8. 어텐션

seq2seq 란, 2개의 RNN 을 연결하여 하나의 시계열 데이터를 다른 시계열 데이터로 변환하는 것이다.

이번 장에서는, seq2seq 와 RNN 의 가능성을 높여주는,

어탠션 이라는 기술에 대해 학습해보자.

seq2seq의 문제점은

어텐션 매커니즘은 seq2seq 를 더 강력하게 해준다.

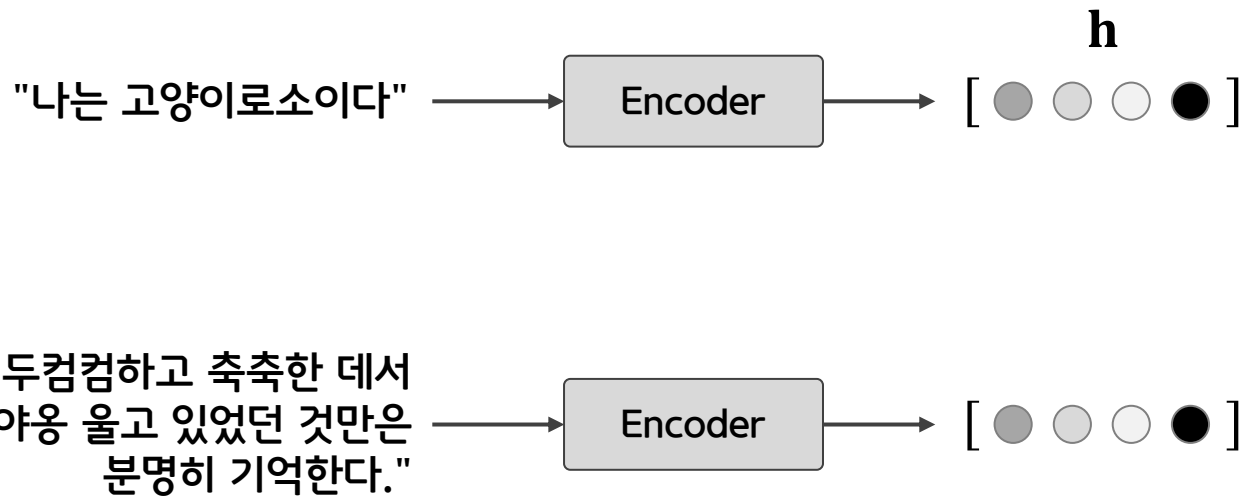
seq2seq가 인간처럼 필요한 정보에만 주목할 수 있게 해주고,
기존의 seq2seq 가 갖고있던 문제점도 해결가능하다.

seq2seq의 문제점은 무엇일까?

seq2seq 에서 encoder 가 시계열데이터를 인코딩하고, 그 인코딩된 정보를 decoder로 전달하는데,
이때 encoder 의 출력은 '고정 길이의 벡터'였다. 이 '고정 길이'에 문제점이 잠재해 있는 것이다.
아무리 입력 문장의 길이가 길다 하더라도 같은 길이의 벡터로 변환한다는 뜻이다.
이는 결국, 필요한 정보가 벡터에 다 담기지 못하는 문제가 생긴다.
그래서 우선 encoder를 개선하고 이어서 decoder 도 개선해야 한다.

seq2seq의 문제점은

입력 문장의 길이에 관계없이, Encoder는 정보를 고정 길이의 벡터로 밀어 넣는다.



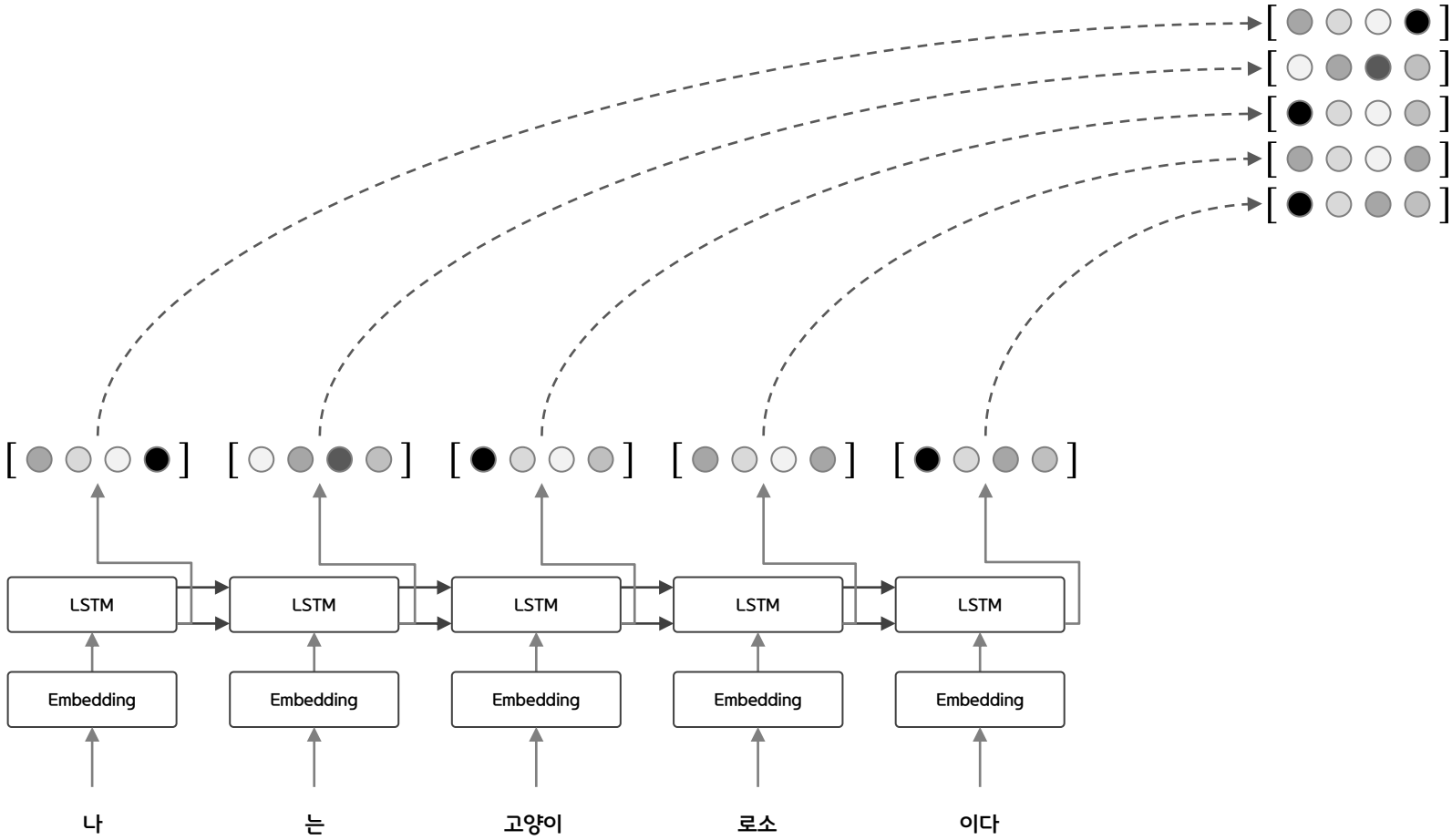
Encoder 개선

우선, encoder를 개선해보자.

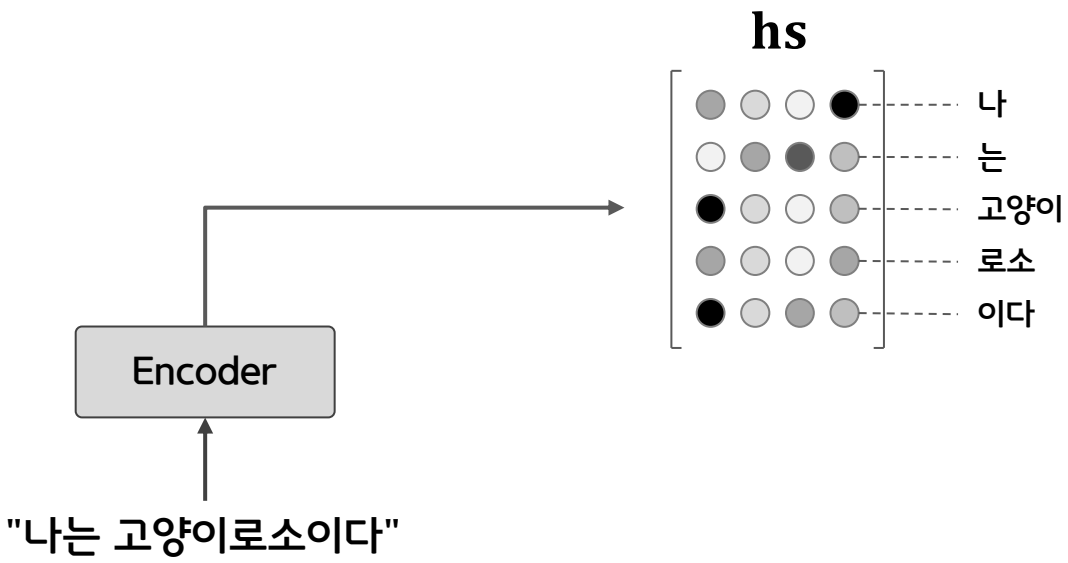
개선 포인트는, encoder 출력의 길이를 입력 문장의 길이에 따라 바꿔주는 것이다.
그러기 위해서, 시각 별(=단어 별) LSTM 계층의 은닉 상태 벡터를 모두 이용하는 것이다.
예를 들어, 5개 단어가 입력되었으면, encoder는 5개의 벡터를 출력한다.

그러면 각 시각별 LSTM 계층의 은닉 상태에는 어떤 정보가 담겨있을까?
직전에 입력된 단어에 대한 정보가 많이 포함되어 있을 것이다.
따라서, encoder가 출력하는 hs 행렬은 각 단어에 해당하는 벡터들의 집합일 것이다.

Encoder의 시각별(단어별) LSTM 계층의 은닉 상태를 모두 이용(hs로 표기)



Encoder의 출력 hs는 단어 수만큼의 벡터를 포함하며, 각각의 벡터는 해당 단어에 대한 정보를 많이 포함한다.



Decoder 개선 ①

이어서 decoder를 개선해보자.

encoder 가 각 단어에 대응하는 LSTM 계층의 은닉 상태 벡터를 hs로 모아 출력하면,
이 hs는 decoder에 전달되어 시계열 변환이 이루어진다.

개선포인트는, decoder가 encoder 의 LSTM 계층의 마지막 은닉상태만을
이용하는 것이 아닌, hs를 전부 활용할 수 있도록 만드는 것이다.

그 전에 인간이 문장을 번역할 때 머릿속에서 어떤 일이 일어날까 생각해보자.

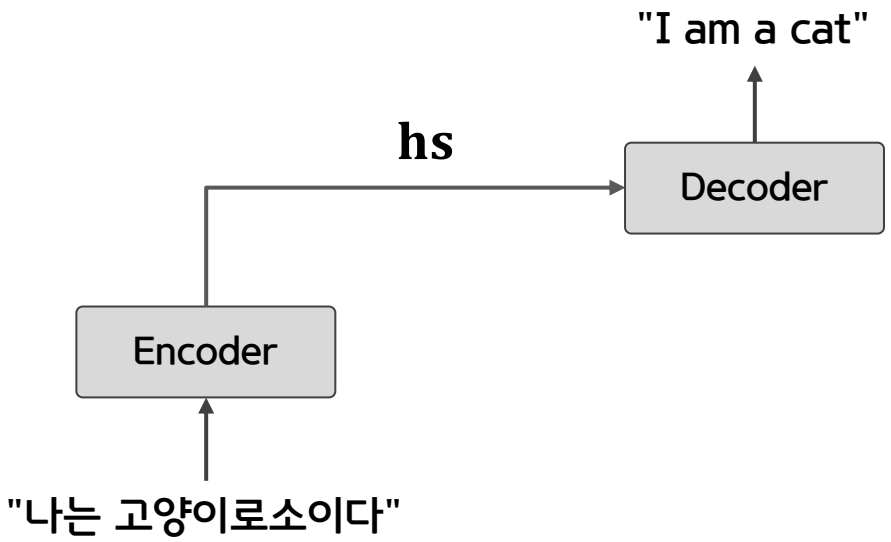
우리는 '어떤 단어'에 주목하여 그 단어의 변환을 수시로 하게 된다.
이를 재현해야 한다.

다시 말하면, 입력과 출력의 여러 단어 중 어떤 단어끼리 서로 관련되어 있는가를 학습시켜야 한다.

즉, 필요한 정보에만 주목하여 그 정보로부터 시계열 변환을 수행해야 한다. 이 구조를 어텐션 이라고 부른다.

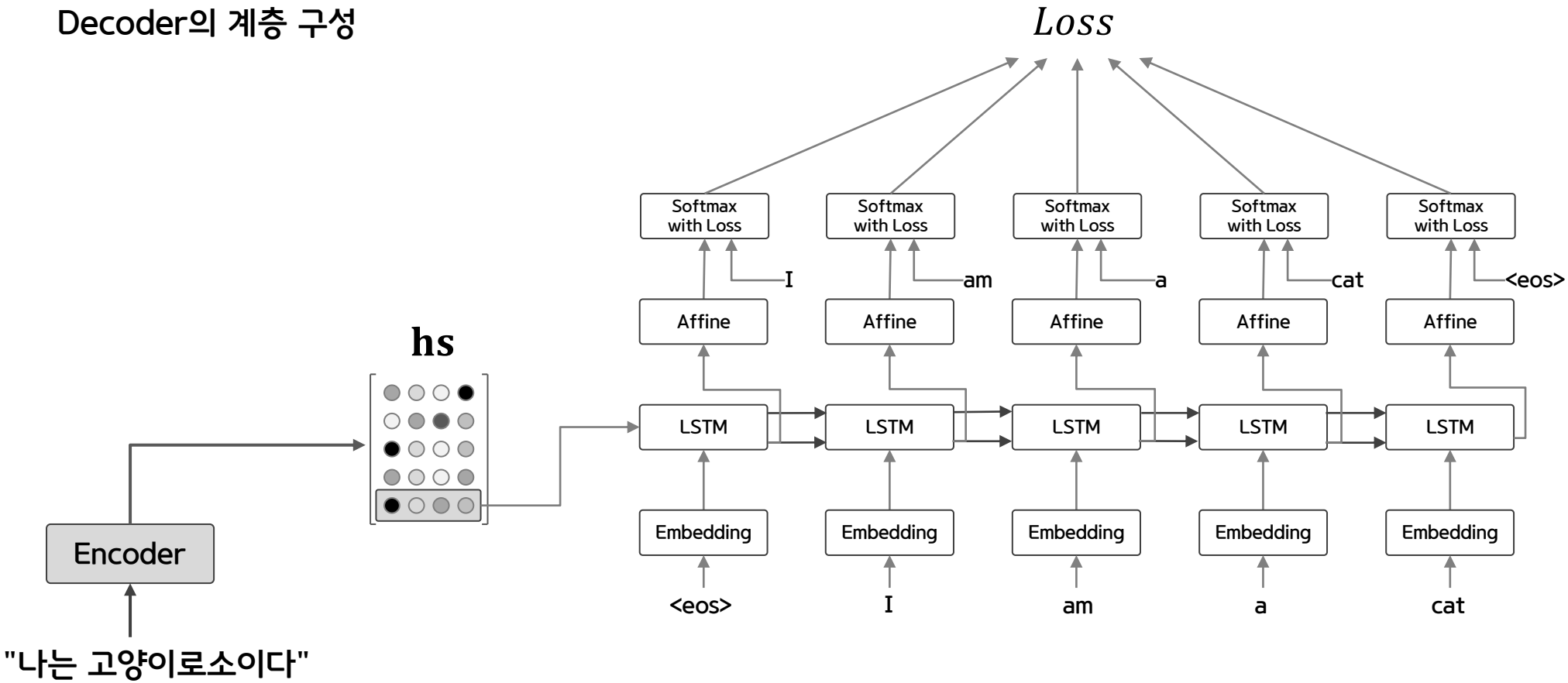
Decoder 개선 ①

Encoder와 Decoder의 관계

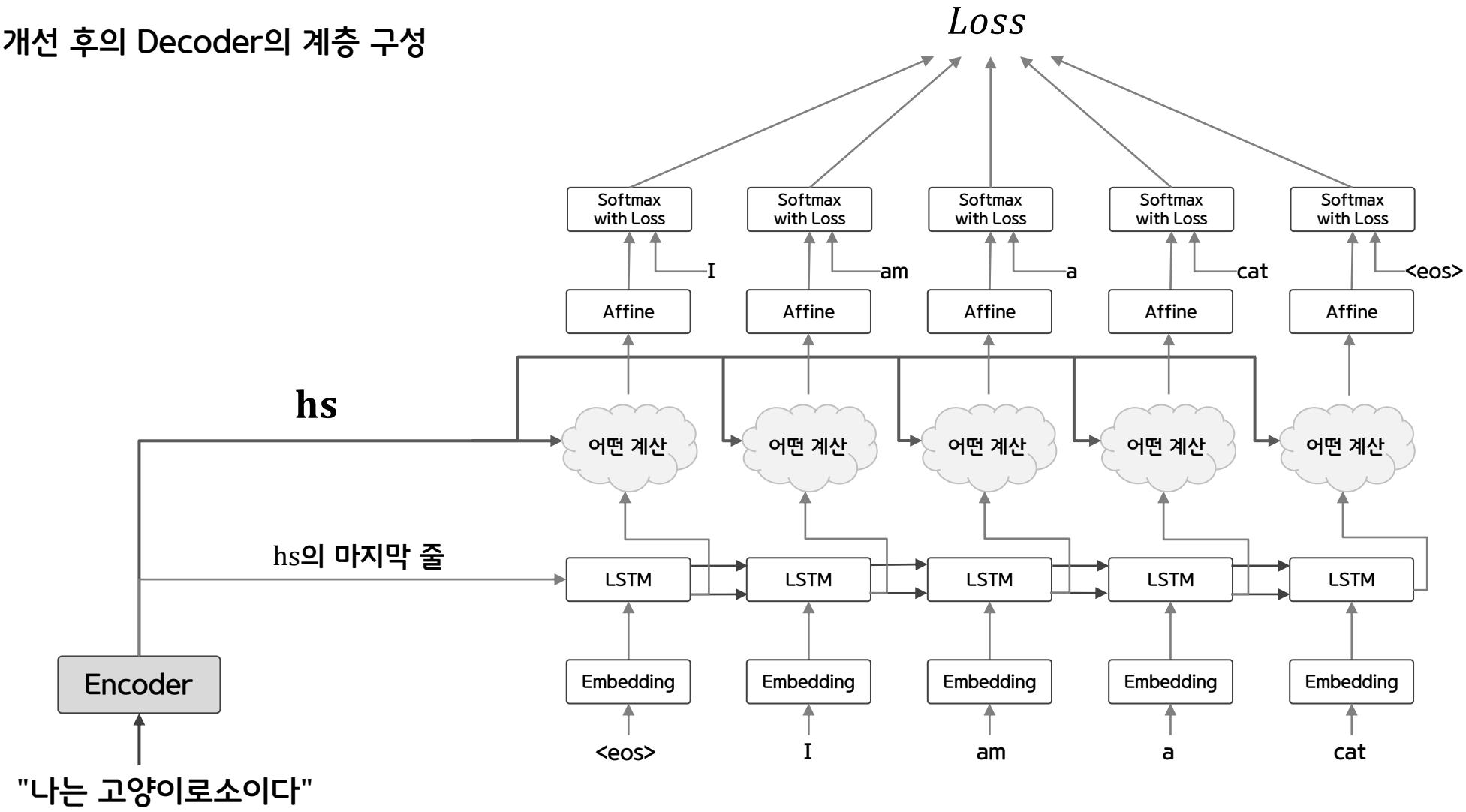


Decoder 개선 ①

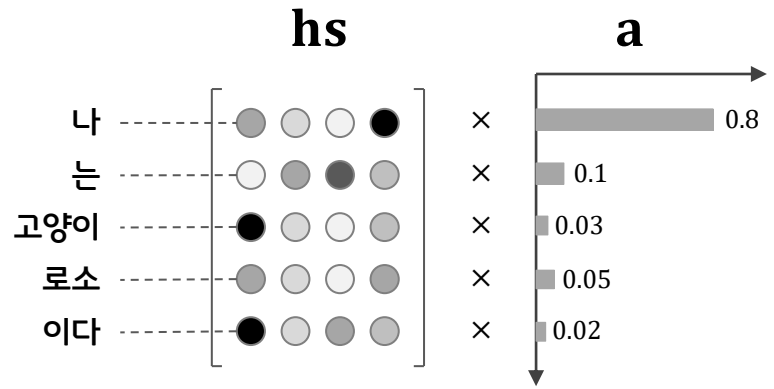
Decoder의 계층 구성



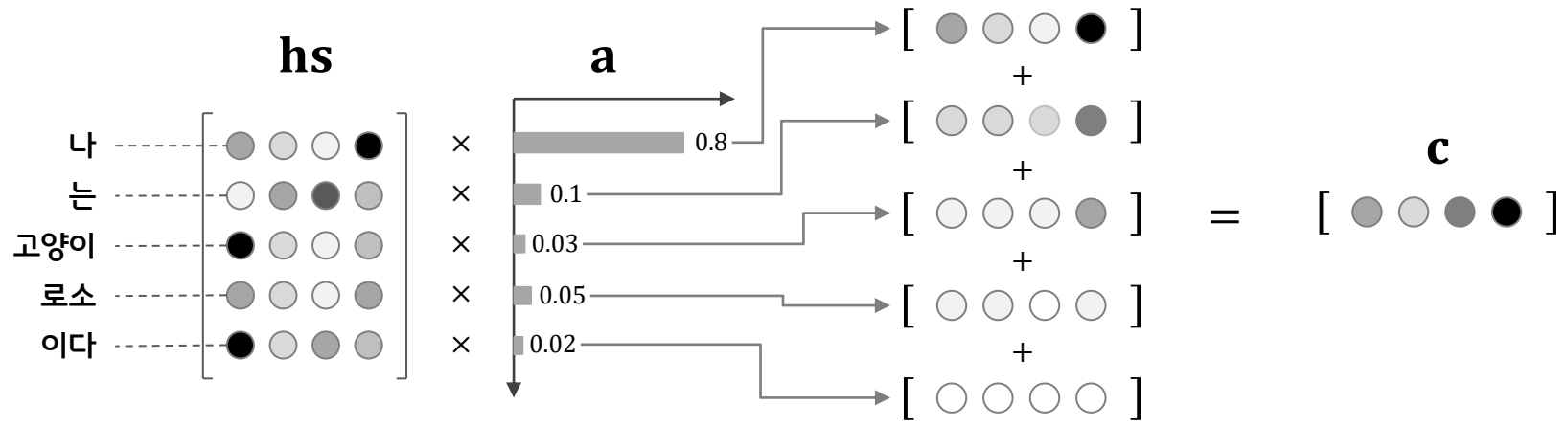
개선 후의 Decoder의 계층 구성



각 단어에 대해서 그것이 얼마나 중요한지를 나타내는 '가중치'를 구한다.



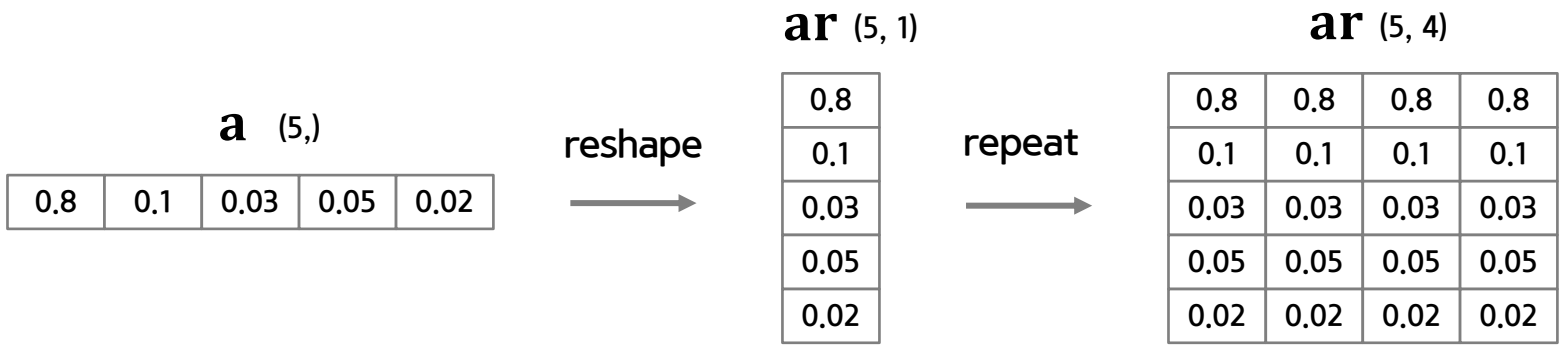
가중합을 계산하여 '맥락 벡터'를 구한다.



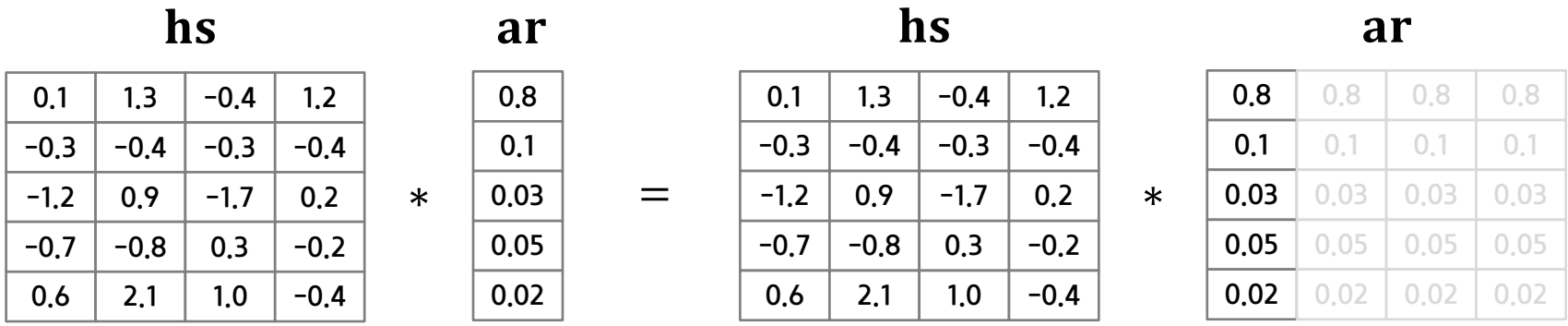
Decoder 개선 ①

※ 코드 참조

reshape()와 repeat() 메서드를 거쳐 a로부터 ar을 생성(변수명 오른쪽에 형상을 표기)

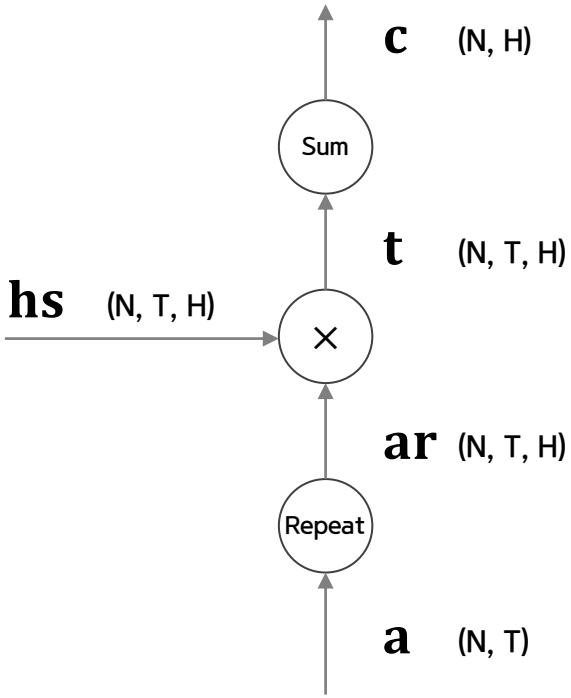


넘파이의 브로드캐스트



※ 코드 참조

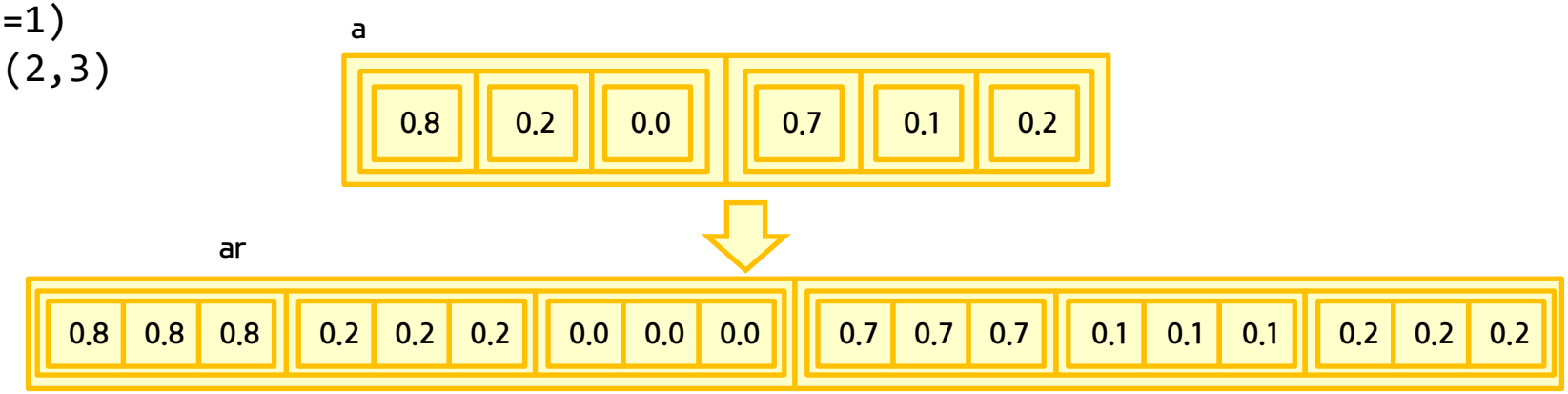
가중합의 계산 그래프



※ 코드 참조

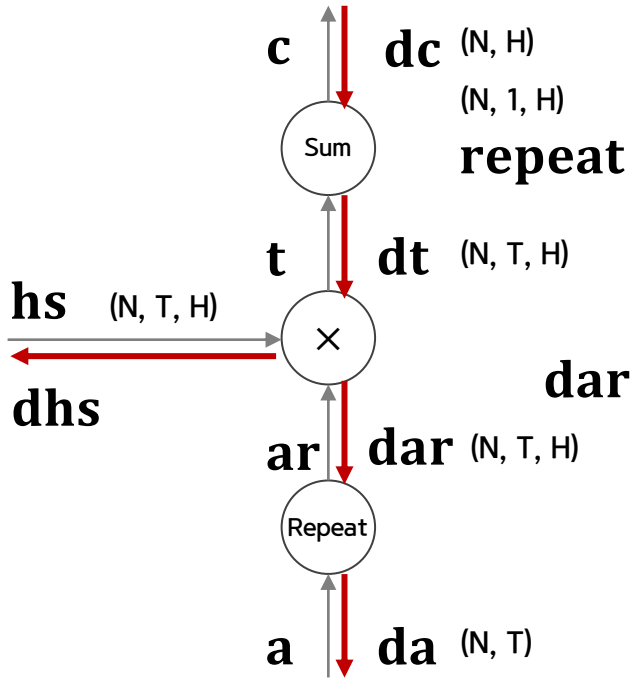
reshape()와 repeat() 메서드를 거쳐 a로부터 ar을 생성(변수명 오른쪽에 형상을 표기)

```
import numpy as np

N, T, H = 2, 3, 3
hs = np.random.randn(N, T, H)
a  = np.random.randn(N, T)

ar = a.reshape(N, T, 1).repeat(H, axis=2)

t = hs * ar
print(t.shape)

c = np.sum(t, axis=1)
print(c.shape)  # (2,3)
```

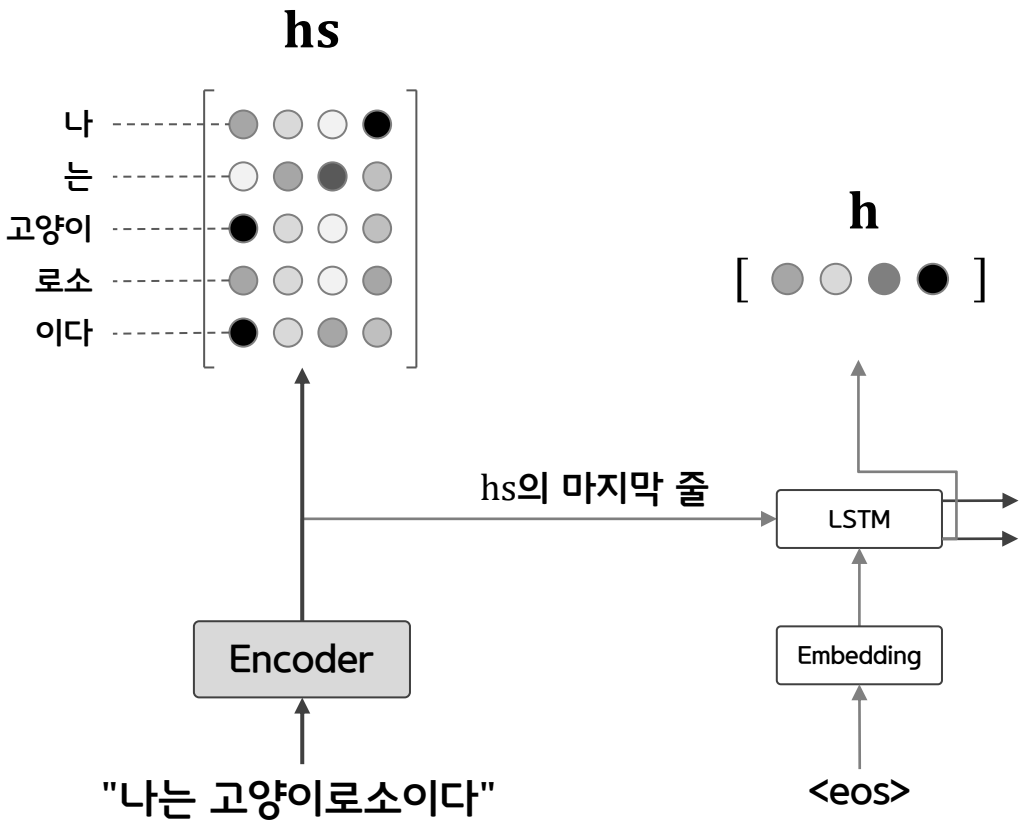


Decoder 개선 ①

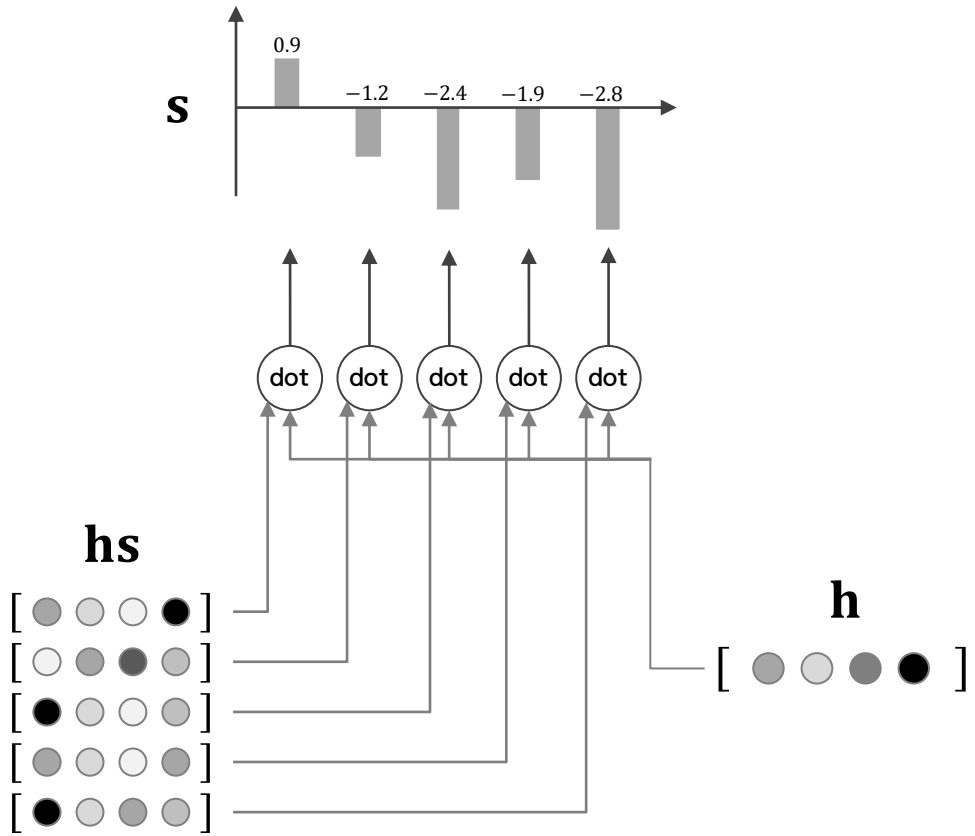
```
def forward(self, hs, a):
        N, T, H = hs.shape

        ar = a.reshape(N, T, 1).repeat(T, axis=1)
        t = hs * ar
        c = np.sum(t, axis=1)

        self.cache = (hs, ar)
        return c

    def backward(self, dc):
        hs, ar = self.cache
        N, T, H = hs.shape
        dt = dc.reshape(N, 1, H).repeat(T, axis=1)
        dar = dt * hs
        dhs = dt * ar
        da = np.sum(dar, axis=2)

        return dhs, da
```



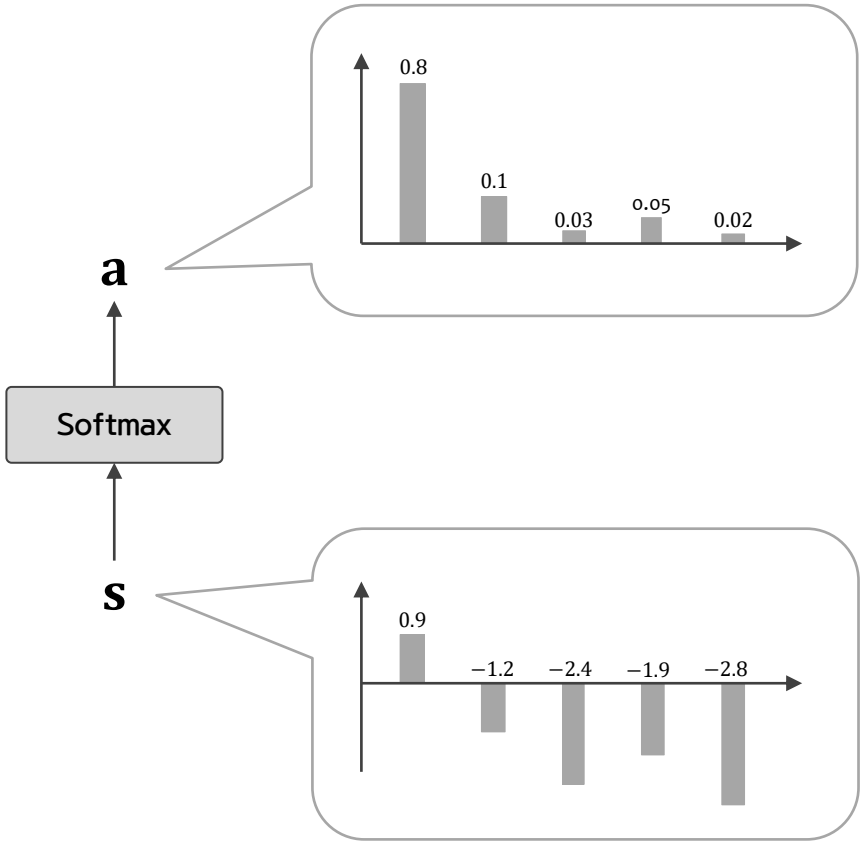
Decoder의 첫 번째 LSTM 계층의 은닉 상태 벡터



내적을 통해 hs의 각 행과 h의 유사도를 산출(내적은 dot 노드로 그림)



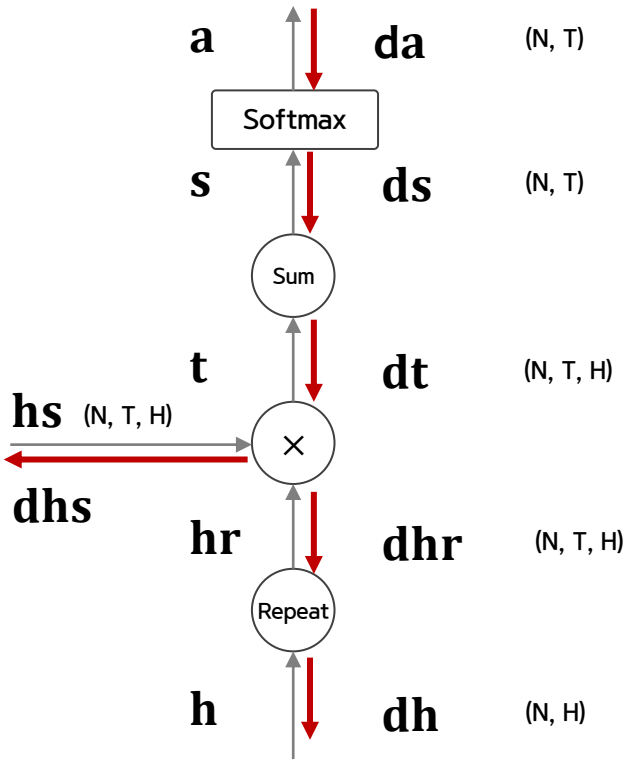
Softmax를 통한 정규화



※ 코드 참조

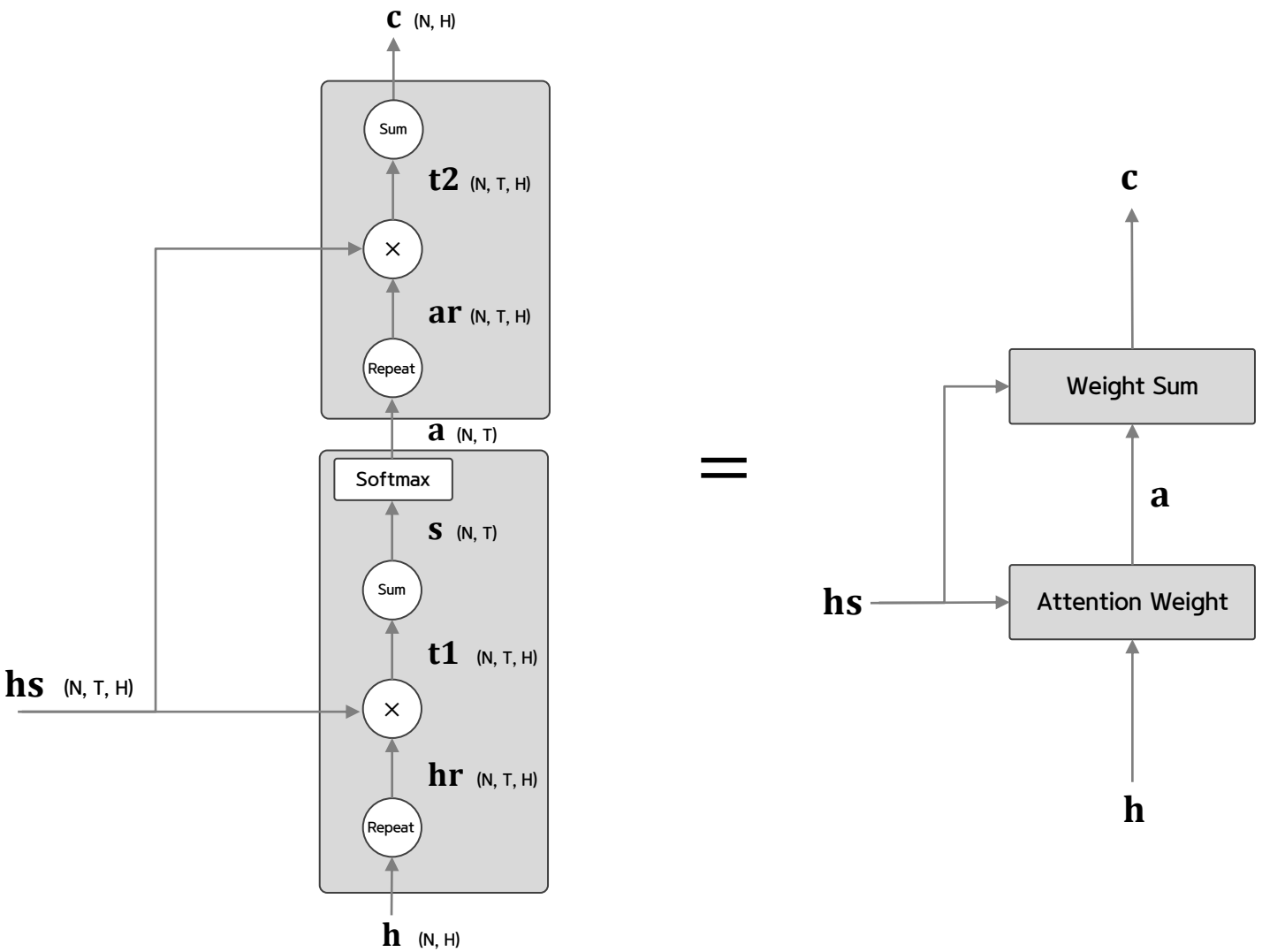
Decoder 개선 ②

```
def forward(self, hs, h):
        N, T, H = hs.shape

        hr = h.reshape(N, 1, H).repeat(T, axis=1)
        t = hs * hr
        s = np.sum(t, axis=2)
        a = self.softmax.forward(s)

        self.cache = (hs, hr)
        return a

    def backward(self, da):
        hs, hr = self.cache
        N, T, H = hs.shape

        ds = self.softmax.backward(da)
        dt = ds.reshape(N, T, 1).repeat(H, axis=2)
        dhs = dt * hr
        dhr = dt * hs
        dh = np.sum(dhr, axis=1)

        return dhs, dh
```

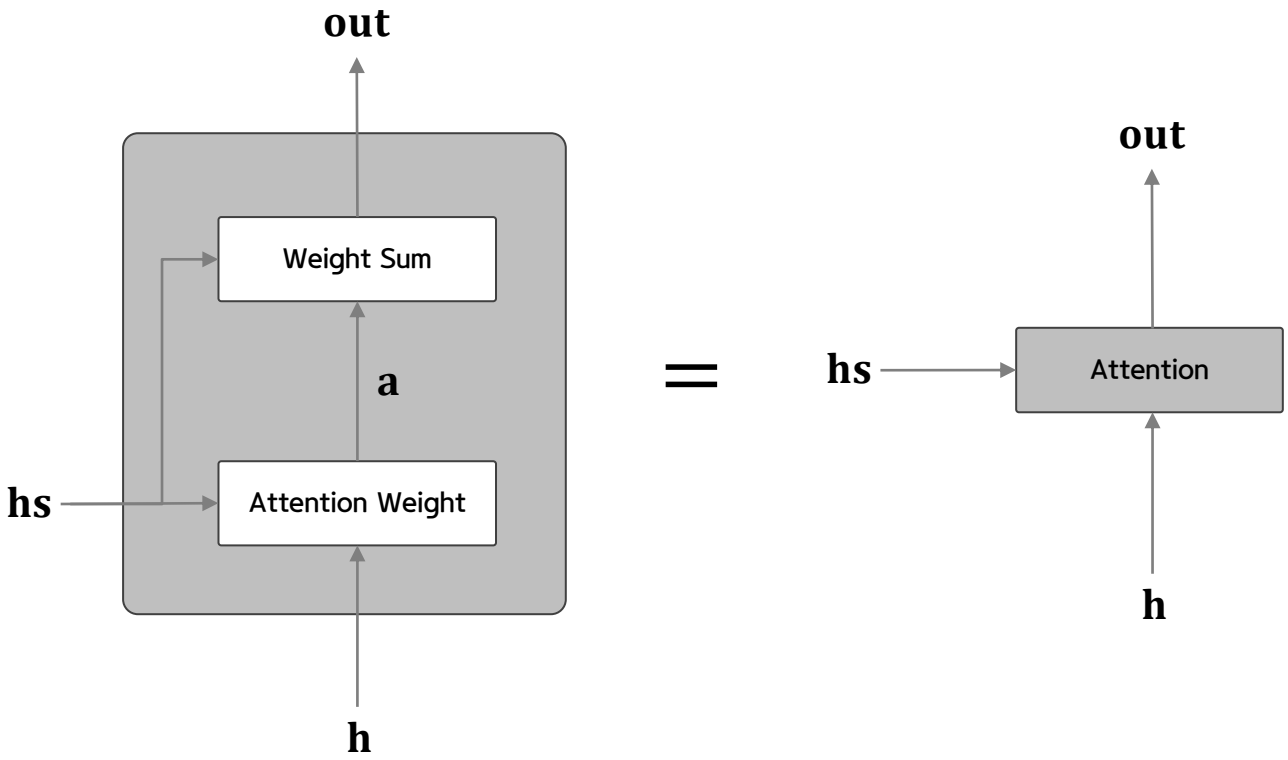


Decoder 개선 ③

맥락 벡터를 계산하는 계산 그래프



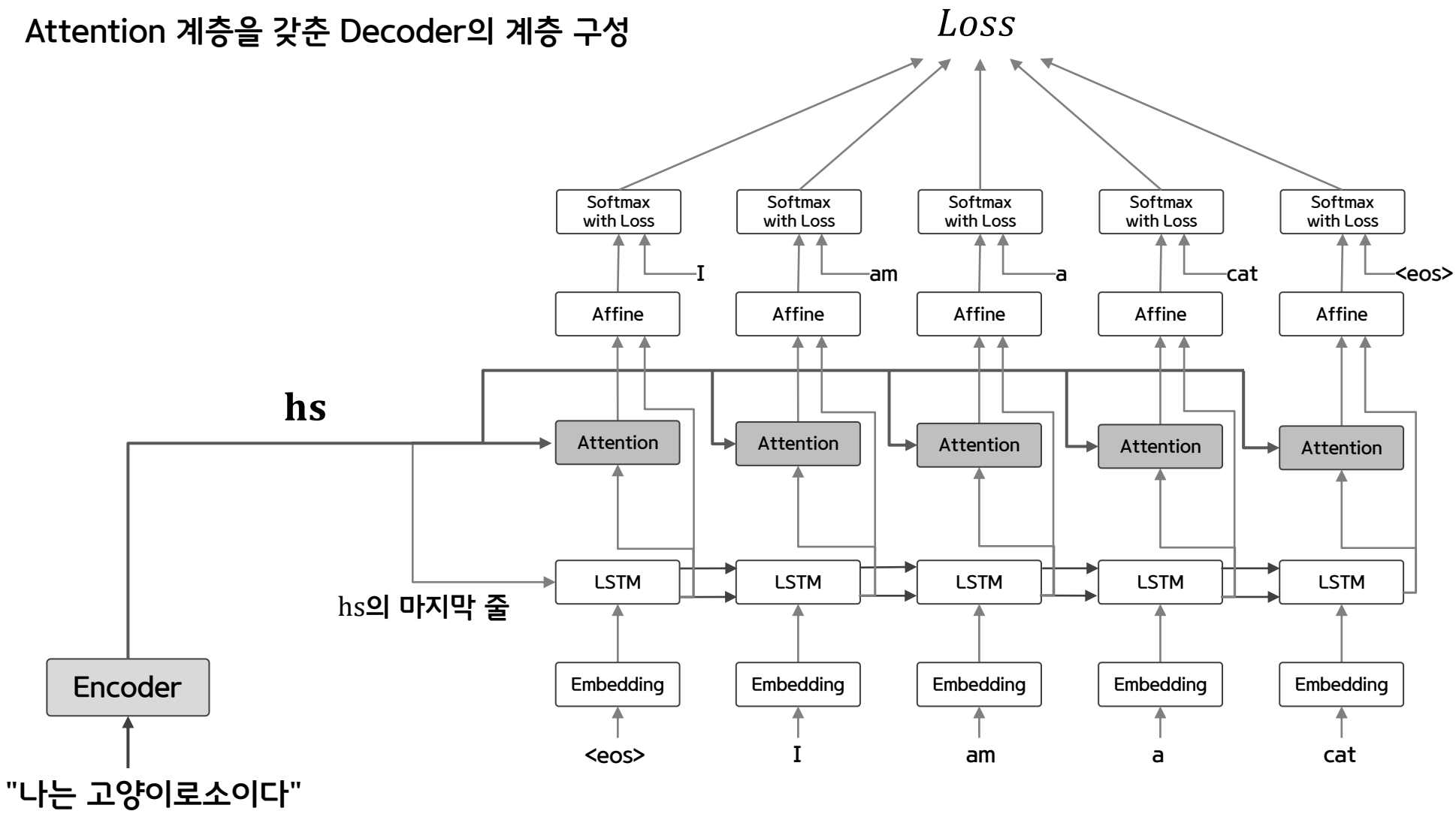
왼쪽 계산 그래프를 Attention 계층으로 정리



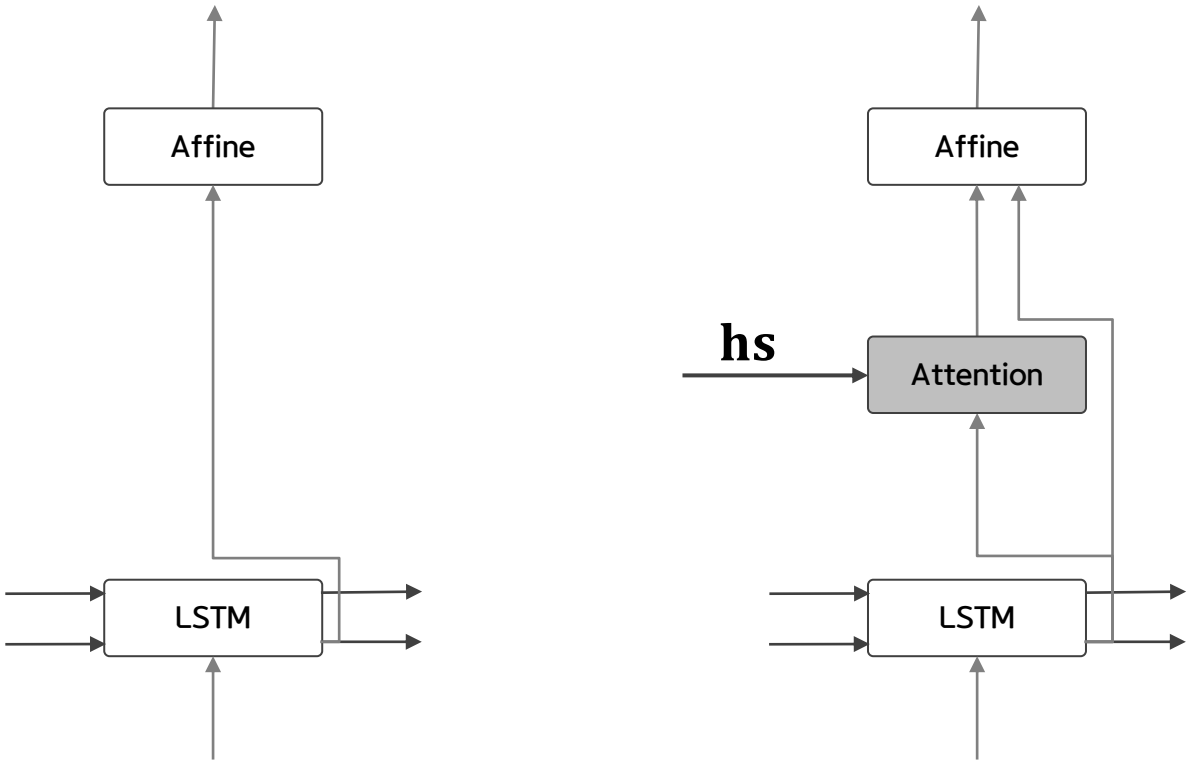
※ 코드 참조

Decoder 개선 ③

Attention 계층을 갖춘 Decoder의 계층 구성

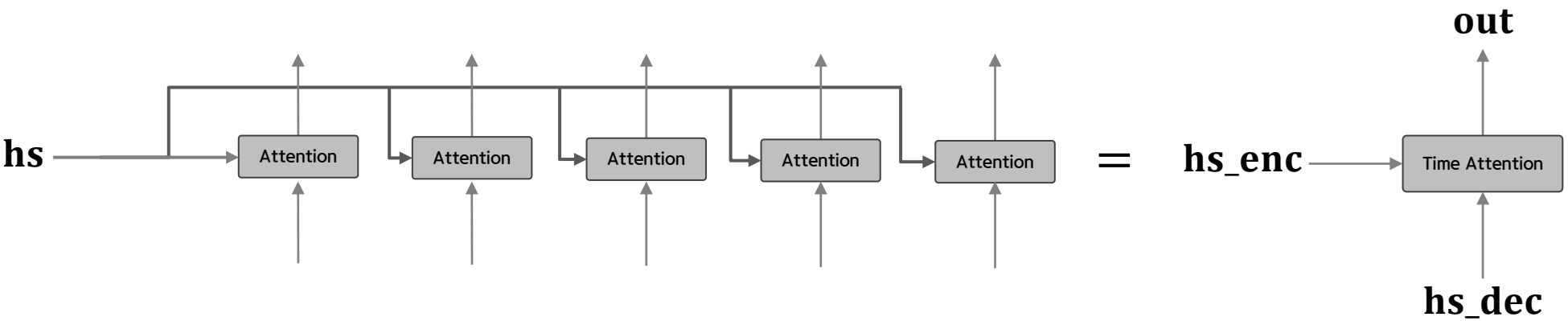


앞 장의 Decoder(왼쪽)와 Attention이 추가된 Decoder(오른쪽) 비교: LSTM 계층에서 Affine 계층 까지만 그림



Decoder 개선 ③

다수의 Attention 계층을 Time Attention 계층으로서 모아 구현



※ 코드 참조

8. 어텐션

8.1 어텐션의 구조

8.2 어텐션을 갖춘 seq2seq 구현

8.3 어텐션 평가

8.4 어텐션에 관한 남은 이야기

8.5 어텐션 응용

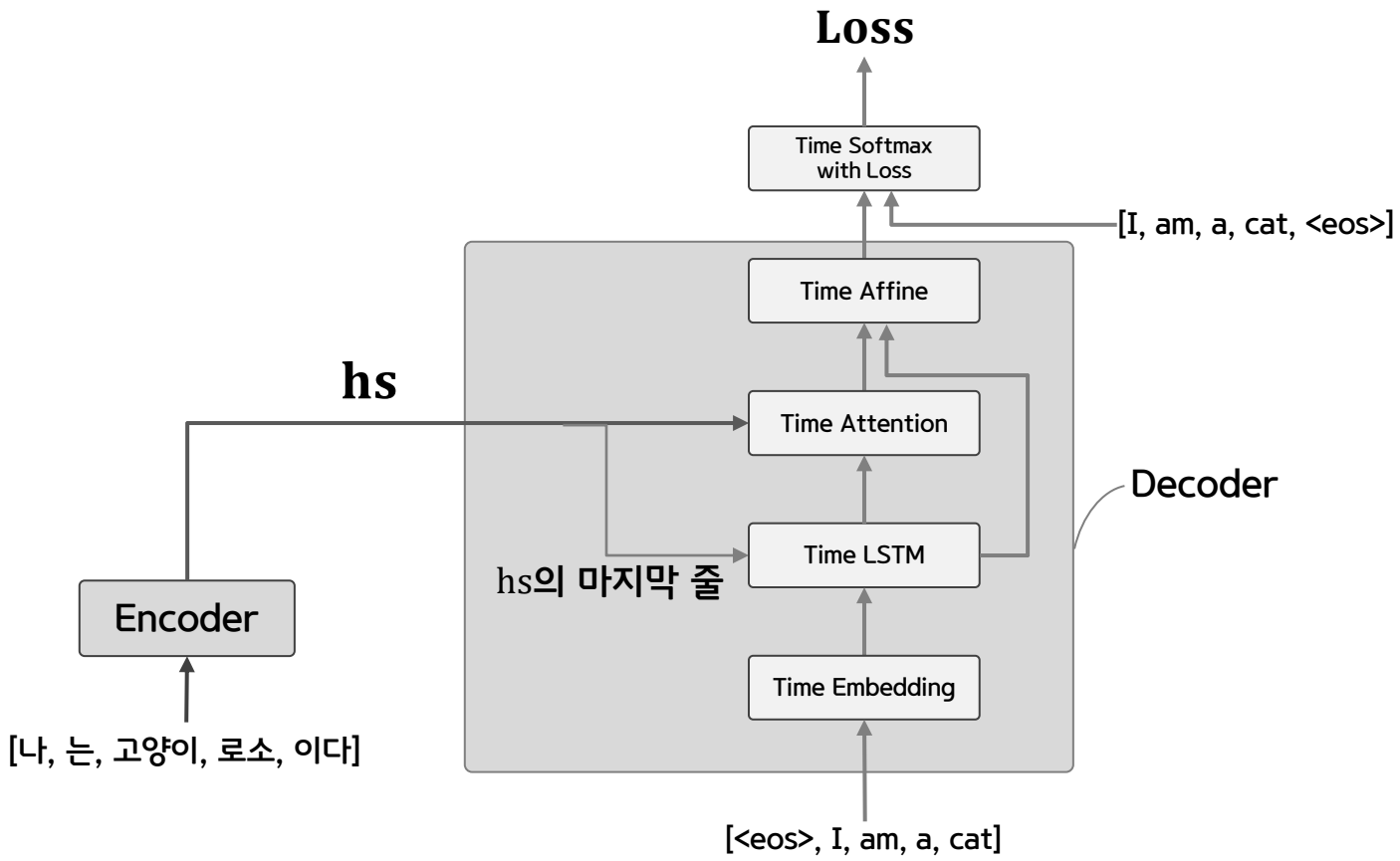
Encoder 구현

forward() 메소드는 LSTM 계층의 모든 은닉 상태 벡터를 반환한다.
인수로 받고있는 Encoder 객체는 앞 장에 LSTM계층의 마지막 은닉 상태 벡터를 갖는데,
여기서 이를 상속받는다.

※ 코드 참조

Decoder 구현

forward() 메서드에서 Time attention 계층의 출력과 LSTM 계층의 출력을 연결한다.
연결시에는 np.concatenate() 사용한다.



※ 코드 참조

8. 어텐션

8.1 어텐션의 구조

8.2 어텐션을 갖춘 seq2seq 구현

8.3 어텐션 평가

8.4 어텐션에 관한 남은 이야기

8.5 어텐션 응용

'날짜 형식'을 변경하는 문제로 어텐션을 갖춘 seq2seq의 효과를 확인해 보았다.
(데이터 크기가 작고, 어느 쪽인가를 맞추는 인위적인 문제)

번역용 데이터셋 중에서는 WMT 가 유명하여 seq2seq의 성능을 평가하는데
자주 이용되지만 크기가 크니(20GB) 부담된다.

자, 예를 들어 "september 27, 1994' 를 '1994-09-27' 같은 표준형식으로 변환해보자.

왜 하필 이문제냐?
이게 그렇게 보기보다 간단하지 않다.
변환 규칙이 나름 복잡하다.

그리고 질문과 답변 즉, 입력과 출력 사이에 알기 쉬운 대응관계가 있기 때문이다.

학습 데이터셋의 형식은 이러하다.

```
september 27, 1994           _1994-09-27
August 19, 2003              _2003-08-19
2/10/93                      _1993-02-10
10/31/90                     _1990-10-31
TUESDAY, SEPTEMBER 25, 1984  _1984-09-25
JUN 17, 2013                 _2013-06-17
april 3, 1996                _1996-04-03
October 24, 1974             _1974-10-24
AUGUST 11, 1986              _1986-08-11
February 16, 2015            _2015-02-16
```

※ 코드 참조

이렇게 학습하면 1에폭부터 빠르게 정답률이 높아져 2에폭째에는 이미 거의 모든 문제를 풀어낸다.

이전 장들에서 다뤘던 단순한 seq2seq,
엿보기를 적용한 seq2seq 모델과 비교했을때 학습 속도 측면에서는 가장 빠르고,

정확도는 엿보기를 적용한 seq2seq과 동등했지만 현실의 시계열 데이터는 복잡하고 길기 때문에
어텐션이 매우 유리하다.

어탠션이 시계열 데이터를 변환할 때 어떤 원소에 주의를 기울이는지
보기 위해 어텐션을 시각 화해보자.

학습이 끝난 AttentionSeq2seq로 날짜 변환을 수행할 때의 어텐션 가중치를 시각화 한다.

가로축은 입력문장, 세로축은 출력문장.

맵의 각 원소는 밝을수록 값이 크다(1.0).

※ 코드 참조

AUGUST 일때 08에 대응하고 있는 것을 볼 수 있다. seq2seq가
August가 8월에 대응한다는 사실을 데이터만 가지고 학습해 낸 것이다.

1983, 26도 1983과 26에 대응하고 있는 것을 볼 수 있다.

어텐션 모델은 인간이 이해할 수 있는 구조나 의미를 제공한다는 점에서 굉장한 의의가 있다.

어텐션을 사용해 단어와 단어의 관련성을 볼 수 있었으므로,

이를 보고 모델의 처리가 인간의 논리를 따르는지 판단이 가능하다.

8. 어텐션

앞에서 단방향 RNN을 보았을 때, LSTM 의 각 시각의 은닉 상태 벡터는 hs로 모아진다.

그리고 Encoder가 출력하는 hs의 각 행에는 그 행에 대응하는 단어의 성분이 많이 포함 되어있다.

글을 왼쪽에서 오른쪽으로 읽기때문에, '나는 고양이로소이다' 라는 문장에서
'고양이'에 대응하는 벡터에는 '나', '는'', '고양이' 까지 총 세 단어의 정보가 인코딩 되어 들어간다.

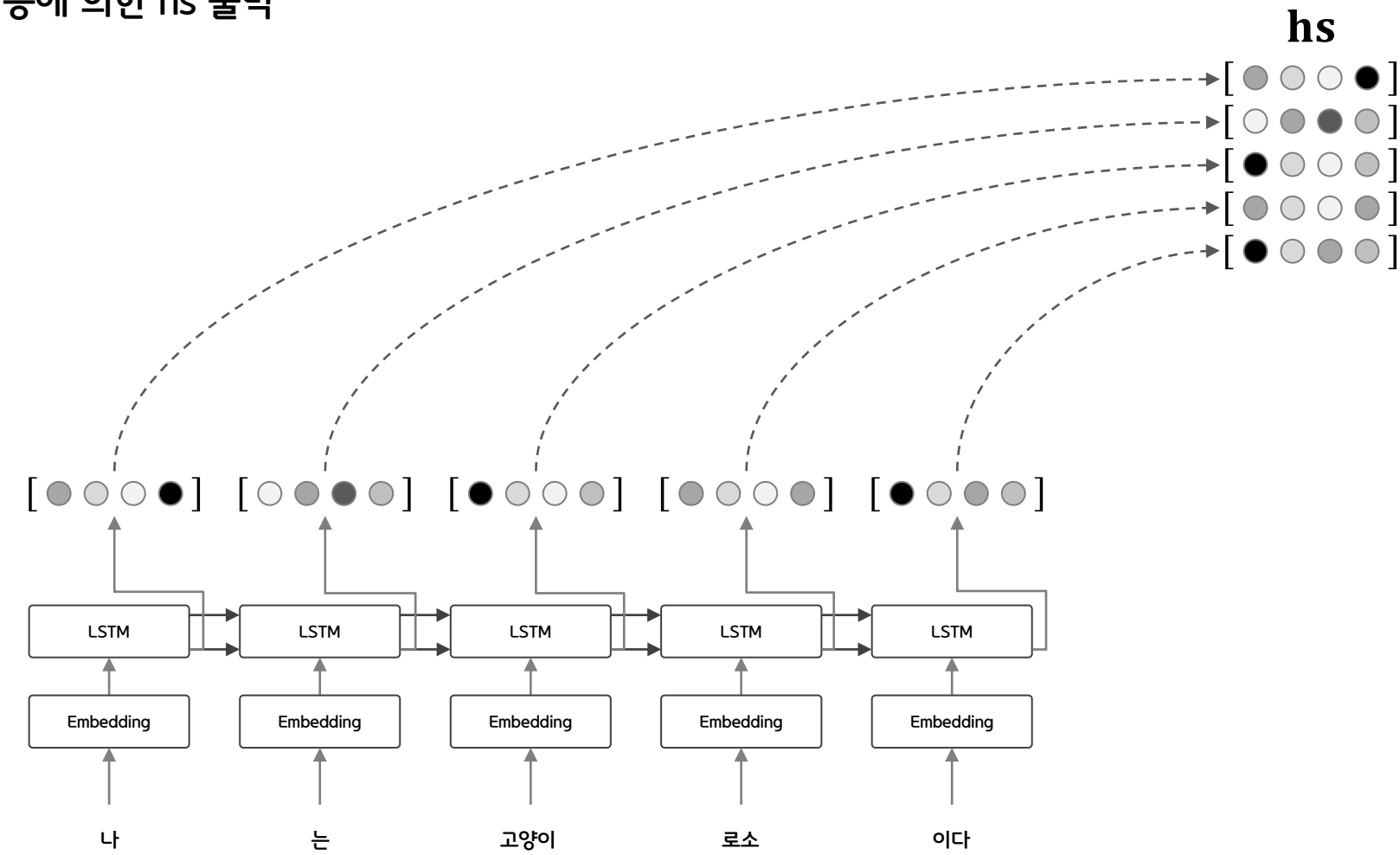
하지만, 이렇게 하지 말고, '고양이'단어의 '주변'정보를 균형 있게 담고 싶을 때
LSTM을 양방향으로 처리하도록 한다. 이것이 양방향 LSTM 이다.

전에 했던 LSTM 계층에, 역방향으로 처리하는 LSTM 계층도 추가한다.

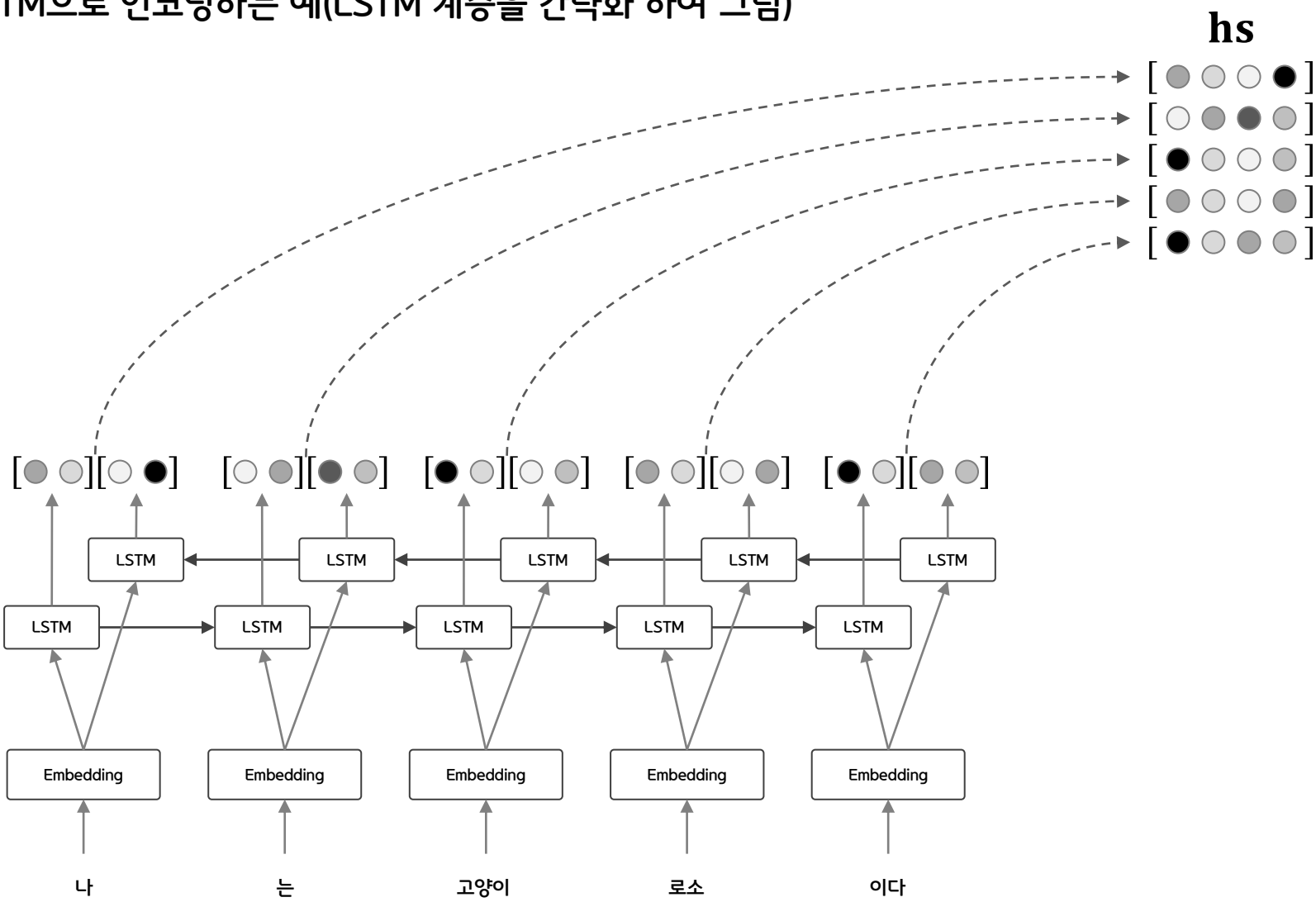
각 시각에서는 이 두 LSTM 계층의 은닉상태를 '연결'시킨 벡터를 최종 은닉 상태로 처리한다.
역방향으로 처리하는 LSTM 계층에 반대순서로 단어를 나열하여 넣는다.

즉, 오른쪽에서 왼쪽으로 처리한다.
이 두 계층을 연결하기만 하면 양방향 LSTM 계층이 나온다.

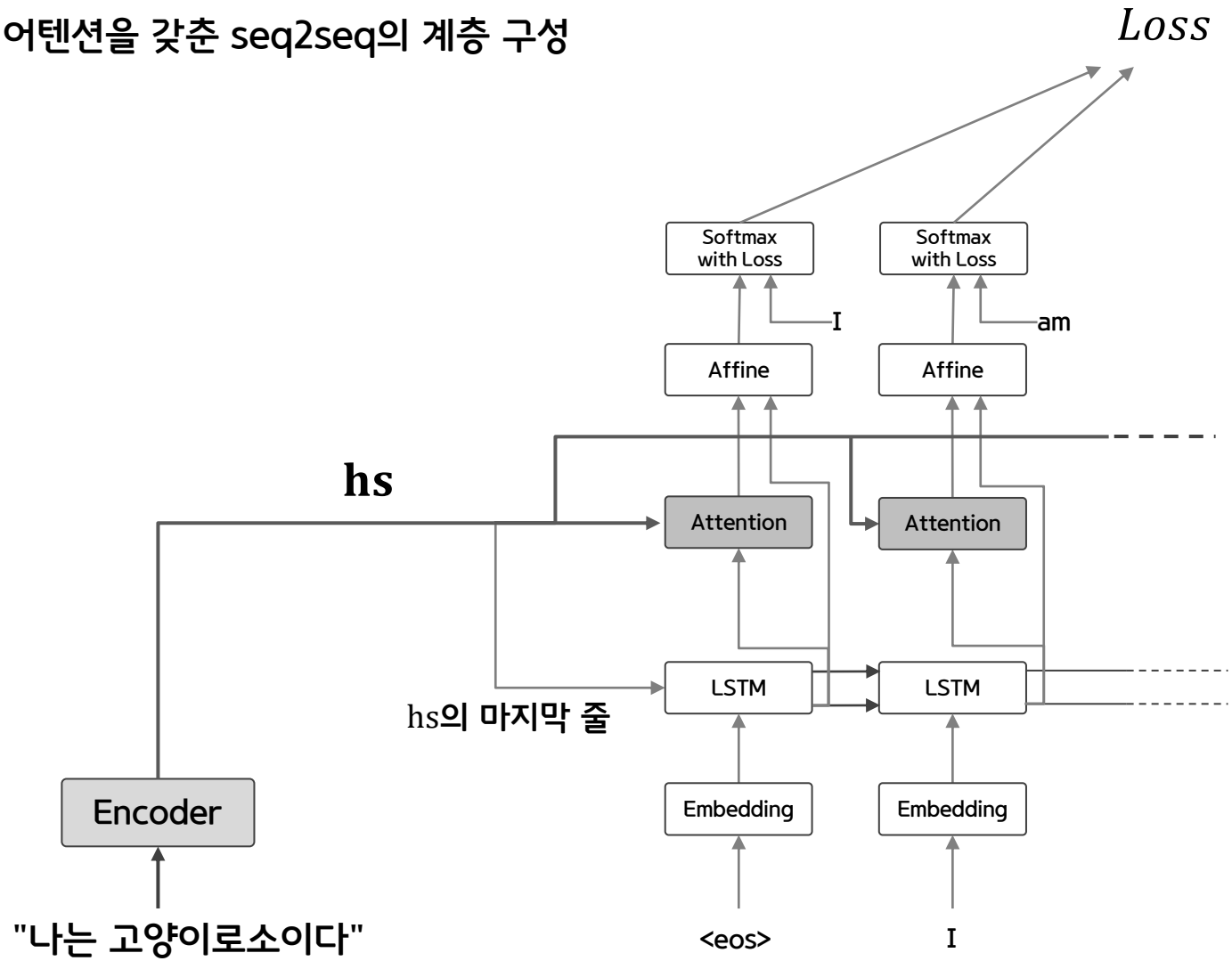
LSTM 계층에 의한 hs 출력



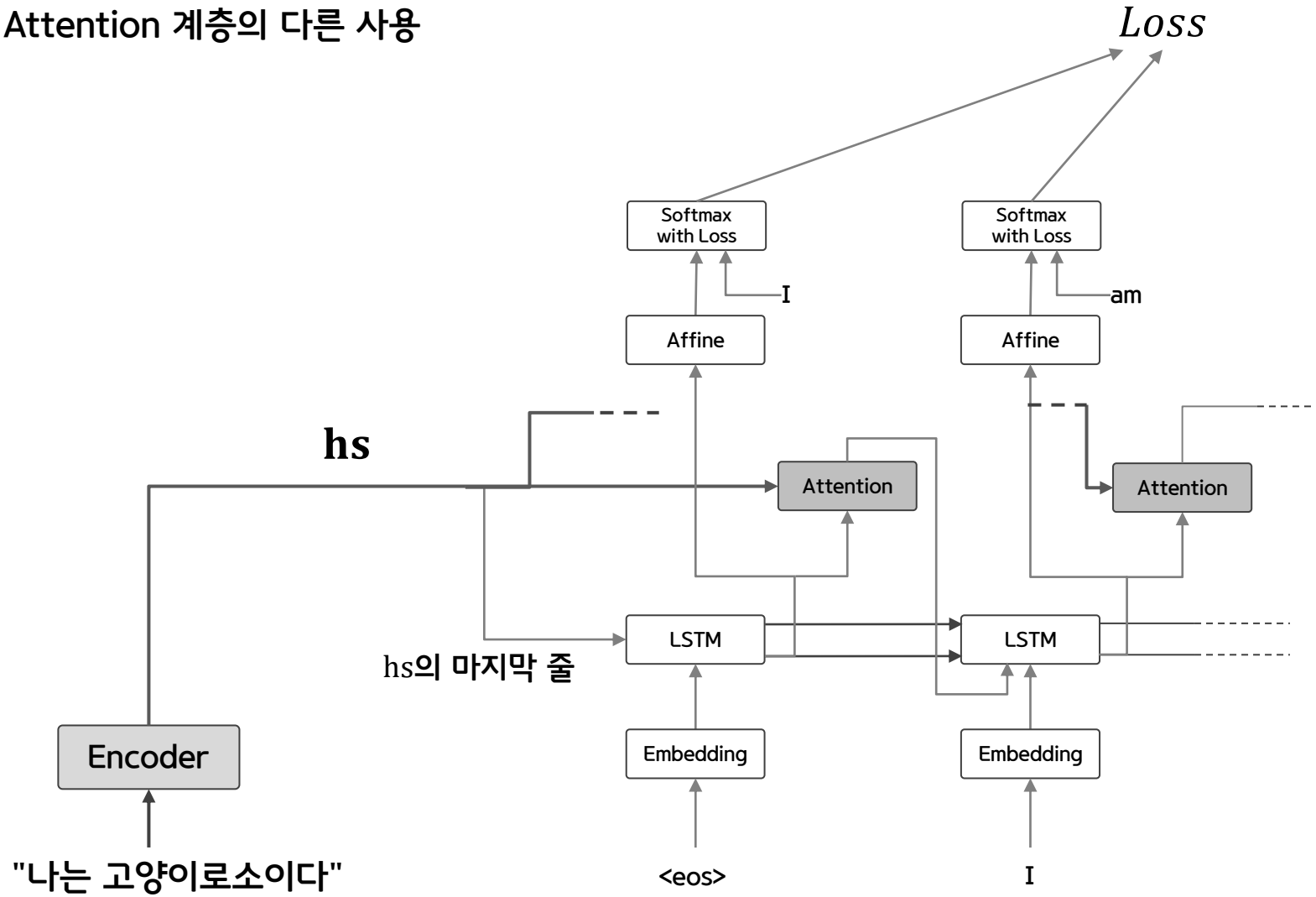
양방향 LSTM으로 인코딩하는 예(LSTM 계층을 간략화 하여 그림)



어텐션을 갖춘 seq2seq의 계층 구성



Attention 계층의 다른 사용



Attention 계층 사용 방법

앞에선 Attention 계층을 LSTM 계층과 Affine 계층 사이에 삽입했지만,

Attention 계층을 이용하는 장소가 정해져 있지는 않다.

LSTM 계층 전에 넣어, LSTM 계층의 입력으로 사용할 수 있다.

뭐가 더 정확도가 높냐는, 실제 데이터에 따라 다르다.

직접 해보기전에는 알 수 없다.

구현 관점에서는 전자의 구성이 쉽다.

seq2seq 심층화와 skip 연결

seq2seq을 심층화할 때 쉽게 떠올릴 수 있는 건 RNN층을 깊게 쌓는 방법이다.
그러면 표현력이 높은 모델을 만들 수 있다.

보통은 Encoder와 Decoder 에서는 같은 층수의 LSTM 계층을 이용하는 것이 일반적인데,
여러가지 변형을 하면서, Decoder의 LSTM 계층의 은닉 상태를 Attention 계층에 입력하고,
Attention 계층의 출력인 맥락벡터를 Decoder의 여러 계층(LSTM 계층과 Affine계층) 으로 전파할 수 있다.

skip 연결은 층을 깊게할 때 사용하는 중요한 기법이다.

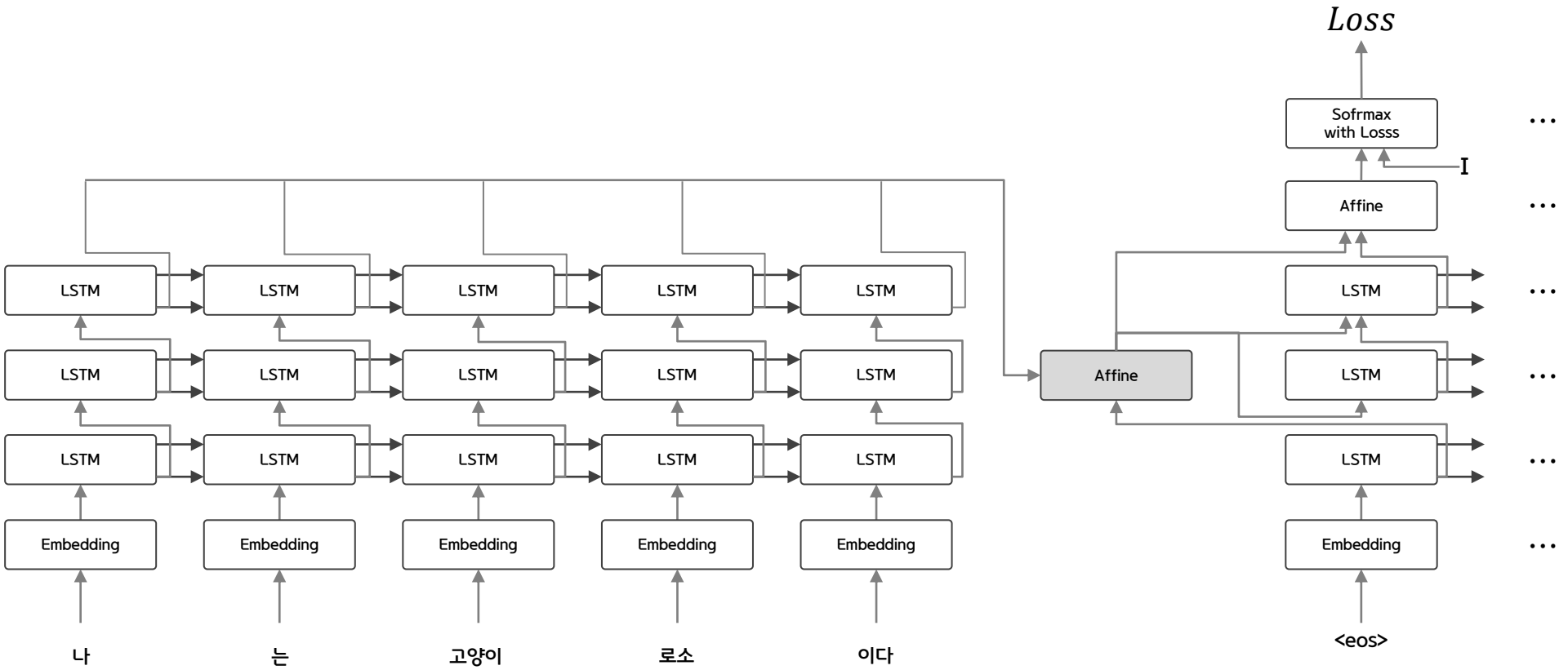
계층을 넘어(=계층을 건너 뛰어) '선을 연결'하는 단순한 기법이다.

이때 skip 연결의 접속부에서는 2개의 출력이 더해진다.

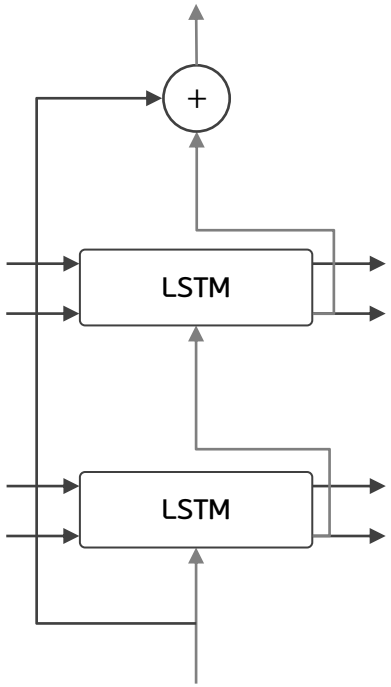
왜냐하면, 덧셈은 역 전파 시 기울기를 그대로 '흘려'보내므로,
skip 연결의 기울기가 아무런 영향을 받지 않고 모든 계층으로 흐르기 때문이다.

따라서 층이 깊어져도 기울기가 소실되지 않고 전파되어 잘 학습된다.

3층 LSTM 계층을 사용한 어텐션을 갖춘 seq2seq



LSTM 계층의 skip 연결 예



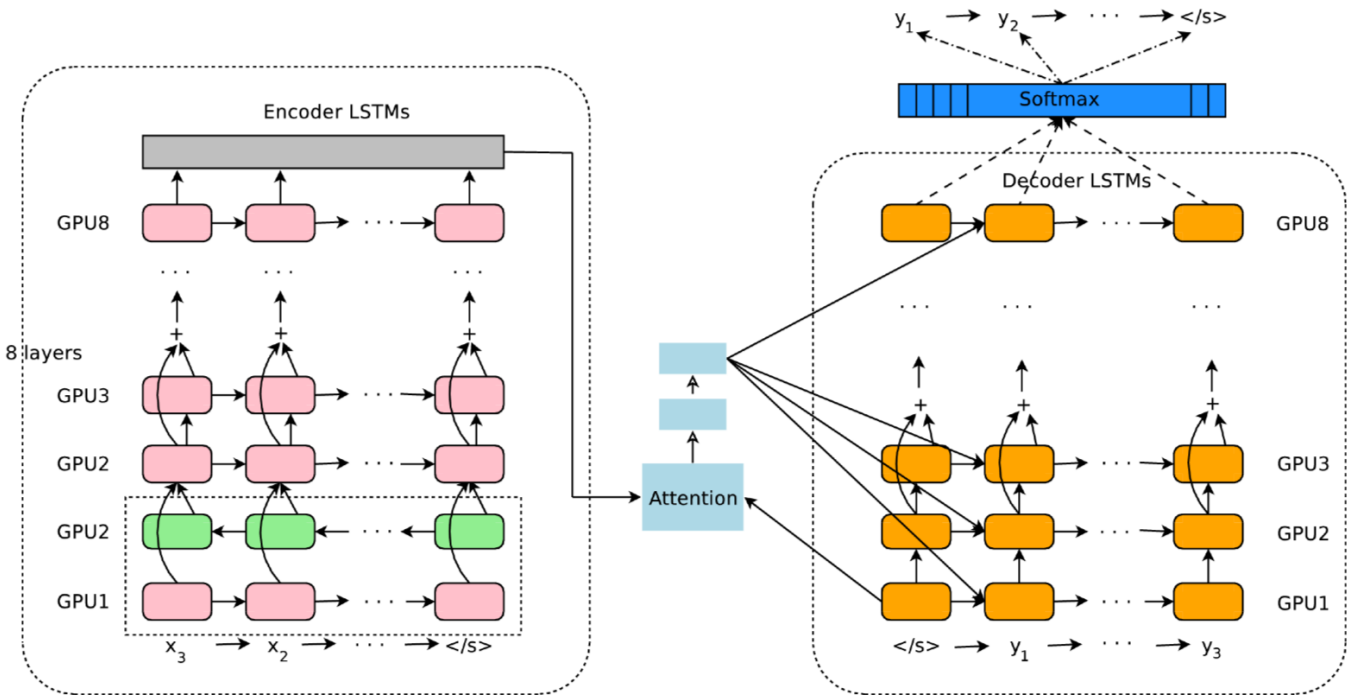
8. 어텐션

최근 딥러닝 연구에서 어텐션이 중요한 기술로 다양한 방면에서 등장한다.

최첨단 연구 3가지를 소개한다.

GNMT의 계층 구성

기계번역의 역사를 보면 다음과 같이 변화해왔다.
규칙 기반 번역 -> 용례 기반 번역 -> 통계 기반 번역
현재는 신경망 기계 번역(NMT) 이 주목받고 있다.
계층 구성은 다음과 같다.



구글 신경망 기계 번역(GNMT)

우리가 앞서 배웠던 어텐션을 갖춘 seq2seq와 마찬가지로
Encoder, Decoder, Attention으로 구성되어 있다.

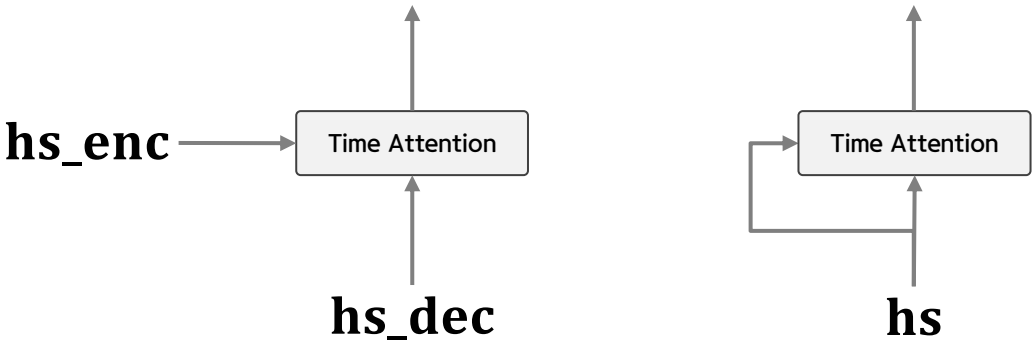
다만, 여기에 번역 정확도를 높이기 위해 LSTM 계층의 다층화, 양방향 LSTM, skip 연결 등을 추가했다.

그리고 학습 시간을 단축하기 위해 GPU로 분산학습을 수행하고 있다.

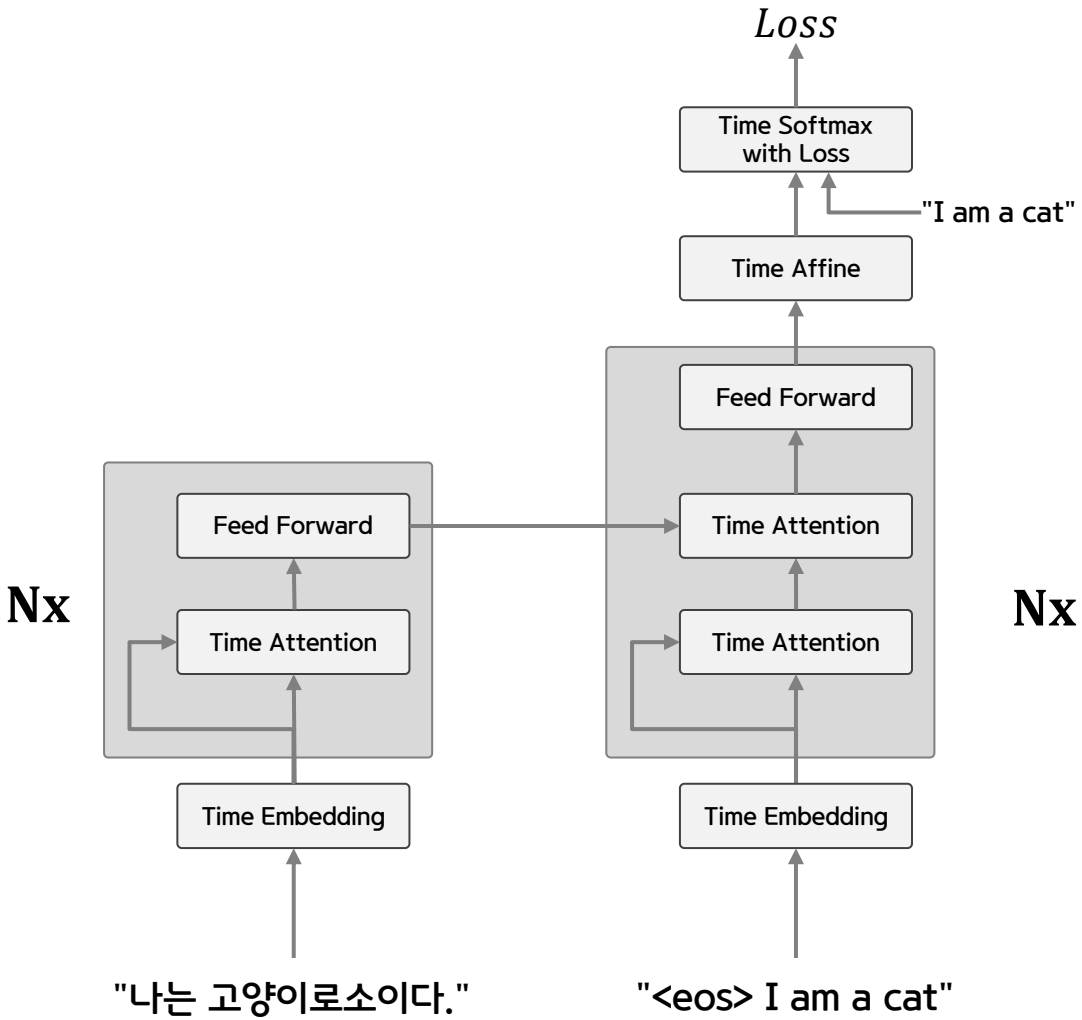
이외에도 낮은 빈도의 단어처리나 추론 고속화를 위한 양자화 등의 연구도 이루어지고 있다.

이로써 점점 사람의 정확도에 가까워지고 있다.

왼쪽이 일반적인 어텐션, 오른쪽이 셀프어텐션



트랜스포머의 계층 구성


Loss
Time Softmax with Loss
"I am a cat"
Time Affine
Feed Forward
Time Attention
Time Attention
Nx
Feed Forward
Time Attention
Nx
Time Embedding
Time Embedding
"나는 고양이로소이다."
"<eos> I am a cat"

RNN의 단점 중 하나는 병렬처리다.

RNN은 이전 시각에 계산한 결과를 이용하여 순서대로 계산하기 때문에
RNN 의 계산을 시간방향으로 병렬계산하기란 기본적으로 불가능하다.

이는 딥러닝 학습이 GPU를 사용한 병렬계산환경에서 이뤄진다는 점을 생각할 때 큰 병목이다.

이것의 해결방안을 제안한 연구 중 트랜스포머 기법이 유명하다.

셀프 어탠션 기술을 이용해 어탠션을 구성하는 것이 핵심이다.
'하나의 시계열 데이터 내에서 각 원소가 다른 원소들과 어떻게 관련되는지'를 살펴보자는 취지다.

앞서 보았던 어텐션에서는 2개의 서로 다른 시계열 데이터(hs_enc, hs_dec) 사이의 대응관계를 구했으나,
셀프 어텐션은 두 입력선이 하나의 시계열 데이터로부터 나온다.

트랜스포머는 RNN 대신 어텐션을 사용한다.
encoder, decoder 모두 셀프 어텐션을 사용한다.

또 피드포워드 신경망(시간 방향으로 독립적으로 처리하는 신경망) 을 넣는다.

은닉층 1개, 활성화 함수로 ReLU 를 이용해 완전연결계층 신경망을 이용한다.