Transformer

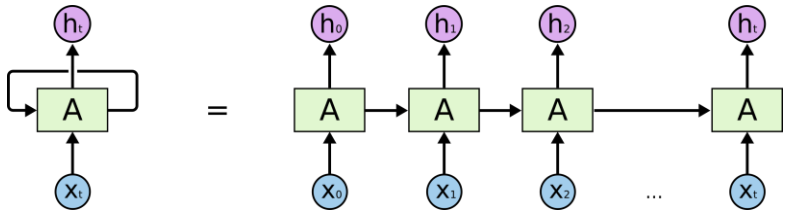
고 지 형

Transformer

Remind: Recurrent Neural Network

RNN의 한계: Long-term dependency

- RNN의 학습 과정: 시퀀스를 입력 받아 hidden state를 순차적으로 업데이트
- 역전파 과정에서 기울기가 소실돼, 오래된 시점 정보를 미래 시점에 잘 전달하지 못하는 long-term dependency 문제 발생
- Bidirectional RNN, LSTM, GRU 등 문제를 일정 부분 해결한 모델이 등장했으나, 태생적 한계 존재



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Transformer

Attention is All You Need

- Self-attention 메커니즘을 통해 RNN의 long-term dependency 문제 해결
- 순차적 학습의 형태가 아니므로 빠른 학습 속도 확보
- 높은 성능으로 LSTM, GRU의 자리를 대체하는 중
- 언어 모델

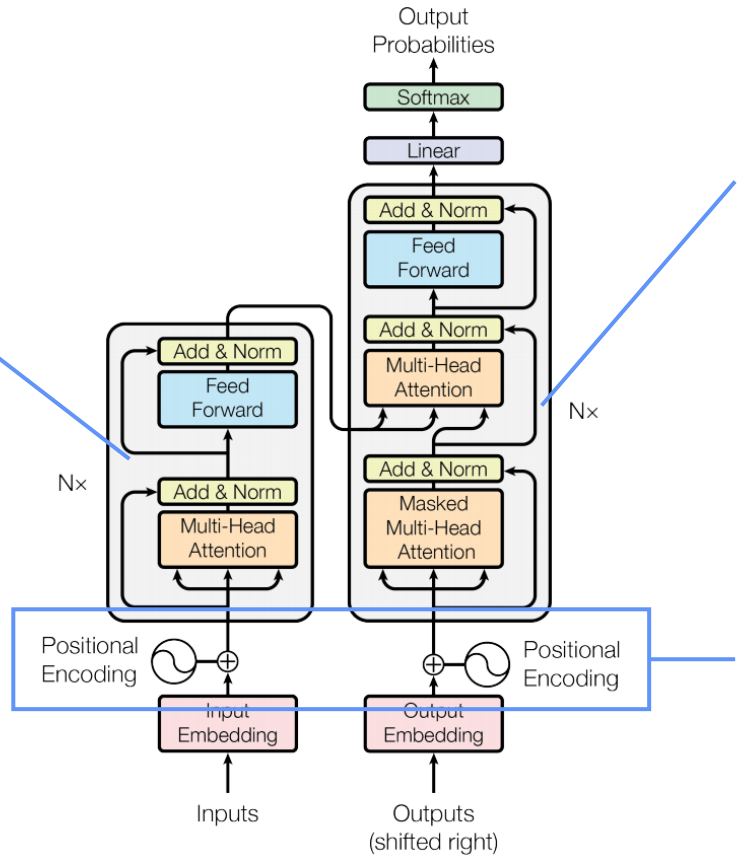
http://jalammar.github.io/illustrated-transformer/

Transformer

Structure

Encoder

- Attention을 활용한 입력 단어 간 관계성 파악
- 단어 간 관계가 함축된 hidden state 벡터를 출력



Decoder

- Encoder를 통해 출력된 hidden state를 고려하여 번역할 단어를 예측

Positional Encoding

- 단어의 순서가 표현된 벡터
- Self-attention은 순서를 고려하지 않아 positional encoding 벡터를 별도로 활용

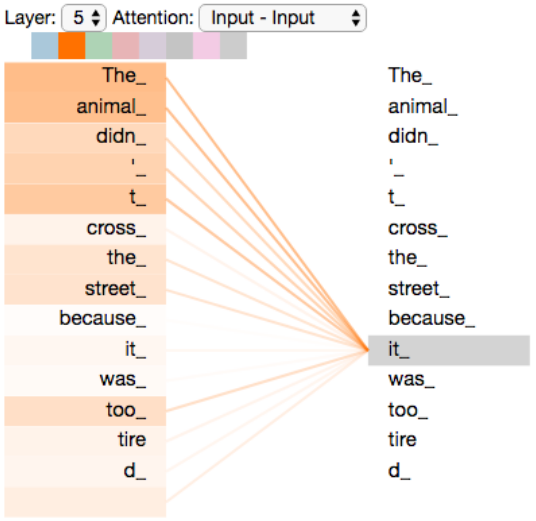
Ashish Vaswani et al., 2017, Attention is All You Need

Transformer

Self-attention

- 입력된 단어 각각에 대한 관계성을 파악하는 메커니즘
- 특정 단어를 입력 받았을 때, 해당 단어가 다른 어떤 단어와 관련되어 있는지 파악할 수 있음

"The animal didn't cross the street because it was too tired"

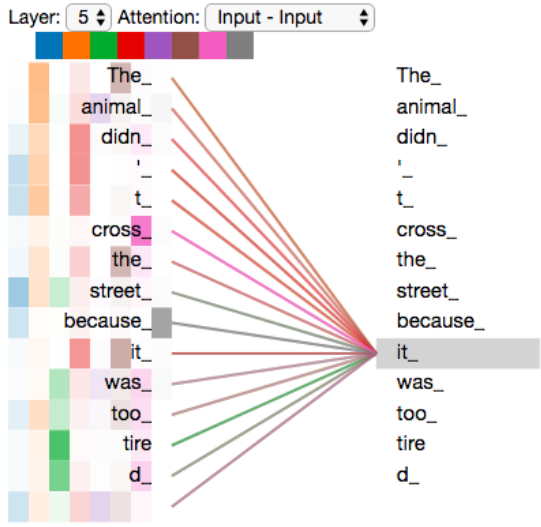


http://jalammar.github.io/illustrated-transformer/

Multi-head Attention

- Transform 모델은 다수의 self-attention을 활용
- 특정 단어를 입력 받았을 때, 해당 단어를 여러 관점에서 바라볼 수 있음

"The animal didn't cross the street because it was too tired"



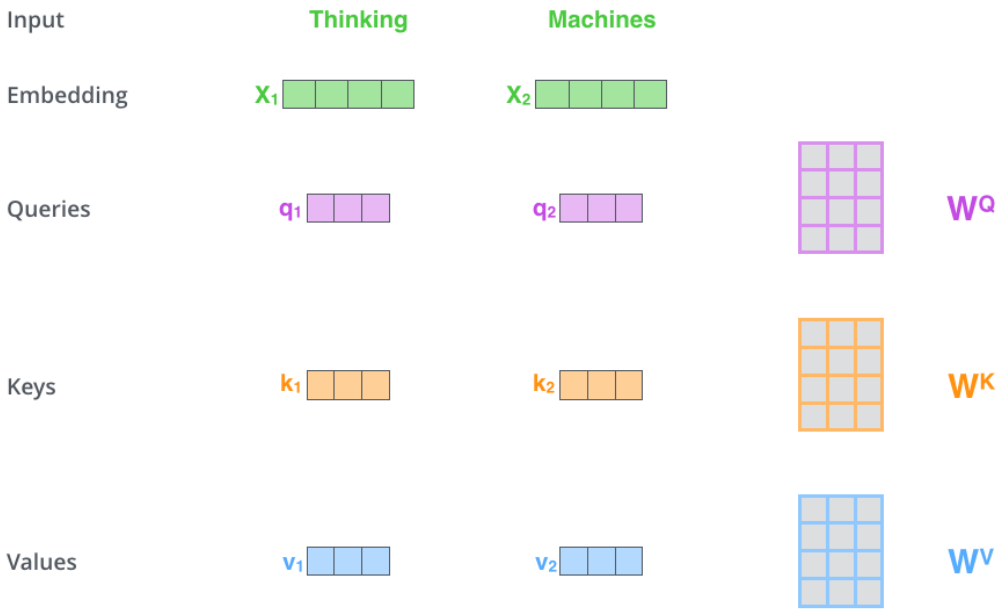
http://jalammar.github.io/illustrated-transformer/

Transformer

Self-attention

계산 과정

1. 입력된 단어를 임베딩
2. 각 임베딩 벡터 x에 대한 Query, Key, Value 벡터를 생성
    - 각각은 Wq, Wk, Wv의 가중치 행렬과 임베딩 벡터를 곱해서 구함
    - *'이 단어랑 나머지 다른 단어들이랑 얼마나 연관되어 있어?'*
    - Query: 위 질문을 던지는 역할
    - Key: 위 질문에 대한 답을 구하기 위해 활용되는 역할
    - Value: 위 질문에 대한 답을 담는 역할



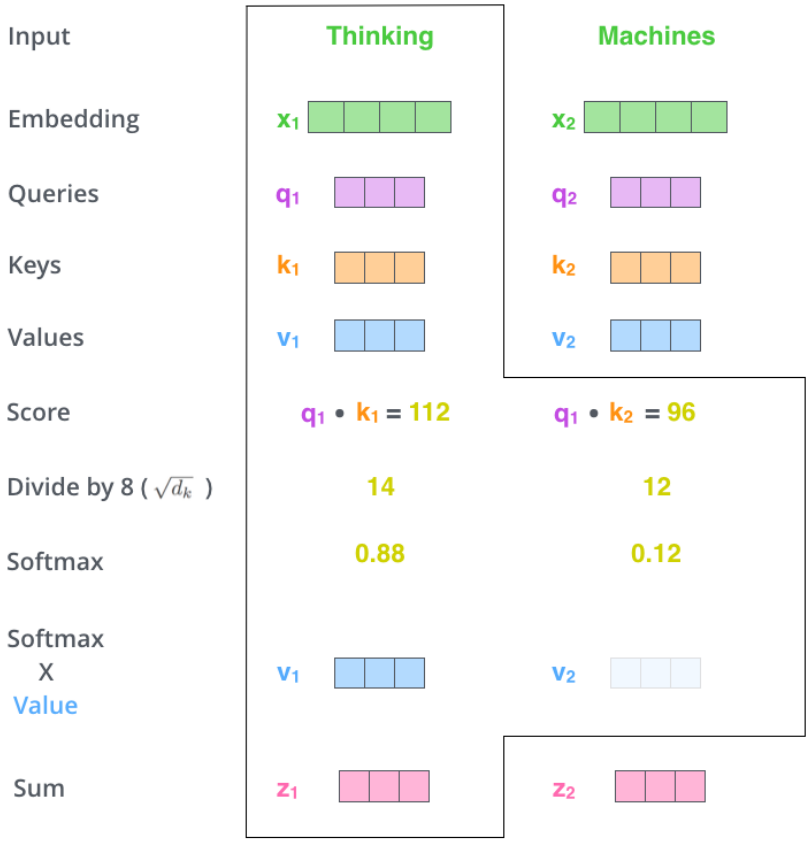
http://jalammar.github.io/illustrated-transformer/

Transformer

Self-attention

계산 과정

3. 기준 단어의 Query 벡터와 각 단어의 Key 벡터를 내적
    - 기준 단어 자신의 Key 벡터도 내적
    - 입력한 단어 수 만큼의 내적 값을 얻음
    - 기준 단어와 연관성이 높을 수록 내적 값도 높음
4. 과정 3에서 구한 내적 값으로 이루어진 벡터에 소프트맥스 적용
5. 각 소프트맥스 성분을 가중치로 각 단어의 Value 벡터를 가중합
6. 가중합한 벡터가 기준 단어에 대한 Attention 벡터



http://jalammar.github.io/illustrated-transformer/

Transformer

Self-attention

행렬 관점에서의 계산 과정

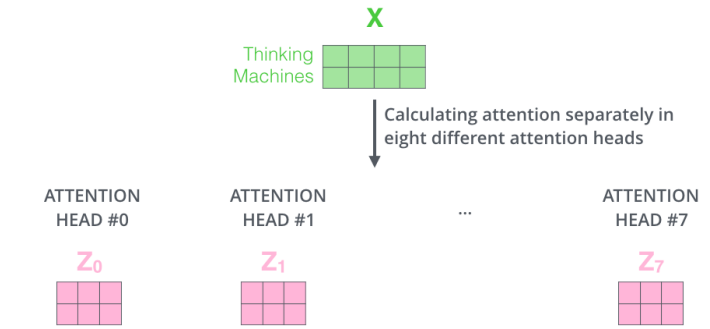
1. 임베딩된 시퀀스 행렬로부터 Query 행렬, Key 행렬, Value 행렬을 구함
2. Query 행렬과 Key 행렬을 행렬곱
    - 각 행은 기준 단어와 모든 단어를 내적한 결과
3. 행렬곱 결과에 소프트맥스를 취한 행렬을 Value 행렬과 행렬곱
    - 소프트맥스는 행을 기준으로 취함
4. Value 행렬과 행렬곱한 결과가 Attention 행렬
    - 각 행이 각 단어에 대한 Attention 벡터

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

http://jalammar.github.io/illustrated-transformer/

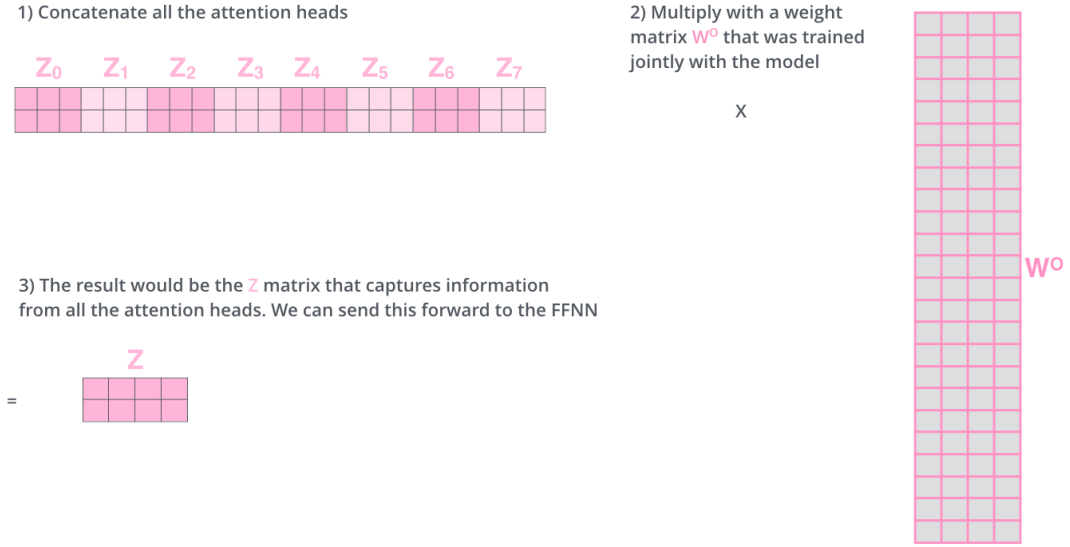
Transformer

Self-attention



Multi-Head Attention

- 다수의 Attention 결과를 Concatenate
- 해당 크기만큼의 가중치 행렬을 행렬곱한 결과를 Attention으로 사용



http://jalammar.github.io/illustrated-transformer/

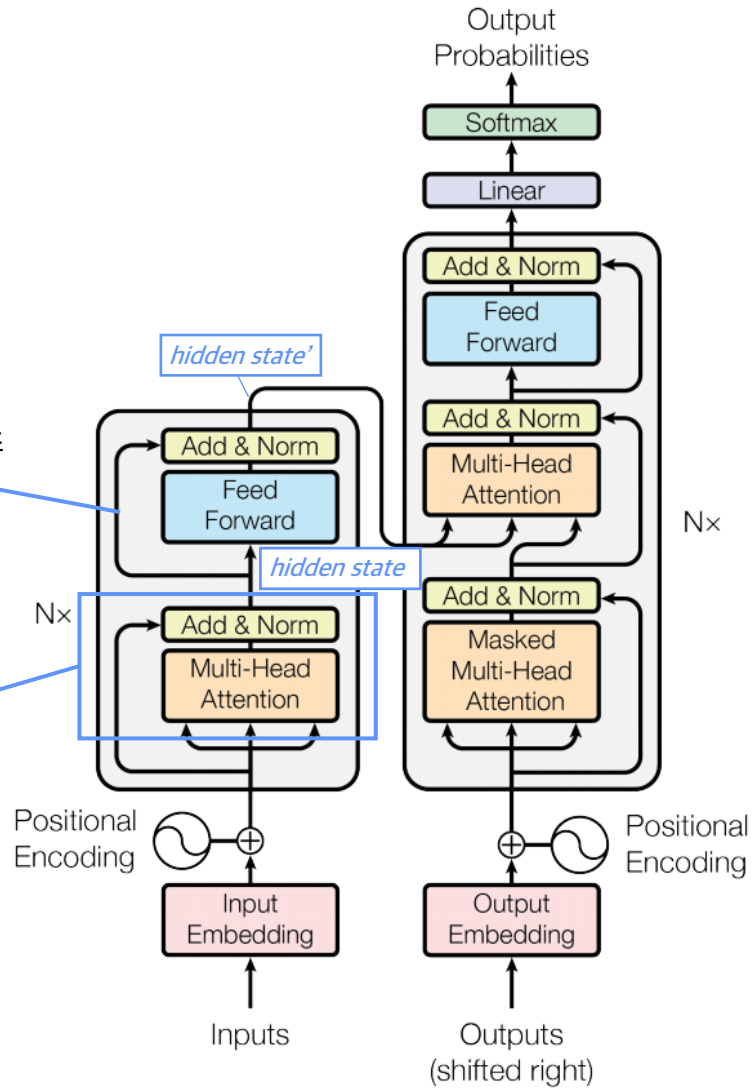
Transformer

Encoder: 입력 단어 간 관계성 파악

Residual Connection

- 출력값과 앞선 입력값을 더하는 구조
- 입력값의 잔차에 대한 학습
- 이전 정보에 대한 유지
- Positional Encoding 정보 유지

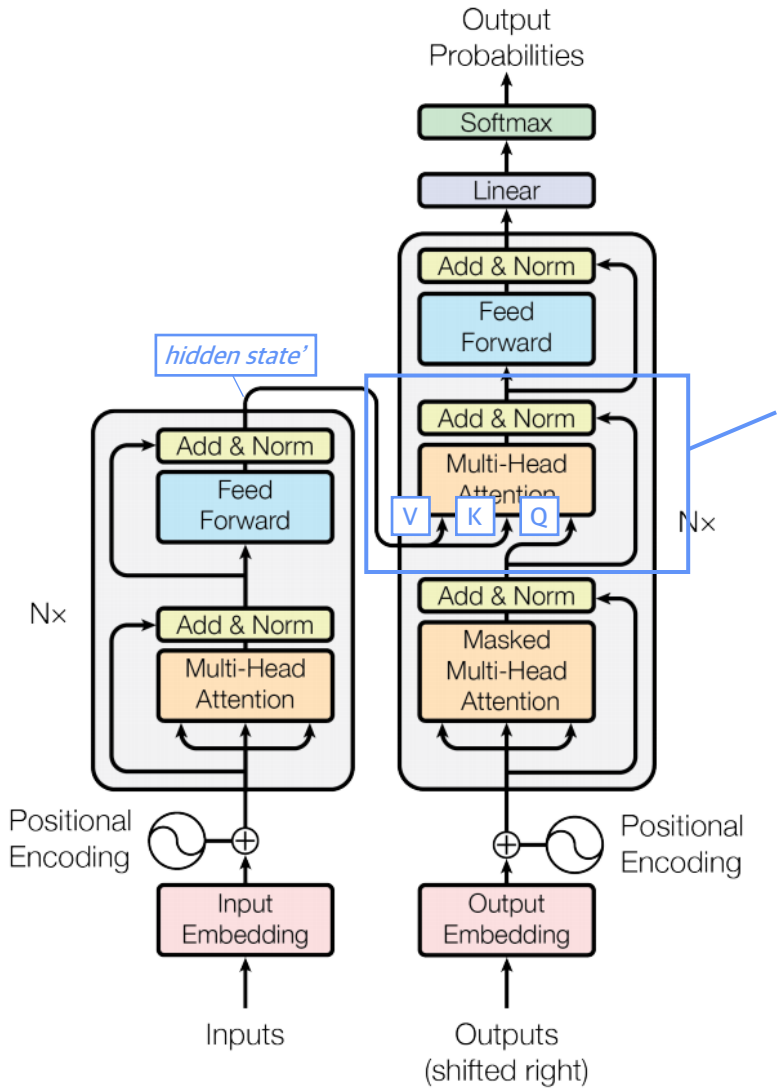
Multi-Head Attention



Ashish Vaswani et al., 2017, Attention is All You Need

Transformer

Decoder: 인코더의 정보에 기반한 번역



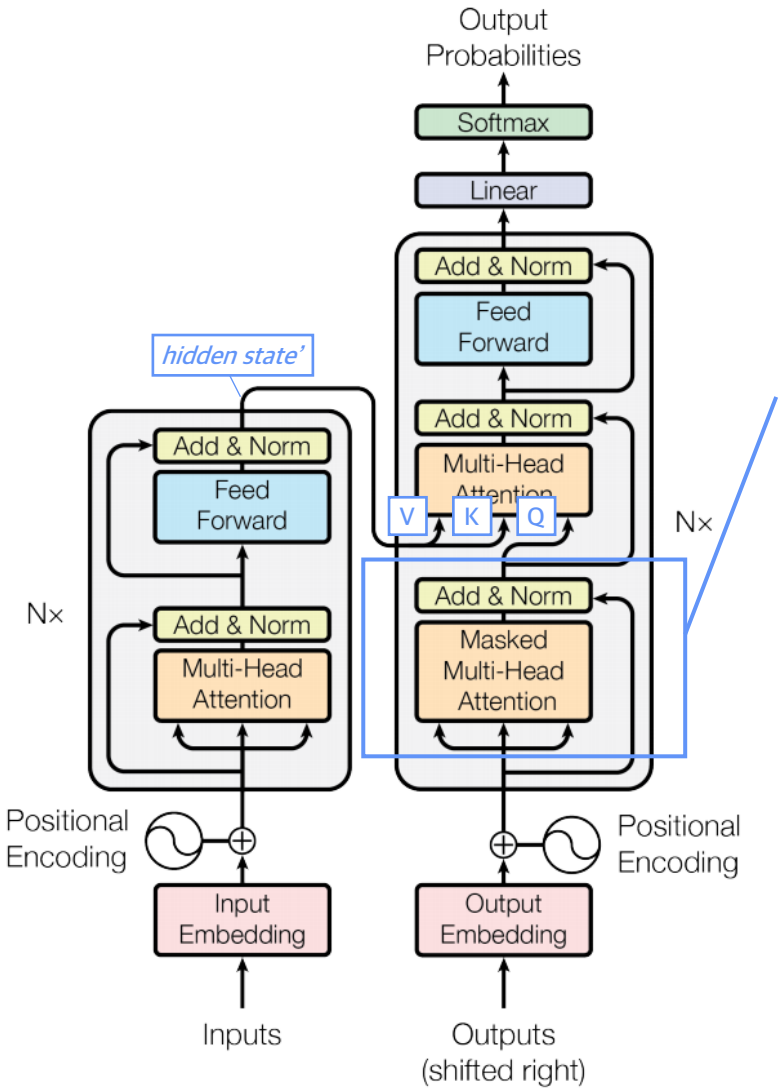
입력 단어의 관계성을 고려한 번역

- 인코더로부터 출력된 hidden state로부터 Key, Value 행렬을 추출
- 디코더의 Masked Multi-Head Attention을 통해 Query 행렬 추출
- *'번역할 이 단어는 번역 전 어떤 단어들과 관련 있을까?'*

Ashish Vaswani et al., 2017, Attention is All You Need

Transformer

Decoder: 인코더의 정보에 기반한 번역



Masked Multi-Head Attention

- 'I love you'를 '난 너를 사랑해'로 번역하는 경우
- '난' 단어를 번역하는 시점에 '너를' 단어는 미래 시점의 단어로, '난'의 예측에 관여할 수 없음
- Attention 과정에서 이러한 Cheating 가능성을 방지하기 위해 마스킹 기법 활용

	<시작 토큰>	난	너를
<시작 토큰>	0.91	0.05	0.04
난	0.42	0.47	0.11
너를	0.25	0.31	0.44

Attention

	<시작 토큰>	난	너를
<시작 토큰>	1	MASK	MASK
난	0.47	0.52	MASK
너를	0.25	0.31	0.44

Masked Attention

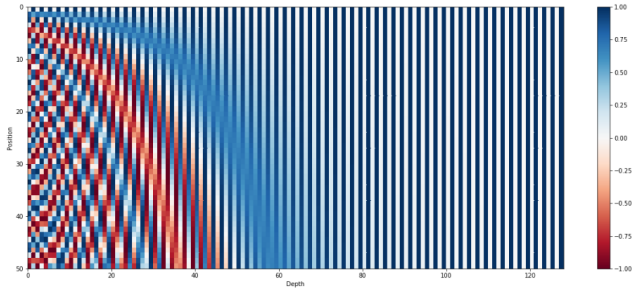
Ashish Vaswani et al., 2017, Attention is All You Need

Transformer

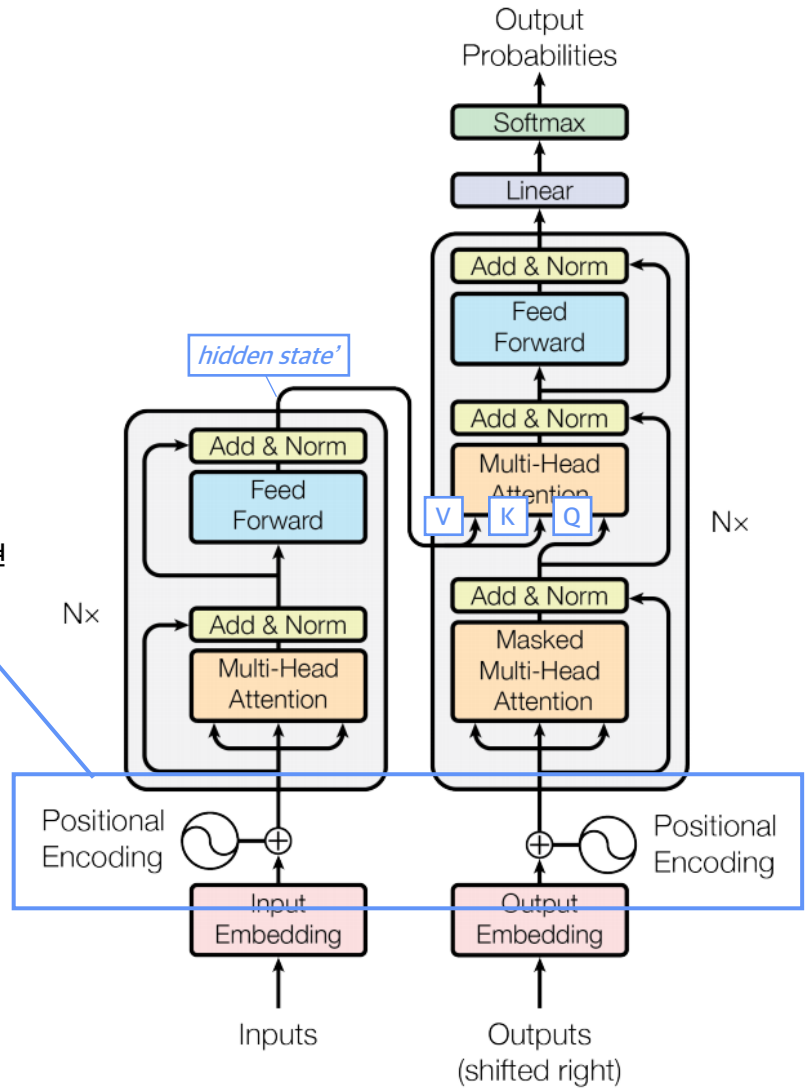
Positional Encoding

Sinusoidal Positional Encoding

- 번역 시 단어 간 순서를 매기기 위한 방법
- 위치 정보를 스칼라가 아닌 벡터로 표현
- 사인, 코사인의 주기함수를 활용
- 단어 위치에 따라 주기를 달리하여 순서를 유일하게(unique) 표현



https://kazemnejad.com/blog/transformer_architecture_positional_encoding/



Ashish Vaswani et al., 2017, Attention is All You Need

감사합니다