

Attention Is All You Need

Transformer

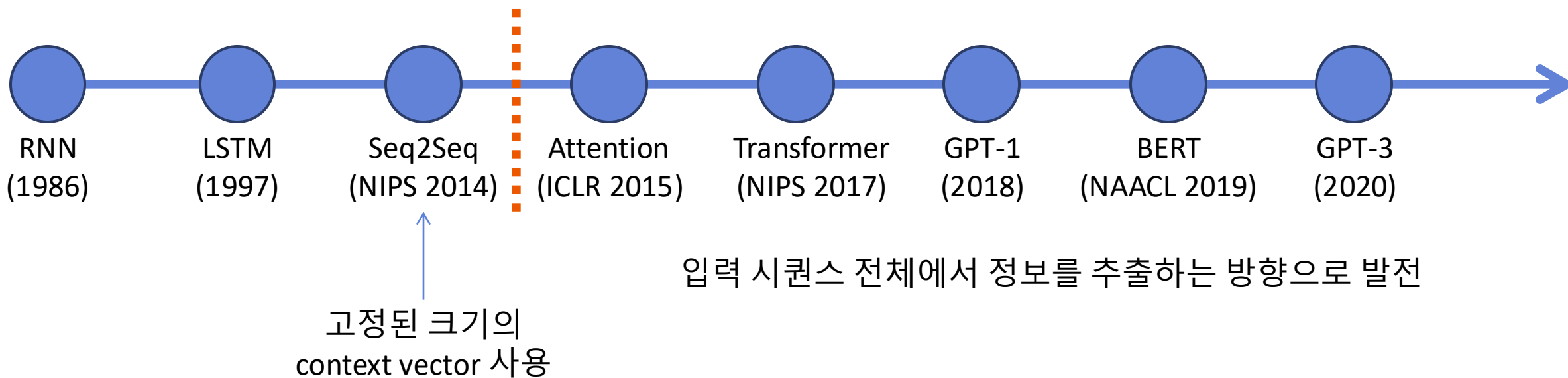
Transformer

- 어텐션 메커니즘 활용
- 최신 자연어 처리 모델의 아키텍처
- Google 번역기, 파파고 등에 활용 - GPT, BERT

딥러닝 기반 기계 번역

- 고성능 모델

- Transformer 아키텍처 기반 ex) GPT - 디코더 활용
BERT - 인코더 활용



딥러닝 기반 기계 번역

- RNN
 - RNN 첫 시작
- LSTM
 - RNN 등장 10년 이후 등장
 - 다양한 시퀀스 정보 모델링 가능 -> 주가예측, 주기함수 예측
- Seq2Seq
 - LSTM 활용한 딥러닝 기반 기술, 현대의 딥러닝 기술이 나오던 시점에 등장
 - LSTM을 활용하여 고정된 크기의 context vector를 사용하는 방식
 - 소스 문장을 전부 고정된 크기의 한 벡터에 압축 필요 - 성능적 한계

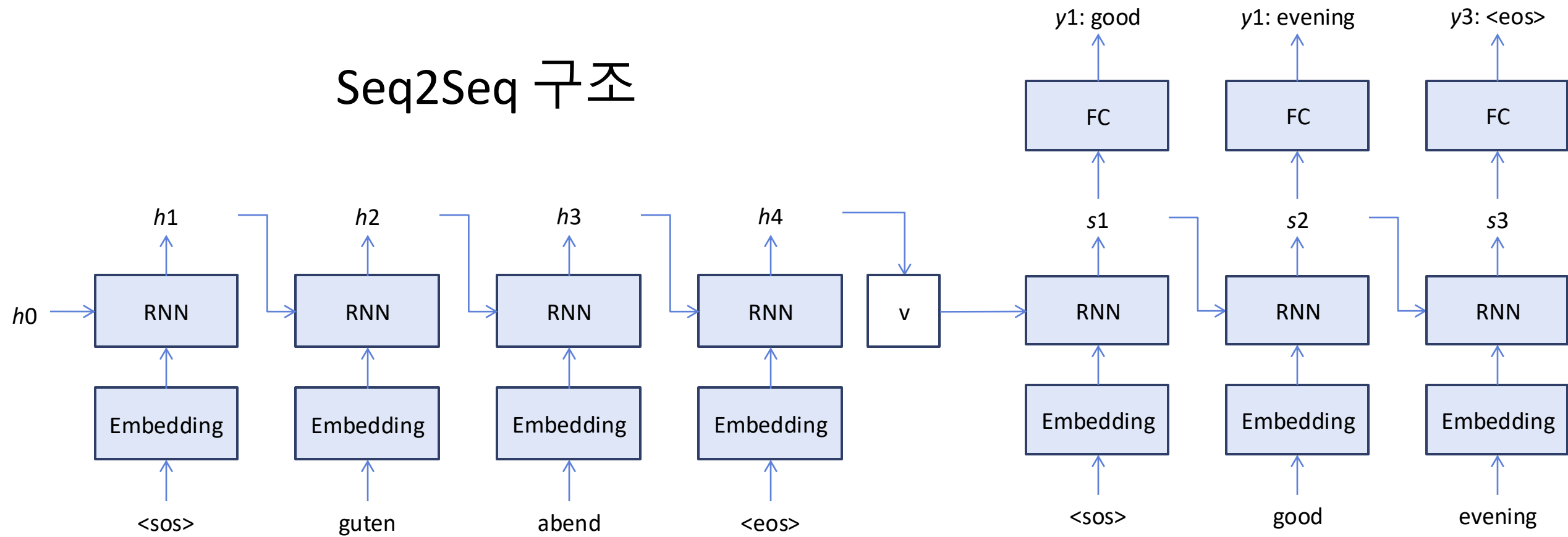
딥러닝 기반 기계 번역

- Attention
 - Seq2Seq 모델에서 어텐션 기법 적용
- Transformer
 - RNN 사용 없이 오직 어텐션 기법에 의존
 - 자연어 처리 기법으로 활용
 - 입력 시퀀스 전체에서 정보를 추출

Seq2Seq

- 한쪽 시퀀스로부터 다른 한 쪽의 시퀀스를 만들어 냄
- context vector에 소스 문장의 정보 압축 (병목 현상 발생)
- 인코더 과정: 입력 시퀀스가 하나의 고정 크기의 context vector 생성
- 디코더 과정: context vector로부터 출력 문장을 만들어 냄

Seq2Seq 구조



Seq2Seq 동작 원리

- 인코더

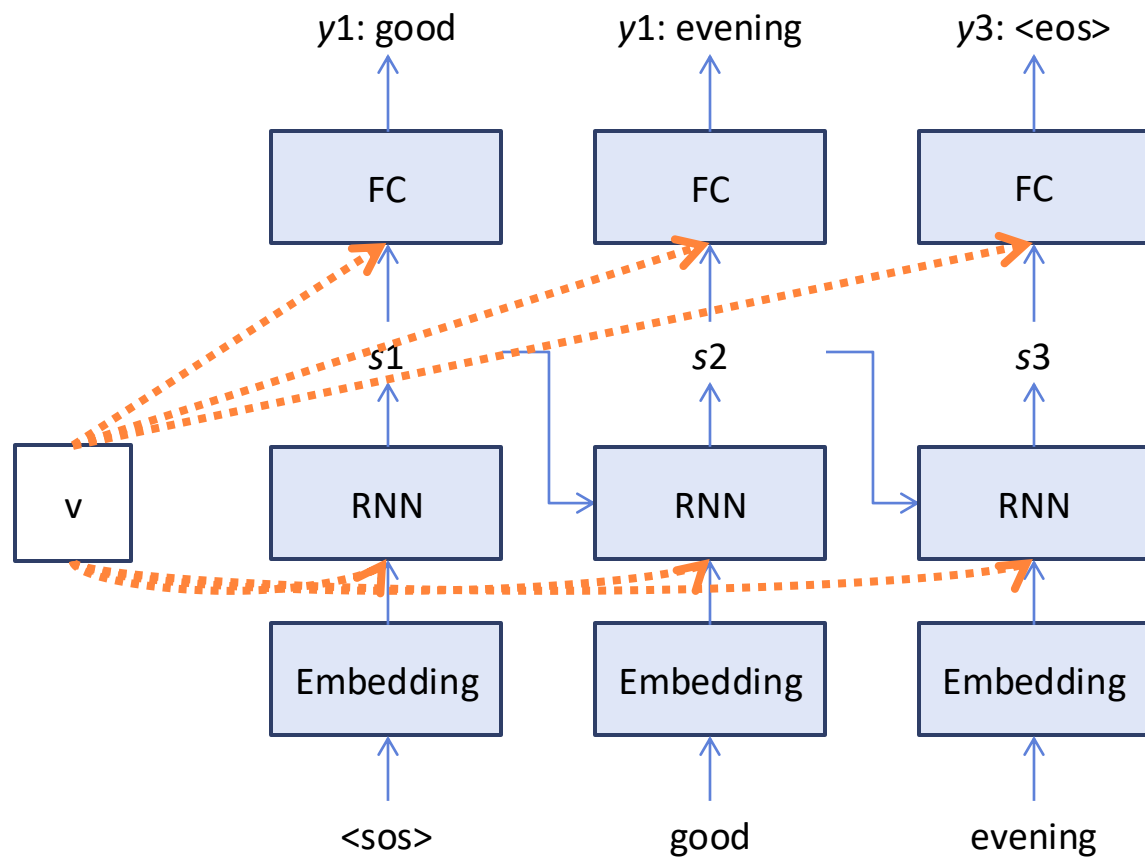
- hidden state: 이전까지 입력되었던 단어들에 대한 정보
- 단어가 입력될 때마다 이전 hidden state 값을 받아서 현재 hidden state 값 갱신
- 마지막 단어가 들어왔을 때의 hidden state 값 = 소스 문장 전체를 대표하는 context vector

- 디코더

- 출력 단어가 들어올 때마다 context vector로부터 매번 hidden state를 만들어서 출력에 내보냄
- <eos>가 나올때 까지 hidden state를 갱신하면서 출력

Seq2Seq 모델의 한계

- 소스 문장은 길거나 짧을 수 있음
 - 고정된 크기의 context vector가 병목 현상의 원인이 될 수 있음
- 새로운 아이디어
 - 고정된 크기의 context vector를 디코더의 RNN셀에 매번 참고
 - 성능은 개선 되지만 여전히 병목 현상은 유지



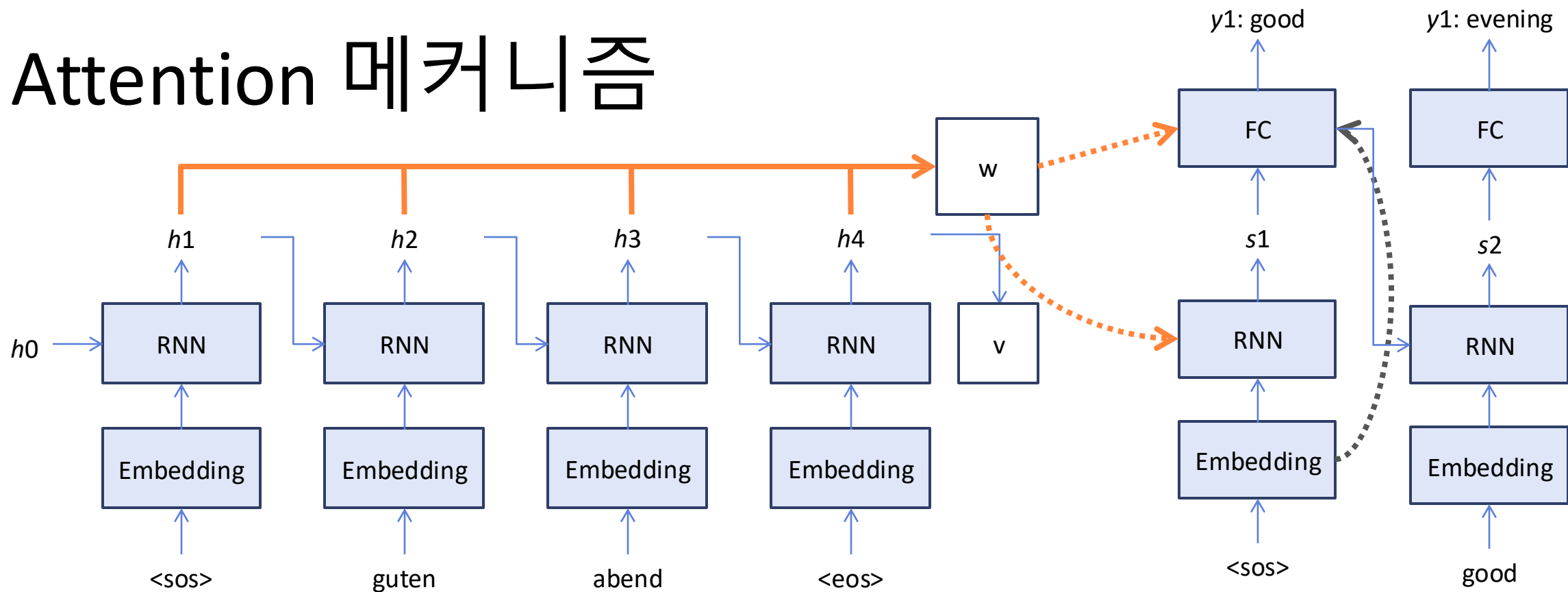
Seq2Seq의 병목 현상 해결방안

- 매번 소스 문장의 출력 전부를 입력으로: Attention 매커니즘
 - GPU는 병렬 처리 가능
 - 소스 문장을 구성하는 각 단어에 대한 출력 값을 전부 특정 행렬에 기록
 - 소스 문장에 대한 전반적인 내용들을 출력할 때마다 반영 가능

Attention 메커니즘

- 디코더 부분에서 매번 hidden state ($s_1, s_2 \dots$) 갱신
- s_2 :
이전 hidden state 값 s_1 과 소스문장의 hidden state ($h_1, h_2 \dots$) 들을 서로 묶어 행렬곱 수행 후 energy 값 계산
 - energy 값: 어떤 단어에 초점을 둘 필요가 있는지 수치화한 값
 - softmax 취해서 확률 값 구함
 - 각 hidden state에 가중치를 부여하여 참고
 - hidden state에 가중치 값을 곱한 후 정규화
 - weight sum을 매번 출력 단어를 만들기 위해 반영

Attention 메커니즘



- 출력되는 모든 hidden state를 별도의 행렬에 기록
- 각 단어들을 똑같이 참고하는 것이 아님, 중요도에 따라 가중치 부여


디코더의 계산 과정

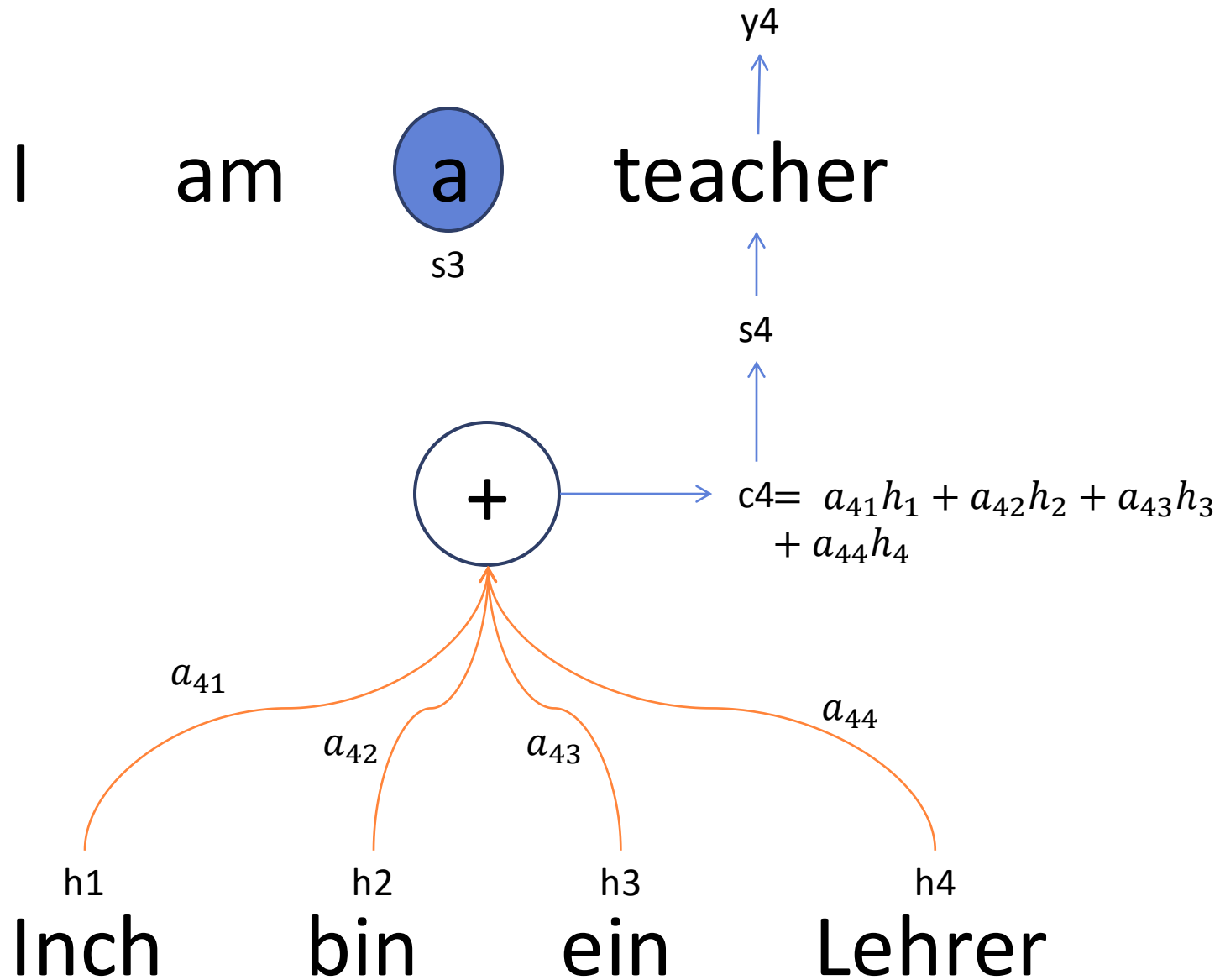
Energy: $e_{ij} = a(s_{i-1}, h_j)$

i : 현재 인코더가 처리중인 인덱스
 j : 인코더 파트의 출력 인덱스

Weight: $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$

Weight sum

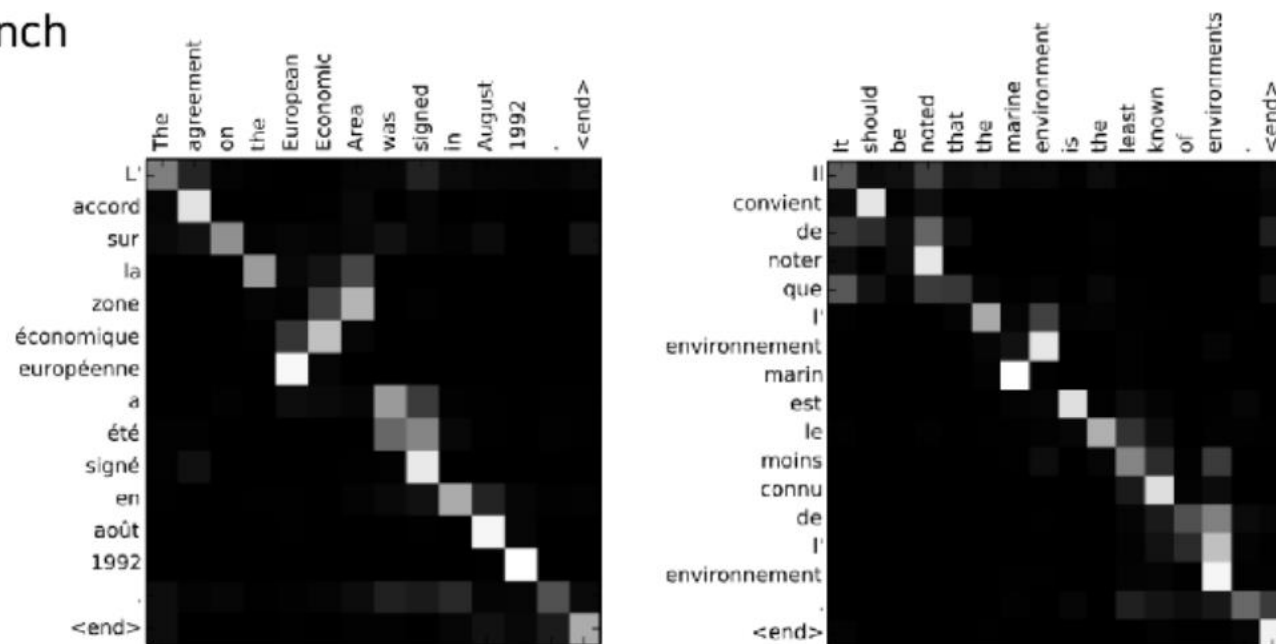

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$



Attention 장점

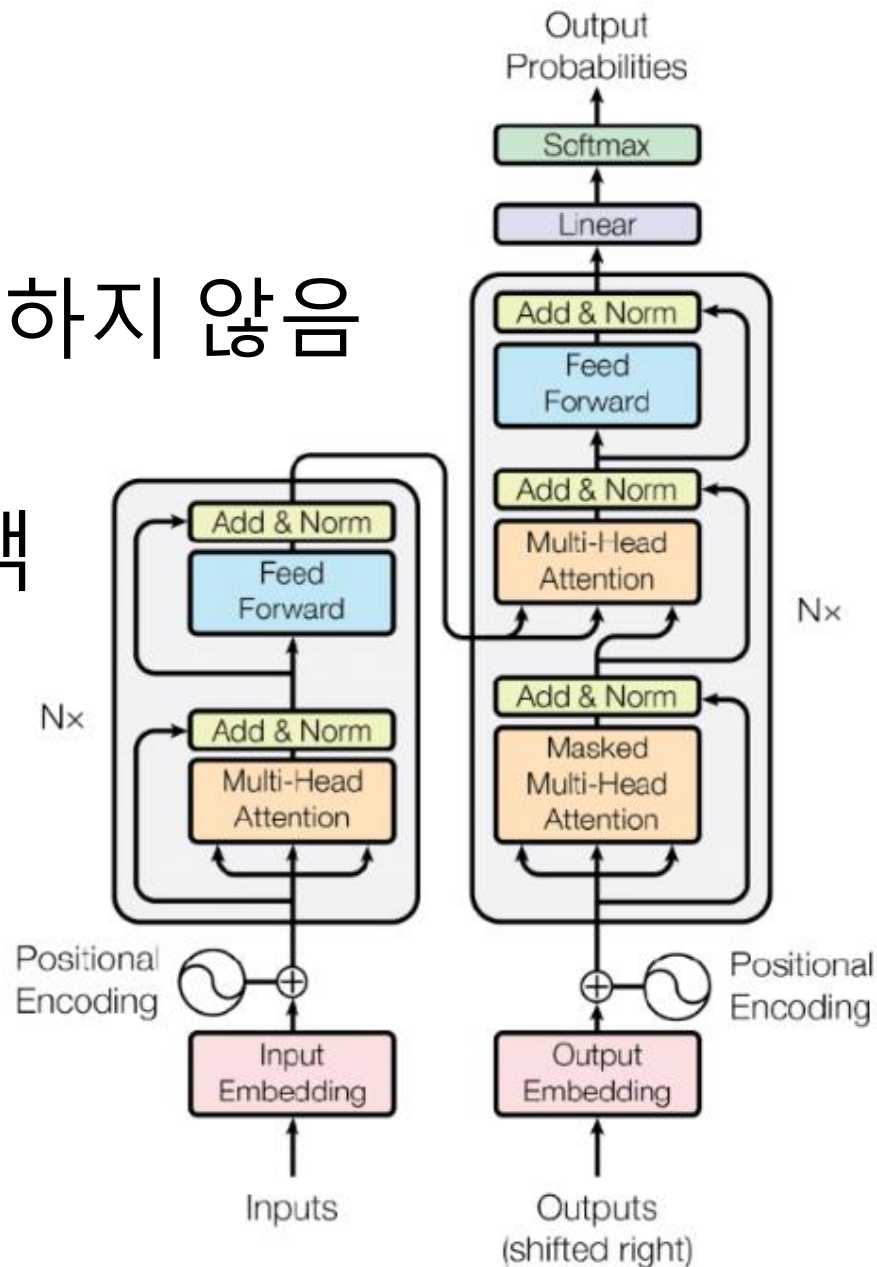
- 시각화 가능
입력에서 어떤 정보를 참고했는지...

English → French



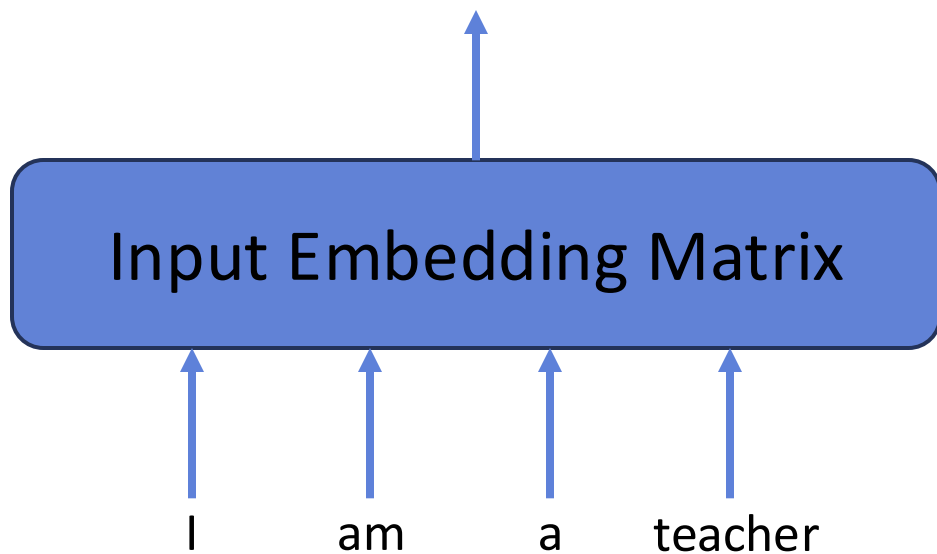
Transformer 구조

- 트랜스포머는 RNN이나 CNN을 필요로 하지 않음
 - Positional Encoding 사용
- BERT와 같은 향상된 네트워크에서 채택
- 인코더와 디코더로 구성
 - Attention 과정을 여러 레이어에서 반복
 - N번 만큼 중첩되어 사용



Transformer 동작 원리

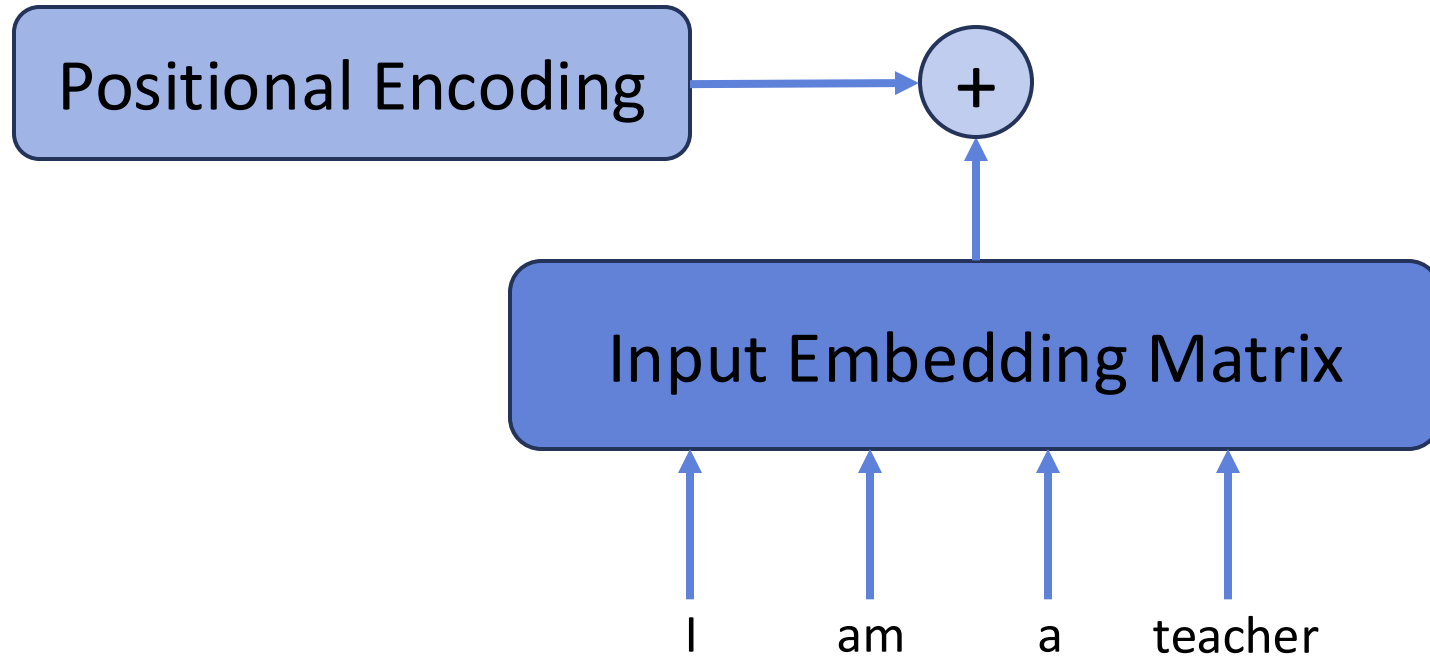
- 입력 값 임베딩



input	Embed dim		
I		...	
am		...	
a		...	
teacher		...	

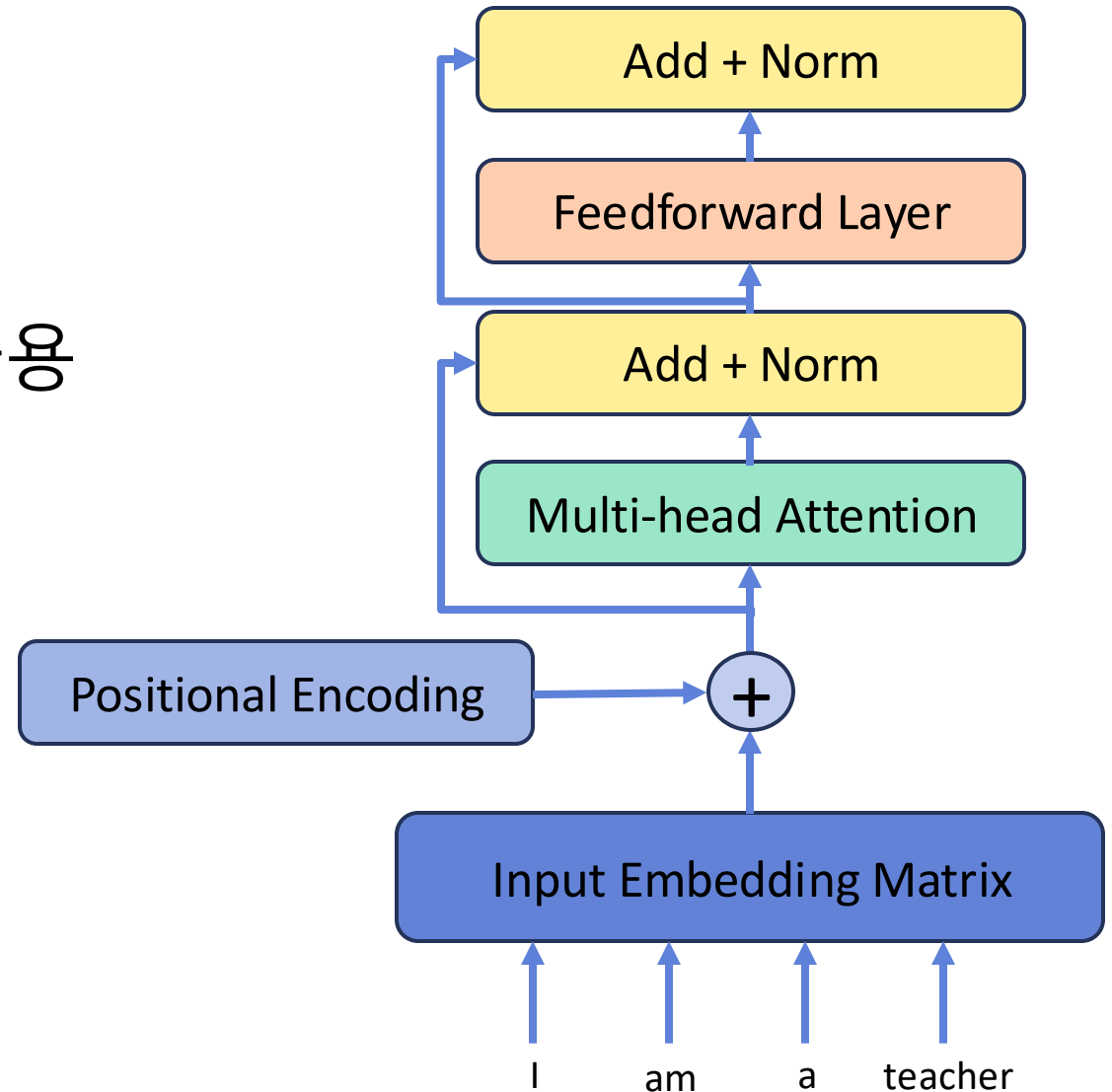
Transformer 동작 원리

- 입력 값 임베딩
 - Positional Encoding 사용 -> RNN을 사용 x



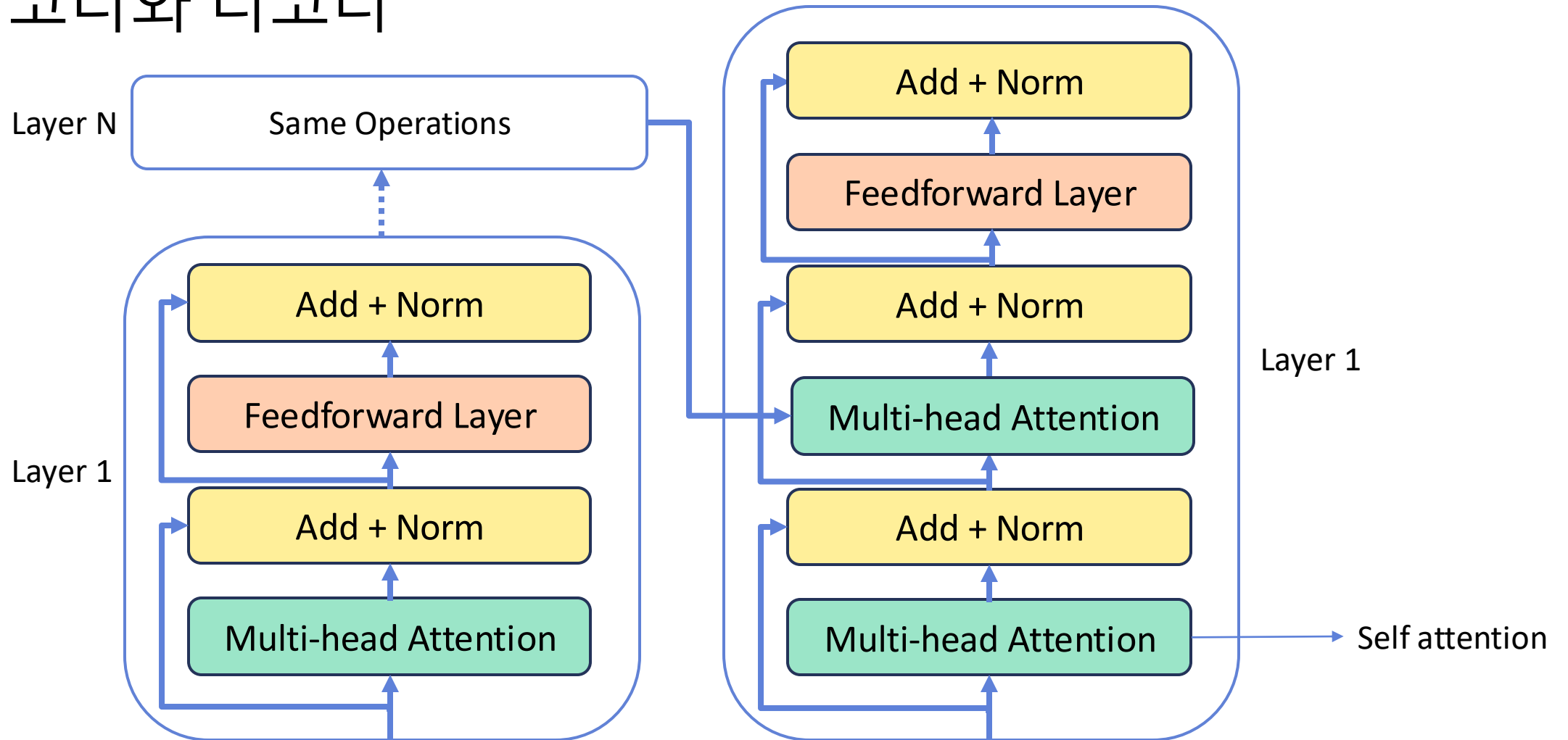
Transformer 동작 원리

- 인코더
 - 임베딩 이후 어텐션 진행
 - 성능 향상을 위해 잔여 학습 사용
 - 어텐션과 정규화 과정 반복
 - 각 레이어는 서로 다른 파라미터



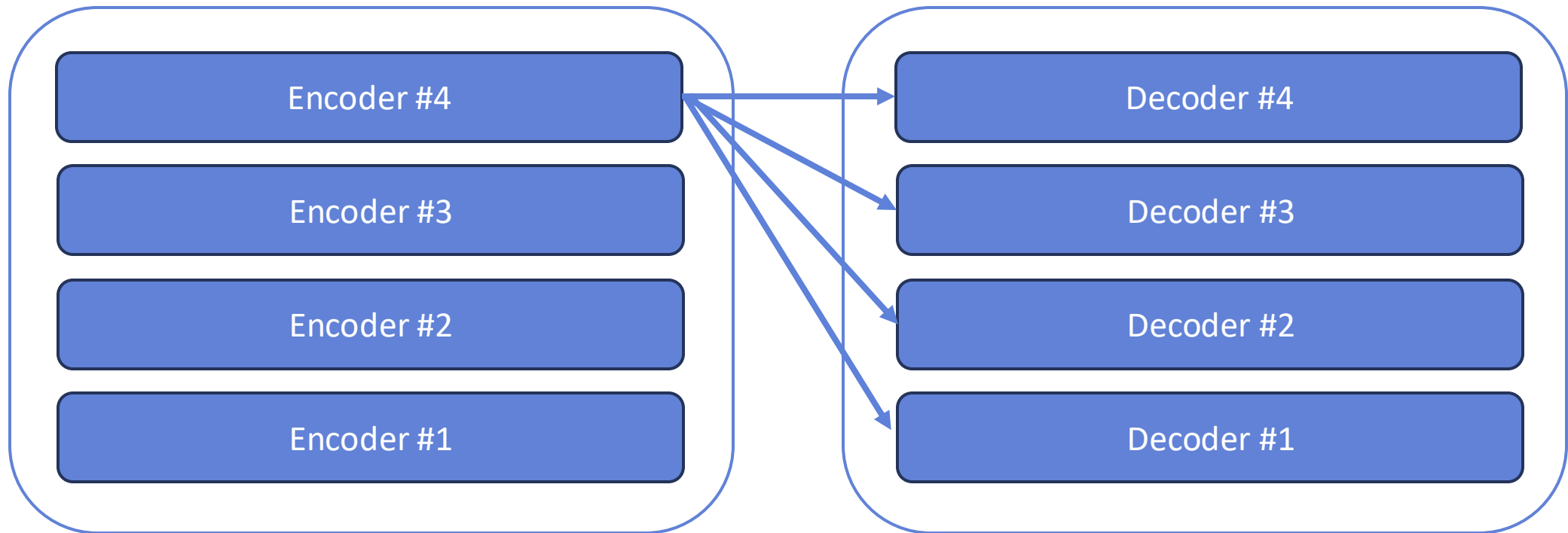
Transformer 동작 원리

- 인코더와 디코더



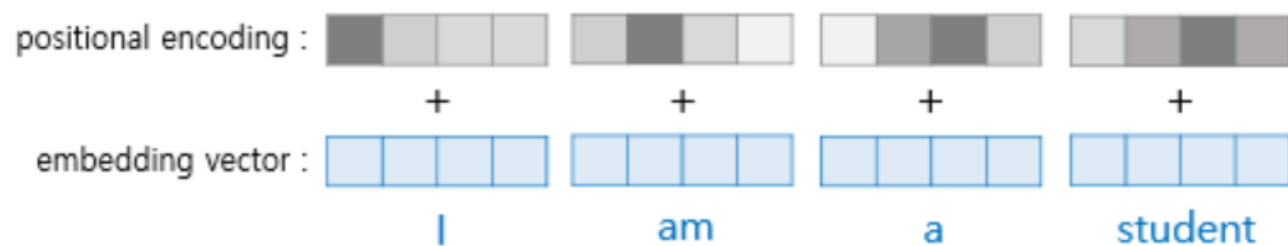
Transformer 동작 원리

- 인코더와 디코더
 - 마지막 인코더 레이어의 출력이 모든 디코더 레이어에 입력
 - <eos> 가 나올 때까지 디코더 사용



Transformer 동작 원리

- Positional encoding



짝수 $PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{model}})$

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

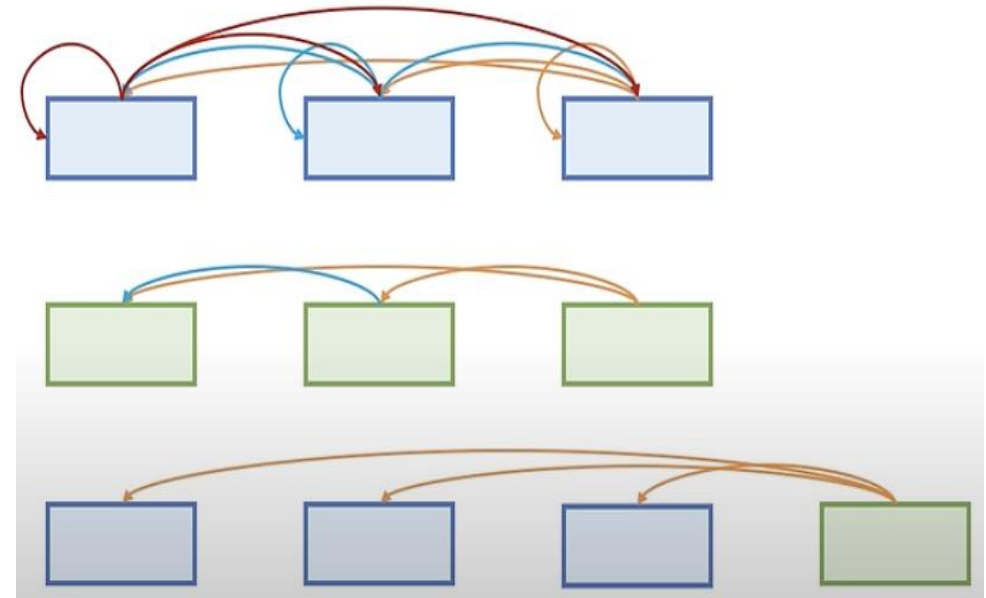
pos : 입력 문장의 임베딩 벡터의 위치

i : 임베딩 차원의 인덱스

d_{model} : 트랜스포머 모든 층의 출력 차원 (논문에서는 512)

Transformer 동작 원리

- Attention
 - Encoder Self-Attention
 - 인코더에서 이루어짐
 - query, key, value가 같음 (벡터의 출처가 같음)
 - Masked Decoder Self-Attention
 - 디코더에서 이루어짐
 - query, key, value가 같음 (벡터의 출처가 같음)
 - Encoder-Decoder Attention
 - 디코더에서 이루어짐
 - query: 디코더 벡터
 - key, value: 인코더 벡터
- Multi-head Attention: 어텐션을 병렬적으로 수행



Transformer 동작 원리

- Attention

- Q = Query: 모든 시점 디코더 셀의 은닉 상태
 - K = Key: 모든 시점 인코더 셀의 은닉 상태
 - V = Value: 모든 시점 인코더 셀의 은닉 상태
- 각 단어의 임베딩을 이용해 생성
행렬 곱셈 연산

문장 내의 단어들끼리 유사도 평가

