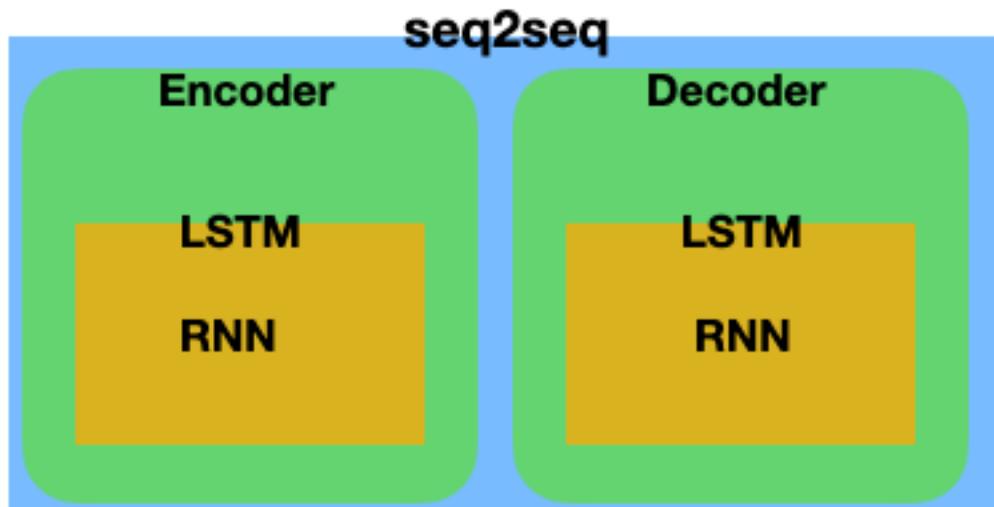


## 시퀀스 투 시퀀스 seq2seq

### 1. Seq2seq 이란

우리는 시퀀스 형태의 입력을 처리할 때 RNN이 효과적이라는 것을 알고 있다. 이번에 우리는 시퀀스 형태의 자료를 입력 받은 뒤 다시 시퀀스 형태로 출력하는 모델에 대해 알아볼 것이다. 이러한 작업을 신경망을 사용하여 수행하는 대표적인 모델은 seq2seq인 것 같다. 이것은 특히 기계 번역 MT(Machine Translation) 분야에서 자주 사용된다. 기계 번역 분야에서는 이 NMT(Neural Machine Translation)의 일종인 seq2seq가 고안된 이후에, 비약적인 발전이 나타났다. 이것이 도입되기 이전에 기계 번역은 SMT(Statistical Machine Translation) 방법론에 의존했는데, 이것은 사용하기에 매우 까다로웠다. 이것은 매우 복잡한 구조를 가졌고, 유저에게 많은 일을 요구했기 때문이다. 하지만 NMT를 사용한 이후부터 유저들은 대개의 경우에 더 좋은 성능을 갖게 되었고, 더 편리하게 기계 번역 기를 만들 수 있었다.

그렇다면, 소개받은 이 seq2seq란 무엇인가? 이것의 구성은 아래의 그림이 잘 요약해주는 것 같다.

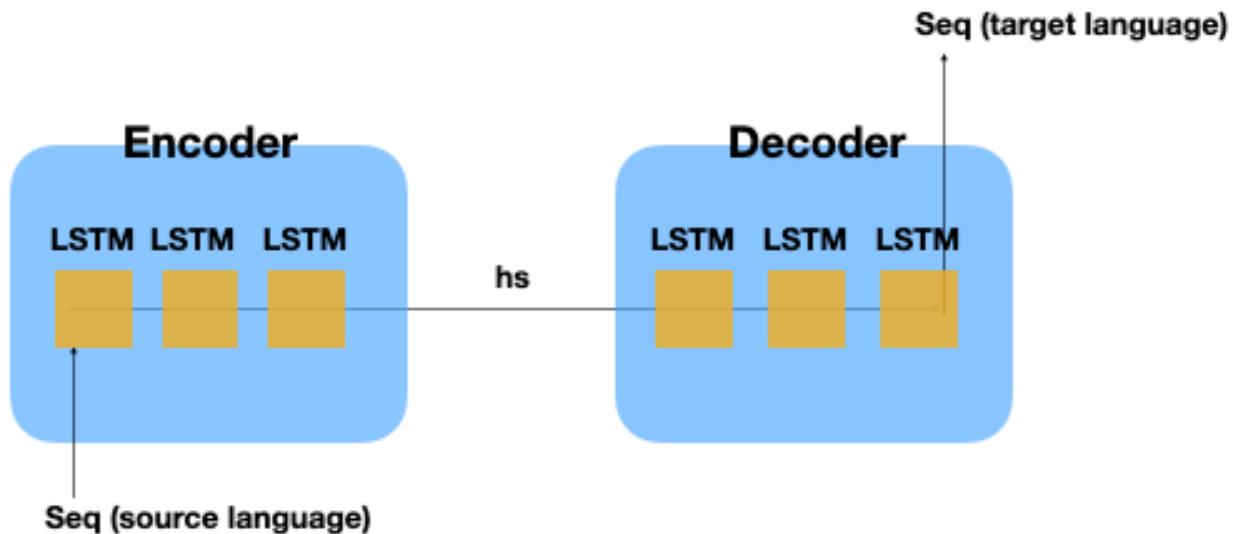


seq2seq는 Encoder와 Decoder로 단 두 개로 구성되어 있다. 주목할 점은 이들은 (논문에 따를 때) RNN의 한 종류인 LSTM로 구성된다는 것이다.<sup>1</sup> 나는 앞에서 seq2seq는 시퀀스 자료를 입력 받아 적절히 처리한 뒤에 다시 시퀀스 자료를 출력한다고 말했다. seq2seq가 이것을 처리하는 과정에 대한 개괄적인 이해는 아래 그림을 통해 얻을 수 있을 것 같다:

---

<sup>1</sup> 이후에 소개된 바에 따르면, LSTM 보다 GRU를 사용 시 더 좋은 성능을 보인다고 한다.

# seq2seq



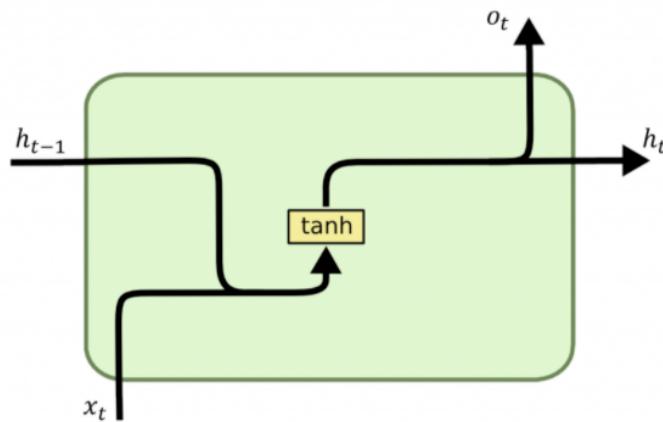
- (1) seq2seq는 인코더 파트에서 (기계 번역 맥락에서 말하자면) 번역하고자 하는 언어로 구성된 시퀀스 형태의 자료를 입력받는다.
- (2) 그것을 LSTM으로 처리한 결과인 은닉 상태 벡터  $hs$  를 디코더의 첫 은닉 상태의 입력 부분으로 전달 해준다.
- (3) 디코더 내에서 적절히 처리된 자료는 마지막에 번역된 언어로 구성된 시퀀스 형태의 자료로 출력된다.

## 2. LSTM (Long Short Term Memory)

seq2seq는 LSTM의 이해만을 요구하는 것 같다. 우리는 LSTM만으로 Encoder와 Decoder를 구성한다는 것을 알았고, 이 모델에서 주목할 한 가지로서 인코더의 결과인 은닉 상태 벡터  $hs$ 가 디코더의 입력 부분으로 전달한다는 것을 알았다. 효과적인 전달을 위해, LSTM에 대해 언급한 뒤 seq2seq으로 나아가는 것은 적절한 것 같다.

## 2.1 왜 LSTM이 요구되는가

LSTM은 RNN에 몇 개의 gate가 추가된 모델이다. 그러한 탓에 일부 프로그래머는 LSTM을 단지 'RNN'이라고 부르기도 한다. 즉, LSTM을 이해하기 위해서는 RNN에 대한 이해가 요구된다. 아래 그림은 하나의 timestep에 대한 RNN을 보여준다.



이전에 보았듯이, RNN은 이전 시점의 은닉 상태와 현 시점의 입력값을 더한 뒤  $\tanh$ 를 적용한 뒤 현 시점의 은닉 상태 벡터  $h_t$ 를 출력한다.

이것의 수식은 다음 같이 표현될 수 있다.

Feed Forward:

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

### RNN의 한계

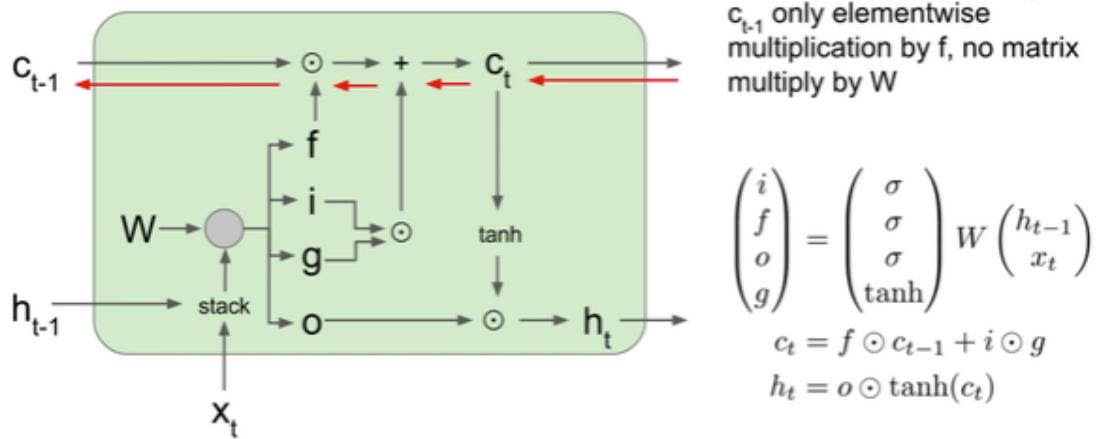
RNN은 시퀀스 데이터를 처리하는 데 효과적이지만, 긴 범위의 시간적 의존성을 가진 data를 처리할 때 **기울기 손실 vanishing gradient** 현상이 나타날 수 있다. 그 까닭은 여러 RNN 계층으로 구성된 어떤 아키텍처에서, 역전파 시에  $\tanh$ 와 MatMul 연산을 지나기 때문이다.  $\frac{dtanh(x)}{dx}$ 의 치역은 0과 1 사이에 있기 때문에, 역전파 시 기울기 grads는 이것을 지날 때마다 작아져서 결국에는 0이 될 것이다. Matmul 연산은 가중치  $W$ 의 초기값을 어떻게 부여하는지에 따라 기울기 손실 또는 기울기 폭발 exploding gradient 현상을 발생하게 만들 수 있다.<sup>2</sup>

<sup>2</sup> 더 자세한 설명은 '밑바닥부터 시작하는 딥러닝2'의 pp. 241 - 245를 보라.

## 위 곤경을 피하는 LSTM

LSTM을 도입하면 위 한계를 피할 수 있다.

### Long Short Term Memory (LSTM): Gradient Flow [Hochreiter et al., 1997]



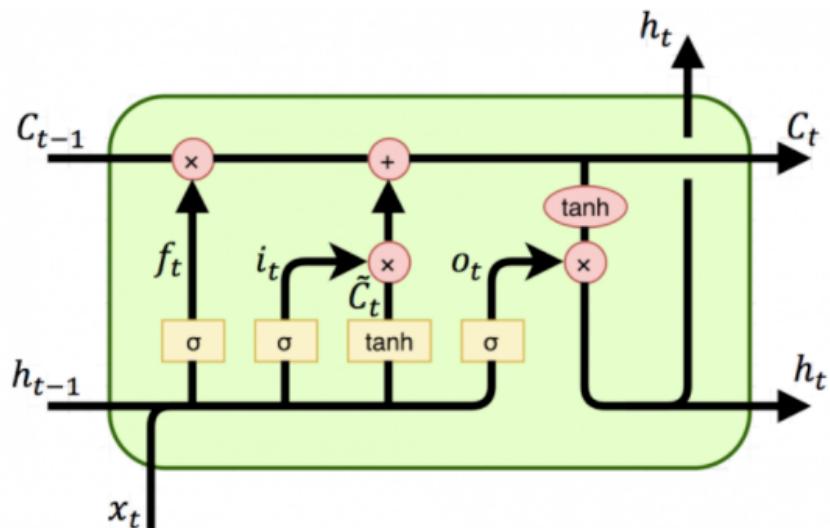
Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 10 - 99 University, May 4, 2017

위 그림에서 보듯이, LSTM에서 도입되는 **기억 셀(memory cell)**은 tanh와 Matmul을 지나지 않기 때문이다. Hamard Product와 덧셈 연산은 기울기 소실 현상을 일으키는 데 어떤 기여도 하지 않는다. 그러한 탓에, 대부분의 경우에 LSTM에서는 기울기 소실 현상을 효과적으로 피할 수 있다.

## 2.2 LSTM 이해하기

먼저, LSTM 전체 구조에 대해 보자.



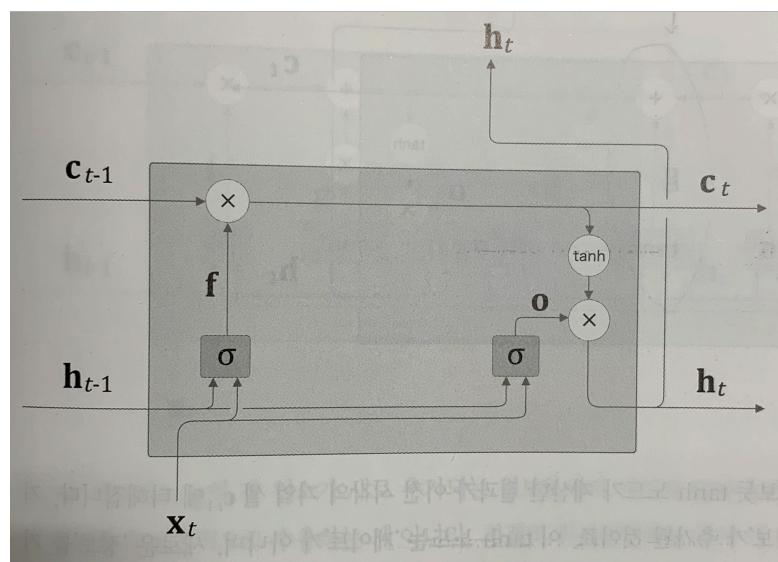
위 그림에서 보듯이, LSTM은 RNN에 4 개의 게이트가 추가되어 있다. 또한, LSTM에서는 기억 셀이 도입되어 있다.  $c$ 로 표현되는 것이 기억 셀들이다.

## 기억 셀 Memory cell

기억 셀은 첫 LSTM에서부터 마지막 LSTM까지 이어지고, 필요한 모든 정보를 담고 있다. 우리는 이것을 이용하여 해당 시점의 은닉 상태 벡터를 구할 수 있다. 더욱이, 앞서 말했듯이, 이것의 도입 덕분에 기울기 소실(및 폭발) 현상을 피해갈 수 있다.

## LSTM의 구조

그것들에 대해 차례대로 알아보자.



아래 파이프 라인을 따라 가보자.  $O$ 와  $h_t$ 로 이어진 라인에 주목해보자. 그것은 Output Gate의 과정을 보여준다.

### (1) Output Gate:

$$O(\text{output}) = \sigma(x_t W_x + h_{t-1} W_h + b)$$

$$h_t = O \odot \tanh(c_t)$$

여기서  $\odot$ 은 element-wise product인 Hamard Product 를 의미한다.

## (2) Forget Gate

이것은 불필요한 정보(기억)을 잊게 하는 역할을 한다. 여기서 불필요한 정보란 오래된 기억을 지칭하는 것 같다.

아까 얘기했던 이전 단어들을 바탕으로 다음 단어를 예측하는 언어 모델 문제로 돌아가보겠다. 여기서 cell state는 현재 주어의 성별 정보를 가지고 있을 수도 있어서 그 성별에 맞는 대명사가 사용되도록 준비하고 있을 수도 있을 것이다. 그런데 새로운 주어가 왔을 때, 우리는 기존 주어의 성별 정보를 생각하고 싶지 않을 것이다.

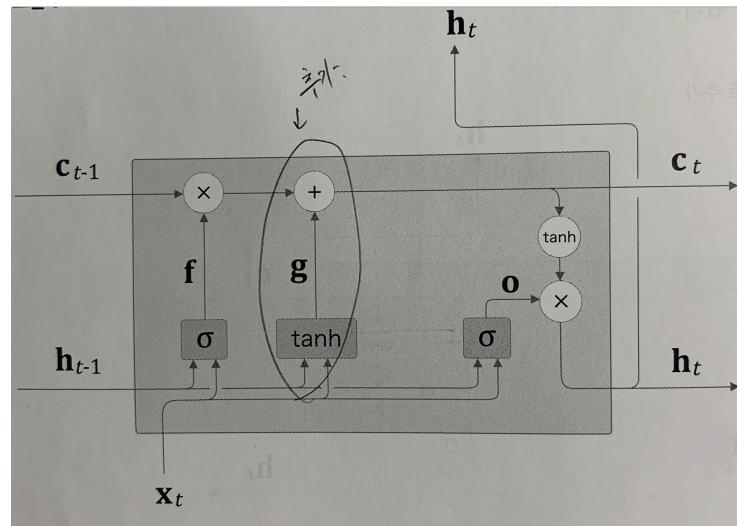
출처: <https://dgkim5360.tistory.com/entry/understanding-long-short-term-memory-lstm-kr> [개발새발로그]

$$f = \sigma(x_t W_x + h_{t-1} W_h + b)$$

여기서  $f$ 는 0부터 1 사이의 값을 갖는데, 0인 경우에는 이전 상태의 정보를 잊고, 1인 경우에는 그 정보를 온전히 보존한 채로 다음 timestep의 state로 전달한다.

$$c = f \odot c_{t-1}$$

## (3) New memory cell

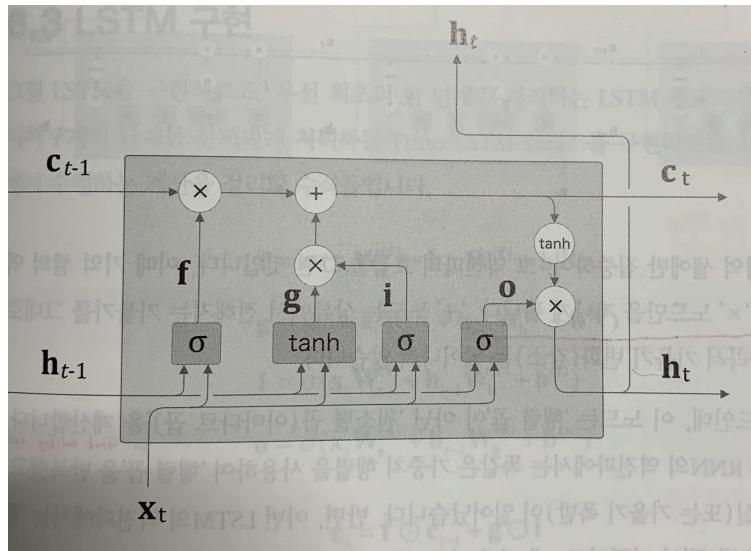


이전 그림에서  $g$  가 있는 선이 새롭게 추가되었다. 이것은 해당 timestep에서 새롭게 기억해야 할 (추가해야 할) 정보를 추가하는 역할을 수행한다.

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

여기서 계산된  $g$ 는 이전 기억 셀  $c_{t-1}$ 과 더해짐으로써 모델에게 새로운 기억(정보)를 추가하게 해준다.

## (4) Input Gate



인풋 게이트는 위에서 계산된  $g$ 의 각 원소가 새로 추가되는 정보로서 얼마나 중요한지를 판단한다. 즉,  $g$ 의 각 원소에 가중치를 부여하는 역할을 수행한다.

$$I = \sigma(x_t W_x + h_{t-1} W_h + b)$$

## LSTM Forward pass 구현

```

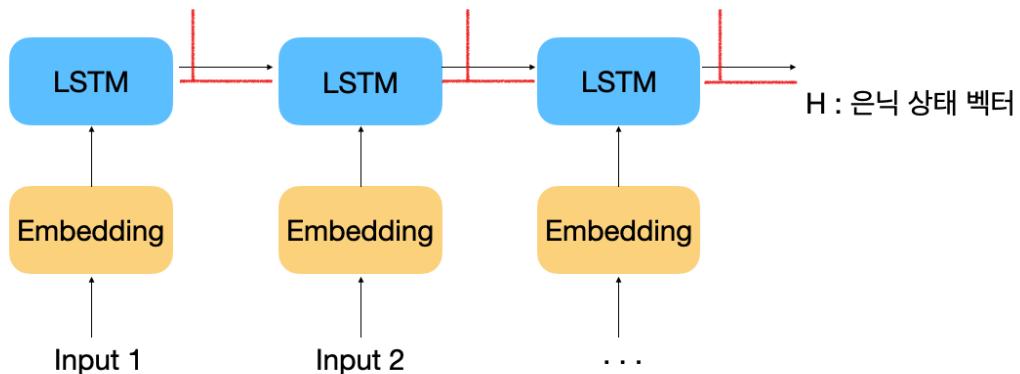
1 import numpy as np
2 import sigmoid
3
4 class LSTM:
5     def __init__(self, Wx, Wh, b):
6         ...
7         Parameters
8         -----
9         Wx: 입력 x에 대한 가중치
10        Wh: 은닉 상태 h에 대한 가중치 매개변수
11        b: 편향
12        ...
13        self.params = [Wx, Wh, b]
14        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
15        self.cache = None
16
17    def forward(self, x, h_prev, c_prev):
18        Wx, Wh, b = self.params
19        N, H = h_prev.shape
20
21        A = np.dot(x, Wx) + np.dot(h_prev, Wh) + b
22
23        f = A[:, :H]
24        g = A[:, H:2*H]
25        i = A[:, 2*H:3*H]
26        o = A[:, 3*H:]
27
28        f = sigmoid(f)
29        g = np.tanh(g)
30        i = sigmoid(i)
31        o = sigmoid(o)
32
33        c_next = f * c_prev + g * i
34        h_next = o * np.tanh(c_next)
35
36        self.cache = (x, h_prev, c_prev, i, f, g, o, c_next)
37        return h_next, c_next

```

### 3. Seq2seq

#### 3.1 Encoder

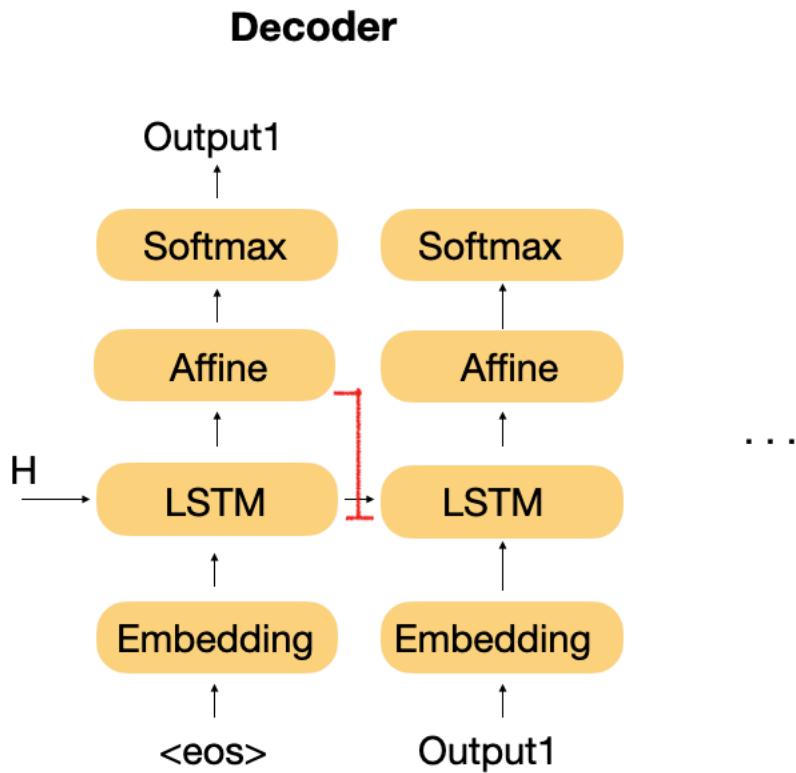
**Encoder** : Input sequence → Hidden state vector



- Embedding: 자연 언어 → 밀집dense 벡터

```
def forward(self, xs):  
    xs = self.embed.forward(xs)  
    hs = self.lstm.forward(xs)  
    self.hs = hs[-1] # 마지막 시각의 은닉 상태만을 추출  
    return hs[:,-1,:]
```

### 3.2 Decoder



```
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
```

### 3.3 seq2seq (main)

```
def forward(self, xs, ts):
    decoder_xs, decoder_ts = ts[:, :-1], ts[:, 1:]

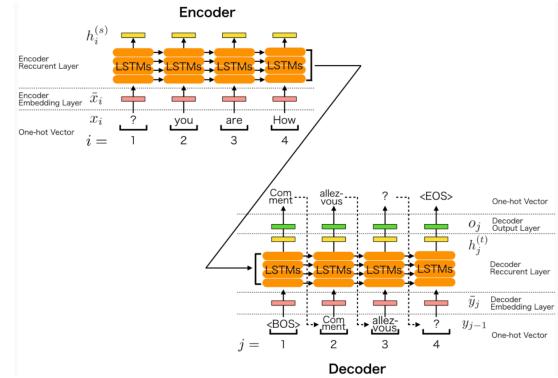
    h = self.encoder.forward(xs)
    score = self.decoder.forward(decoder_xs, h)
    loss = self.softmax.forward(score, decoder_ts)
    return loss
```

### 3.4 The moral of the Original paper<sup>3</sup>

While the LSTM is capable of solving problems with long term dependencies, we discovered that the LSTM learns **much better when the source sentences are reversed** (the target sentences are not reversed). By doing so, the LSTM's test perplexity dropped from 5.8 to 4.7, and the test BLEU scores of its decoded translations increased from 25.9 to 30.6.

While we do not have a complete explanation to this phenomenon, we believe that it is caused by

번역하고자 하는 문장 (source sentences)의 입력을 **거꾸로** 하면, 복잡도perplexity는 떨어지고, BLEU( 번역 기 평가 지표)는 상승한다.



Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." *Advances in neural information processing systems*. 2014.