dots, \boldsymbol{k}_n]$$

$$V = [\boldsymbol{v}_0, \boldsymbol{v}_1, \dots, \boldsymbol{v}_n]$$

$$C = \text{softmax}\left(\frac{K^T Q}{\sqrt{d_k}}\right)$$

$$\boldsymbol{a} = C^T V = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

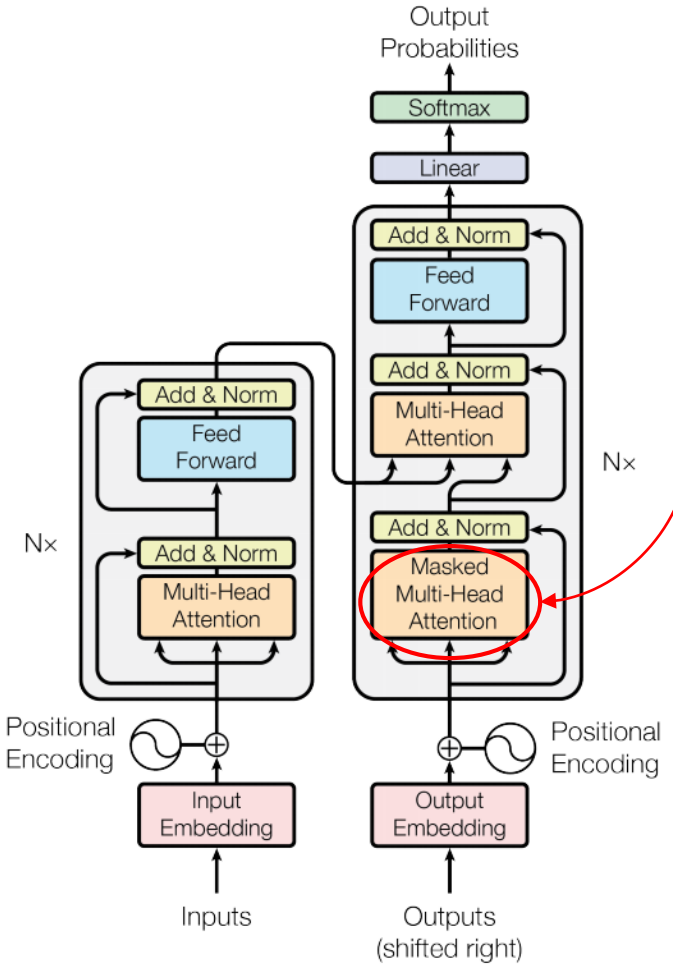
Multi-Head Attention



- Linear 연산 (Matrix Mult)를 이용해 Q, K, V의 차원을 감소
  Q와 K의 차원이 다른 경우 이를 이용해 동일하게 맞춤
- h개의 Attention Layer를 병렬적으로 사용 – 더 넓은 계층
- 출력 직전 Linear 연산을 이용해 Attention Value의 차원을 필요에 따라 변경
- 이 메커니즘을 통해 병렬 계산에 유리한 구조를 가지게 됨

$$\text{Linear}_i(V) = VW_{V,i} \quad W_{V,i} \in \mathbb{R}^{d_v \times d_{model}}$$
$$\text{Linear}_i(K) = KW_{K,i} \quad W_{K,i} \in \mathbb{R}^{d_k \times d_{model}}$$
$$\text{Linear}_i(Q) = QW_{Q,i} \quad W_{Q,i} \in \mathbb{R}^{d_q \times d_{model}}$$

Masked Multi-Head Attention



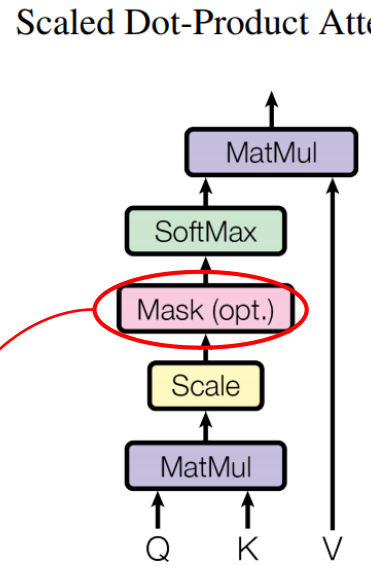
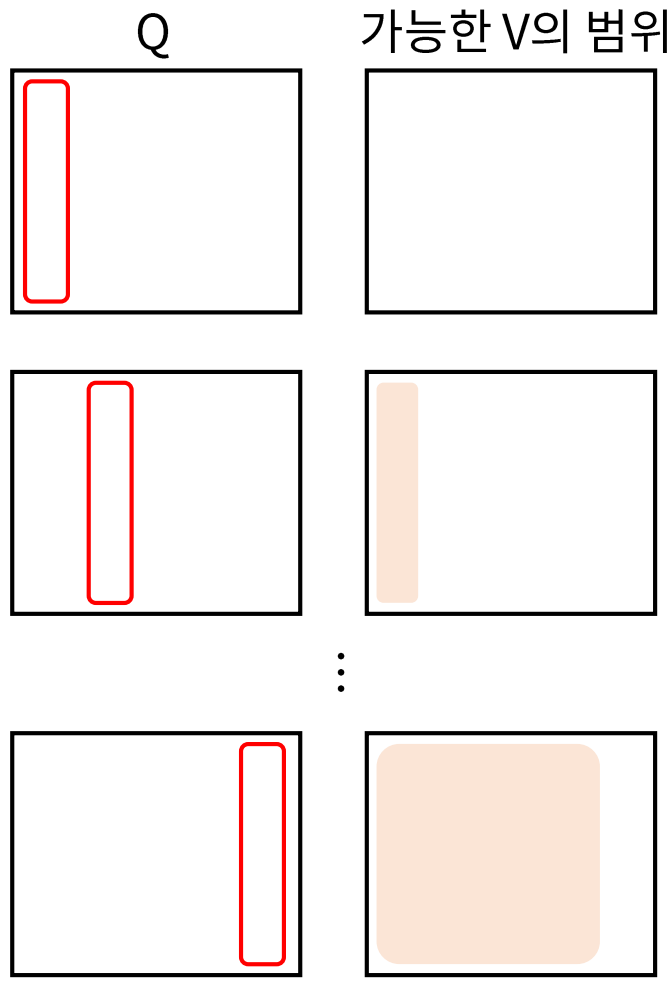


Figure 1: The Transformer - model architecture.



Self-Attention에서 자기 자신을 포함한 미래의 값과는 Attention을 구하지 않기 때문에, Masking을 사용한다.

Multi-Head Attention in Action

Self-Attention

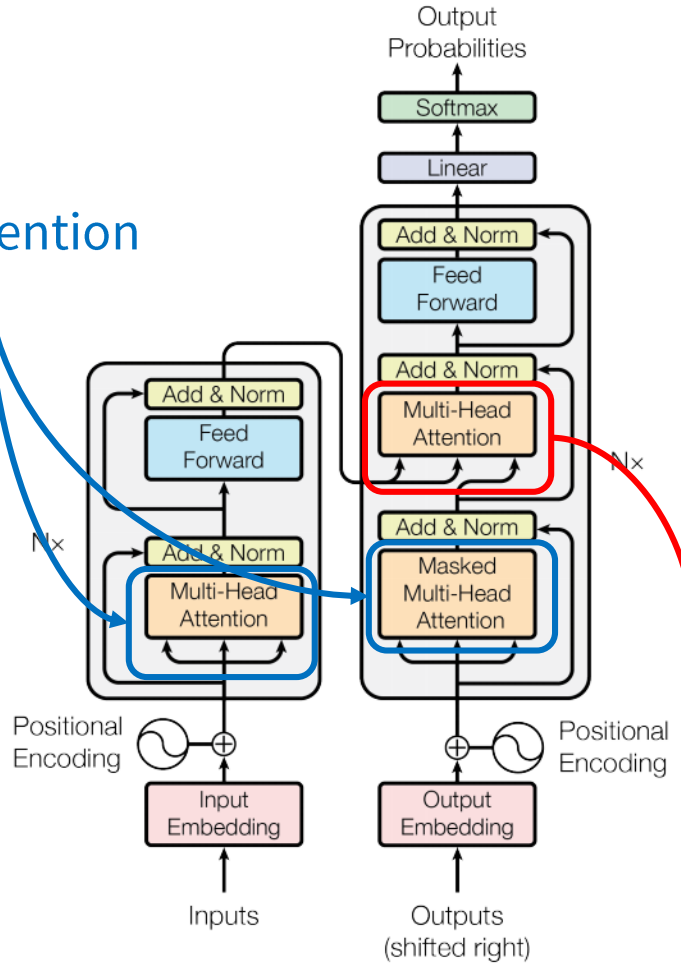
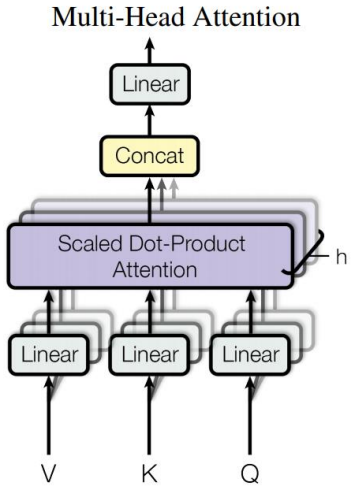
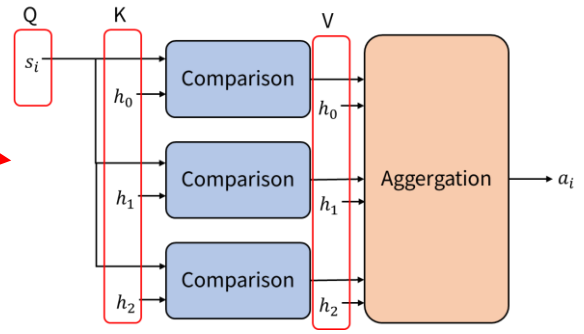


Figure 1: The Transformer - model architecture.



V, K, Q 순으로 표기됨을 확인



Seq2seq의 Attention과 동일한 구조

Position-wise Feed-Forward

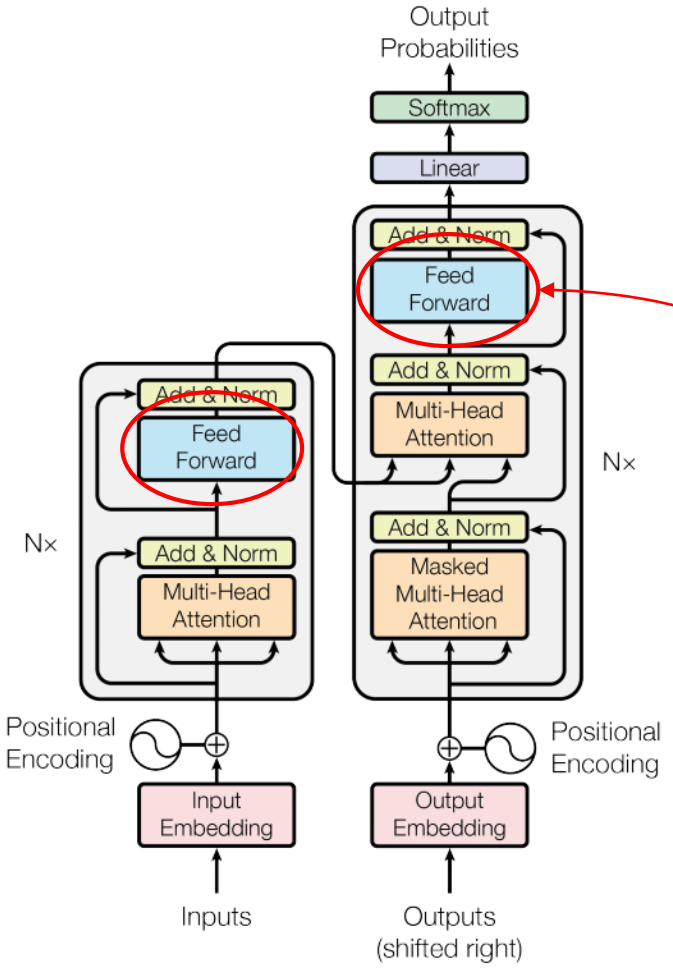
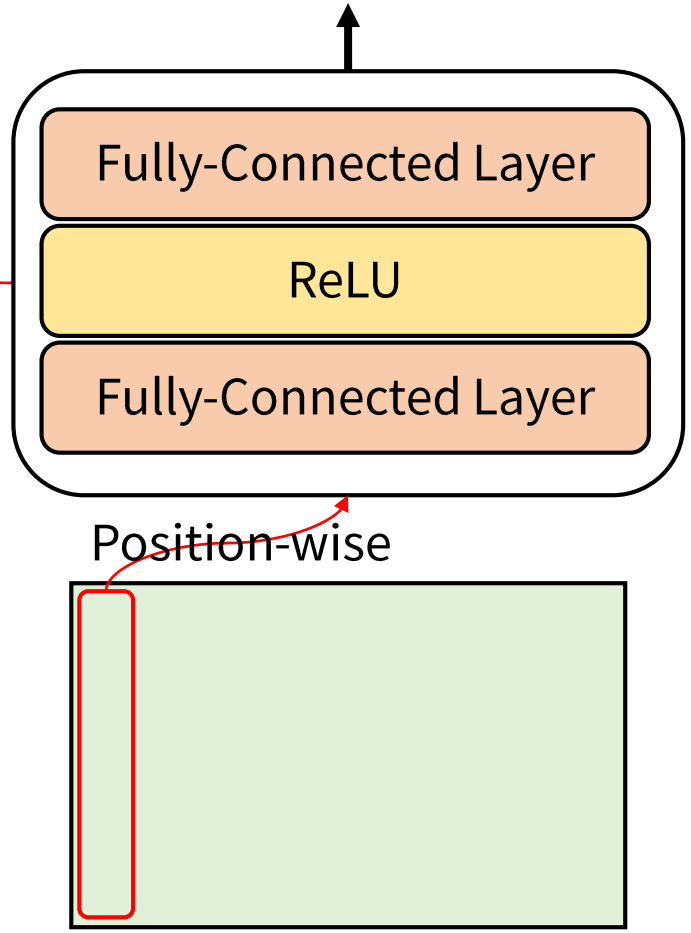


Figure 1: The Transformer - model architecture.



Add & Norm

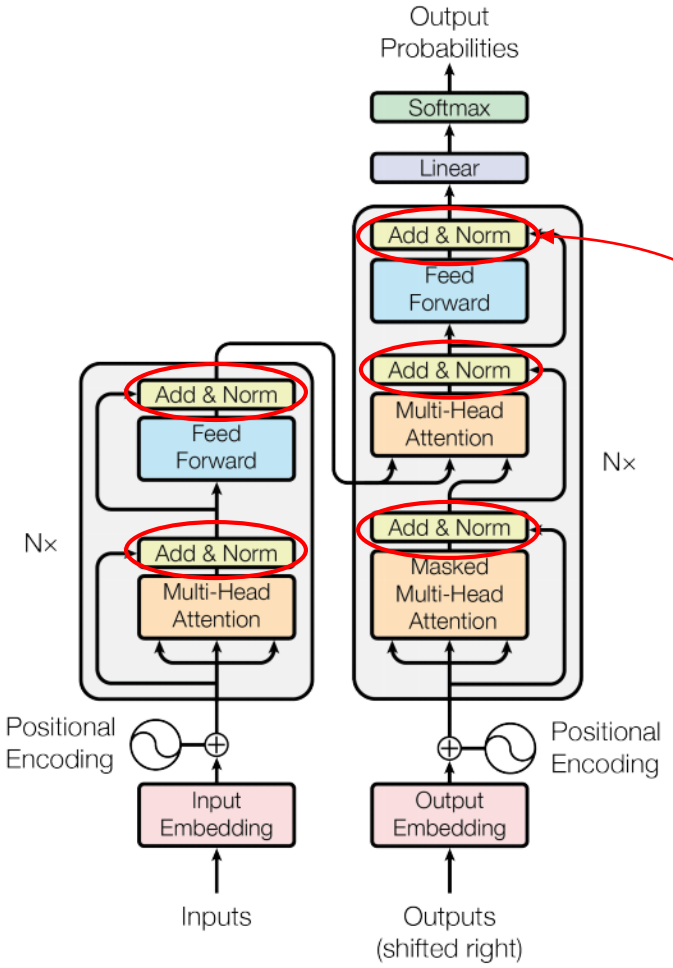
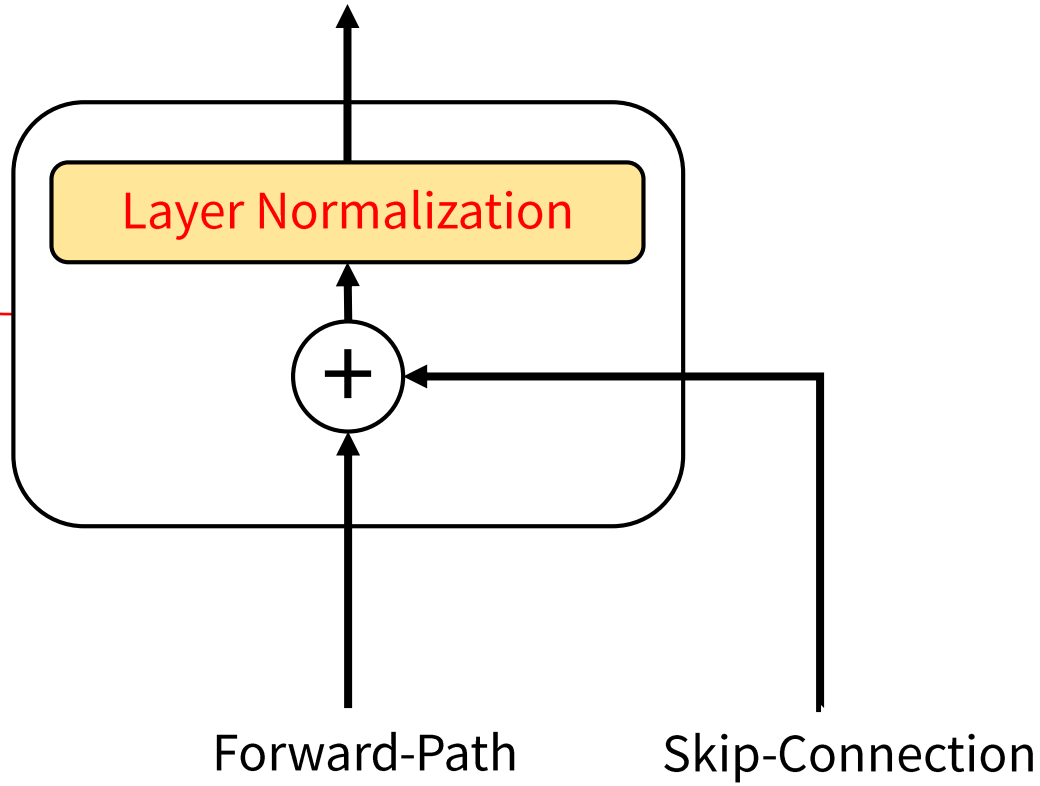


Figure 1: The Transformer - model architecture.



Output Softmax

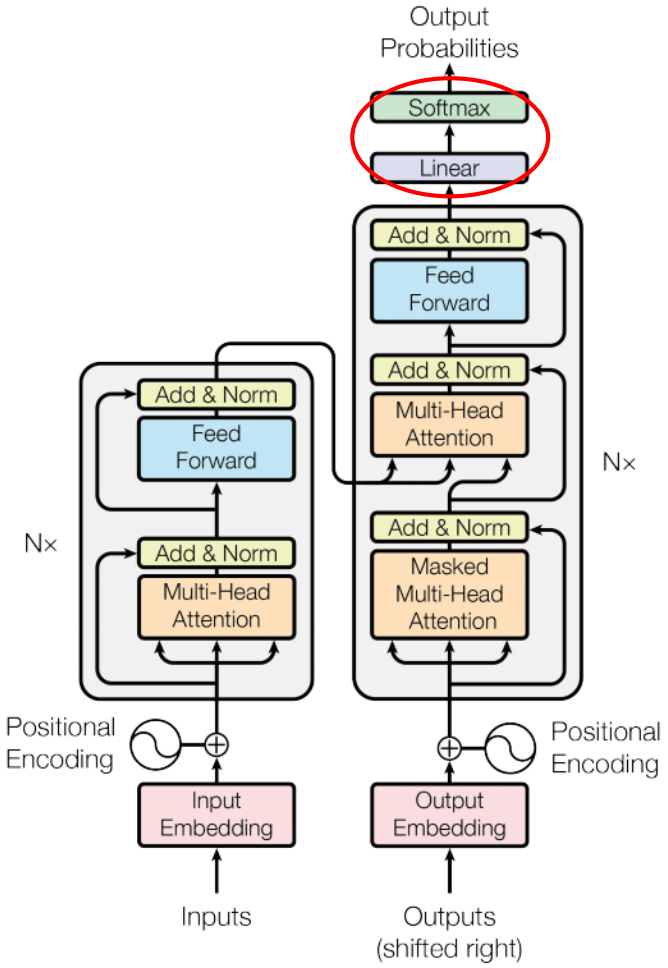


Figure 1: The Transformer - model architecture.

- Linear 연산을 이용해 출력 단어 종류의 수에 맞춤
- Softmax를 이용해 어떤 단어인지 Classification문제 해결

Attention is really all you need

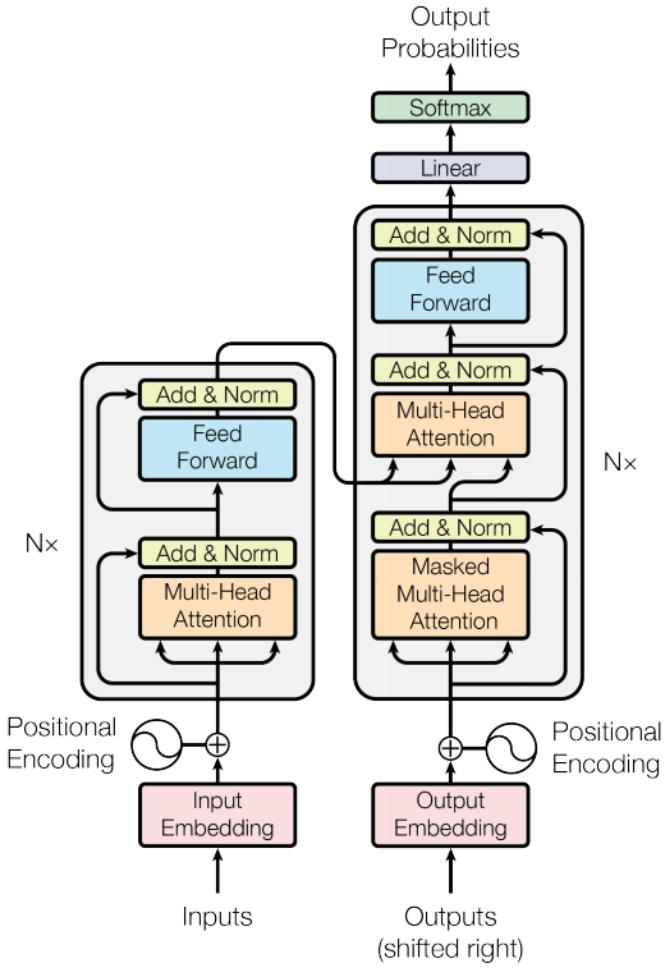


Figure 1: The Transformer - model architecture.

다시 한번 아래 내용을 곱씹어 봅시다.

- Seq2seq와 유사한 Transformer 구조 사용
- 제안하는 Scaled Dot-Product Attention과, 이를 병렬로 나열한 Multi-Head Attention 블록이 알고리즘의 핵심
- RNN의 BPTT와 같은 과정이 없으므로 병렬 계산 가능
- 입력된 단어의 위치를 표현하기 위해 Positional Encoding 사용