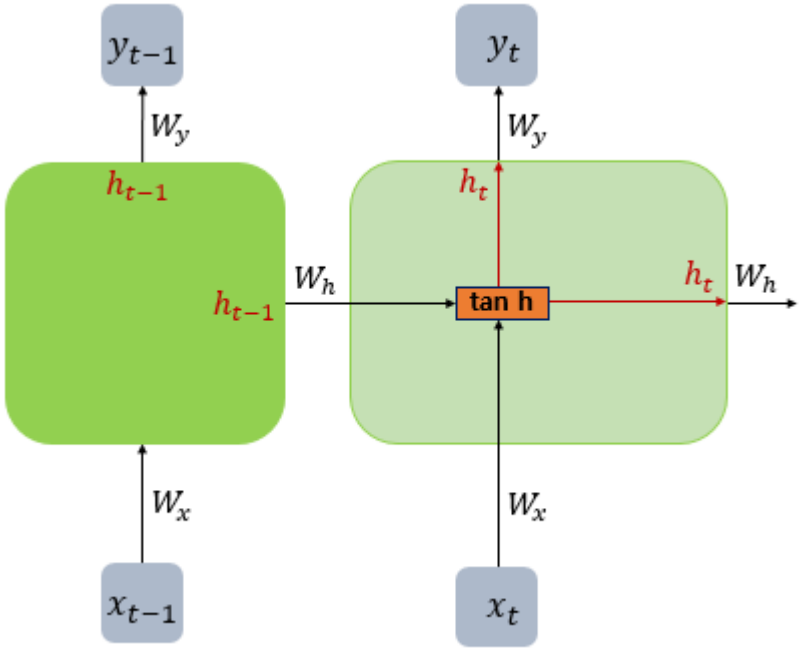
[seq2seq]

17기 사이언스 김찬

[seq2seq]

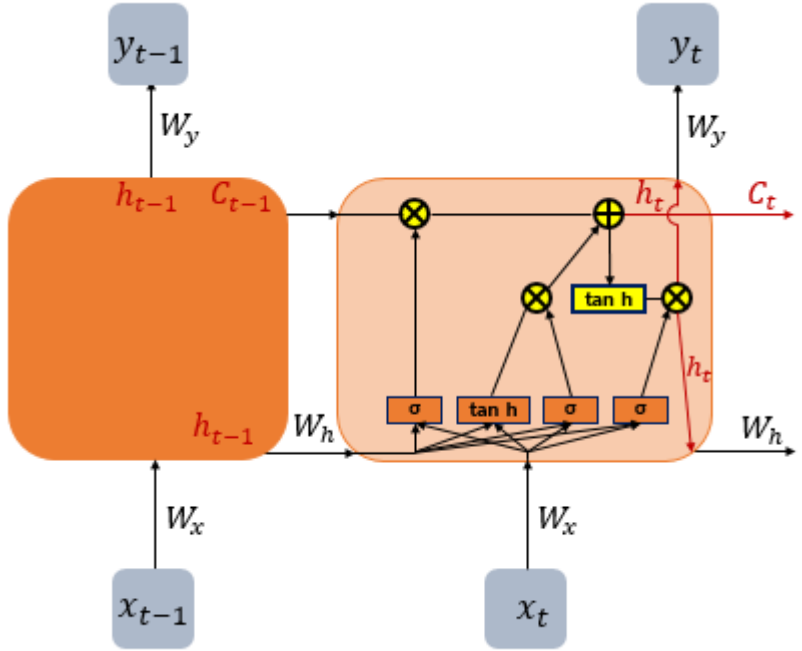
1. RNN, LSTM 간단 복습
2. Seq2seq 모델
3. 실습

[RNN]



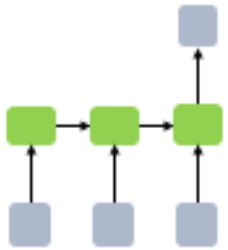
$$h_t = tanh(W_x x_t + W_h h_{t-1} + b)$$

[LSTM]



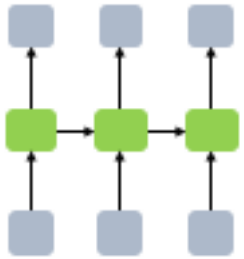
$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i)$$
$$g_t = tanh(W_{xg} x_t + W_{hg} h_{t-1} + b_g)$$
$$f_t = \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f)$$
$$C_t = f_t \circ C_{t-1} + i_t \circ g_t$$
$$o_t = \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o)$$
$$h_t = o_t \circ tanh(c_t)$$

[RNN 유형]



다 대 일(many-to-one)

언어모델
- 일련의 단어를 집어 넣으면, 맨 마지막 단어에 무엇이 들어올지 정하는 분류 문제
- 맨 마지막 타임 스텝에서의 출력만 필요



다 대 다(many-to-many)

번역기
- 일련의 단어를 집어 넣으면, 그에 해당하는 일련의 단어들을 결정하는 문제
- 전체 타임 스텝에서의 출력만 필요

[return_sequnces, return_state]

```
train_X = [[[0.1, 4.2, 1.5, 1.1, 2.8], [1.0, 3.1, 2.5, 0.7, 1.1], [0.3, 2.1, 1.5, 2.1, 0.1], [2.2, 1.4, 0.5, 0.9, 1.1]]]
train_X = np.array(train_X, dtype=np.float32)
print(train_X.shape)
```

```
(1, 4, 5)
```

```
rnn = SimpleRNN(3)
# rnn = SimpleRNN(3, return_sequences=False, return_state=False)와 동일.
hidden_state = rnn(train_X)

print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
```

```
hidden state : [[-0.866719    0.95010996 -0.99262357]], shape: (1, 3)
```

[return_sequnces, return_state]

```
train_X = [[[0.1, 4.2, 1.5, 1.1, 2.8], [1.0, 3.1, 2.5, 0.7, 1.1], [0.3, 2.1, 1.5, 2.1, 0.1], [2.2, 1.4, 0.5, 0.9, 1.1]]]
train_X = np.array(train_X, dtype=np.float32)
print(train_X.shape)
```

```
(1, 4, 5)
```

```
lstm = LSTM(3, return_sequences=False, return_state=True)
hidden_state, last_state, last_cell_state = lstm(train_X)

print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
print('last hidden state : {}, shape: {}'.format(last_state, last_state.shape))
print('last cell state : {}, shape: {}'.format(last_cell_state, last_cell_state.shape))
```

```
hidden state : [[-0.00263056  0.20051427 -0.22501363]], shape: (1, 3)
last hidden state : [[-0.00263056  0.20051427 -0.22501363]], shape: (1, 3)
last cell state : [[-0.04346419  0.44769213 -0.2644241 ]], shape: (1, 3)
```

[return_sequnces, return_state]

```
train_X = [[[0.1, 4.2, 1.5, 1.1, 2.8], [1.0, 3.1, 2.5, 0.7, 1.1], [0.3, 2.1, 1.5, 2.1, 0.1], [2.2, 1.4, 0.5, 0.9, 1.1]]]
train_X = np.array(train_X, dtype=np.float32)
print(train_X.shape)
```

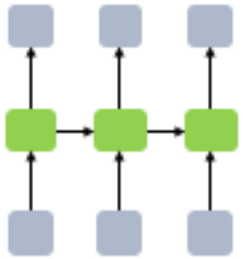
```
(1, 4, 5)
```

```
lstm = LSTM(3, return_sequences=True, return_state=True)
hidden_states, last_hidden_state, last_cell_state = lstm(train_X)

print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
print('last hidden state : {}, shape: {}'.format(last_hidden_state, last_hidden_state.shape))
print('last cell state : {}, shape: {}'.format(last_cell_state, last_cell_state.shape))
```

```
hidden states : [[[ 0.1383949   0.01107763 -0.00315794]
  [ 0.0859854   0.03685492 -0.01836833]
  [-0.02512104  0.12305924 -0.0891041 ]
  [-0.27381724  0.05733536 -0.04240693]]], shape: (1, 4, 3)
last hidden state : [[-0.27381724  0.05733536 -0.04240693]], shape: (1, 3)
last cell state : [[-0.39230722  1.5474017  -0.6344505 ]], shape: (1, 3)
```

[seq2seq]



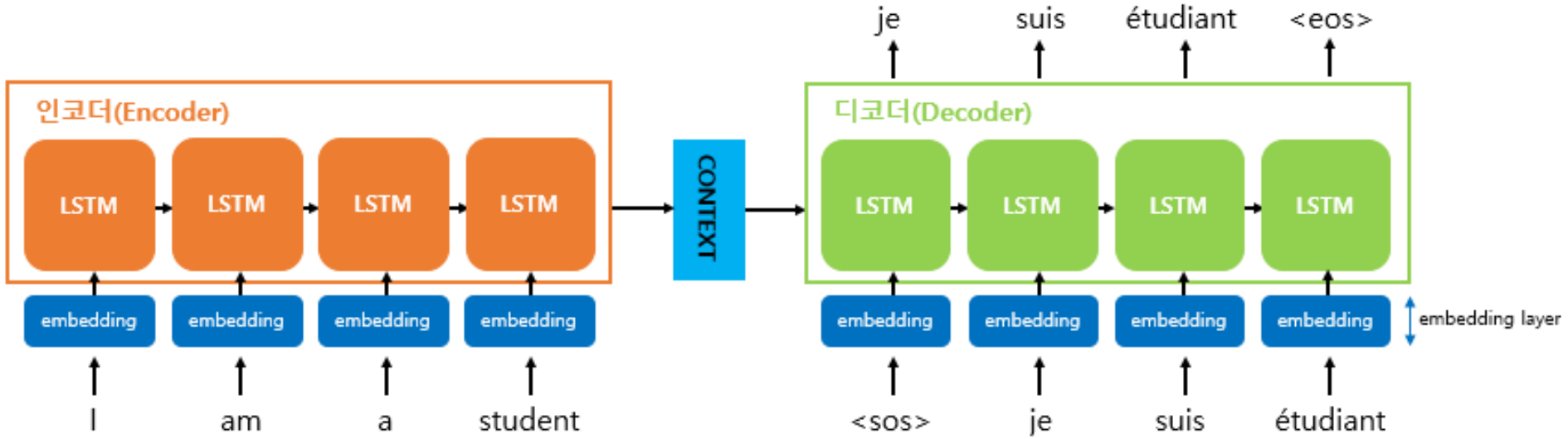
다 대 다(many-to-many)

입력 문장 길이 (입력 데이터의 타임 스텝) ≠ 출력 문장 길이 (출력 데이터의 타임 스텝)

My cat has one blue eye, one gold eye.
-> 내 고양이는 파란 눈과 금빛 눈을 가지고 있다.

Encoder – Decoder 형식의 모델로 문제 해결!

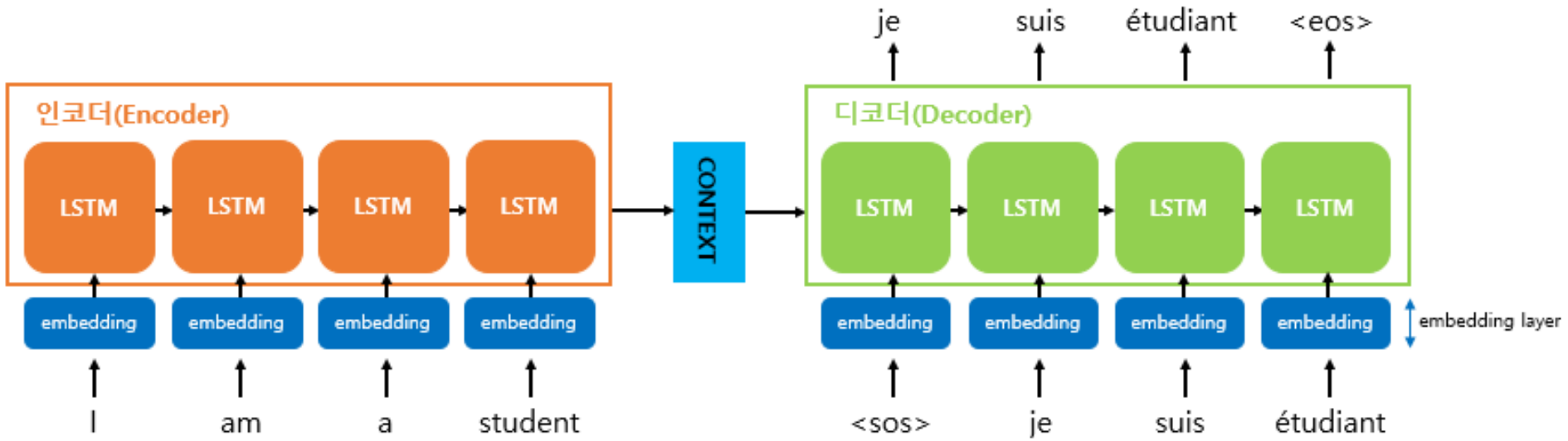
[seq2seq]



Encoder
- 입력 문장에 해당하는 context 형성 (last cell state, last hidden state)

Decoder
- 초기 상태의 cell state, hidden state <- context 로 설정
- 일종의 bigram, 입력 단어 다음으로 나올 단어 출력

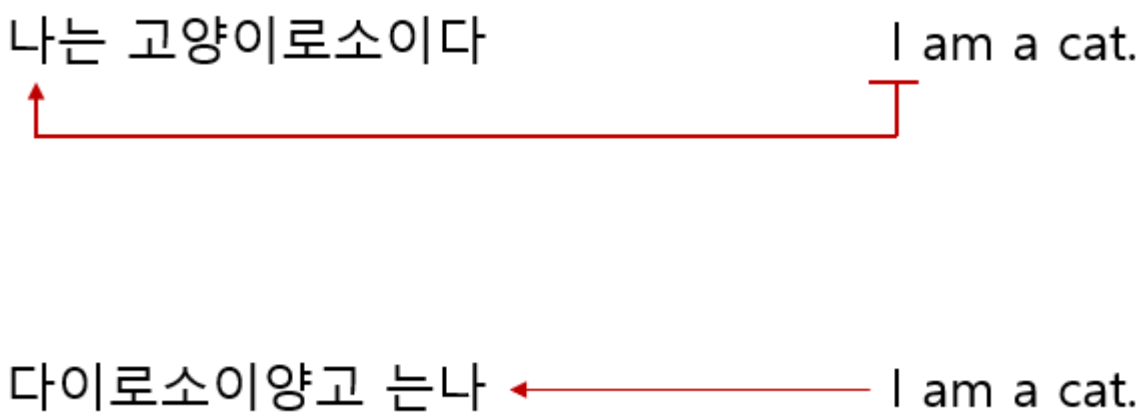
[교사 강요]



훈련 시 decoder 입출력
- 입력 : <sos> + sentence
- 출력 : sentence + <eos>

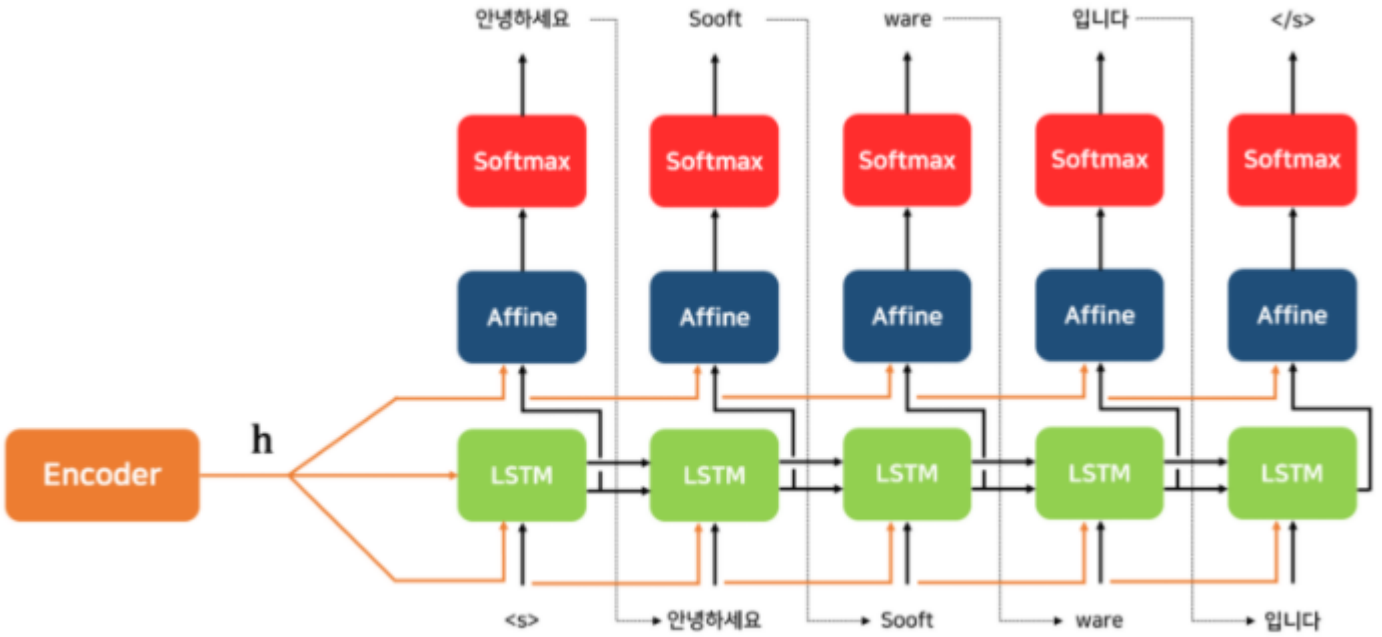
테스트 시 decoder 입출력
- 입력 : 이전 타임 스텝에서의 출력 단어
- 출력 : 다음에 올 단어

[개선안 - reverse]



입력 데이터 반전
- 입력 데이터의 앞 쪽과, 출력 데이터의 앞 쪽의 거리가 짧아져서 학습속도 및 정확도 상승
- 물론 평균적인 거리는 그대로
- 시계열 데이터에서 앞 쪽의 예측 결과가 좋으면 뒤 쪽도 좋은 결과로 이어지는 경우가 많기 때문이라 '추정'
- "Sequence to sequence learning with neural networks." Advances in neural information processing system. 2014.

[개선안 - peeky]



Context vector가 맨 처음 state로만 쓰이는 것이 아니라, 매 타임 스텝에서의 입력과 함께 들어감
- context vector는 입력 데이터의 정보를 압축한 vector이므로, 이것이 매번 들어가면 더 좋은 효과를 나타냄
- "learning phrase representation using RNN encoder-decoder for statistical machine translation" Cho, Kyunhyun 2014.

[실습 간단 소개]

plus.txt

```
468+797 1265
646+744 1390
949+470 1419
860+513 1373
542+772 1314
244+757 1001
117+357 474
379+304 683
290+621 911
668+564 1232
587+183 770
543+506 1049
196+549 745
736+813 1549
674+432 1106
133+24  157
977+692 1669
270+549 819
418+603 1021
487+611 1098
```

목표
- 'A+B' 에 해당하는 문자열을 입력하면, 그 결과를 반환
- "더하기 번역기" 모델
- "더하기 번역기" 모델 + peeky

데이터 파일
- 'A+B'와 C(=A+B)가 ₩t를 기준으로 나누어져 있음
- 0<= A,B <= 999
- 0 <= C <= 1998
- 총 5만개

[출처]

- Wikidocs.net – 딥 러닝을 이용한 자연어 처리 입문 (9장, 15장)
- https://velog.io/@dscwinterstudy – [밑바닥 부터 시작하는 딥러닝2] 7장. RNN을 사용한 문장 생성
- https://blog.naver.com/PostView.nhn?blogId=sooftware&logNo=221784419691
- http://www.manythings.org/anki/