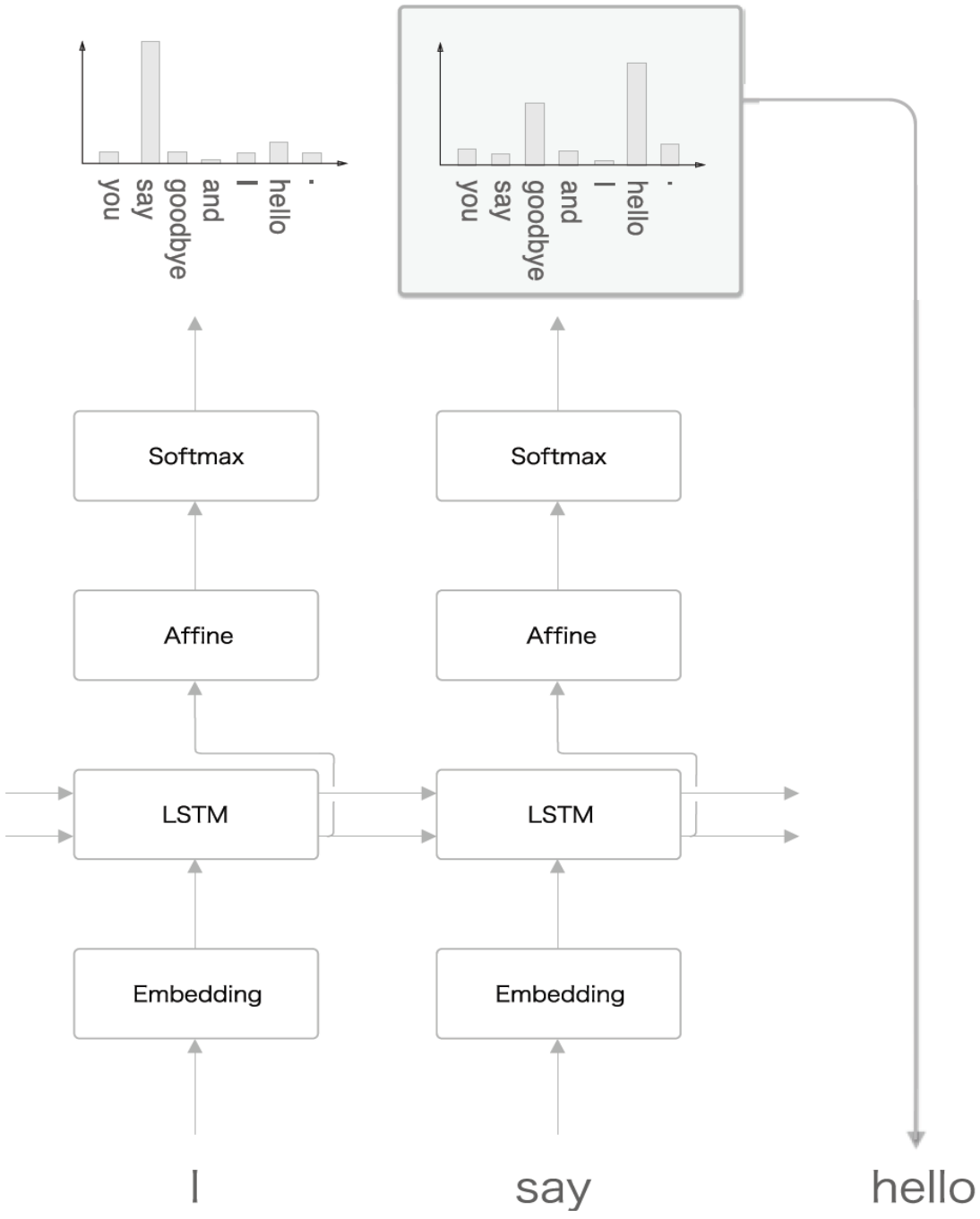
chapter 7 RNN을 사용한 문장 생성

seq2seq: sequence to sequence, RNN 2개를 연결하는 방법으로 구현

기계번역, 챗봇, 메일의 자동 답신에 응용

7.1 언어 모델을 사용한 문장 생성

그림 7-4 확률분포 출력과 샘플링을 반복한다.



7.1.2 문장 생성 구현

ch7/rnnlm_gen.py

```
from ch06.rnnlm import Rnnlm
from ch06.better_rnnlm import BetterRnnlm
class RnnlmGen(Rnnlm): # 상속
    def generate(self, start_id, skip_ids=None, sample_size=100):
        word_ids = [ start_id ] # 최소 시작 단어
        x = start_id
        while len(word_ids) < sample_size:
            x = np.array(x).reshape(1, 1) #신경망 처리 자료 형태로
            score = self.predict(x)
            p = softmax(score.flatten())
            sampled = np.random.choice(len(p), size=1, p=p) #샘플링
            if (skip_ids is None) or (sampled not in skip_ids):
                x = sampled
                word_ids.append(int(x))
        return word_ids
# skip_ids: <unk>희소단어 N 숫자 <eos> 문장구분
```

문장생성: ch07/generate_text.py 읽기

더 그럴듯한 문장생성: ch07/generate_better_text.py 읽기

BetterRnnlmGen() (rnnlm_gen.py에 존재) 으로 generate

7.2 seq2seq

언어, 음성, 동영상 등의 시계열 데이터를 다른 시계열 데이터로 변환

그림 7-5 Encoder와 Decoder가 번역을 수행하는 예

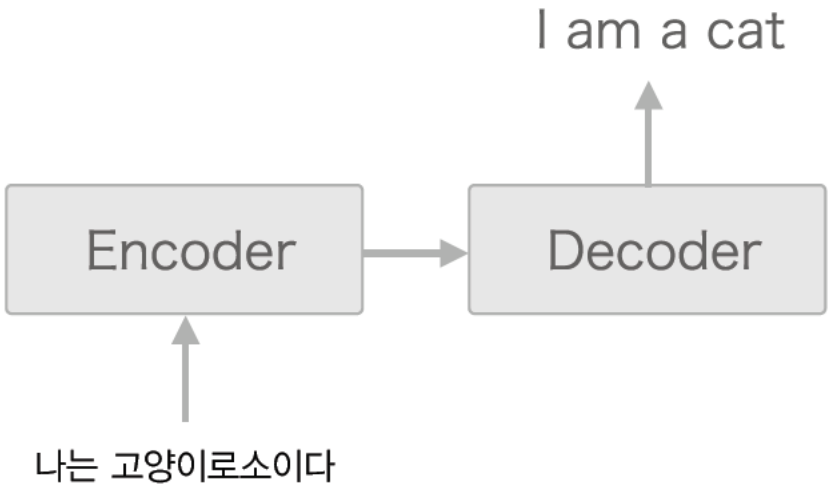


그림 7-6 Encoder를 구성하는 계층

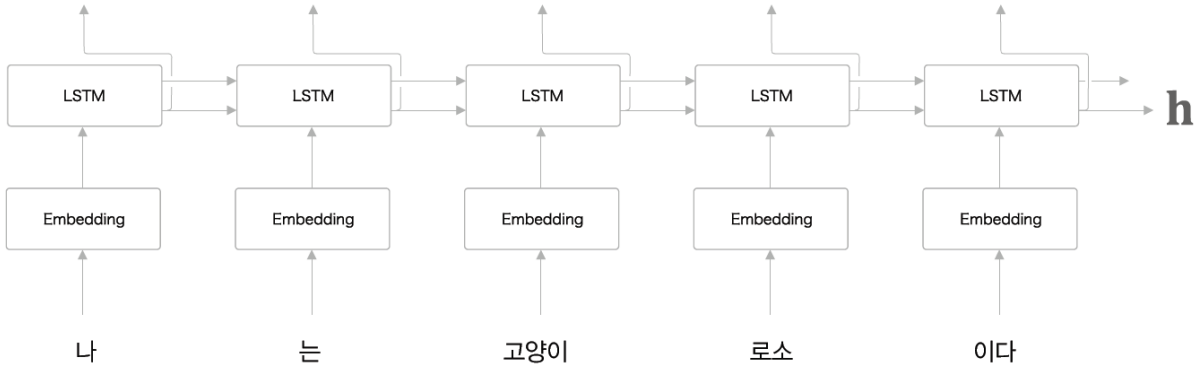


그림 7-7 Encoder는 문장을 고정 길이 벡터로 인코딩한다.

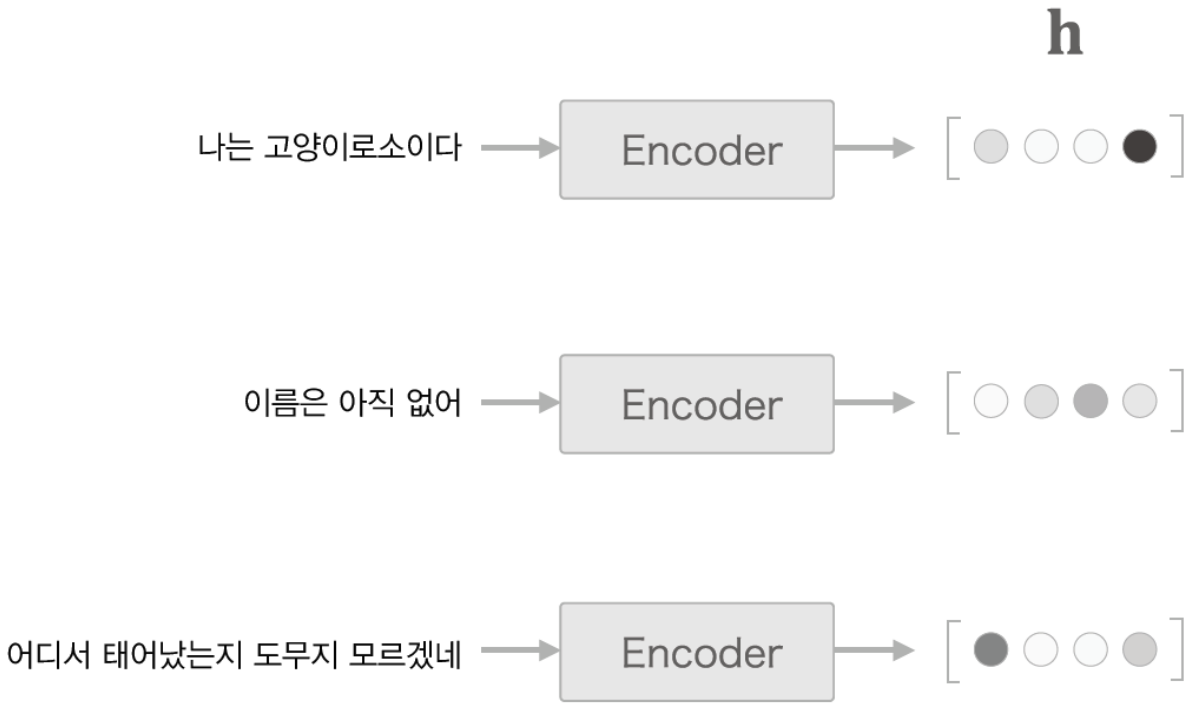


그림 7-8 Decoder를 구성하는 계층

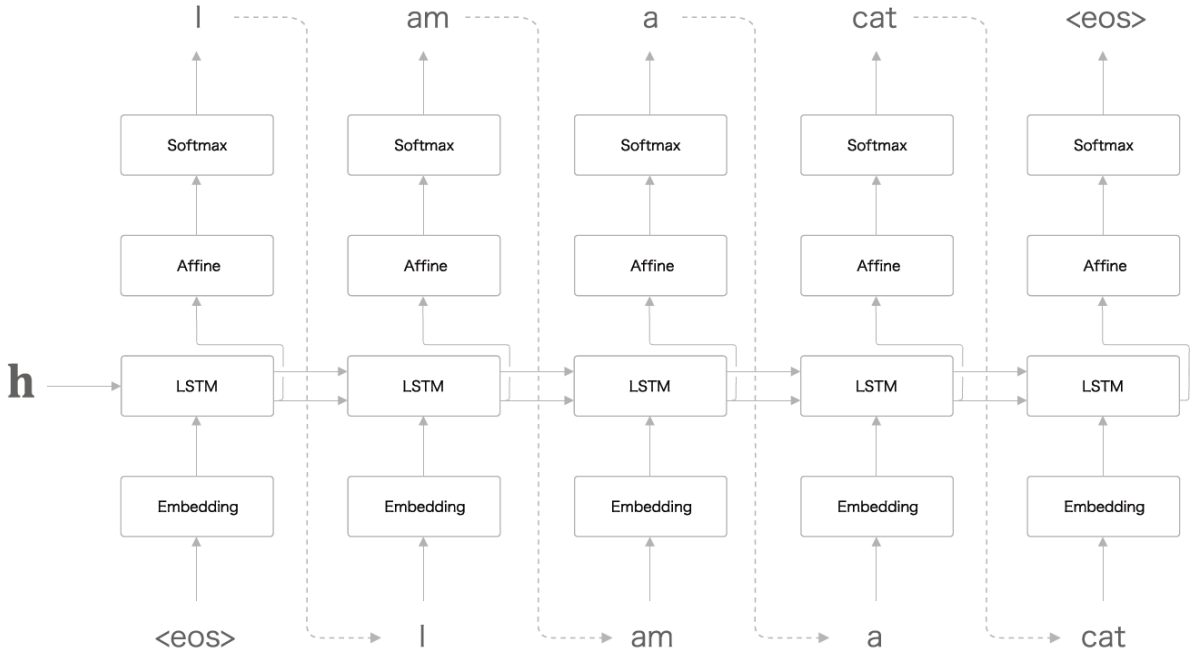
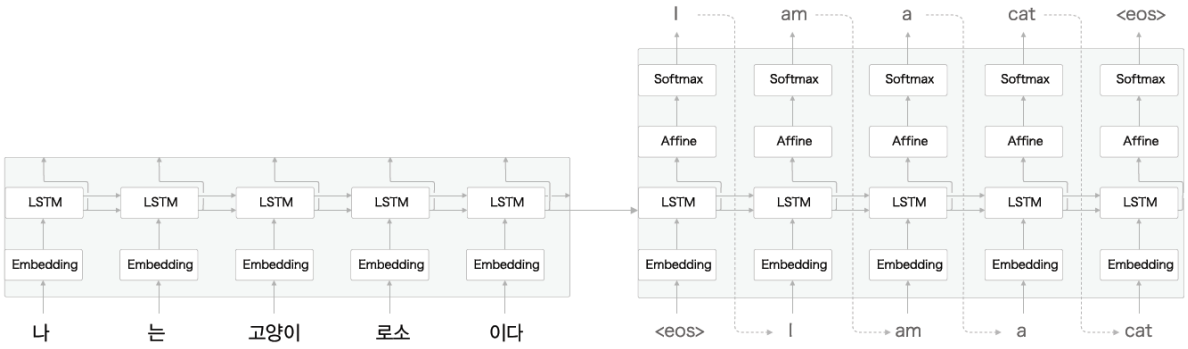
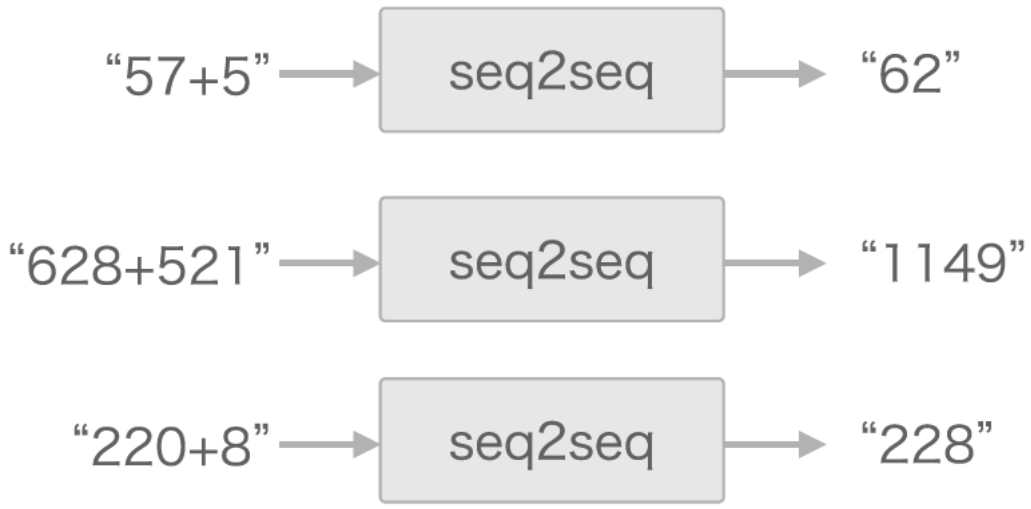


그림 7-9 seq2seq의 전체 계층 구성



7.2.2 시계열 데이터 변환용 장난감 문제

그림 7-10 seq2seq에 덧셈 예제들을 학습시킨다.



7.2.3 가변 길이 시계열 데이터

그림 7-11 미니배치 학습을 위해 '공백 문자'로 패딩을 수행하여 입력 · 출력 데이터의 크기를 통일한다.

입력

5	7	+	5			
6	2	8	+	5	2	1
2	2	0	+	8		

출력

_	6	2		
_	1	1	4	9
_	2	2	8	

질문과 정답을 구분하기 위한 _ , decoder 에 문자열 생성 신호 줌

패딩은 무시하는 손실함수 필요

7.2.4 덧셈 dataset

세자리 숫자까지만 처리 가능

그림 7-12 '덧셈' 학습 데이터: 공백 문자space는 회색 가운뎃점으로 표기

```
16+75  _91
52+607 _659
75+22  _97
63+22  _85
795+3  _798
706+796_1502
8+4    _12
84+317 _401
9+3    _12
6+2    _8
18+8   _26
85+52  _137
9+1    _10
8+20   _28
5+3    _8
```

Lines: 50,000 Chars: 650,000 650 KB

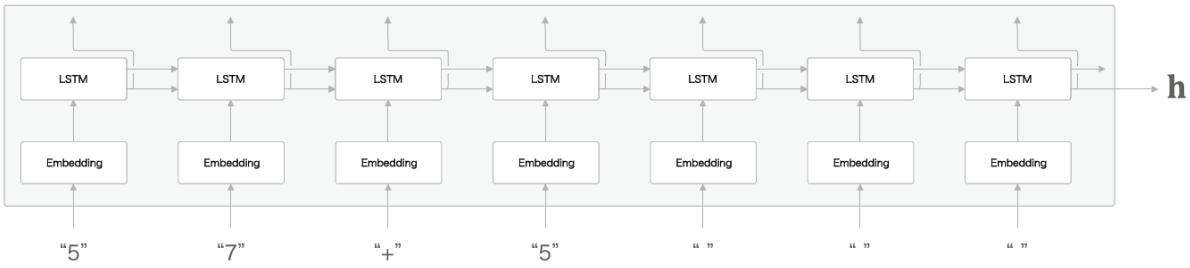
ch7/show_addition_dataset.py 실행

```
# coding: utf-8
import sys
sys.path.append('..')
from dataset import sequence #숫자 자료 처리
```

```
(x_train, t_train), (x_test, t_test) = ₩
        sequence.load_data('addition.txt', seed=1984) #자료 분리
char_to_id, id_to_char = sequence.get_vocab() #문자 ←→ id


print(x_train.shape, t_train.shape)
print(x_test.shape, t_test.shape)
# (45000, 7) (45000, 5)
# (5000, 7) (5000, 5)


print(x_train[0])
print(t_train[0])
# [ 3  0  2  0  0 11  5]
# [ 6  0 11  7  5]


print(''.join([id_to_char[c] for c in x_train[0]]))
print(''.join([id_to_char[c] for c in t_train[0]]))
# 71+118
# _189
```

7.3 seq2seq 구현

그림 7-14 Encoder의 계층 구성



ch07/seq2seq.py 중 Encoder 읽기

그림 7-17 Decoder의 계층 구성(학습 시)

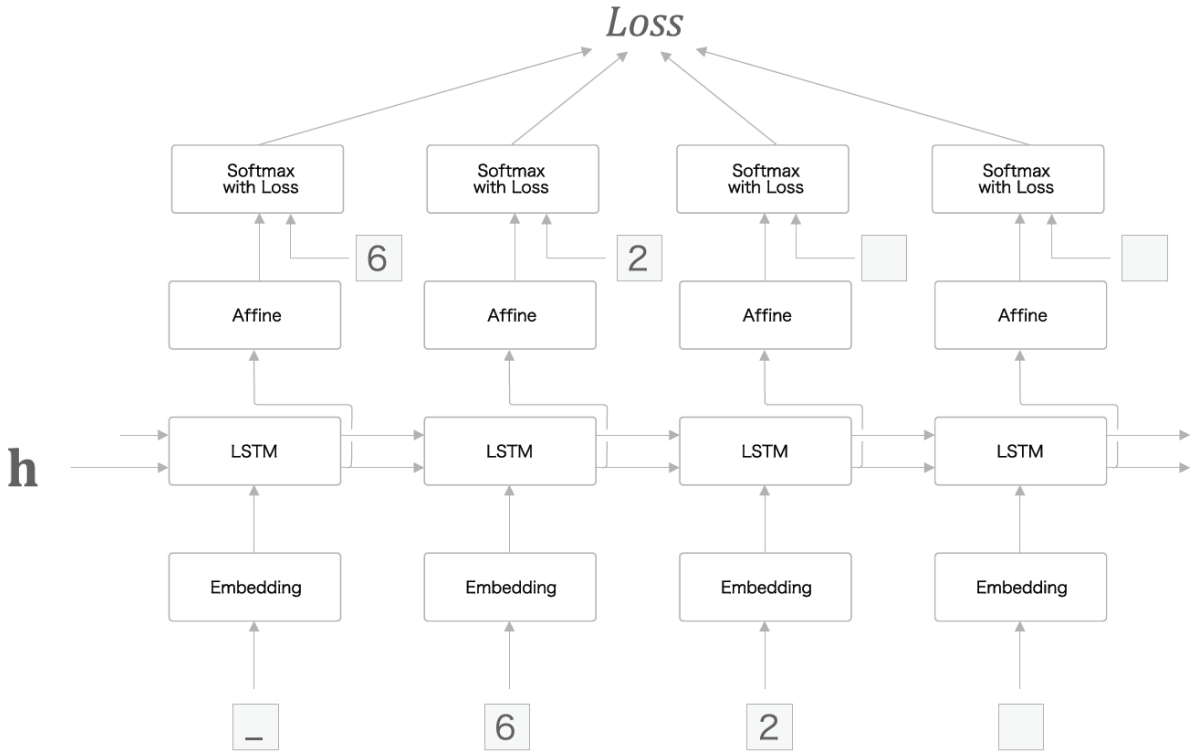
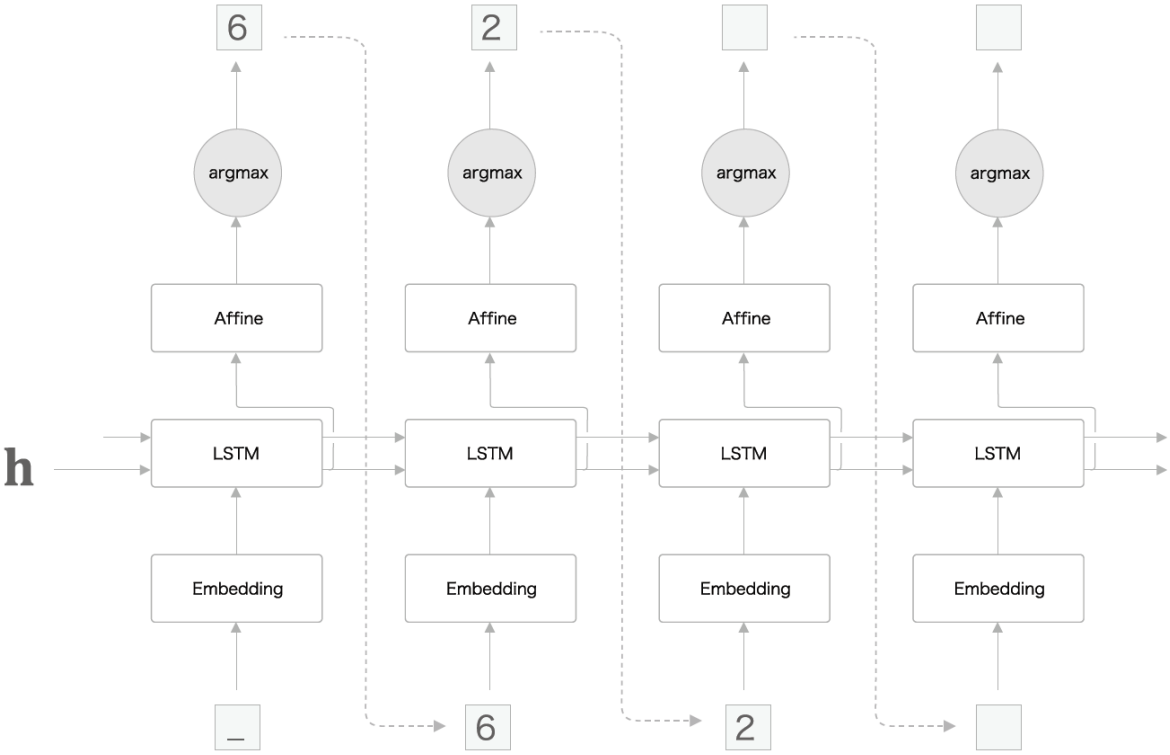


그림 7-18 Decoder의 문자열 생성 순서: argmax 노드는 Affine 계층의 출력 중 값이 가장 큰 원소의 인덱스(문자 ID)를 반환한다.



ch07/seq2seq.py 중 Decoder 읽기

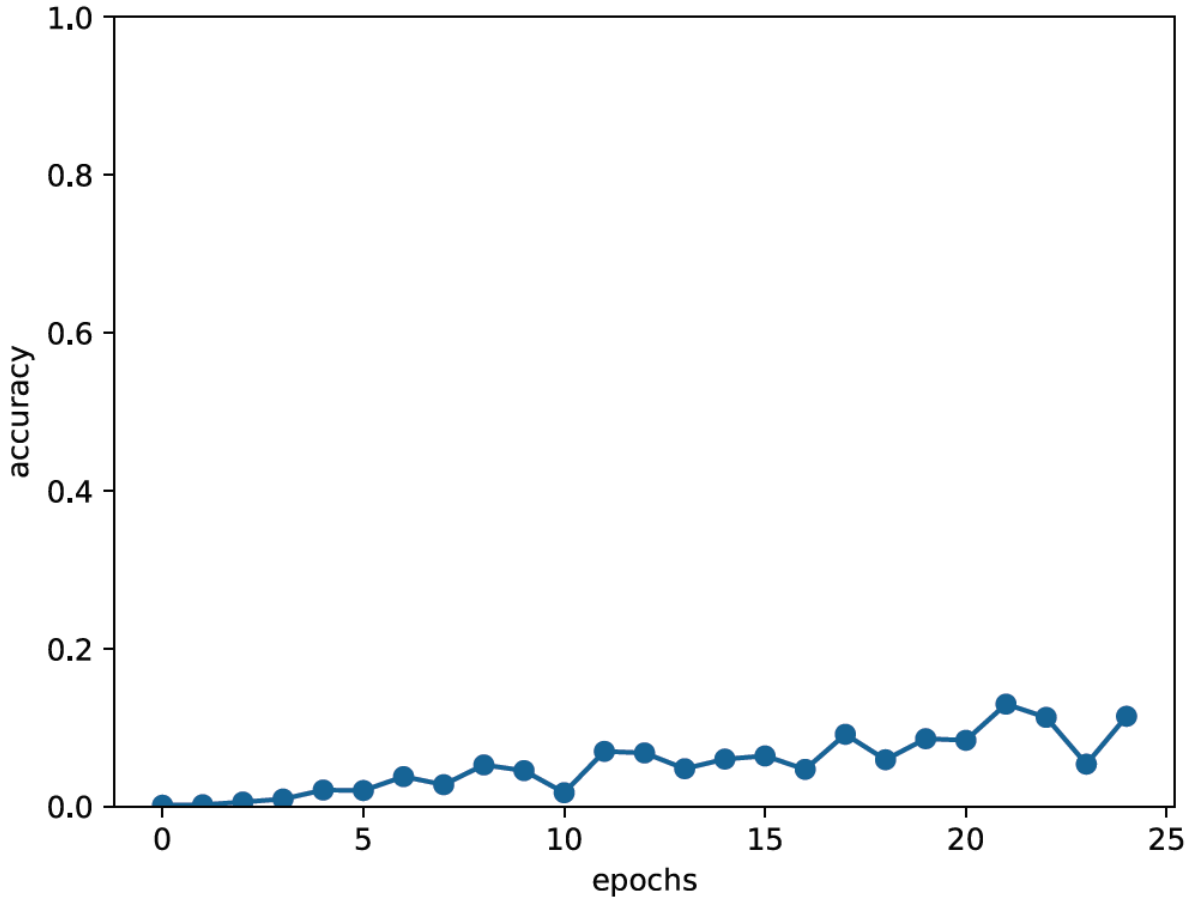
7.3.3 Seq2seq 클래스

ch07/seq2seq.py 중 Seq2seg 읽기

7.3.4 seq2seq 평가

ch07/train_seq2seq.py 읽기

그림 7-22 정답률 추이



7.4 seq2seq 개선

7.4.1 입력 데이터 반전 (reverse)

그림 7-23 입력 데이터를 반전시키는 예

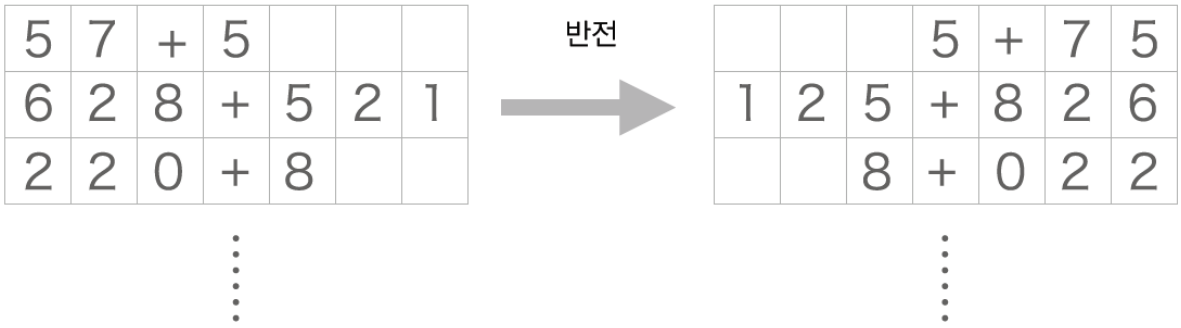
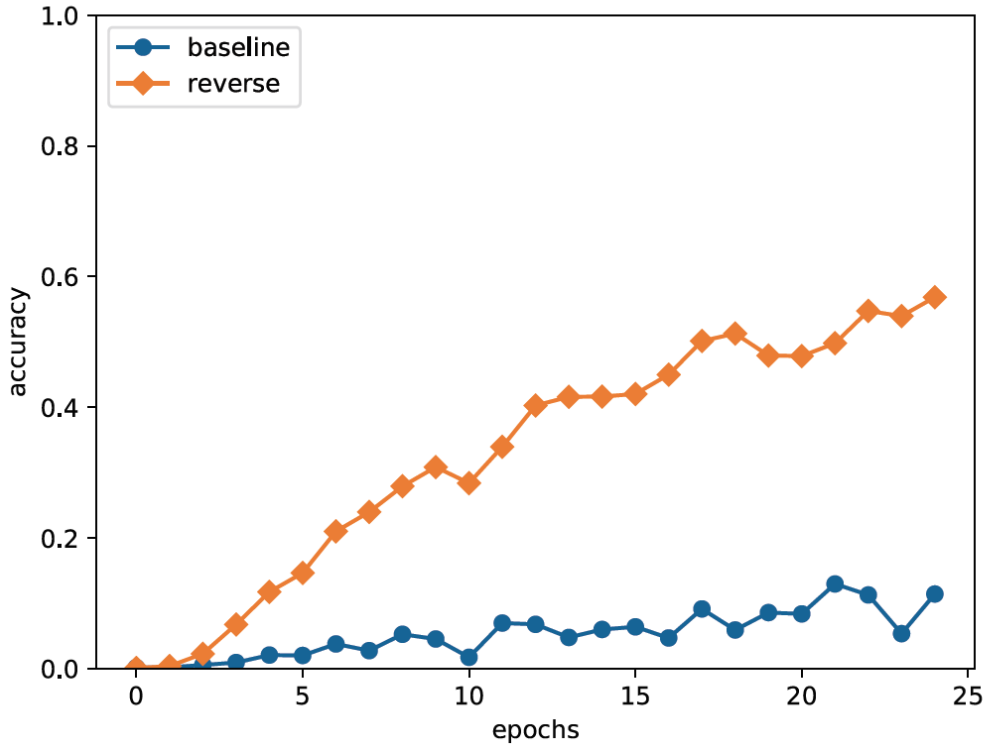


그림 7-24 seq2seq의 정답률 추이: baseline은 앞 절의 결과, reverse는 입력 데이터를 반전시킨 결과



7.4.2 엿보기 (peeky)

그림 7-26 개선 후: Encoder의 출력 **h**를 모든 시각의 LSTM 계층과 Affine 계층에 전해준다.

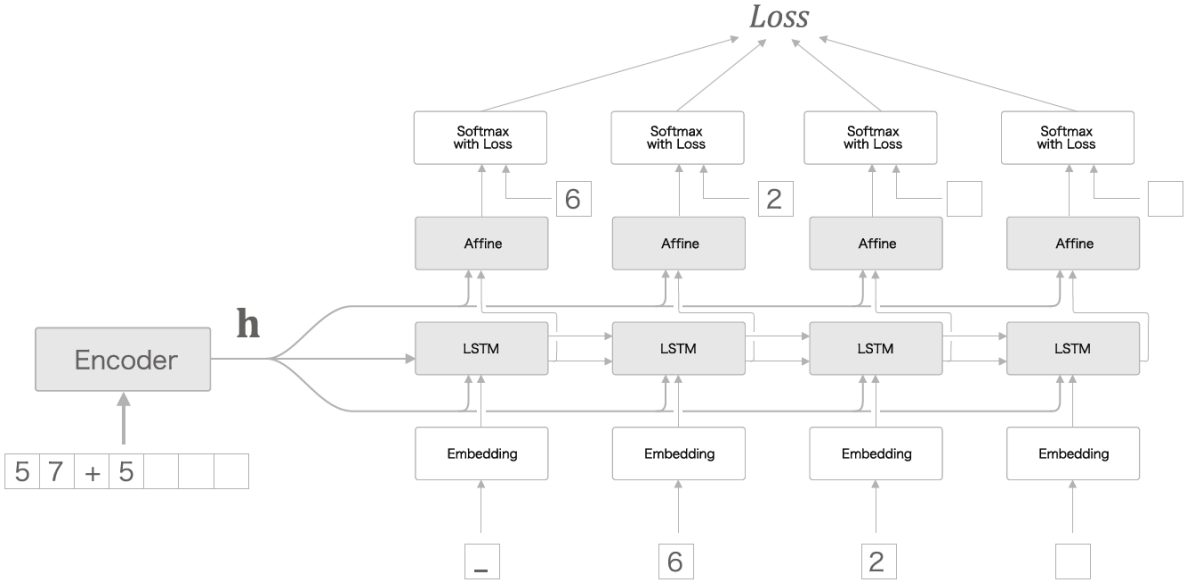
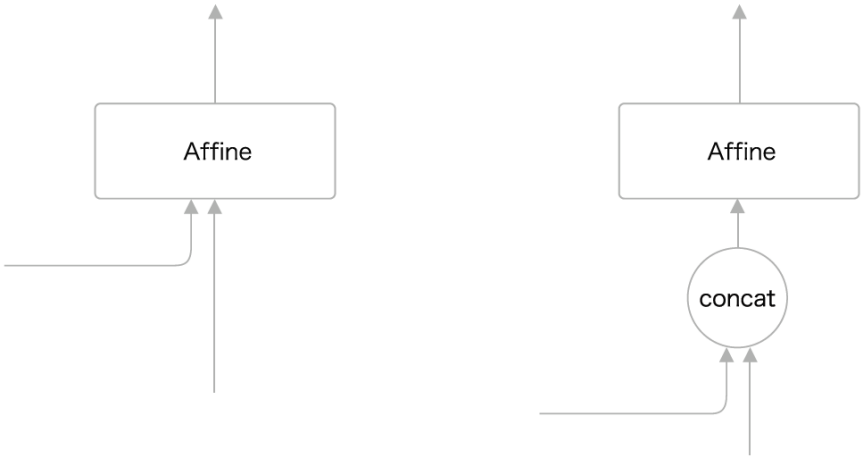


그림 7-27 Affine 계층에 입력이 2개인 경우(왼쪽)를 정확하게 그리면, 그 두 입력을 연결한 하나의 벡터가 입력되는 것이다(오른쪽).



ch07/peeky_seq2seq.py 읽기

초기화: lstm: H+D, affine: H + H

forward():

x =[[1,2],[3,4]] np.repeat(x,3,axis=0)

Out[7]:

array([[1, 2],

[1, 2],

[1, 2],

[3, 4],

[3, 4],

[3, 4]])

backward(): 앞부분 H는 앞에 더해진 h 처리부분, 시간별로 더해지므로 모두 더해서 역전파

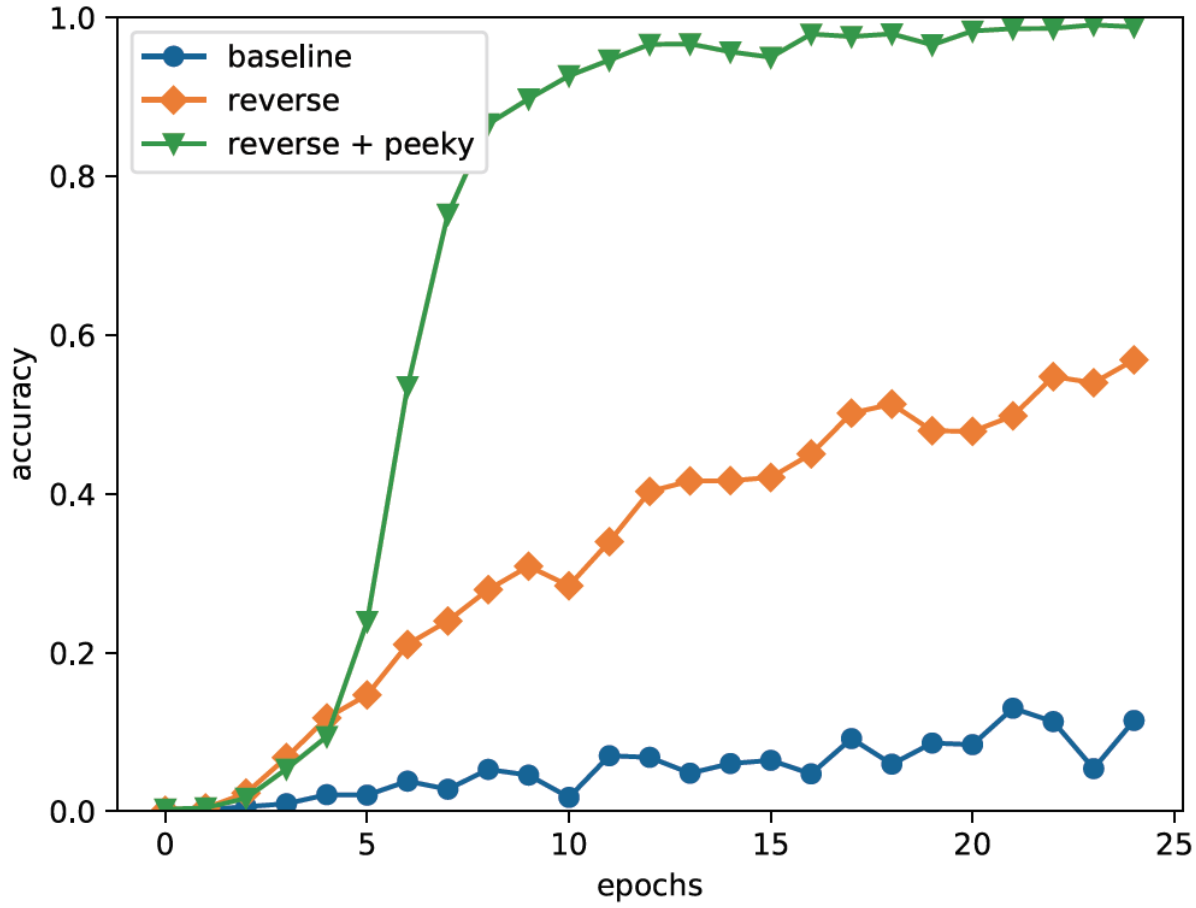
class PeekySeq2seq(Seq2seq): 상속

ch07/train_seq2seq.py에서

is_reverse = False # True

model = PeekySeq2seq(vocab_size, wordvec_size, hideen_size) 사용

그림 7-28 'reverse + peeky' 조합: 두 가지 개선을 모두 적용한 결과



7.5 seq2seq를 이용하는 application

p328 부터 읽기

chapter 8 attention

8.1 attention 구조: 주목

8.1.1 seq2seq의 문제점

그림 8-1 입력 문장의 길이에 관계없이, Encoder는 정보를 고정 길이의 벡터로 밀어 넣는다.

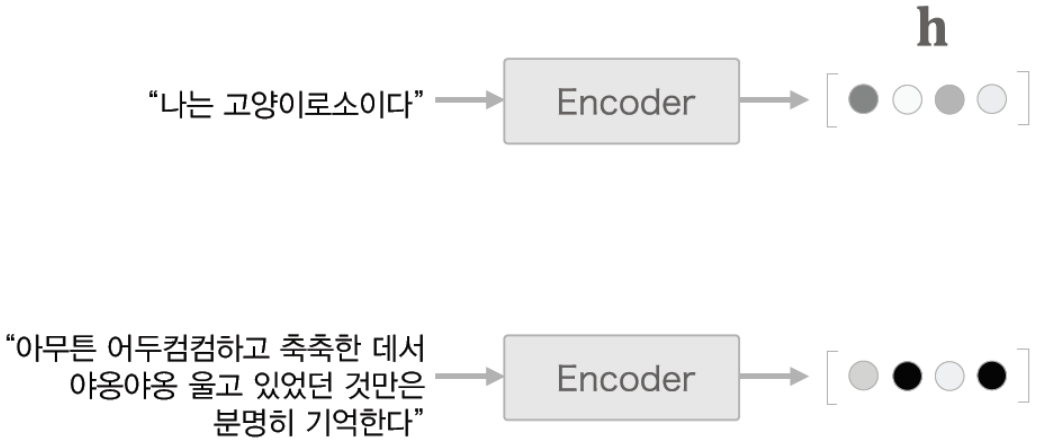
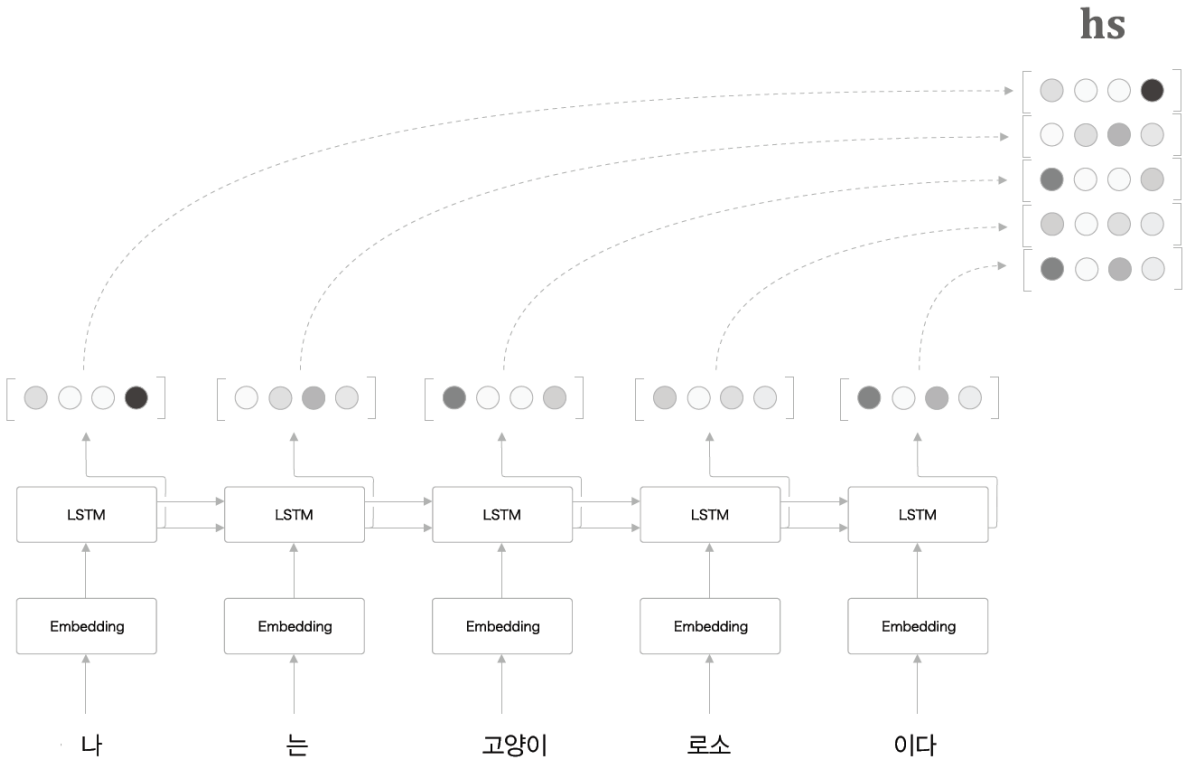


그림 8-2 Encoder의 시각별(단어별) LSTM 계층의 은닉 상태를 모두 이용(**hs**로 표기)



8.1.3 Decoder의 개선 1

필요한 정보에만 주목하여 그 정보로부터 시계열 변환을 수행

그림 8.2의 hs에서 골라냄 → 선택은 미분 불가능 → 가중치 적용

그림 8-6 개선 후의 Decoder의 계층 구성

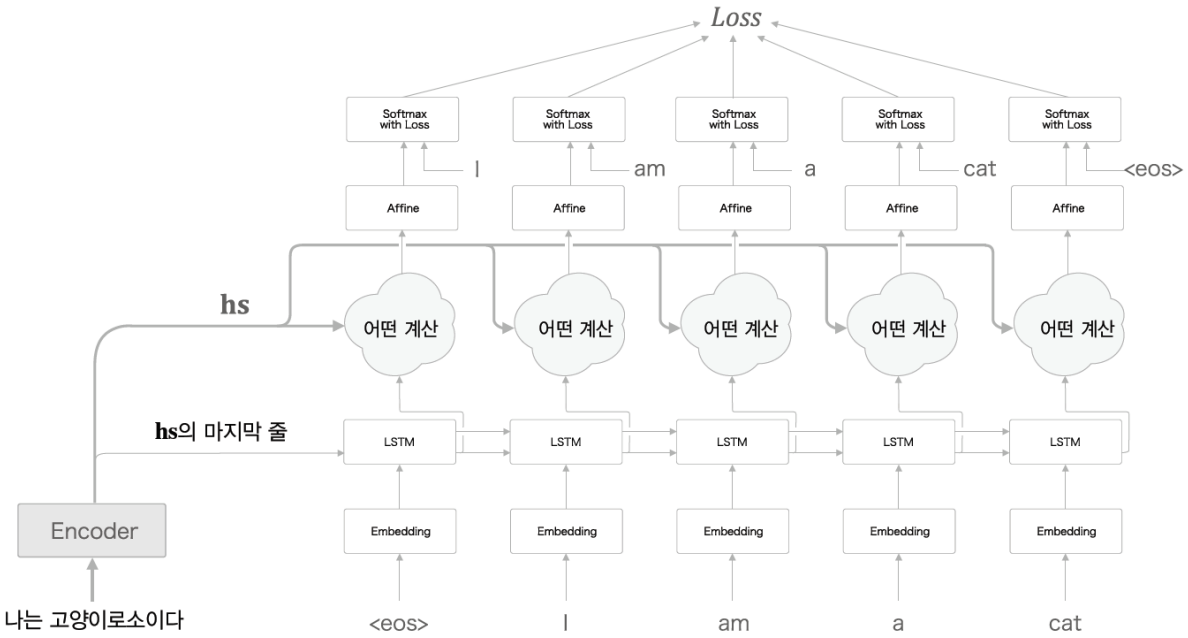
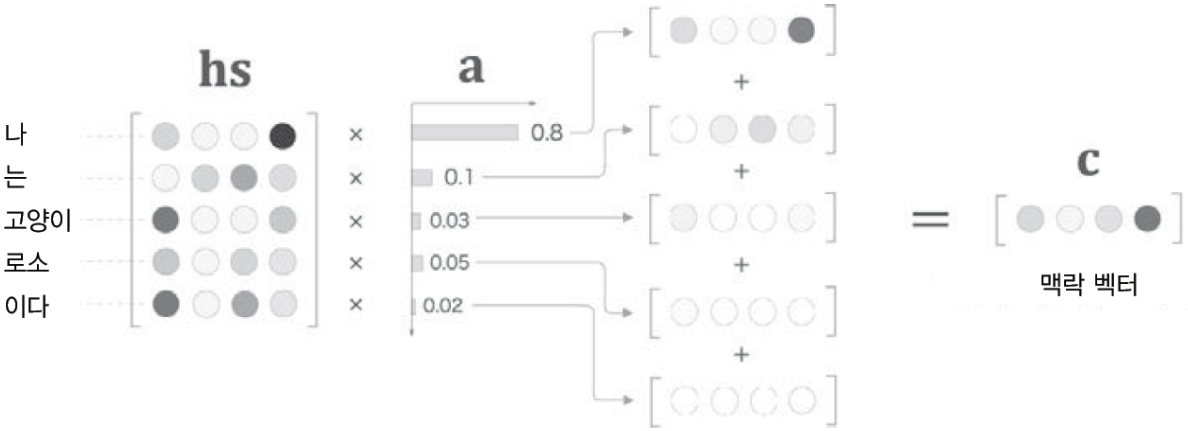


그림 8-8 가중합을 계산하여 '맥락 벡터'를 구한다.



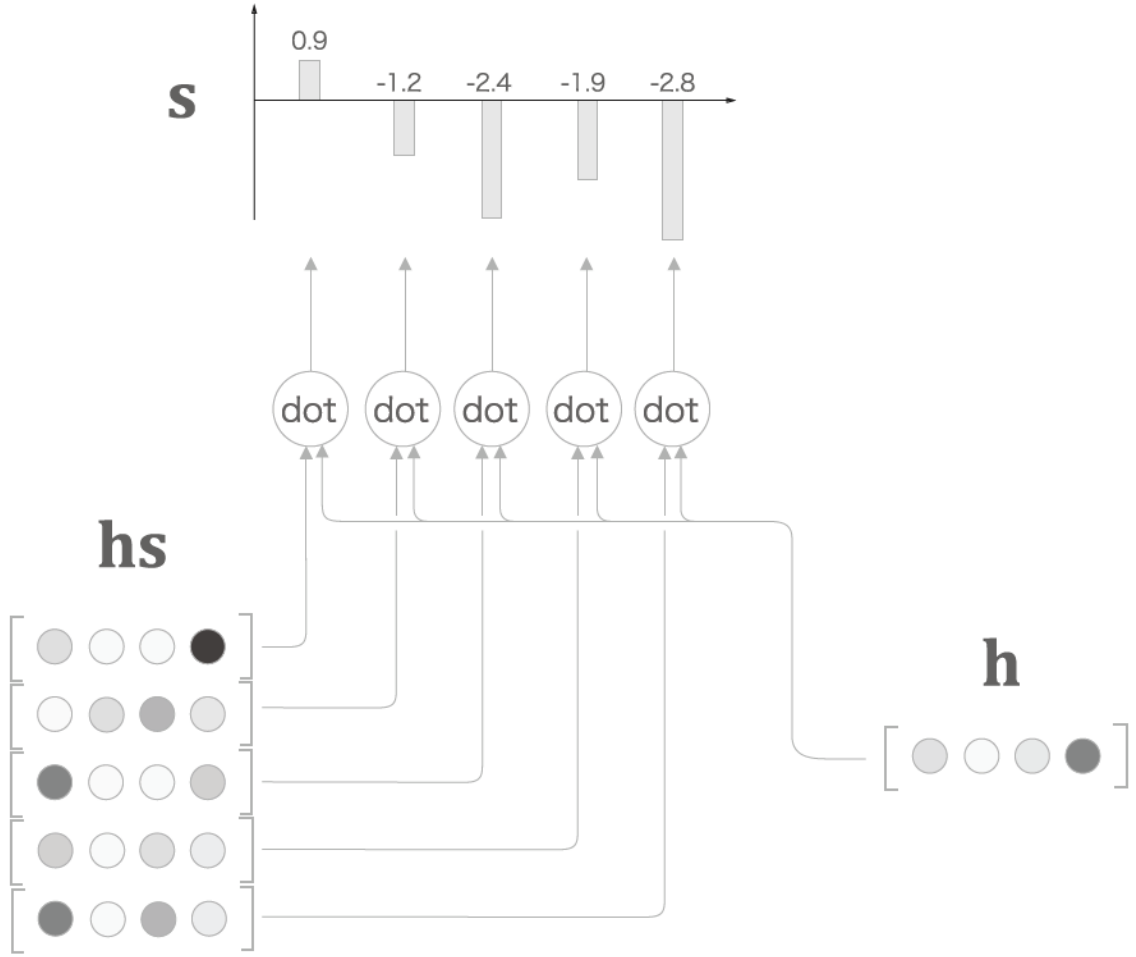
ch08/attention_layer.py의 class WeightSum: 읽기, 함수처럼 동작

repeat의 역전파는 sum, sum의 역전파는 repeat

8.1.4 Decoder의 개선 2

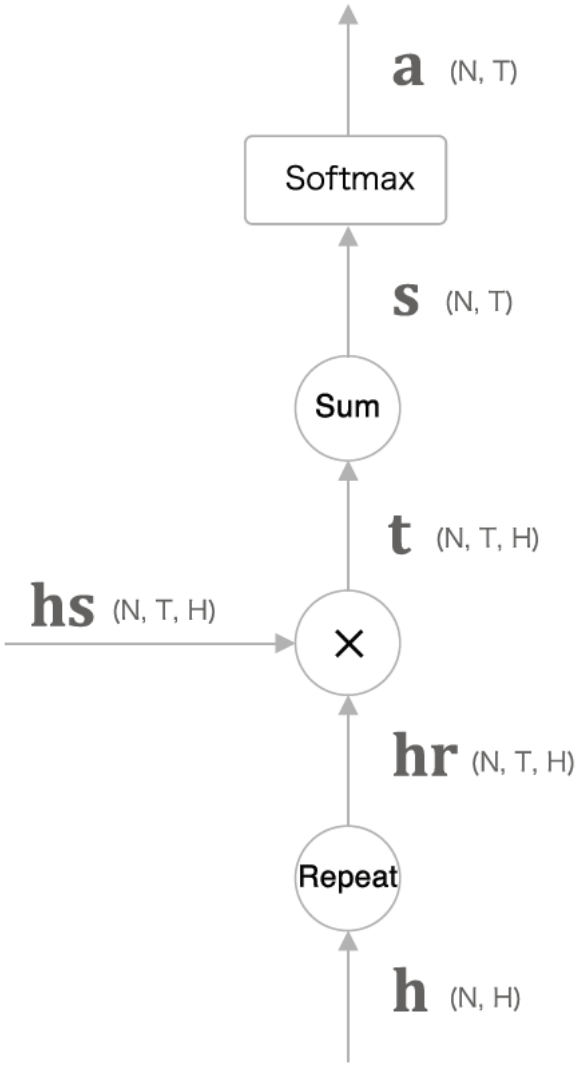
가중치 a는 어떻게 얻나?

그림 8-13 내적을 통해 **hs**의 각 행과 **h**의 유사도를 산출(내적은 dot 노드로 그림)



S를 가중치 a로 바꾸기 위해 softmax

그림 8-15 각 단어의 가중치를 구하는 계산 그래프



ch08/attention_layer.py의 class AttentionWeight: 읽기

8.1.4 Decoder의 개선 3

앞 두 단계를 합침

그림 8-16 맥락 벡터를 계산하는 계산 그래프

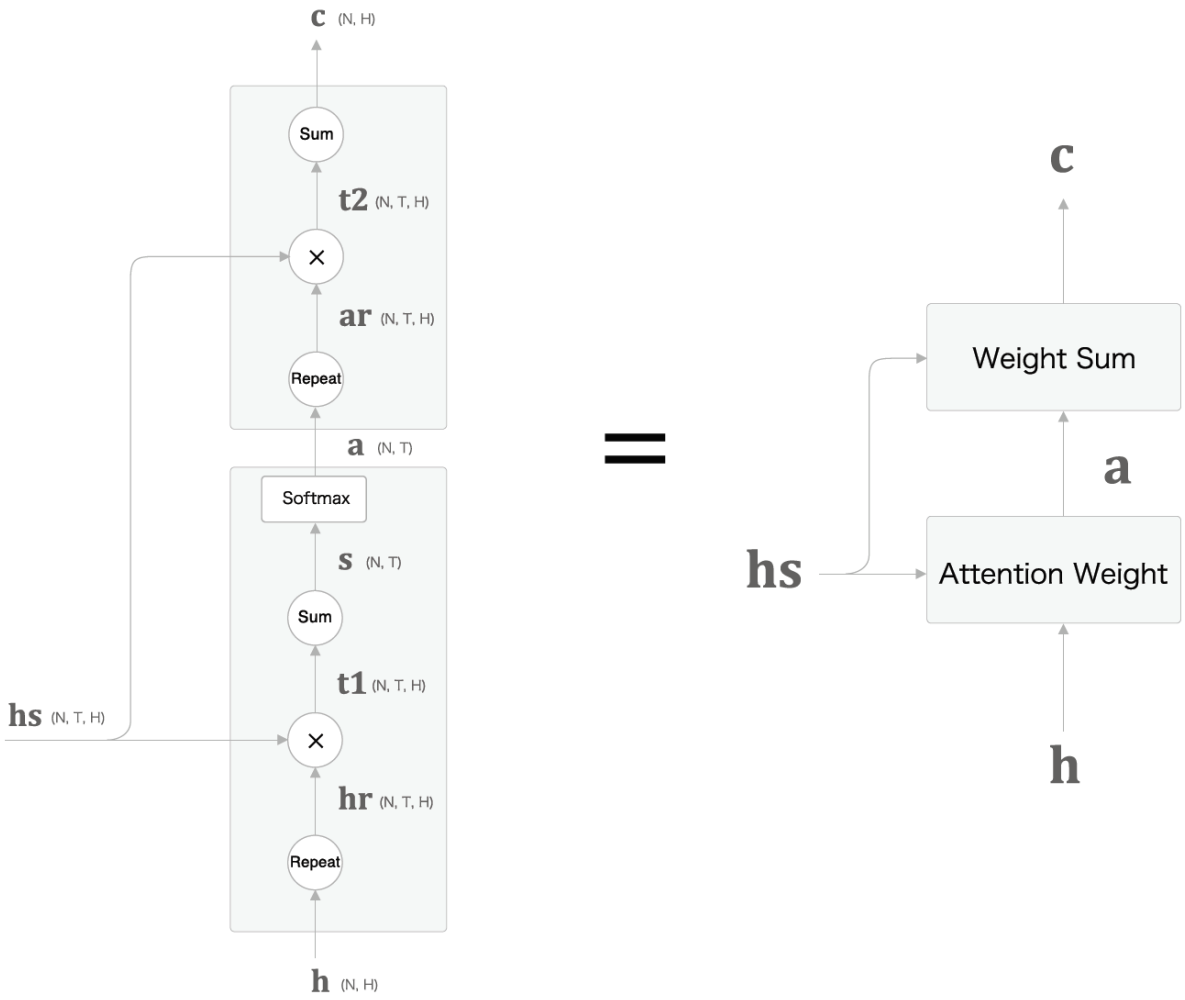


그림 8-17 왼쪽 계산 그래프를 Attention 계층으로 정리

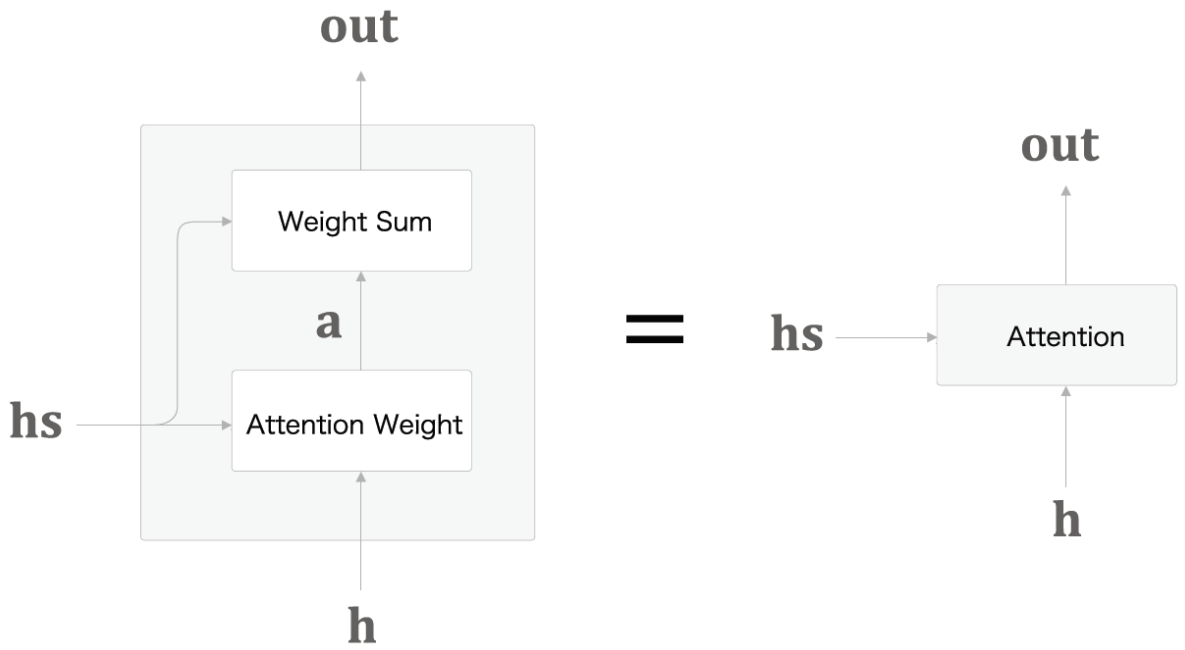


그림 8-18 Attention 계층을 갖춘 Decoder의 계층 구성

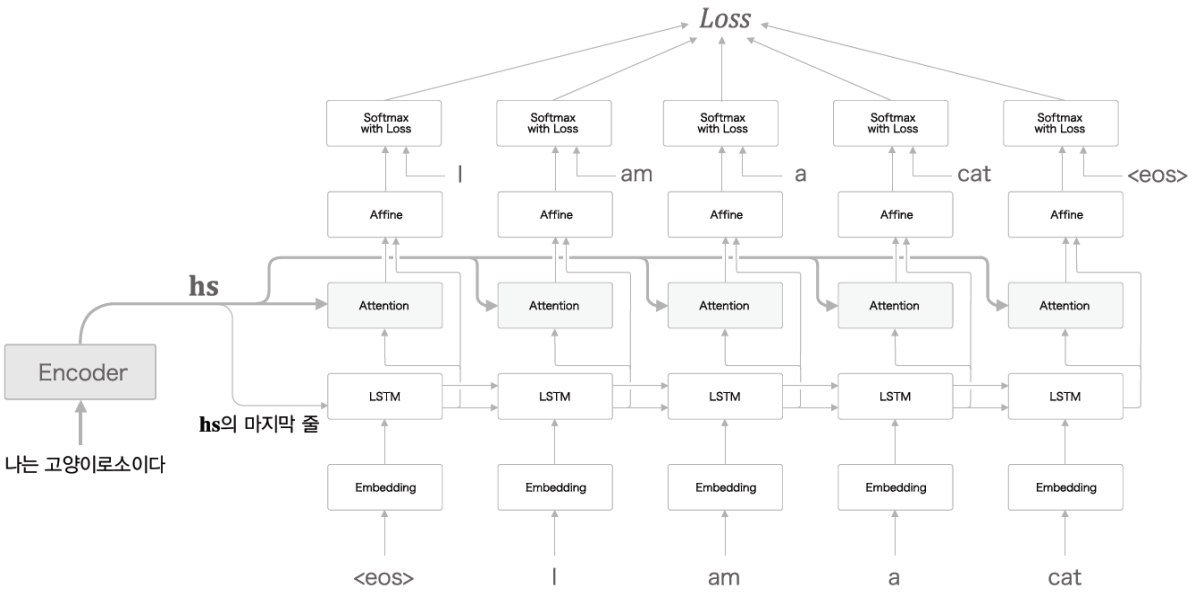
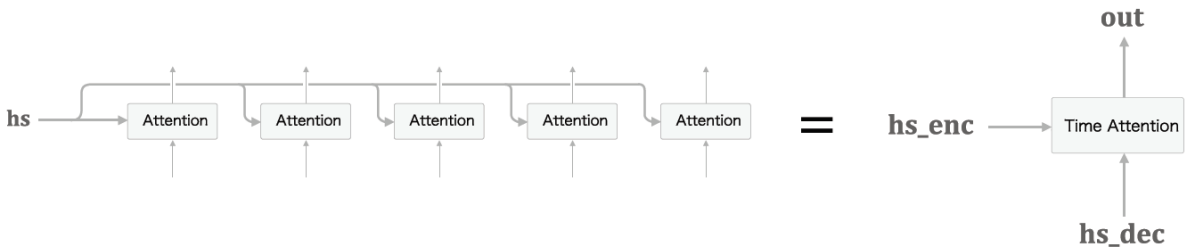


그림 8-20 다수의 Attention 계층을 Time Attention 계층으로서 모아 구현



ch08/attention_layer.py의 class TimeAttention: 읽기

backward(): dhs_enc는 hs_enc가 여러 시간에 decoder로 입력되므로 역전파에서는 더해야 함

8.2 attention을 갖춘 seq2seq 구현

ch08/attention_seq2seq.py 읽기

여러 hs 전달

8.2.3 seq2seq 구현

ch08/attention_seq2seq.py 의 class AttentionSeq2seq(Seq2seq):

8.3 attention 평가

자료 예) WMT 영어/프랑스(영어/독일어) 학습 데이터 쌍, 20GB

8.3.1

그림 8-22 날짜 형식 변환의 예

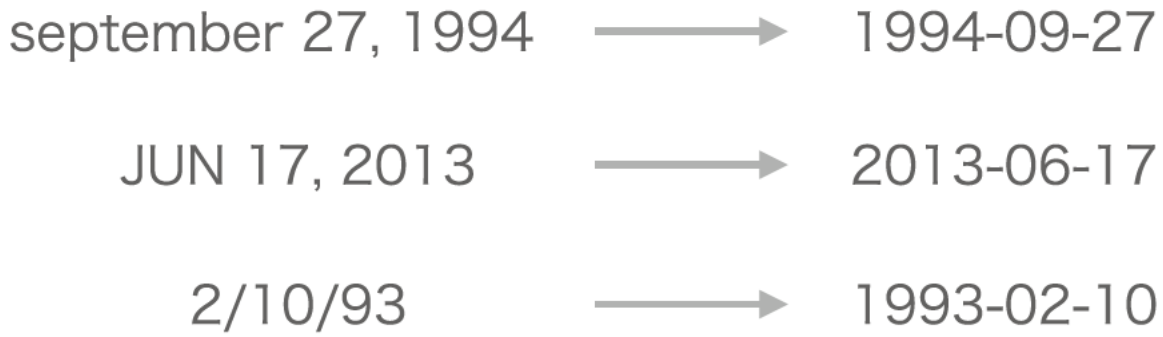
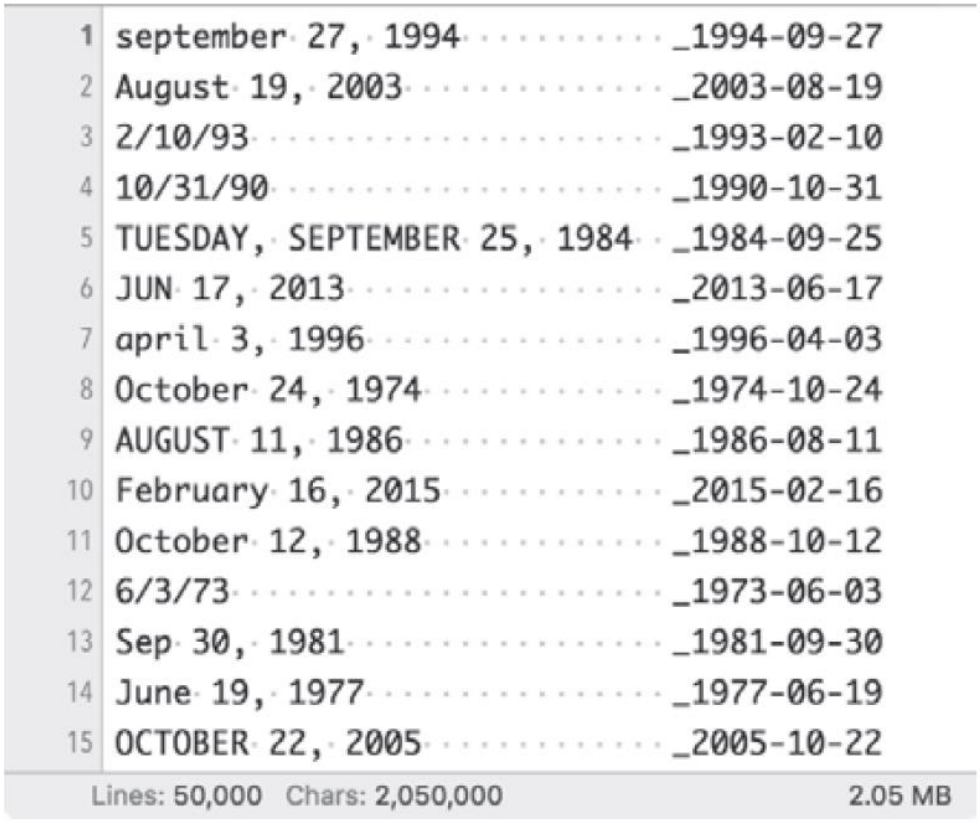


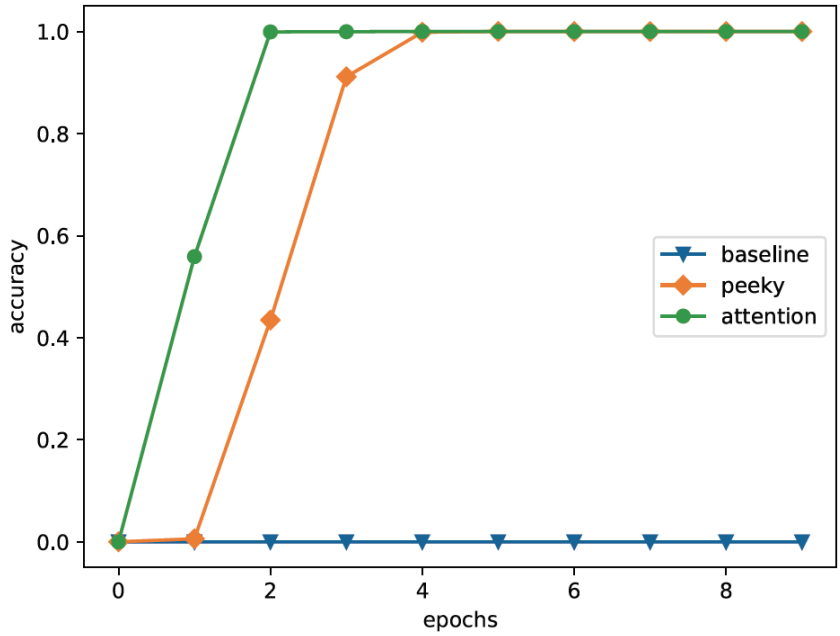
그림 8-23 날짜 형식 변환을 위한 학습 데이터: 공백 문자는 회색 가운뎃점으로 표기



8.3.2 attention을 갖춘 seq2seq의 학습

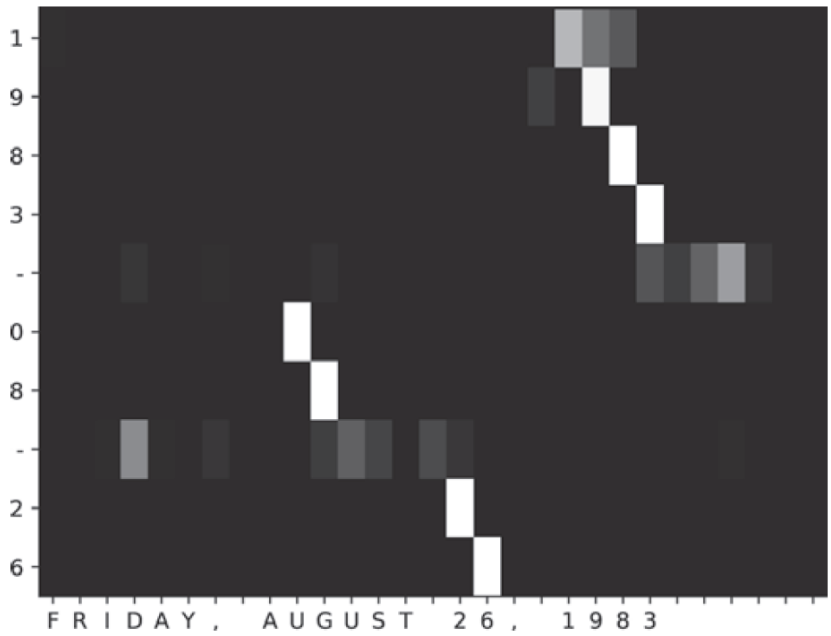
ch08/train.py 읽기

그림 8-26 다른 모델과의 비교: 'baseline'은 앞 장의 단순한 seq2seq, 'peeky'는 엿보기를 적용한 seq2seq(입력 문장 반전은 모든 모델에서 사용)



8.3.3 어텐션 시각화

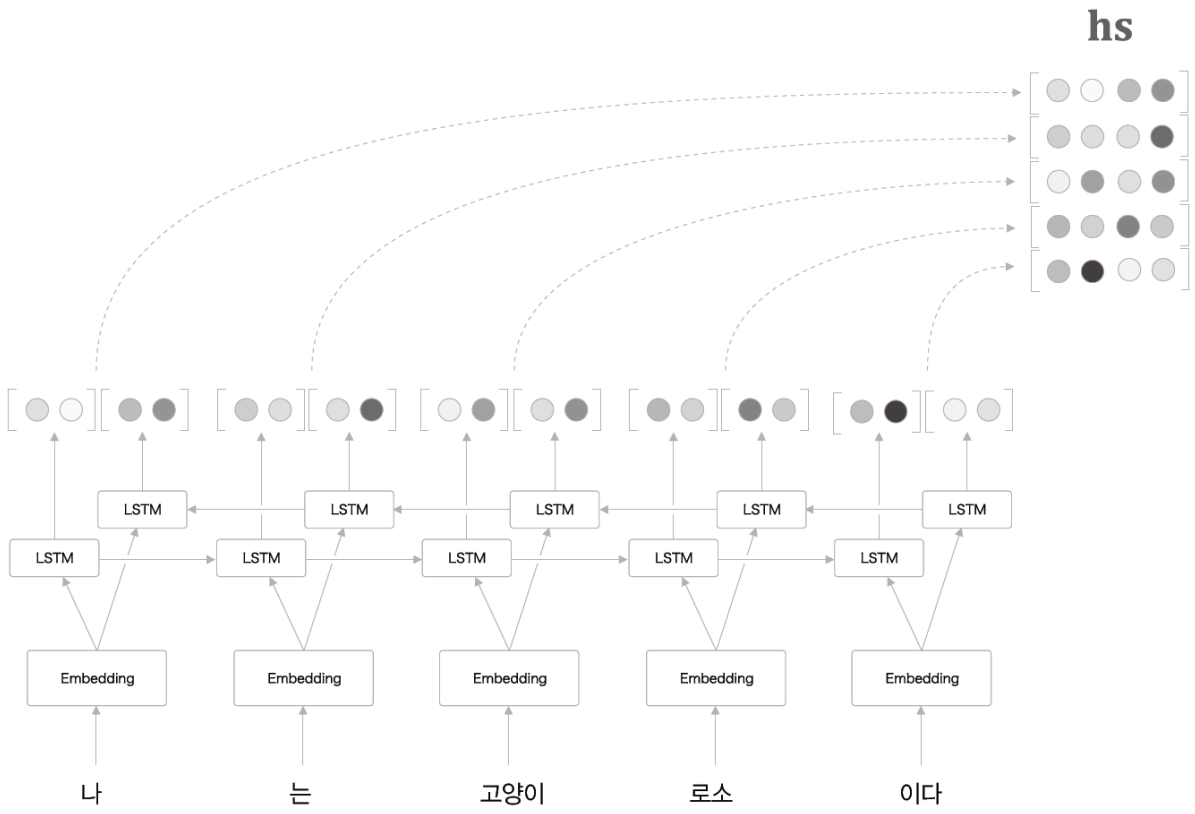
그림 8-27 학습된 모델을 사용하여 시계열 변환을 수행했을 때의 어텐션 가중치 시각화: 가로축은 입력 문장, 세로축은 출력 문장, 맵의 각 원소는 밝을수록 값이 크다(1.0에 가깝다).



출력 1983에 입력 1983을 주목함이 그림으로 보임

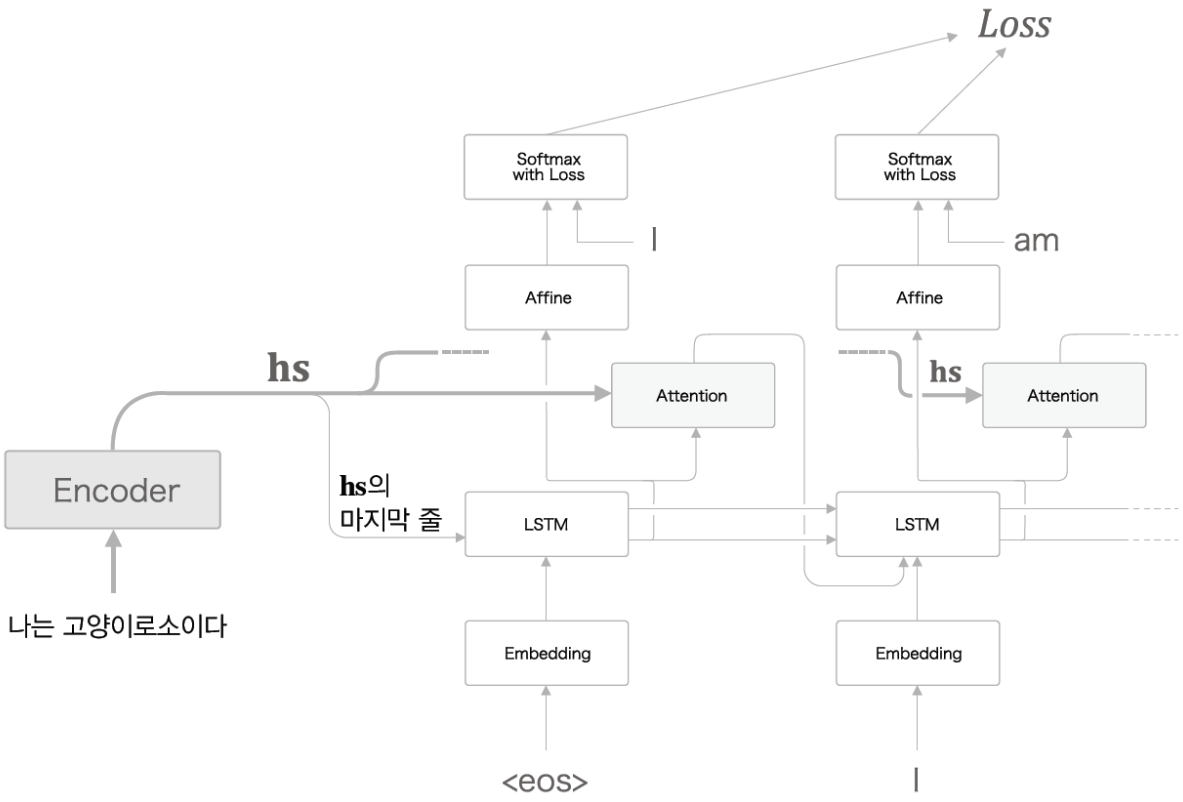
8.4 어텐션에 관한 남은 이야기

그림 8-30 양방향 LSTM으로 인코딩하는 예(LSTM 계층을 간략화하여 그림)



은닉벡터는 좌우 양쪽에서 얻음

그림 8-32 Attention 계층의 다른 사용 예(문헌 [48]을 참고하여 단순화한 신경망 구성)



affine이 아니라 LSTM이 주목을 사용

그림 8-33 3층 LSTM 계층을 사용한 어텐션을 갖춘 seq2seq

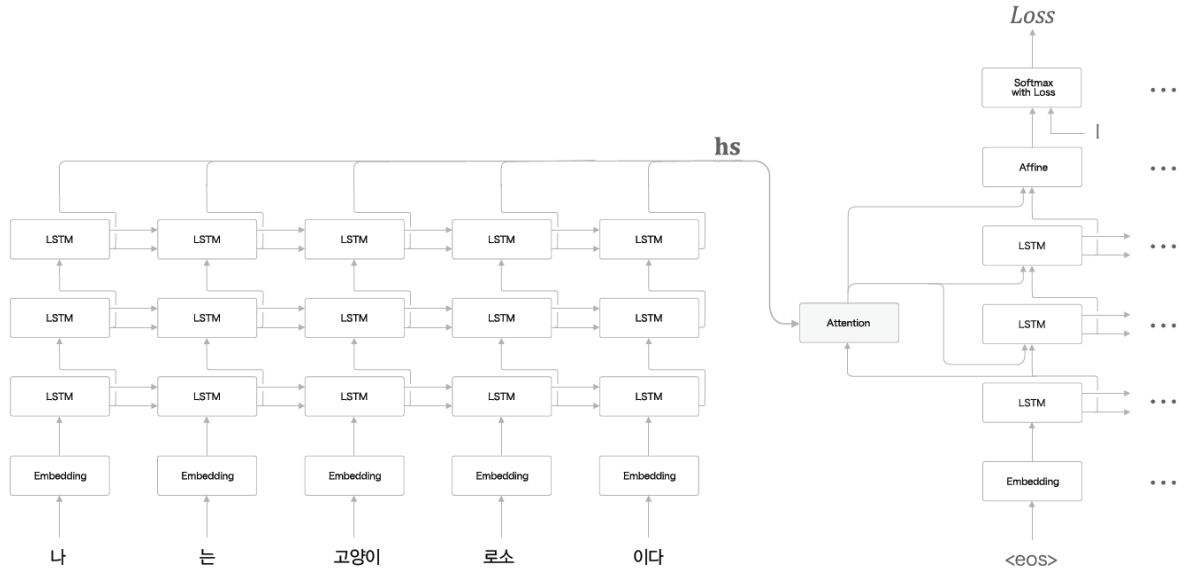
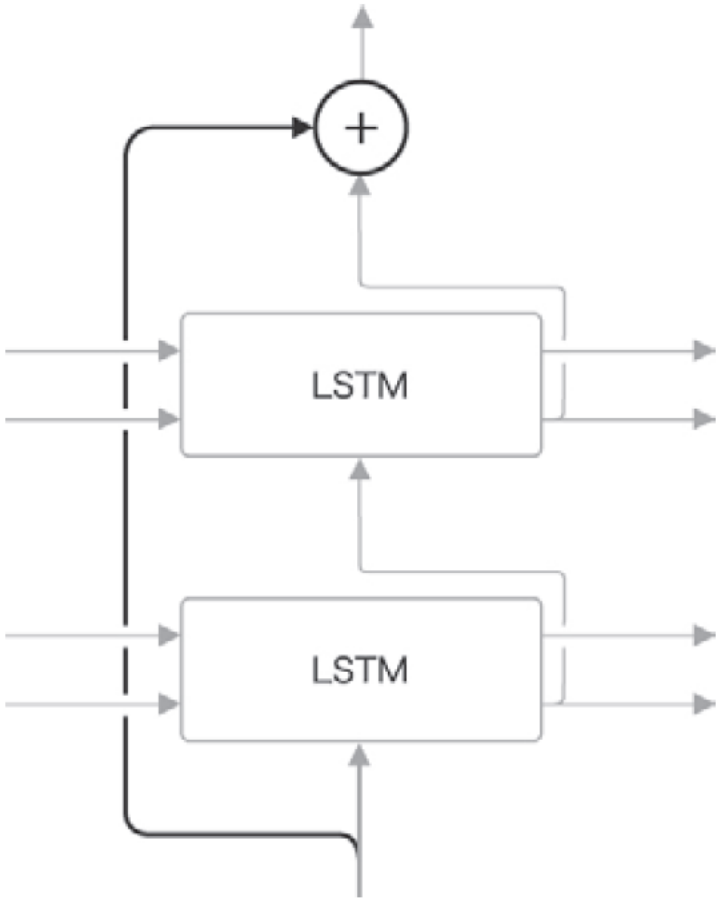


그림 8-34 LSTM 계층의 skip 연결 예

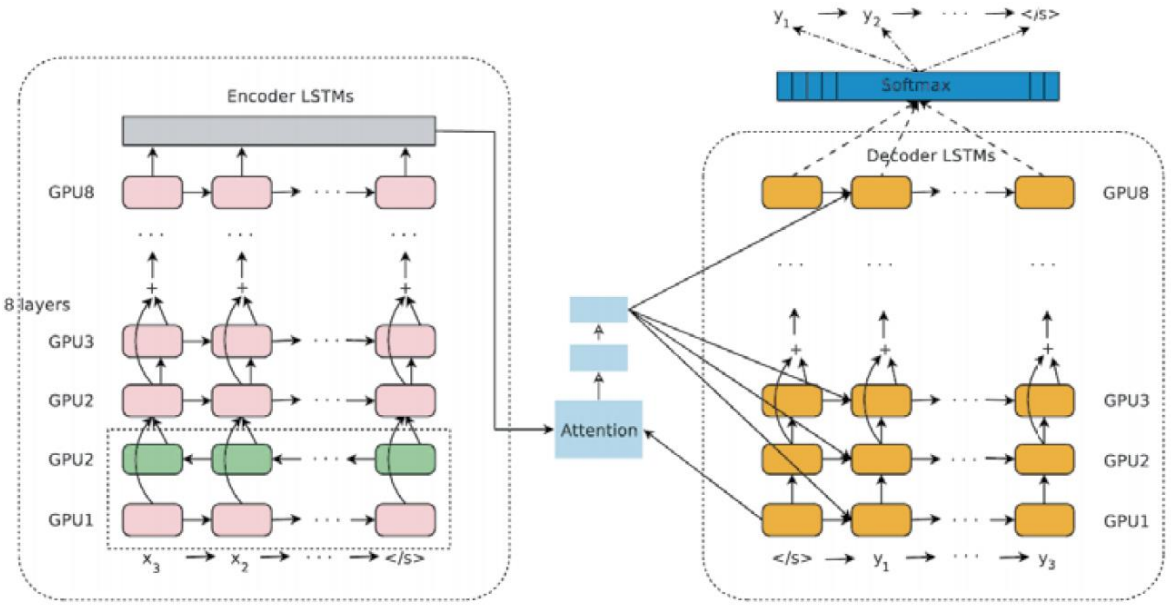


기울기 소실/폭발 완화

8.5 어텐션 응용

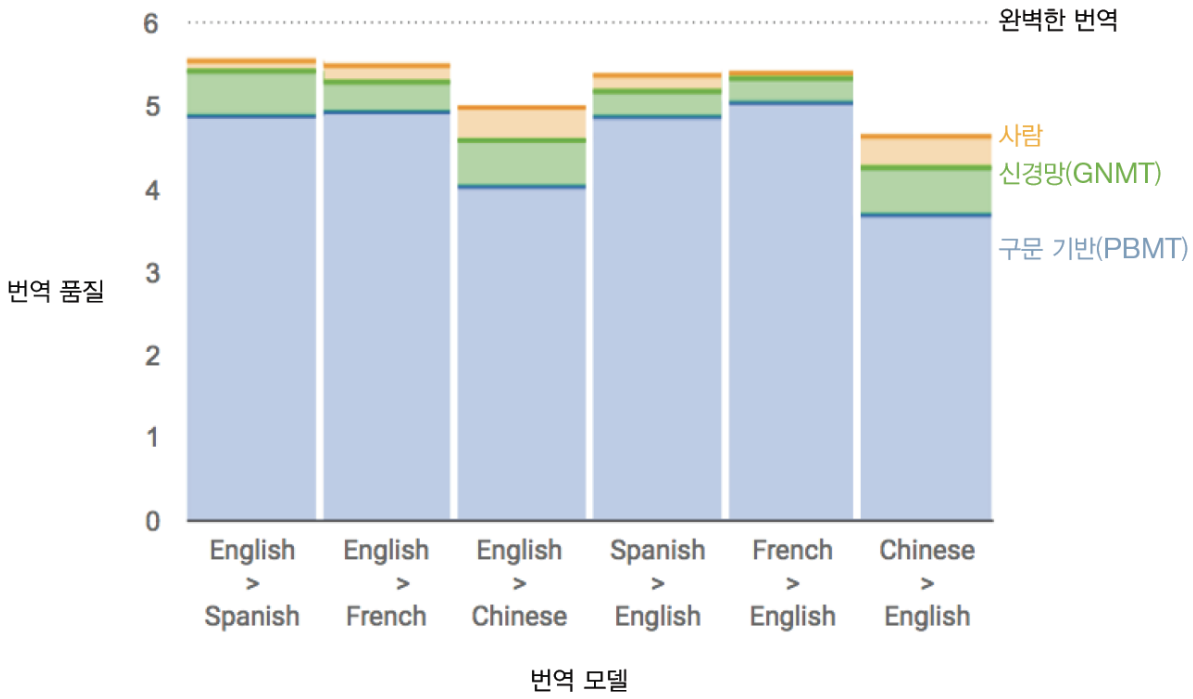
Google Neural Machine Translation

그림 8-35 GNMT의 계층 구성(문헌 [50]에서 발췌)



LSTM 다층화/양방향, skip 연결, 다수 GPU 분산학습

그림 8-36 GNMT의 정확도 평가: 세로축은 번역 품질이며, 사람이 0~6점으로 평가함(문헌 [51]에서 발췌)



100개의 GPU로 6일 학습으로 한 개의 모델 → 8개 모델 확장중

8.5.2 transformer

RNN은 병렬화 어려워 CNN으로 연구: self-attention이용

그림 8-38 트랜스포머의 계층 구성(문헌 [52]를 참고로 단순화한 모델)

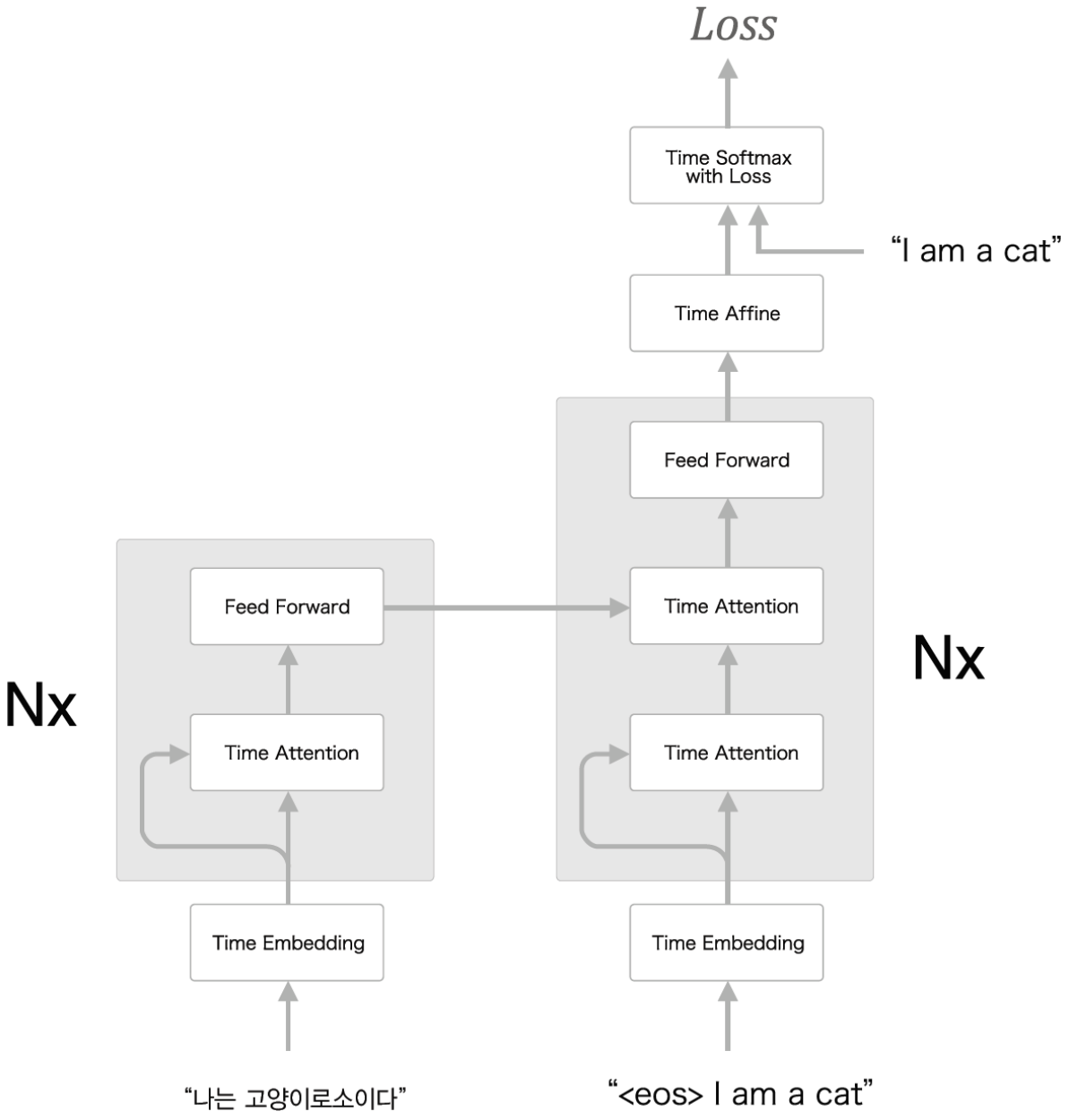
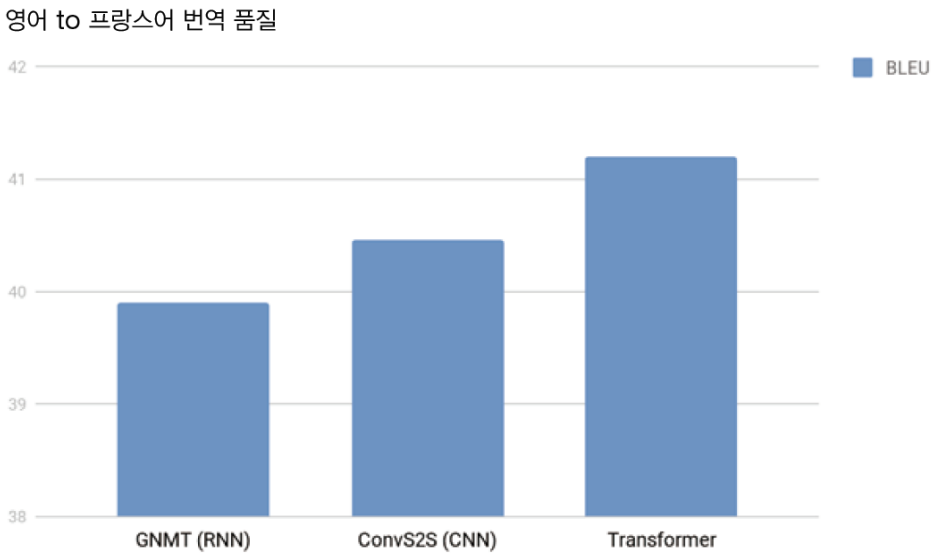


그림 8-39 벤치마크용 번역 데이터인 WMT를 사용하여 '영어 to 프랑스어' 번역 정확도를 평가한 결과: 세로축은 번역 정확도 척도인 BLEU 점수이며, 높을수록 좋다. (문헌 [53]에서 발췌)

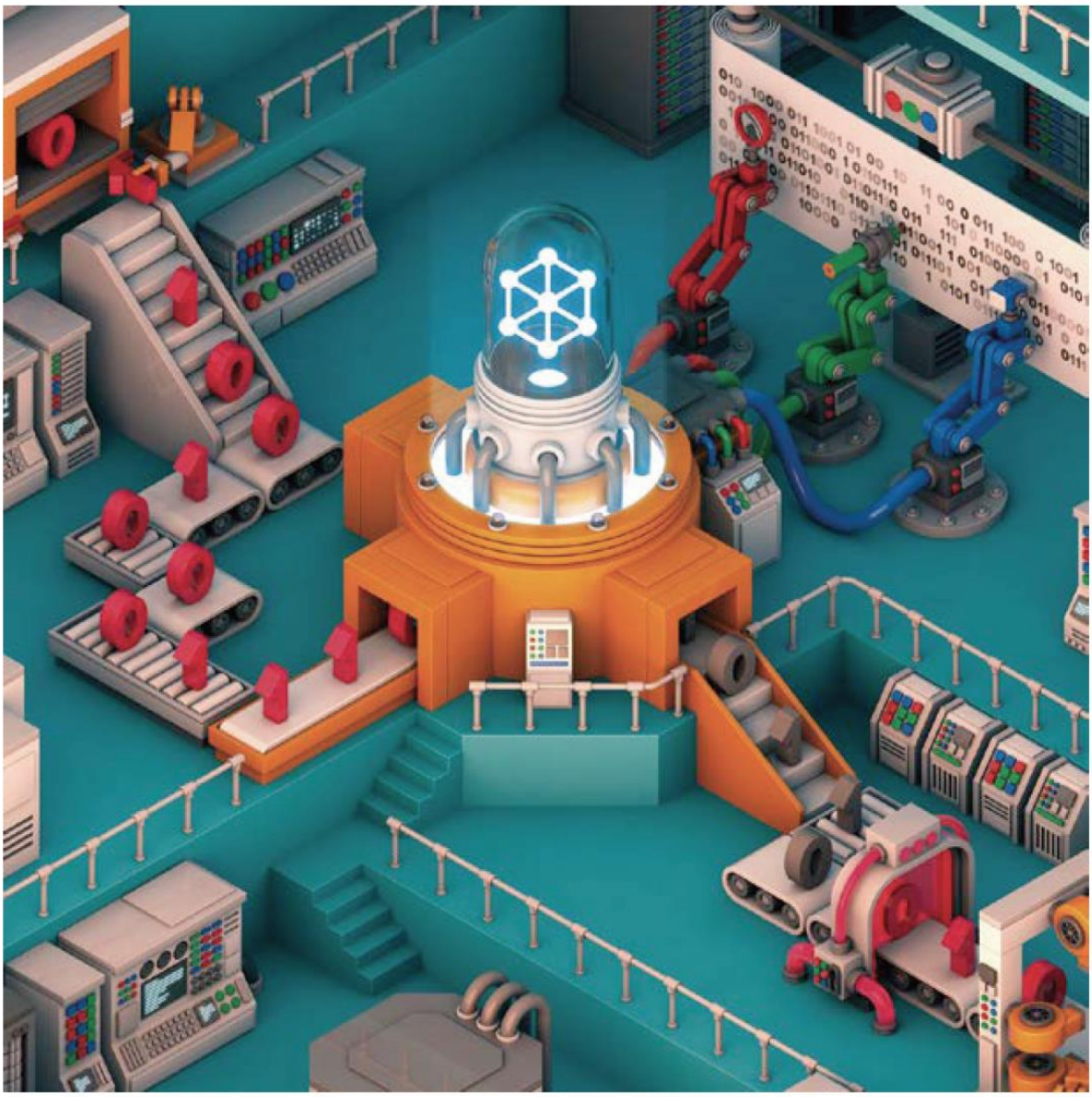


어텐션을 RNN에 적용하지 않고 RNN을 대체

8.5.3 Neural Turing Machine

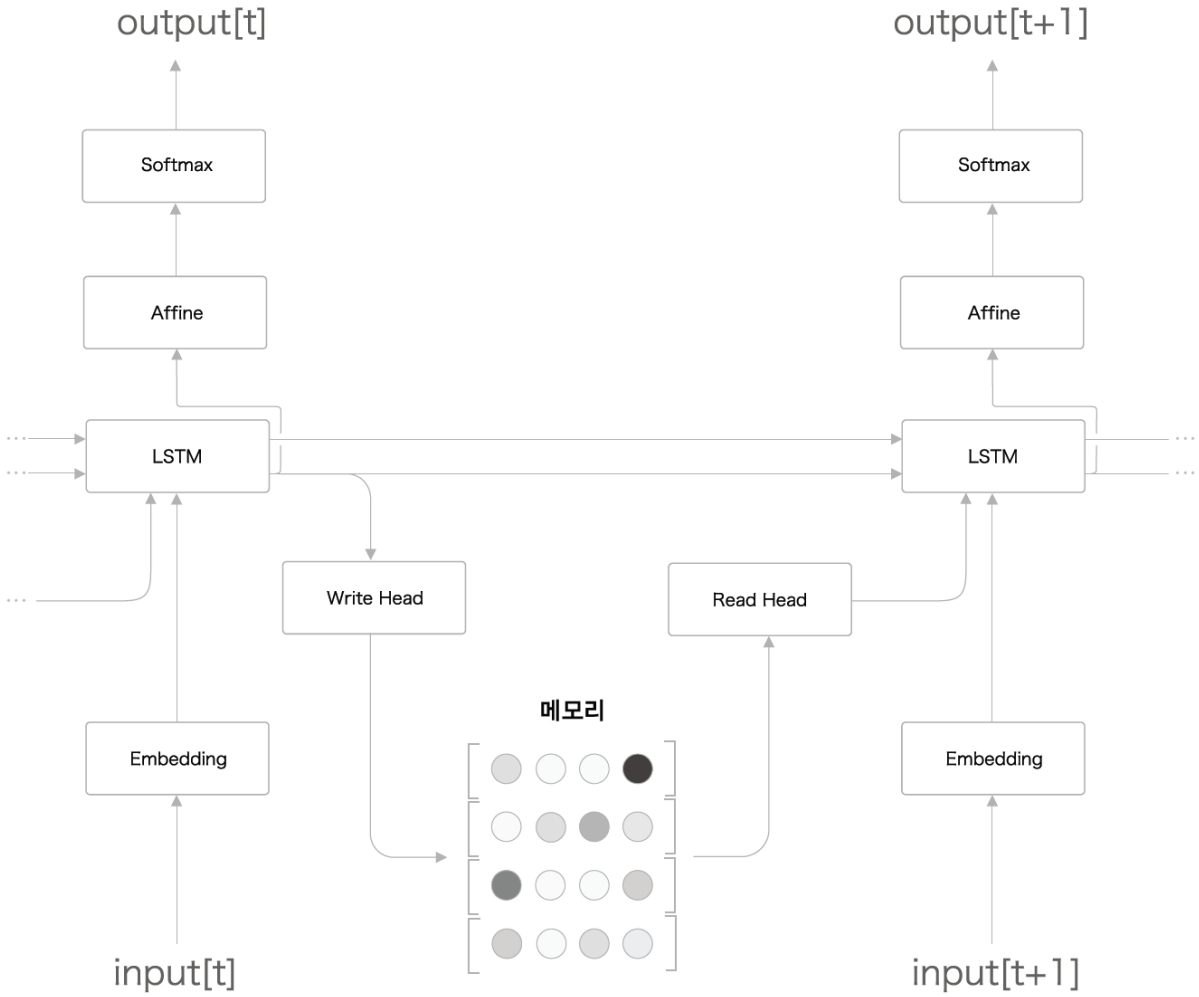
RNN의 외부 메모리에 attention을 통해 필요한 정보를 읽거나 씀

그림 8-40 NTM을 시각화한 그림(문헌 [57]에서 발췌)



돋보기, 펜, 지우개를 보라

그림 8-41 NTM의 계층 구성: 메모리 쓰기와 읽기를 수행하는 Write Head 계층과 Read Head 계층이 새로 등장



컨텐츠 기반 어텐션 : 기존 방법

메모리 위치 기반 어텐션 : 생략 1차원 합성곱 연산

외부메모리를 사용함으로써 긴 시계열 문제도 해결중

---------- 끝 ----------