

RNN 이론 및 실습

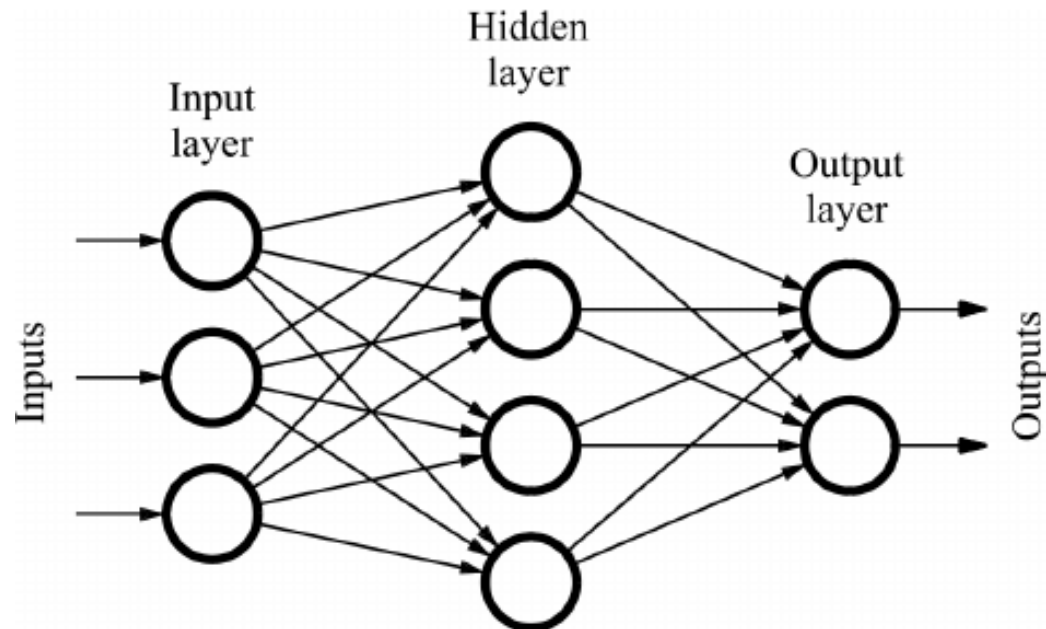
주재걸 교수님 연구실
DAVIAN Lab.

강경필

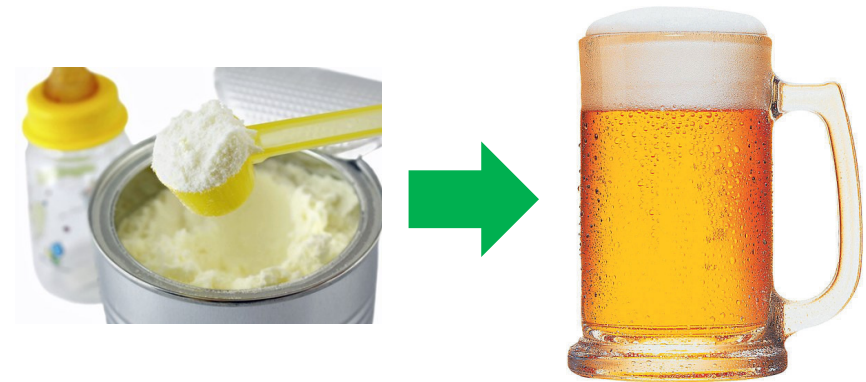
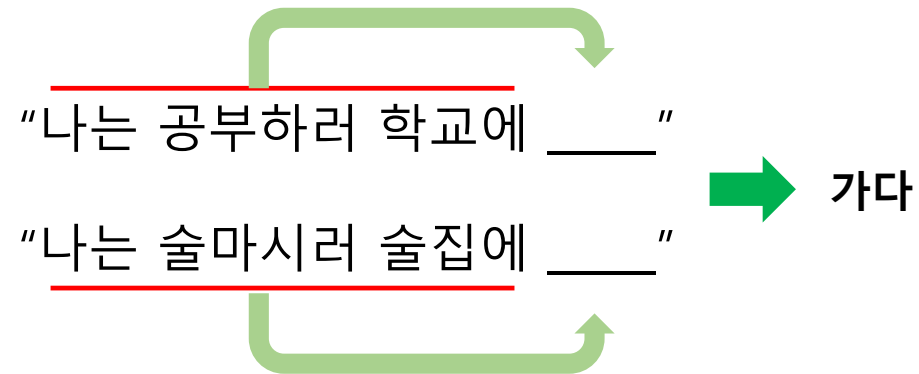
Feed Forward Neural Networks

FFN은 각 피쳐들의 조합을 통해 추론함.

하지만 FFN은 시간적/순서적 정보를 고려할 수 없음!



Sequential Information



Sequential Information

"나는 학교에 간다" → "나" -> "는" -> "학교" -> "에" -> "간다"

Sequential Information

"나는 학교에 간다"



"나" -> "는" -> "학교" -> "에" -> "간다"

State 1 - "나": 대상

Sequential Information

"나는 학교에 간다"



"나" -> "는" -> "학교" -> "에" -> "간다"

State 1 - "나": 대상

State 2 - "나는": 대상의 상태

Sequential Information

"나는 학교에 간다"



"나" -> "는" -> "학교" -> "에" -> "간다"

State 1 - "나": 대상

State 2 - "나는": 대상의 상태

State 3 - "나는 학교": 대상의 상태와 객체

Sequential Information

"나는 학교에 간다"



"나" -> "는" -> "학교" -> "에" -> "간다"

State 1 - "나": 대상

State 2 - "나는": 대상의 상태

State 3 - "나는 학교": 대상의 상태와 객체

State 4 - "나는 학교에": 대상이 객체를 향해

Sequential Information

"나는 학교에 간다"



"나" -> "는" -> "학교" -> "에" -> "간다"

State 1 - "나": 대상

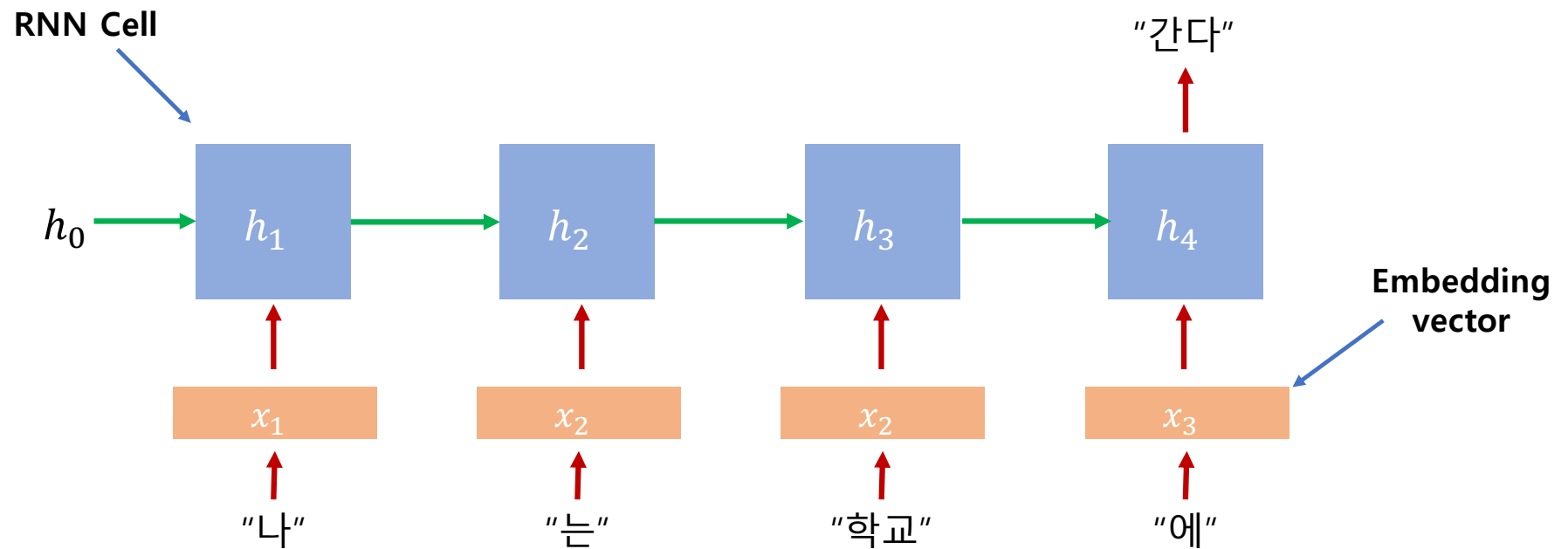
State 2 - "나는": 대상의 상태

State 3 - "나는 학교": 대상의 상태와 객체

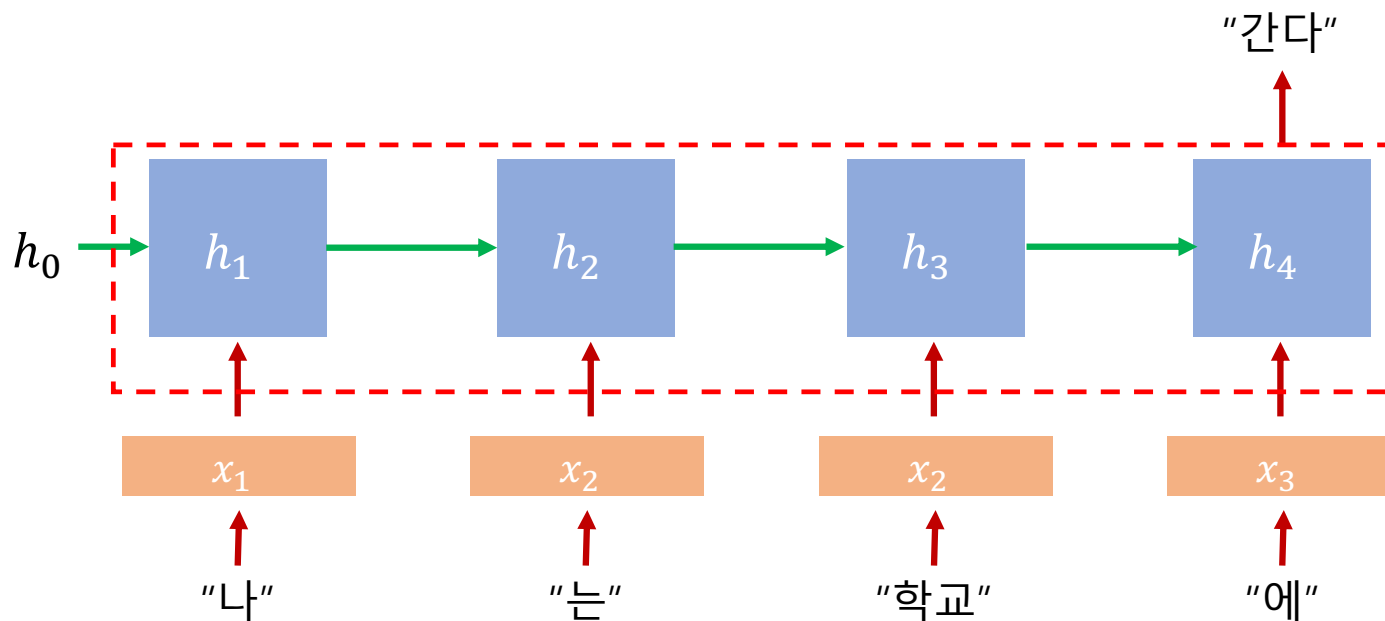
State 4 - "나는 학교에": 대상이 객체를 향해

~을 장소를 향해 가는 것이므로 그 다음 상태는
간다는 동사가 어울림!

Recurrent Neural Networks (RNN)

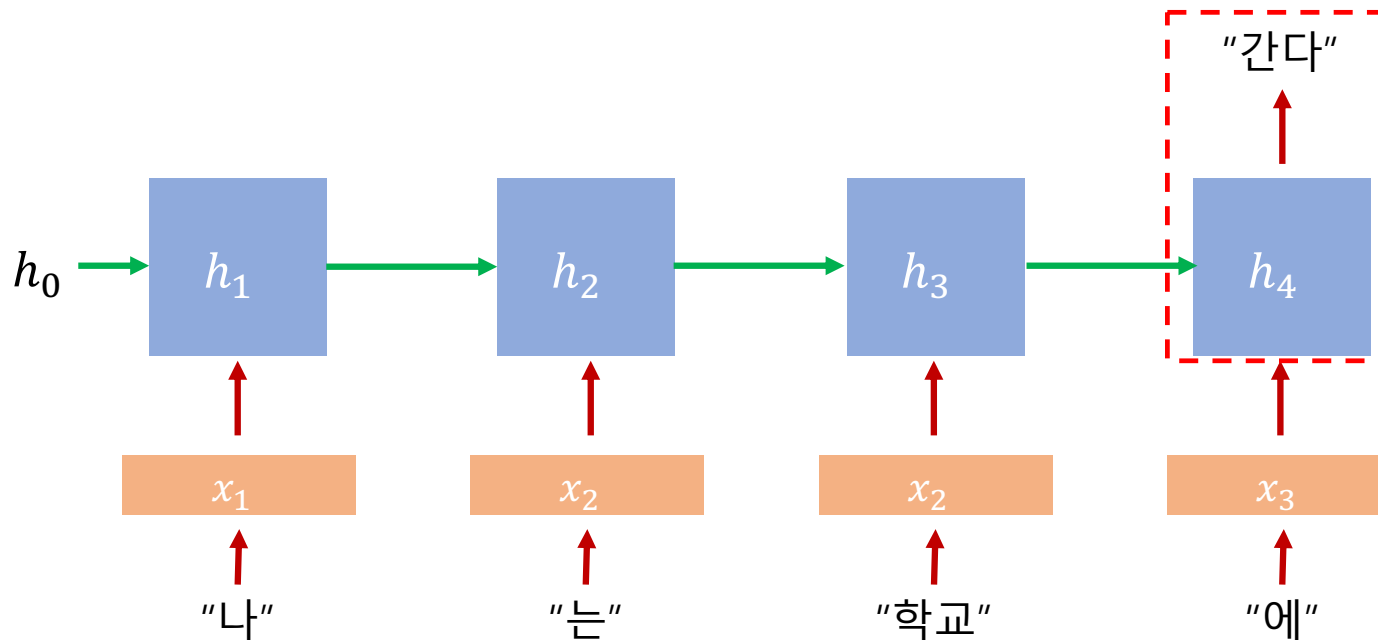


Recurrent Neural Networks (RNN)



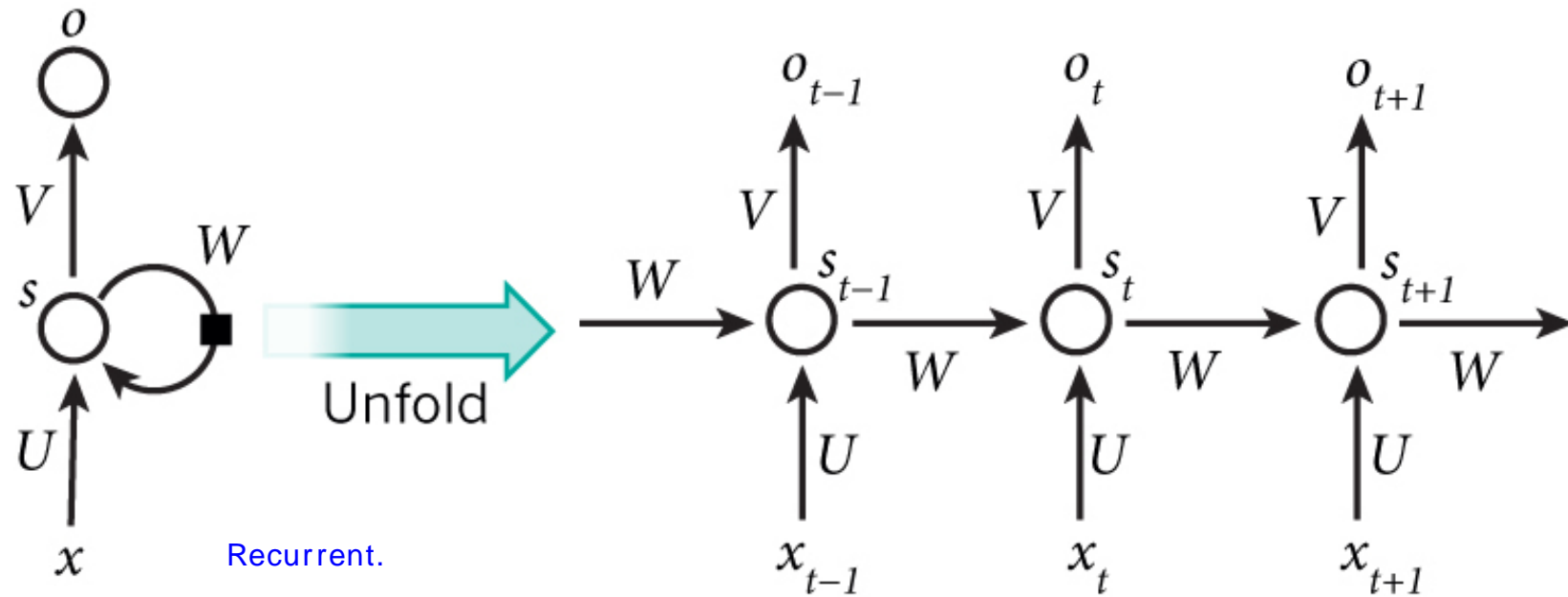
1. 각 인풋이 들어올 때마다 그 때의 상태(벡터)를 계산

Recurrent Neural Networks (RNN)



1. 각 인풋이 들어올 때마다 그 때의 상태(벡터)를 계산
2. 예측할 순간의 상태를 이용하여 예측

Recurrent Neural Networks (RNN)



Recurrent.

RNN

U, W, V

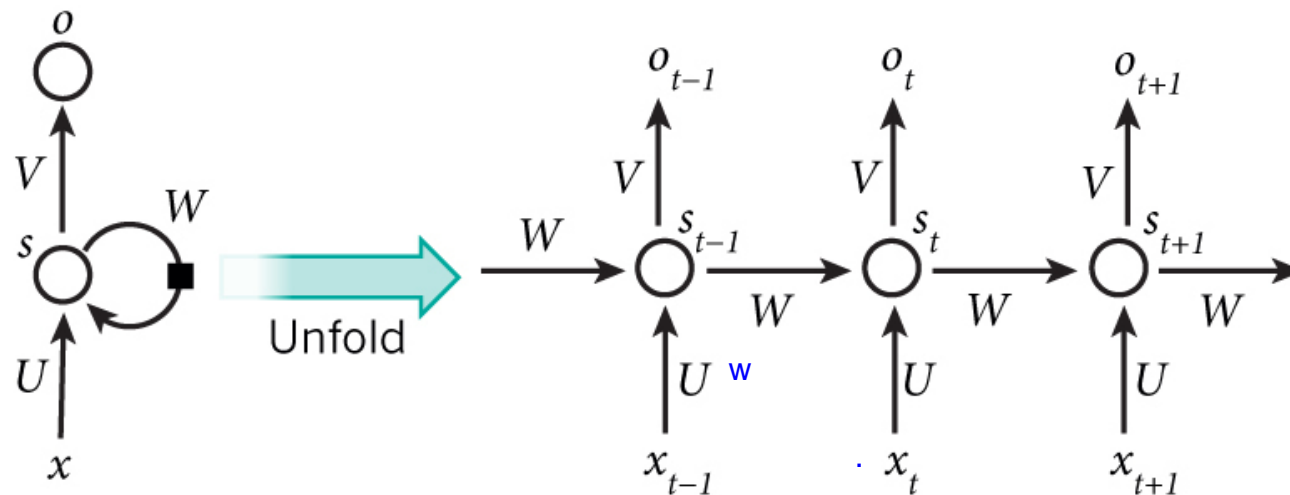
가

가

.

*** _RNN) . w w , 0
 - w 0 가 .() .
 -) 가 (cnn)
 -)_) 가 .
 - : + w=1
 - : LSTM._

Recurrent Neural Networks (RNN)



$$h_t = \tanh(Ux_{t-1} + Ws_{t-1})$$

$$o_t = Vs_t$$

(UXt1+Wst - 1)
- 1~1

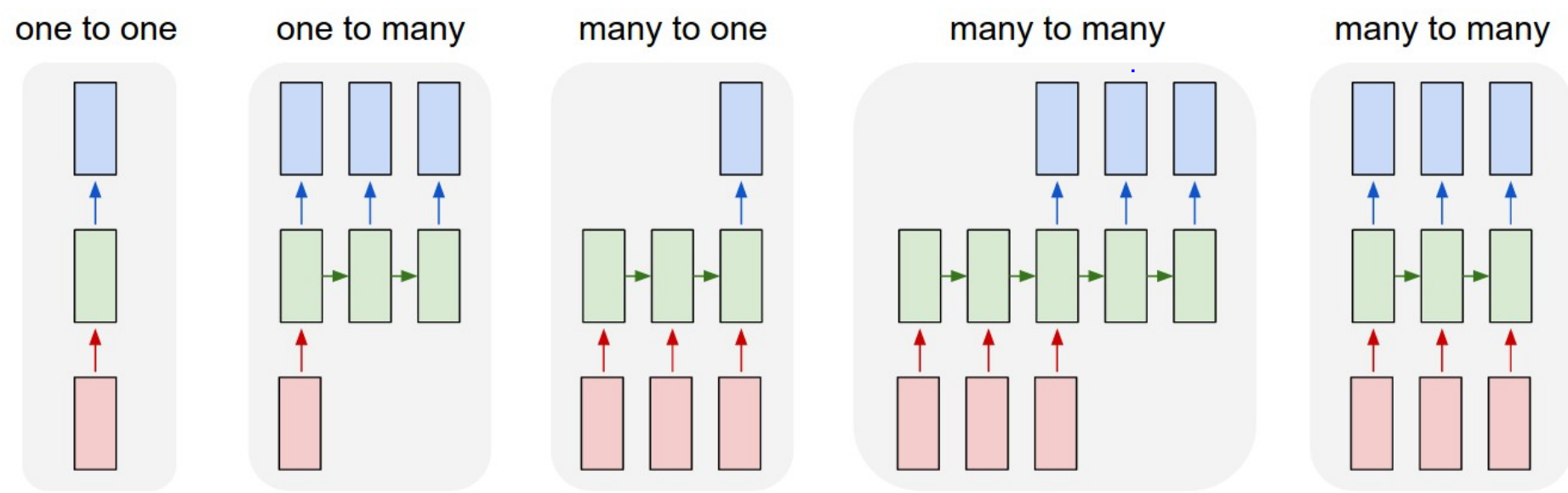
state가 ,

*** _RNN) w 가 .(w 0
 - w 0 가 .(w 0
 -) 가 (cnn)
 -)_) 가 .
 - : + w=1

Recurrent Neural Networks (RNN)

+ LSTM._ + GRU+
 >> LSTM: w
 >>

CNN RNN 가 :가 ?!

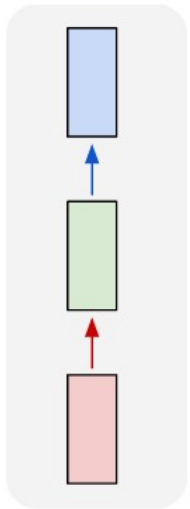


one-to-one: FFN과 같음!
 one-to-many: Image Captioning, Generation 등

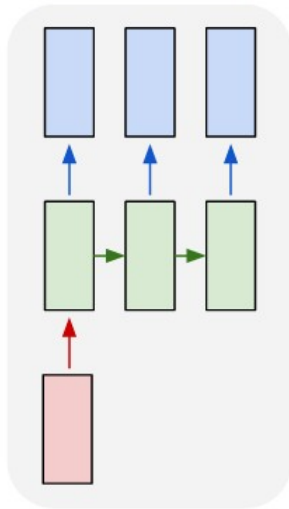
+ 가 가

Recurrent Neural Networks (RNN)

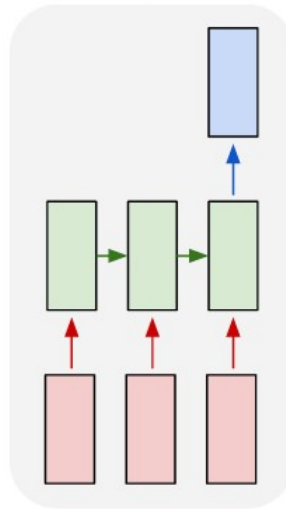
one to one



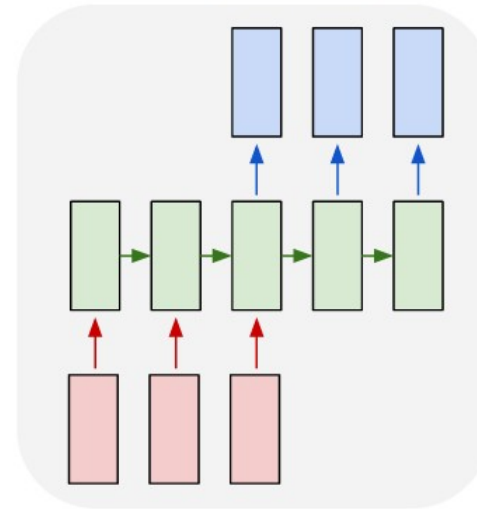
one to many



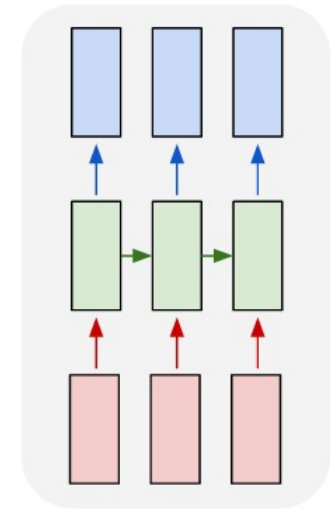
many to one



many to many



many to many

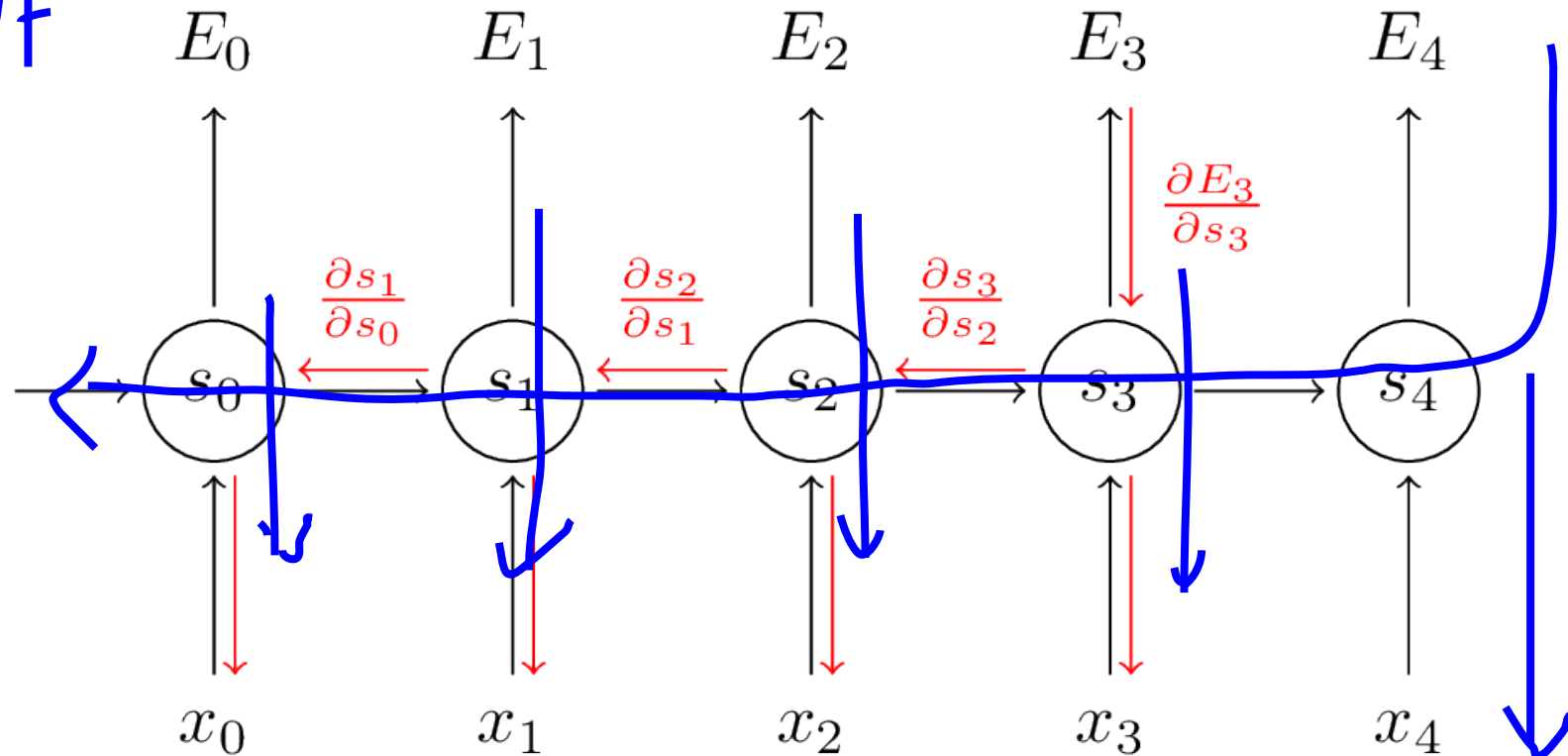


_ x0 y0 yn 가 loss

$$W = \partial L_1 + \partial L_2 + \partial L_3$$

RNNs – Backpropagation through Time (BPTT)

Gradient



INFORMATION LOSS

: gpu

- RNN

- $x_2 > x_1 > x_0$.

가

가

가

Weakness of RNNs – Information loss

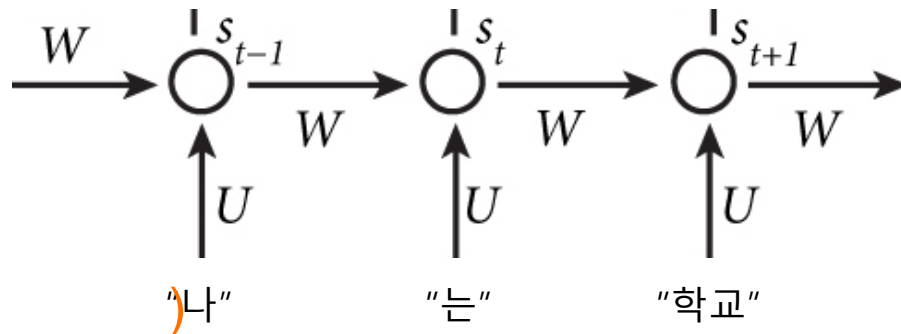
Gradient W () 가 1

- s_4
가

○ w

0

w 가 1



0 ○

$$s_1 = \tanh(Ux_1 + Ws_0)$$

$$s_2 = \tanh(Ux_2 + Ws_1) = \tanh(Ux_2 + W \tanh(Ux_1 + Ws_0))$$

$$s_3 = \tanh(Ux_3 + Ws_2) = \tanh(Ux_3 + W \tanh(Ux_2 + W \tanh(Ux_1 + Ws_0)))$$

초기 상태 벡터는 거대한 매트릭스 곱으로 0에 수렴함!

RNN)

- 0 ○

RNN)

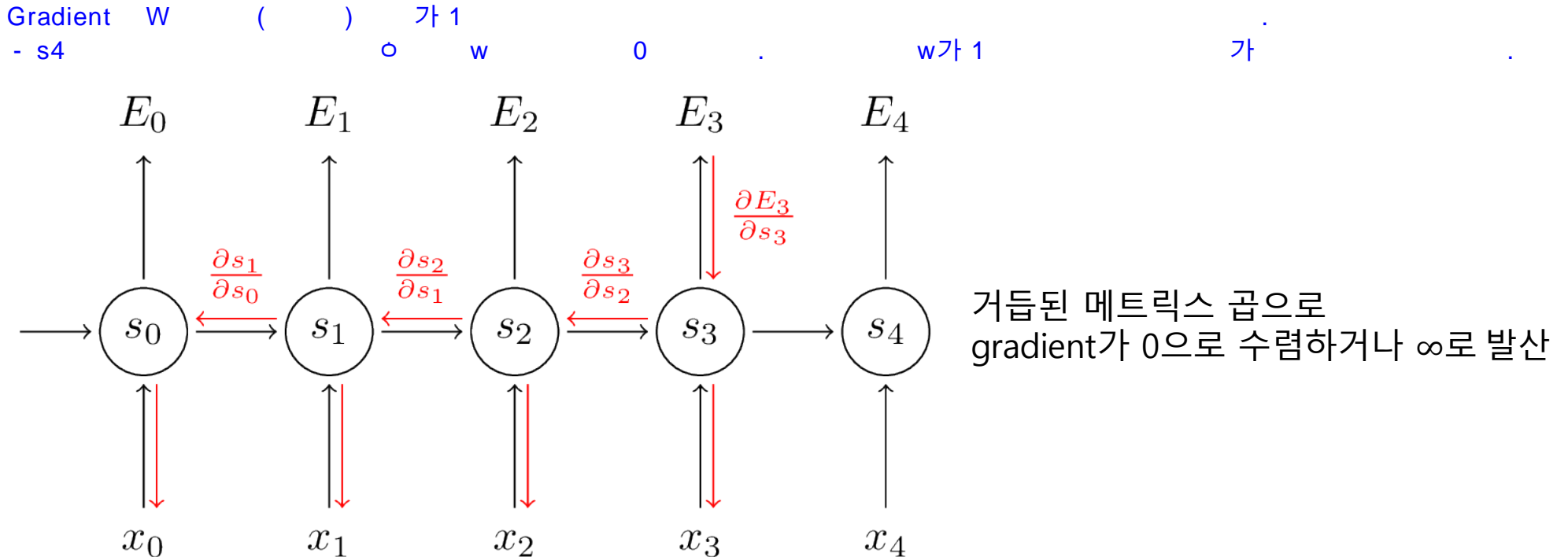
- CNN

)

- RNN

(CNN

Weakness of RNNs – Gradient Vanishing/Exploding



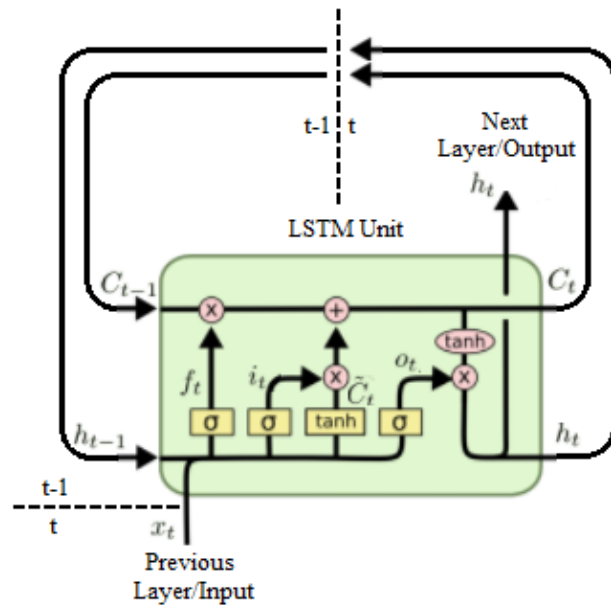
$$f(x_1) = Wx_1$$

$$f(x_2) = W^2x_1$$

$$f(x_{100}) = W^{99}x_1$$

$$f(x_t) = Wx_{t-1}$$

Long-Short Term Memory (LSTM)



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

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

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

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Forget Gate

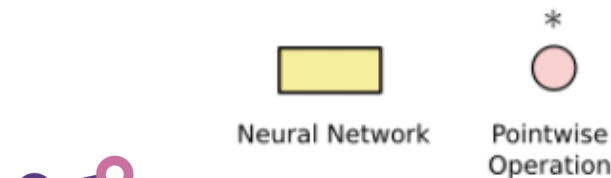
이전 상태를 얼마나 잊을지

Input Gate

현재 입력 정보를 얼마나 받을지

Output Gate

새로 계산된 정보를 얼마나 상태정보로 출력할지



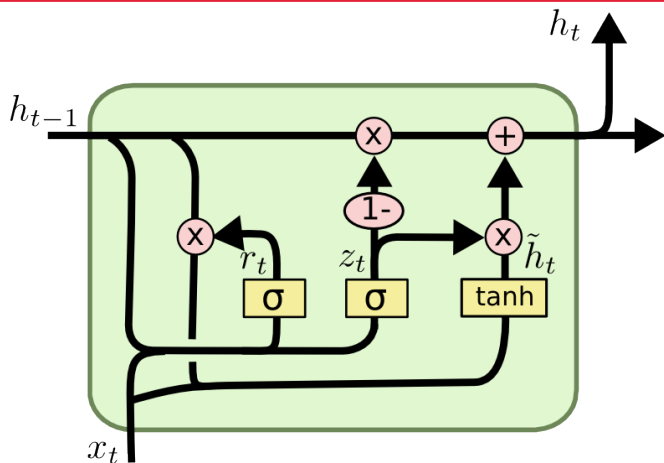
$$f(x_2) = Wx_1 + x_1$$

$$f(x_3) = Wx_2 + x_2$$

$$= W(Wx_1 + x_1) + Wx_1 + x_1$$

$$= W^2x_1 + 2Wx_1 + x_1$$

Gated Recurrent Unit (GRU)



Update Gate

새로 계산된 정보와 이전 정보를 어느 정도로 조합할지

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

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

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

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

Forget Gate

이전 상태를 얼마나 잊을지

In PyTorch ...

CLASS `torch.nn.GRU(*args, **kwargs)`

Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned}r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}\end{aligned}$$

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two GRUs together to form a *stacked GRU*, with the second GRU taking in outputs of the first GRU and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights b_{ih} and b_{hh} . Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each GRU layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional GRU. Default: `False`

Inputs: input, h_0

- **input** of shape $(seq_len, batch, input_size)$: tensor containing the features of the input sequence. The input can also be a packed variable length sequence. See [torch.nn.utils.rnn.pack_padded_sequence\(\)](#) for details.
- **h_0** of shape $(num_layers * num_directions, batch, hidden_size)$: tensor containing the initial hidden state for each element in the batch. Defaults to zero if not provided. If the RNN is bidirectional, `num_directions` should be 2, else it should be 1.

In PyTorch ...

CLASS `torch.nn.LSTM(*args, **kwargs)`

Inputs: `input`, `(h_0, c_0)`

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned}i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\c_t &= f_t * c_{(t-1)} + i_t * g_t \\h_t &= o_t * \tanh(c_t)\end{aligned}$$

- **input** of shape $(seq_len, batch, input_size)$: tensor containing the features of the input sequence. The input can also be a packed variable length sequence. See

`torch.nn.utils.rnn.pack_padded_sequence()` or

`torch.nn.utils.rnn.pack_sequence()` for details.

- **h_0** of shape $(num_layers * num_directions, batch, hidden_size)$: tensor containing the initial hidden state for each element in the batch. If the LSTM is bidirectional, `num_directions` should be 2, else it should be 1.
- **c_0** of shape $(num_layers * num_directions, batch, hidden_size)$: tensor containing the initial cell state for each element in the batch.

If `(h_0, c_0)` is not provided, both **h_0** and **c_0** default to zero.

In PyTorch ...

CLASS `torch.nn.GRU(*args, **kwargs)`

Inputs: input, h_0

Applies a multi-layer gated recurrent unit (GRU) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

$$\begin{aligned}r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}\end{aligned}$$

- **input** of shape $(seq_len, batch, input_size)$: tensor containing the features of the input sequence. The input can also be a packed variable length sequence. See [torch.nn.utils.rnn.pack_padded_sequence\(\)](#) for details.
- **h_0** of shape $(num_layers * num_directions, batch, hidden_size)$: tensor containing the initial hidden state for each element in the batch. Defaults to zero if not provided. If the RNN is bidirectional, num_directions should be 2, else it should be 1.

감사합니다

Any Questions?

rudvlf0413@korea.ac.kr