23-1 비정형데이터분석

Seq2Seq Learning ~ BERT

Youtube 요약 영상 과제 2

고려대학교 산업경영공학과

팀 : 지란지교

팀원 : 박새란 이지윤

Content

01

Seq2Seq Learning

Seq2Seq Learning

✓ 통계적 번역의 한계와 NMT의 등장

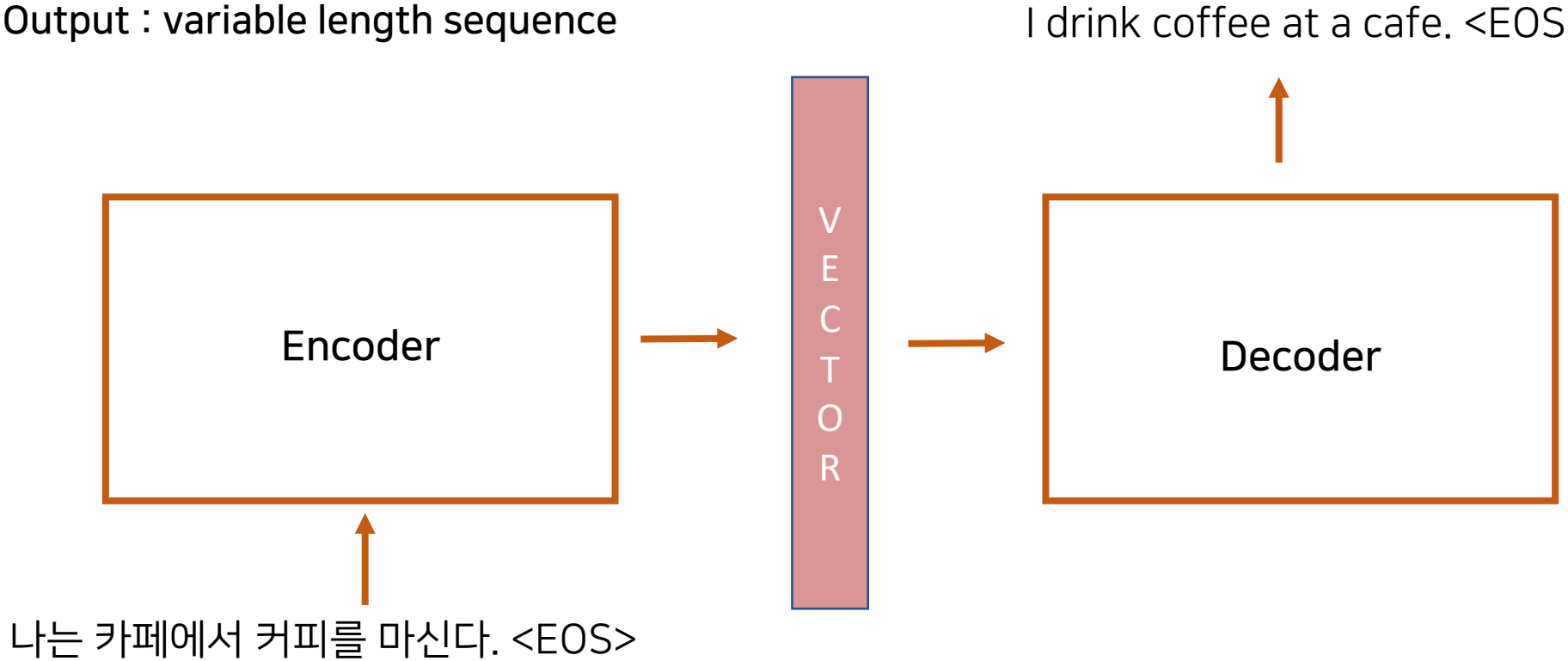
- 통계적인 방법
  - 입력 문장을 단어 혹은 구문 단위로 자른 후(phrase-based)
  - 통계적으로 본래 의미와 가장 가까운 번역 결과를 제시
- 문제점1 : 동음이의어 구분의 어려움
  - 배(pear), 배(ship)
- 문제점2 : 어순이 다름
  - 한국어 : 주어 -> 목적어 -> 서술어
  - 영어 : 주어 -> 서술어 -> 목적어
- 문제점3 : 전체 문맥 반영의 어려움.

문장 단위의 번역이 필요!

Seq2Seq Learning

✓ Encoder-Decoder 구조

- Input : variable length sequence
- Fixed size vector
- Output : variable length sequence



Seq2Seq Learning

✓ Encoder-Decoder 구조

- 두개의 RNN 모델 (encoder, decoder)
- 동일한 색상은 동일한 weight를 가지고 있음.
- Context vector
  - 입력된 모든 토큰들에 대한 정보를 담고 있는 벡터
  - Encoder의 last hidden state

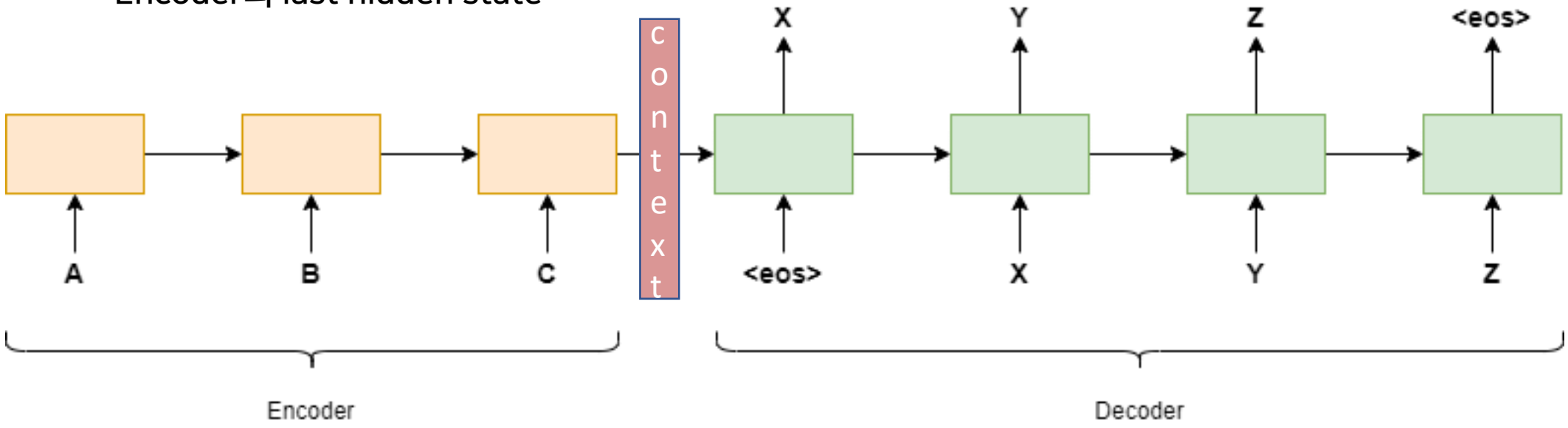


Figure 1: This is the architecture of a Seq2seq model: the first RNN is called encoder and the second one is decoder. In this case, the model receives an input sentence "ABC" and produces "XYZ" as the output sentence (the input and output may have different lengths).

Seq2Seq Learning

✓ Encoder-Decoder 구조

- 두개의 RNN 모델 (encoder, decoder)
- 동일한 색상은 동일한 weight를 가지고 있음.
- Context vector (다양하게 사용 가능함.)
  - 입력된 모든 토큰들에 대한 정보를 담고 있는 벡터
  - Encoder의 last hidden state

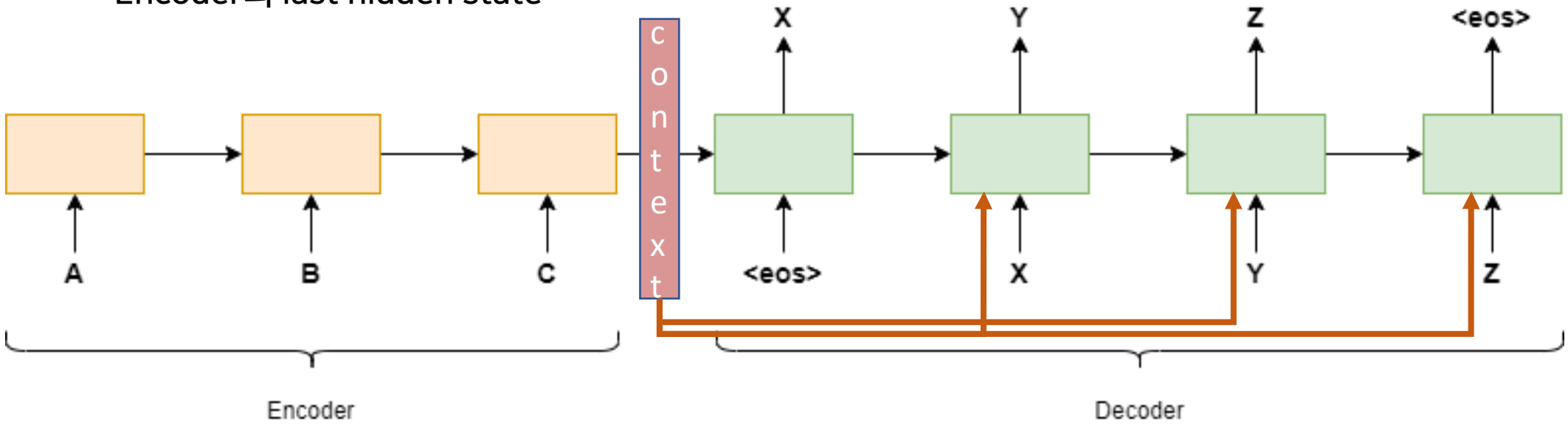


Figure 1: This is the architecture of a Seq2seq model: the first RNN is called encoder and the second one is decoder. In this case, the model receives an input sentence "ABC" and produces "XYZ" as the output sentence (the input and output may have different lengths).

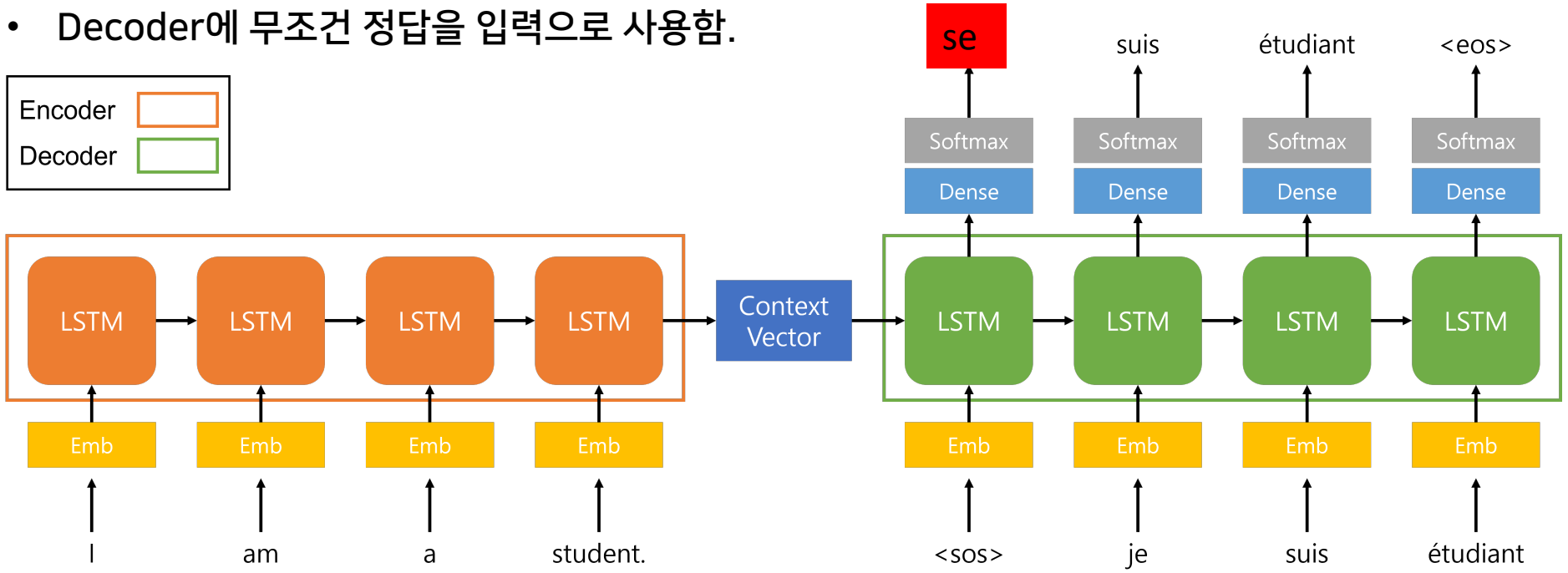
Seq2Seq Learning

✓ Training

- RNNLM
  - Context vector와 y(<t)이 주어졌을 때 y_t의 probability가 최대화 되는 방향으로 학습됨.

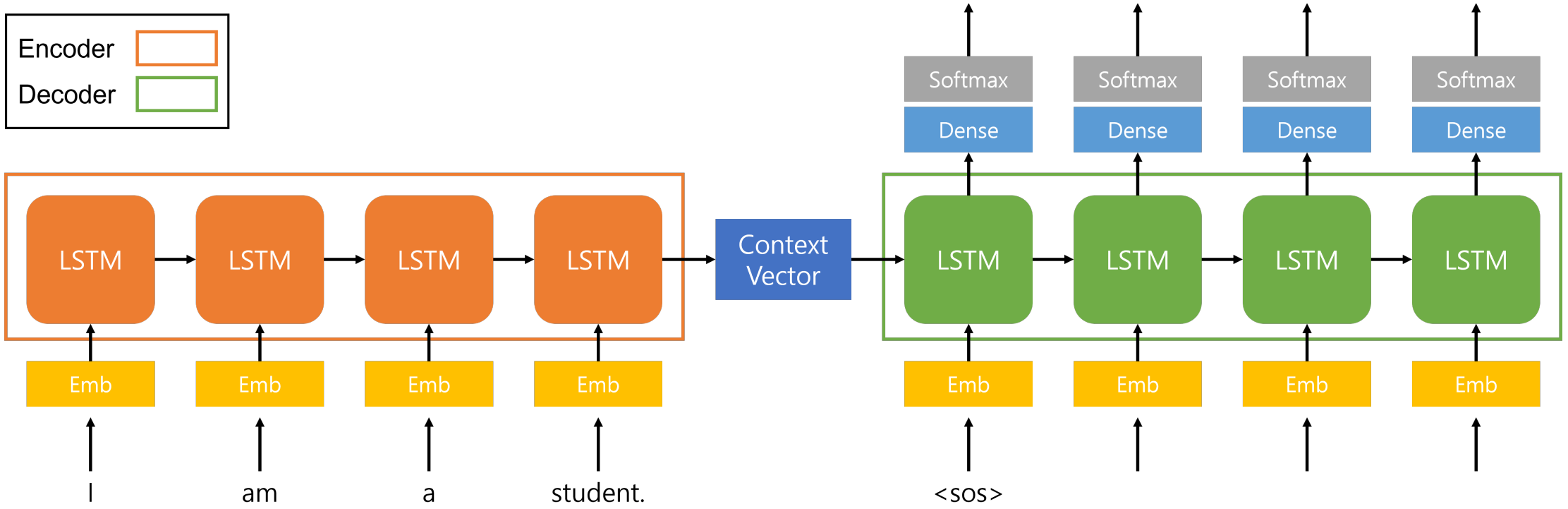
$$\max_{\boldsymbol{\theta}} \frac{1}{N} \sum_{n=1}^{N} \log p_{\boldsymbol{\theta}}(\mathbf{y}_n \mid \mathbf{x}_n)$$

- Teacher Forcing
  - Decoder에 무조건 정답을 입력으로 사용함.



Seq2Seq Learning

✓ Inference



https://heekangpark.github.io/nlp/attention#kramdown_%EC%97%AC%EB%8B%B4--seq2seq-%EB%AA%A8%EB%8D%B8%EC%9D%98-%ED%95%99%EC%8A%B5-%EB%B0%A9%EB%B2%95---%EA%B5%90%EC%82%AC-%EA%B0%95%EC%9A%94teacher-forcing

Seq2Seq Learning

✓ Sequence to Sequence Learning with Neural Networks(NIPS 10 September 2014)

- Encoder-Decoder로 어떤 모델을 사용해야 할까요?
  - RNN은 long-term dependency에 한계가 있음.
  - LSTM으로 바꾸어 사용하여 성능 향상이 있었음.
  - LSTM이 깊을수록 성능 향상이 있었음(해당 논문에서 4 layers로 사용)
- Input의 토큰 순서를 거꾸로 하였을 때 성능 향상
  - [나는, 카페, 에서, 커피를, 마신다.] ➡️ [마신다, 커피를, 에서, 카페, 나는]

02

Attention in Seq2Seq Learning

Attention in Seq2Seq Learning

✓ Seq2Seq model의 한계

- 입력 문장을 고정된 크기의 context vector에 압축해야 함.
  - 입력 문장이 굉장히 길어지면 정보 손실이 발생하게 됨.
- RNN계열의 모델 사용
  - Gradient Vanishing / Exploding

Attention in Seq2Seq Learning

✓ Attention mechanism의 가정

디코더에서 X를 출력할 때의 hidden state는

X와 가장 유사한 단어가 인코더에 입력된 직후의 hidden state와 유사할 것이다.

Ex) Z와 가장 유사한 단어 B : 빨간 박스들의 hidden state가 유사

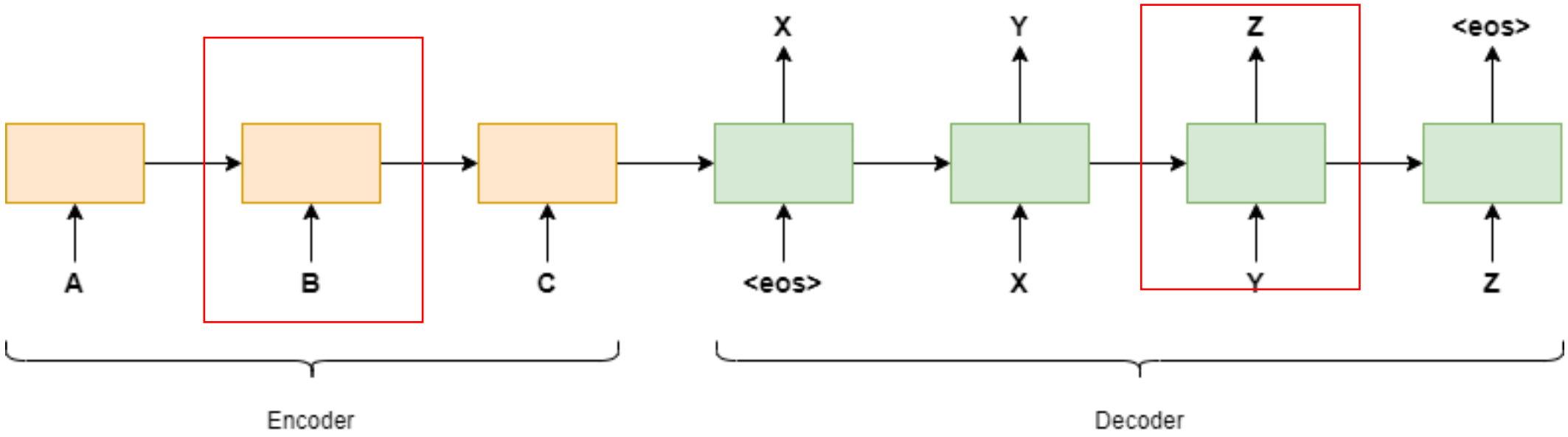


Figure 1: This is the architecture of a Seq2seq model: the first RNN is called encoder and the second one is decoder. In this case, the model receives an input sentence "ABC" and produces "XYZ" as the output sentence (the input and output may have different lengths).

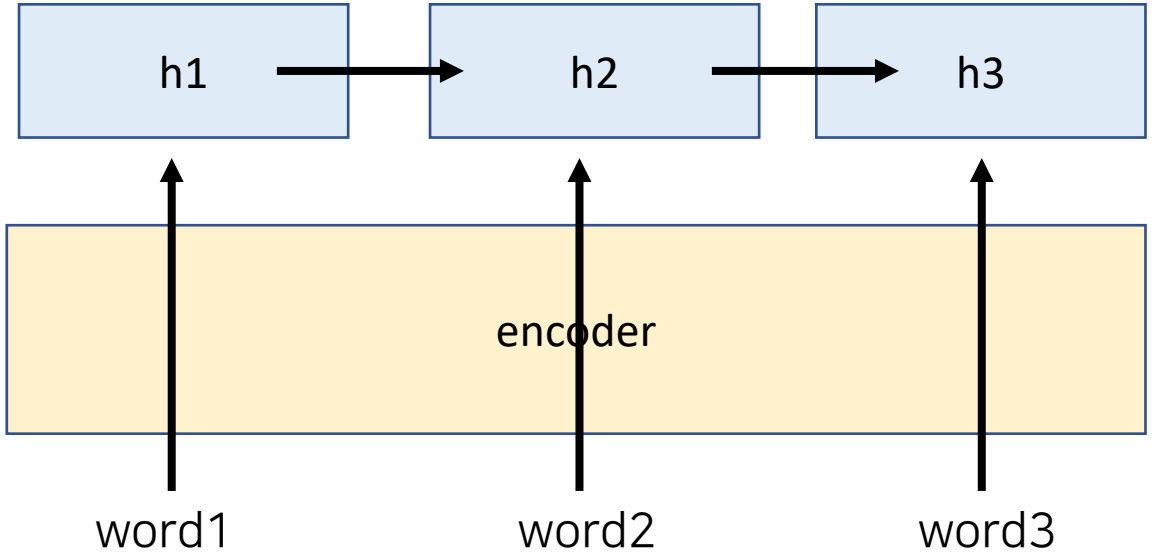
Attention in Seq2Seq Learning

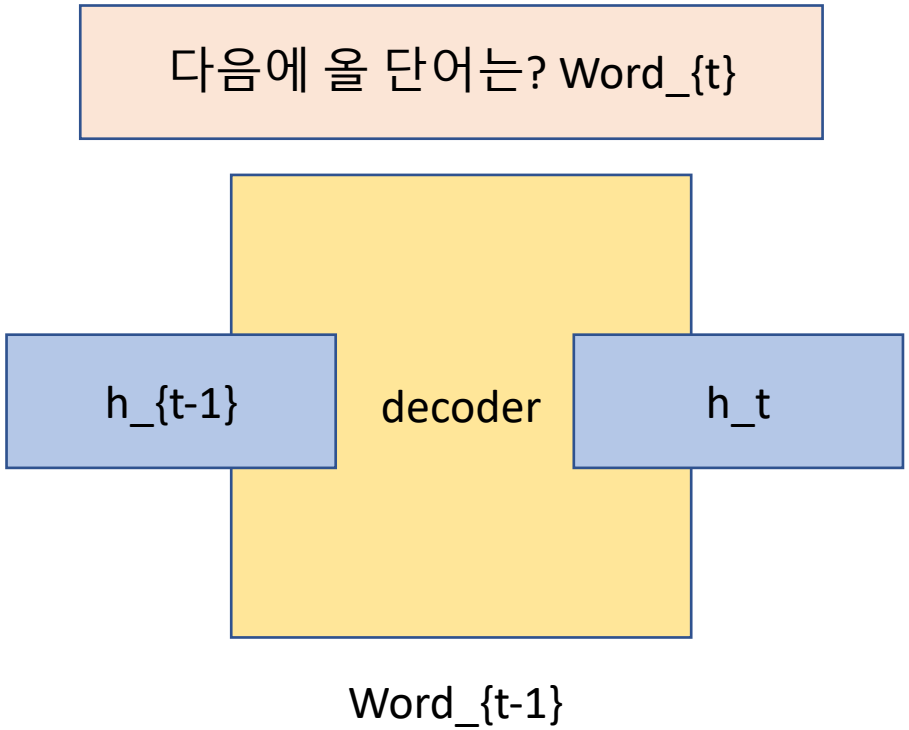
✓ Attention mechanism의 가정에 따른 Process

1. 디코더의 각 스텝에서는 encoder의 모든 step의 hidden state와 유사도를 측정한다.
2. 유사도를 이용하여 인코더의 모든 hidden state에 대한 가중합을 구하여 context vector를 구한다.
   - 모든 hidden state를 이용하므로 입력 시퀀스가 길어져도 모든 정보를 반영할 수 있다.
   - 앞부분 hidden state와 뒷부분 hidden state가 확률 형태로 반영되기 때문에 gradient가 소실에 대한 위험을 줄일 수 있다.
3. Context vector를 기반으로 다음에 올 단어를 예측한다.

Attention in Seq2Seq Learning

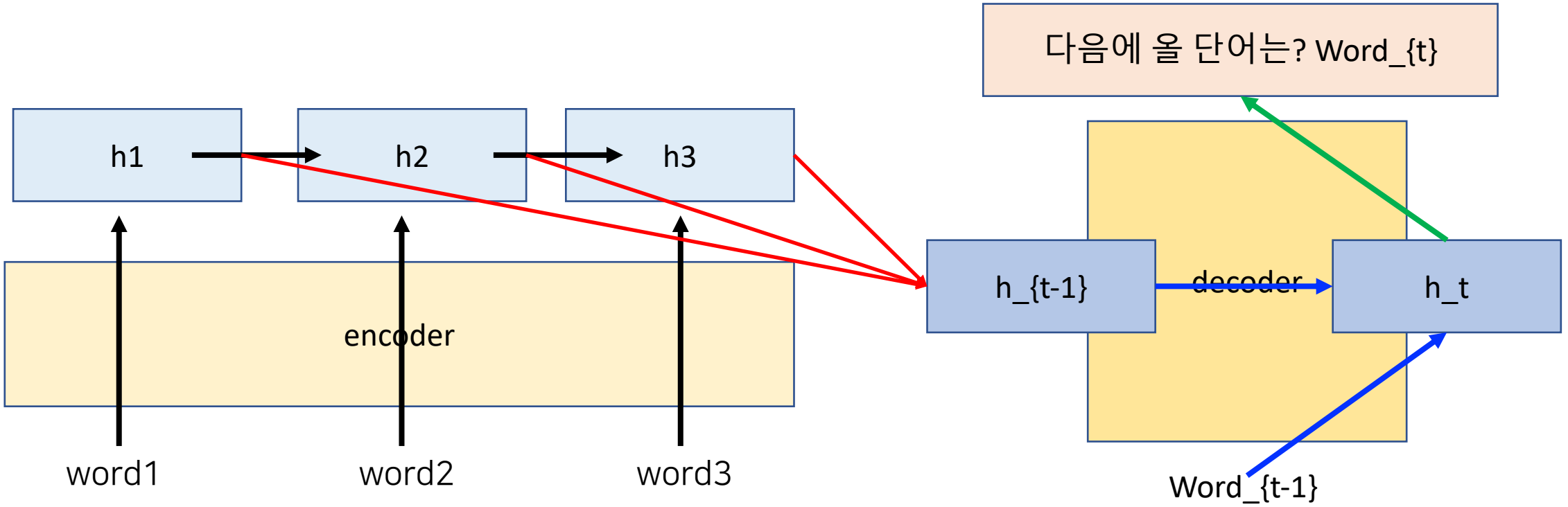
✓ 디코더의 각 스텝에서는 encoder의 모든 step의 hidden state와 유사도를 측정한다.





Attention in Seq2Seq Learning

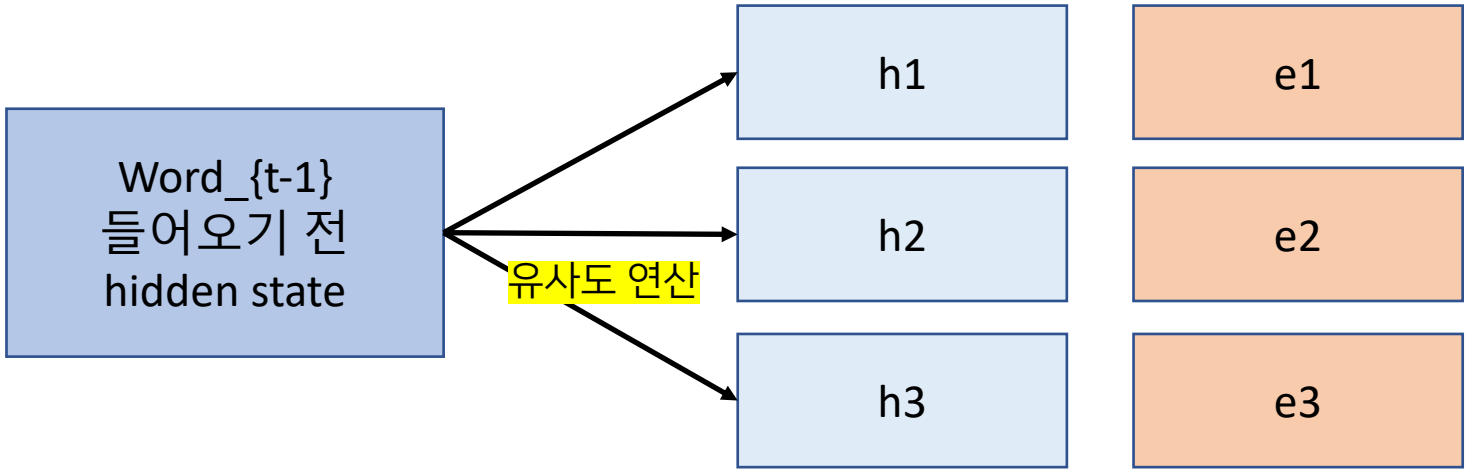
✓ 디코더의 각 스텝에서는 encoder의 모든 step의 hidden state와 유사도를 측정한다.



Attention in Seq2Seq Learning

✓ **디코더의 각 스텝에서는 encoder의 모든 step의 hidden state와 유사도를 측정한다.**

- s_{t-1} : decoder의 이전 시점의 hidden state
- h : encoder의 hidden state들



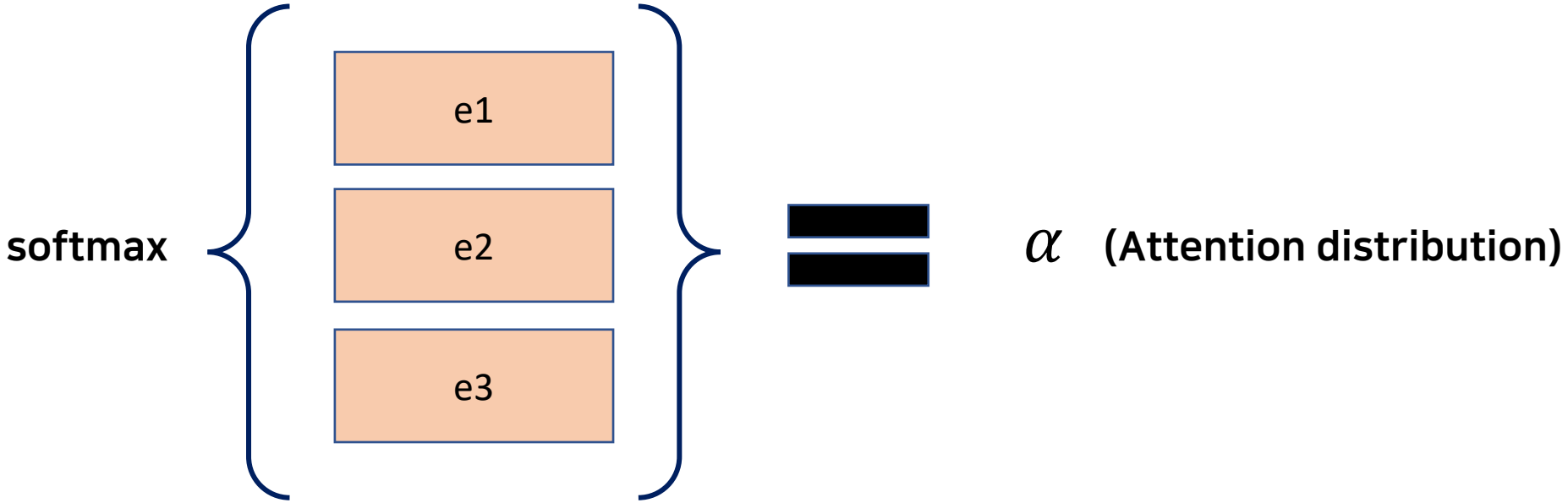
$$\mathbf{e} = [e_1, e_2, \cdots, e_N]$$

(단, $e_k = f(s_{t-1}, h_k)$. $(k = 1, 2, \cdots, N)$)

f는 유사도 함수로 다양하게 사용 가능
(ex. Dot product, 학습 가능한 파라미터)

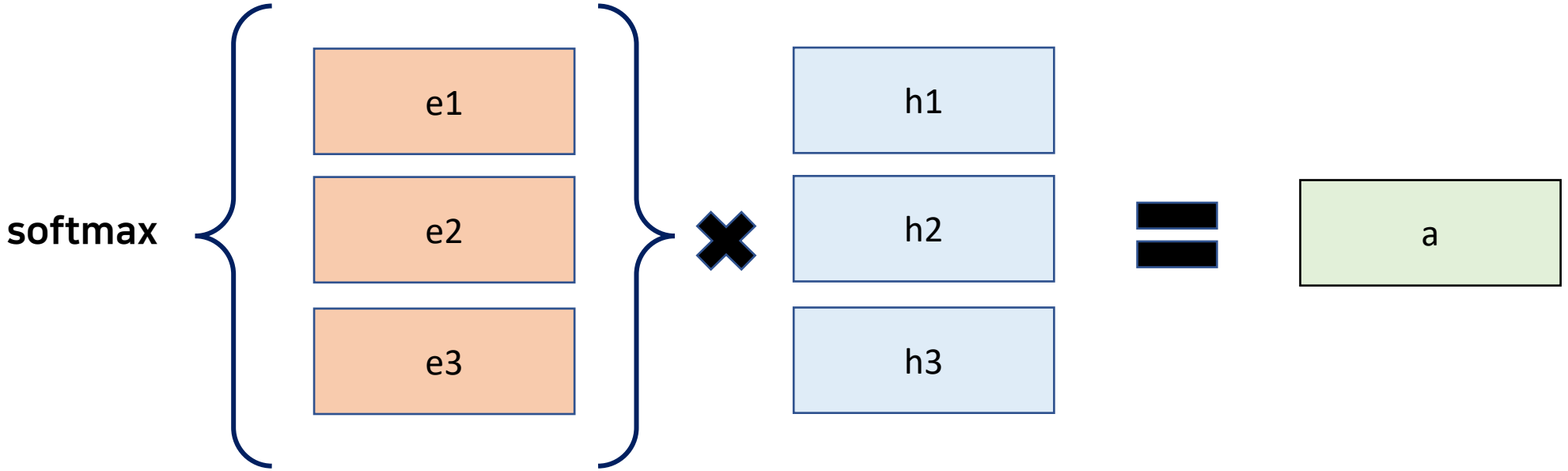
Attention in Seq2Seq Learning

✓ 유사도를 확률로 바꾸고, 인코더의 hidden state에 대한 가중합을 구한다.



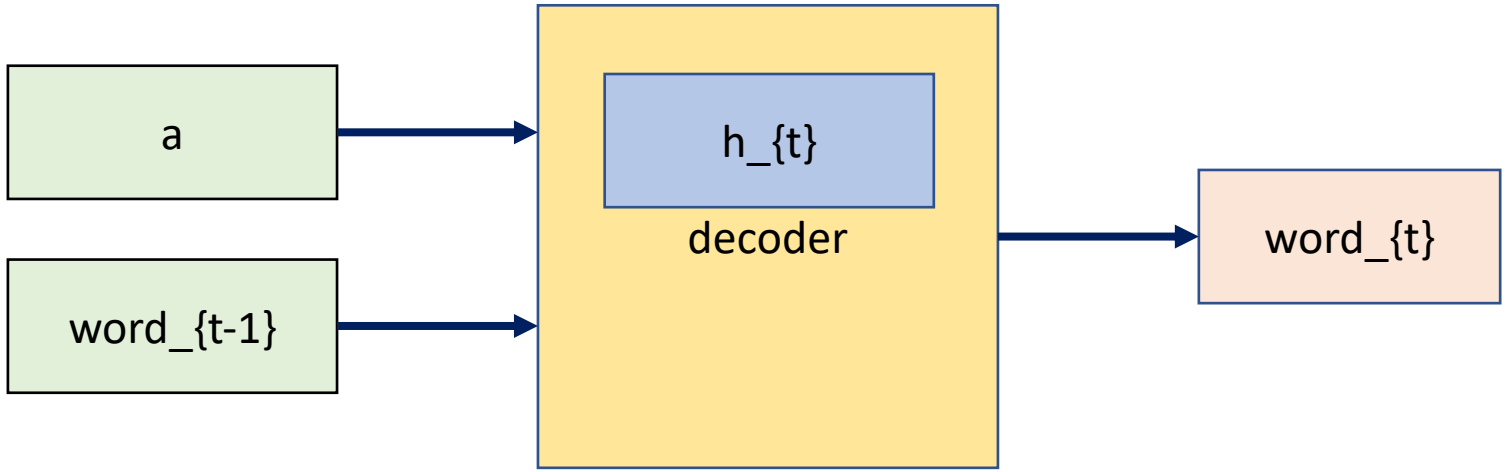
Attention in Seq2Seq Learning

✓ 유사도를 확률로 바꾸고, 인코더의 hidden state에 대한 가중합을 구한다.



Attention in Seq2Seq Learning

✓ 어텐션 값과 디코더의 입력 단어 벡터를 합쳐 디코더 hidden state를 업데이트 하고, 다음에 올 단어를 예측한다.



03 Transformer

Transformer

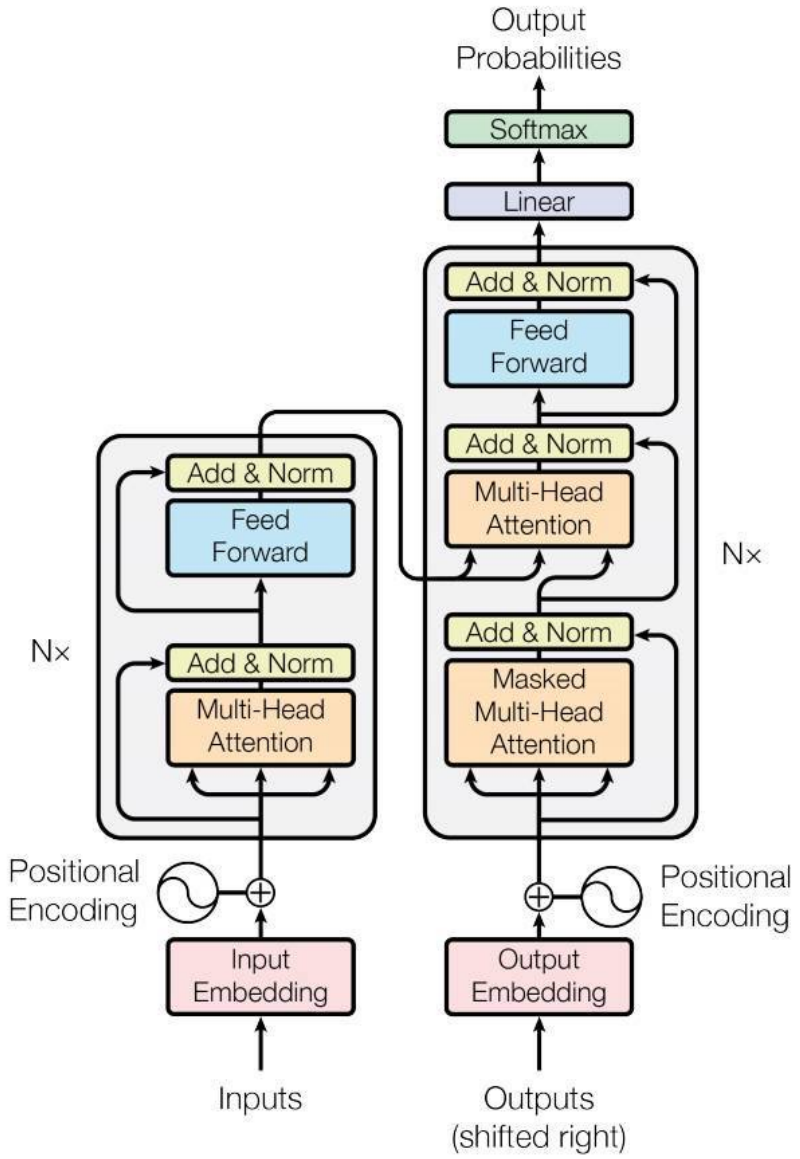
✓ 등장 배경

- RNN기반의 모델 문제점
  - RNN의 특징 : 이전 시점의 hidden state를 이용해서 현 시점의 hidden state를 업데이트함.
  - 문제점 : sequential computation 문제, 병렬적인 계산 불가, 긴 텍스트의 한계, 메모리 제한

Transformer

✓ 모델 구조

- Encoder * 6개
  - Multi-Head Attention
  - Feed Forward Neural Network
- Decoder * 6개
  - Multi-Head Attention (self-attention)
  - Multi-Head Attention (encoder-decoder attention)
  - Feed Forward Neural Network

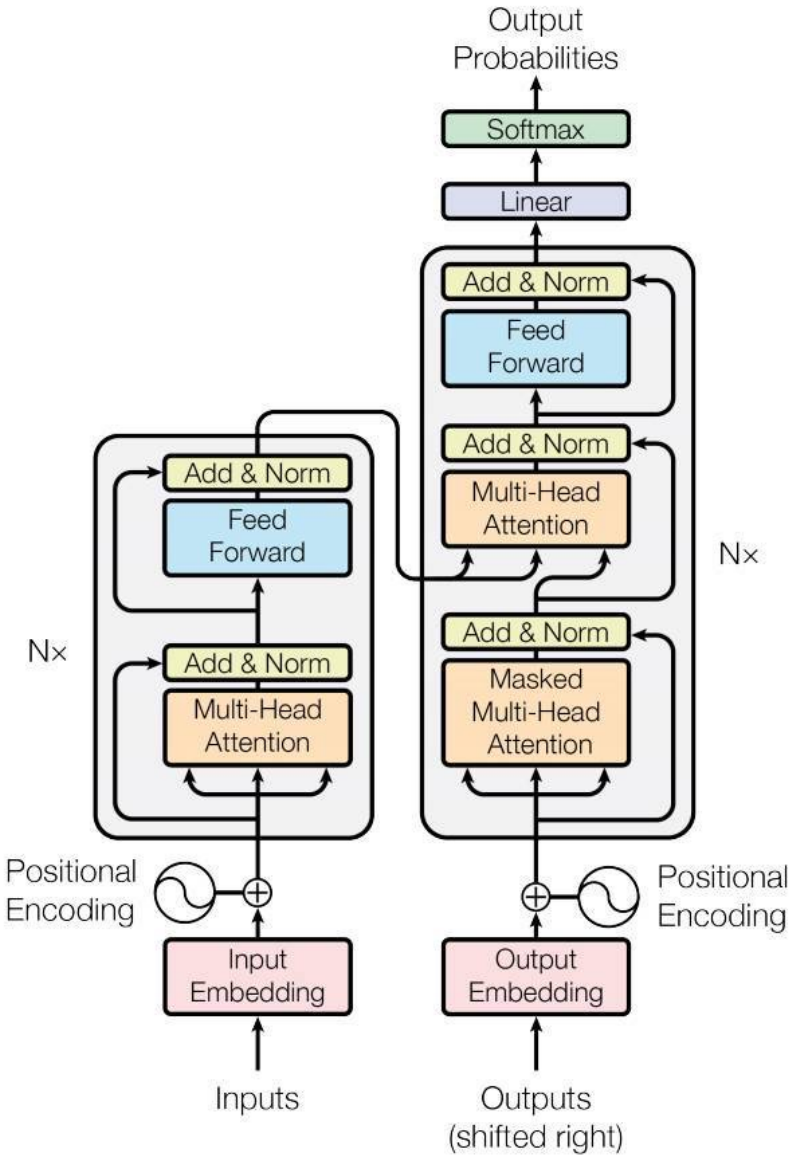


Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. ArXiv, abs/1706.03762.

Transformer

✓ 모델 내 각 모듈의 역할

- Encoder
  - Multi-Head Attention
    - 여러 view를 가진 어텐션
  - Feed Forward Neural Network
    - 여러 view를 합쳐서 하나의 vector로 만들어주는 과정
- Decoder
  - Multi-Head Attention (self-attention)
    - 현 시점의 뒷부분은 masking하고 앞부분에 대해서만 attention
  - Multi-Head Attention (encoder-decoder attention)
    - Encoder의 정보들을 반영한 attention *(Seq2seq learning에 등장한 개념)*
  - Feed Forward Neural Network



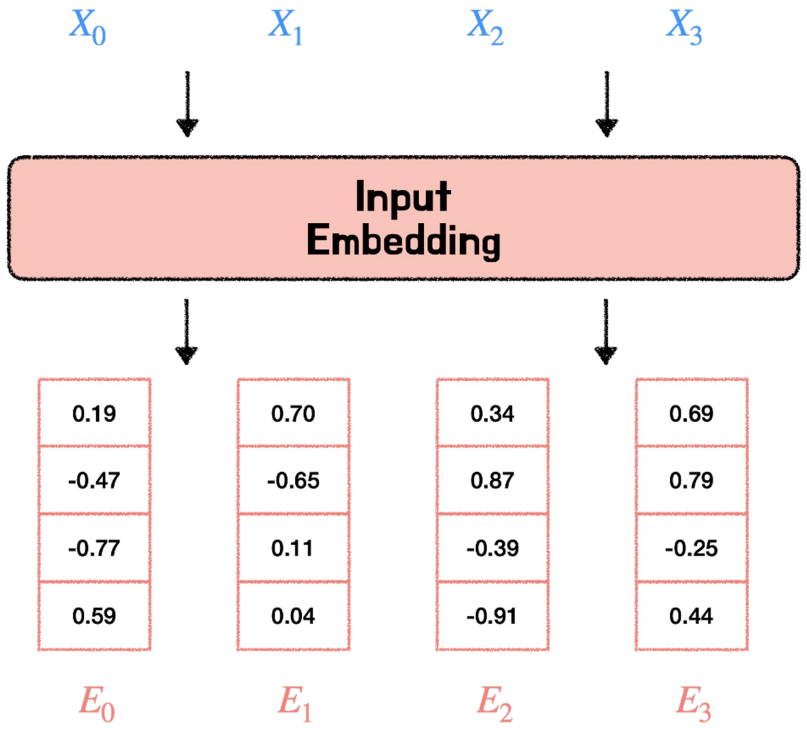
Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. ArXiv, abs/1706.03762.

Transformer

✓ Transformer의 Input

1. Input Embedding
   - input sequence : [나는, 카페, 에서, 커피, 를, 마신다.]
   - Tokenizer : [3, 25, 99, 21, 7, 100]
   - Embedding Vector

     [[0.012, 0.24, 0.3335, ..], [0.35, 0.24, ···], ···]
     - Shape = (sequence length, embedding size)
     - 미리 학습된 것(word2vec)을 사용할 수도 있고, 랜덤하게 초기화해서 사용할 수 있음.



https://www.blossominkyung.com/deeplearning/transfomer-positional-encoding

Transformer

✓ Transformer의 Input

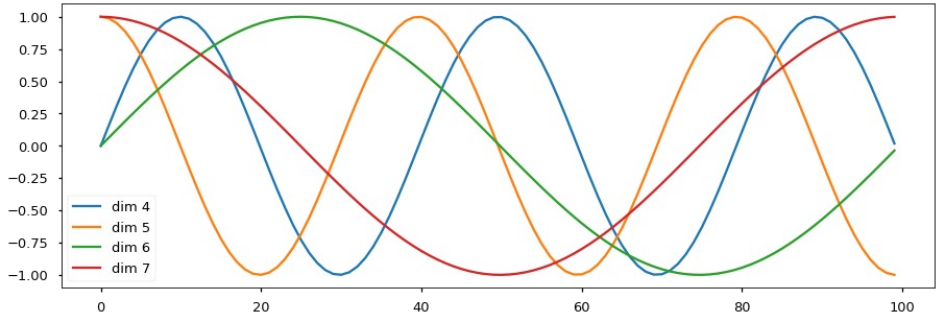
2. Positional Encoding

- 필요성
  - 나는 점심에 피자를 먹고 저녁에 치킨을 먹었다.
  - 나는 점심에 맥주를 먹고 저녁에 피자를 먹었다.
  - 토큰의 순서가 바뀌었지만, 위치 정보가 반영되지 않으면 두 sequence는 동일한 의미를 가진다.
- 조건
  - 시퀀스 길이와 상관없이 각 위치에 해당하는 식별자를 가져야 한다.
    - Input문장이 바뀌어도 같은 위치에 있는 토큰들은 같은 위치 임베딩 값을 가진다.
    - (k/단어수)를 위치 임베딩으로 사용할 수 없음.
  - 위치 임베딩 값이 너무 크면 안된다.
    - Input embedding에 더해져서 사용되기 때문에 위치 정보가 너무 커져버려 의미 정보를 잃게 된다.

✓ Transformer의 Input

2. Positional Encoding

- Cosine & Sine 함수
  - [-1, 1]의 값을 가지는 주기함수
    - 장점1 : 동일한 위치에 동일한 값을 부여할 수 있음.
    - 장점2 : 너무 크지 않은 값
  - 하지만, 주기함수를 가지면 다른 위치에 동일한 위치 임베딩 값을 가지게 되지 않을까?
    - 해결 : 다양한 주기를 가진 sine, cosine 함수를 사용한다.
    - 위치 임베딩이 10차원이라면 10개의 다른 주기를 가진 sine, cosine함수를 이용함.

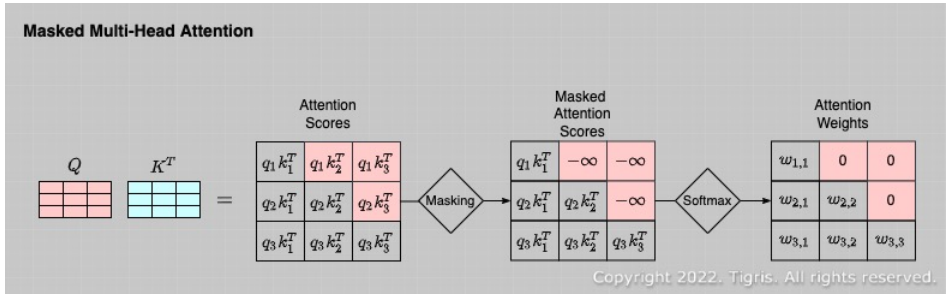


https://voidism.github.io/notes/2020/01/26/What-has-the-positional-embedding-learned/

Transformer

✓ Transformer에 있는 3가지 attention

1. Encoder내의 self-attention
   - Query = key = value = 이전 시점의 hidden state
2. Decoder내의 masked self-attention
   - 현시점보다 뒤의 시점에 해당하는 부분은 masking 처리 : 미래에 해당하는 내용은 가리기
   - Masking되는 부분까지 동시에 연산 가능(-inf)



3. Decoder내의 encoder-decoder attention
   - Query : 이전 시점의 hidden state
   - Key, value : Encoder의 모든 step의 hidden states

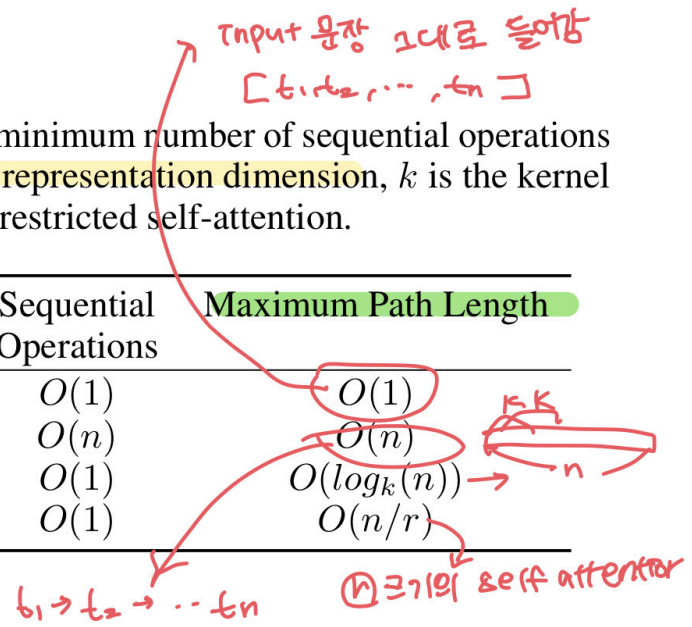
Transformer

✓ why Self-Attention

- total computational complexity per layer : O(1)
- 최소한의 연속적 연산에 의해 측정하는, 병렬적 계산이 가능한 양
- long-range dependecies사이의 max path length 비교

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

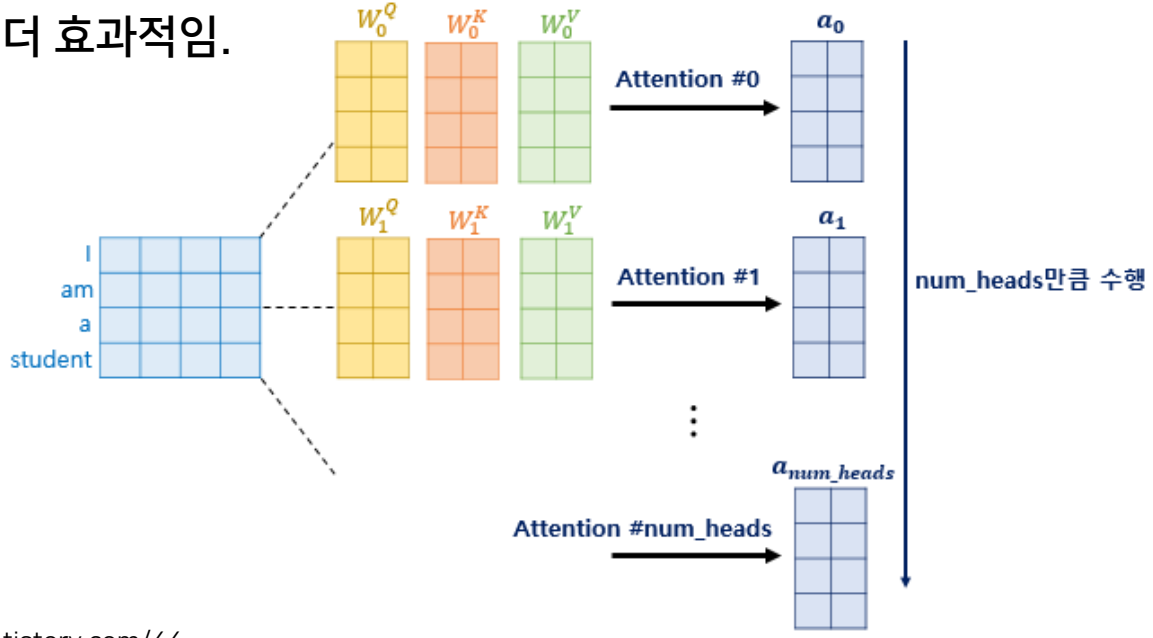


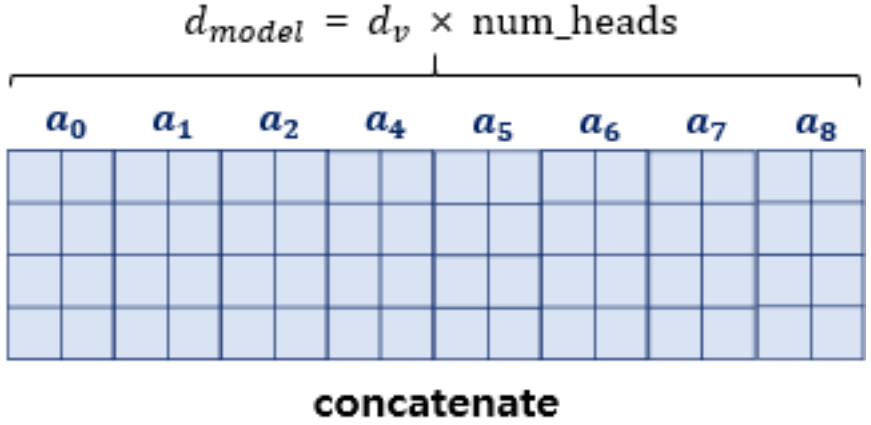
Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., & Polosukhin, I. (2017). Attention is All you Need. ArXiv, abs/1706.03762.

Transformer

✓ Multi-head Attention

- 1개의 self-attention = (4,8)
- 4개의 self-attention = (4,2) * 4
- 성능 : (4,8) < (4,2) * 4
- 효율 : (4,8) < (4,2) * 4 *weight 수를 줄일 수 있어 속도가 빨라지며 병렬적 연산까지 가능*
- 효과 : 여러 사람의 시각에서 attention을 반영하는 것과 같은 결과. 한명이 보는 것보다 여러 명이 보는 것이 더 효과적임.



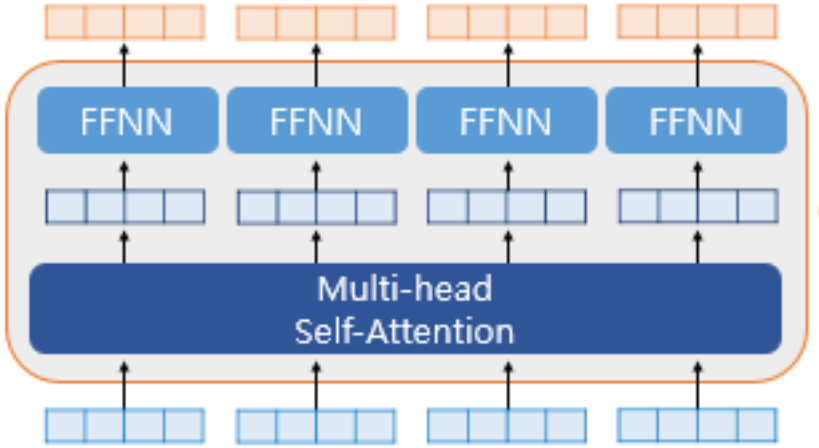


https://codingopera.tistory.com/44

Transformer

✓ Position-wise FFNN

- 각 head에서 나온 vector는 자신의 관점에 치우쳐진 정보가 담겨있음.
- FFNN의 역할 : Multi-head attention의 결과를 골고루 반영하여 하나의 벡터로 만들어 줌.
- Position-wise가 붙는 이유?
  - 각 토큰마다 FFNN을 통과함
  - 아래에 보이는 FFNN은 모두 같은 weight를 가진 레이어



X
(seq_len, d_model)

F1 = W1* x +b1

F1 shape : (seq_len, d_ff)
= (seq_len, 2048)

F2 = W2 * max(0,F1) + b2

F2 shape = (seq_len, d_model)
= (seq_len, 512)

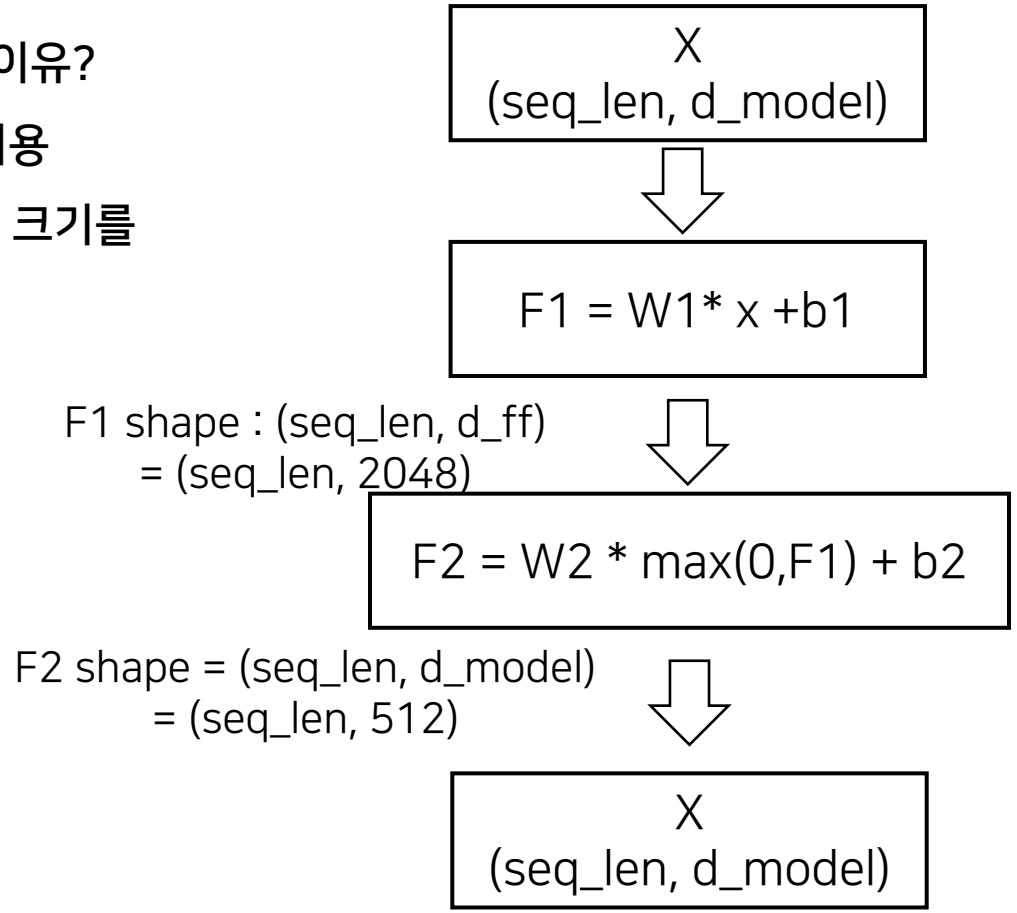
X
(seq_len, d_model)

https://wikidocs.net/31379

Transformer

✓ Position-wise FFNN

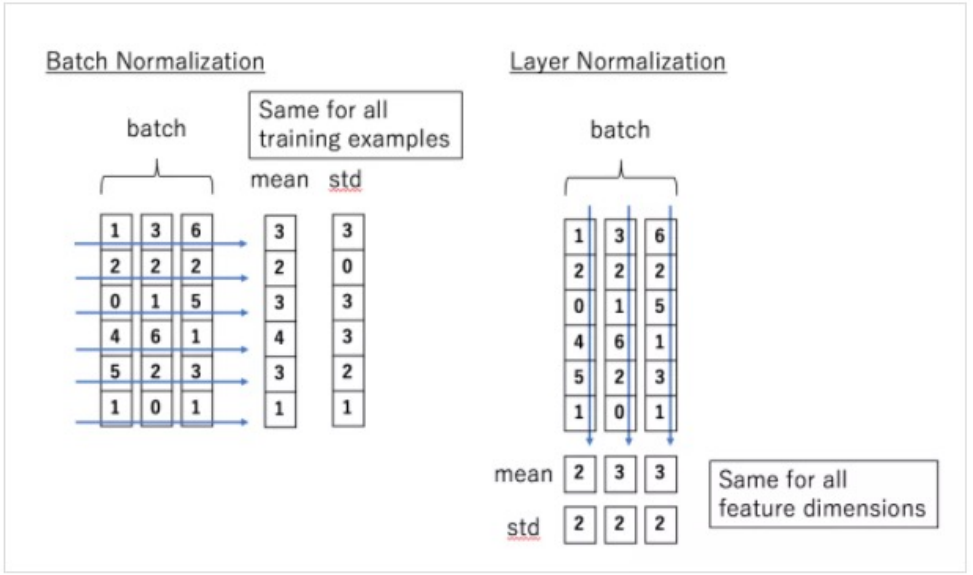
- D_model -> D_ff -> D_model로 Projection하는 이유?
  - 각 head의 정보를 섞기 위해서 projection을 이용
  - 추가적으로 transformer는 input과 output의 크기를 동일하게 맞추려고 함.



https://github.com/yukyunglee/Transformer_Survey_Study/blob/main/Presentation/TSS_%231_Transformer_Basic.pdf

✓ Add & Norm

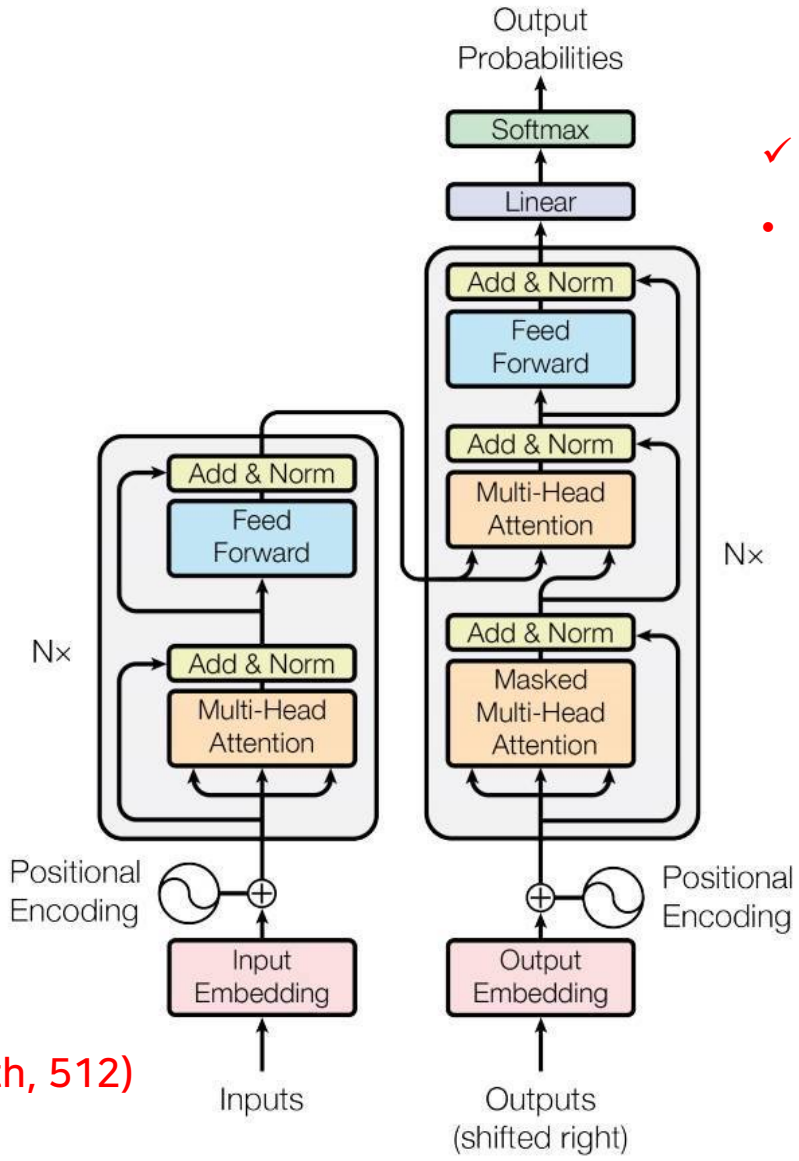
- **Add : Residual Connection** (Deep Residual Learning for Image Recognition. CVPR 2016 [paper])
  - Transformer의 레이어가 깊어지면서 경사 소실 문제 발생 우려
  - Residual Connection은 ResNet에서 처음 제안한 구조로 모델의 깊이가 깊어져도 경사 소실 문제가 발생하지 않고 전체 파라미터 업데이트가 일어 남.
- Layer Normalization
  - Batch Normalization은 mini-batch를 기준으로 평균과 분산을 계산함.
  - Layer Normalization은 input을 기준으로 평균과 분산을 계산함.
  - 효과 : Input scale에 영향을 받지 않아 안정적인 학습이 가능함.



https://velog.io/@stapers/%EB%85%BC%EB%AC%B8-%EC%8A%A4%ED%84%B0%EB%94%94-Week4-5-Attention-is-All-You-Need

Transformer

✓ Parameter 수



✓ FFNN

- W1 = (512, 2048)
- W2 = (2048, 512)

✓ Attention

- Head 개수 : 8
- Query, key, value : (512, 64)

Input = (max length, 512)

✓ Linear

- 512 * 50,000(vocab size)

- Encoder * 6
  - Attention * 8
    - 4,718,592개
  - FFNN
    - 12,582,912개
- Decoder * 6
  - Attention * 8 * 2
    - 9,437,184개
  - FFNN
    - 12,582,912개
- Linear
  - 25,600,000개