

# 밑바닥부터 시작하는 딥러닝 2: 7장 RNN을 사용한 문장 생성

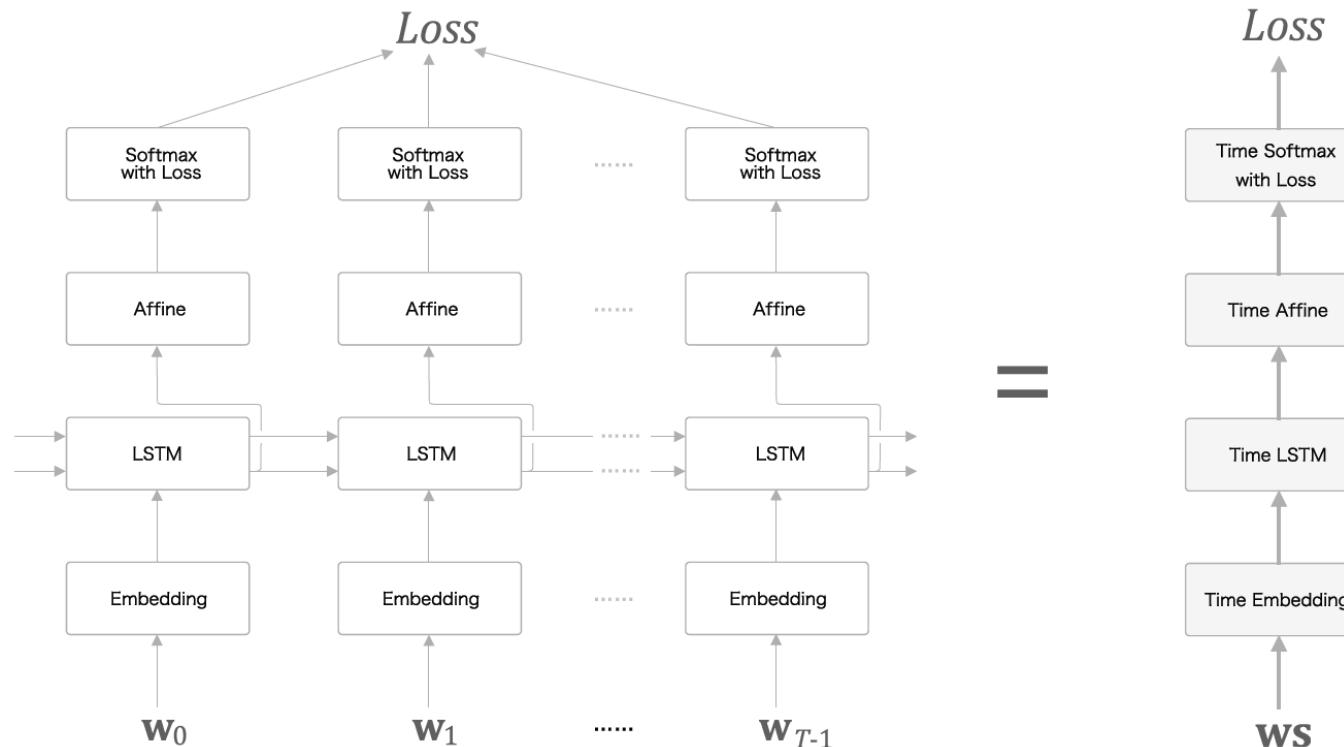
2020/03/19

# 7. RNN을 사용한 문장 생성

- 언어 모델을 사용해 '문장 생성'을 수행하는 애플리케이션을 구현
  - 말뭉치를 사용해 학습한 언어 모델을 이용하여 새로운 문장을 생성
  - 개선된 언어 모델을 이용해 더 자연스러운 문장을 생성
- seq2seq(sequence to sequence): 한 시계열 데이터를 다른 시계열 데이터로 변환
  - RNN 두 개를 연결하는 방법으로 구현

# 7.1 언어 모델을 사용한 문장 생성: 1. RNN을 사용한 문장 생성의 순서

- 앞 장에서 LSTM을 이용해 구현한 언어 모델을 가지고 문장을 생성시켜보자.



# 7.1.1 RNN을 사용한 문장 생성의 순서

- eg) "you say goodbye and I say hello"라는 말뭉치로 학습한 언어 모델

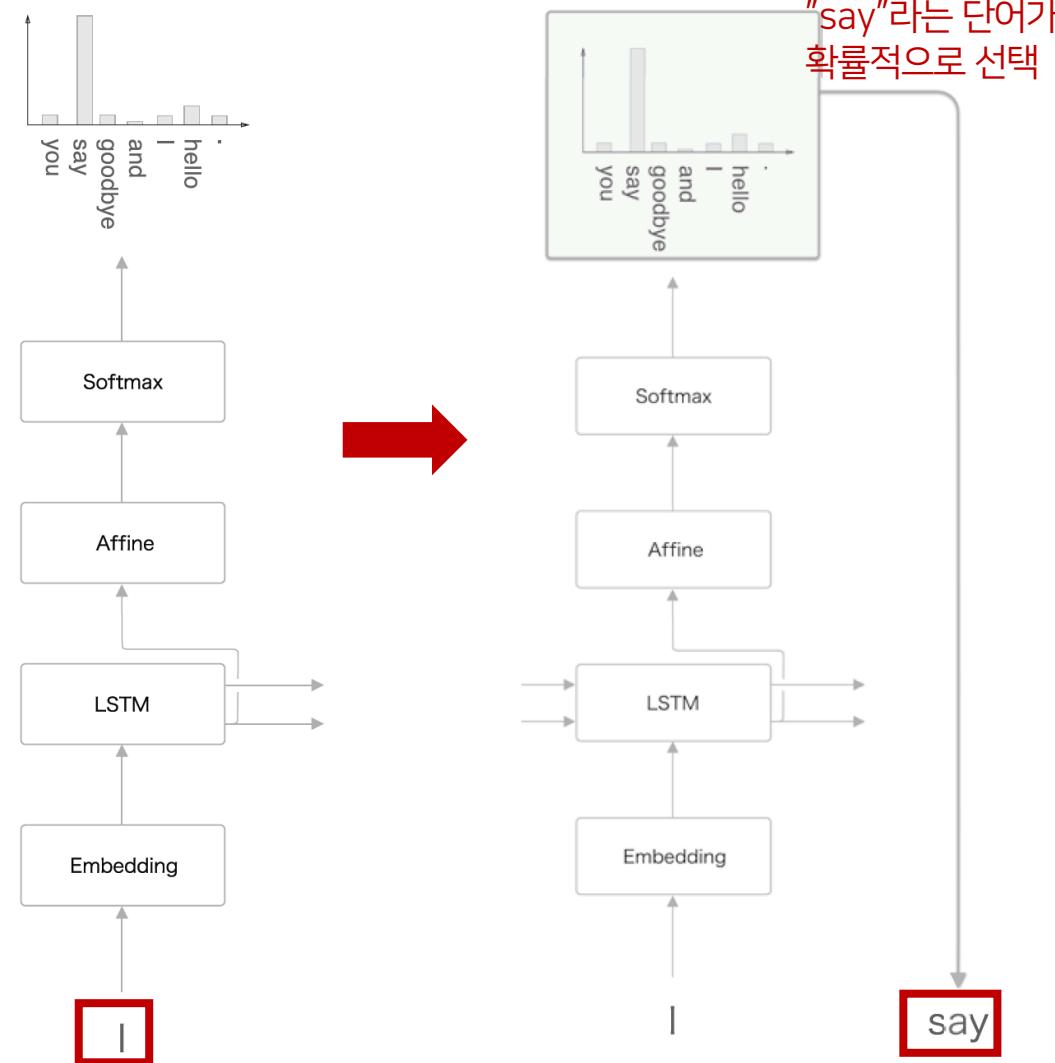
- 언어 모델은 지금까지 주어진 단어들로부터 다음에 출현하는 단어의 확률 분포를 출력
- 이 결과를 기초로 다음 단어를 생성하게 된다.

## 1) 확률이 가장 높은 단어를 선택

- 확률이 가장 높은 단어를 선택할 뿐이므로 결과가 일정하게 정해지는 '결정적'인 방법

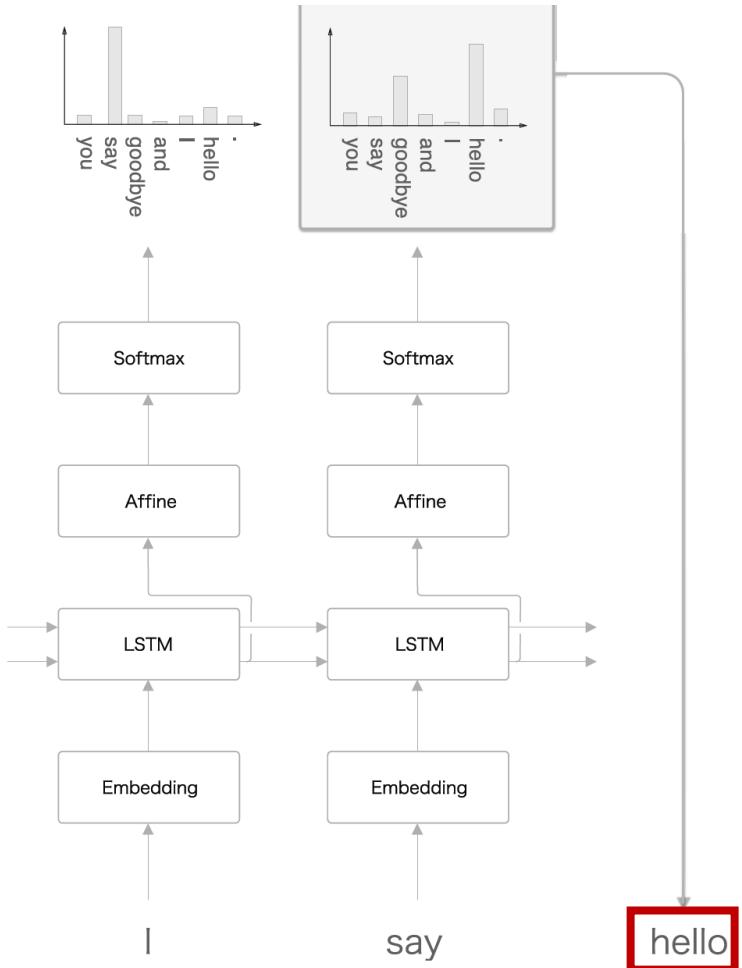
## 2) 각 후보 단어의 확률에 맞게 선택

- '확률적'으로 단어를 선택하는 방법으로 확률이 높은 단어는 선택되기 쉽고 낮은 단어는 선택되기 어려움
- 매번 선택되는 단어가 다를 수 있음.



# 7.1.1 RNN을 사용한 문장 생성의 순서

- 계속해서 생성된 단어를 언어 모델에 입력해 다음 단어의 확률분포를 얻고, 이를 기초로 다음에 출현할 단어를 샘플링
- 이 작업을 반복해 원하는 만큼 단어를 생성해낸다!
- 이렇게 생성된 문장은 training 데이터에는 존재하지 않는, **새로 생성된 문장**
  - 언어 모델은 훈련 데이터를 암기한 것이 아니라 훈련 데이터에서 사용된 **단어의 정렬 패턴을 학습한 것**
  - 언어 모델이 말뭉치로부터 단어의 출현 패턴을 올바르게 학습하도록 만 들어 주는 것이 중요하다.



## 7.1.2 문장 생성 구현

- 앞에서 만든 Rnnlm 클래스를 상속해 RnnlmGen 클래스를 만들어 문장을 생성하는 메서드를 추가하자.

```
class RnnlmGen(Rnnlm):  
    def generate(self, start_id, skip_ids=None, sample_size=100):  
        word_ids = [start_id]  
  
        x = start_id  
        while len(word_ids) < sample_size:  
            x = np.array(x).reshape(1, 1)  
            score = self.predict(x) # 각 단어의 점수를 출력  
            p = softmax(score.flatten())# Softmax 함수를 이용해 정규화 해 확률분포 p를 얻음  
  
            sampled = np.random.choice(len(p), size=1, p=p) # 확률분포 p로부터 다음 단어를 샘플링  
            if (skip_ids is None) or (sampled not in skip_ids):  
                x = sampled  
                word_ids.append(int(x))  
  
        return word_ids  
  
    def get_state(self):  
        return self.lstm_layer.h, self.lstm_layer.c  
  
    def set_state(self, state):  
        self.lstm_layer.set_state(*state)
```

- start\_id: 최초로 주는 단어의 ID
- sample\_size: 샘플링 하는 단어의 수
- skip\_ids: 단어 ID의 리스트로 이 리스트에 속하는 단어 ID는 샘플링되지 않도록 해준다
  - PTB 데이터셋의 <unk>나 N과 같이 전처리 된 단어를 샘플링 하지 않도록 해줌.

## 7.1.2 문장 생성 구현

```
corpus, word_to_id, id_to_word = ptb.load_data('train')
vocab_size = len(word_to_id)
corpus_size = len(corpus)

model = RnnlmGen()
#model.load_params('Rnnlm.pkl')

# start 문자와 skip 문자 설정
start_word = 'you'          # 첫 단어로 'you'
start_id = word_to_id[start_word]
skip_words = ['N', '<unk>', '$']
skip_ids = [word_to_id[w] for w in skip_words]
# 문장 생성
word_ids = model.generate(start_id, skip_ids)
txt = ' '.join([id_to_word[i] for i in word_ids])
txt = txt.replace('<eos>', '\n')
print(txt)
```

- 더 나은 언어 모델을 사용해 결과를 개선해보자!

you detectors datapoint ivan cypress heels favored lynch computer termina  
te ranged patterns subsequent reason serial tone leading named mid-1980s  
irresponsible trucks turns changing urge first-quarter results floating p  
resumably succeeds speculation steer limits corporation legislation virtu  
e extract vested pleaded station approaches diminished u.s.a. lawmakers d  
ue john shack substantial potent current-carrying habit olympia cable-tv  
suing retreat culture 'd identify insure passage sixth recovered hepatiti  
s cheap taxes namely royal notion reinvestment spacecraft education fried  
ralph aborted devaluation outsiders texans cities\abc obtaining sam camb  
odia amendments prizes house malcolm robots forfeiture that municipals de  
fendants non-violent legislator probing tries satisfy necessarily held ma  
nagua period participating rey

모델의 가중치 초기값으로 무작위한 값을 사용했을 때

you say or supporting any sad other ratio next year for a certain number  
of business.

the organizations show removed hedges to indicate the tax benefits for a  
willingness to declaring and instrument and two areas and spreads it for  
home loyalty.

congressional republicans.

the rothschilds do well.

seems equally guilty.

the press also agreed to increase fannie mae capital basket loans instea  
d of a drought.

one big trend act ended unchanged at least compared with minnesota and c  
raven a dollar.

since maturity was also representing yesterday by the end of monday 's

학습한 가중치 매개변수로 문장을 생성했을 때

### 7.1.3 더 좋은 문장으로

- 앞 장에서 RNNLM을 개선해 더 좋은 모델을 구현
  - 이 가중치를 이용해 문장을 생성해보면 다음과 같음.
  - 좀 더 자연스러운 문장이 생성되었으며, 문법적으로도 더 잘 맞는 문장을 생성
    - 모델을 좀 더 개선하고 더 큰 말뭉치를 사용하면 더 자연스러운 문장을 생성할 수 있다!
  - “the meaning of life is”라는 문장을 주고 이어지는 말을 생성하도록 만들 수도 있음.
    - 모델에 [‘the’, ‘meaning’, ‘of’, ‘life’]라는 단어를 차례로 주어 순전파를 수행하면 LSTM 계층에는 그 정보가 유지
    - 그런 다음 ‘is’를 첫 단어로 입력하면 “the meaning of life is”에 이어지는 문장 생성이 가능하다.

```
start_words = 'the meaning of life is'
start_ids = [word_to_id[w] for w in start_words.split(' ')]

for x in start_ids[:-1]:
    x = np.array(x).reshape(1, 1)
    model.predict(x)
```

you waited for a recession of disappointments under capital.  
in an interview with the government officials said they say they 're willing to risk about the case of which the press 's initial credibility will cost to the bush administration.  
but some mr. gorbachev has no viable problem if a implications grow from the administration would be packaged the corporate bankers.  
initially even he said that the republicans ' contribution is making up the lawsuit.  
under the rules only a few more mr. kemp has added the federal government and ultimately would provide options on his

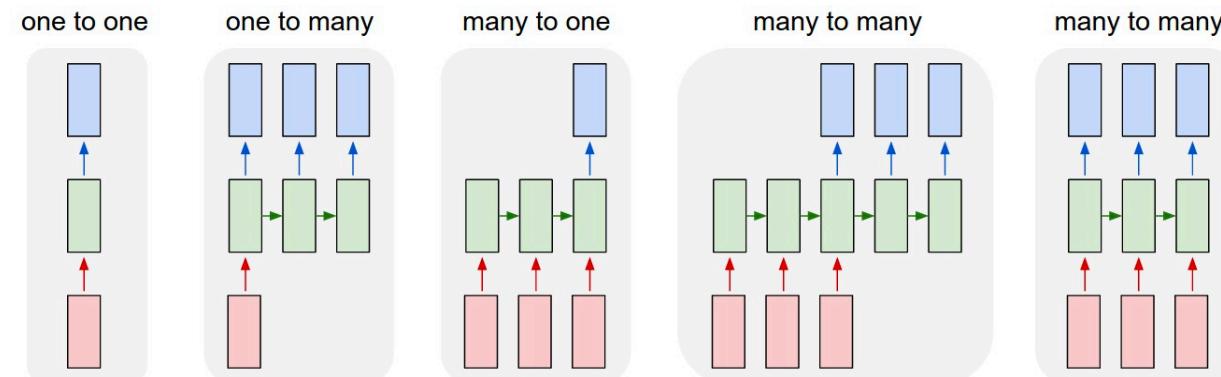
the meaning of life is perhaps a steady commitment to the deal for a financial association.

they have a monopoly with people and its brokers.  
most retailers are nigel gorbachev 's wild battle against some others and top he has successfully signaled they would double proven economic problems in japan.

the japanese have all grown together here in attracting carefully global demands and improved import commitments for more than a decade.  
they believe we really think campbell would blame it with the long-term natural gas market says malcolm 's sport ask drug services.  
meanwhile mr. kageyama has a computer-guided

## 7.2 seq2seq

- 세상에는 다양한 종류의 시계열 데이터가 존재
  - 언어 데이터, 음성 데이터, 동영상 데이터 ...
- 이러한 시계열 데이터가 입력과 출력인 문제도 다양하게 존재
  - 지금부터는 시계열 데이터를 다른 시계열 데이터로 변환하는 모델을 살펴보자.
  - 2개의 RNN을 이용하는 seq2seq(sequence to sequence) 기법



## 7.2.1 seq2seq의 원리

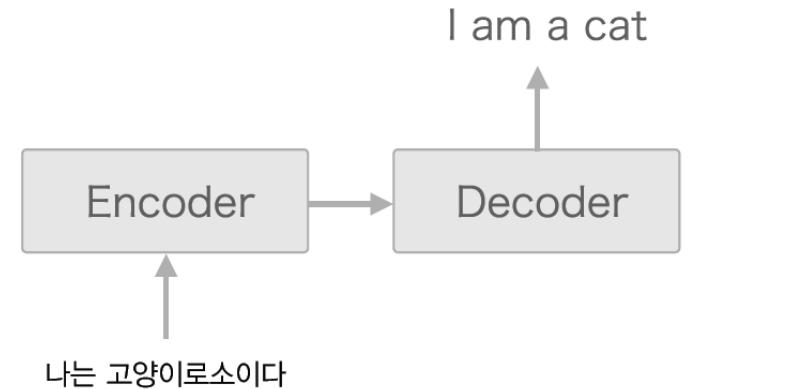
- seq2seq => **Encoder-Decoder 모델**

- **Encoder**: 입력 데이터를 인코딩(부호화)
  - **Decoder**: 인코딩된 데이터를 디코딩(복호화)

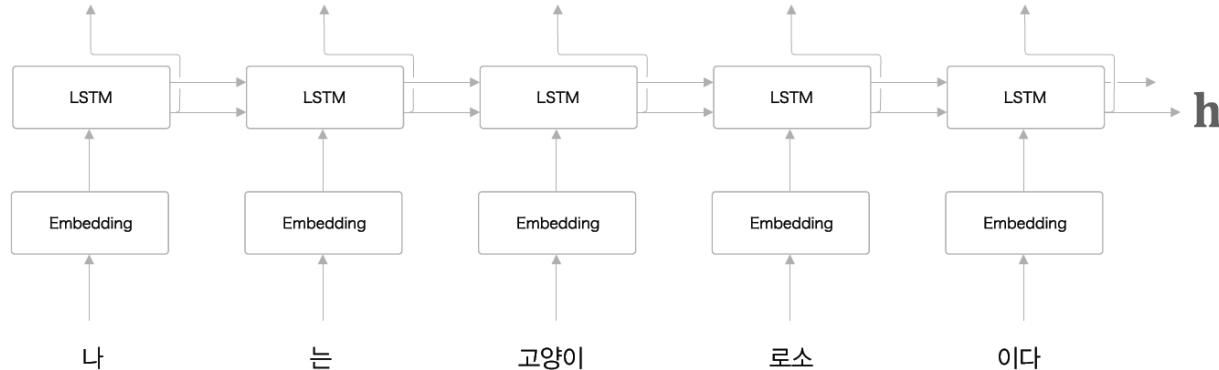
- eg) “나는 고양이로소이다” => “I am a cat”

- Encoder가 “나는 고양이로소이다”라는 출발어 문장을 인코딩
    - 인코딩한 정보를 Decoder에 전달하고, Decoder가 도착어 문장을 생성
    - 이때 Encoder가 인코딩한 정보에는 번역에 필요한 정보가 조밀하게 응축되어 있으며, Decoder는 조밀하게 응축된 이 정보를 바탕으로 도착어 문장을 생성

- 이렇게 인코더와 디코더가 협력해 시계열 데이터를 다른 시계열 데이터로 변환
  - 이때 인코더와 디코더로 **RNN**을 사용할 수 있다.

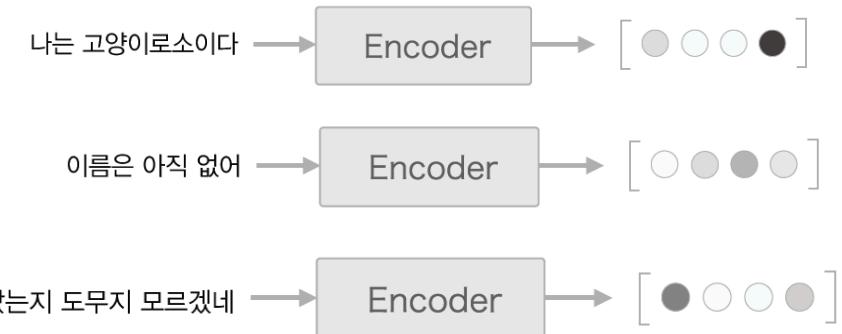


## 7.2.1 seq2seq의 원리



- **Encoder:**

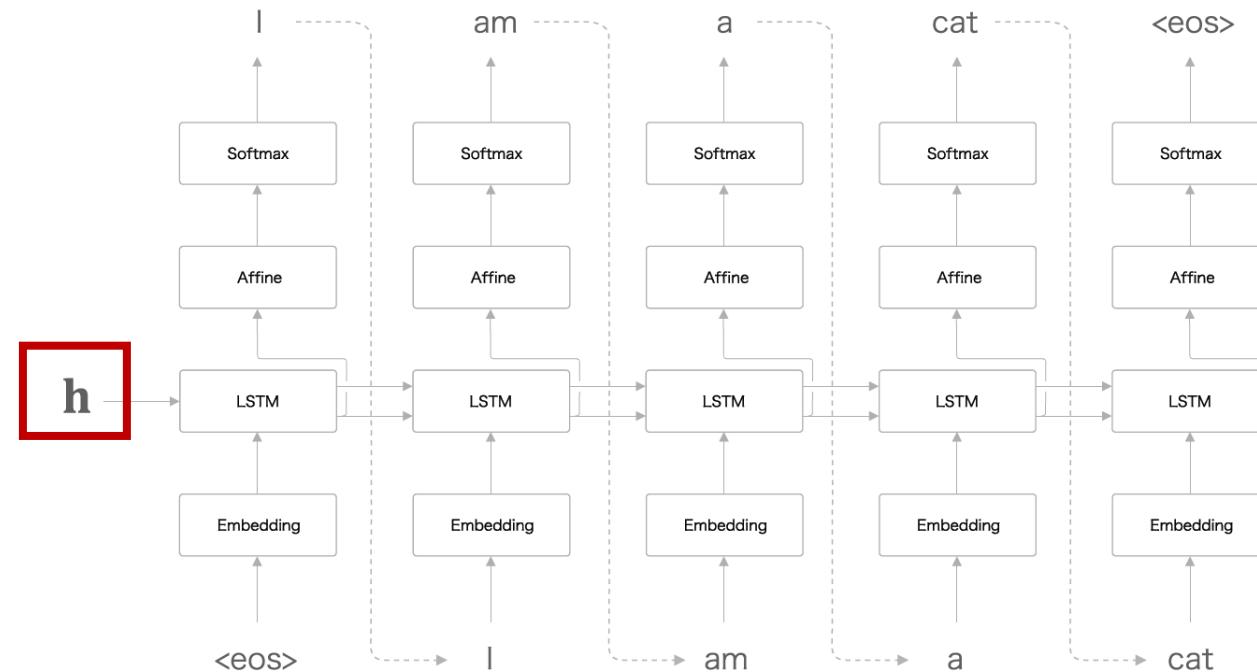
- 시계열 데이터를  $h$ 라는 은닉 상태 벡터로 변환
  - 그림은 LSTM을 이용했으며, 우리말 문장을 단어 단위로 쪼개 입력한다고 가정
  - $h$ : LSTM 계층의 마지막 은닉 상태, 입력 문장(출발어)을 번역하는 데 필요한 정보가 인코딩 됨.
  - 이 때 LSTM의 은닉 상태  $h$ 는 고정 길이 벡터
    - 즉, '인코딩한다' = '임의 길이의 문장을 고정 길이 벡터로 변환한다'



## 7.2.1 seq2seq의 원리

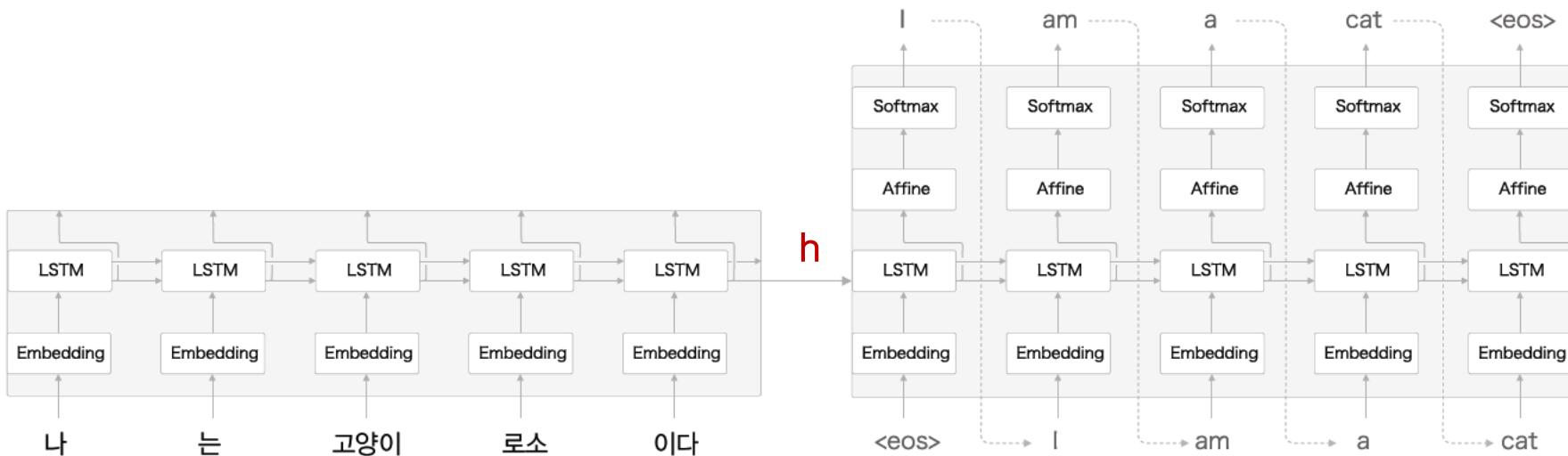
- Decoder

- Encoder와 완전히 같은 구성이지만, LSTM 계층이  $h$ 를 입력 받는다는 점에서 차이를 보임.
  - 인코더는 LSTM 계층이 아무것도 받지 않음(은닉 상태로 영벡터를 받음)



## 7.2.1 seq2seq의 원리

- seq2seq는 LSTM 두 개(Encoder의 LSTM과 Decoder의 LSTM)으로 구성
  - LSTM 계층의 은닉 상태  $h$ 가 **인코더와 디코더를 이어주는 역할을 수행**
  - **순전파**때는 인코더에서 인코딩된 정보가 은닉 상태  $h$ 를 통해 디코더에 전해짐.
  - **역전파**때는 기울기가 은닉 상태  $h$ 를 통해 디코더로부터 인코더로 전해짐.



## 7.2.2 시계열 데이터 변환용 장난감 문제

- 장난감 문제(toy problem): 머신러닝을 평가하고자 만든 간단한 문제

- eg) 더하기

- “57+5” => “62” 와 같이 학습

- 문자 단위로 분할해 처리

- “57+5” => [‘5’, ‘7’, ‘+’, ‘5’]



## 7.2.3 가변 길이 시계열 데이터

- 즉 '덧셈'을 문자 리스트로써 다룸
  - 이때 덧셈 문장("57+5"나 "628+521")이나 대답("62", "1149")의 문자 수가 문제마다 다르다.
  - 즉 샘플마다 시간 방향의 크기가 다른 '가변 길이 시계열 데이터'를 다루게 됨.
    - 신경망 학습 시 '미니배치 처리'를 하려면 추가로 처리를 해 주어야 함!
      - 미니배치로 학습 시 다수의 샘플을 한꺼번에 처리하므로 한 미니배치에 속한 샘플들의 데이터 형상이 모두 똑같아야 함.
      - 가변 길이 시계열 데이터를 미니배치로 학습하기 위한 가장 단순한 방법은 패딩(padding)을 사용하는 것.
- 패딩(padding): 원래의 데이터에 의미 없는 데이터를 채워 모든 데이터의 길이를 균일하게 맞추는 기법
  - 왼쪽 그림과 같이 입력 데이터의 길이를 통일하고, 남는 공간에는 의미 없는 데이터(공백 등)을 채움
  - 이번 예시에서는 0~999 사이의 숫자 2개만 더하기로 한다.
    - '+'까지 포함하면 최대 문자 수는 7
    - 덧셈 결과는 최대 4 문자( $999 + 999 = 1998$ )
    - 질문과 정답을 구분하기 위해 출력 앞에 구분자(\_)를 붙임
      - 이 구분자는 디코더에 문자열을 생성하라고 알리는 신호로 사용
    - 출력 데이터는 총 5문자
  - 이렇게 패딩을 적용해 데이터 크기를 통일시키면 가변 길이 시계열 데이터도 처리할 수 있음!
  - 하지만 원래 존재하지 않던 패딩용 문자까지 seq2seq가 처리

입력	출력
5 7 + 5	_ 6 2
6 2 8 + 5 2 1	_ 1 1 4 9
2 2 0 + 8	_ 2 2 8

## 7.2.3 가변 길이 시계열 데이터

- 따라서 패딩을 적용해야 하지만, 정확성이 중요하다면 seq2seq에 패딩 전용 처리를 추가해야 한다.
  - 디코더에 입력된 데이터가 패딩 => 손실의 결과에 반영하지 않도록 처리
    - Softmax with Loss 계층에 '마스크 기능'을 추가해 해결
  - 인코더에 입력된 데이터가 패딩 => LSTM 계층이 이전 시각의 입력을 그대로 출력
    - 즉, LSTM 계층은 마치 패딩이 존재하지 않았던 것처럼 인코딩 가능
  - 이번 장에서는 이해 난이도를 낮추기 위해 패딩용 문자도 특별히 구분하지 않고 일반 데이터처럼 다루기로 하자.

## 7.2.4 덧셈 데이터셋

- 다음과 같은 데이터를 학습 데이터로 사용
- 이와 같은 seq2seq용 학습 데이터를 쉽게 처리할 수 있도록 책에서 전용 모듈을 제공
  - load\_data(file\_name, seed):
    - file\_name으로 지정한 텍스트 파일을 읽어 텍스트를 문자 ID로 변환
    - 이를 훈련 데이터와 테스트 데이터로 나눠 반환
    - seed는 랜덤 값의 초기값으로 훈련 데이터와 테스트 데이터로 나누기 전 전체 데이터를 뒤섞을 때 사용한다.
  - get\_vocab():
    - 문자와 문자 ID의 대응 관계를 담은 딕셔너리들(char\_to\_id, id\_to\_char)을 반환

```
# 7.2.4 덧셈 데이터셋
(x_train, t_train), (x_test, t_test) = \
    sequence.load_data('addition.txt', seed=1984)
char_to_id, id_to_char = sequence.get_vocab()

print(x_train.shape, t_train.shape)
print(x_test.shape, t_test.shape)

(45000, 7) (45000, 5) train 데이터: 45,000개
(5000, 7) (5000, 5) test 데이터: 5,000개

print(char_to_id)

{'1': 0, '6': 1, '+': 2, '7': 3, '5': 4, ' ': 5, '_': 6, '9': 7, '2': 8, '0': 9, '3': 10, '8': 11, '4': 12}
```

```
print(x_train[0])
print(t_train[0])

[ 3  0  2  0  0 11  5] ["7", "1", "+", "1", "1", "8", " "]
[ 6  0 11  7  5] ["_", "1", "8", "9", " "]

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
```

84+317	_401	7+208	_215
9+3	_12	723+8	_731
6+2	_8	144+14	_158
18+8	_26	5+7	_12
85+52	_137	5+46	_51
9+1	_10	9+15	_24
8+20	_28	621+0	_621
5+3	_8	81+89	_170
1+3	_4	846+84	_930
57+10	_67	27+63	_90
		4+582	_586

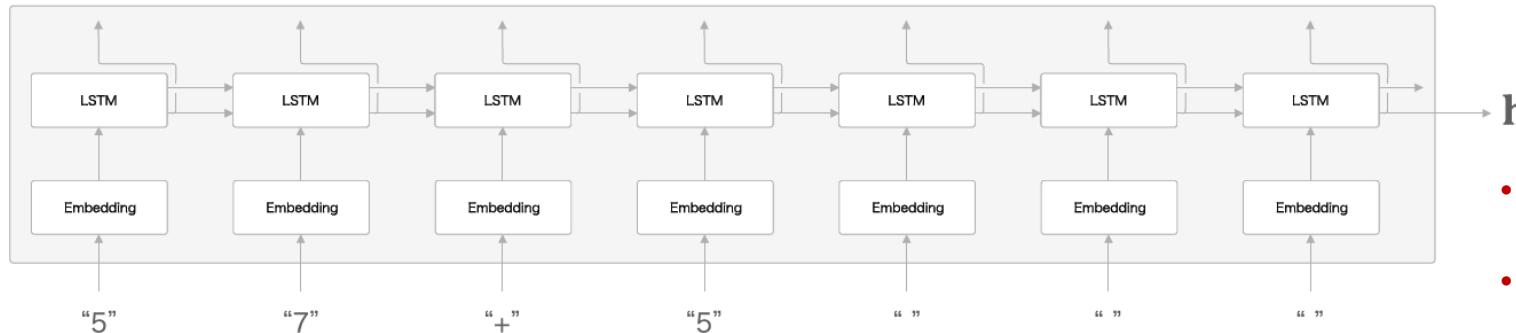
...

정석대로라면 데이터셋을 3개(훈련용, 검증용, 테스트용)으로 나누어 사용해야 한다.

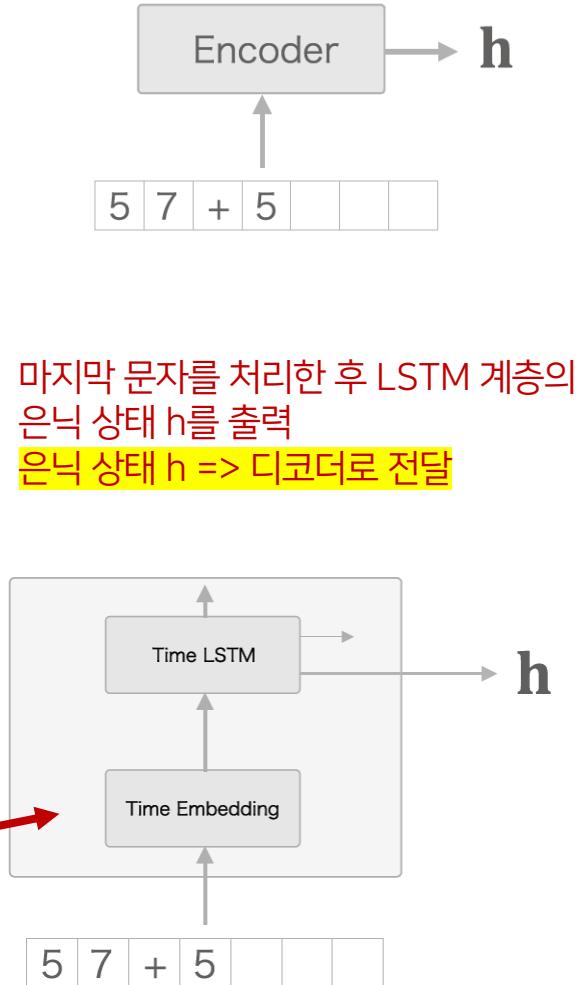
훈련용 데이터로는 학습을, 검증용 데이터로는 하이퍼파라미터를 튜닝하며, 테스트 데이터로는 모델의 성능을 평가한다

# 7.3 seq2seq 구현: 1. Encoder 클래스

- Encoder 클래스는 문자열을 받아 벡터  $h$ 로 변환한다.
  - LSTM 계층을 이용해 Encoder를 구성한다.



- Embedding 계층과 LSTM 계층으로 구성된다.
- Embedding 계층**에서는 문자 ID를 문자 벡터로 변환하며, 이 문자 벡터는 LSTM 계층으로 입력됨.
- LSTM 계층**은 오른쪽으로는 은닉 상태와 셀을 출력 / 위쪽으로는 은닉 상태만 출력
  - 이 구성에서는 더 위에 다른 계층이 없으므로 위쪽 출력은 폐기됨.
- 시간 방향을 한꺼번에 처리하도록 Time 계층을 이용하자.



## 7.3.1 Encoder 클래스

- Encoder 클래스:

```
# 7.3.1 Encoder 클래스
class Encoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx = (rn(D, 4 * H) / np.sqrt(D)).astype('f')
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f')

        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=False) # Time LSTM 계층이 상태를 유지하지 않기 때문에
        self.params = self.embed.params + self.lstm.params
        self.grads = self.embed.grads + self.lstm.grads
        self.hs = None
```

- vocab\_size: 어휘 수(문자의 종류)
  - 우리 예시에서는 0~9의 숫자, '+', '.', '\_' 총 13가지 문자를 사용
- wordvec\_size: 문자 벡터의 차원 수
- hidden\_size: LSTM 계층의 은닉 상태 벡터의 차원 수

# Time LSTM 계층이 상태를 유지하지 않기 때문에  
stateful=False로 설정

- 5장과 6장에서는 '긴 시계열 데이터'가 하나뿐인 문제를 다  
루었다면, 7장에서는 '짧은 시계열 데이터'가 여러 개인 문제  
이기 때문
- 따라서 문제마다 LSTM의 은닉 상태가 초기화된다.

## 7.3.1 Encoder 클래스

- Encoder 클래스:

```
def forward(self, xs):  
    xs = self.embed.forward(xs)  
    hs = self.lstm.forward(xs)  
    self.hs = hs  
    return hs[:, -1, :]
```

- 순전파:

- Time Embedding 계층과 Time LSTM 계층의 forward() 메서드를 호출
- Time LSTM 계층의 마지막 시각의 은닉 상태만을 추출
- 그 값을 인코더의 forward() 메서드의 출력으로 반환

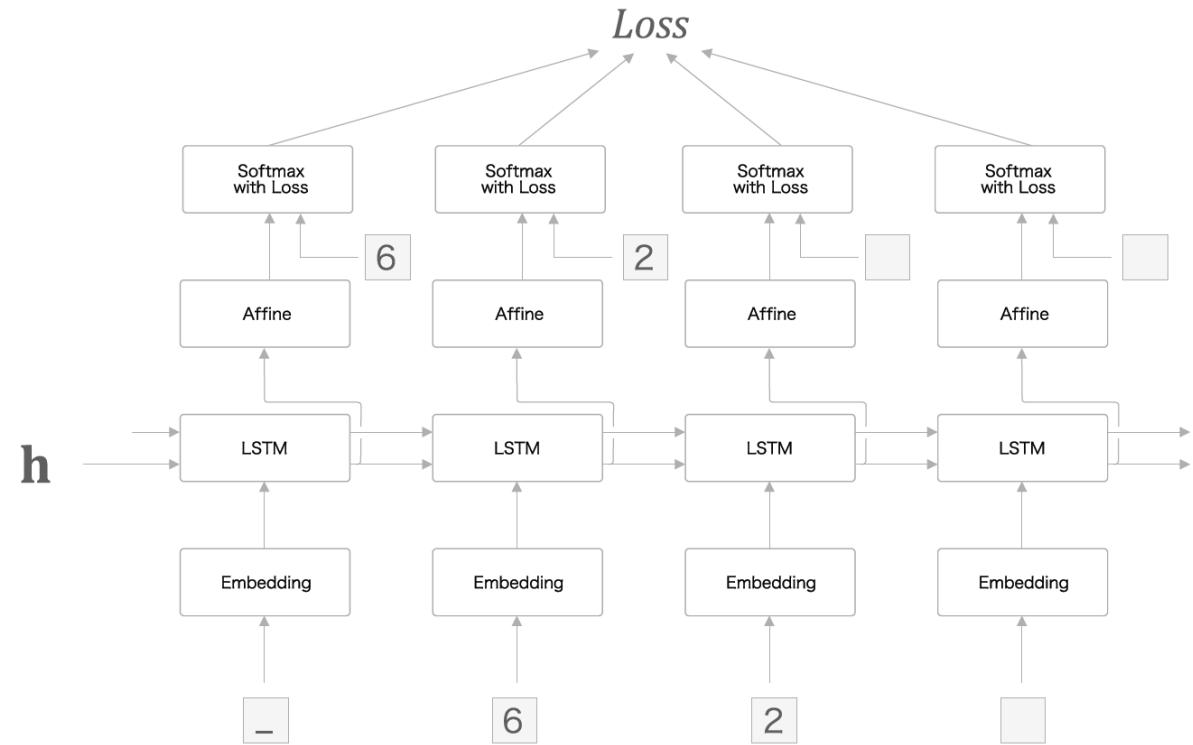
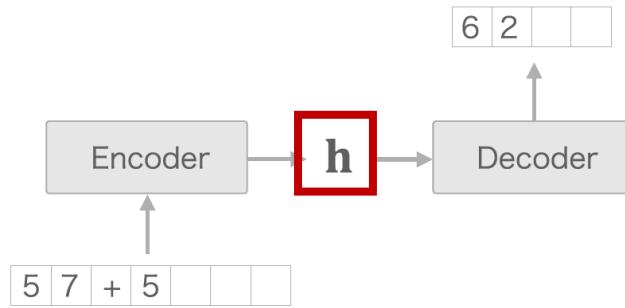
```
def backward(self, dh):  
    dhs = np.zeros_like(self.hs)  
    dhs[:, -1, :] = dh  
  
    dout = self.lstm.backward(dhs)  
    dout = self.embed.backward(dout)  
    return dout
```

- 역전파:

- LSTM 계층의 마지막 은닉 상태에 대한 기울기가 dh 인수로 전해짐
  - dh: 디코더가 전해주는 기울기
- 원소가 모두 0인 텐서 dhs를 생성, dh를 dhs의 해당 위치에 할당
- Time LSTM 계층과 Time Embedding 계층의 backward() 메서드를 호출

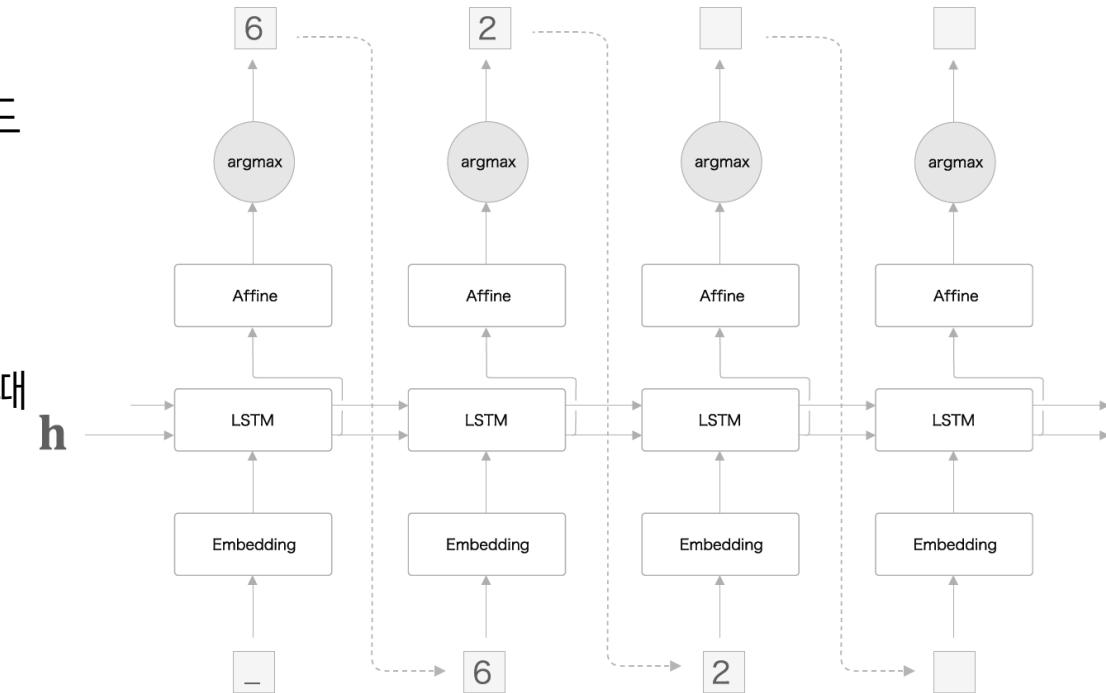
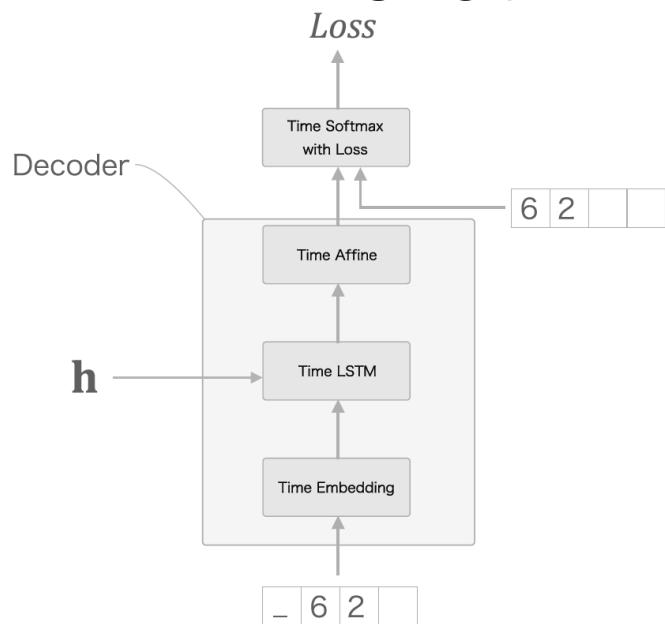
## 7.3.2 Decoder 클래스

- Decoder 클래스는 Encoder 클래스가 출력한  $h$ 를 받아 목적으로 하는 다른 문자열을 출력
  - LSTM 계층을 이용해 Decoder를 구성한다.
  - 점수가 가장 높은 문자를 하나 골라 출력한다.



## 7.3.2 Decoder 클래스

- Decoder가 문자열을 생성시키는 흐름은 다음과 같다:
  - 'argmax' 노드: 최댓값을 가진 원소의 인덱스를 선택하는 노드
  - Softmax 계층을 사용하지 않고 Affine 계층이 출력하는 점수가 가장 큰 문자 ID를 선택
    - Softmax 계층은 입력된 벡터를 정규화
    - 이때 벡터의 각 원소 값은 달라지지만 대소 관계는 바뀌지 않기 때 문에 Softmax 계층을 생략할 수 있다.



- 학습 시와 문장 생성 시에 Softmax 계층을 다르게 취급
  - 학습 시에만 Softmax 계층을 사용하고, 문장 생성 시에는 사용하지 않는다.
  - Softmax with Loss 계층은 이후 구현하는 seq2seq 클래스에서 처리하도록 한다.
- Time Embedding, Time LSTM, Time Affine 계층으로 구성

## 7.3.2 Decoder 클래스

- Decoder 클래스:

```
# 7.3.2 Decoder 클래스
class Decoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx = (rn(D, 4 * H) / np.sqrt(D)).astype('f')
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f')
        affine_W = (rn(H, V) / np.sqrt(H)).astype('f')
        affine_b = np.zeros(V).astype('f')

        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=True)
        self.affine = TimeAffine(affine_W, affine_b)

        self.params, self.grads = [], []
        for layer in (self.embed, self.lstm, self.affine):
            self.params += layer.params
            self.grads += layer.grads
```

- 인코더의 출력  $h$ 를 디코더의 Time LSTM 계층의 상태로 설정
- 따라서 인코더와는 달리  $\text{stateful}=\text{True}$
- 한번 설정된 이 은닉 상태는 자설정되지 않고, 인코더의  $h$ 를 유지하면서 순전파가 이루어진다.

```
def forward(self, xs, h):
    self.lstm.set_state(h)

    out = self.embed.forward(xs)
    out = self.lstm.forward(out)
    score = self.affine.forward(out)
    return score

def backward(self, dscore):
    dout = self.affine.backward(dscore)
    dout = self.lstm.backward(dout)
    dout = self.embed.backward(dout)
    dh = self.lstm.dh # dh: LSTM 계층의 시간
                      방향으로의 기울기
    return dh
```

- backward():

- 위쪽의 Softmax with Loss 계층으로부터 기울기  $dscore$ 를 받아 Time Affine, Time LSTM, Time Embedding 계층 순서로 전파
- $dh$ : LSTM 계층의 시간 방향으로의 기울기
  - 이 기울기를 꺼내 Decoder 클래스의 backward() 의 출력으로 반환

## 7.3.2 Decoder 클래스

- Decoder 클래스:

문장 생성을 담당하는 메서드

```
def generate(self, h, start_id, sample_size):
    sampled = []
    sample_id = start_id
    self.lstm.set_state(h)

    for _ in range(sample_size):
        x = np.array(sample_id).reshape((1, 1))  문자를 한개씩 주고
        out = self.embed.forward(x)
        out = self.lstm.forward(out)
        score = self.affine.forward(out)

        sample_id = np.argmax(score.flatten()) Affine 계층이 출력하는 점수가 가장 큰 문자 ID를 선택
        sampled.append(int(sample_id))

    return sampled
```

- h: 인코더로부터 받는 은닉 상태
- start\_id: 최초로 주어지는 문자 ID
- sample\_size: 생성하는 문자 수

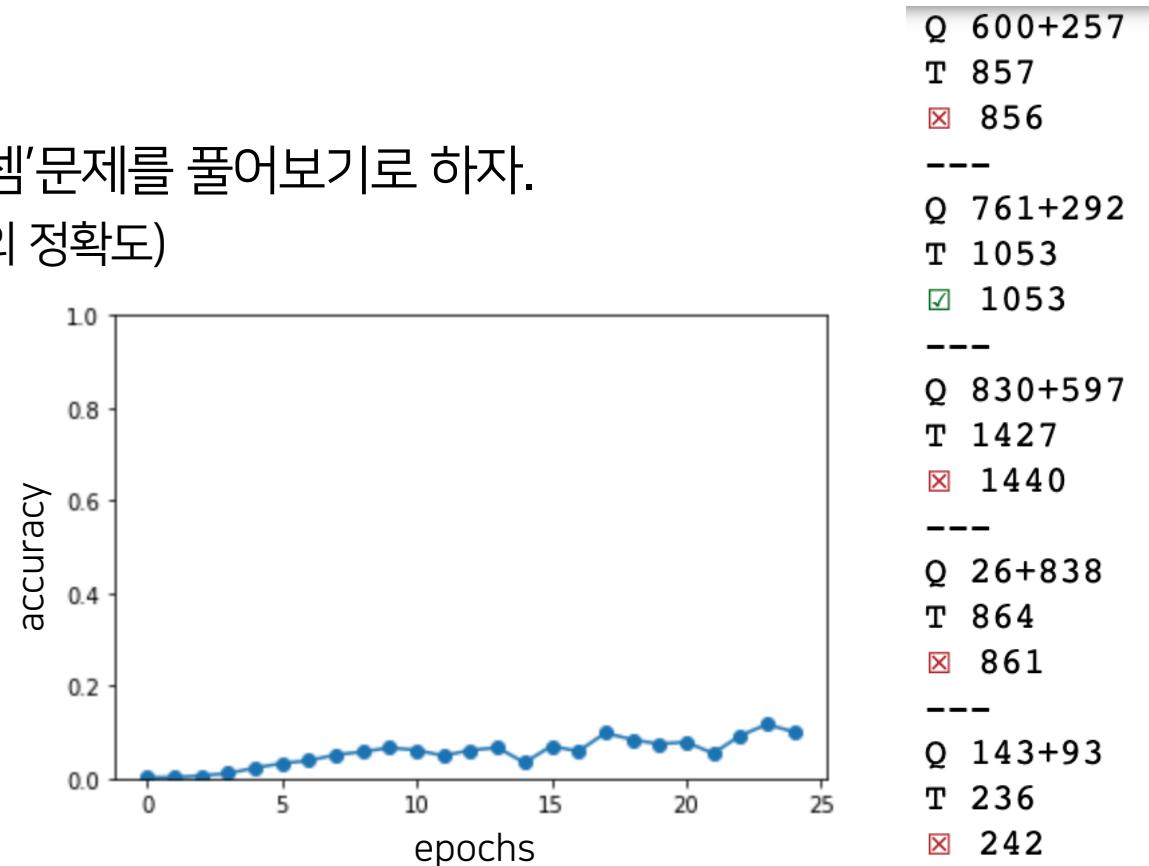
### 7.3.3 Seq2seq 클래스

- Seq2seq 클래스:

```
class Seq2seq(BaseModel):  
    def __init__(self, vocab_size, wordvec_size, hidden_size):  
        V, D, H = vocab_size, wordvec_size, hidden_size  
        self.encoder = Encoder(V, D, H)  
        self.decoder = Decoder(V, D, H)  
        self.softmax = TimeSoftmaxWithLoss()  
  
        self.params = self.encoder.params + self.decoder.params  
        self.grads = self.encoder.grads + self.decoder.grads  
  
    def forward(self, xs, ts):  
        decoder_xs, decoder_ts = ts[:, :-1], ts[:, 1:]  
  
        h = self.encoder.forward(xs)  
        score = self.decoder.forward(decoder_xs, h) encoder 클래스와 decoder 클래스를 연결  
        loss = self.softmax.forward(score, decoder_ts)  
        return loss  
  
    def backward(self, dout=1):  
        dout = self.softmax.backward(dout)  
        dh = self.decoder.backward(dout)  
        dout = self.encoder.backward(dh)  
        return dout  
  
    def generate(self, xs, start_id, sample_size):  
        h = self.encoder.forward(xs)  
        sampled = self.decoder.generate(h, start_id, sample_size)  
        return sampled
```

## 7.3.4 seq2seq 평가

- seq2seq의 학습은 기본적인 신경망의 학습과 같은 흐름으로 이뤄진다.
  1. 학습 데이터에서 미니배치 선택
  2. 미니배치로부터 기울기를 계산
  3. 기울기를 사용해 매개변수를 갱신
- 앞에서 만든 seq2seq 클래스를 이용해 '덧셈' 문제를 풀어보기로 하자.
  - 평가 척도: 정답률(에폭마다 테스트 데이터의 정확도)
  - 정답 : 정답 / ✗: 오답
  - 에폭이 거듭함에 따라 정답률이 증가하고는 있지만, 25 에폭까지의 정답률은 10%
  - 학습을 진행할수록 더 정확해질 여지 0
- 같은 문제를 더 잘 학습할 수 있도록 seq2seq를 개선해보자!



# 7.4 seq2seq 개선:

## 1. 입력 데이터 반전(Reverse)

- 입력 데이터의 순서를 반전시킨다[41]
  - 학습 진행이 빨라져 결과적으로 최종 정확도도 좋아진다.

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

→

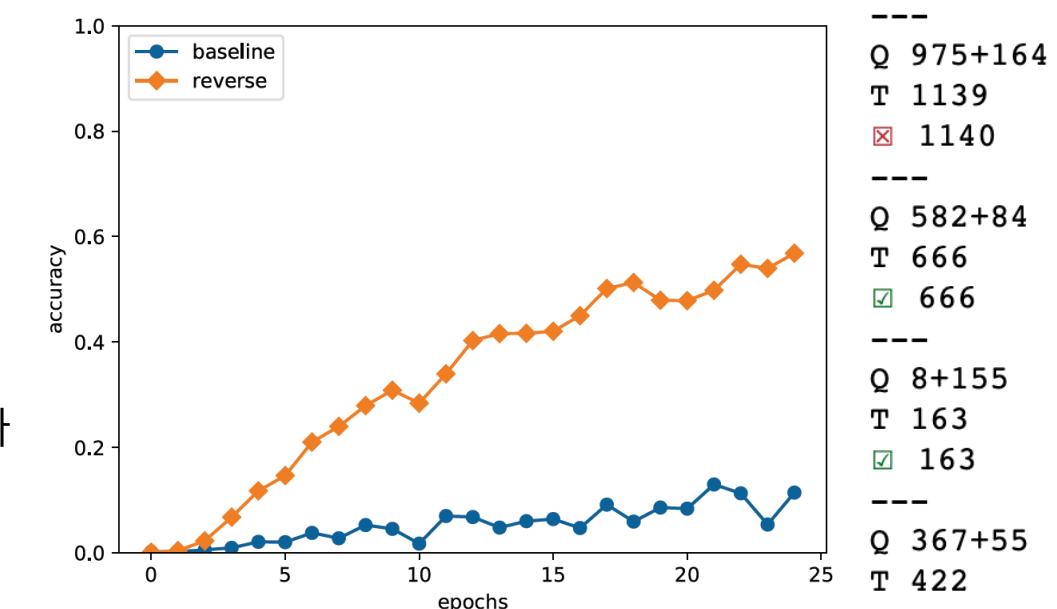
			5	+	7	5
1	2	5	+	8	2	6
	8	+	0	2	2	

- 배열의 행을 반전시키려면 다음과 같은 표기법을 사용하면 된다.

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

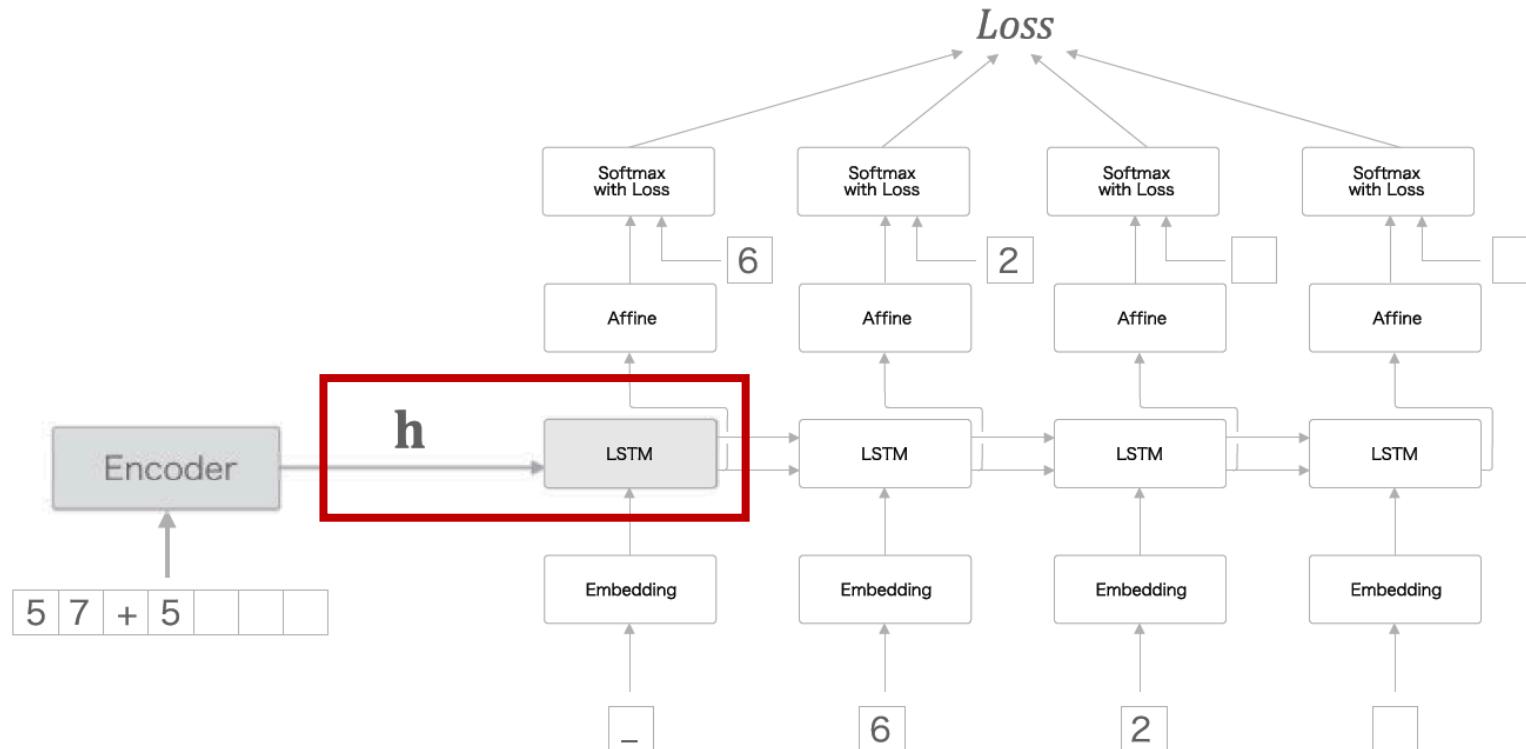
- 단지 입력 데이터를 반전시킨 것 만으로 학습 진행이 개선

- 25에폭: 정답률 50%
- **기울기 전파가 원활해지기 때문**
- eg) “나는 고양이로소이다” -> “I am a cat”
  - ‘나’ -> ‘I’로 변환되는 과정에서, ‘나’가 ‘I’까지 가려면 ‘는’, ‘고양이’, ‘로소’, ‘이다’까지 총 네 단어 분량의 LSTM을 거침
  - 하지만 입력문을 반전시킬 경우(이다 로소 고양이 는 나), ‘나’와 ‘I’는 바로 옆이 되었으므로 기울기가 직접 전해짐
  - 이처럼 입력 문장의 첫 부분에서 대응하는 단어와 가까워지므로 기울기가 더 잘 전해져서 학습 효율이 좋아진다.



## 7.4.2 엿보기(Peeky)

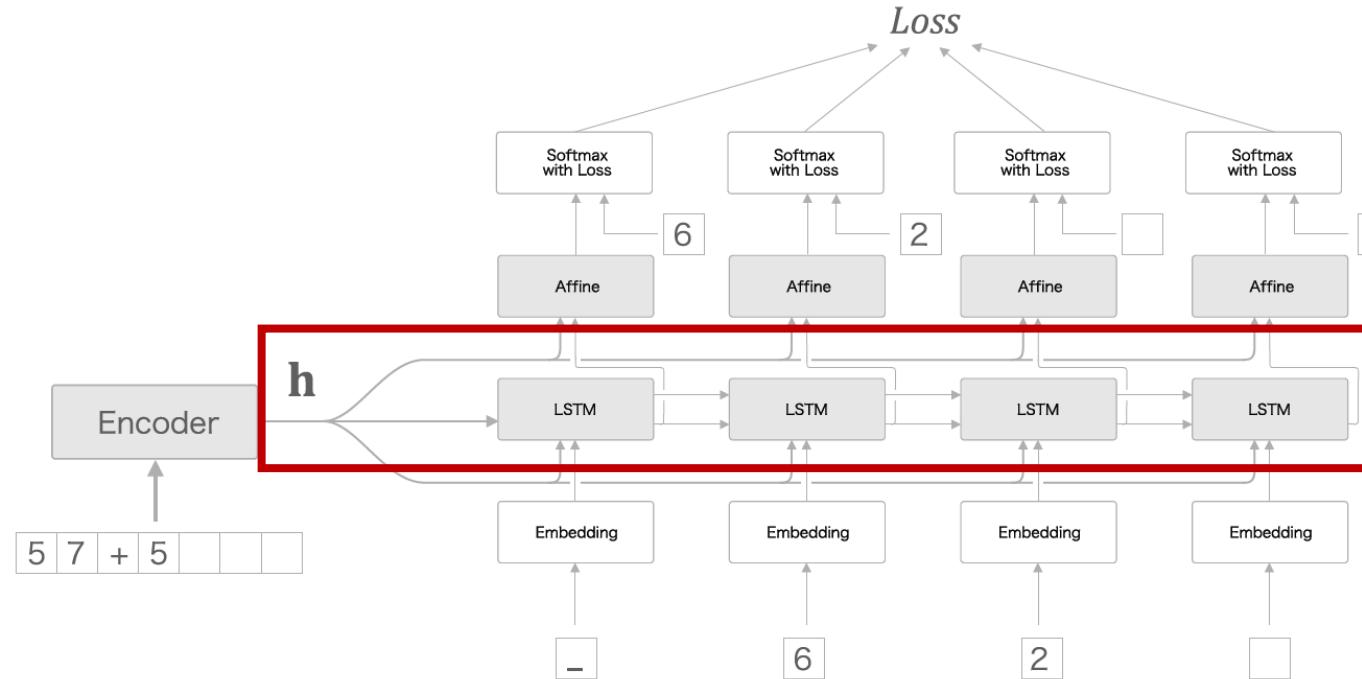
- seq2seq의 인코더: 입력 문장을 고정 길이 벡터  $h$ 로 변환
  - 이때 이  $h$  안에는 디코더에게 필요한 정보가 모두 담겨 있음
  - 즉,  $h$ 가 디코더에 있어서는 유일한 정보
- 하지만 현재의 seq2seq는 **최초 시각의 LSTM 계층만이 벡터  $h$ 를 이용하고 있음**



[42] Cho, Kyunghyun, et al: "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation." arXiv preprint arXiv: 1406.1078(2014).

## 7.4.2 엿보기(Peeky)

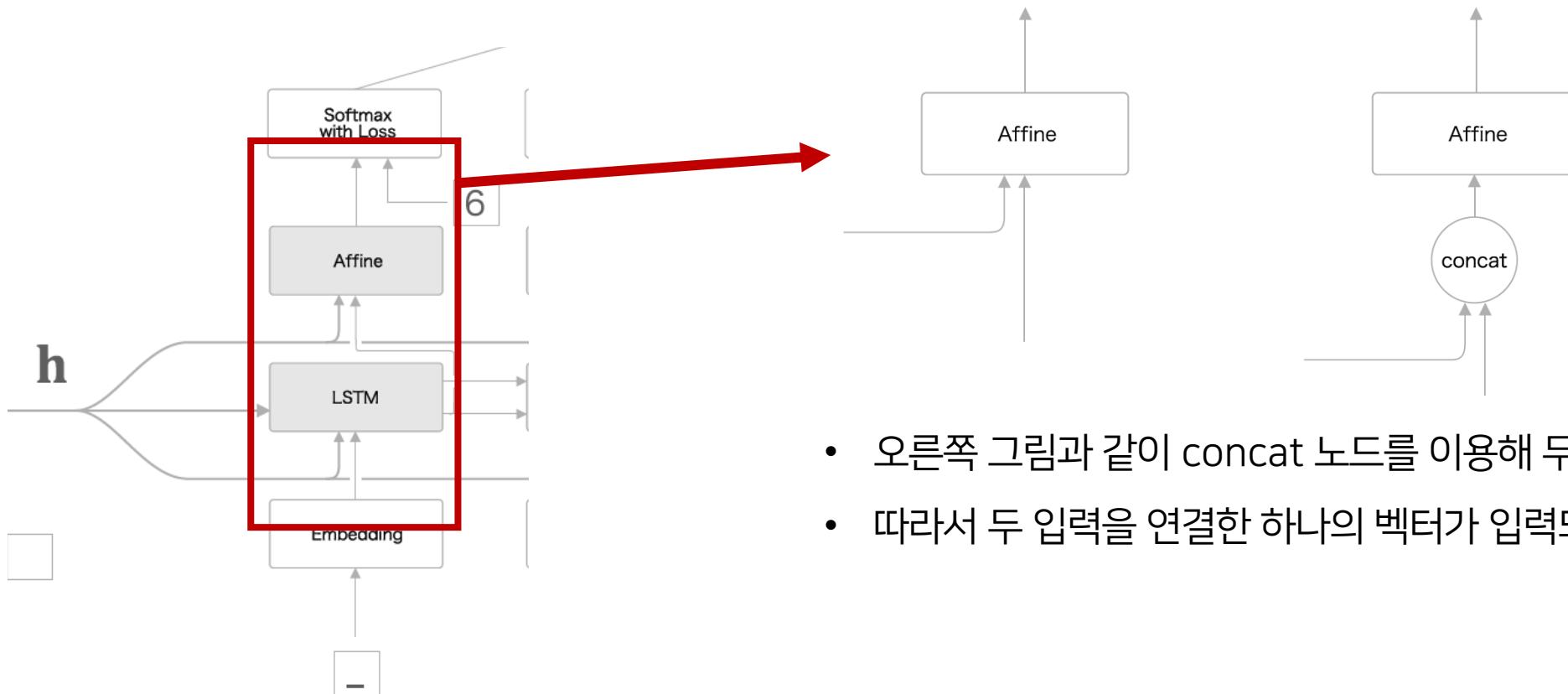
- 이렇게 중요한 정보가 담긴 인코더의 출력  $h$ 를 디코더의 다른 계층에도 전해주자.



- 기존에는 하나의 LSTM만이 소유하던  $h$ 를 여러 계층이 공유
  - 다른 계층도 인코딩된 정보를 '엿본다'라고 해석 가능
  - 따라서 이 개선을 더한 디코더를 'Peaky Decoder'라고 한다[42]

## 7.4.2 엿보기(Peeky)

- 이때 LSTM 계층과 Affine 계층에는 2개의 벡터가 입력되고 있다.
  - 이는 실제로는 두 벡터가 연결(concatenate)된 것을 의미



## 7.4.2 엿보기(Peeky)

- Peeky Decoder 클래스:

```
class PeekyDecoder:  
    def __init__(self, vocab_size, wordvec_size, hidden_size):  
        V, D, H = vocab_size, wordvec_size, hidden_size  
        rn = np.random.randn  
  
        embed_W = (rn(V, D) / 100).astype('f')  
        lstm_Wx = (rn(H + D, 4 * H) / np.sqrt(H + D)).astype('f')  
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')  
        lstm_b = np.zeros(4 * H).astype('f')  
        affine_W = (rn(H + H, V) / np.sqrt(H + H)).astype('f')  
        affine_b = np.zeros(V).astype('f')  
  
        self.embed = TimeEmbedding(embed_W)  
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=True)  
        self.affine = TimeAffine(affine_W, affine_b)  
  
        self.params, self.grads = [], []  
        for layer in (self.embed, self.lstm, self.affine):  
            self.params += layer.params  
            self.grads += layer.grads  
        self.cache = None
```

Encoder가 인코딩한 벡터도  
입력되기 때문에 가중치 매개  
변수의 형상이 그만큼 커짐

```
def forward(self, xs, h):  
    N, T = xs.shape  
    N, H = h.shape  
  
    self.lstm.set_state(h)  
  
    out = self.embed.forward(xs) h를 시계열만큼 복제해 저장  
    hs = np.repeat(h, T, axis=0).reshape(N, T, H)  
    out = np.concatenate((hs, out), axis=2) hs와 Embedding 계층의 출력을 연결  
    out = self.lstm.forward(out)  
    out = np.concatenate((hs, out), axis=2) hs와 LSTM 계층의 출력을 연결  
    score = self.affine.forward(out)  
    self.cache = h  
    return score
```

- 이때 Encoder 부분은 이전과 같기 때문에 그대로 사용한다.

## 7.4.2 엿보기(Peeky)

- Peeky Decoder 클래스:

```
def backward(self, dscore):
    H = self.cache

    dout = self.affine.backward(dscore)
    dout, dhs0 = dout[:, :, H:], dout[:, :, :H]
    dout = self.lstm.backward(dout)
    dembed, dhs1 = dout[:, :, H:], dout[:, :, :H]
    self.embed.backward(dembed)

    dhs = dhs0 + dhs1
    dh = self.lstm.dh + np.sum(dhs, axis=1)
    return dh
```

```
def generate(self, h, start_id, sample_size):
    sampled = []
    char_id = start_id
    self.lstm.set_state(h)

    H = h.shape[1]
    peeky_h = h.reshape(1, 1, H)
    for _ in range(sample_size):
        x = np.array([char_id]).reshape((1, 1))
        out = self.embed.forward(x)

        out = np.concatenate((peeky_h, out), axis=2)
        out = self.lstm.forward(out)
        out = np.concatenate((peeky_h, out), axis=2)
        score = self.affine.forward(out)

        char_id = np.argmax(score.flatten())
        sampled.append(char_id)

    return sampled
```

- 이때 Encoder 부분은 이전과 같기 때문에 그대로 사용한다.

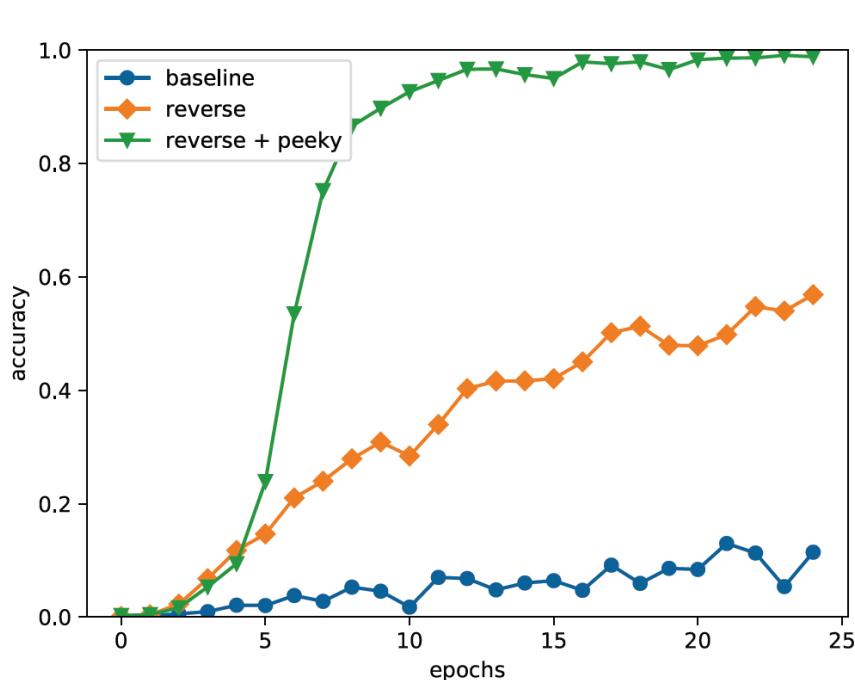
## 7.4.2 엿보기(Peeky)

- PeekySeq2seq 클래스:

```
class PeekySeq2seq(Seq2seq):  
    def __init__(self, vocab_size, wordvec_size, hidden_size):  
        V, D, H = vocab_size, wordvec_size, hidden_size  
        self.encoder = Encoder(V, D, H)  
        self.decoder = PeekyDecoder(V, D, H)  PeekyDecoder를 사용  
        self.softmax = TimeSoftmaxWithLoss()  
  
        self.params = self.encoder.params + self.decoder.params  
        self.grads = self.encoder.grads + self.decoder.grads
```

## 7.4.2 엿보기(Peeky)

- Peeky를 추가로 적용하자 결과가 월등히 좋아짐
  - 10 에폭을 넘어서면서 정답률이 90%를 넘고, 최종적으로는 100%에 가까워졌다!



Q 77+85	T 162	162
---		
Q 975+164	T 1139	1139
---		
Q 582+84	T 666	666
---		
Q 8+155	T 163	163
---		
Q 367+55	T 422	422

Peeky를 이용하게 되면 신경망의 가중치 매개변수가 커지기 때문에 계산량도 늘어난다. 따라서 이번 절의 실험 결과는 커진 매개변수만큼의 핸디캡을 감안해야 하며, 또 seq2seq의 정확도는 하이퍼파라미터에 영향을 크게 받기 때문에 실험 시 주의해야 한다.

- 이렇게 Reverse와 Peeky를 사용해 seq2seq를 개선
  - 다음 장에서 **Attention**으로 더 크게 개선해보자!

# 7.5 seq2seq를 이용하는 애플리케이션: 1. 챗봇

- **챗봇**: 사람과 컴퓨터가 텍스트로 대화를 나누는 프로그램
  - 대화는 '상대의 말'과 '자신의 말'로 구성
  - 따라서 '상대의 말'을 '자신의 말'로 변환하는 문제로 볼 수 있다. => seq2seq

그림 7-29 seq2seq 기반 챗봇의 대화 예(문헌 [43]에서 발췌해 우리말로 번역함)

기계: 안녕하세요.  
사람: 안녕하세요.  
기계: 어떤 OS를 사용하시죠?  
사람: 리눅스요.  
.....  
기계: 실행할 때 어떤 오류가 발생하죠?  
사람: VPN 어찌고 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.



## 7.5.2 알고리즘 학습

- 소스코드도 자연어와 마찬가지로 문자로 쓰여진 시계열 데이터
  - 몇 줄에 걸친 코드라도 하나의 문장으로 처리할 수 있다.
  - 따라서 소스코드를 그대로 seq2seq에 입력할 수 있으며, 원하는 답과 대조하여 학습이 가능하다.

그림 7-30 파이썬으로 작성된 코드의 예: Input은 입력, Target은 출력(문헌 [44]에서 발췌)

**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.

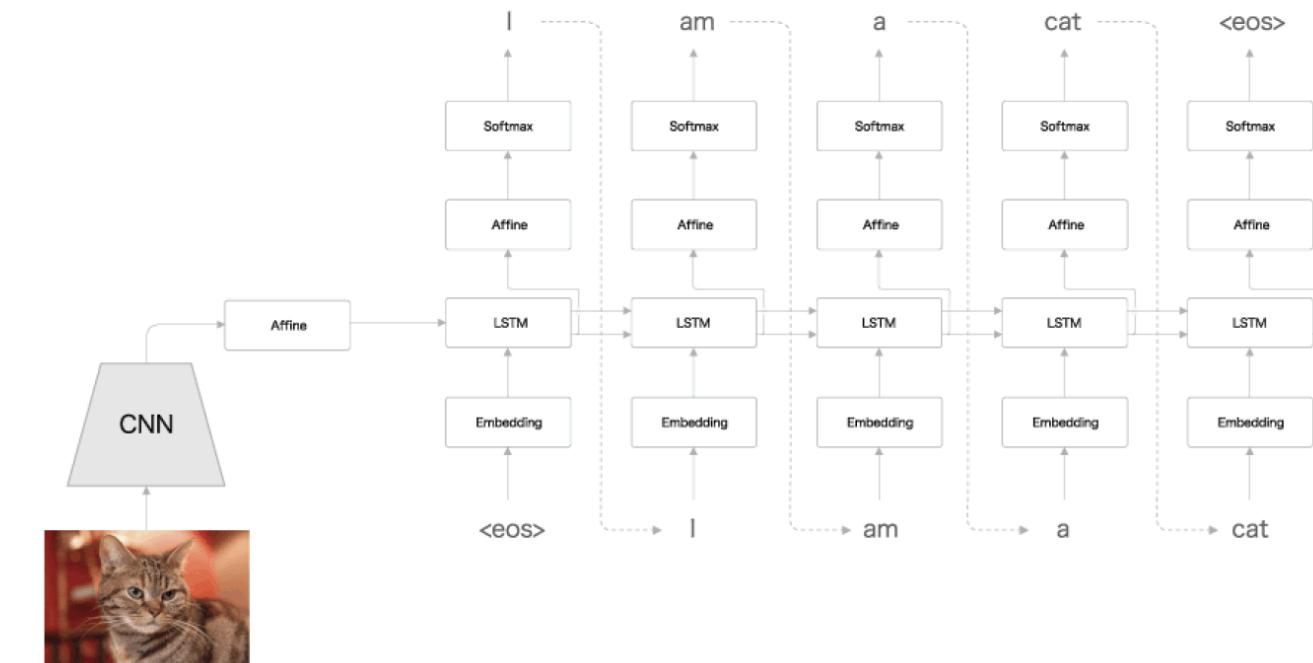
- 다음 장에서는 RNN을 확장한 NTM(Neural Turing Machine) 모델을 소개
  - NTM 모델로 컴퓨터(튜링 머신)가 메모리를 읽고 쓰는 순서를 학습하여 알고리즘을 재현

[45] Vinyals, Oriol, et al: "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on, IEEE, 2015

[46] Karpathy, Andrej, and Li Fei-Fei: "Deep Visual-Semantic Alignments for Generating Image Descriptions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

## 7.5.3 이미지 캡셔닝

- seq2seq는 텍스트 외에도 이미지나 음성 등 다양한 데이터 처리 가능
- **이미지 캡셔닝(Image Captioning)[45][46]:** '이미지'를 '문장'으로 변환하는 기법
  - 인코더로 **CNN**을 사용, 디코더는 이전과 같음
    - CNN을 이용해 이미지를 인코딩하며, 최종 출력은 특징 맵(feature map)
    - 이 특징 맵은 3차원(높이\*폭\*채널)이므로 디코더의 LSTM이 처리할 수 있도록 변환
      - 특징 맵을 1차원으로 평탄화(flattening)한 후 Affine 계층에서 변환
    - 변환된 데이터를 디코더에 전달



## 7.5.3 이미지 캡셔닝

그림 7-32 이미지 캡셔닝의 예: 이미지를 텍스트로 변환(문헌 [47]에서 발췌)

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 스キー 타는 사람)



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



## 7.6 정리

- RNN을 이용해 문장을 생성해보았다.
  - 6장에서 다른 RNN을 사용한 언어 모델에 문장을 생성하는 기능을 추가
- Encoder와 Decoder를 연결한 seq2seq에 대해 알아보았다.
  - 이를 이용한 간단한 덧셈 문제를 학습
  - 이를 개선하는 두개의 아이디어(Reverse와 Peeky) 구현 및 실험
- 다음 장에서는 'Attention'을 이용해 seq2seq를 한층 더 개선해보자!