15기 정규세션

ToBig's 14기 고경태

모델 심화
Recurrent Neural Network

References

먼저, 다음 오픈소스를 참고하여 강의자료를 만들었음을 밝힙니다.
설명이 부족한 부분은 아래의 링크에서 자세;히 공부하면 좋을 것 같습니다.

Idea Factory KAIST – 딥러닝 홀로서기 #26, #28

Idea Factory KAIST – YouTube

Contents

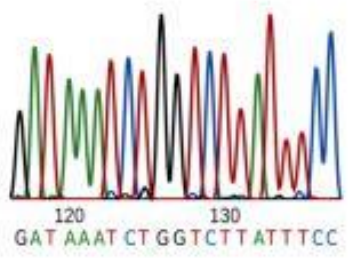
15기 정규세션

ToBig's 14기 고경태

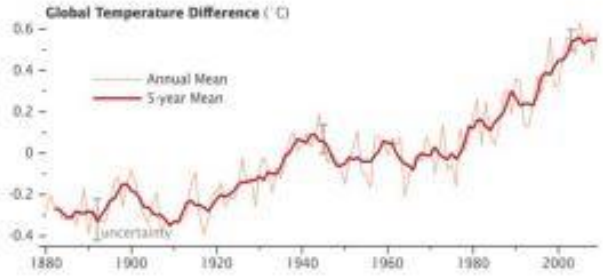
Sequential data

순차 데이터(Sequential data)

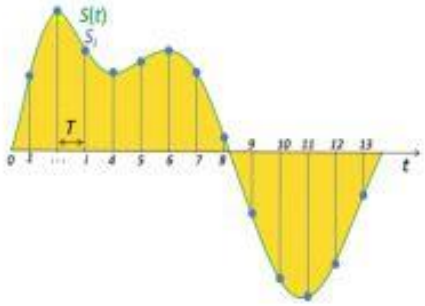
순서가 의미가 있으며, 순서가 달라질 경우 의미가 손상되는 데이터



DNA 염기 서열



기온 변화 혹은 주가

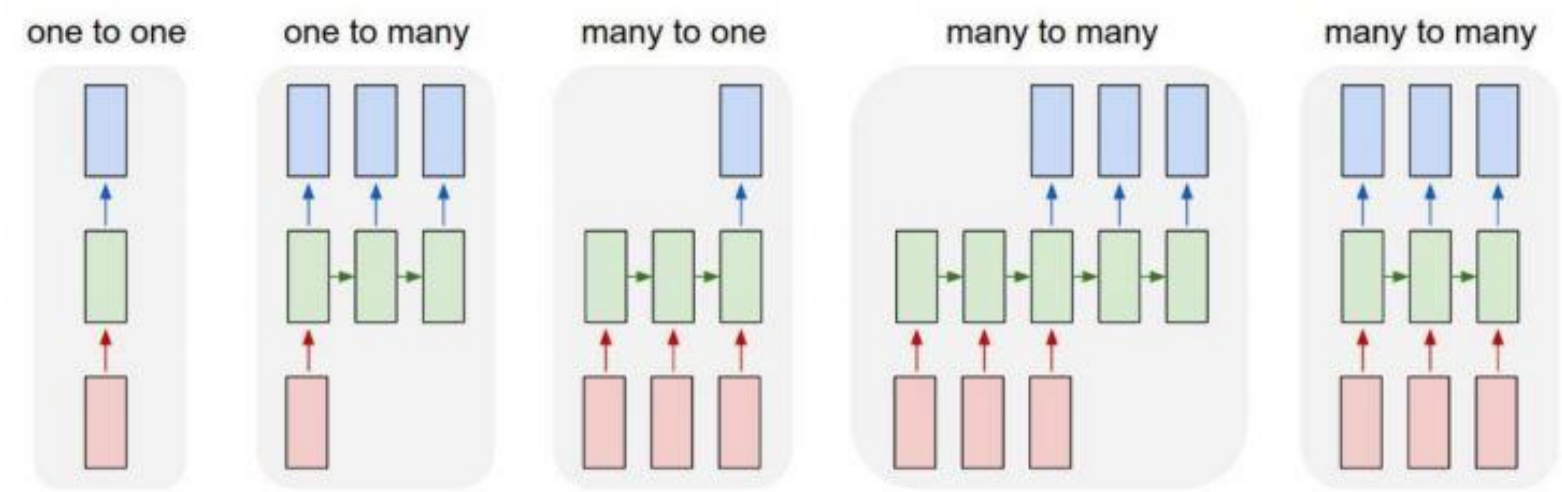


신호

자연어

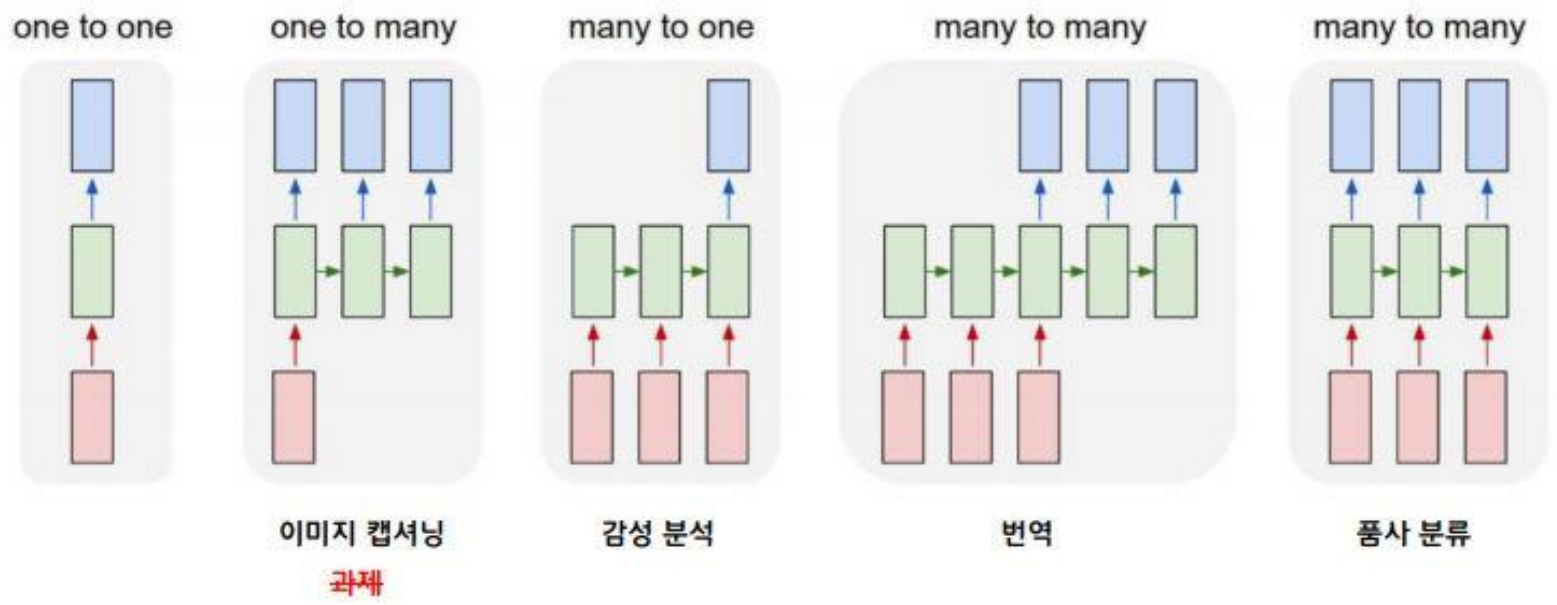
순차 데이터(Sequntial data)

다양한 태스크를 모델링할 수 있습니다.



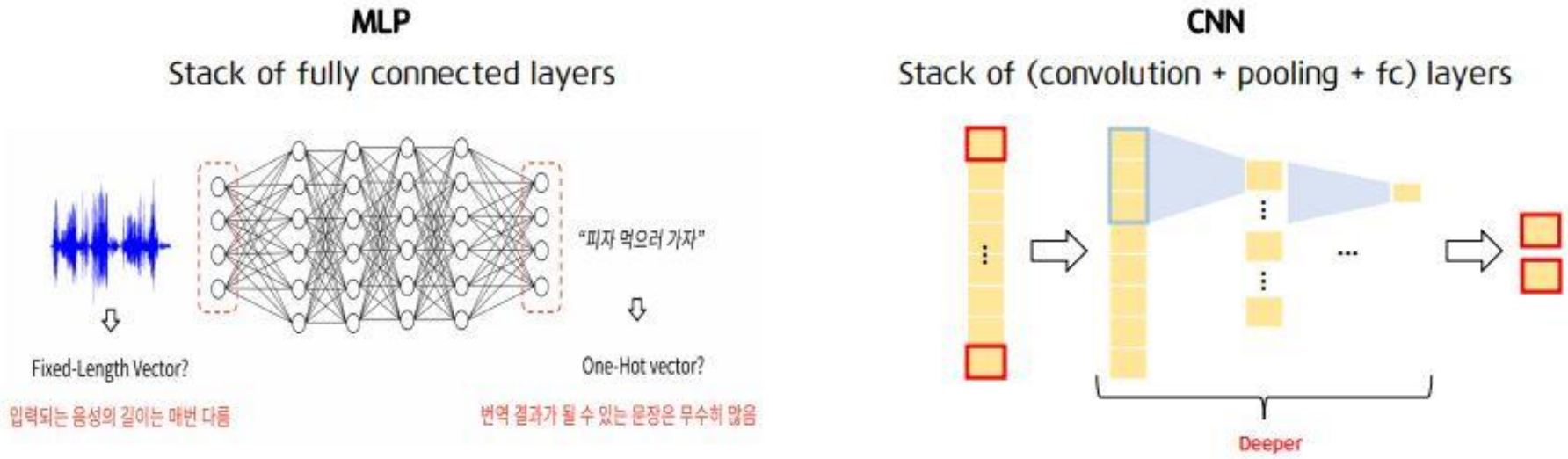
순차 데이터(Sequntial data)

다양한 태스크를 모델링할 수 있습니다.



순차 데이터(Sequntial data)

지금까지 배운 구조를 적용한다면...

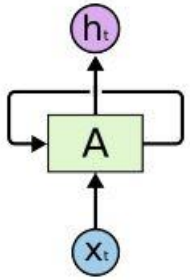


Sequential data에 알맞은 구조가 있을까요?

15기 정규세션

ToBig's 14기 고경태

Vanilla RNN



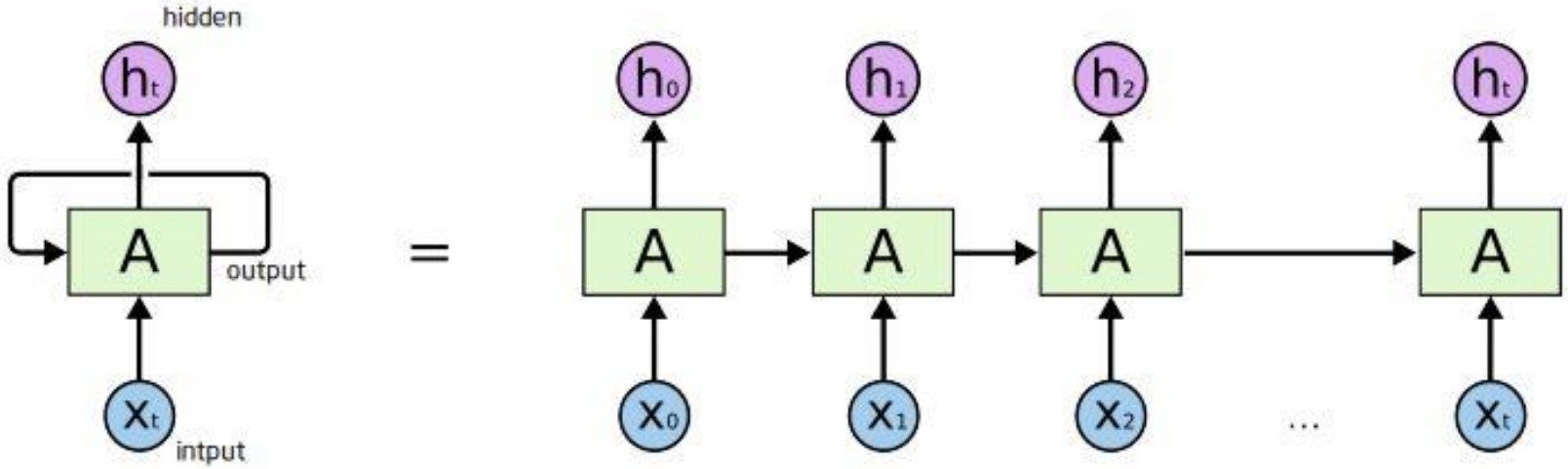
Unit 02 | **Vanilla RNN**

Recurrent Neural Network(RNN)

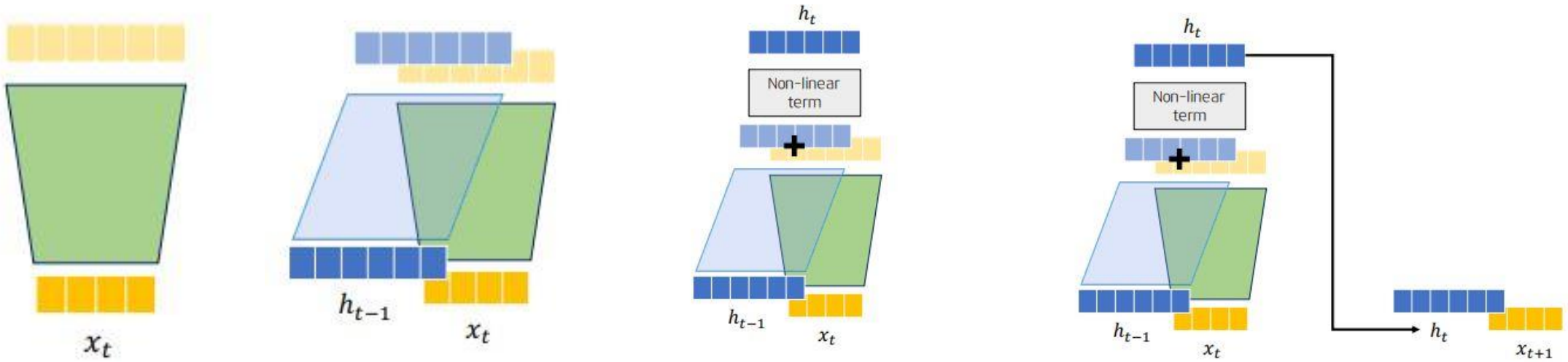
순환 구조를 이루고 있는 인공 신경망

RNN의 핵심!

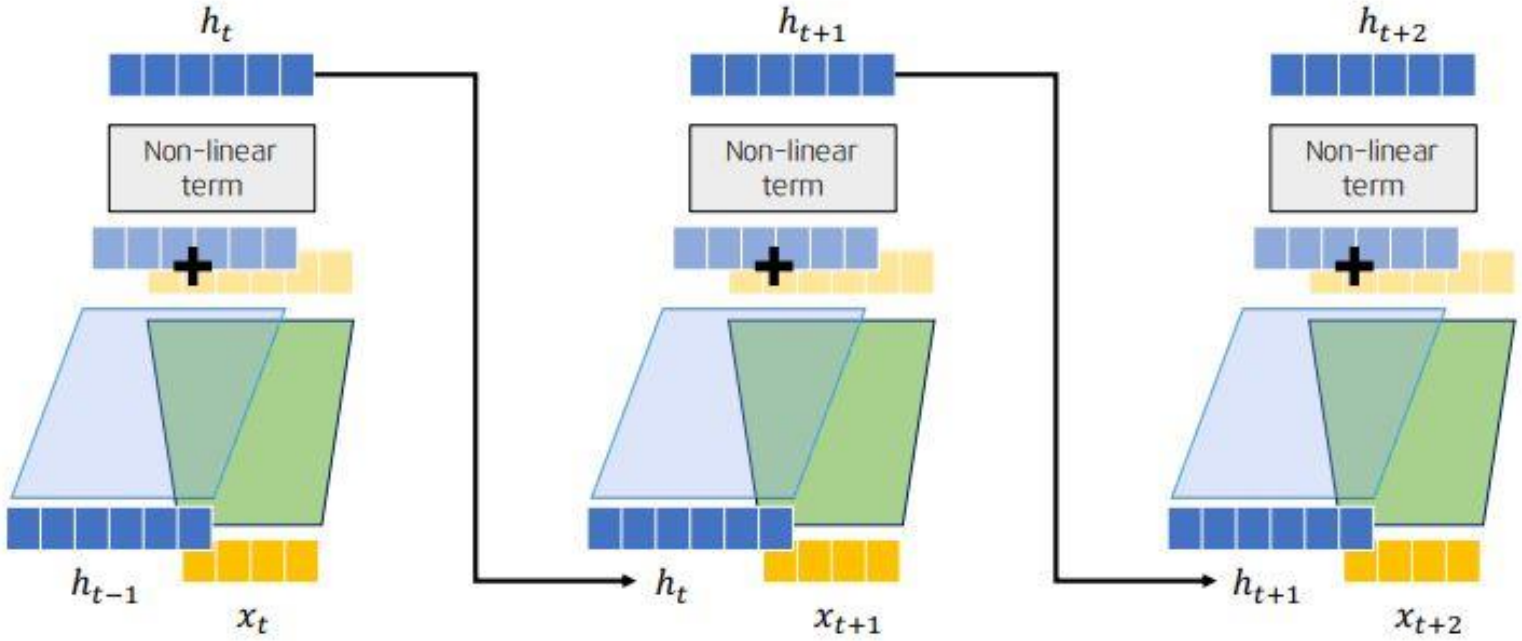
Hidden feature를 만들기 위해 현재 state의 input과 이전 state의 output을 사용합니다.



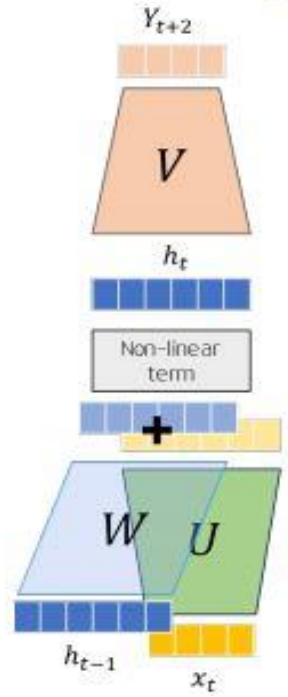
Recurrent Neural Network(RNN) 내부를 뜯어보자!



Recurrent Neural Network(RNN) 내부를 뜯어보자!



Recurrent Neural Network(RNN) 수식을 뜯어보자!



forward

$$f: tanh$$

$$h_t = f(Ux_t + Wh_{t-1})$$
$$y_t = f(Vh_t)$$

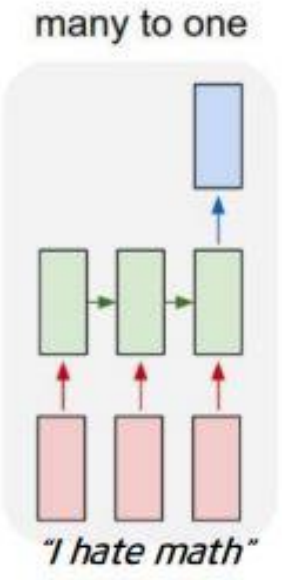
loss

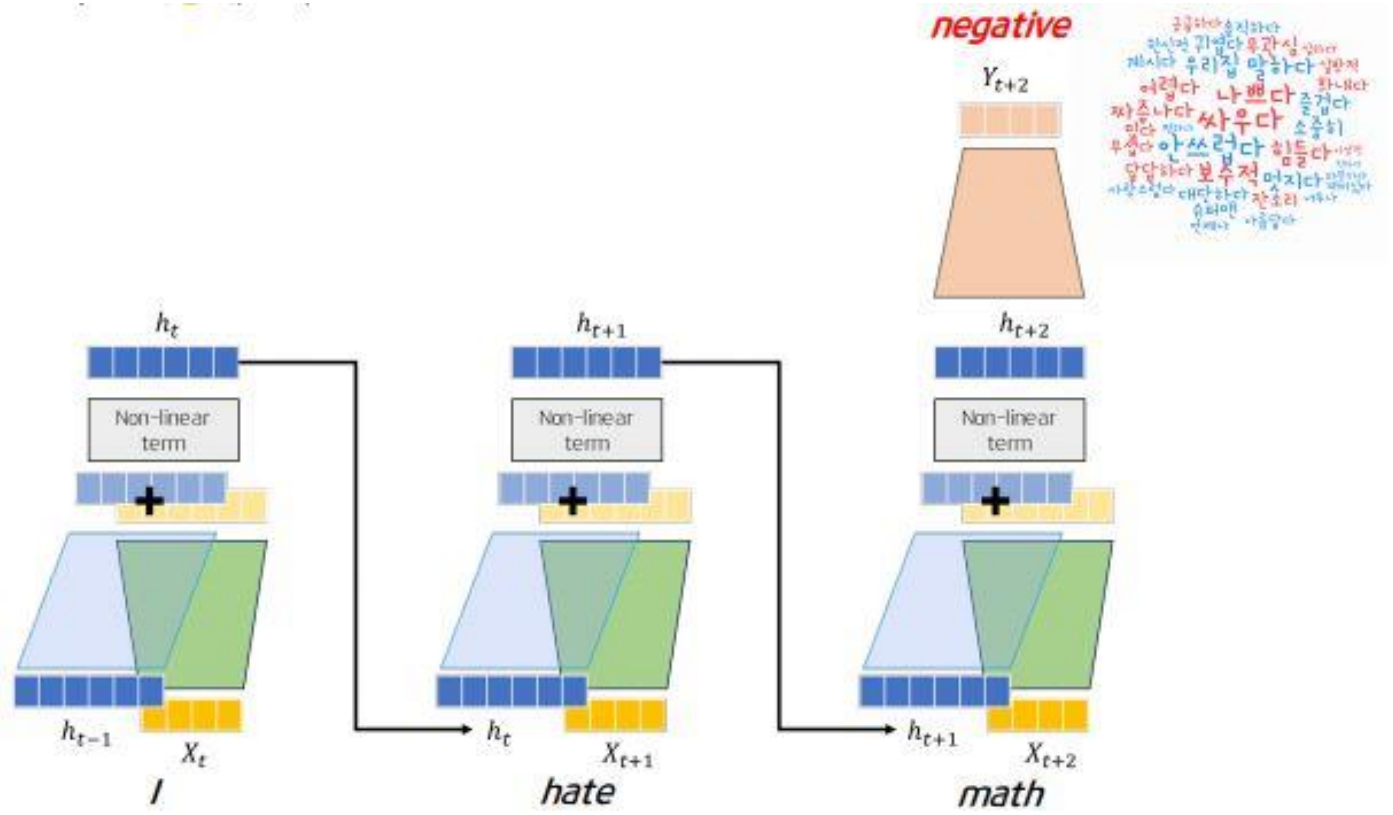
$$Loss(\theta) = \sum_t loss(y_{true,t}, y_{pred,t})$$

$$Classification \rightarrow CrossEntropy$$

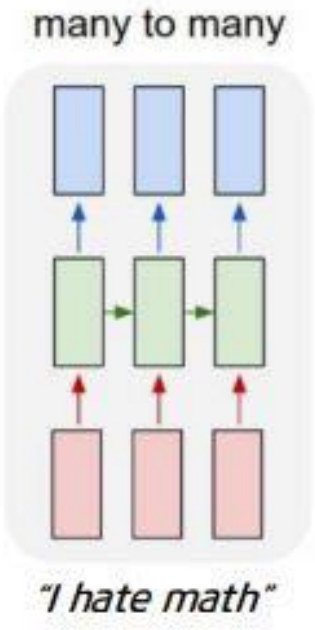
$$Regression \rightarrow MSE$$

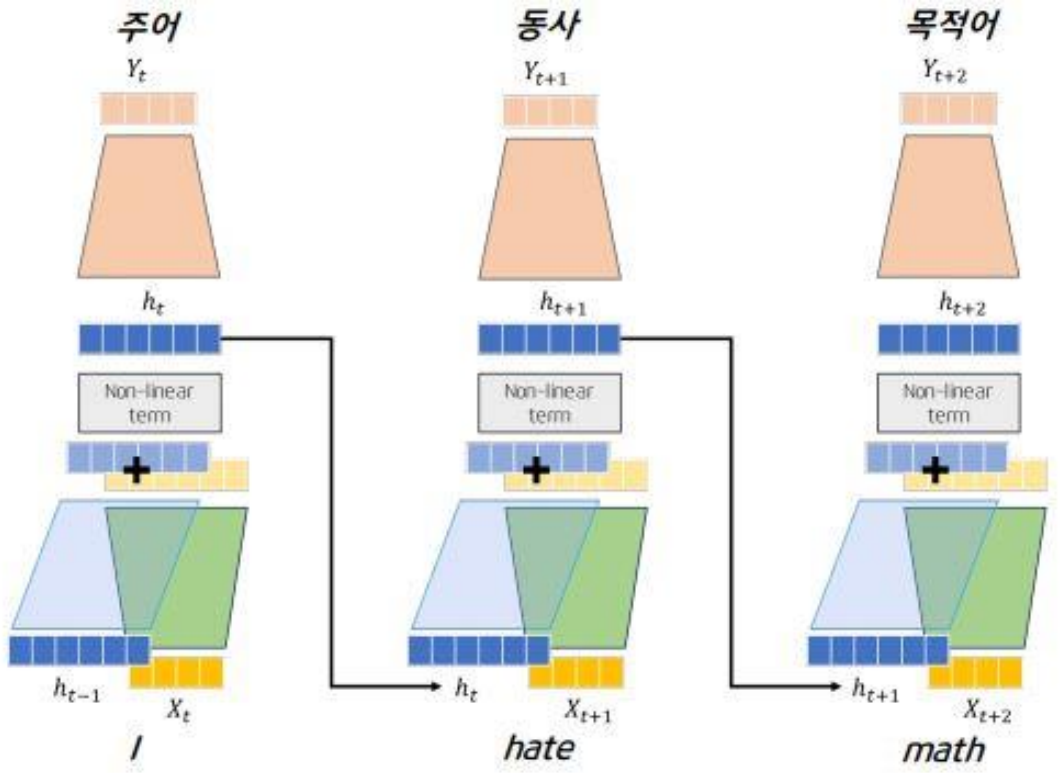
Recurrent Neural Network(RNN) 적용해보자!



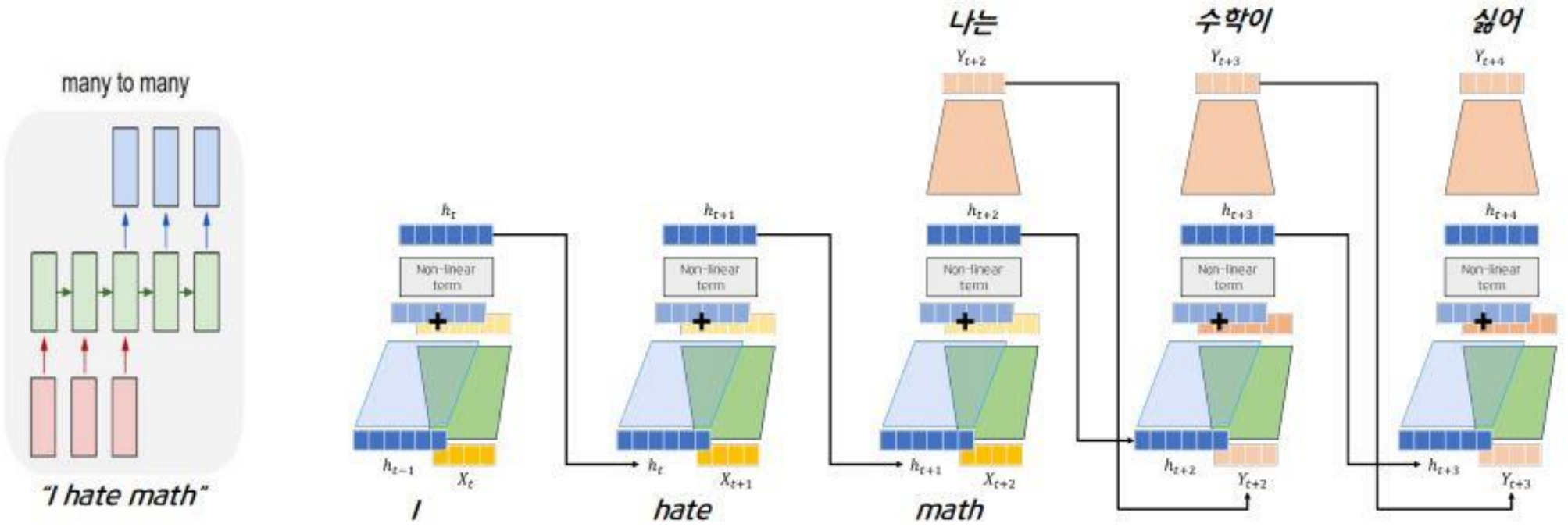


Recurrent Neural Network(RNN) 적용해보자!

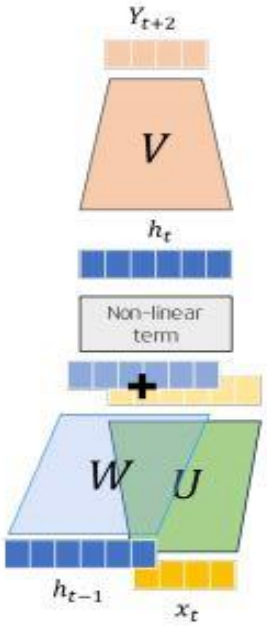




Recurrent Neural Network(RNN) 적용해보자!



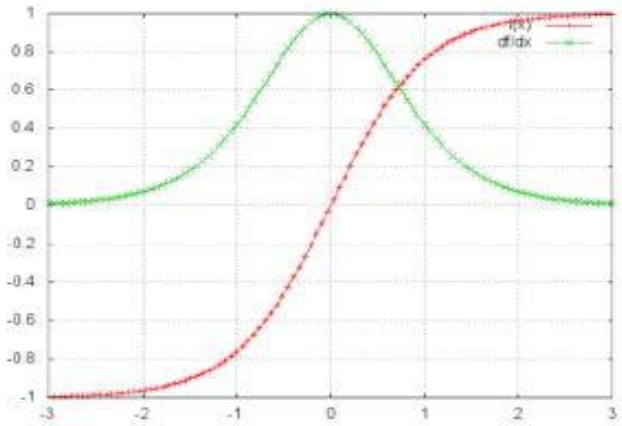
Recurrent Neural Network(RNN) Vanishing gradient



$$h_{t-2} = tanh(W[h_{t-3}, x_{t-2}])$$
$$h_{t-1} = tanh(W[h_{t-2}, x_{t-1}])$$
$$h_t = tanh(W[h_{t-1}, x_t])$$

$$h_t = tanh(W[\tanh(..\tanh(h_{t-3})), x_t])$$

tanh가 너무 많다 !



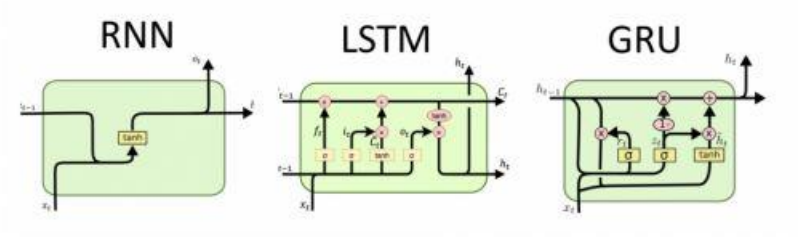
Graph of $tanh(x)$ (red) $\frac{d}{dx}tanh(x)$ (green)

Vanilla RNN로 long sequence를 학습하는 것은 힘듭니다 :(

15기 정규세션

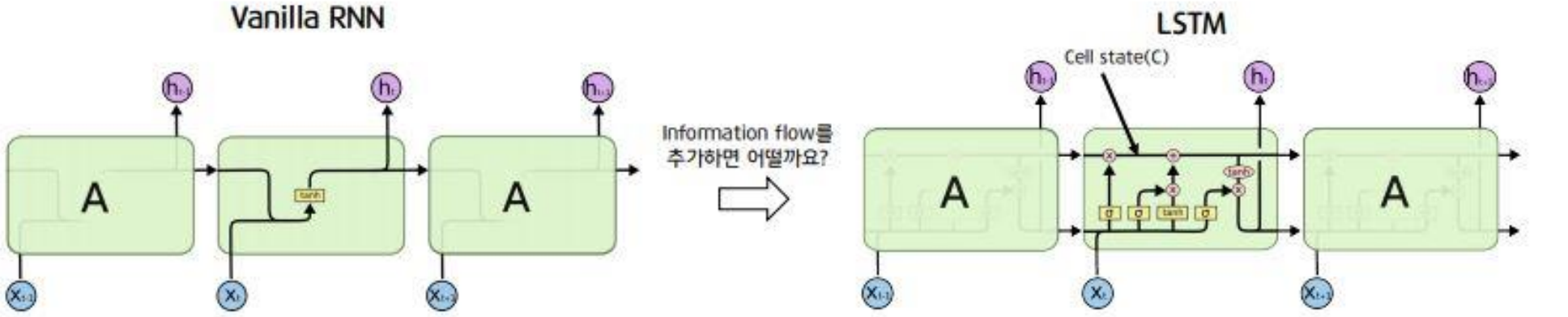
ToBig's 14기 고경태

LSTM and GRU



Long Short Term Memory Network

Vanilla RNN vs LSTM



h_t: 이전 step의 정보 저장 + output 역할을 하자

h_t 의 역할이 너무 많습니다.

$C(cell\ state)$: 남길 건 남기고, 잊어버릴 건 잊어버리고,
새로 추가할 건 추가하자

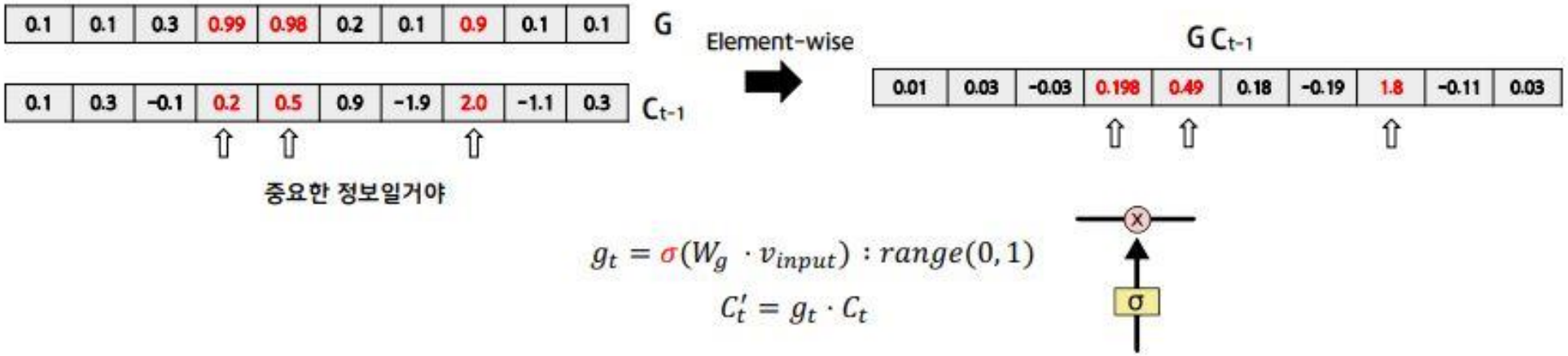
중요한 정보만 계속 흘러가도록!

h_t : C 를 적당히 가공해서 내보내자

Gate

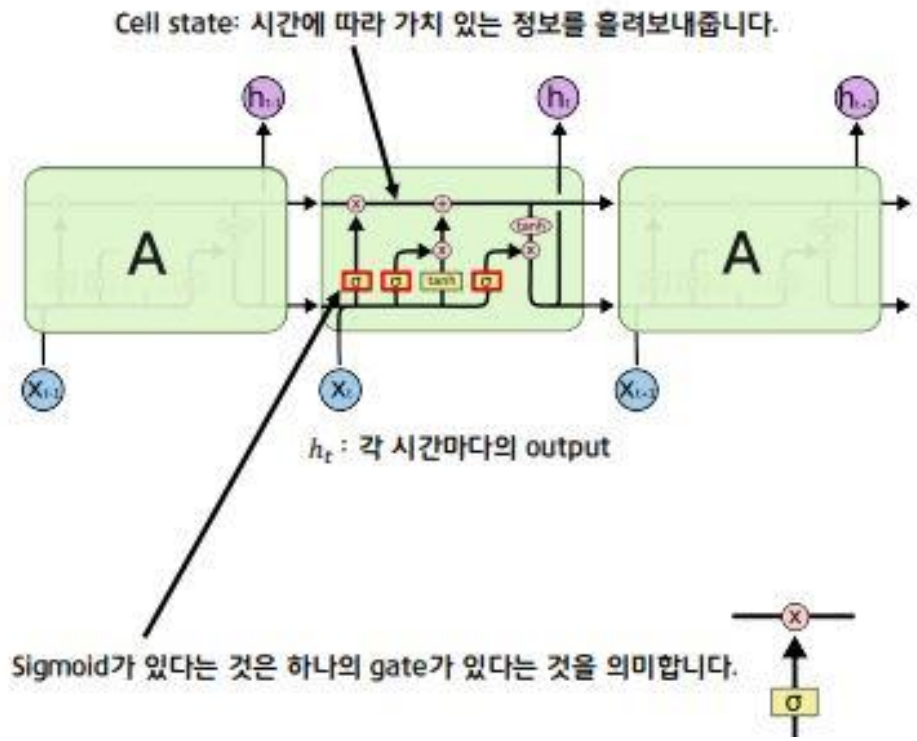
Element wise로 계수(coefficient)를 곱해주는 것

Control whether pass or block the information of each dimension with coefficient **0~1**



Long Short Term Memory Network

내부를 뜯어보자



남길 건 남기고, 잊어버릴 건 잊어버리고,
새로 추가할 건 추가해서
cell state에 중요한 정보만 계속 흘러가도록!

남길 건 남기고, 잊어버릴 건 잊어버리고,
새로 추가할 건 추가해서
cell state에 중요한 정보만 계속 흘러가도록!

남길 건 남기고, 잊어버릴 건 잊어버리고,
새로 추가할 건 추가해서
cell state에 중요한 정보만 계속 흘러가도록!

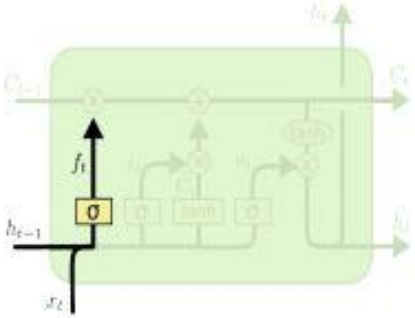
⇩

Gate!

Long Short Term Memory Network

수식을 뜯어보자

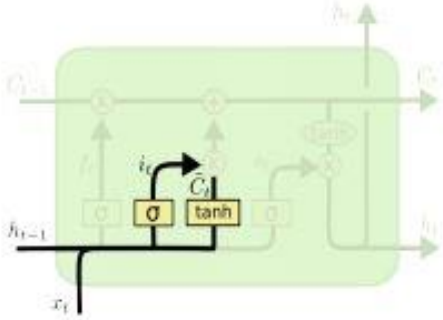
① Forget gate를 만든다.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

C_{t-1} 에서 불필요한 정보를 지웁니다.

② Input gate를 만든다.



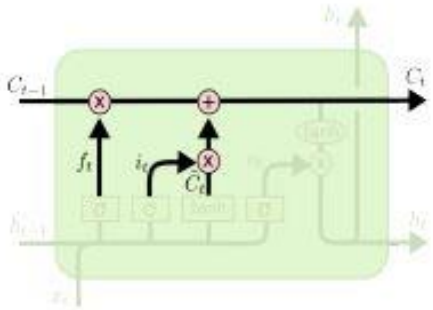
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

C_{t-1}에 새로운 input x_t와 h_{t-1}를 보고 중요한 정보를 넣습니다.

Long Short Term Memory Network

수식을 뜯어보자

③ Cell state를 업데이트 한다.



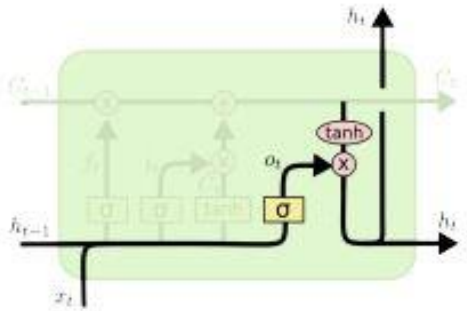
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(과거 / 현재)

f_t 를 이용해서 C_{t-1} 의 일부 정보를 날리고,

i_t 를 이용해서 $\tilde{C}_t$ 정보를 추가합니다.

④ output gate를 만든다.



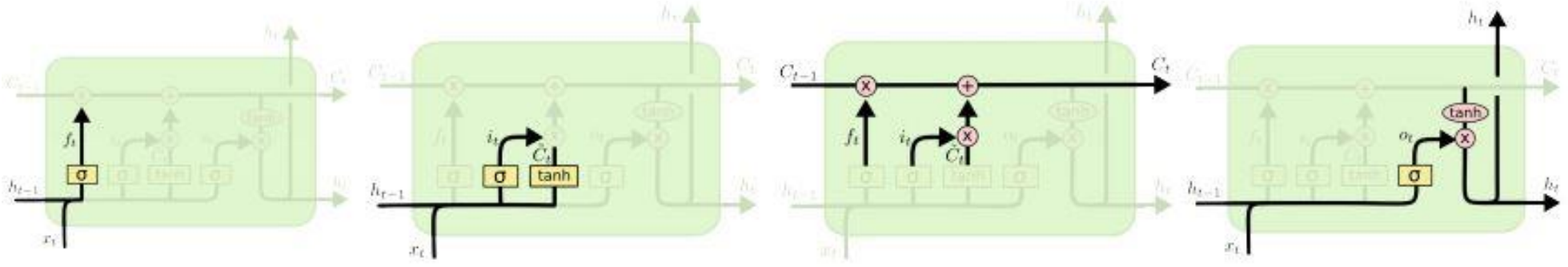
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

C_t를 적절히 가공해 h_t를 만듭니다.

가공은 output gate o_t로 합니다.

Long Short Term Memory Network 정리

남길 건 남기고, 잊어버릴 건 잊어버리고, 새로 추가할 건 추가해서 cell state에 중요한 정보만 계속 흘러가도록!



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

남길 건 남기고
잊어버릴 건 잊어버리고

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

새로 추가할 건 추가해서

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

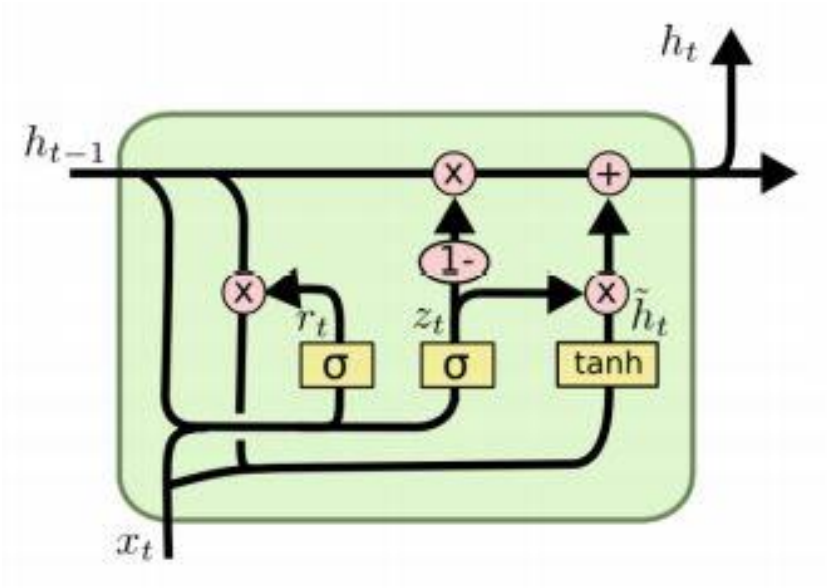
cell state에 중요한 정보만
계속 흘러가도록

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

C 를 적당히 가공해서 내보내자

Gate Recurrent Unit LSTM VS GRU

LSTM의 간소화



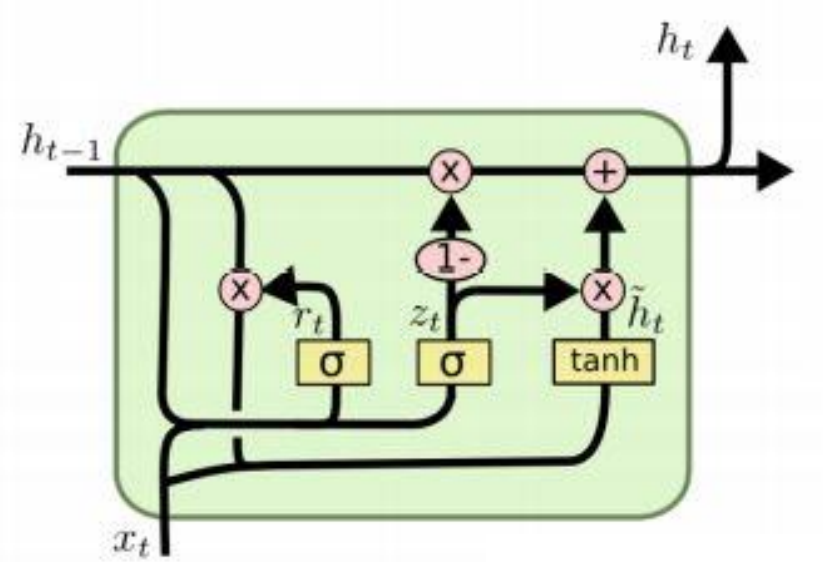
탄생 배경:
LSTM은 gate 수가 너무 많습니다.
이것을 줄일 수 없을까요?

GRU의 특징:

- cell state가 없고, hidden state만 존재함.
- Forget + input gate를 결합시킴.
- Reset gate를 추가함.

여전히 vanishing gradient를 해결할 수 있을까요?

Gate Recurrent Unit 수식을 뜯어보자



$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

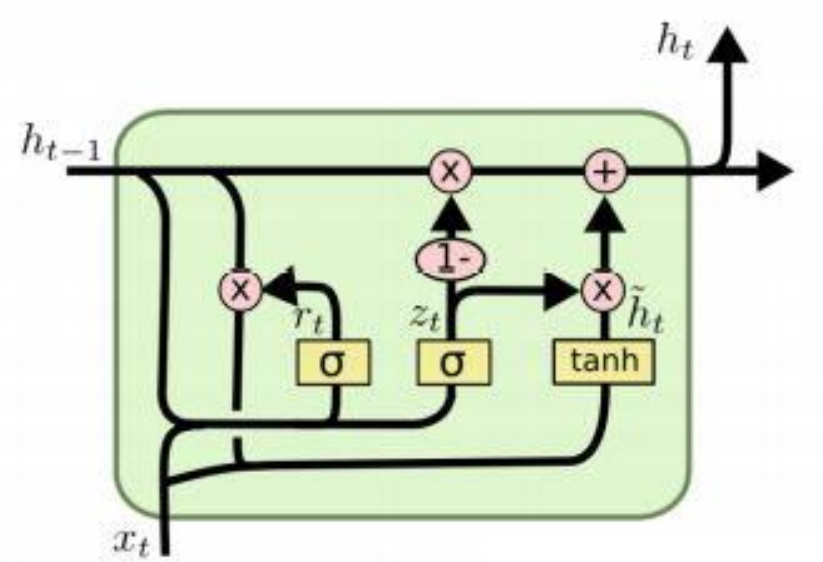
$$\tilde{h}_t = tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

1. Reset gate를 계산해서 임시 h_t를 만든다.
2. Update gate를 통해 h_{t-1}과 $\tilde{h}_t$ 간의 비중을 결정한다.
3. z_t를 이용해 최종 h_t를 계산한다.

Gate Recurrent Unit 수식을 뜯어보자



$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

0에 가까운 값이 되면 'reset'이 되어, 새로운 input이 시작이 될 수 있다.

$$\tilde{h}_t = tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

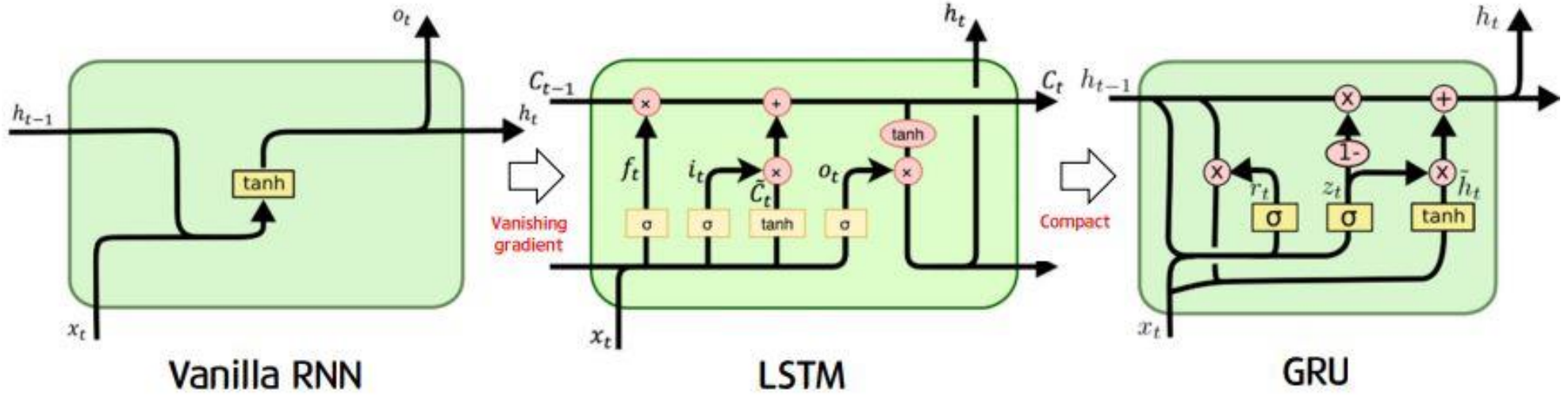
과거와 현재 정보의 비중을 결정

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

past current

1. Reset gate를 계산해서 임시 h_t를 만든다.
2. Update gate를 통해 h_{t-1}과 $\tilde{h}_t$ 간의 비중을 결정한다.
3. z_t를 이용해 최종 h_t를 계산한다.

Summary



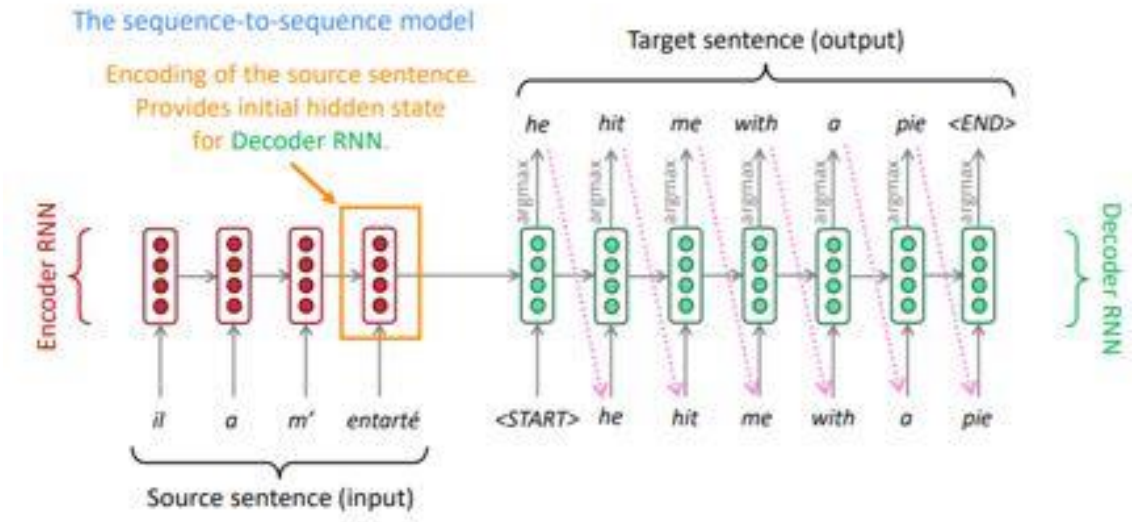
LSTM과 GRU는 Vanishing gradient를 어떻게 해결하려고 했을까요?

15기 정규세션

ToBig's 14기 고경태

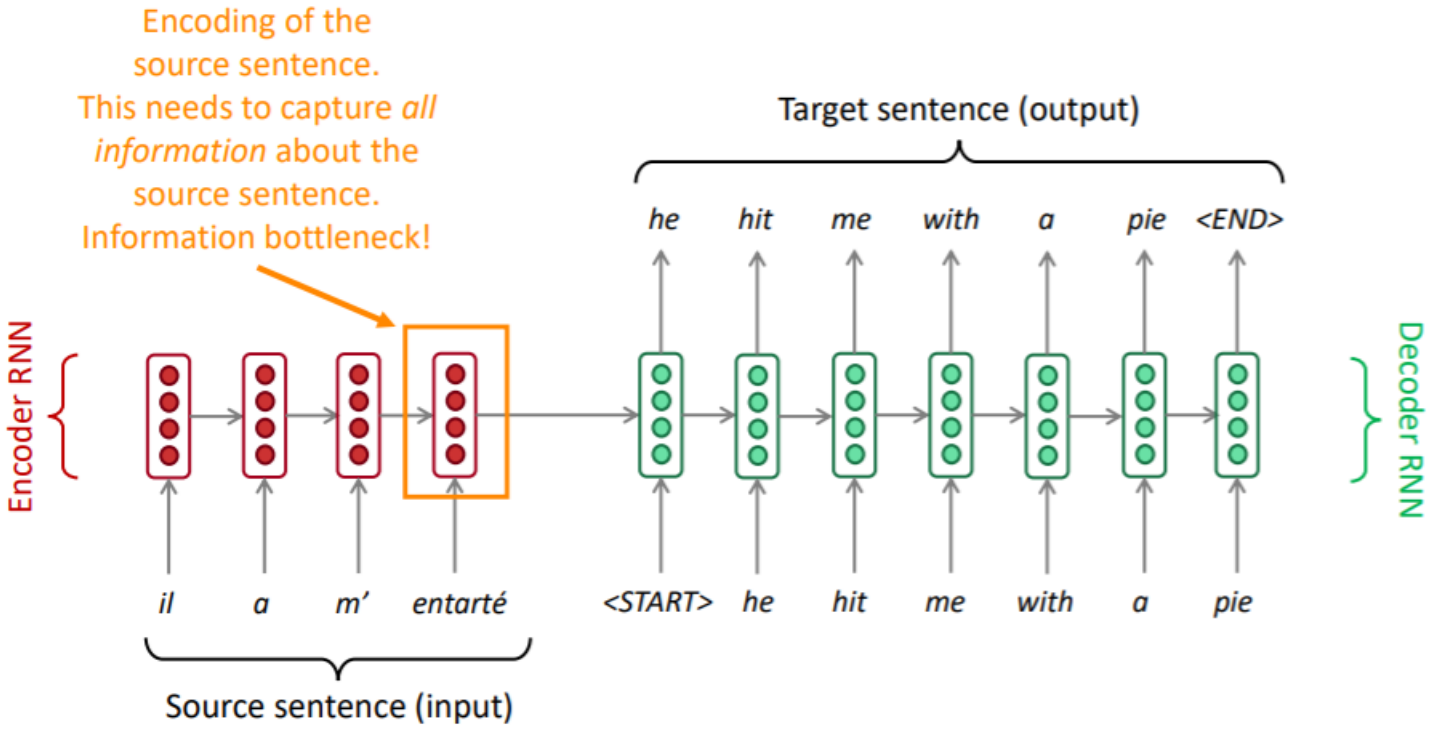
Attention

Sequence-to-sequence



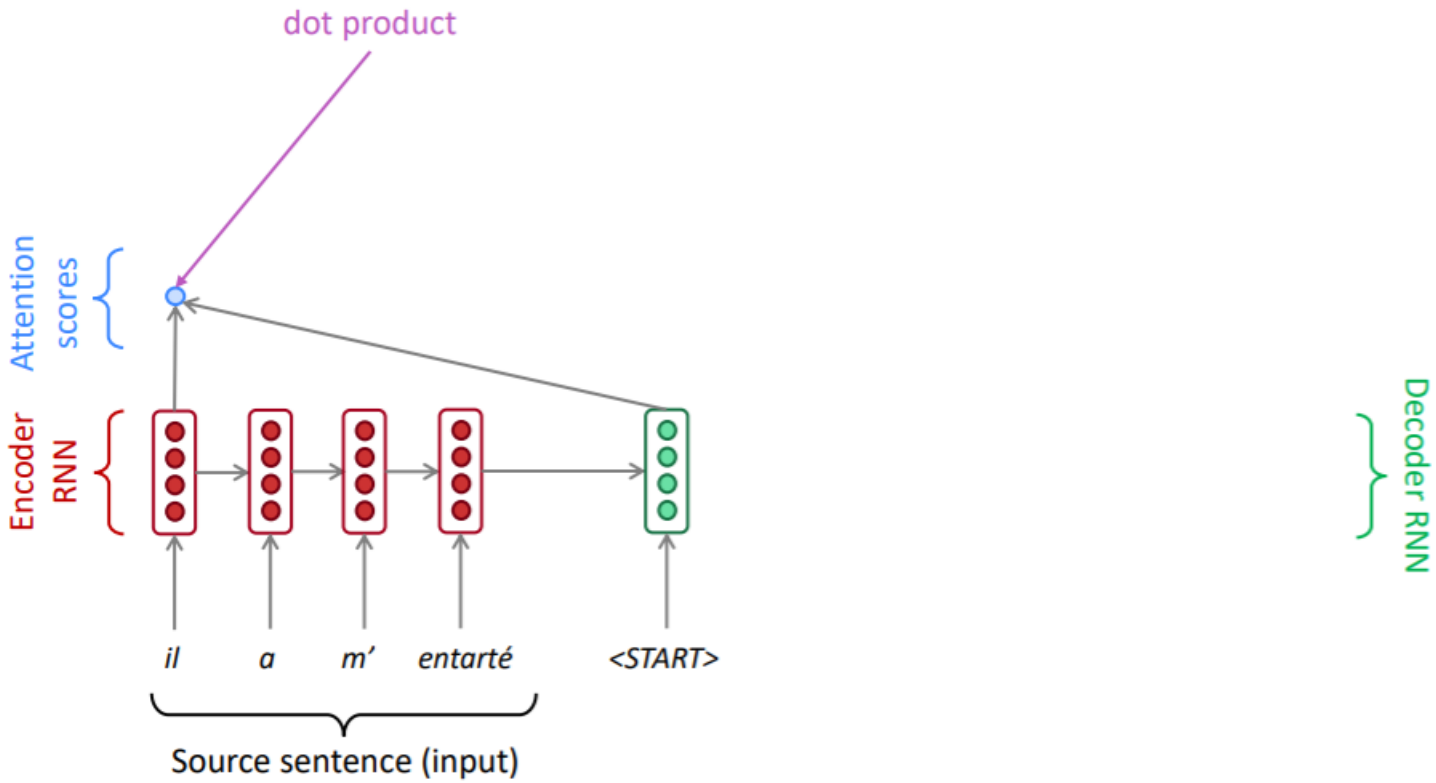
두개의 RNN을 포함한 seq-to-seq

Sequence-to-sequence : the bottleneck problem

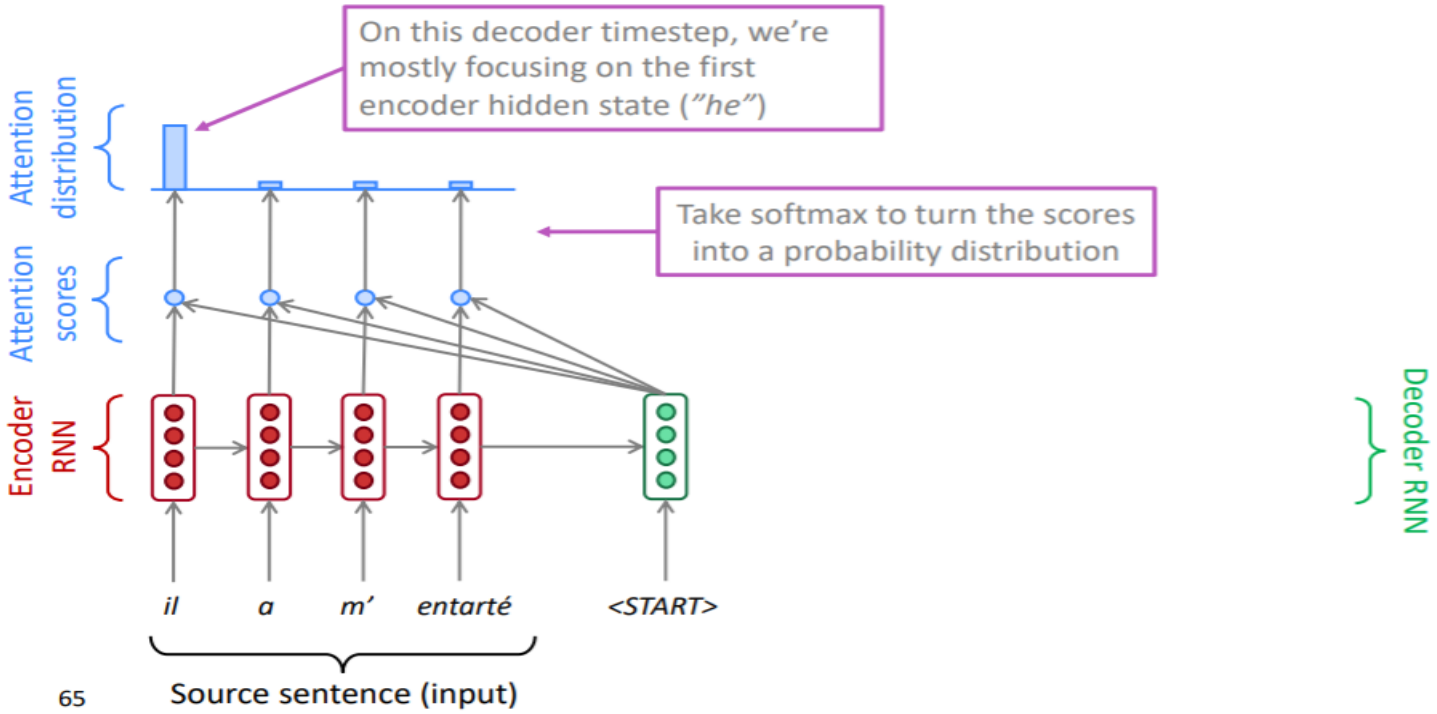


맨 끝에서 모든 정보를 캡쳐 강요
→ 너무 많은 압력 → 병목문제

Sequence-to-sequence with attention

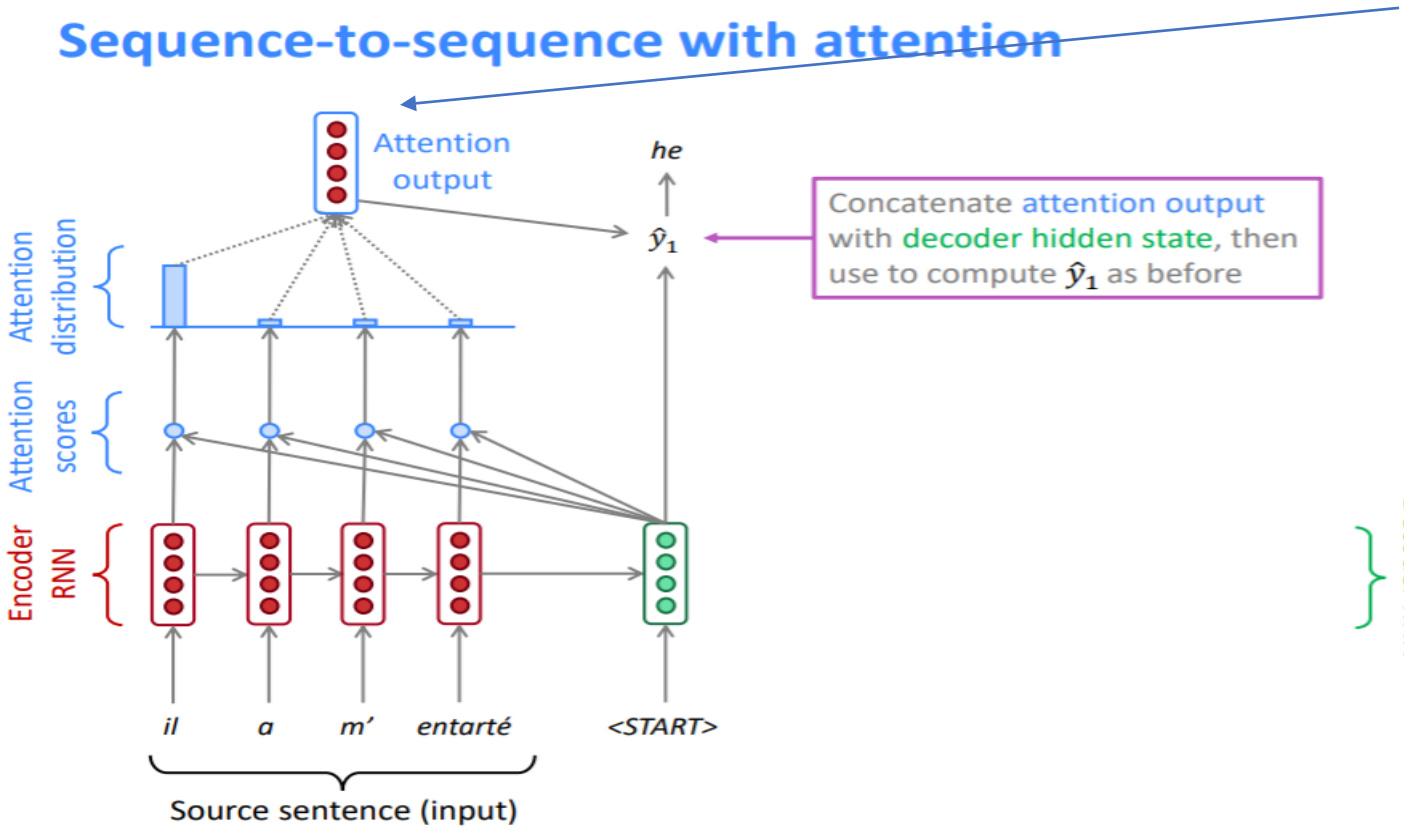


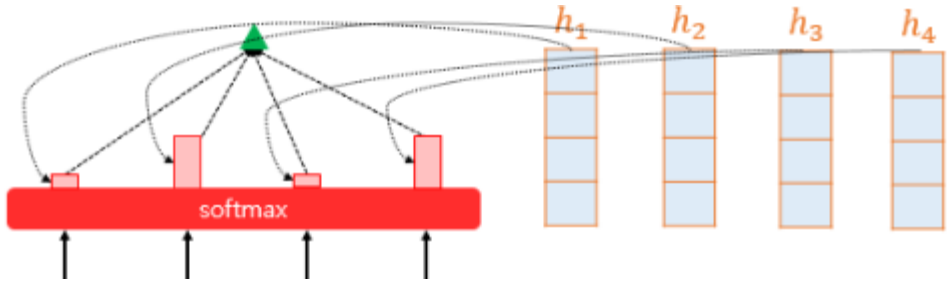
Sequence-to-sequence with attention



모든 인코더의 step마다 반복!
Attention을 주는 것

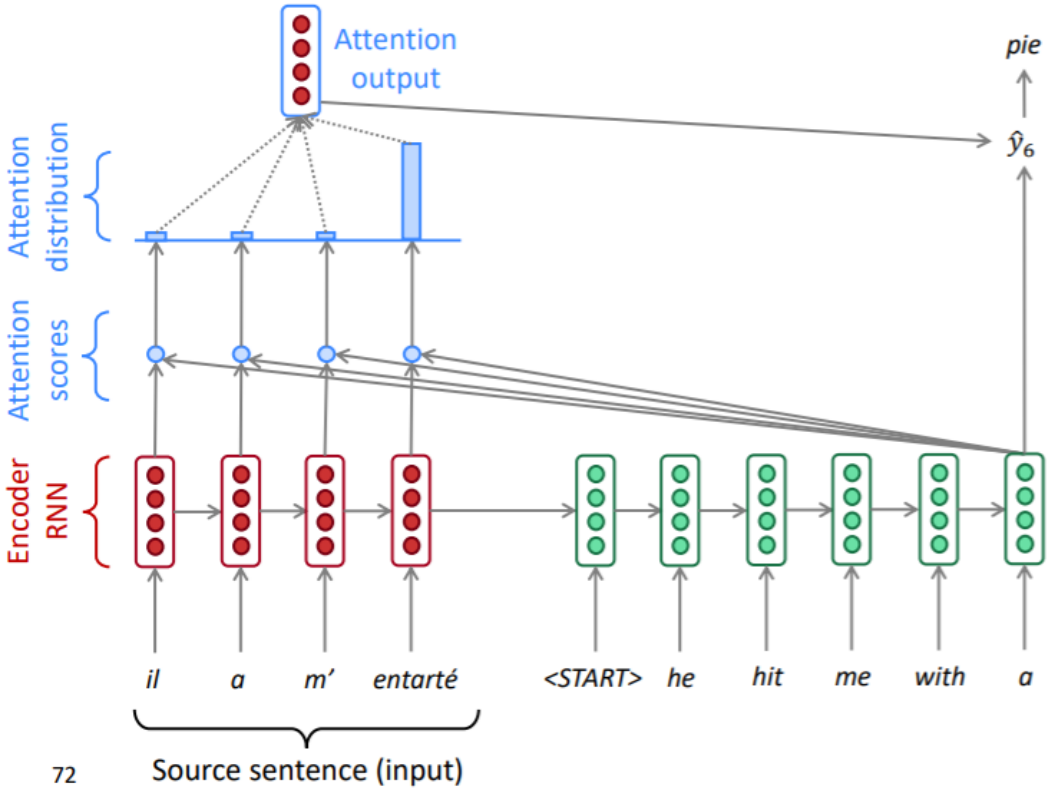
Sequence-to-sequence with attention





Attention의 output과 decode의 hidden state의 결합 → y1(hat)을 계산

Sequence-to-sequence with attention

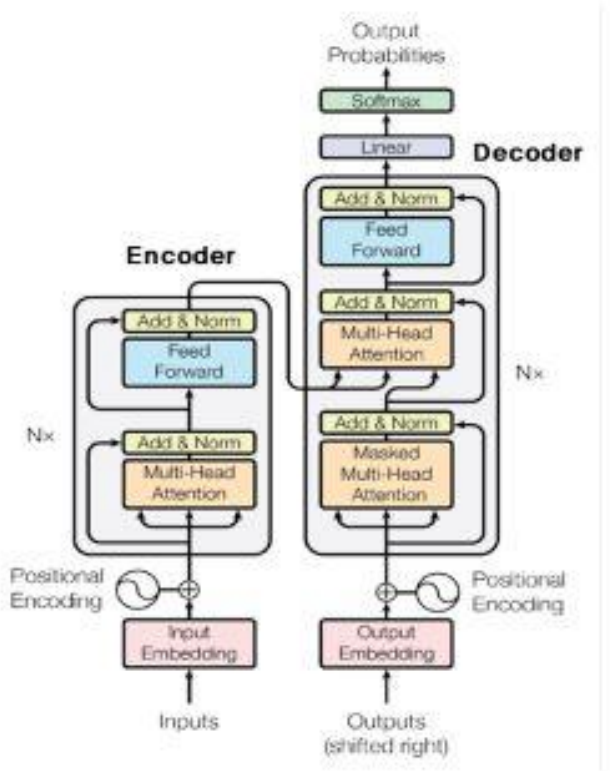


decoder에서도 같은 행동을 반복!

Unit 05 | Transformer

Pretrained model

Transformer



Keyword

- ✓ Masked attention
- ✓ Multi head attention
- ✓ Self attention
- ✓ Positional encoding

과제 소개

과제. ResNet / Transformer 논문 리뷰 (택1)

둘 중 하나를 선택해서 리뷰해주세요! (hwp, word, markdown 등 형식은 자유입니다.)
제시한 키워드를 바탕으로 작동원리를 설명하고, 사용하는 이유를 중심으로 설명주세요.
다른 리뷰 글을 참고하셔도 괜찮지만, 그대로 가져왔다고 생각되는 경우 과제 반려합니다.
(참고하신 글은 꼭 reference로 달아주세요~)

과제 소개

논문 리뷰 Keyword

〈 ResNet 〉

(Deep residual learning)

- ✓Degradation Problem
- ✓Residual Learning
- ✓Skip Connection
- ✓Identity mapping

〈 Transformer 〉

(Attention is all you need)

- ✓Masked attention
- ✓Self attention
- ✓Multi head attention
- ✓Positional encoding

들어주셔서 감사합니다.