# Attention, Transformer

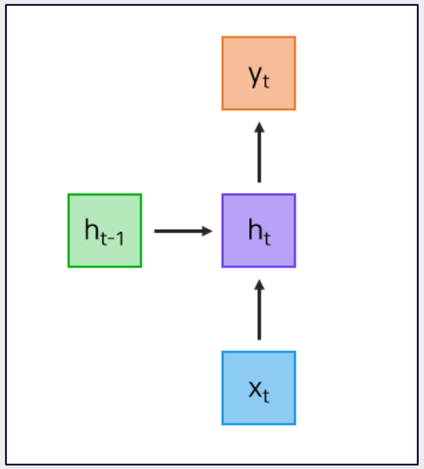
## Deep Session 9차시

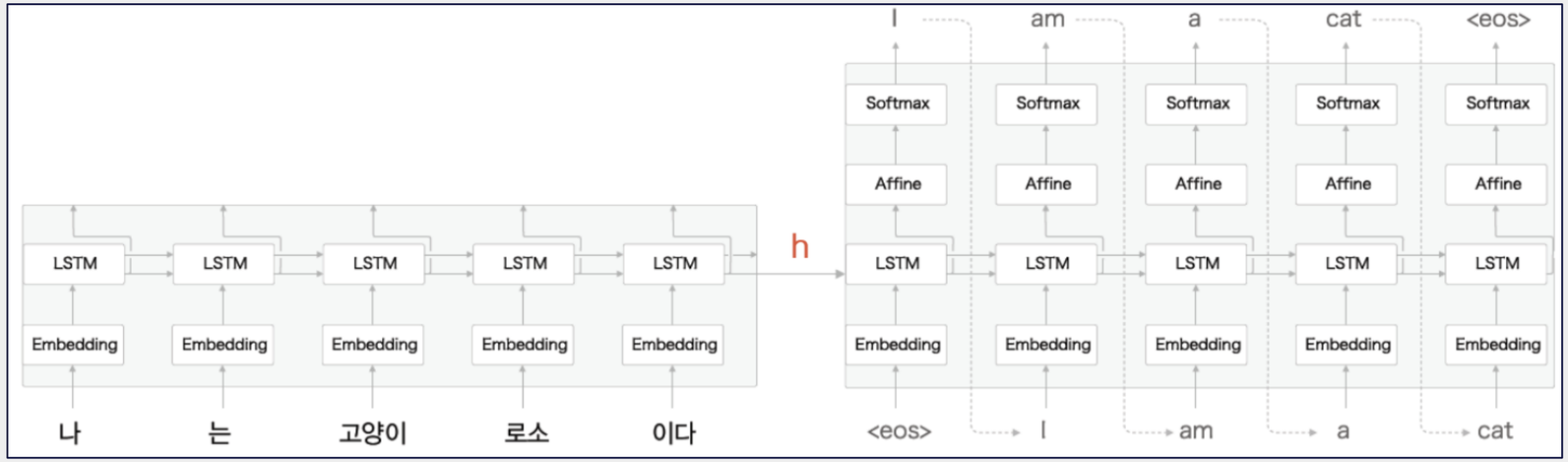
빅데이터 분석 학회 D&A

# CONTENTS.

# Attention 등장 배경

## Seq2Seq의 문제점

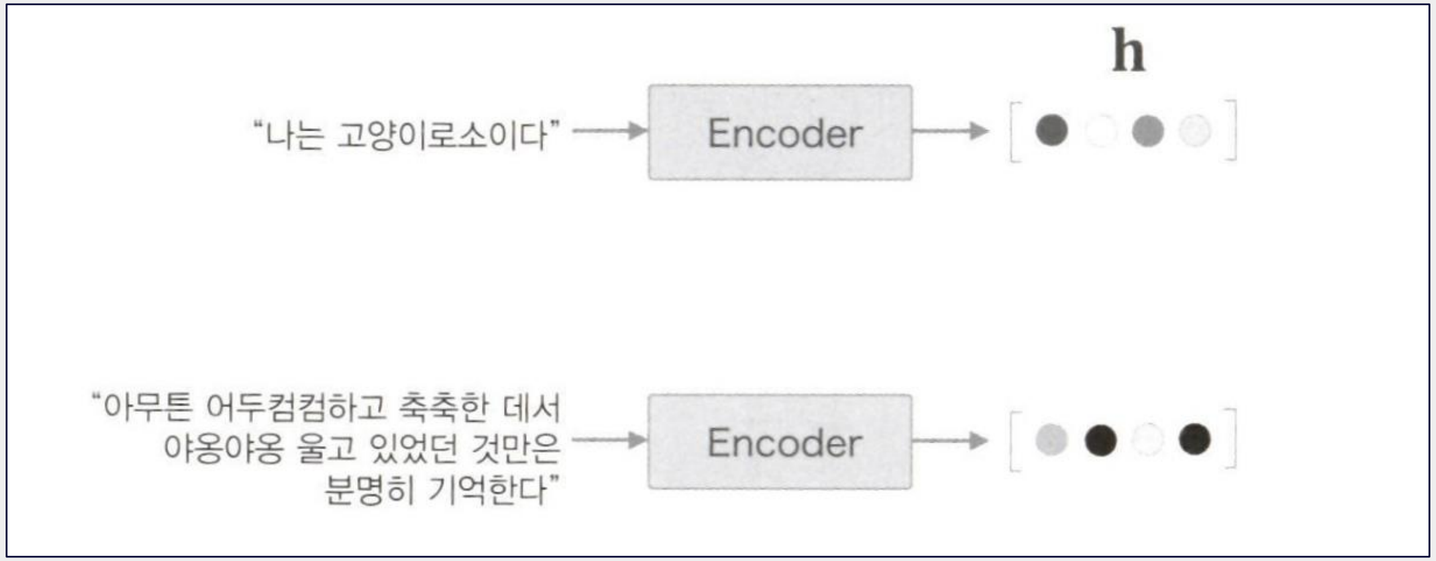



1. RNN 기반의 Seq2Seq 모델은 이전 단계의 hidden state $h_{t-1}$와 현재 입력 $x_t$를 이용해 새로운 hidden state $h_t$ 를 생성함
   → 이 과정이 순차적으로 이루어지기 때문에 병렬 처리가 어려움
2. 입력 문장의 길이가 길어질수록 RNN 학습 과정에서 기울기 소실 (Gradient Vanishing) 문제가 발생할 수 있음
3. 입력 문장의 길이에 상관없이 항상 고정 길이의 벡터를 출력함 → 이로 인해 입력 문장의 정보를 전부 활용하지 못함

# Attention 등장 배경

## Seq2Seq의 문제점



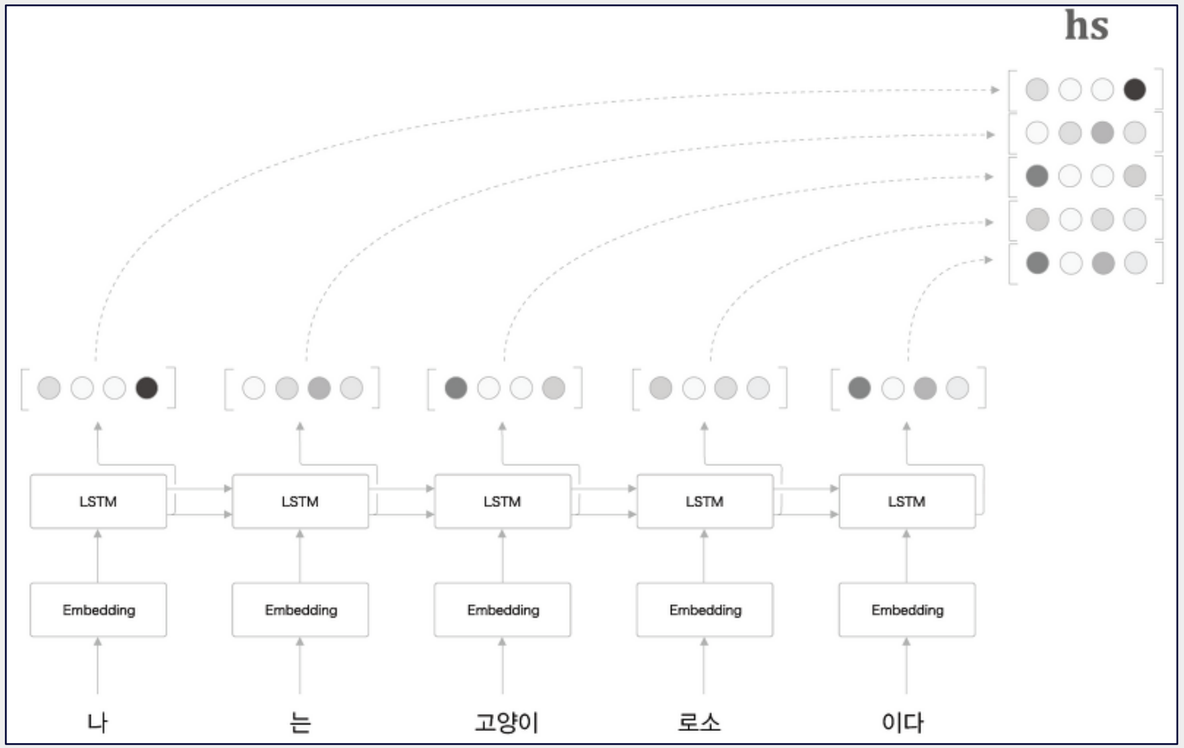

입력 문장의 길이에 상관없이 항상 고정 길이의 벡터(마지막 hidden state)를 출력함 → 이로 인해 입력 문장의 정보를 전부 활용하지 못함

→ 입력 문장의 길이에 맞게 Encoder의 출력 벡터의 길이를 변화시키면 어떨까?

Attention

# Encoder 개선

## 은닉층의 마지막 시점을 출력하는 것이 아닌 모든 시점에서의 벡터를 출력하자!




각 시점에서의 나오는 출력 벡터

→ 해당 시점 입력 단어의 정보 + 이전 시점에서의 정보

만약 모든 벡터를 활용한다면?

→ 각 시점에서의 정보를 최대화하여 활용할 수 있음

모든 벡터를 활용하는 방법은?

→ 각 시점에서의 출력을 하나의 행렬로 묶어 Decoder로 전달하자!

행렬을 Decoder의 어느 부분에 전달해야 할까?

→ 각 시점에서의 정보를 모두 가지고 있으니 모든 시점에 전달하자!

→ Encoder에서 전달받은 행렬을 어떻게 잘 활용할 수 있을까?

Attention

# Decoder 개선

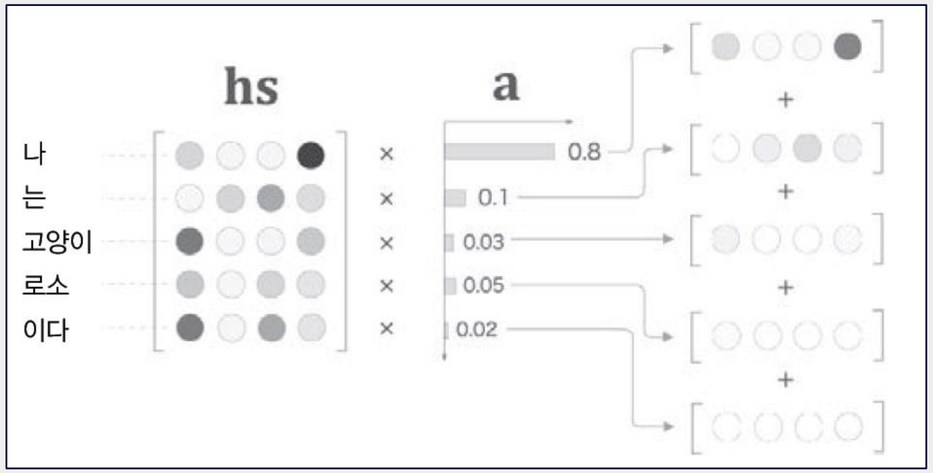
## 전달받은 행렬에서 필요한 정보만 뽑아내자!

본인 시점에 맞는 벡터만 선택해서 활용하자

→ 순전파 과정에선 가능하나 오차역전파 계산이 불가능

그럼 오차역전파 계산이 가능하면서도 원하는 벡터만 선택할 수 있는 방법은?

→ 행렬 내 각 벡터의 중요도를 계산하여 가중치 합을 구하자!



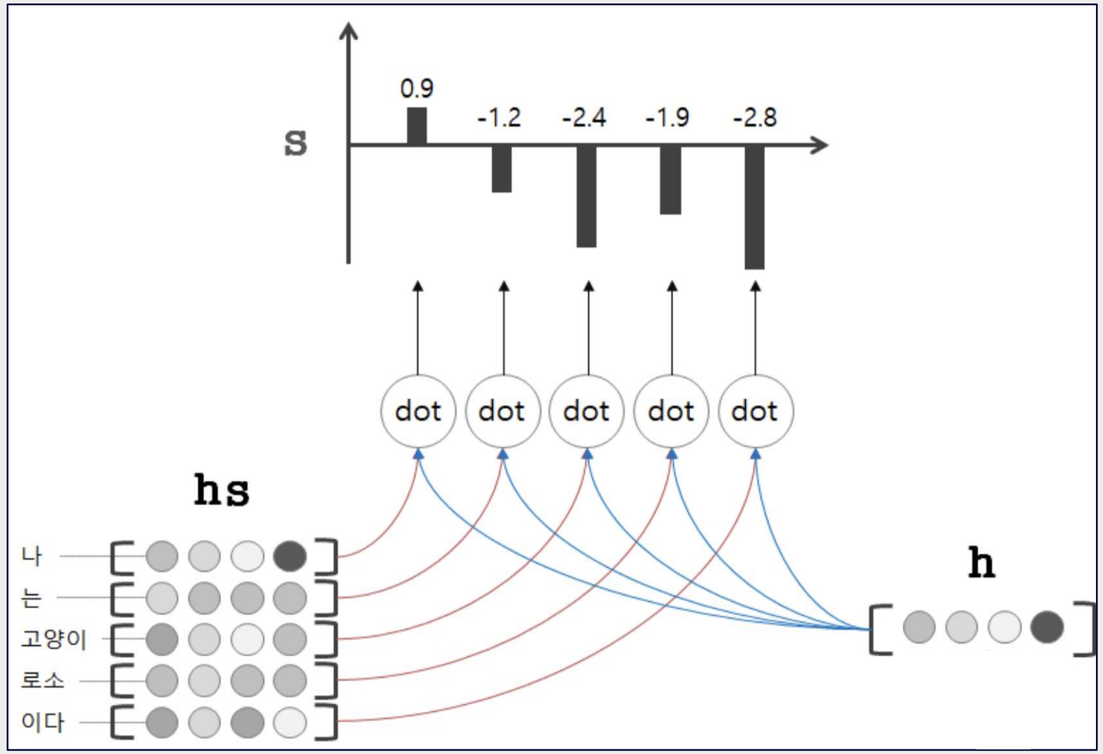

벡터의 중요도(a)

→ Decoder의 각 시점에서의 은닉 상태 벡터와 행렬의 각 벡터가 얼마나 유사한지

→ 벡터의 중요도는 어떻게 구하는 걸까?

# Decoder 개선

## hs 행렬의 각 벡터가 해당 시점 h 벡터와 얼마나 유사한지 구하자!

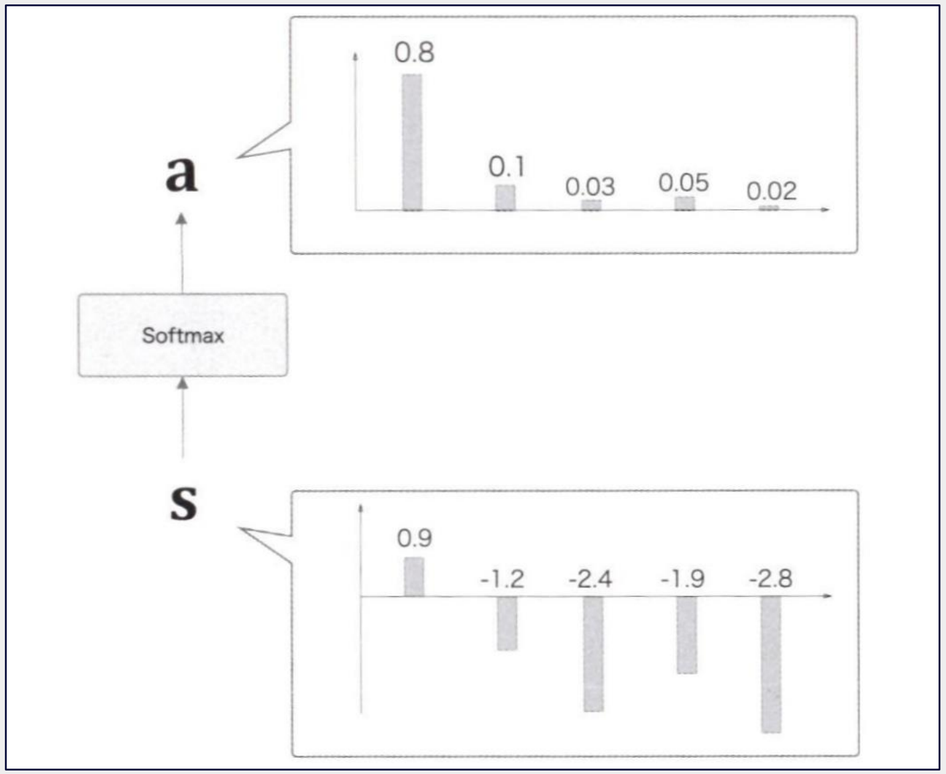



얼마나 유사한지는 어떻게 구할까?

- hs 행렬과 h 벡터를 내적한다!
- 내적한 값이 크다 = 두 벡터의 방향이 유사하다
- 내적 값이 큰 벡터가 더 많이 유사하다는 것을 알 수 있음

# Decoder 개선

## hs 행렬의 각 벡터가 해당 시점 h 벡터와 얼마나 유사한지 구하자!

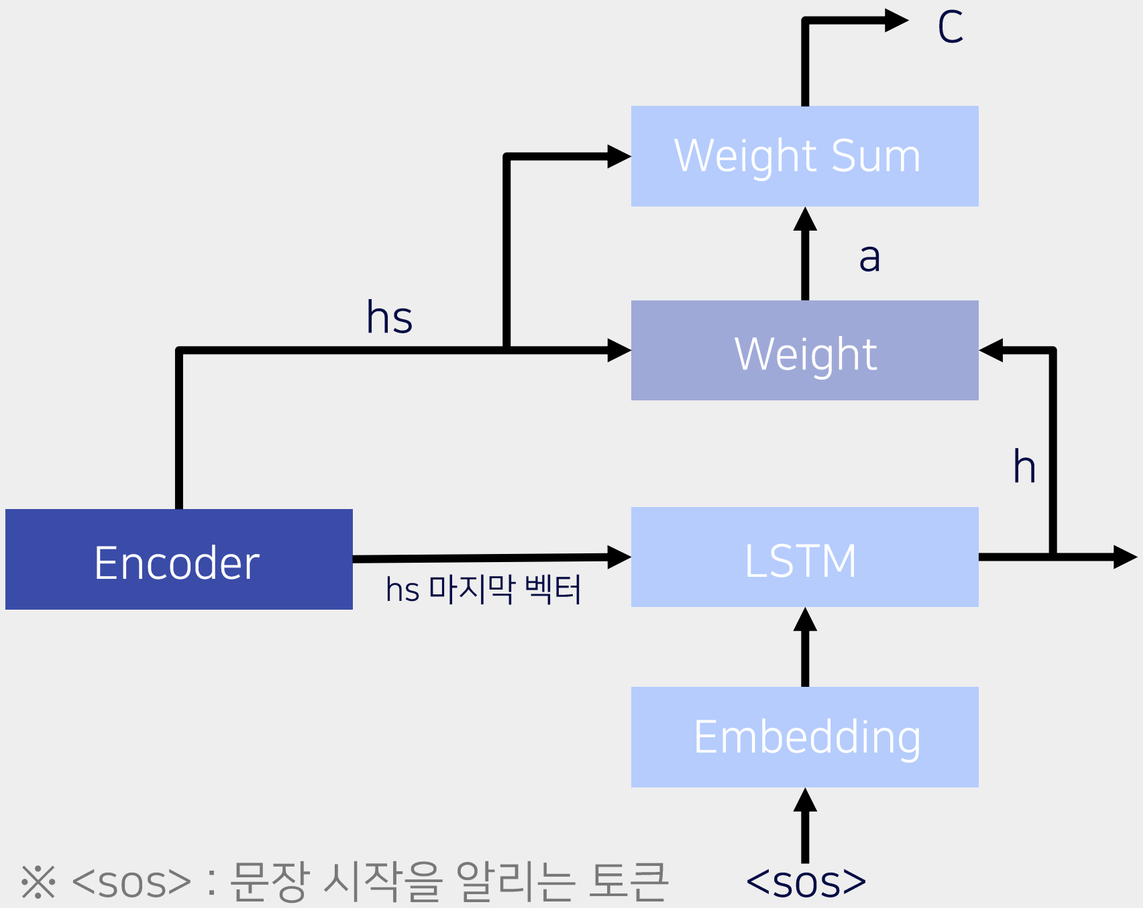



내적 값 s를 그대로 활용하면 될까?

- 구하고자 하는 것은 가중치이므로 그대로 사용하면 안됨
- 내적 값 s에 softmax 함수를 적용하여 정규화함
- 내적 값 s를 바로 사용하면 값이 정규화되지 않아 가중치가 불안정함
  softmax 함수를 통해 값을 0과 1 사이로 변환하고, 합이 1이 되도록 하여 각 가중치가 데이터의 상대적 중요도를 반영하게 함

Attention

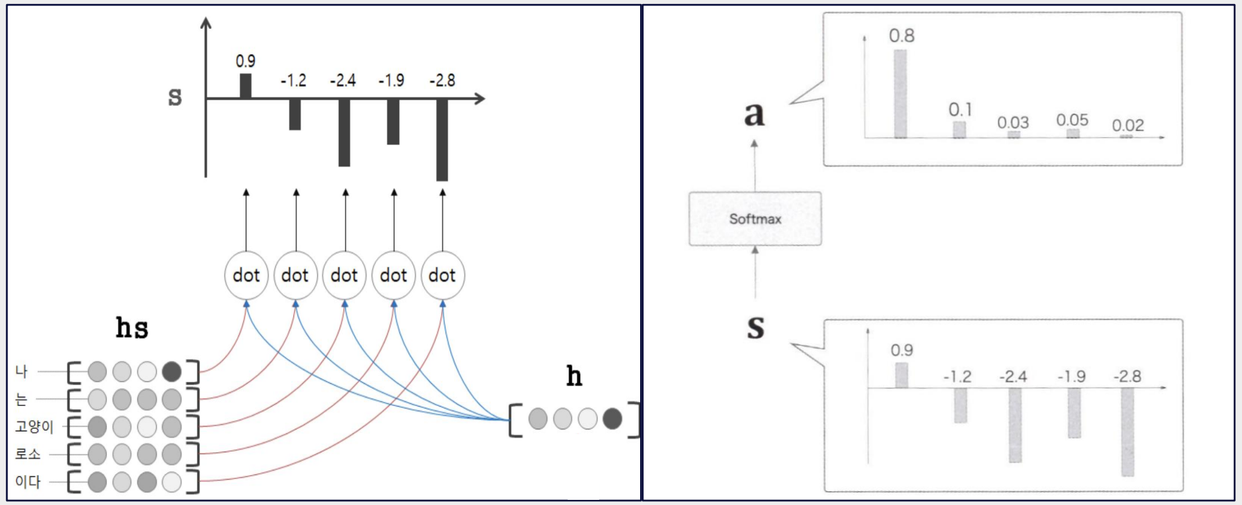
# Decoder 개선




※ <sos> : 문장 시작을 알리는 토큰

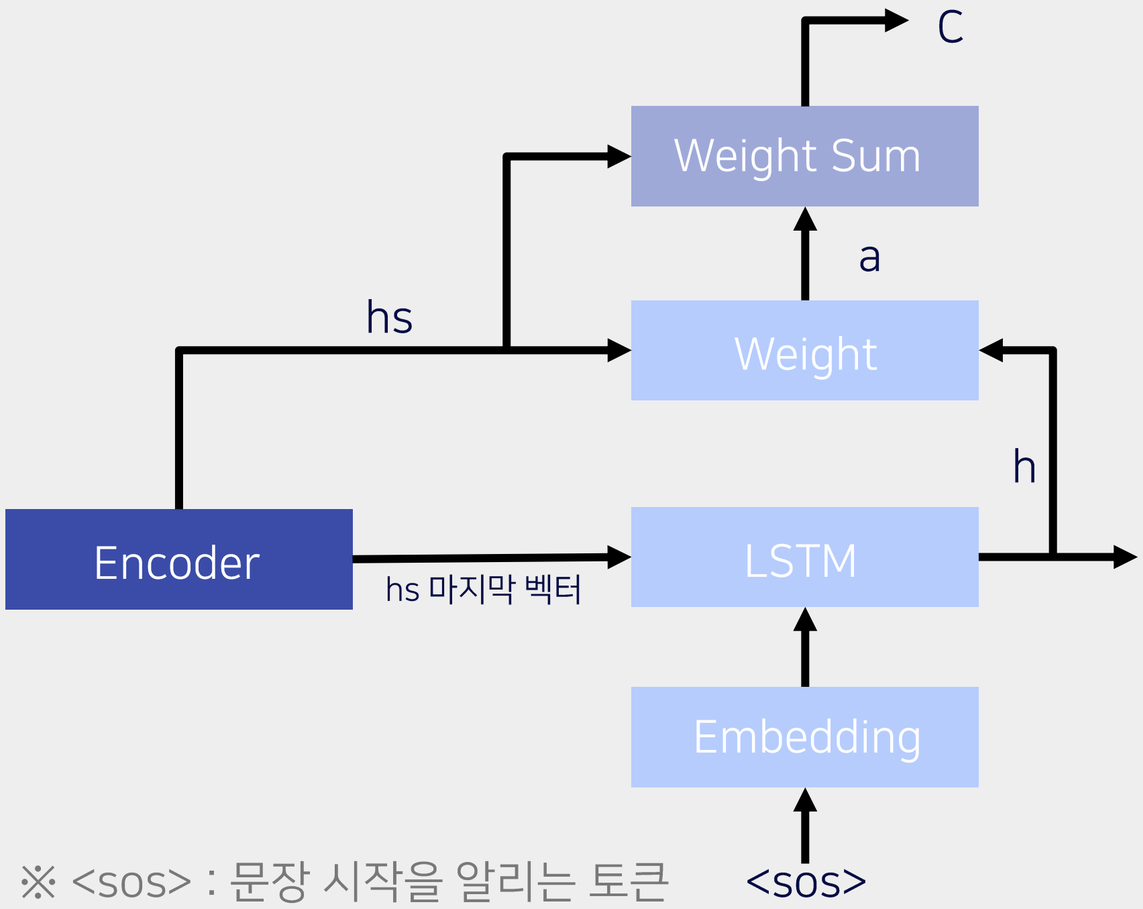
## Weight 계층

hs 행렬과 h 벡터의 내적 + softmax 함수 적용하여 가중치(a)를 구함



Attention

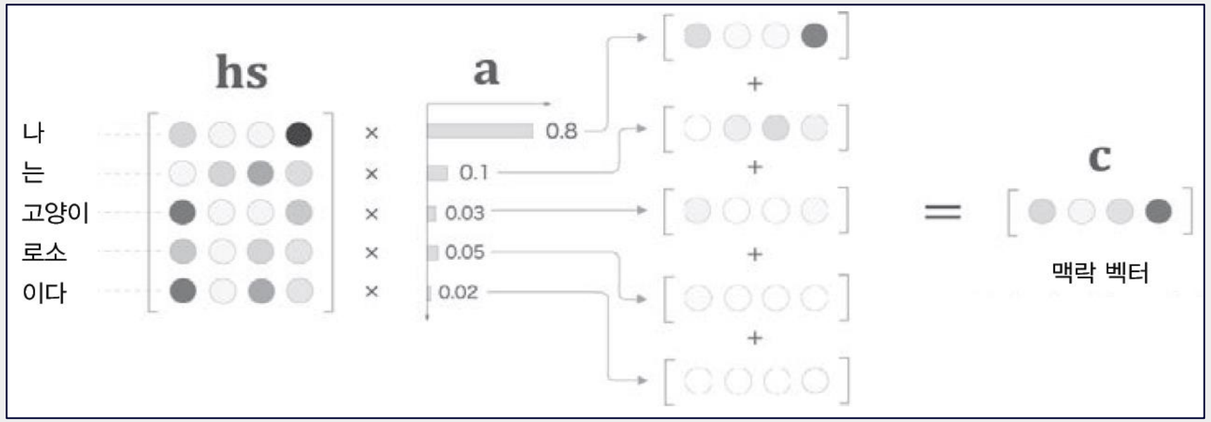
# Decoder 개선




※ <sos> : 문장 시작을 알리는 토큰

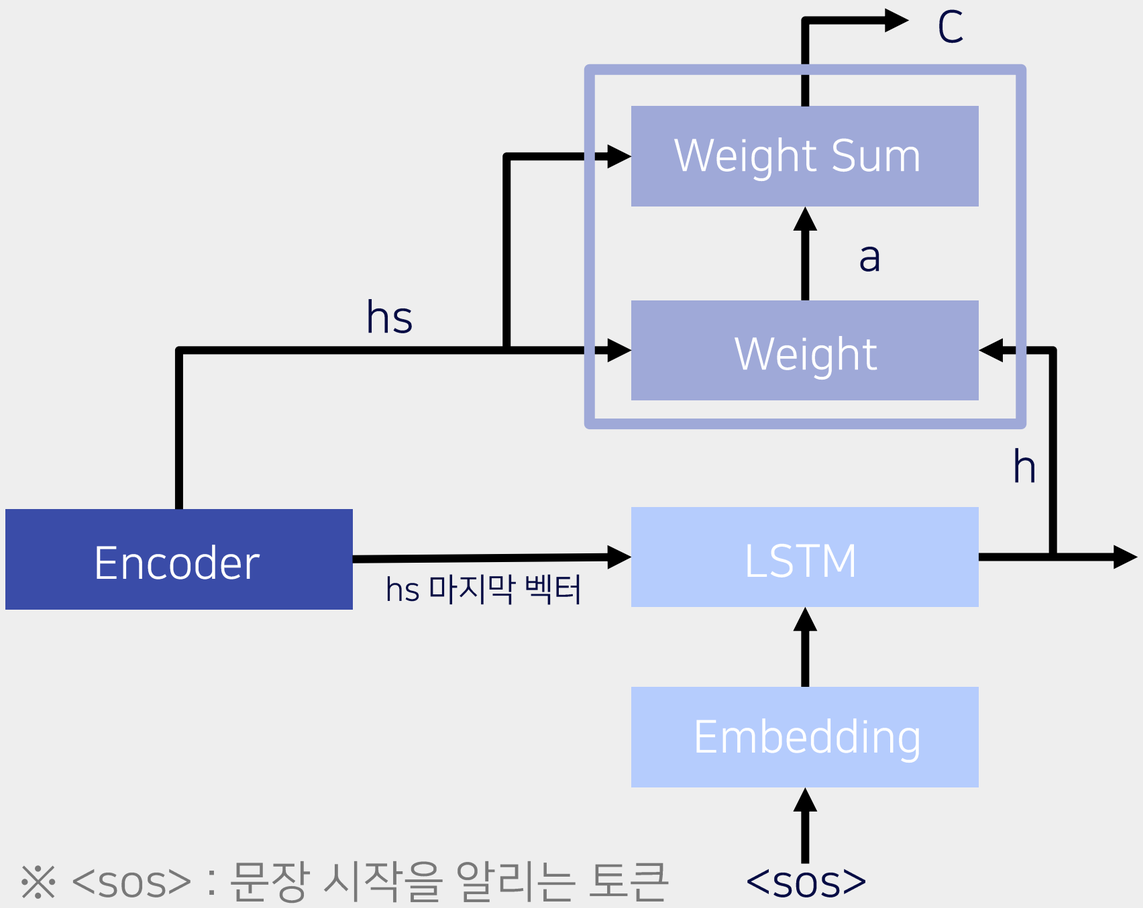
## Weight Sum 계층

- Weight 계층으로부터 입력 받은 가중치(a)와 hs 행렬을 곱함
- 곱한 결과인 맥락벡터(C)를 출력함



Attention

# Decoder 개선




※ <sos> : 문장 시작을 알리는 토큰

## Attention 계층

- Weight 계층 + Weight Sum 계층
- 맥락 벡터 C를 출력함

맥락 벡터 C란?

→ Attention 메커니즘에서, Encoder에서 전달받은 hs 행렬 중 현 시점에 가장 필요한 정보를 담은 벡터

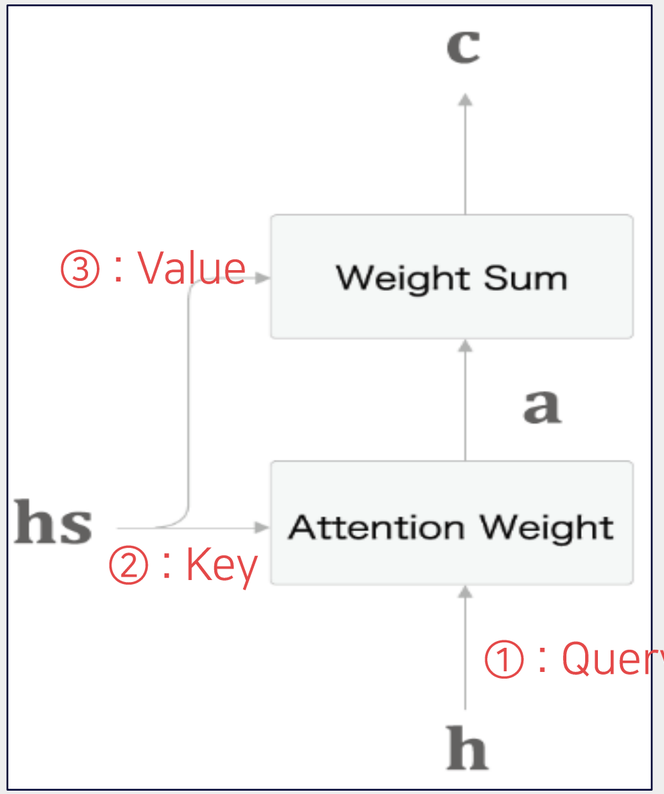
맥락 벡터 C를 사용해 단어를 예측하면?

→ Decoder가 입력 문장의 특정 부분에 집중할 수 있어, 모델이 입력 문장의 중요한 정보들을 효과적으로 활용할 수 있음

# Transformer
# 논문 리뷰

# Supplementary Information

## Query, Key, Value




기존 Attention

1. 특정 시점의 Decoder의 hidden state(h)
2. 모든 시점의 Encoder의 hidden state(hs)
3. 모든 시점의 Encoder의 hidden state(hs)

각 값에 이름을 지어보자!

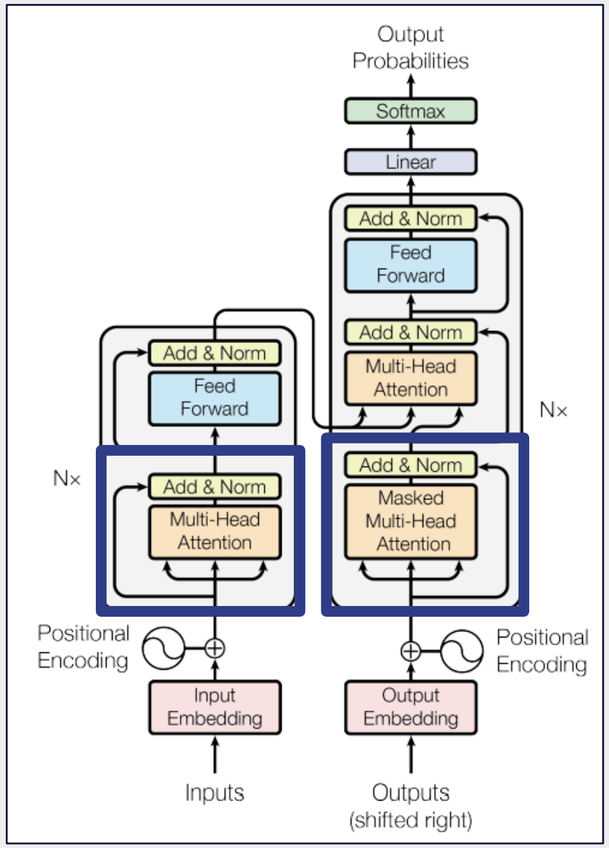
Query : 특정 시점의 Decoder의 hidden state(h)

Key : 모든 시점의 Encoder의 hidden state(hs)

Value : 모든 시점의 Encoder의 hidden state(hs)

# Supplementary Information

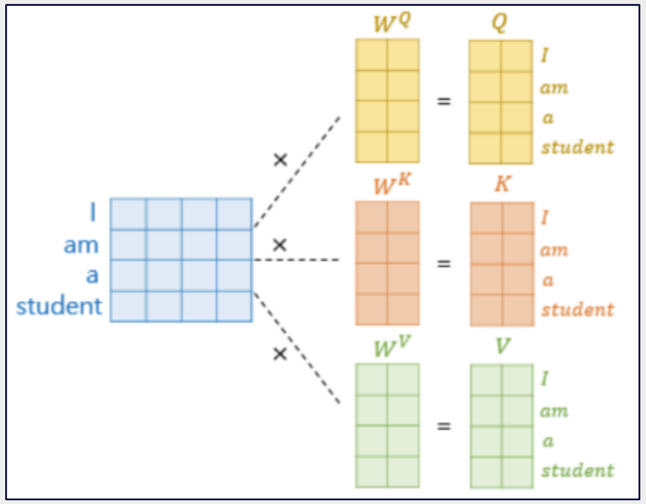
## Self-Attention




해당 부분은 기존 Attention과는 달리 입력 정보가 혼합되어 입력되지 않는다!

하나의 입력만으로 어떻게 3개의 입력으로 나눌 수 있을까?

→ 하나의 행렬에 서로 다른 가중치 합을 통해 3개로 나누자!

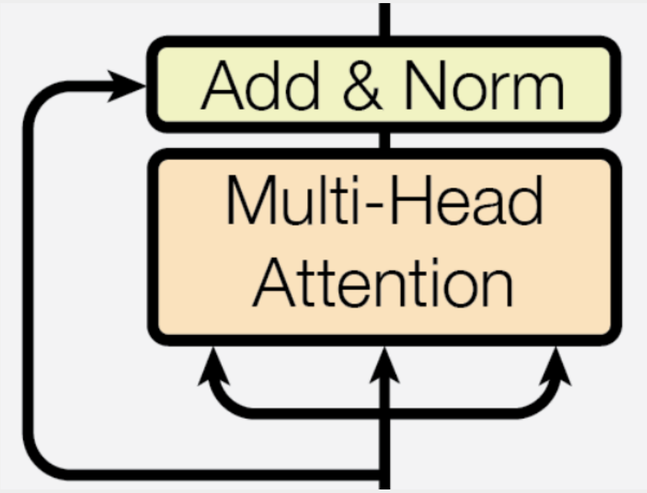



스스로의 입력만으로 Attention을 계산

→ Self-Attention!

# Supplementary Information

## skip-Connection + Add & Norm




Multi-Head Attention의 출력 형상을 조정하는 이유는?

- Skip-Connection이 적용되기 때문
- Skip-Connection을 적용하기 위해서는 두 행렬의 형상이 동일해야 함

Add : Skip-Connection된 행렬 + Multi-Head Attention의 출력 행렬

Norm : Add한 행렬을 정규화(Normalization)

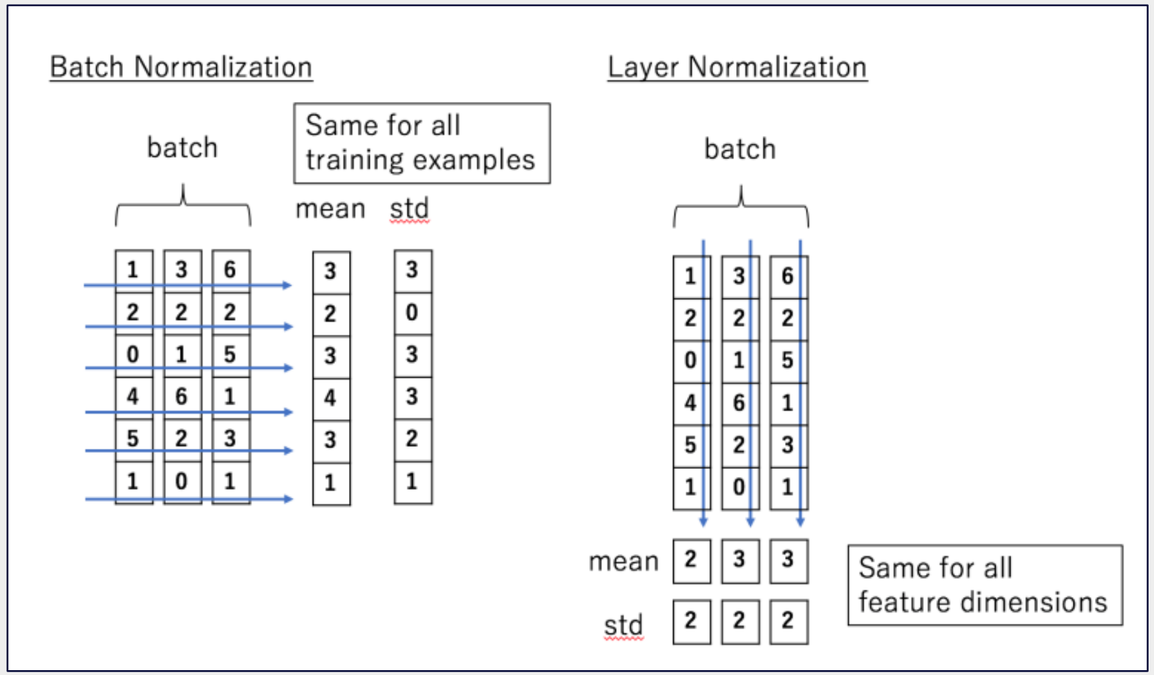
→ 정규화 기법 : Layer Normalization

Skip-Connection과 정규화를 하는 이유는?

→ 기울기 소실 문제를 해결하기 위함

# Supplementary Information

## Layer Normalization vs Batch Normalization




Batch Normalization

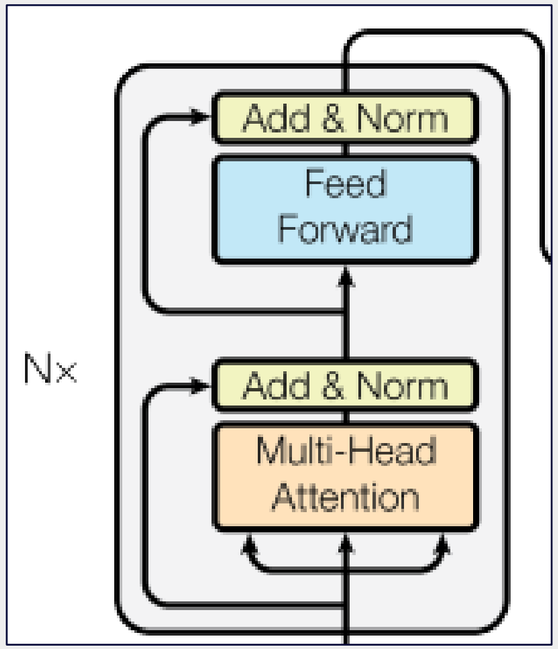
→ 배치마다 평균과 분산을 활용하여 데이터의 분포를 정규화

Layer Normalization

→ 입력 데이터마다 평균과 분산을 활용하여 데이터의 분포를 정규화

# Supplementary Information

## Transformer Encoder



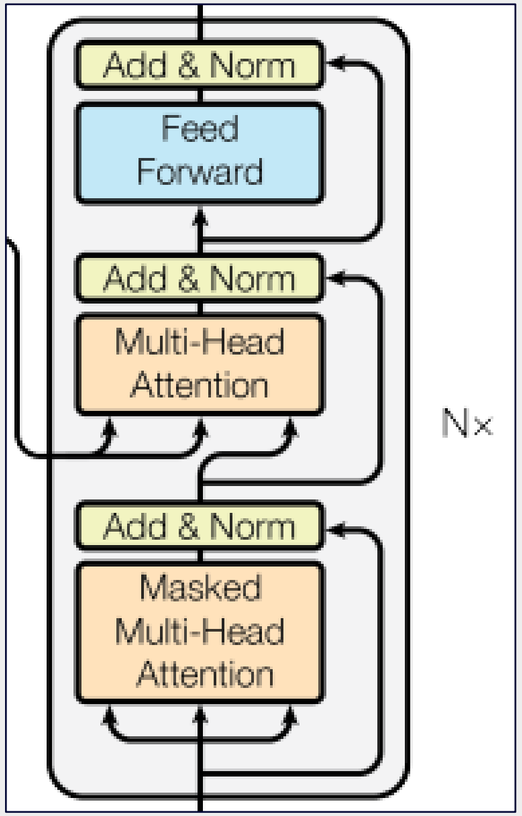

Transformer Encoder 과정

1. 입력된 행렬을 Query, Key, Value 3개로 나눈다.
2. Multi-Head Attention 계층을 거친다.
3. Skip-Connection이 적용된 행렬과 Add & Norm을 수행한다.
4. Feed Forward 계층을 거친다.
5. Skip-Connection이 적용된 행렬과 Add & Norm을 수행한다.
6. 해당 행렬로 다시 1번 과정을 수행한다.

위 과정을 총 N번 수행한다. (논문에서는 이를 총 6번 수행)

# Supplementary Information

## Transformer Decoder



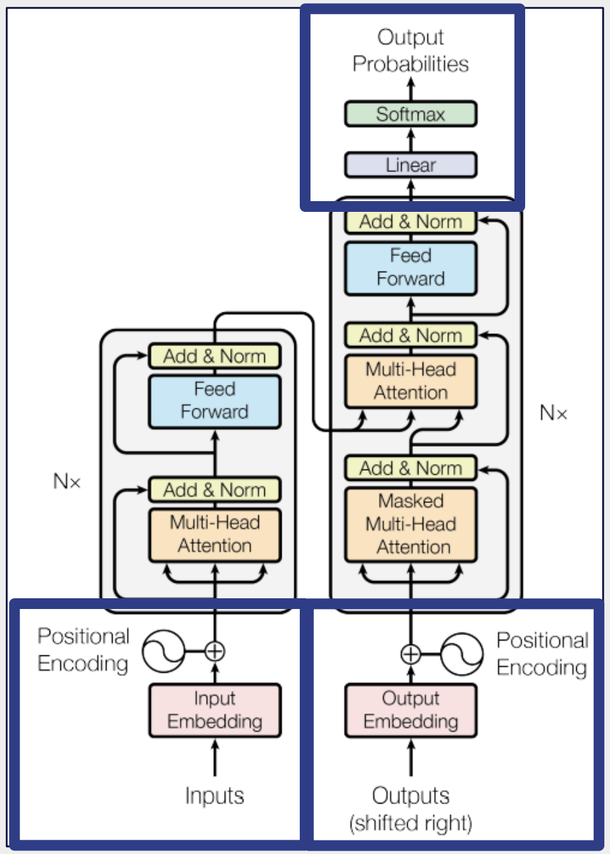

Transformer Decoder 과정

1. 입력된 행렬을 Query, Key, Value 3개로 나눈다.
2. Masked Multi-Head Attention 계층을 거친다.
3. Skip-Connection이 적용된 행렬과 Add & Norm을 수행한다.
4. Encoder로부터 입력된 행렬을 Key, Value, 아래에서 입력된 행렬을 Query로 사용한다.
5. Multi-Head Attention 계층을 거친다.
6. Skip-Connection이 적용된 행렬과 Add & Norm을 수행한다.
7. Feed Forward 계층을 거친다.
8. Skip-Connection이 적용된 행렬과 Add & Norm을 수행한다.
9. 해당 행렬로 다시 1번 과정을 수행한다.

위 과정을 총 N번 수행한다. (논문에서는 이를 총 6번 수행)

# Supplementary Information

## Transformer 최초 입력 & 최종 출력




Transformer의 최종 출력

→ 기존의 모델들과 마찬가지로 Decoder의 출력을 softmax 함수를 통해 확률로 변환하여 각 단어의 확률을 출력함

Encoder & Decoder의 입력

- Embedding 과정을 거쳐 단어가
- 이때 사용되는 Embedding은 Learned Embedding

Learned Embedding 이란?

→ Embedding 값을 고정해두는 것이 아닌, 학습 과정에서 가중치를 업데이트 하듯이 계속하여 갱신해 나가는 방식

이때, 처음보는 것이 등장함 → Positional Encoding이란 무엇일까?

# THANK YOU

## Deep Session 9차시

빅데이터 분석 학회 D&A