Attention is all you need

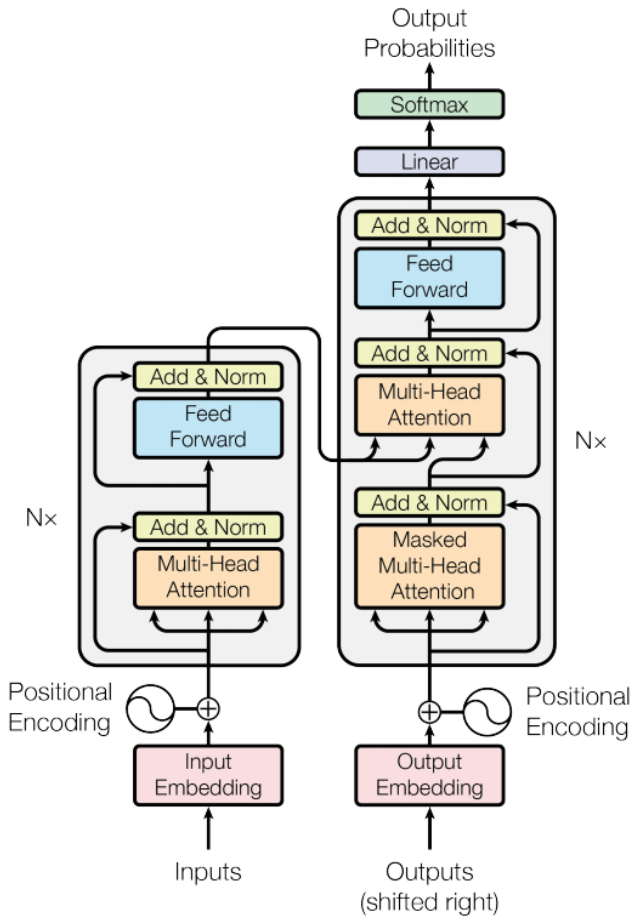


Figure 1: The Transformer - model architecture.

Transformer model에 사용된 수식도 하나하나 되짚어보면서 알아보자.

Inputs & Output

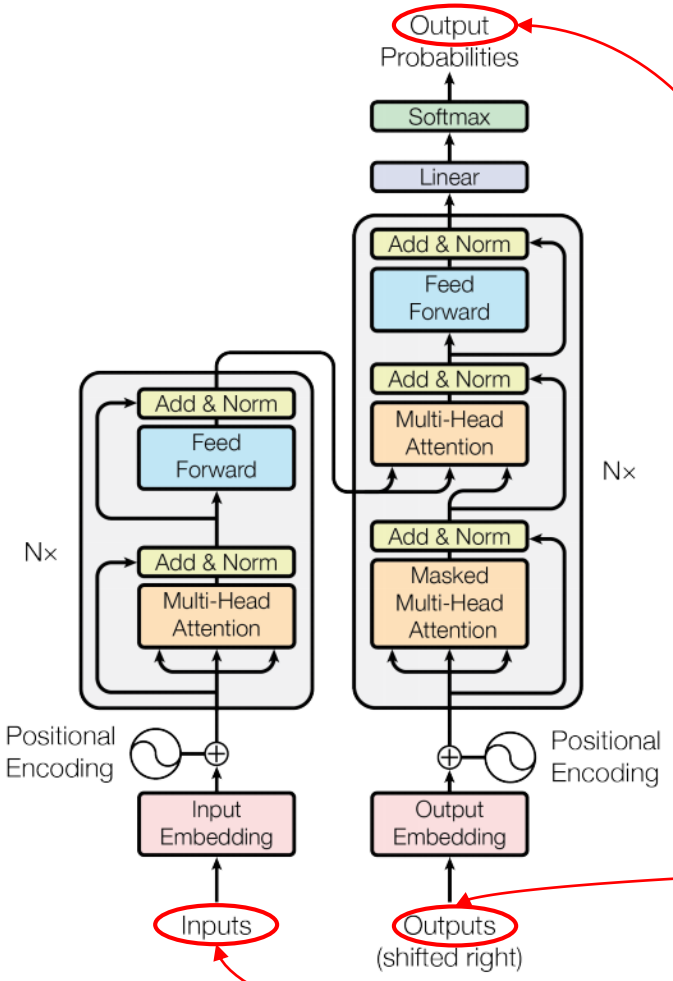


Figure 1: The Transformer - model architecture.

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

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

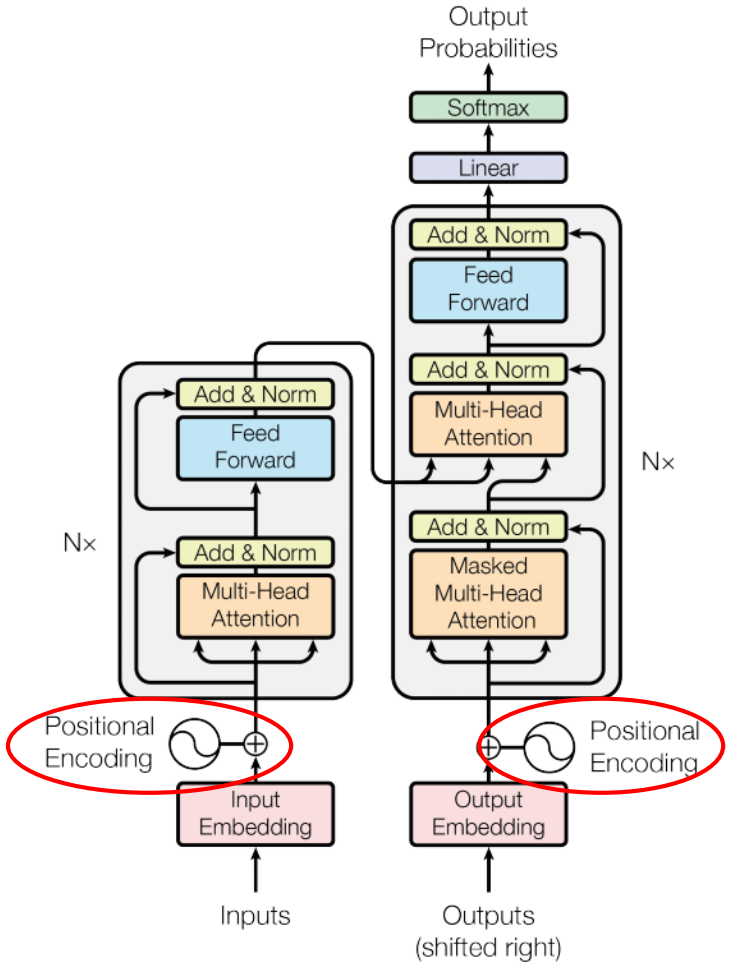
$\hat{Y} = [\mathbf{0}, y_0, \dots y_{m-1}]$

출력 Sequence 길이 m

출력 단어의 가짓수

<SOS>

Positional Encoding



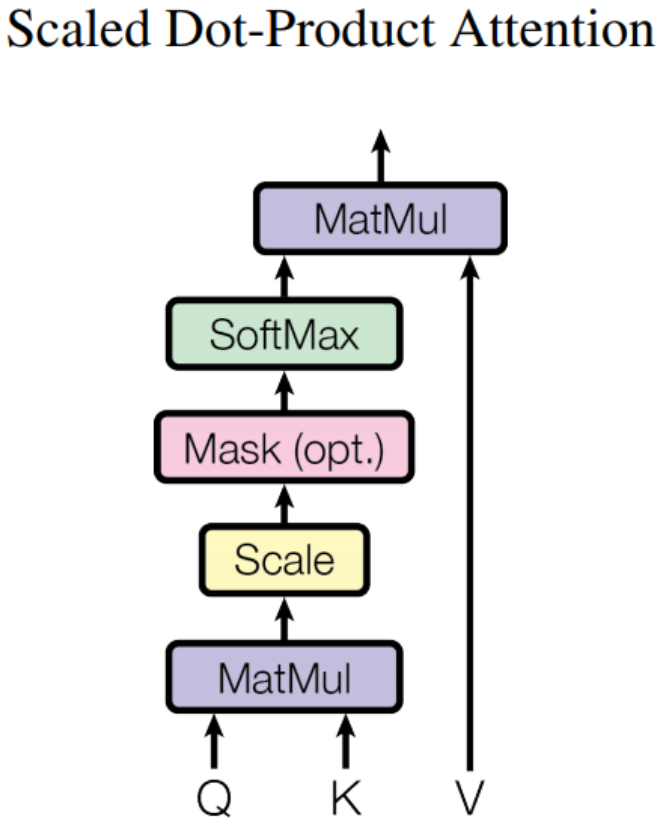
- 시간적 위치별로 고유의 Code를 생성하여 더하는 방식
- 전체 Sequence의 길이 중 상대적 위치에 따라 고유의 벡터를 생성하여 Embedding된 벡터에 더해줌

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

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

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

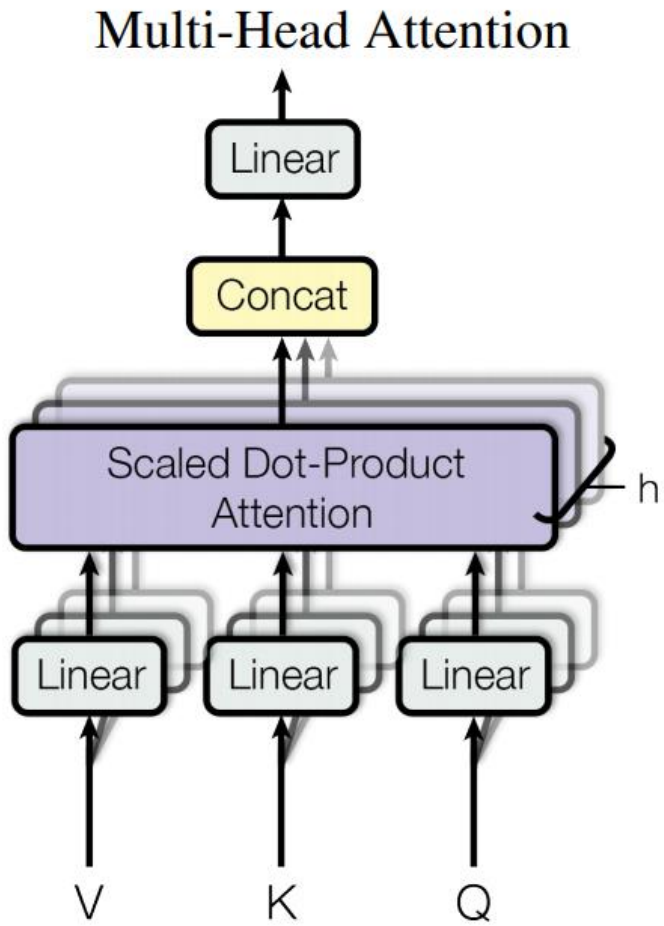
$$K = [\boldsymbol{k}_0, \boldsymbol{k}_1, \dots, \boldsymbol{k}_n]$$

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

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

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

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

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

Transformer Model Review

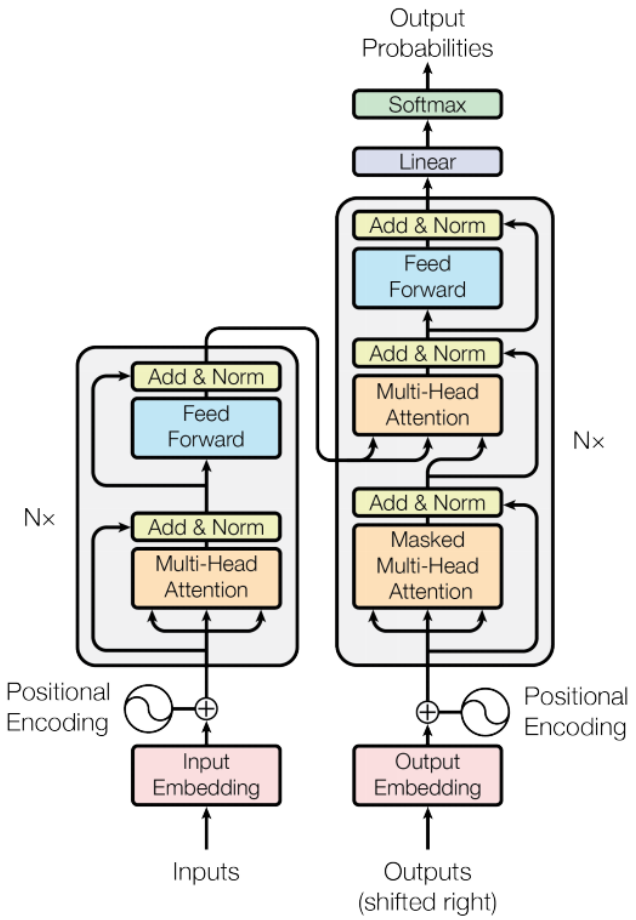


Figure 1: The Transformer - model architecture.

이제 꽤나 친숙하게 느껴질겁니다.아직 다루지 않은 Layer Normalization은 8장에서 다룹니다.