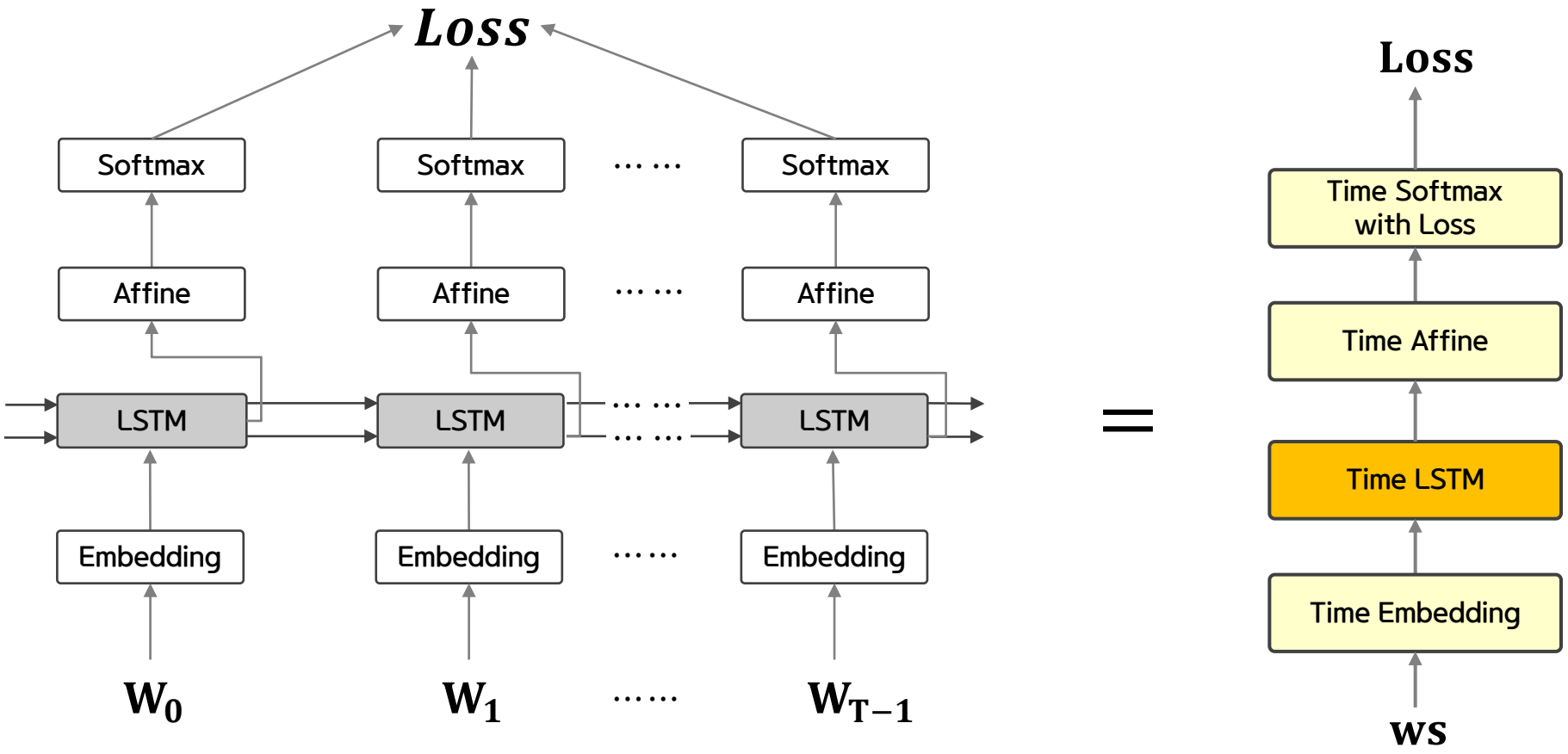
7. RNN을 사용한 문장 생성

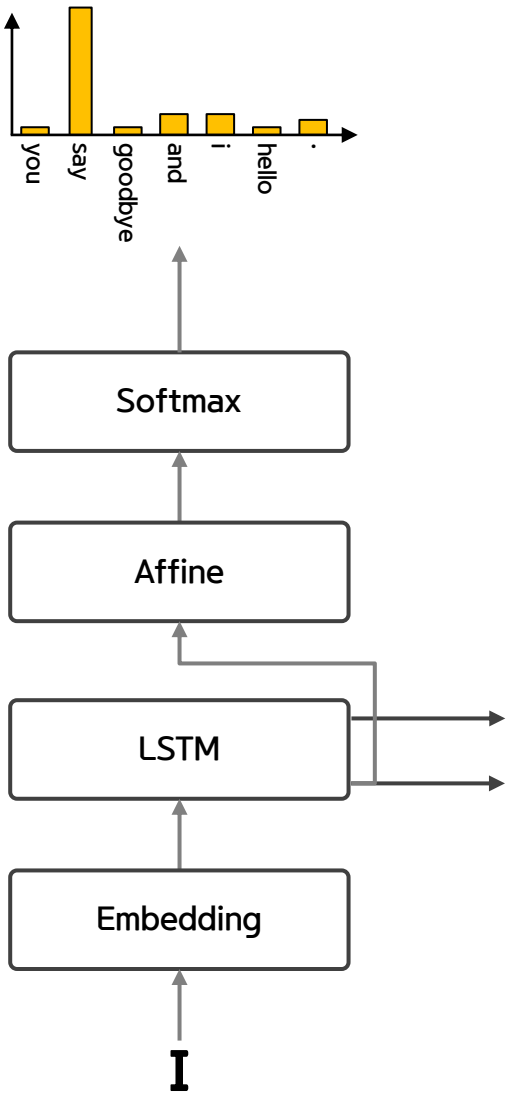
RNN 을 사용한 문장 생성의 순서

앞 장에서는 LSTM 계층을 이용하여 언어 모델을 구현했는데,
그 모델의 신경망 구성은 다음 그림처럼 생겼다.
그리고 시계열 데이터를 T개분 만큼 모아 처리하는 Time LSTM 과 Time Affine 계층 등을 만들었다.

오른쪽은 시계열 데이터를 한꺼번에 처리하는 Time계층을 사용했고, 왼쪽은 같은 구성을 펼친 모습



언어 모델은 다음에 출현할 단어의 확률분포를 출력한다.

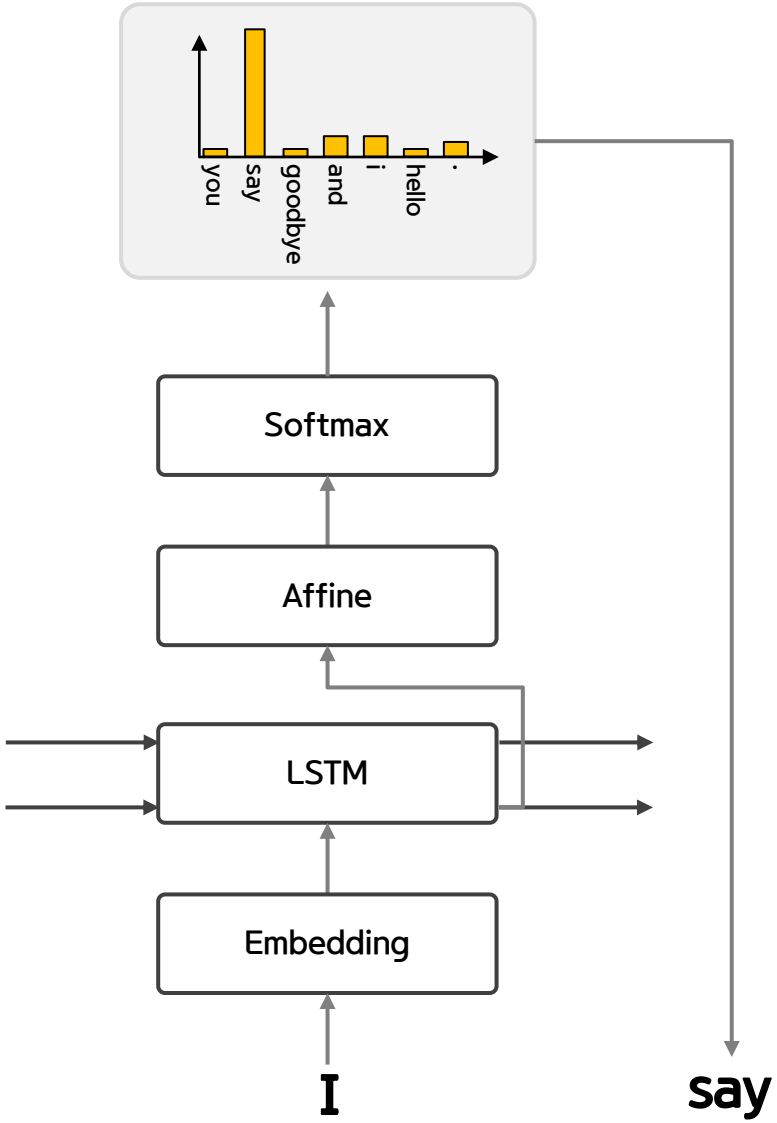


이제 언어 모델에게 문장을 생성시키는 순서를 설명하겠다.
이번에도 친숙한 "you say goodbye and I say hello."라는
말뭉치로 학습한 언어 모델을 예로 생각해보겠다.

이 학습된 언어 모델에 "I"라는 단어를 입력으로 주면 어떻게 될까?
그러면 이 언어 모델은 다음 그림과 같은 확률 분포를 출력한다고 한다.

RNN 을 사용한 문장 생성의 순서

확률분포대로 단어를 하나 샘플링한다.



언어 모델은 지금까지 주어진 단어들 에서 다음에 출현하는
단어의 확률 분포를 출력한다.
이 결과를 기초로 다음 단어를 새로 생성하려면 어떻게 해야 할까?

첫 번째로, 확률이 가장 높은 단어를 선택하는 방법을 떠올릴 수 있다.
확률이 가장 높은 단어를 선택할 뿐이므로 결과가 일정하게
정해지는 결정적인 방법이다. (?)

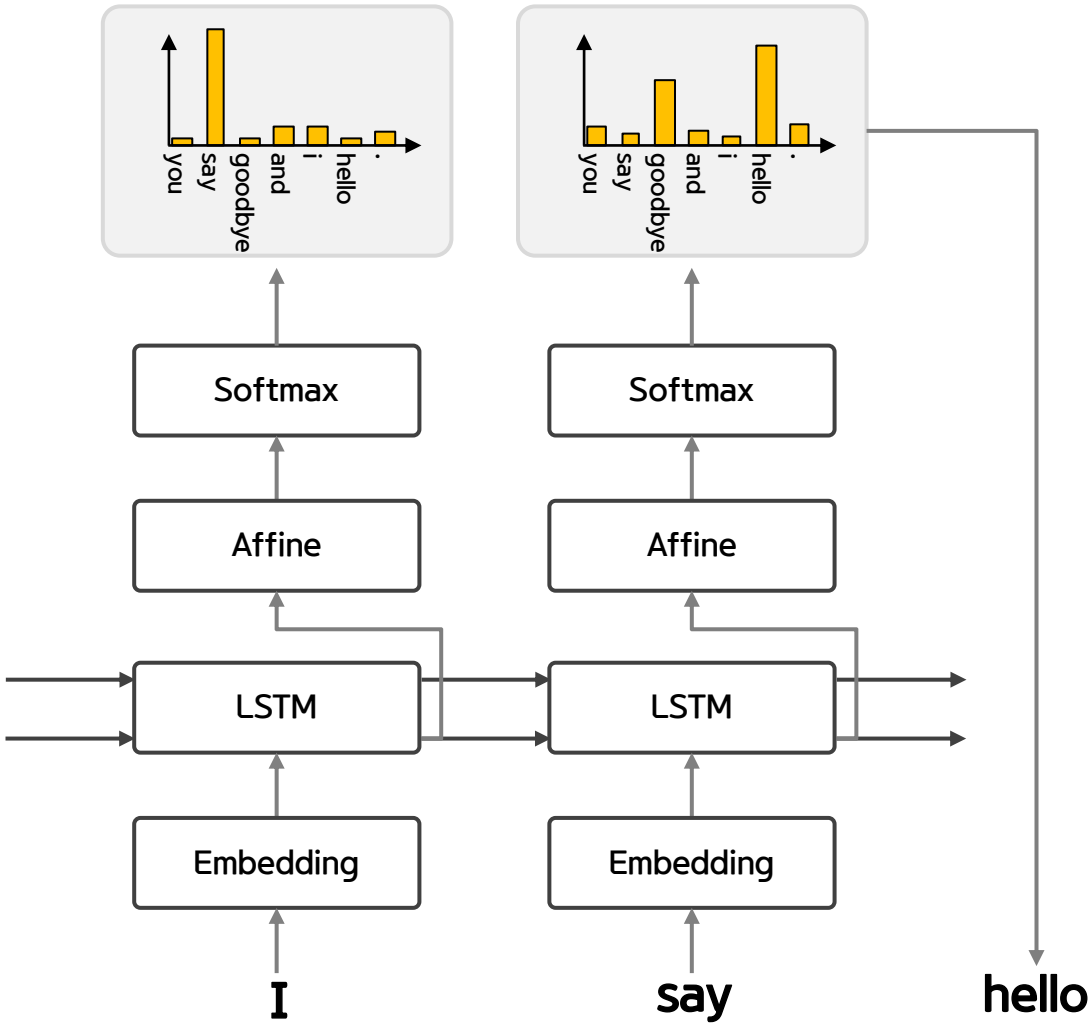
또한, 확률적으로 선택하는 방법도 생각할 수 있다.
각 후보 단어의 확률에 맞게 선택하는 것으로, 확률이 높은 단어는
선택되기 쉽고, 확률이 낮은 단어는 선택되기 어려워진다.

이 방식에서는 선택되는 단어(샘플링 단어)가 매번 다를 수 있다.

우리는 매번 다른 문장을 생성하도록 하겠다.
그 편이 생성되는 문장이 다양해 져서 재미있을 것이다.

RNN 을 사용한 문장 생성의 순서

확률분포 출력과 샘플링을 반복한다.



그러면 계속해서 두 번째 단어를 샘플링해보자.
즉, 방금 생성한 단어인 say를 언어 모델에
입력하여 다음 단어의 확률 분포를 얻는다.

그런 다음 그 확률분포를 기초로 다음에
출현할 단어를 샘플링 한다.

다음은 이 작업을 원하는 만큼 반복한다.
그러면 새로운 문장을 생성할 수 있다.

여기에서 주목할 것은 이렇게 생성한 문장은
훈련 데이터에는 존재하지 않는, 말 그대로 새로운
생성된 문장이라는 것이다.

왜냐하면 언어 모델은 훈련 데이터를 암기한 것이
아니라, 훈련 데이터에서 사용된 단어의
정렬 패턴을 학습한 것이기 때문이다.
만약 언어 모델이 말뭉치로부터 단어의 출현 패턴을
올바르게 학습할 수 있다면, 그 모델이 새로
생성하는 문장은 우리 인간에게도 자연스럽고
의미가 통하는 문장일 것으로 기대할 수 있다.

※ 코드 참조

더 좋은 문장으로

좋은 언어 모델이 있으면 좋은 문장을 기대할 수 있다.
앞 장에서 더 좋은 언어 모델을 BetterRnnlm 라는 클래스로 구현했다.
여기에 문장 생성 기능을 추가하겠다.

이 모델을 한 단계 더 개선하고 한층 더 큰 말뭉치를 사용하면 더 자연스러운 문장을 생성해줄 것이다.

※ 코드 참조

7. RNN을 사용한 문장 생성

7.1 언어 모델을 사용한 문장 생성

7.2 seq2seq

7.3 seq2seq 구현

7.4 seq2seq 개선

7.5 seq2seq를 이용하는 애플리케이션

시계열 데이터는 많다.
언어 데이터, 음성 데이터, 동영상 데이터는 모두 시계열 데이터이다.

그리고 이러한 시계열 데이터를 또 다른 시계열 데이터로 변환하는 문제도 숱하게 생각할 수 있다.
예컨대 기계 번역이나 음성 인식을 들 수 있다.

그 외에도 챗봇처럼 대화하는 애플리케이션이나 컴파일러처럼
소스 코드를 기계어로 변환하는 작업도 생각해볼 수 있다.

이처럼 입력과 출력이 시계열 데이터인 문제는 아주 많다.

지금부터 우리는 시계열 데이터를 다른 시계열 데이터로 변환하는 모델을 생각해볼 것이다.

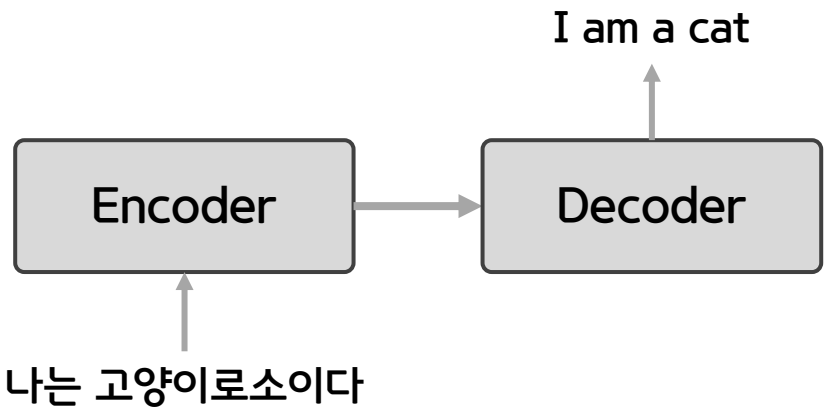
이를 위한 기법으로 여기에서는 2개의 RNN을 이용하는 seq2seq 라는 방법을 살펴보겠다.

seq2seq 의 원리

seq2seq 를 Encoder Decoder 모델이라고도 한다.
여기에는 2개의 모듈, Encoder 와 Decoder 가 등장한다.
문자 그대로 Encoder 는 입력 데이터를 인코딩(부호화)하고
Decoder 는 인코딩된 데이터를 디코딩(복호화)한다.

그럼 seq2seq 의 구조를 구체적인 예를 들어 설명하겠다.
우리말을 영어로 번역하는 예를 살펴보자.
"나는 고양이로소이다" 문장을 "I am a cat"으로 번역해보자.

Encoder와 Decoder가 번역을 수행하는 예



seq2seq 의 원리

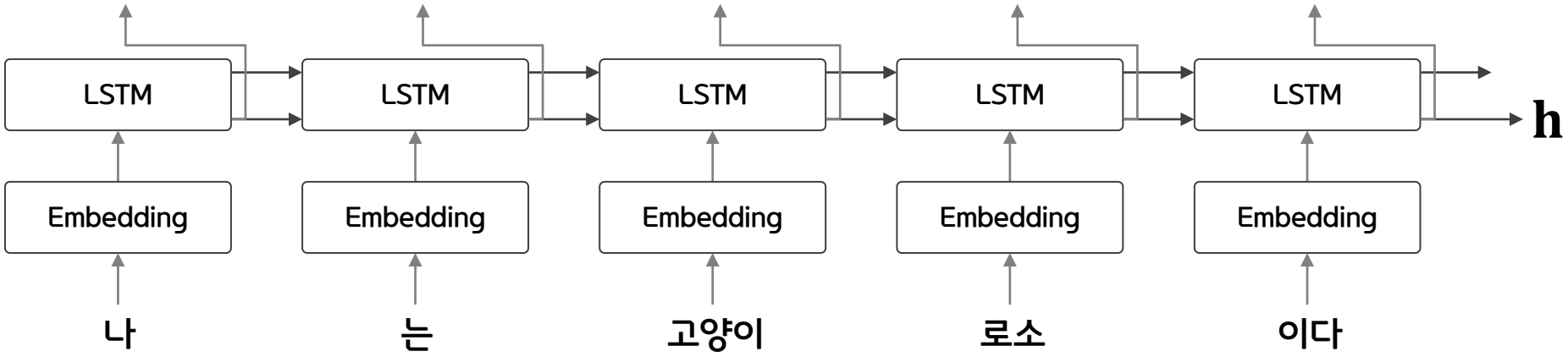
Encoder 가 "나는 고양이로소이다"라는 출발어 문장을 인코딩한다.
이어서 그 인코딩한 정보를 Decoder 에 전달하고, Decoder 가 도착어 문장을 생성한다.
이때 Encoder 가 인코딩한 정보에는 번역에 필요한 정보가 조밀하게 응축되어 있다.
Docder 는 조밀하게 응축된 정보를 바탕으로 도착어 문장을 생성하는 것이다.

이것이 seq2seq 의 전체 그림이다.
Encoder 와 Decoder 가 협력하여 시계열 데이터를 다른 시계열 데이터로 변환하는 것이다.
그리고 Encoder 와 Decoder 로 RNN을 사용할 수 있다.

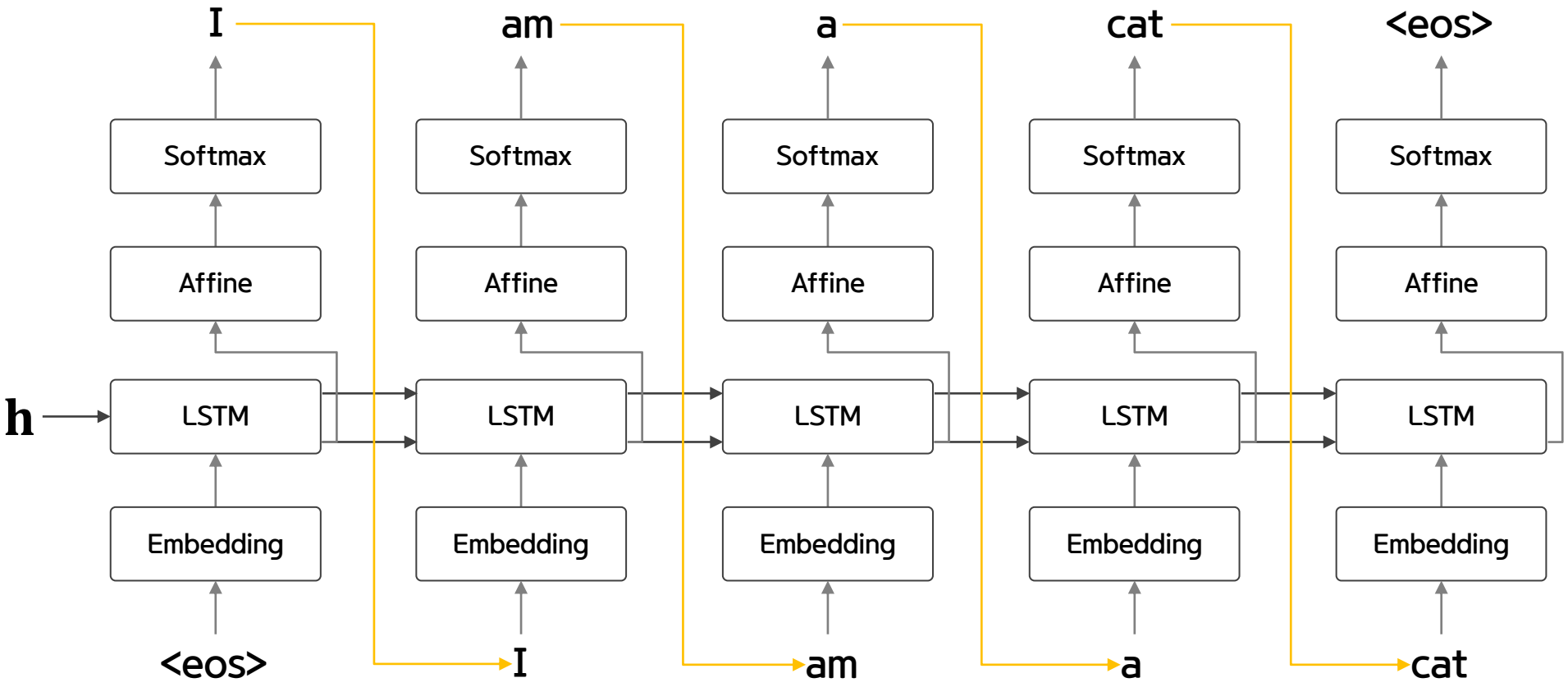
이제 전체 과정을 자세히 살펴보자.
우선 Encoder 의 처리에 집중해보자.

Encoder의 계층을 다음과 같이 구성된다.

Encoder를 구성하는 계층



Decoder를 구성하는 계층



seq2seq 의 원리

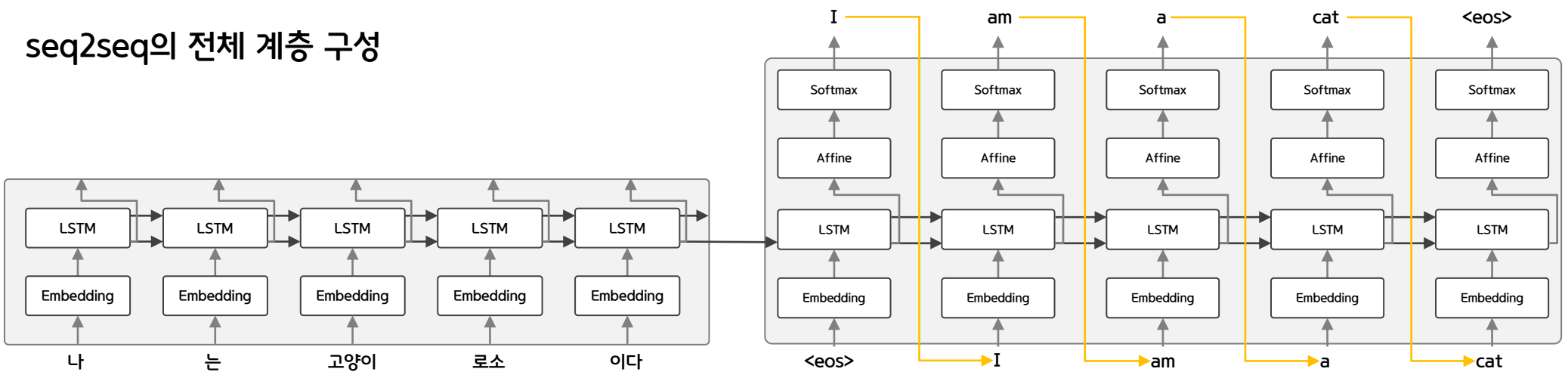
seq2seq 는 LSTM 두 개로 구성된다.
(Encoder 의 LSTM, Decoder 의 LSTM)

이때 LSTM 계층의 은닉 상태가 Encoder 와 Decoder 를 이어주는 가교가 된다.

순전파 때는 Encoder 에서 인코딩된 정보가 LSTM 계층의 은닉 상태를 통해 Decoder 에 전해진다.

그리고 seq2seq 의 역전파 때는 이 가교를 통해 기울기가 Decoder 로부터 Encoder 로 전해진다.

seq2seq의 전체 계층 구성



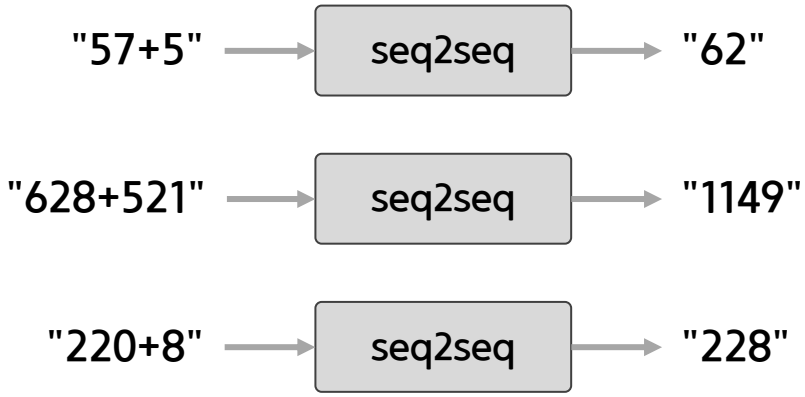
시계열 데이터 변환용 장난감 문제

지금부터, 다룰 문제에 관해 설명하겠다.

우리는 시계열 변환 문제의 예로 더하기를 다루었다.
구체적으로는 57+5와 같은 문자열을 seq2seq 에 건네면 62라는 정답을 내놓도록 학습시킬 것이다.

참고로 이와 같이 머신러닝을 평가하고자 만든 간단한 문제를 장난감 문제라고 한다.

seq2seq에 덧셈 예제들을 학습시킨다.



시계열 데이터 변환용 장난감 문제

셈은 우리 인간에게는 쉬운 문제이다.
그러나 seq2seq 는 덧셈에 대해 (정확하게 덧셈의 논리에 대해) 아무것도 모른다.
seq2seq 는 덧셈의 샘플로부터 거기서 사용되는 문자의 패턴을 학습한다.
과연 이런 식으로 해서 덧셈의 규칙을 올바르게 학습할 수 있는 걸까?

우리는 지금까지 word2vec 이나 언어 모델 등에서 문자를 단어 단위로 분할해왔다.
하지만 문장을 반드시 단어로 분할해야 하는 건 아니다.
실제로 이번 문제에서는 단어가 아닌 문자 단위로 분할한다.
문자 단위 분할이란, 예컨대 57+5 가 입력되면, [5,7,+,5] 라는 리스트로 처리하는 걸 말한다.

가변 길이 시계열 데이터

우리는 덧셈을 문자 리스트로써 다루기로 했다.
이때 주의할 점은 덧셈 문장(5,7,+,5)이나 그 대답의 문자 수(6,2)가 문제마다 다르다는 것이다.

이처럼 이번 덧셈 문제에서는 샘플마다 데이터의 시간 방향 크기가 다르다.
가변 길이 시계열 데이터를 다룬다는 뜻이다.
따라서 신경망 학습 시 미니배치 처리를 하려면 무언가 추가 노력이 필요하다.

가변 길이 시계열 데이터를 미니배치로 학습하기 위한 가장 단순한 방법은 패딩을 사용하는 것이다.

패딩이란 원리의 데이터에 의미 없는 데이터를 채워 모든 데이터의 길이를 균일하게 맞추는 기법이다.

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	

이번 문제에서는 0~999 사이의 숫자 2개만 더하기로 하겠다.
따라서 +까지 포함하면 입력의 최대 문자 수는 7,
자연스럽게 덧셈 결과는 최대 4문자이다.

더불어 정답 데이터에도 패딩을 수행해 모든 샘플 데이터의 길이를 통일한다.
그리고 질문과 정답을 구분하기 위해 출력 앞에 구분자로 _를 붙이기로 한다.
그 결과 출력 데이터는 총 5문자로 통일한다.
참고로, 이 구분자는 Decoder 에 문자열을 생성하라고 알리는 신호로 사용된다. (?)

이처럼 패딩을 적용해 데이터 크기를 통일시키면 가변 길이 시계열 데이터도 처리할 수 있다.
그러나 원리는 존재하지 않던 패딩용 문자까지 seq2seq 가 처리하게 된다.
따라서 패딩을 적용해야 하지만 정확성이 중요하다면 seq2seq 에 패딩 전용 처리를 추가해야 한다.

예컨대 Decoder 에 입력된 데이터가 패딩이라면 손실의 결과에 반영하지 않도록 해야 한다.
Softmax with Loss 계층에 마스크 기능을 추가해 해결할 수 있다.

한편 Encoder 에 입력된 데이터가 패딩이라면 LSTM 계층이 이전 시각의 입력을 그대로 출력하게 한다.
즉, LSTM 계층은 마치 처음부터 패딩이 존재하지 않았던 것처럼 인코딩할 수 있다.

이번 장에서는 이해 난이도를 낮추기 위해 패딩용 문자(공백 문자)도 특별히 구분하지 않고
일반 데이터처럼 다루겠다.

'덧셈' 학습 데이터: 공백 문자space 는 회색 가운데 점으로 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
```

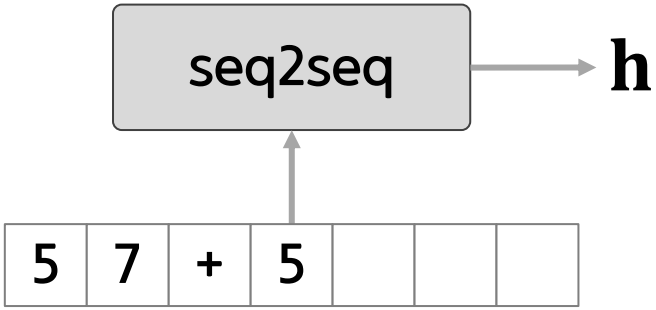
Lines: 50,000 Chars: 650,000 650 KB

※ 코드 참조

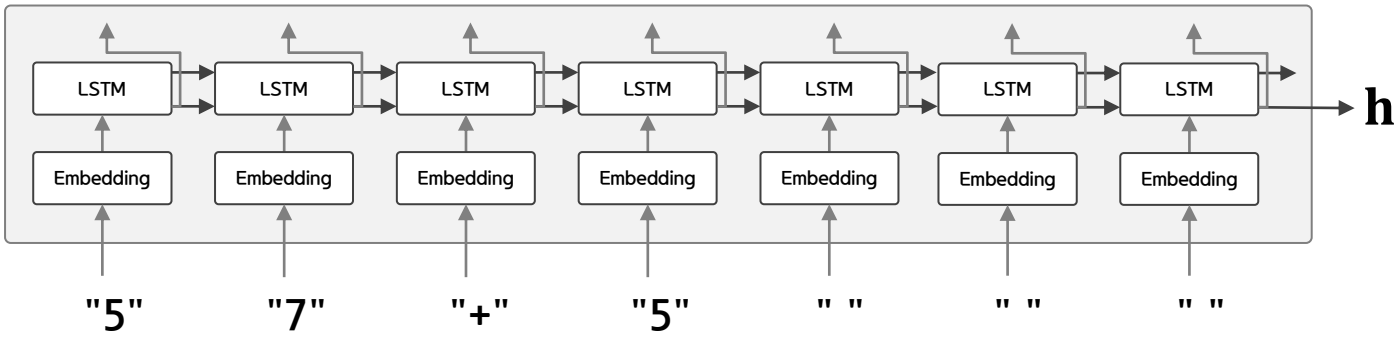
7. RNN을 사용한 문장 생성

Encoder 클래스는 다음 그림처럼 문자열을 받아 벡터 h로 변환한다.

Encoder의 입출력



Encoder의 계층 구성



Encoder 클래스

Encoder 클래스는 Embedding 계층과 LSTM 계층으로 구성된다.

Embedding 계층에서는 문자 ID를 문자 벡터로 변환한다.
그리고 이 문자 벡터가 LSTM 계층으로 입력된다.

LSTM 계층은 시간 방향(오른쪽)으로 은닉 상태와 셀을 출력하고
위쪽으로는 은닉 상태만 출력한다.

이 구성에서 더 위에는 다른 계층이 없으니 LSTM 계층의 위쪽 출력은 폐기된다.
Encoder 에서는 마지막 문자를 처리한 후 LSTM 계층이 은닉 상태 h를 출력한다.

그리고 이 은닉 상태 h가 Decoder 로 전달된다.

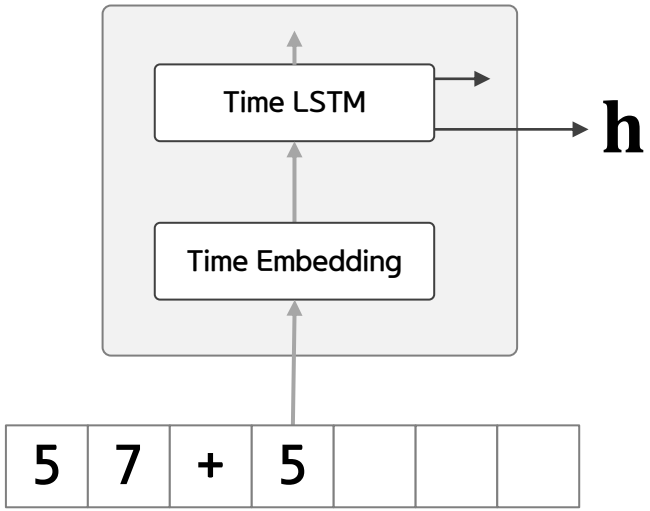
Encoder 에서는 LSTM 의 은닉 상태만을 Decoder 에 전달한다.
LSTM의 셀도 Decoder 에 전달할 수는 있지만,
LSTM 의 셀을 다른 계층에 전달하는 일은 일반적으로 흔치 않다.

LSTM 의 셀은 자기 자신만 사용한다는 전제로 설계되었기 때문이다.

Encoder 클래스

그런데 우리는 시간 방향을 한꺼번에 처리하는 계층을 Time LSTM 계층이나
Time Embedding 계층으로 구현했다.
이러한 Time 계층을 이용하면 Encoder 는 다음 그림처럼 된다.

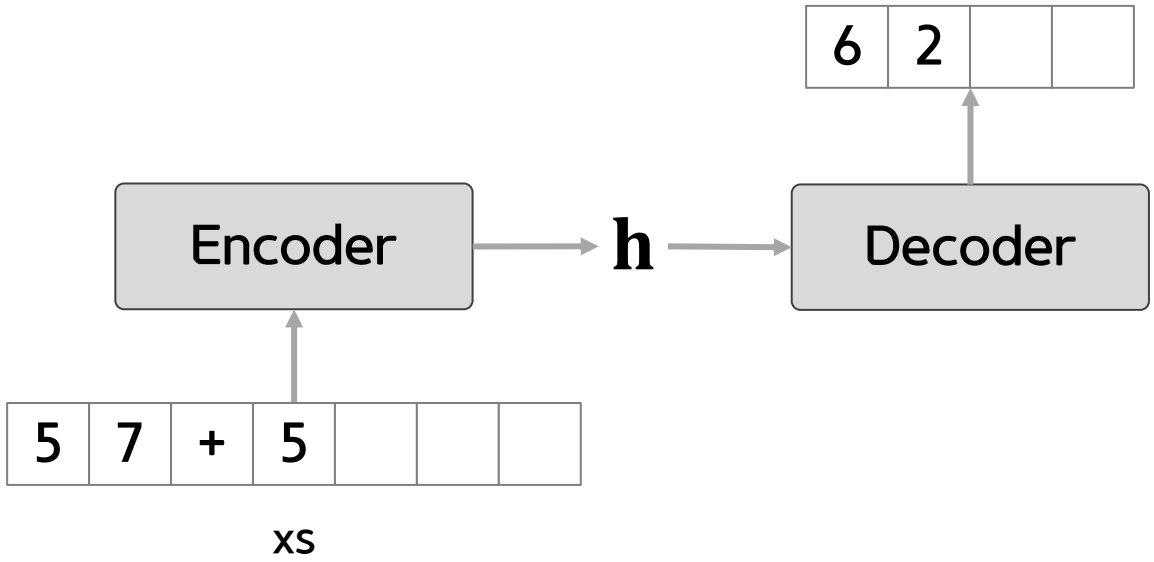
Encoder를 Time 계층으로 구현한다.



※ 코드 참조

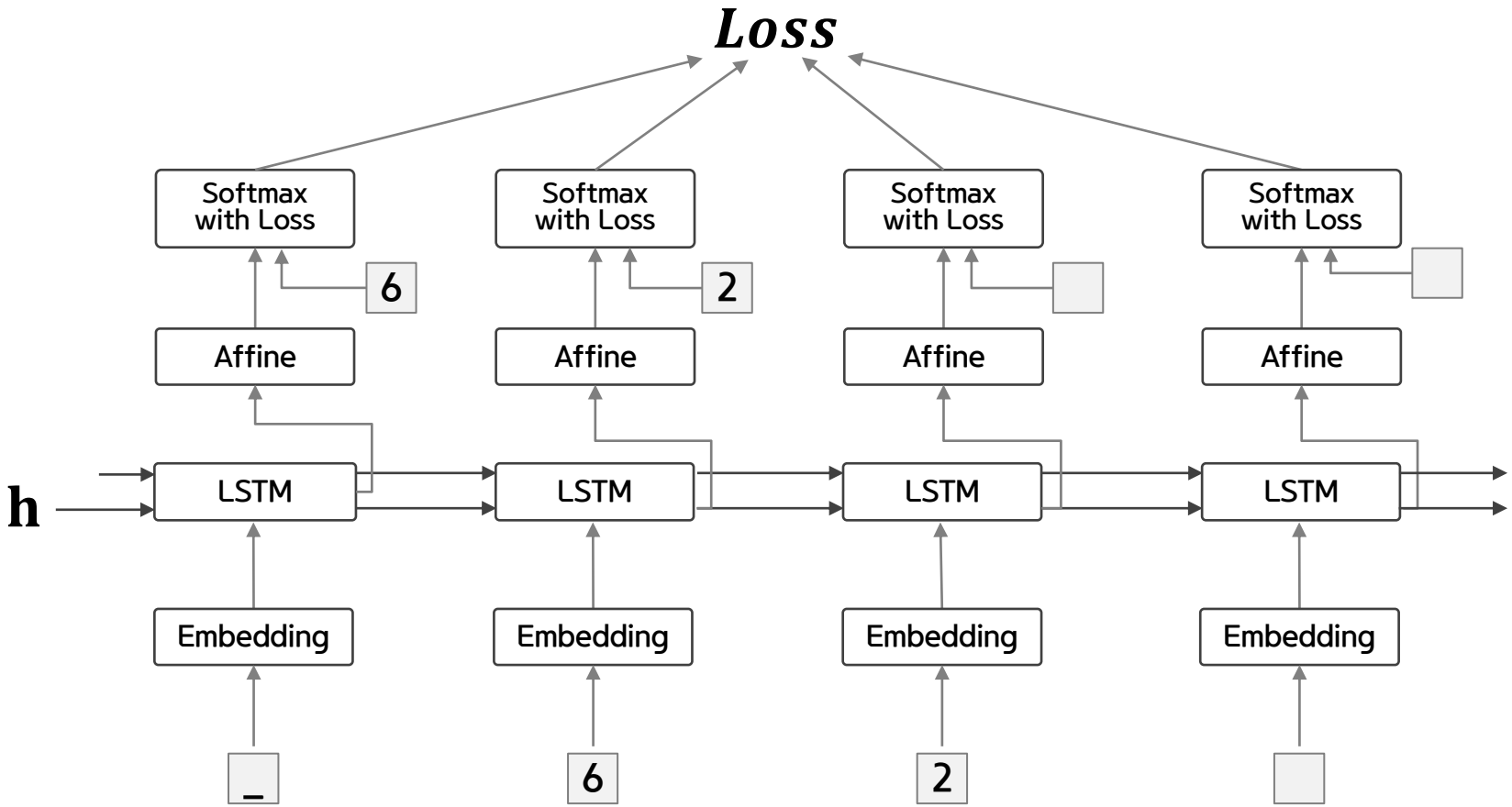
Decoder 클래스는 Encoder 클래스가 출력한 h를 받아 목적으로 하는 다른 문자열을 출력한다.

Encoder와 Decoder



Decoder 는 RNN으로 구현할 수 있다.
Encoder 와 마찬가지로 LSTM 계층을 사용하면 되며,
이때 Decoder 의 계층 구성은 다음 그림과 같다.

Decoder의 계층 구성(학습 시)



RNN으로 문장을 생성할 때 학습 시와 생성 시의 데이터 부여 방법이 다르다.
학습 시는 정답을 알고 있기 때문에 시계열 방향의 데이터를 한꺼번에 보여줄 수 있다.
한편, 추론 시(새로운 문자열을 생성할 때)에는 최초 시작을 알리는 구분 문자(여기서는 _) 하나만 준다.
그리고 그 출력으로부터 문자를 하나 샘플링하여, 그 샘플링 문자를 다음 입력으로 사용하는 과정을 반복한다.

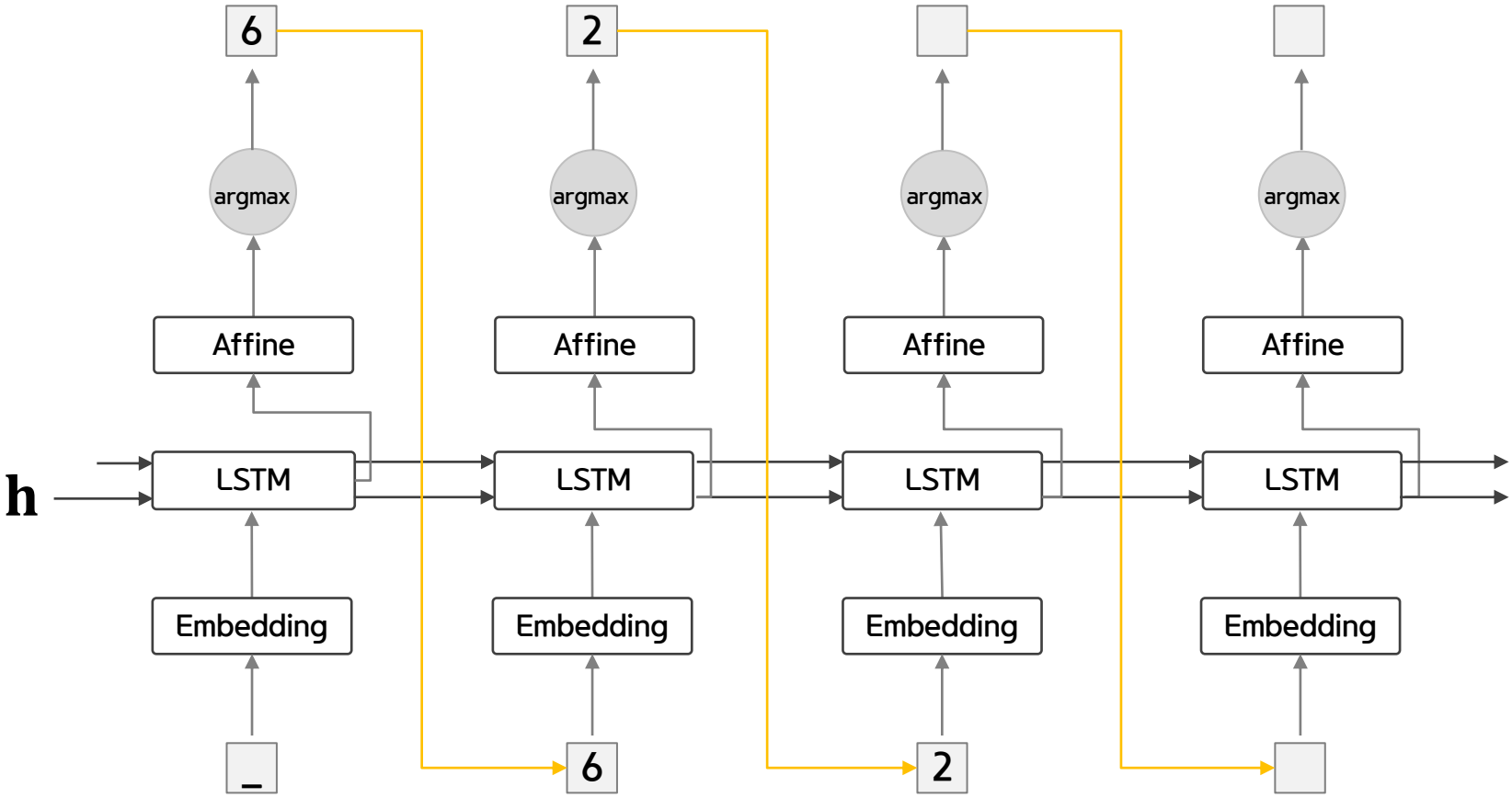
그런데 7.1절에서 문장을 생성할 때는 소프트맥스 함수의 확률분포를 바탕으로 샘플링을 수행했기 때문에
생성되는 문장이 확률에 따라 달라졌다.

이와 달리 이번 문제는 뎃셈이므로 이러한 확률적인 비결정성을 배제하고 결정적인 답을 생성하고자 한다.
그래서 이번에는 점수가 가장 높은 문자 하나만 고르겠다.

즉, 확률적이 아닌 결정적으로 선택한다.

다음 그림은 Decoder 가 문자열을 생성시키는 흐름을 보여준다.

Decoder의 문자열 생성 순서: argmax 노드는 Affine 계층의 출력 중 값이 가장 큰 원소의 인덱스(문자ID)를 반환한다.

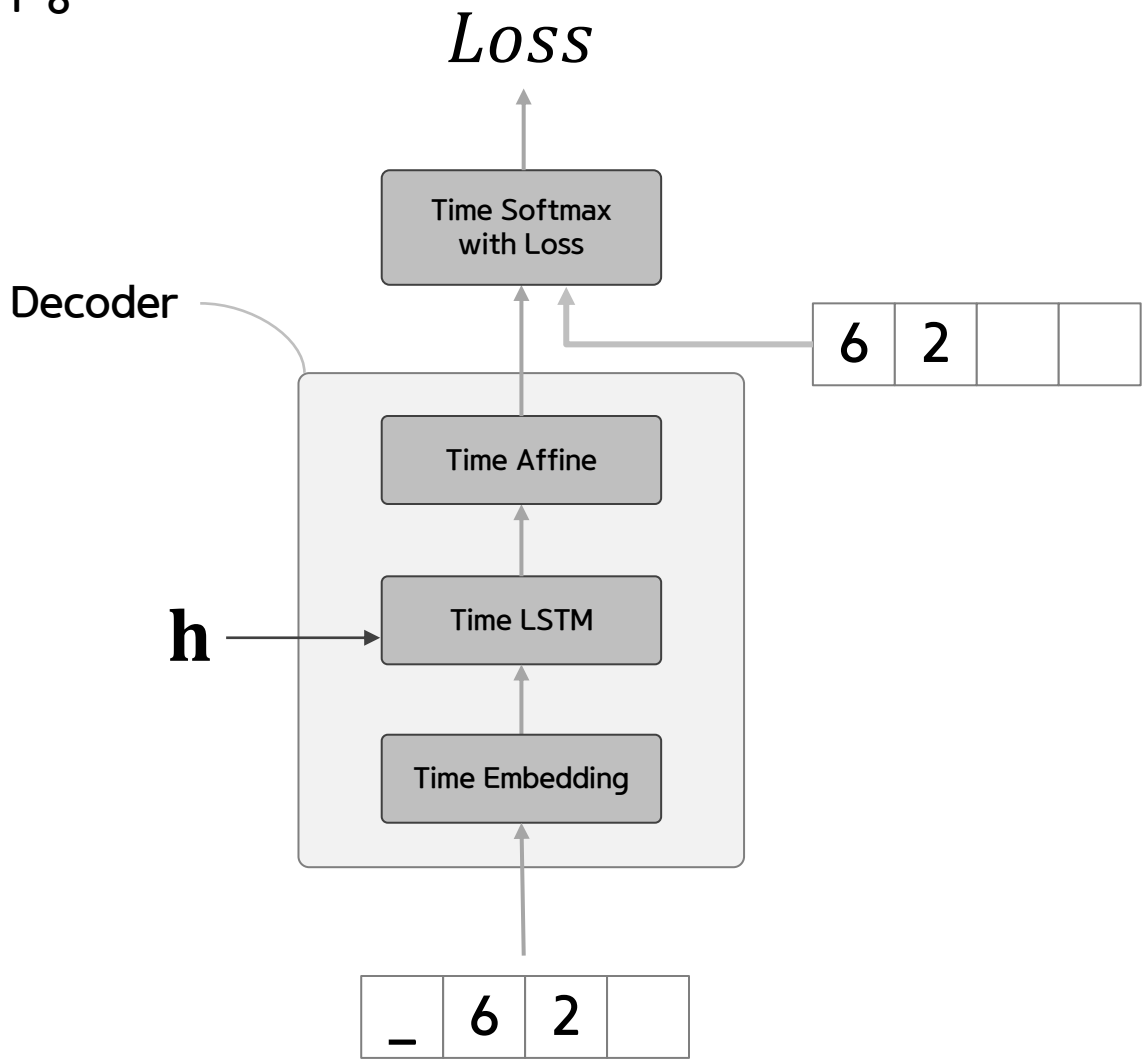


argmax 라는 못 보던 노드가 등장한다.
바로 최댓값을 가진 원소의 인덱스(여기서는 문자ID)를 선택하는 노드이다.
구성은 앞 절에서 본 문장 생성 때의 구성과 같다.
다만 이번에는 Softmax 계층을 사용하지 않고, Affine 계층이 출력하는 점수가 가장 큰 문자ID 를 선택한다.

Softmax 계층은 입력된 벡터를 정규화한다.
이 정규화 과정에서 벡터의 각 원소의 값이 달라지지만, 대소 관계는 바뀌지 않는다.
따라서 위 그림의 경우 Softmax 계층을 생략할 수 있다.

설명한 것처럼 Decoder 에서는 학습 시와 생성 시에 Softmax 계층을 다르게 취급한다.
그러니 Softmax with Loss 계층은 이후에 구현하는 Seq2seq 클래스에서 처리하기로 하고,
Decoder 클래스는 Time Softmax with Loss 계층의 앞 까지만 담당하기로 한다.

Decoder 클래스의 구성



Decoder 클래스

※ 코드 참조

Seq2seq 클래스

Seq2seq 클래스는 Encoder 클래스와 Decoder 클래스를 연결하고,
Time Softmax with Loss 계층을 이용해 손실을 계산한다.

※ 코드 참조

seq2seq 의 학습은 기본적인 신경망의 학습과 같은 흐름으로 이뤄진다.

1. 학습 데이터에서 미니배치를 선택하고
2. 매니배치로부터 기울기를 선택하고
3. 기울기를 사용하여 매개변수를 갱신한다.

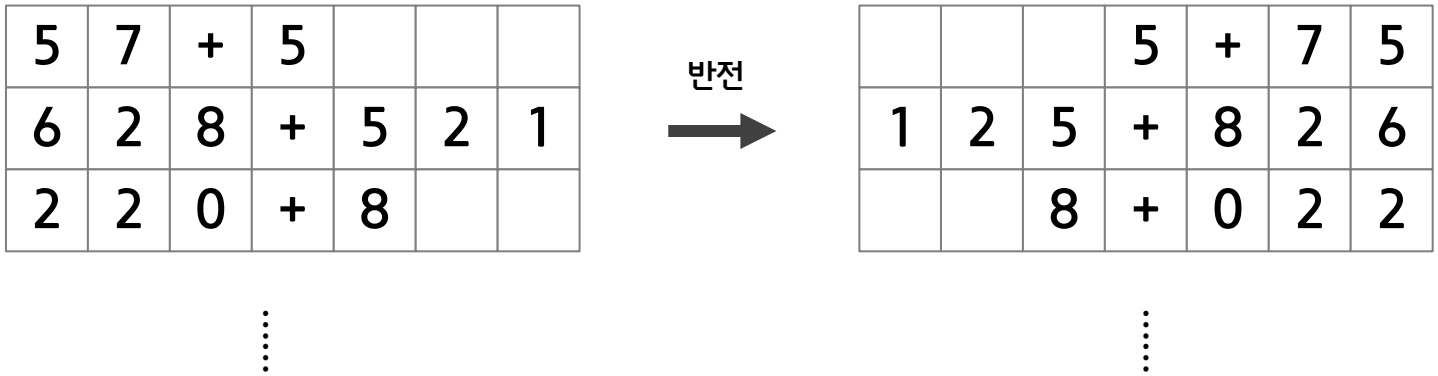
Tainer 클래스를 사용해 이 규칙대로 작업을 수행한다.
매 애폭마다 seq2seq 가 테스트 데이터를 풀게 하여(문자열을 생성하여) 학습 중간중간 정답률을 측정한다.

※ 코드 참조

7. RNN을 사용한 문장 생성

입력 데이터 반전(Reverse)

입력 데이터 반전시키는 예



입력 데이터를 반전시키는 트릭을 사용하면 많은 경우 학습 진행이 빨라져서,
결과적으로 최종 정확도도 좋아진다고 한다.

그럼 실제로 코드를 살펴보자.

```
x_train, x_test = x_train[:, ::-1], x_test[:, ::-1]
```

입력 데이터 반전(Reverse)

물론 데이터를 반전시키는 효과는 어떤 문제를 다루느냐에 따라 다르지만,
대부분의 경우 더 좋은 결과로 이어진다.

그러면 왜 입력 데이터를 반전시키는 것만으로 학습의 진행이 빨라지고 정확도가 향상되는 걸까?
이론적인 것은 잘 모르겠지만, 직관적으로는 기울기 전파가 원활해지기 때문이라고 생각한다.

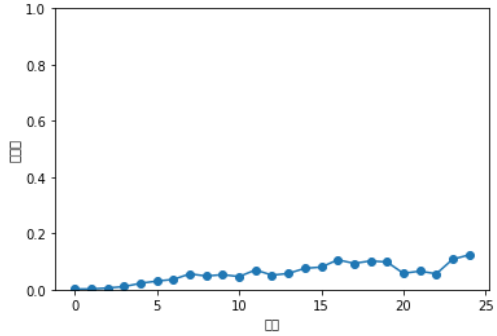
예를 들어 "나는 고양이로소이다"를 "I am a cat"으로 번역하는 문제에서,
'나'라는 단어가 'I'로 변환되는 과정을 생각해보자.
이때 '나'로부터 'I'까지 가려면 '는','고양이','로소','이다' 까지 총 4 단어 분량의 LSTM 계층을 거쳐야 한다.
따라서 역전파 시, 'I'로부터 전해지는 기울기가 '나'에 도달하기까지, 그 먼 거리만큼 영향을 더 받게 된다.

여기서 입력문을 반전시키면, 즉 "이다 로소 고양이 는" 순으로 바꾸면 어떻게 될까?
이제 '나'와 'I'는 바로 옆이 되었으니 기울기가 직접 전해진다.
이처럼 입력 문장의 첫 부분에서는 반전 덕분에 대응하는 변환 후 단어와 가까우므로 (그런 경우가 많아지므로),
기울기가 더 잘 전해져서 학습 효율이 좋아진다고 생각할 수 있다.

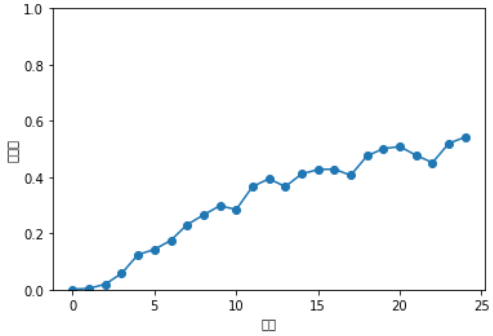
다만, 입력 데이터를 반전해도 단어 사이의 평균적인 거리는 그대로이다.

입력 데이터 반전(Reverse)

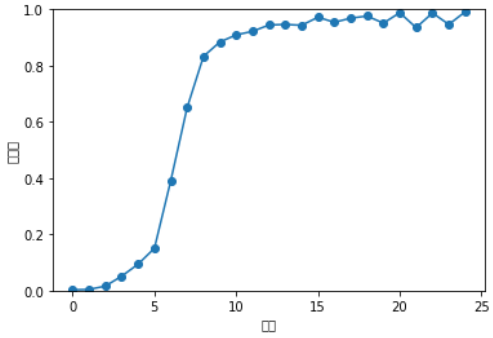

```
...
---
Q 761+292
T 1053
X 1052
---
Q 830+597
T 1427
X 1426
---
Q 26+838
T 864
O 864
---
Q 143+93
T 236
O 236
---
검증 정확도 54.260%
```



reverse 전



reverse 후



peeky 적용 후

엿보기 (Peeky)

이어서 seq2seq 의 두 번째 개선이다.

주제로 곧장 들어가기 전에 seq2seq의 Encoder 동작을 한번 더 살펴보자.

Encoder는 입력 문장(문제 문장)을 고정 길이 벡터 h로 변환한다.
이때 h 안에는 Decoder 에게 필요한 정보가 모두 담겨 있다.

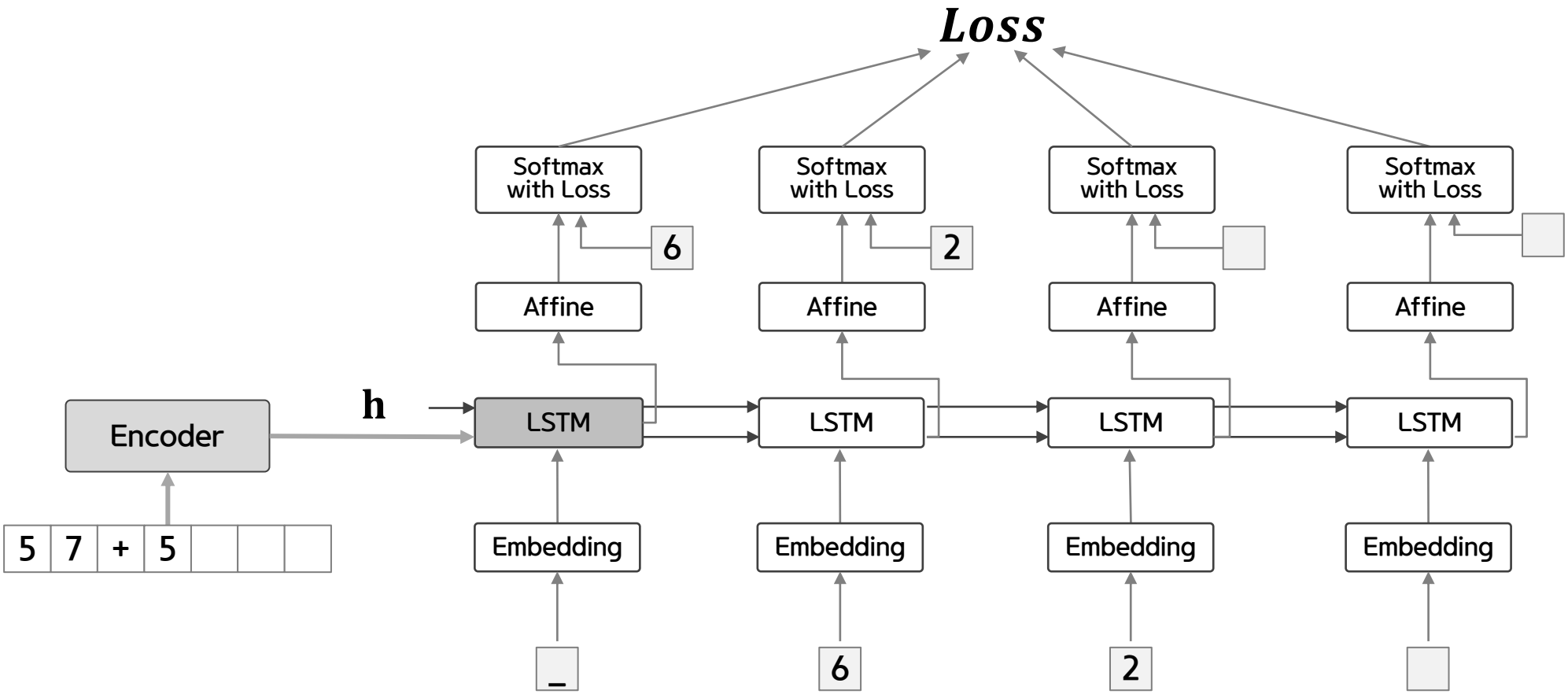
즉, h가 Decoder 에 있어서는 유일한 정보인 셈이다.

그러나 현재의 seq2eq 는 다음 그림과 같이 최초 시각의 LSTM 계층만이 벡터 h를 이용하고 있다.

이 중요한 정보인 h를 더 활용할 수는 없을까?

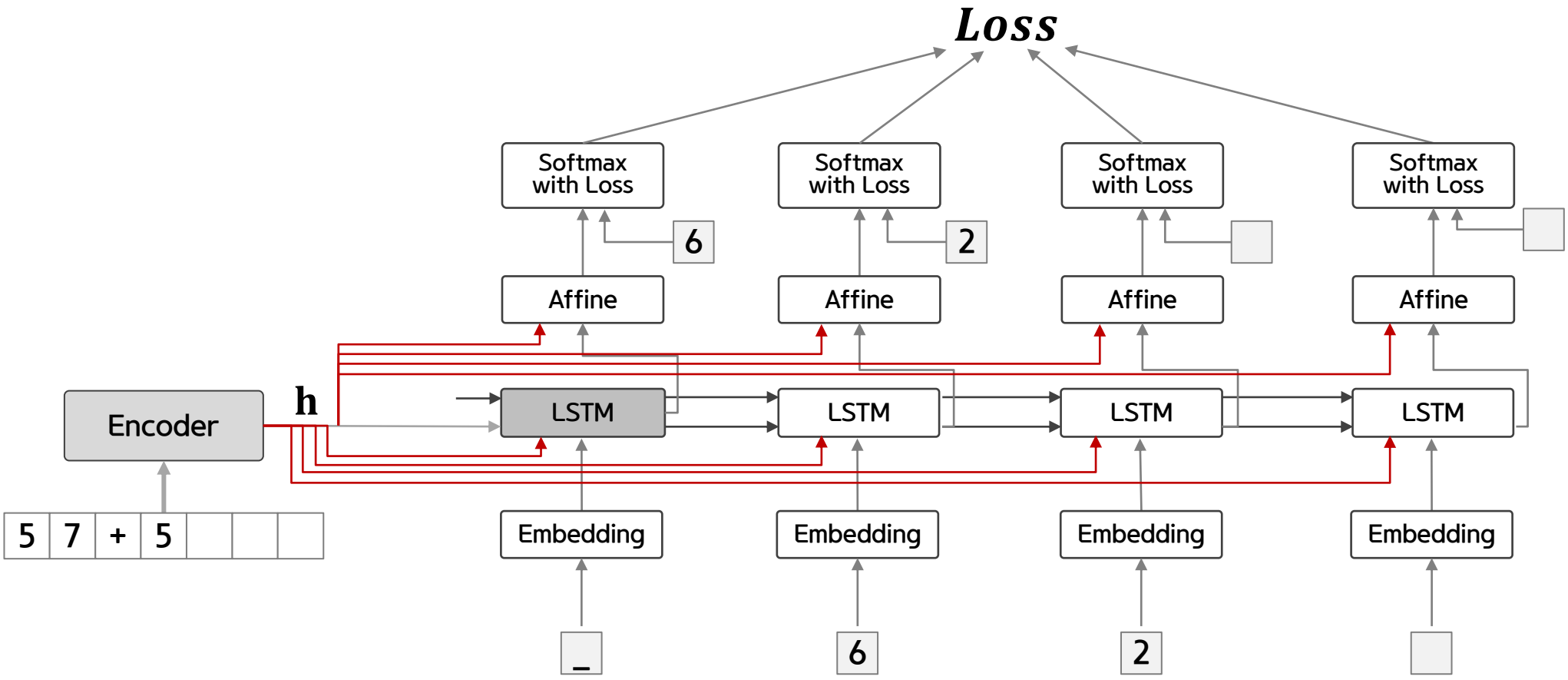
엿보기 (Peeky)

개선 전: Encoder의 출력 h는 첫 번째 LSTM 계층만이 받는다.



엿보기 (Peeky)

개선 후: Encoder의 출력 h를 모든 시각의 LSTM 계층과 Affine 계층에 전해준다.



엿보기 (Peeky)

그림과 같이 모든 시각의 Affine 계층과 LSTM 계층에 Encoder 의 출력 h를 전해준다.
이전 그림과 비교해보면, 기존에는 하나의 LSTM 만이 소유하던 중요 정보 h를
여러 계층(예에서는 8계층)이 공유함을 알 수 있다.
이는 집단 지성에 비유할 수 있다.
즉, 중요한 정보를 한 사람이 독점하는 것이 아니라,
많은 사람과 공유한다면 더 올바른 결정을 내릴 가능성이 커질 것이다.

이 개선안은 인코딩된 정보를 Decoder 의 다른 계층에도 전해주는 기법이다.
달리 보면, 다른 계층도 인코딩된 정보를 엿본다고 해석할 수 있다.
엿보다를 영어로 peek 이라고 하기 때문에 이 개선을 더한 Decoder 를 Peeky Decoder 라고 한다.
마찬가지로 Peeky Decoder 를 이용하는 seq2seq를 Peeky seq2seq 라고 한다.

앞 그림에서는 LSTM 계층과 Affine 계층에 입력되는 벡터가 2개씩이 되었다.
이는 실제로는 두 벡터가 연결된 것을 의미한다.
따라서 두 벡터를 연결시키는 concat 노드를 이용해

다음 그림처럼 그려야 정확한 계산 그래프가 된다.

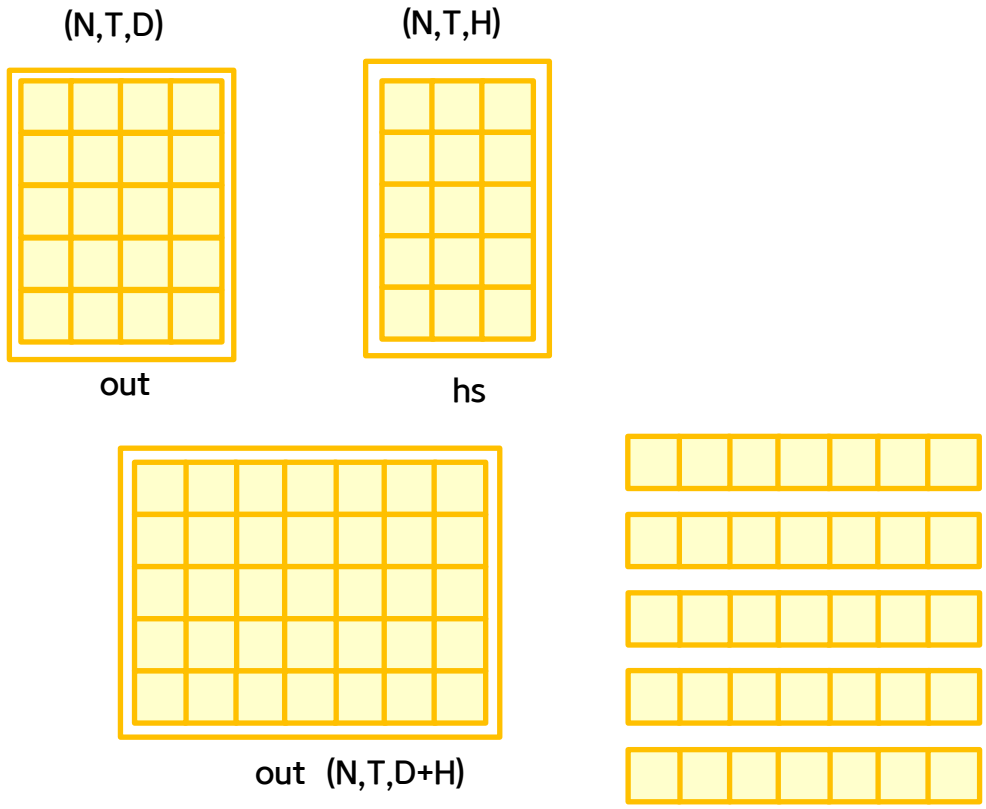
엿보기 (Peeky)

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

※ 코드 참조

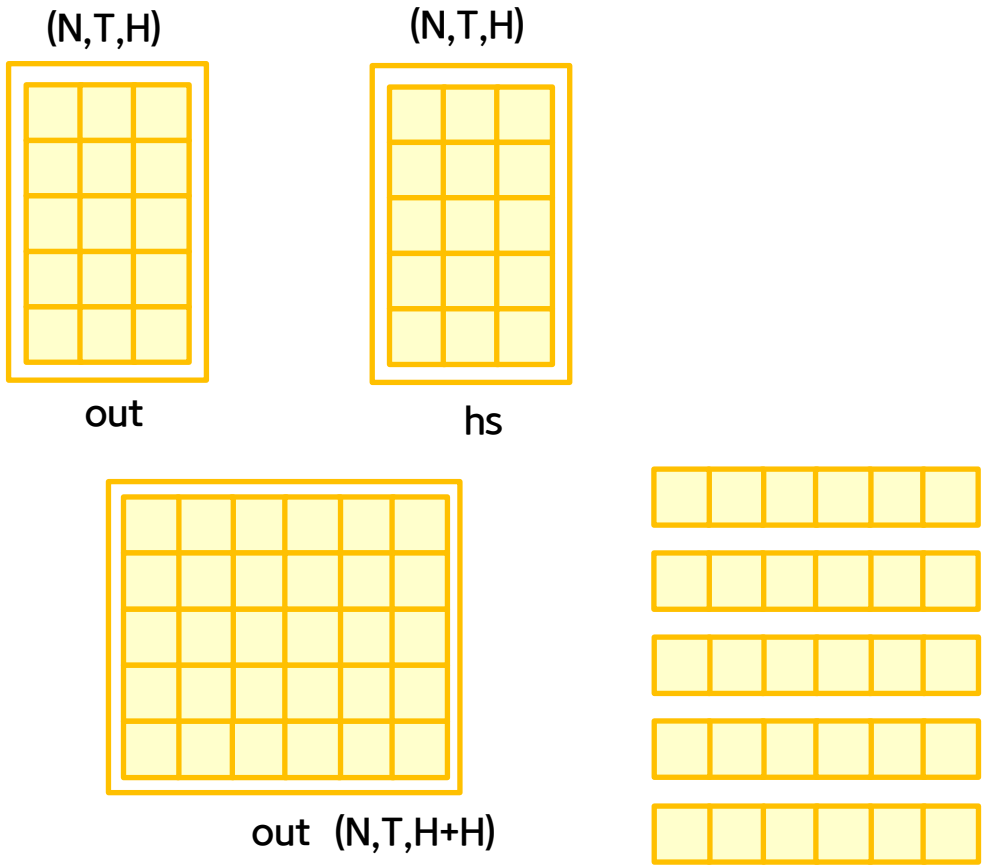
엿보기 (Peeky)

```
out = self.embed.forward(xs)
hs = np.repeat(h, T, axis=0).reshape(N, T, H)
out = np.concatenate((hs, out), axis=2)
```



엿보기 (Peeky)

```
out = self.lstm.forward(out)
out = np.concatenate((hs, out), axis=2)
```



엿보기 (Peeky)

Peeky 를 추가로 적용하자 seq2seq 의 결과가 월등히 좋아졌다.
이상의 실험 결과에서 Reverse 와 Peeky 가 함께 효과적으로 작동하고 있음을 알 수 있다.
입력 문장을 반전시키는 Reverse, 그리고 Encoder 의 정보를 널리 퍼지게 하는
Peeky 덕분에 만족할 만한 결과를 얻었다.

여기서 수행한 개선은 작은 개선이라 할 수 있다.
큰 개선은 다음 장에서 추가할 계획이다.
바로 어텐션이라는 기술로,
seq2seq 를 극적으로 진화시킬 수 있다.

이번 절의 실험은 주의해야 한다.
Peeky 를 이용하게 되면, 신경망은 가중치 매개변수가 커져서 계산량도 늘어난다.
따라서 이번 절의 실험 결과는 커진 매개변수만큼의 핸디캡을 감안해야 한다.

또한 seq2seq 의 정확도는 하이퍼 파라미터에 영향을 크게 받는다.
예제에서의 결과는 믿음직했지만, 실제 문제에서는 그 효과가 달라질 것이다.

7. RNN을 사용한 문장 생성

seq2seq 는 한 시계열 데이터를 다른 시계열 데이터로 변환한다.
이 시계열 데이터를 변환하는 프레임워크는 다양한 문제에 적용할 수 있다.
다음은 구체적인 예이다.

- 기계 번역: 한 언어의 문장을 다른 언어의 문장으로 변환
- 자동 요약: 긴 문장을 짧게 요약된 문장으로 변환
- 질의응답: 질문을 응답으로 변환
- 메일 자동 응답: 받은 메일의 문장을 답변 글로 변환

seq2seq 는 2개가 짝을 이루는 시계열 데이터를 다루는 문제에 이용할 수 있다.
자연어 외에도 음성이나 영상 등에도 이용할 수 있다.
얼핏 보기에는 seq2seq 가 적용될 수 없을 것 같은 문제라도 입력/출력 데이터를
전처리하면 seq2seq 를 적용할 수 있는 경우도 있다.

이번 절에서는 seq2seq 를 사용한 애플리케이션을 몇 가지 소개한다.

챗봇은 사람과 컴퓨터가 텍스트로 대화를 나누는 프로그램이다.
챗봇에도 seq2seq를 사용할 수 있다.
대화라는 것은 상대의 말과 자신의 말로 구성되기 때문에
상대의 말을 자신의 말로 변환하는 문제로 볼 수 있다.

즉, 대화의 텍스트 데이터가 준비되면 그것으로 seq2seq 를 학습시킬 수 있다.

seq2seq 기반의 챗봇의 대화 예

기계: 안녕하세요.
사람: 안녕하세요.
기계: 어떤 OS를 사용하시죠?
사람: 리눅스요.
.....
기계: 실행할 때 어떤 오류가 발생하죠?
사람: VPN 어쩌고 하는 오류가 나옵니다.
기계: VPN을 사용하여 네트워크에 연결할 때 오류 메시지가 뭐라고 뜨나요?
사람: 연결이 되지 않거나 혹은 그와 비슷한 상태...
.....
기계: 지금 집인가요, 아니면 회사인가요?
사람: 집이요.
기계: 그럼, 〈URL〉에 가서 선생님의 이름과 암호를 입력해보세요.
사람: 했어요.
기계: 〈NAME〉에서 로그인하여 잘 되는지 확인해보세요.
사람: 오, 잘 되네요!
기계: 좋습니다. 다른 문제는 없나요?
사람: 없어요, 매우 잘 됩니다.

파이썬으로 작성된 코드의 예

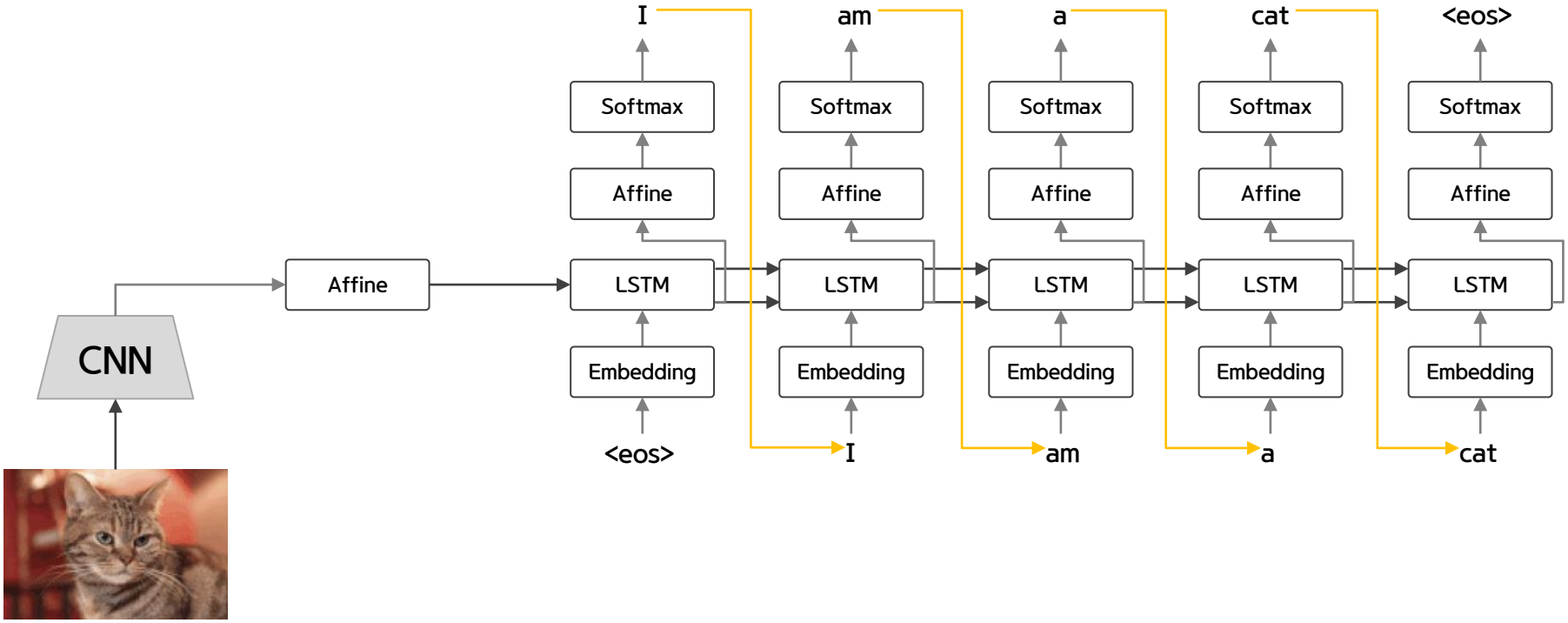
```
input:
    j=8584
    for x in range(8):
        j+=920
    b=(1500+j)
    print((b+7567))
target: 25011
```

```
input:
    i=8827
    c=(i-5347)
    print(((c+8704)) if 2641<8500 else 5308)
target: 12184
```

지금까지는 seq2seq 가 텍스트를 다루는 예만을 보았다.
하지만 seq2seq 는 텍스트 외에도, 이미지나 음성 등 다양한 데이터를 처리할 수 있다.

이번절에서는 이미지를 문장으로 변환하는 이미지 캡셔닝을 살펴본다.

이미지 캡셔닝은 이미지를 문장으로 변환한다.
이 문제도 다음 그림과 같이 seq2seq 의 틀에서 해결할 수 있다.



그림은 우리에게 친숙한 신경망 구성이다.
지금 까지와 다른 점은 Encoder 가 LSTM 에서 합성곱 신경망(CNN)으로 바뀐게 전부다.
겨우 LSTM을 CNN으로 대체한 것 만으로 seq2seq 는 이미지도 처리할 수 있다.
그림에서 CNN 에 대해 살짝 보충해보자.

이 예에서는 이미지의 인코딩을 CNN이 수행한다. 이때 CNN의 최종 출력은 특징 맵이다.

특징 맵은 3차원(높이,폭,패털)이므로, 이를 Decoder 의 LSTM이 처리할 수 있도록 손질해야 한다.
그래서 CNN의 특징 맵을 1차원으로 평탄화한 후 완전연결인 Affine 계층에서 변환한다.

그런 다음 변환된 데이터를 Decoder 에 전달하면, 문장 생성을 수행할 수 있다.

위 그림의 CNN에 CGG나 ResNet 등의 입증된 신경망을 사용하고,
가중치로는 다른 이미지 데이터넷으로 학습을 끝낸 것을 이용한다.

이렇게 하면 좋은 인코딩을 얻을 수 있고, 좋은 문장을 생성할 수 있다.

이제 seq2seq 가 이미지 캡셔닝을 수행한 예를 몇 가지 살펴보자.
여기에 사용된 신경망은 앞 그림을 기초로 한 것이다.

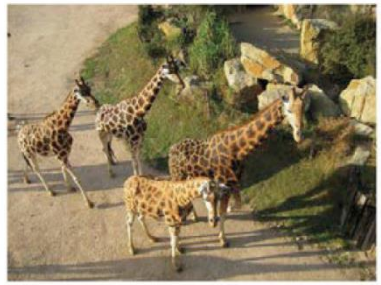
이미지 캡셔닝의 예: 이미지를 텍스트로 변환

A person on a beach flying a kite(해변에서 연 날리는 사람)

A black and white photo of a train on a train track
(선로 위의 열차를 찍은 흑백 사진)

A person skiing down a snow covered slope
(눈 덮인 슬로프에서 스키 타는 사람)

A group of giraffe standing next to each other
(횡대로 줄지어 서 있는 기린 떼)



주제: RNN을 이용한 문장 생성

6장에서 다룬 RNN을 사용한 언어 모델을 손질하여,
문장 생성 기능을 추가했다.

또한 seq2seq 에게 간단한 덧셈 문제를 학습시키는 데 성공했다.

seq2seq 는 Encoder 와 Decoder 를 연결한 모델로,
결국 2개의 RNN을 조합한 단순한 구조이다.
하지만 단순함에도 불구하고, seq2seq 는 매우 큰 가능성을 지니고 있어서 다양한 애플리케이션에 적용할 수 있다.

이번 장에서는 seq2seq 를 개선하는 아이디어 2개를 살펴봤다.

1. Reverse
2. Peeky

다음장에서는 seq2seq 를 한 층 더 개선할 것이다. (어텐션)
어텐션은 딥러닝에서 가장 중요한 기법 중 하나이다.

다음장에서는 어텐션 매커니즘을 설명하고 구현하여,
한층 더 강력한 seq2seq 를 구현한다.