Transformer

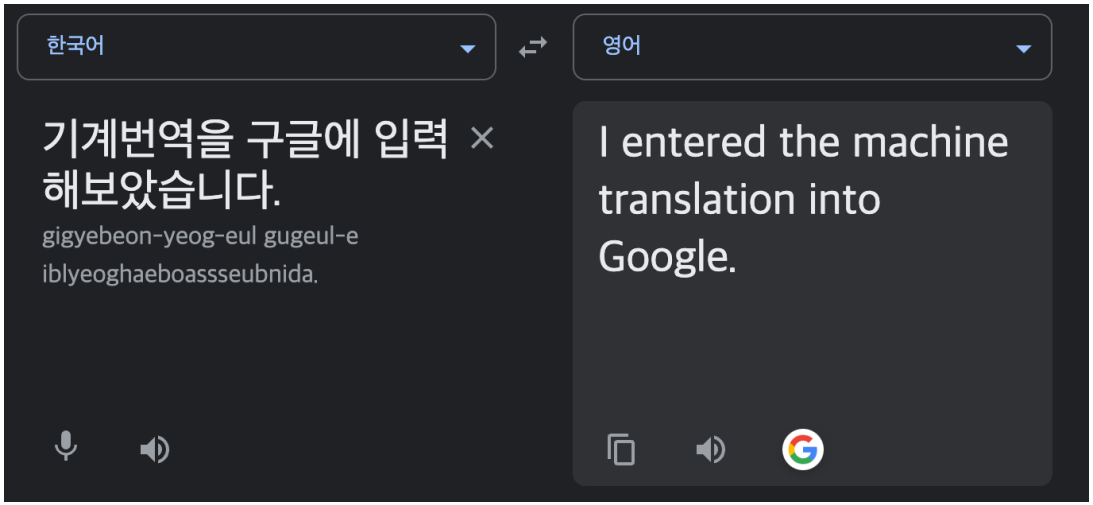
Attention is All You Need

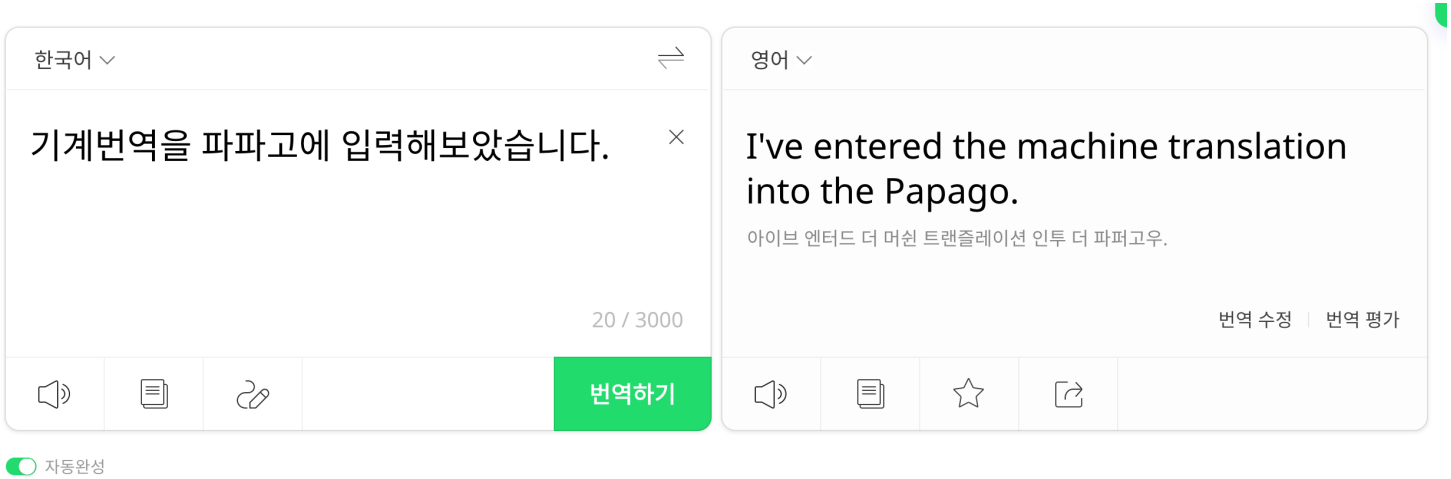
2020010537 백수지

목차

- 기계번역 개념
- Transformer 논문
- Model Architecture
- Encoder
- Decoder

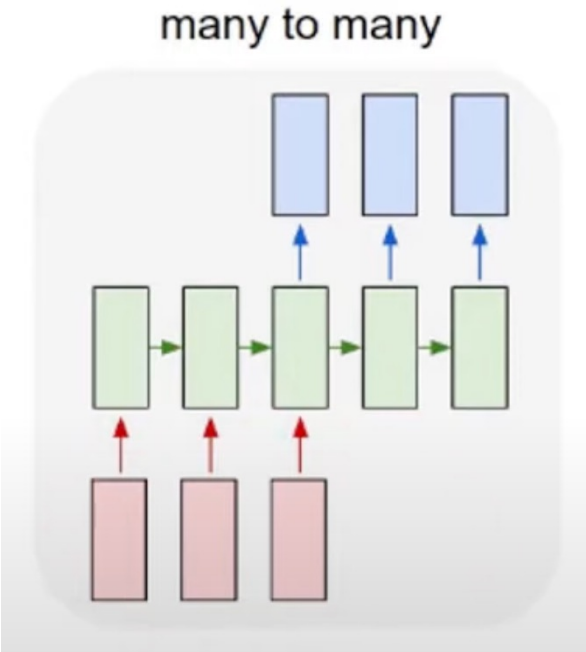
기계번역의 개념


한국어
영어
기계번역을 구글에 입력해보았습니다.
gigyebeon-yeog-eul gugeul-e iblyeoghaeboassseubnida.
I entered the machine translation into Google.


한국어
영어
기계번역을 파파고에 입력해보았습니다.
20 / 3000
번역하기
I've entered the machine translation into the Papago.
아이브 엔터드 더 머쉰 트랜즐레이션 인투 더 파퍼고우.
번역 수정
번역 평가
자동완성

기계번역의 개념

기계 번역의 기본적인 방법 : many to many



나는 치킨을 먹었다.

Output

Neural Network

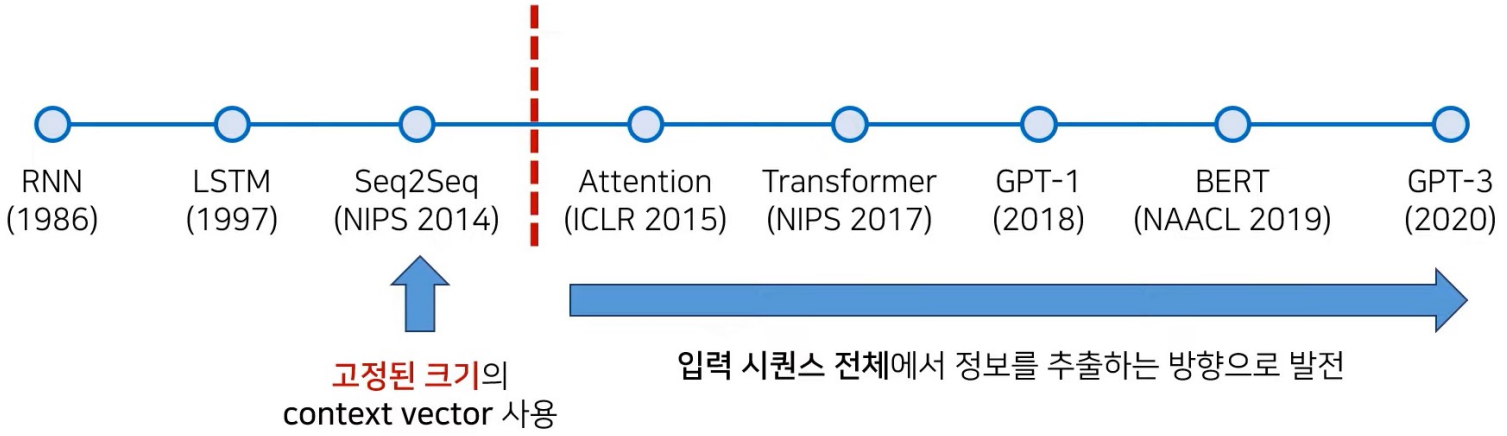
Word Embedding (단어 -> 벡터)

Input words

I ate chicken.

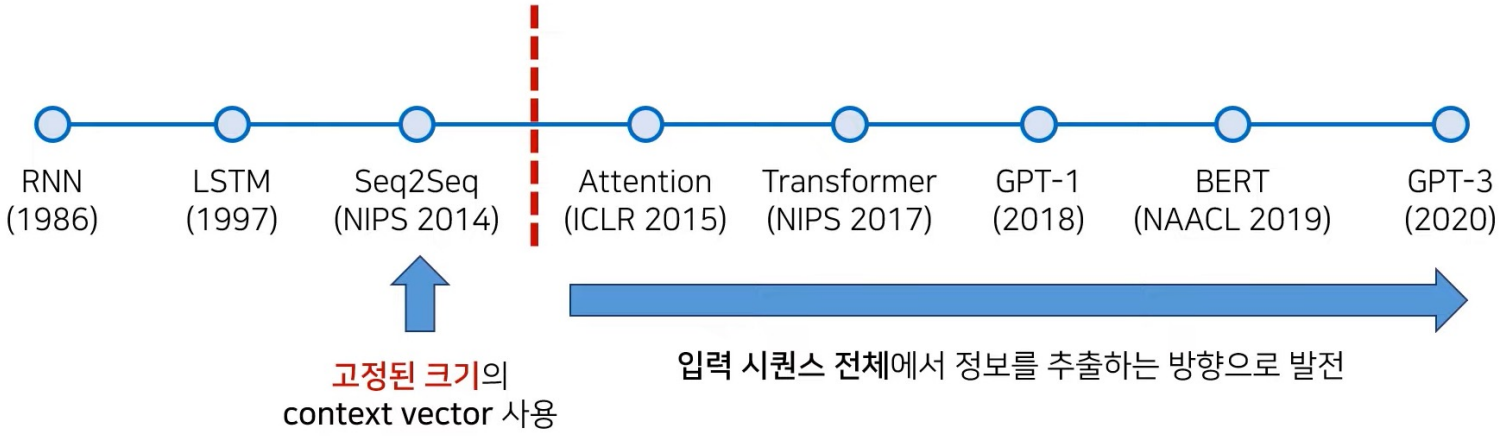
기계번역 발전 Timeline

- 초기에는 **현재 단어를 통해 다음 단어를 유추하는** Recurrent Neural Network (RNN) 순환 신경망을 사용
- 문장이 길어지면 **Vanishing Gradient 문제 (기울기 소실) 발생**
- LSTM에서는 최근 기억 + Input data 를 조합하며 과거의 기억을 수정해가면서 사용하게 됨
- 하나의 셀에 4개의 신경망을 가지고 있어 속도가 느림
- Seq2Seq 모델에서는 고정된 크기의 Context Vector로 문장을 압축시켜 디코딩하는 방식으로 번역을 수행
- 압축으로 인해 하나의 문맥 벡터가 전체의 문맥을 이해하지 못하여 bottleneck 현상 발생, 성능 하락



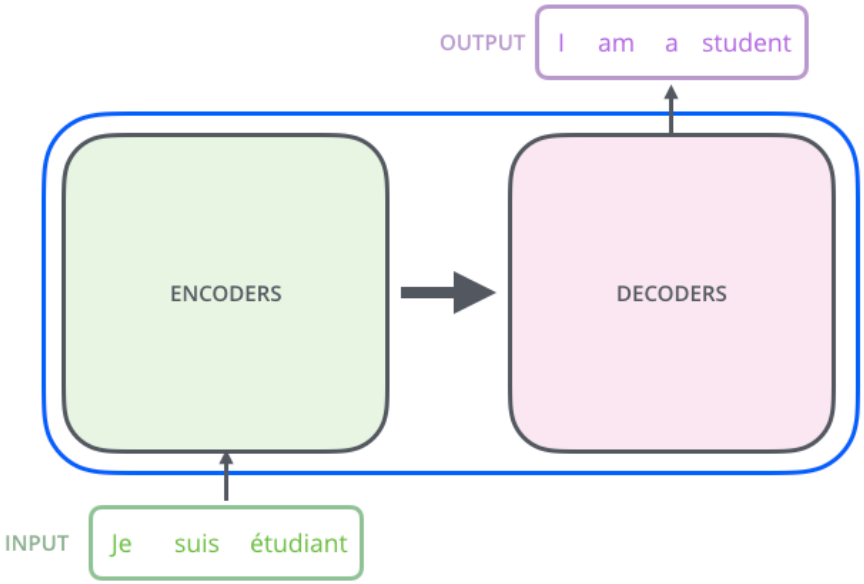
기계번역 발전 Timeline

- 2015년, Attention 개념 등장
  - Sequence distance에 무관하게 서로 간의 dependency 를 모델링
- 2017년 Attention 개념을 업그레이드 하여 Transformer 논문 출시
- 2018년 GPT1, 2019년 BERT와 GPT2, 2020년 GPT3 까지 이어서 발전



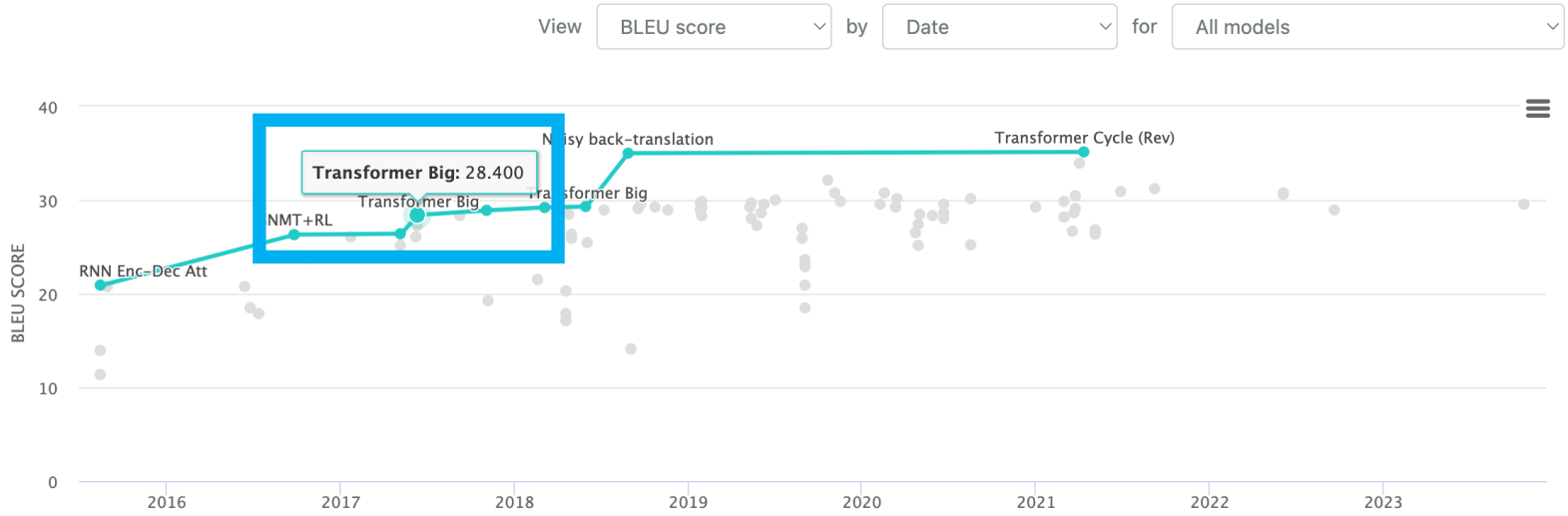
Attention is All You Need

독일어 영어



Attention is All You Need

- 2017년, Google Brain, Research 팀에서 발표한 논문
- 크고 제한적인 데이터들을 효과적으로 처리할 수 있는 기술
- 온전히 **Attention mechanism 만을 활용,** 순환없이 입력값과 출력값 간 Global Dependency (전역 의존성) 을 모델링
- WMT 2014 data set을 이용해서 영어를 독일어로 번역하는 작업, 영어를 불어로 번역하는 작업에서 훨씬 개선된 성능을 보여줌
- 크거나 한정된 학습 데이터를 가지고서도 다른 task들에 성공적으로 일반화될 수 있음을 보임

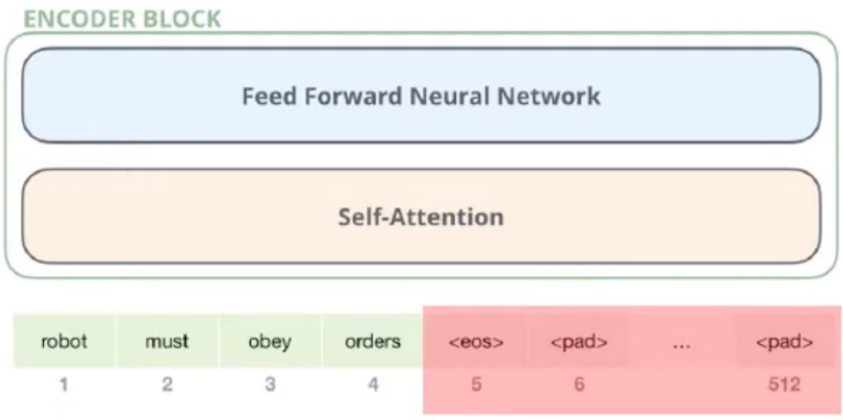


Machine Translation on WMT2014 English-German

Model Architecture

Encoder

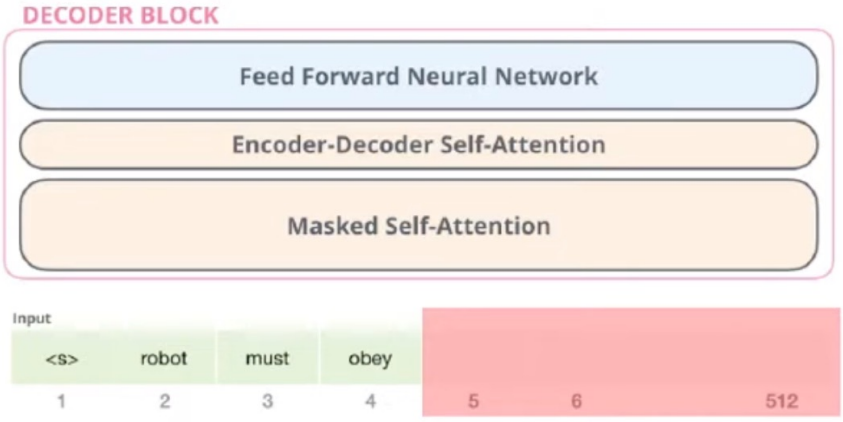
Self-Attention레이어와
Feed Forward Neural Network 로 구성



문장 끝에 end 를 나타내는 token을 넣고, 빈 곳은 padding으로 채움

Decooder

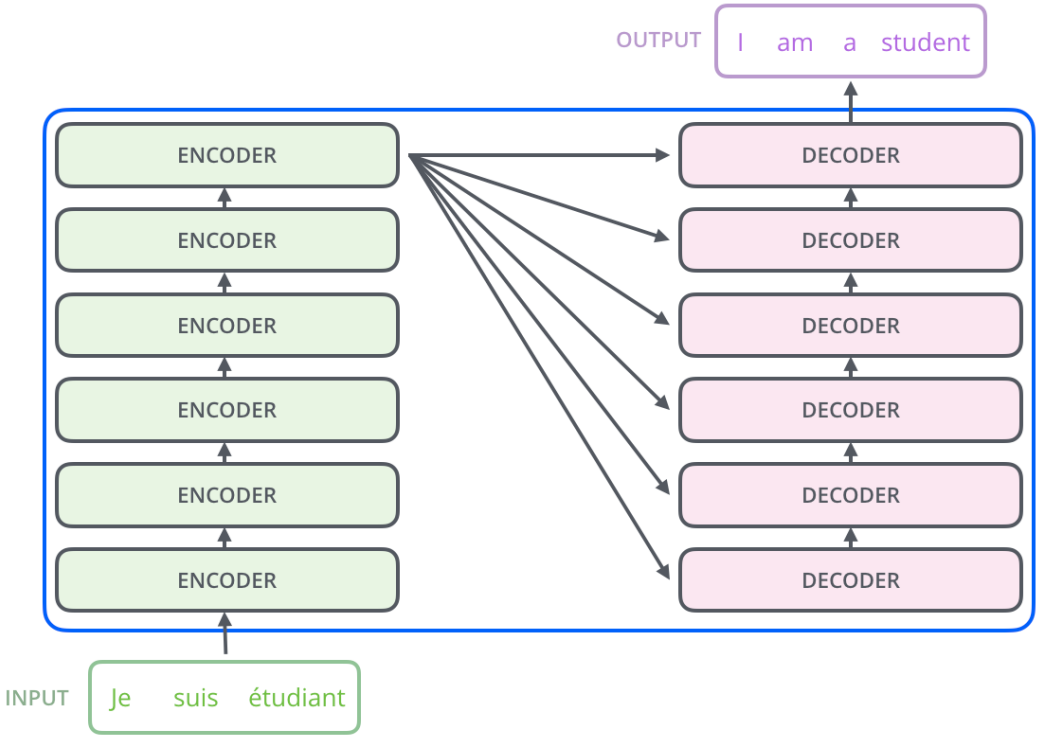
Masked Self-Attention 레이어와
Encoder-Decoder Self-Attention 레이어
Feed Forward Neural Network 로 구성



각 word 를 input으로 넣을 때 치팅을 방지하기 위해, 뒷 부분은 아예 넣어주지 않음

Input Vector의 Dimension : 512

Attention is All You Need



6개의 인코더와 6개의 디코더로 구성

Model Architecture

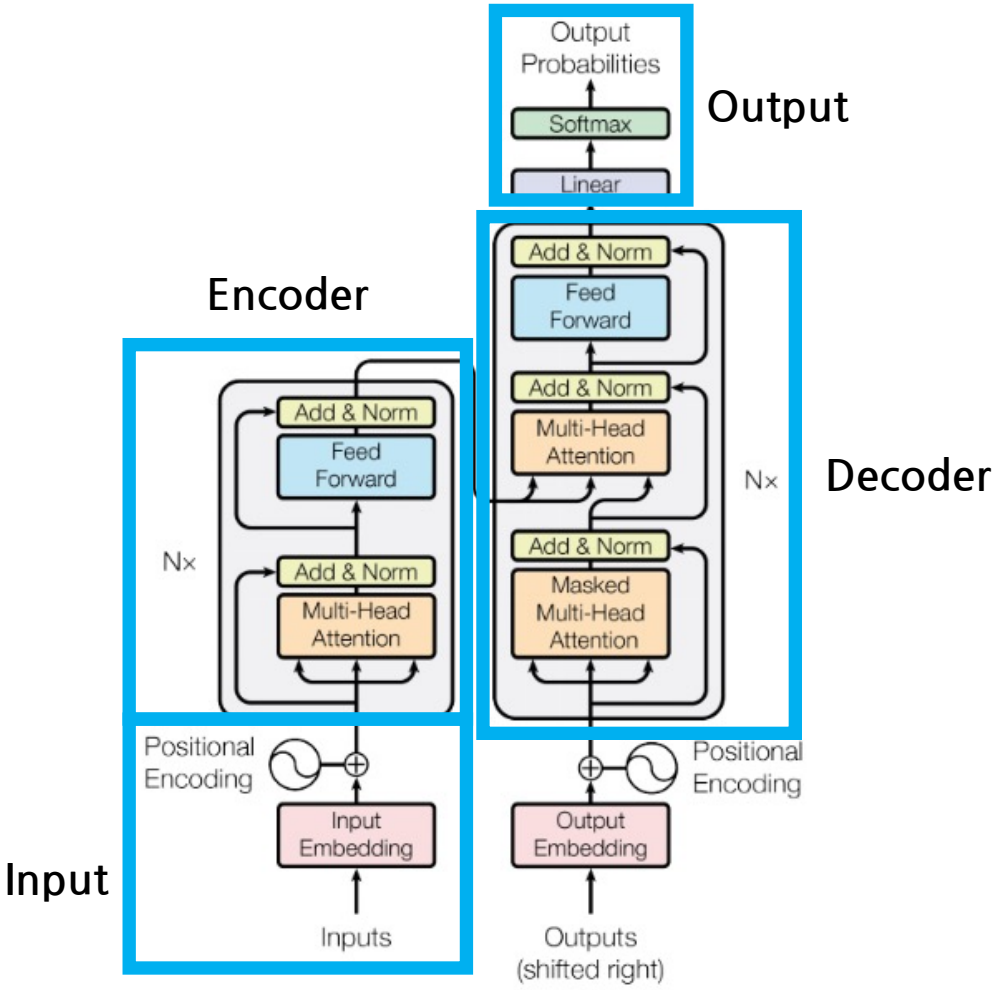
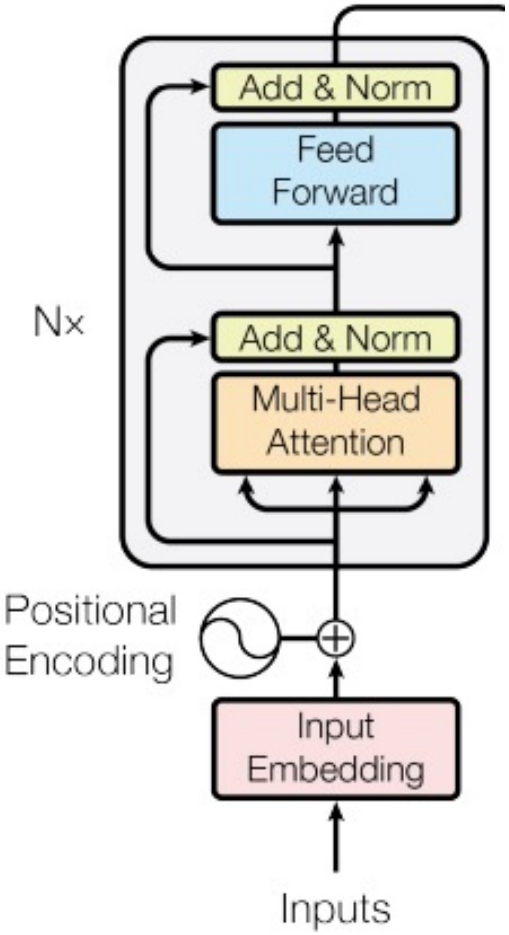


Figure 1: The Transformer - model architecture.

- 구성
  - Input - 독일어
  - Encoder
  - Decoder
  - Output - 영어
- RNN이나 CNN을 전혀 사용하지 않는 구조
  - 대신 Positional Encoding(위치 정보 저장)
- 인코더는 N번만큼 중첩 가능

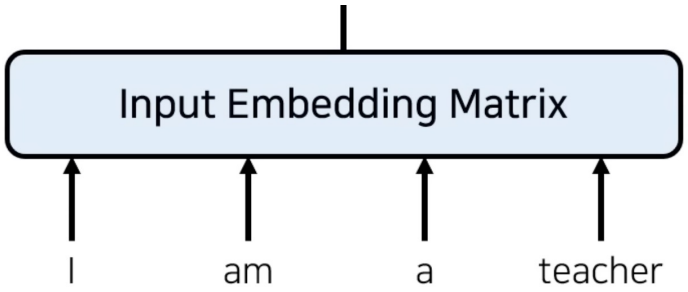
Encoder



Encoder Step

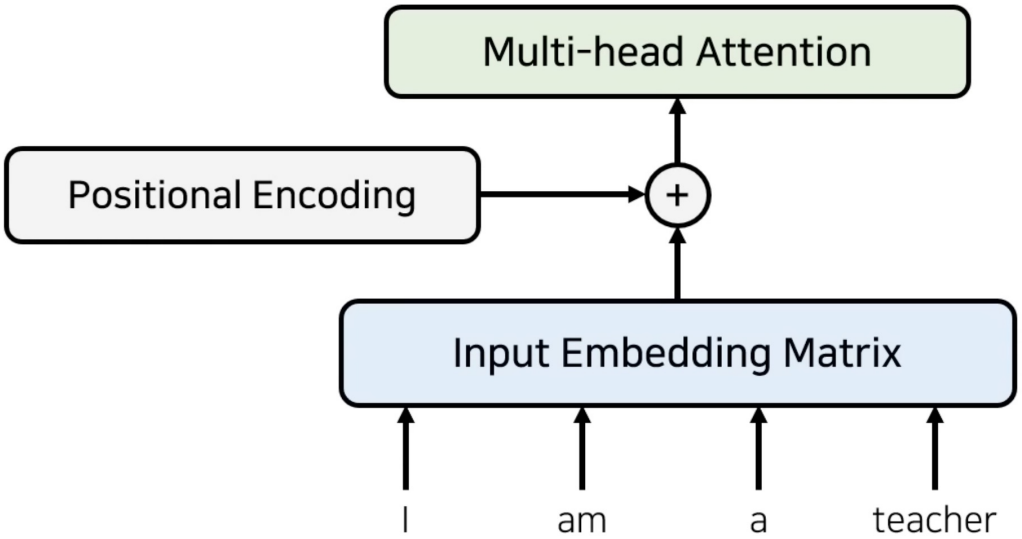
1. 단어의 embedding이 들어갔을 때, **positional encoding을 적용한 후 더해줌**
2. 더해준 형태의 Matrix를 **Multi-Head-Attention에 적용**
3. **Residual network (잔차학습) 의 형태로, 그 전의 값을 더해준 후 normalization을 적용**
4. **Feed Forward network 에 적용**한 후 **Add&Norm을 수행**
5. 위의 과정을 **N번 수행**해서 encoder에 대한 embedding을 만든다.

Input Embedding



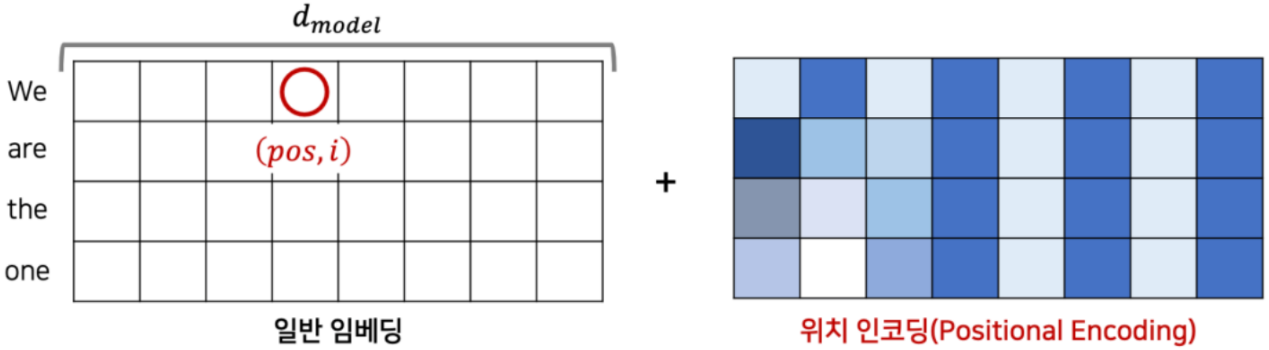
- 단어를 벡터로 변환하는 작업
- 벡터의 크기는 row : 단어의 수, column : embedding_dim (논문에서는 512)

Input Embedding



- Positional Encoding
  - Input Embedding Matrix와 같은 Demension을 가진 encoding 정보를 넣어줌
  - 단어의 상대적 위치정보를 주기함수를 활용해 저장
  - 학습데이터 보다 더 긴 문장이 들어와도 에러 없이 인코딩 값을 줄 수 있음

Input Embedding



- Word embedding 벡터에 positional encoding 벡터를 더함
- 자기 자신 = 0, 단어사이의 거리가 멀수록 값이 커짐

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

Self-Attention

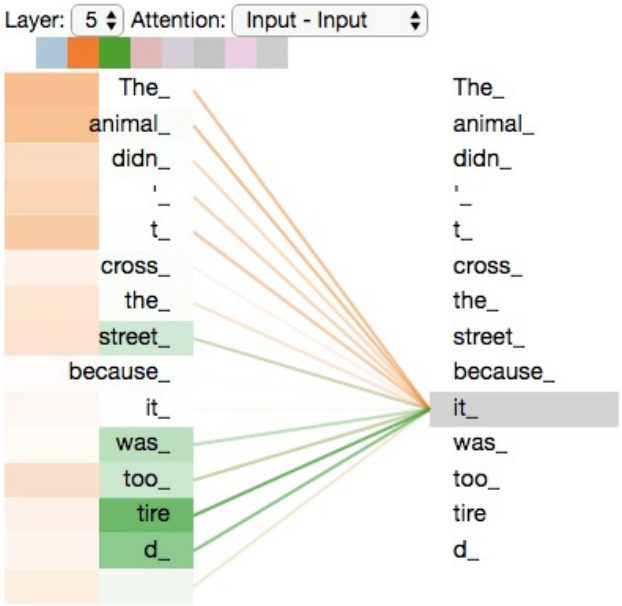
The animal didn’t cross the street because **it** was too tired.

위 문장에서 it이 의미하는 것은?

Self-Attention

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

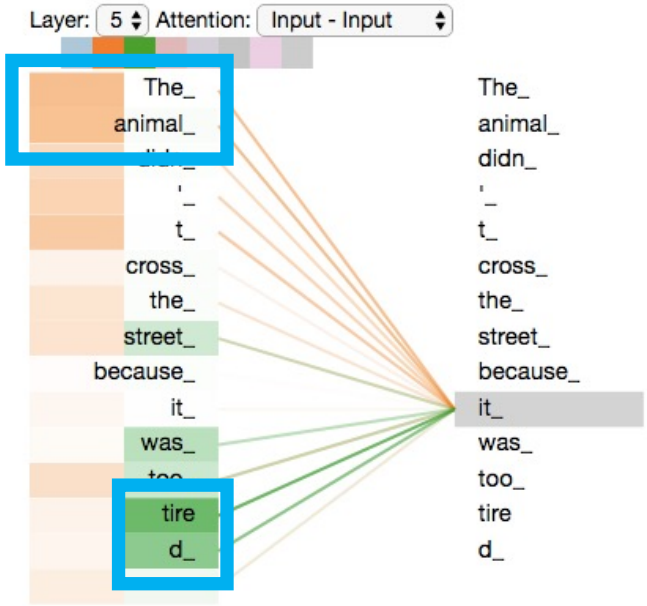
It이 의미하는 것이 animal임을 기계가 판단할 수 있도록 함



Self-Attention

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

It이 의미하는 것이 animal임을 기계가 판단할 수 있도록 함

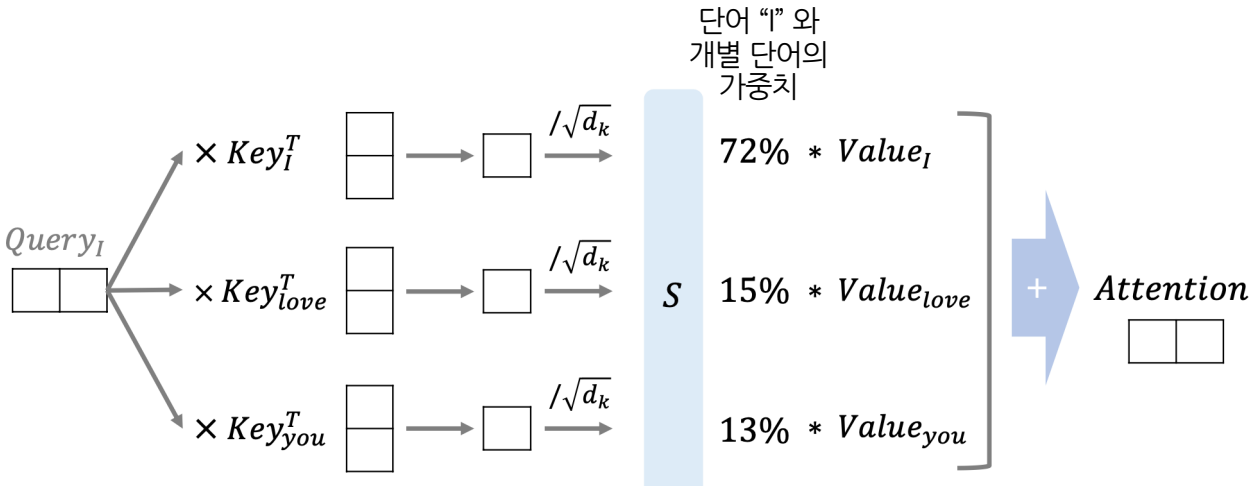


- 입력된 하나의 문장 안에서 문장을 이루는 단어들이 서로에 대해 갖는 중요도를 구함
- 주황색 레이어는 "The animal" 에 집중하고 있으며, 초록색 레이어는 "tired"에 집중하고 있음
- **문맥 상 동물이 tired 하다는 의미이기 때문에, 2개의 단어가 모두 중요하다는 것을 파악할 수 있음**
- 지시대명사가 무엇을 가르키는 지 아는 것에 유용
- 단어들 간 가중치를 시각화해 볼 수 있음

Self-Attention

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

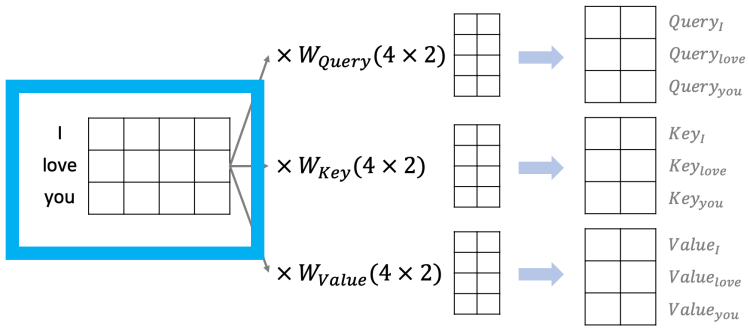
- Query(Q) : Word matrix (단어 행렬) 영향을 받는 단어 A
- Key(K) : Similarity matrix (유사도 행렬) 영향을 주는 단어 B를 나타내는 변수
- Value(V) Weight matrix (가중치 행렬), 그 영향에 대한 가중치
- 모든 쿼리와 key에 대해 dot product를 계산하고, softmax 에 들어가는 값을 normalization 하기 위해 $\sqrt{d_k}$ 로 나눠주고, weight을 적용하기 위해 value에 softmax 함수를 적용



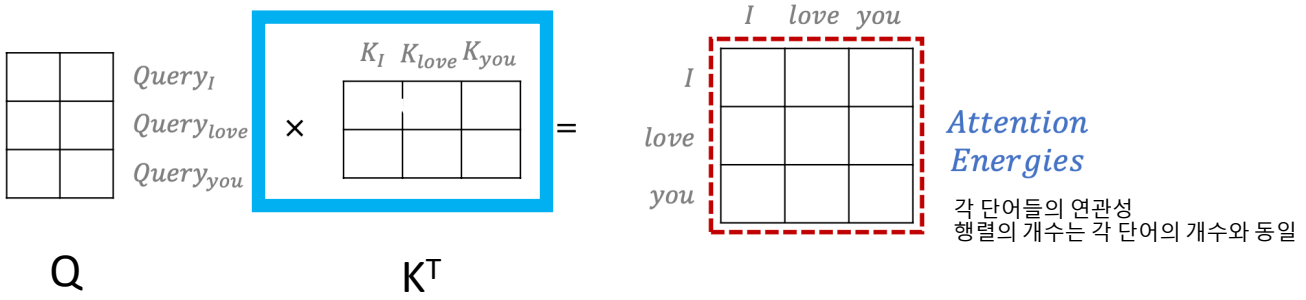
I love you 문장 예시

Softmax : 분류될 클래스가 n개 일 때 , n차원의 벡터를 입력받아, 각 클래스에 속할 확률을 추정

Self-Attention

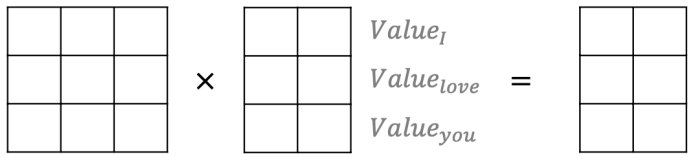


행렬 곱셈 연산을 이용해 한꺼번에 연산 가능

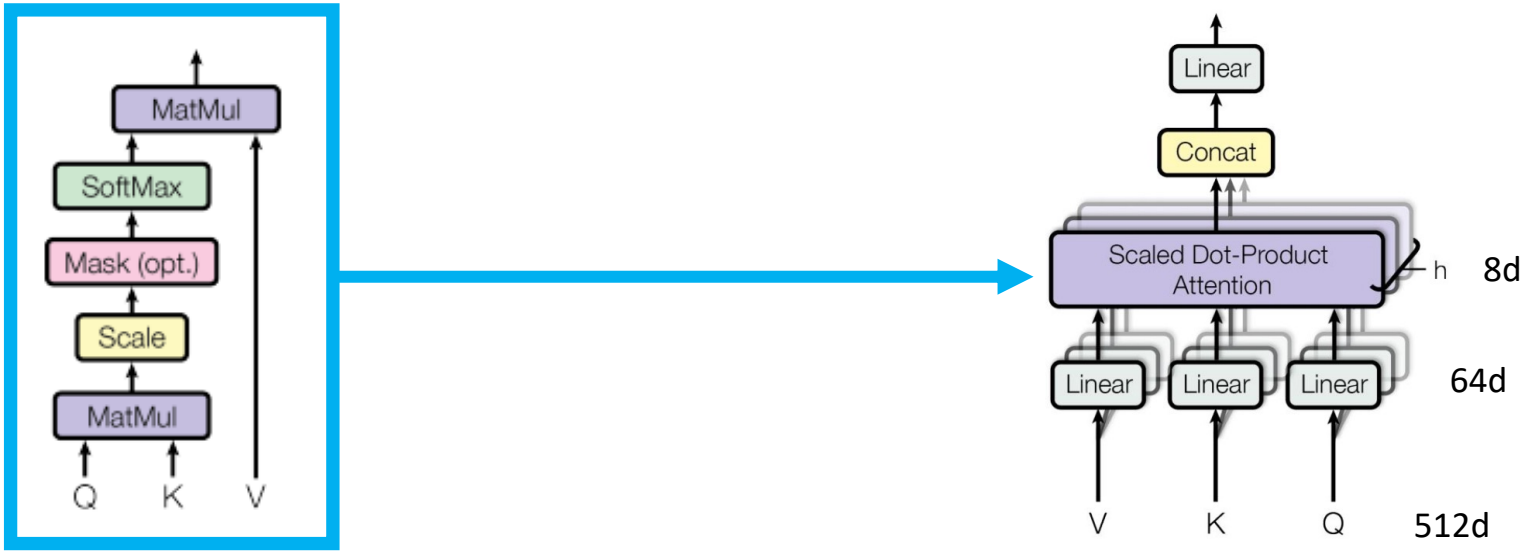


각 행마다 (단어마다) Key에 대한 확률 값을 구함

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V = Attention$$

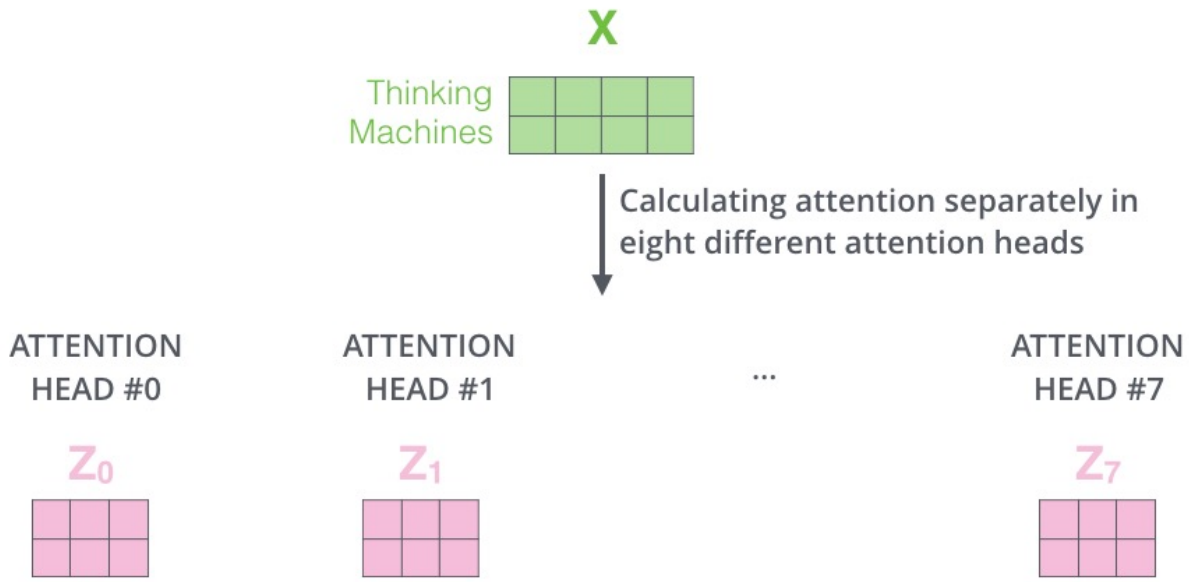


Multi Head Attention (Scaled dot-product attention)



- 어떤 입력 문장을 서로 다른 h개의 attention 컨셉을 만들어서 더욱 다양한 특징을 학습할 수 있도록 함
- 입력값은 h개로 구분됨 -서로 다른 V, K, Q 를 h개 생성
  - 논문에서는 8개를 사용
- Head 로부터 나온 attention 값을 Concat 수행하여 일 자로 붙인 뒤 linear 레이어를 거치게 됨 -> input과 output의 Dimension 이 일치

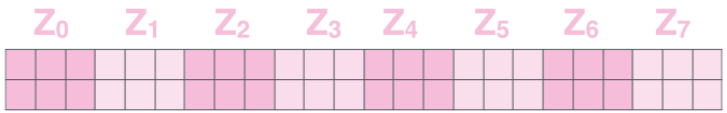
Multi Head Attention (Scaled dot-product attention)



- 하나의 단어를 8개의 head 를 생성하여 각각 다른 컨셉을 가지도록 세팅

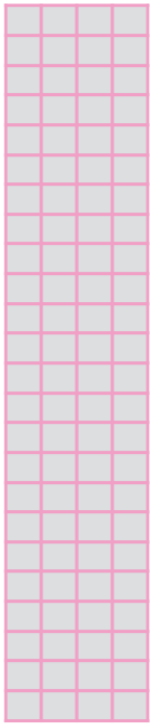
Multi Head Attention (Scaled dot-product attention)

1) Concatenate all the attention heads



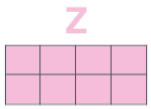
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



- Concat - 생성된 모든 attention head를 이어 붙임
- 가중치 (Weight) Matrix를 곱하여 초기 Dimension 과 맞춤

Self-attention 의 장점

- **Complexity per Layer**
  - n: 단어의 수, d: 차원 일 때
  - $n * d^2$ 보다 $n^2 * d$ 확률이 낮을 가능성이 더 높음

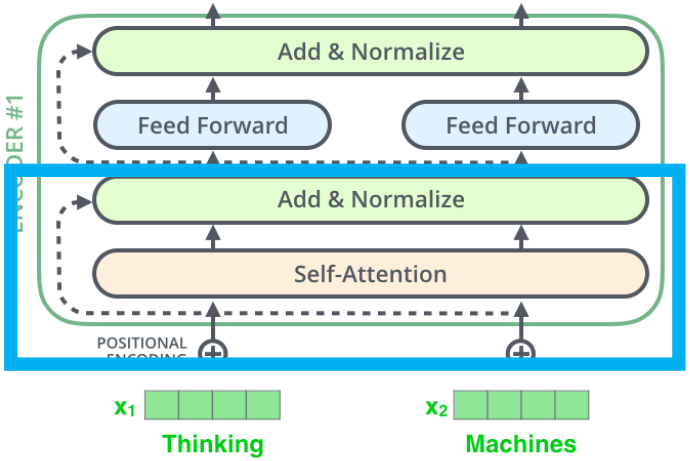
- **Sequential Operations**
  - RNN은 순차적으로 입력받아 처리하므로 총 n번의 RNN cell을 거치지만
  - Self-attention 은 벡터 연산을 병렬로 처리할 수 있음

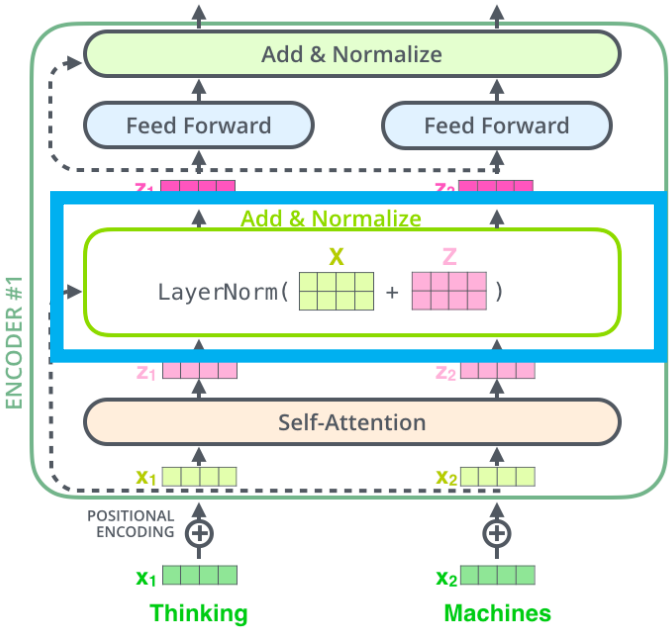
- **Maximum path length** (I love you -> 사랑해 에서 "I" 와 "사랑해" 사이의 길이 6)
  - 모든 단어들 사이의 Correlation information (상관관계)를 계산하여 더해주기 때문에 복잡도를 1로 볼 수 있으며, long range dependency (장거리 종속성)을 쉽게 학습 가능

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

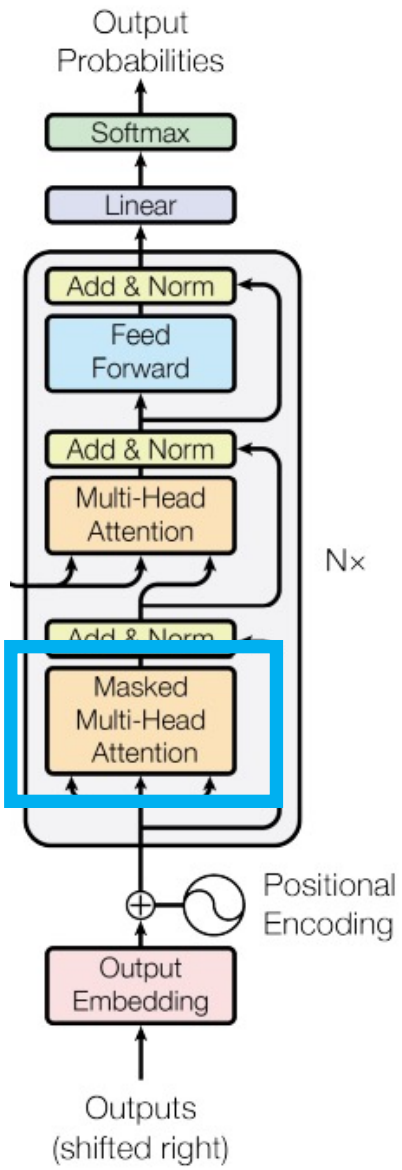
Encoder 특징

- Skip-connection 적용
  - Input vector를 그대로 한번 더 더해주어, 기울기 소실과 폭발 문제를 완화
- Layer Normalization 사용
  - 입력 분포를 정규화 시켜 학습 효율을 증진시킴





Decoder



Decoder Step

1. word embedding을 넣어준 후 **positional embedding**을 수행
2. (Masked) Multi-Head-Attention을 수행
3. Add&Norm을 수행
4. 인코더와 달리 **디코더의 입력값을 Query**로 사용하고, **인코더의 최종 출력값**을 **key와 value로 사용**하여 MHA 수행
5. Add&Norm을 수행한 후, FFN에 입력한 다음에 다시 Add&Norm을 수행
6. 위의 과정을 총 N번 반복
7. Linear를 통해 softmax 에 들어갈 로직 생성
8. softmax를 통해 각 단어들에 대한 확률 산출

Masked Multi Head Attention

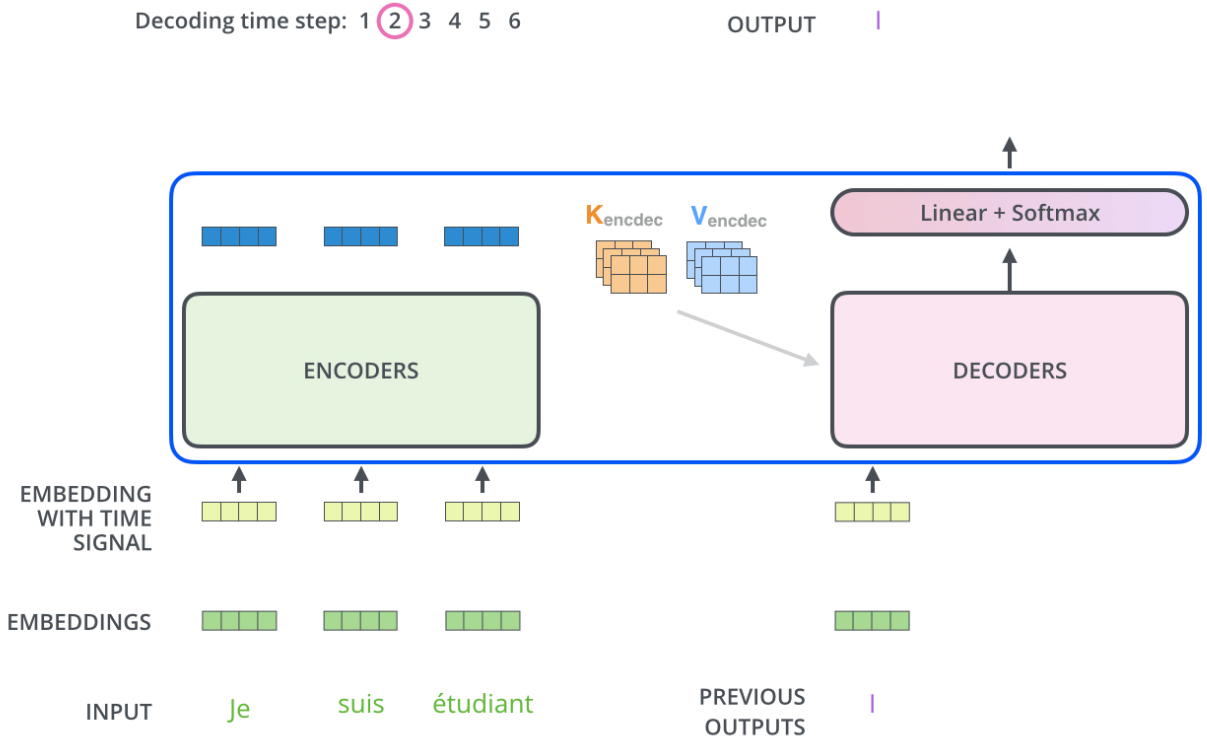
- 특정 토큰 이전의 토큰들만 고려하고, 그 다음 토큰은 mask 처리
- 타겟하는 문장에서 각 단어가 다음 단어가 무엇인지 알 수 없도록 처리하기 위함

현재 decoding 하고자 하는 단어 뒤에 있는 단어에 무한대 값을 주어,
Softmax 함수를 취할 때 0에 수렴하도록 함

I played soccer 에서, played 를 decoding 할 때는 soccer 를 알지 못하도록 함

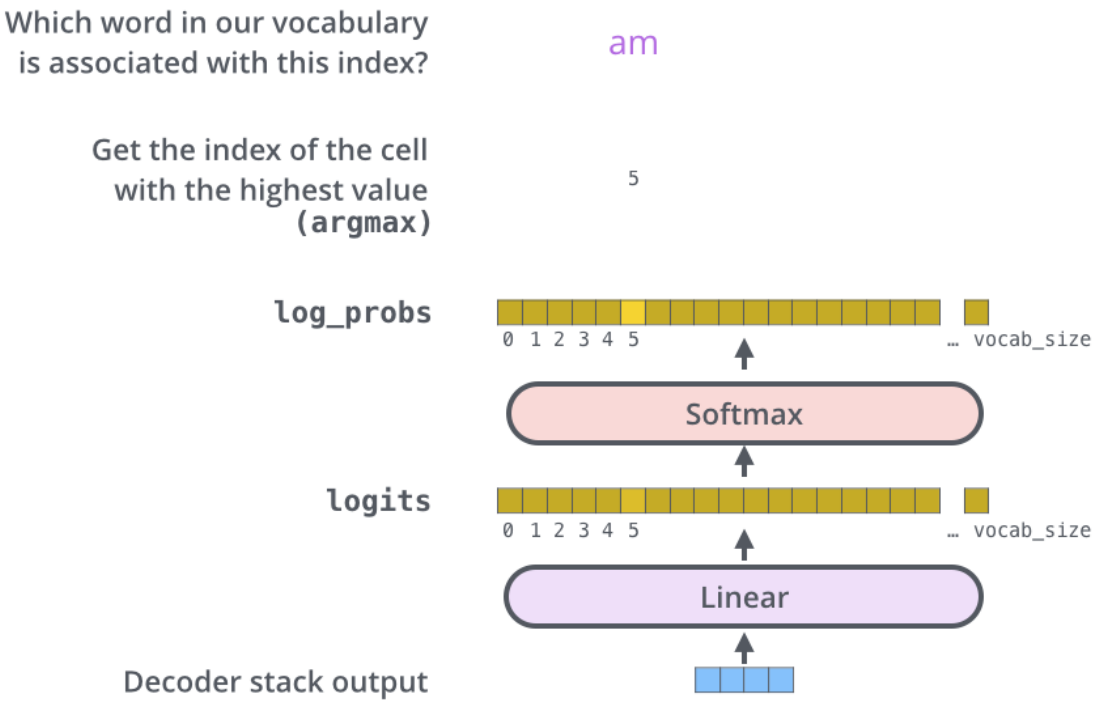
Encoder-Decoder Attention

- 디코더의 각 위치에서, 디코더의 현재 위치에서의 쿼리(Query)를 이용하여 인코더의 출력 특성(키(Key)와 값(Value))에 attention을 적용
- 디코더는 입력 시퀀스의 모든 위치에 대한 정보를 활용할 수 있게 되며, 특히 번역 작업에서는 입력 문장의 문맥을 보다 효과적으로 인코딩
- 번역이나 다른 시퀀스 생성 작업에서 전체 문맥을 고려한 결과를 얻을 수 있도록 합니다. 입력과 출력 간의 긴거리 의존성(Long-Distance Dependencies)을 처리하는 데 도움



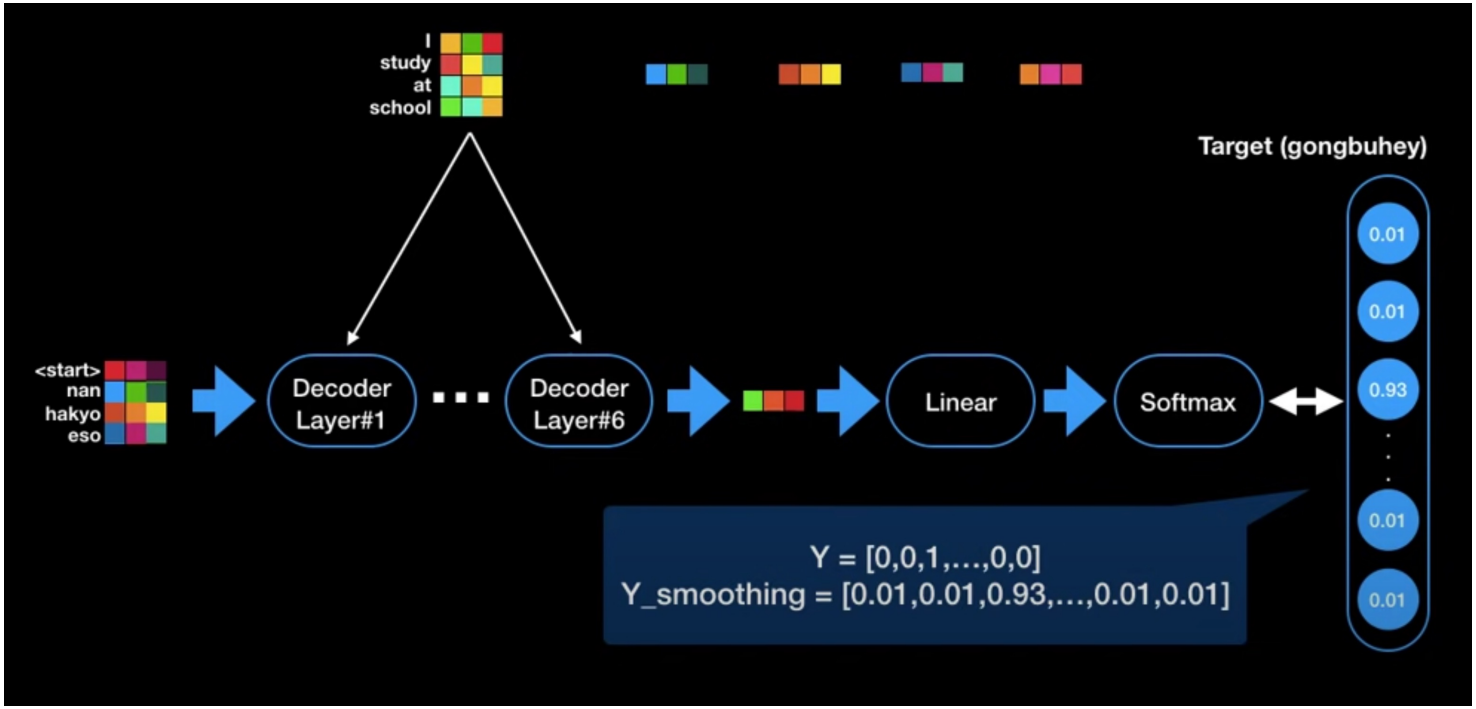
Output Layer

- 출력된 확률값으로 다음 토큰 단어를 예측하는 레이어
- 디코더에서 나온 output vector 는 neural network 를 거쳐 1차원 vector 로 나오게 됨
- Softmax 를 거쳐 최종 확률이 높은 단어가 output 으로 나옴



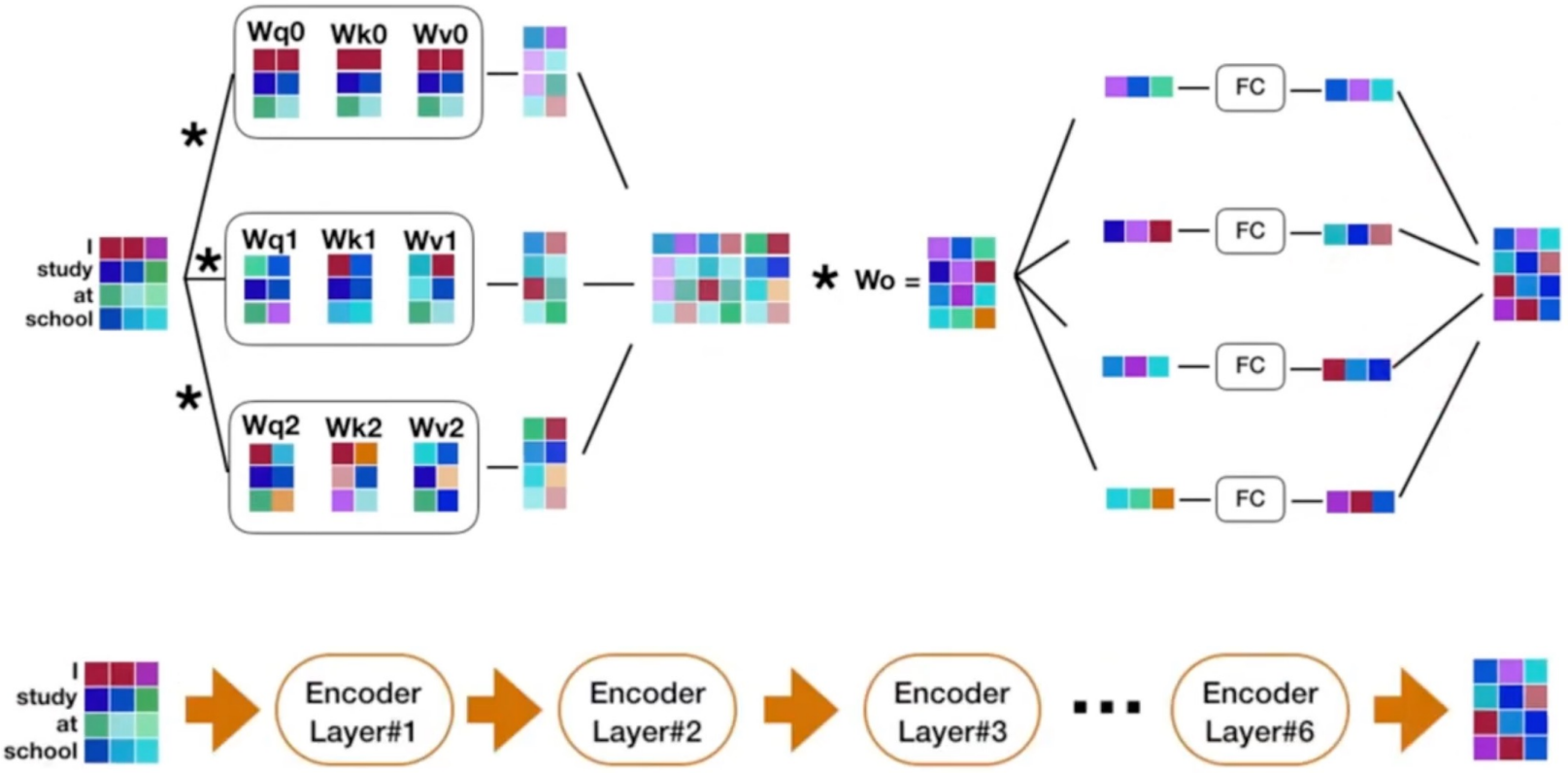
Label Smoothing

- 학습 시 너무 학습 데이터에 치중하지 못하도록 보완하는 기술
- 1에는 가깝지만 1이 아니고, 0에는 가깝지만 0이 아닌 값을 만듬
  - 정답에 가까우면 1, 오답에 가까우면 0
- “Thank you → 고마워, 감사합니다” 일 때 (label 이 noisy 할 때)
  두 학습 데이터 모두 잘못된 것이 아니기 때문에, label smoothing을 통해 둘 다 정답에 가까운 값으로 남아있게 됨



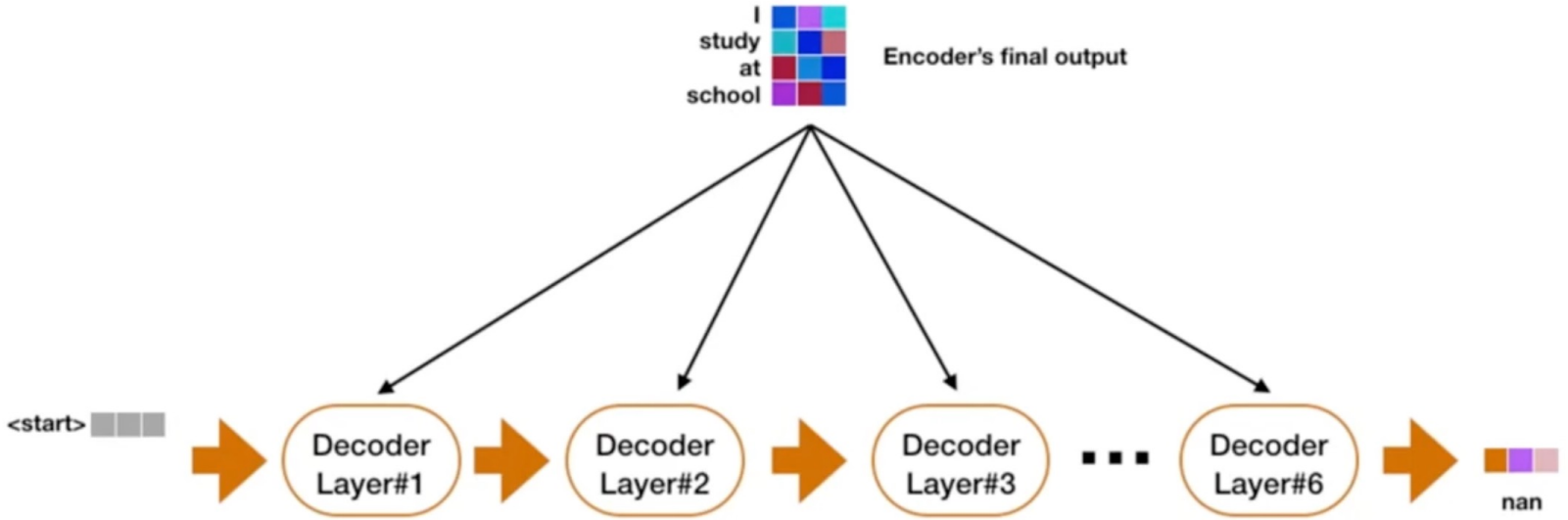
Summary

Encoding



Summary

Decoding

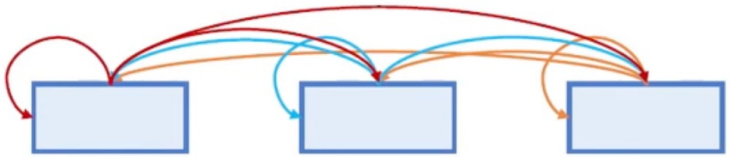


Summary

3번의 Attention

Encoder Self-Attention:

단어 간 중요도와 Dependency 파악

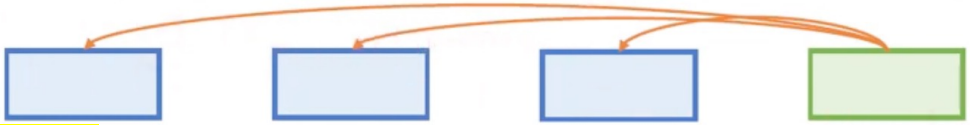


Masked Decoder Self-Attention:

현재 단어가 다음 단어에 영향받지 않도록 도움

Encoder-Decoder Attention:

입력과 출력 간의 긴거리 의존성을 처리하는 데 도움



감사합니다.