



# *Summer NLP*

## Lec 05. Sequence Modeling Part 1

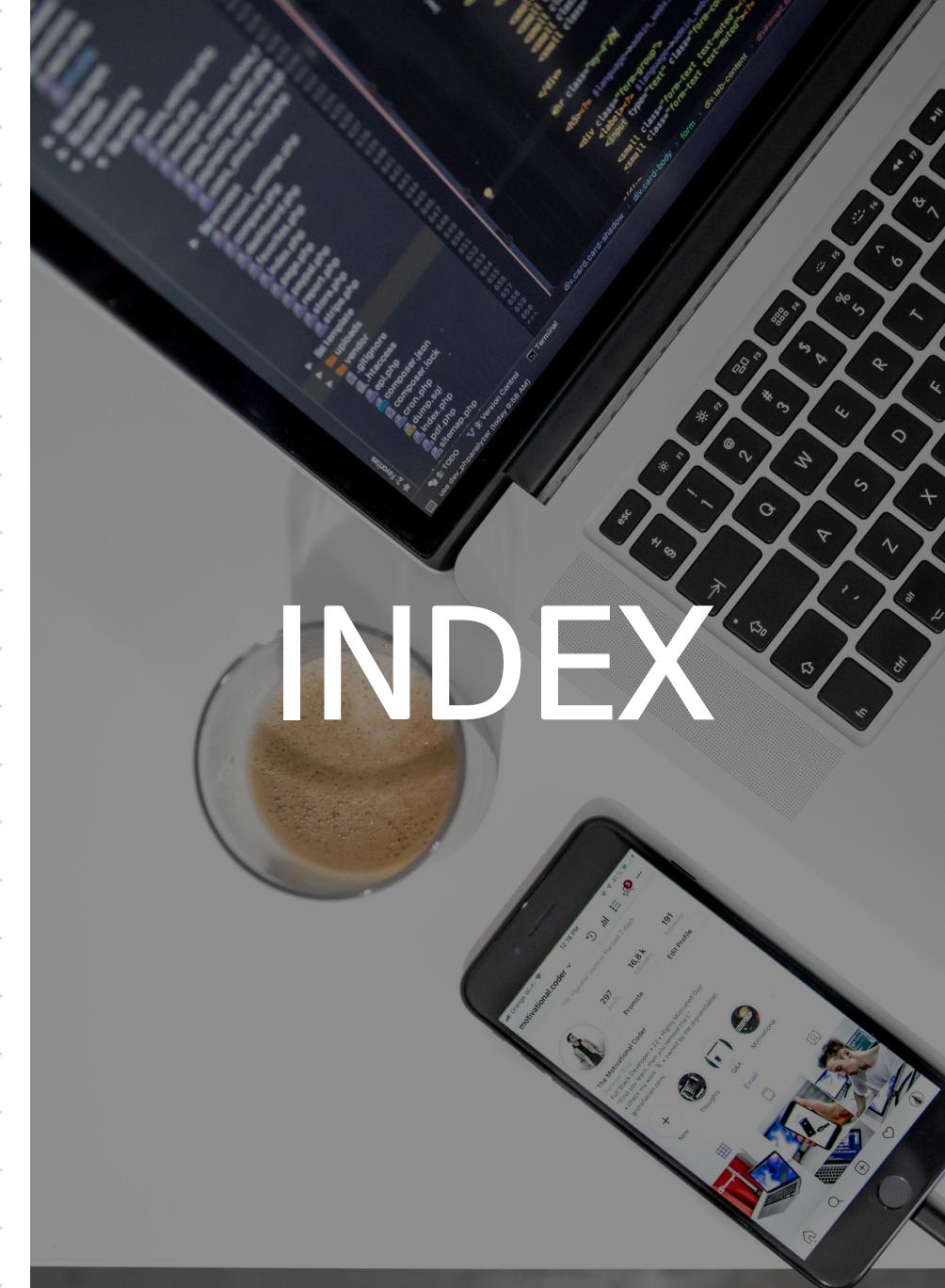
파이토치로 배우는 자연어 처리 CHAPTER 6

| Elman RNN

| 실습 : Surname Classification

| Gating : LSTM and GRU

| 실습 : Surname Generation



## Method 2. 언어 모델 “단어가 어떤 순서로 쓰였는가?”

## Elman RNN

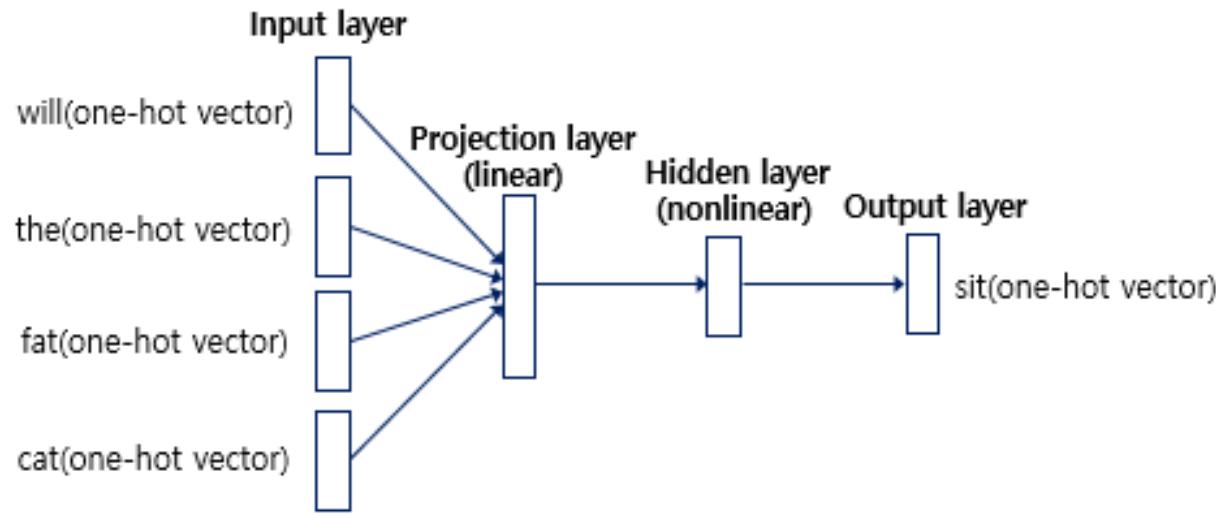
$$P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{n=1}^n P(w_n | w_1, \dots, w_{n-1})$$

| Example                       | Probability |
|-------------------------------|-------------|
| The cat <b>sat</b> on the mat | 0.95        |
| The cat <b>sad</b> on the mat | 0.20        |
| <hr/>                         |             |
| High wind tonight             | 0.97        |
| Large wind tonight            | 0.31        |

## 언어 모델 (Language Model)

단어 시퀀스에 확률을 부여하는 모델로, 단어의 순서와 연관이 있음.

# Elman RNN



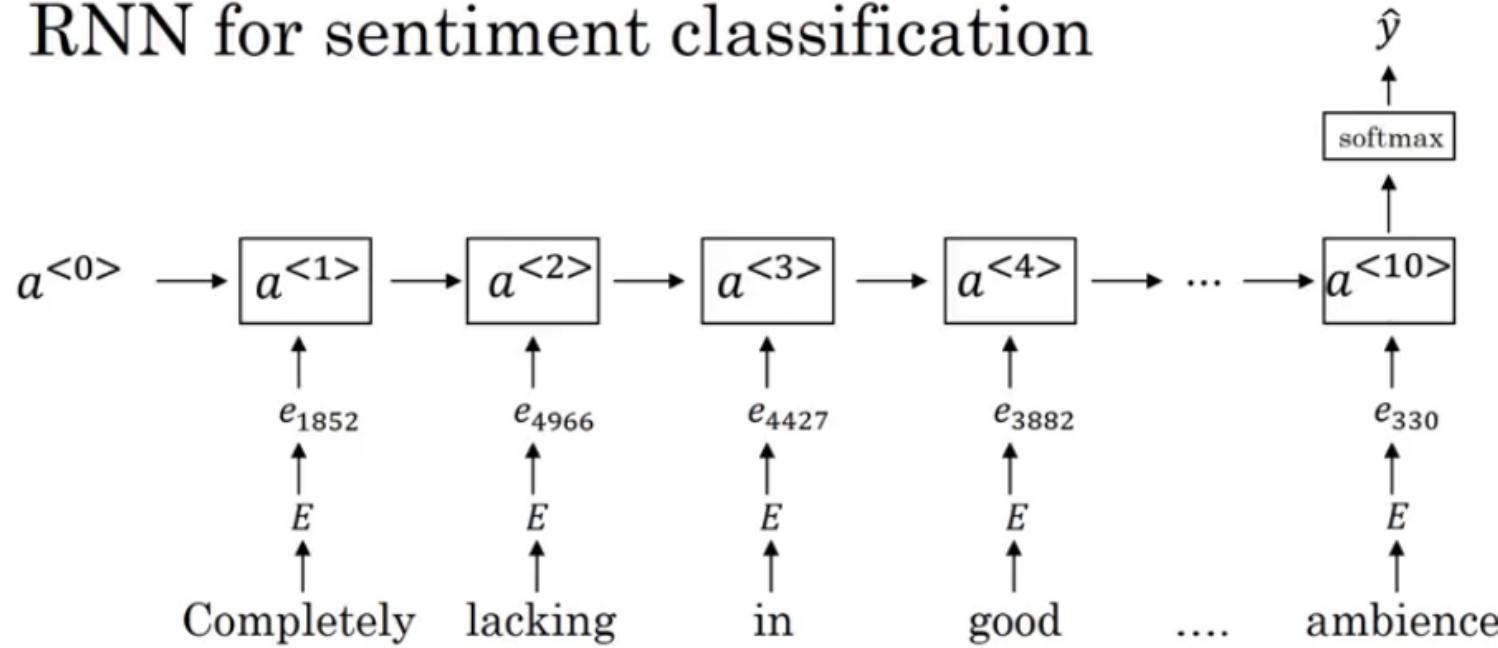
## 언어모델의 목적

단어의 의미를 하나의 벡터로 내용을 함축.

## Sequence Modeling 문장 정보를 어떻게 함축할 수 있을까?

# Elman RNN

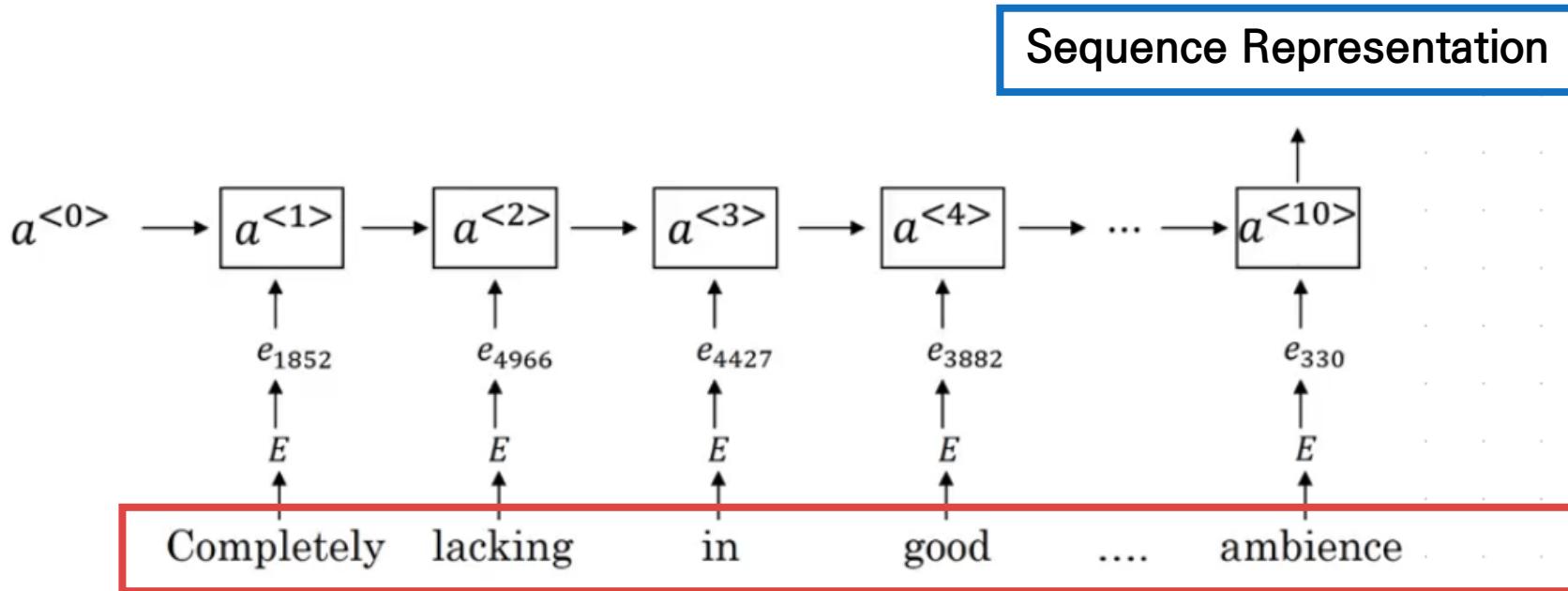
## RNN for sentiment classification



## Sequence Modeling with Recurrent NN

각 단어를 순서대로 입력시키며 Hidden State를 Update

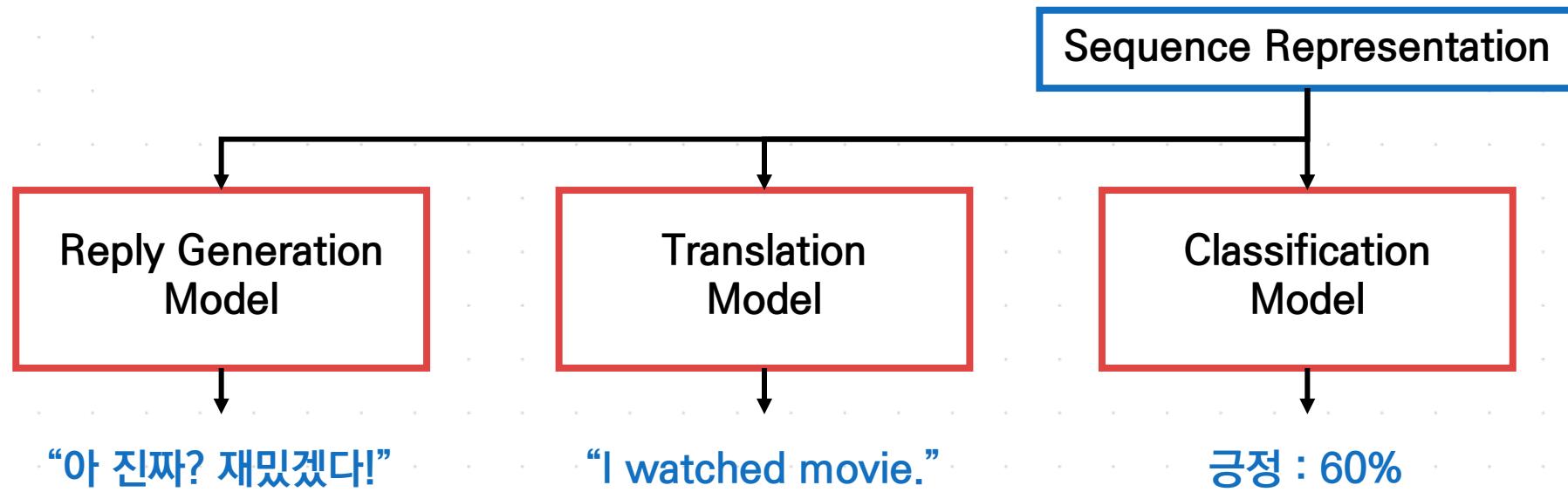
# Elman RNN



## What do we do with Hidden State?

문장 전체의 의미를 벡터에 함축하며, Sequence Representation이라고도 부름.

# Elman RNN



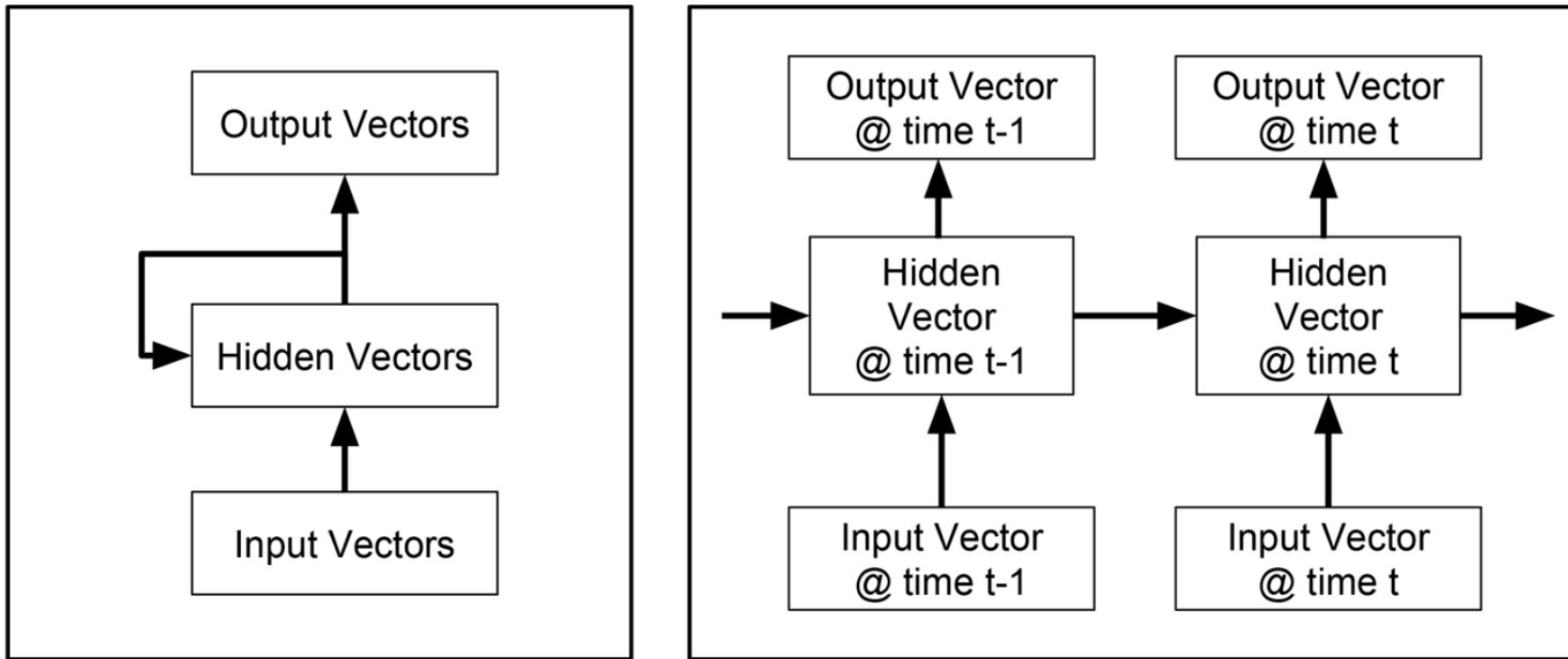
## What do we do with Hidden State?

이렇게 만들어낸 Representation 뒤에 모델을 붙여 다양한 Task를 해결할 수 있음.

**How can we get the “Hidden State”**

네트워크가 어떻게 Sequence로부터 Hidden State를 추출해낼까?

# Elman RNN



## Recurrent Neural Network

이전 timestep의 hidden state를 현재 timestep에 ‘되먹임’

# Elman RNN



제프리 엘먼 (Jeffrey Elman) 이미지 더보기

미국 심리언어학자

영어에서 번역됨 - Jeffrey Locke Elman은 샌디에고 캘리포니아 대학교에서 미국 심리학자이자인지 과학 교수로 재직했습니다. 그는 신경 네트워크 분야를 전문으로했습니다.

위키백과(영어)

원래 설명 보기 ▾

출생: 1948년 1월 22일

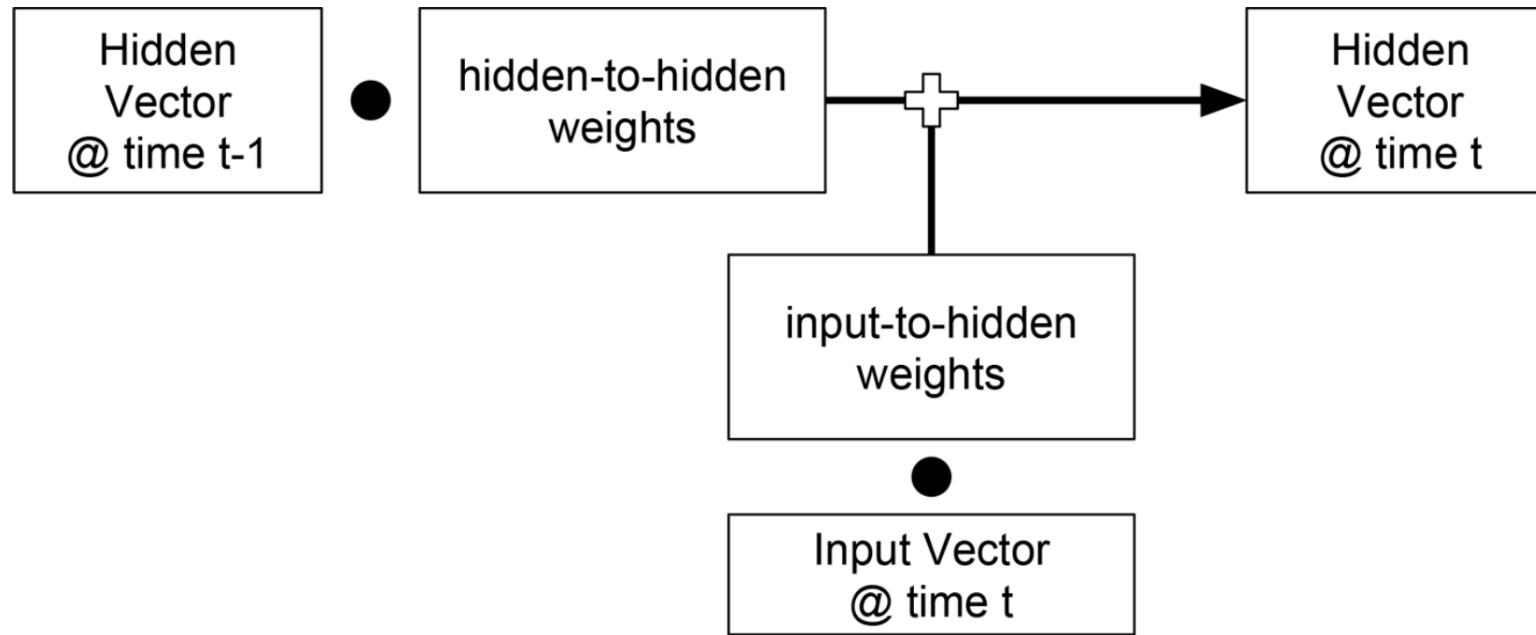
사망 정보: 2018년 6월 28일

수상: Rumelhart Prize

학력: 하버드 대학교, 텍사스 대학 - 오스틴

**Recurrent Neural Network : ‘ElmanRNN’**  
아이디어를 제시한 사람의 이름을 따서 ElmanRNN이라 부름

# Elman RNN



## Output of Recurrent Neural Network

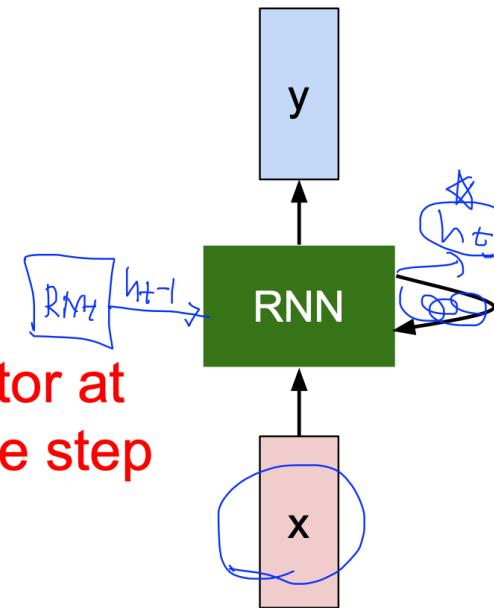
이전 timestep의 hidden state를 hidden-hidden weight랑 곱하고,  
현재 timestep의 input vector를 input-hidden weight랑 곱해서, 둘을 더함.

# Recurrent Neural Network

We can process a sequence of vectors  $\mathbf{x}$  by applying a recurrence formula at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

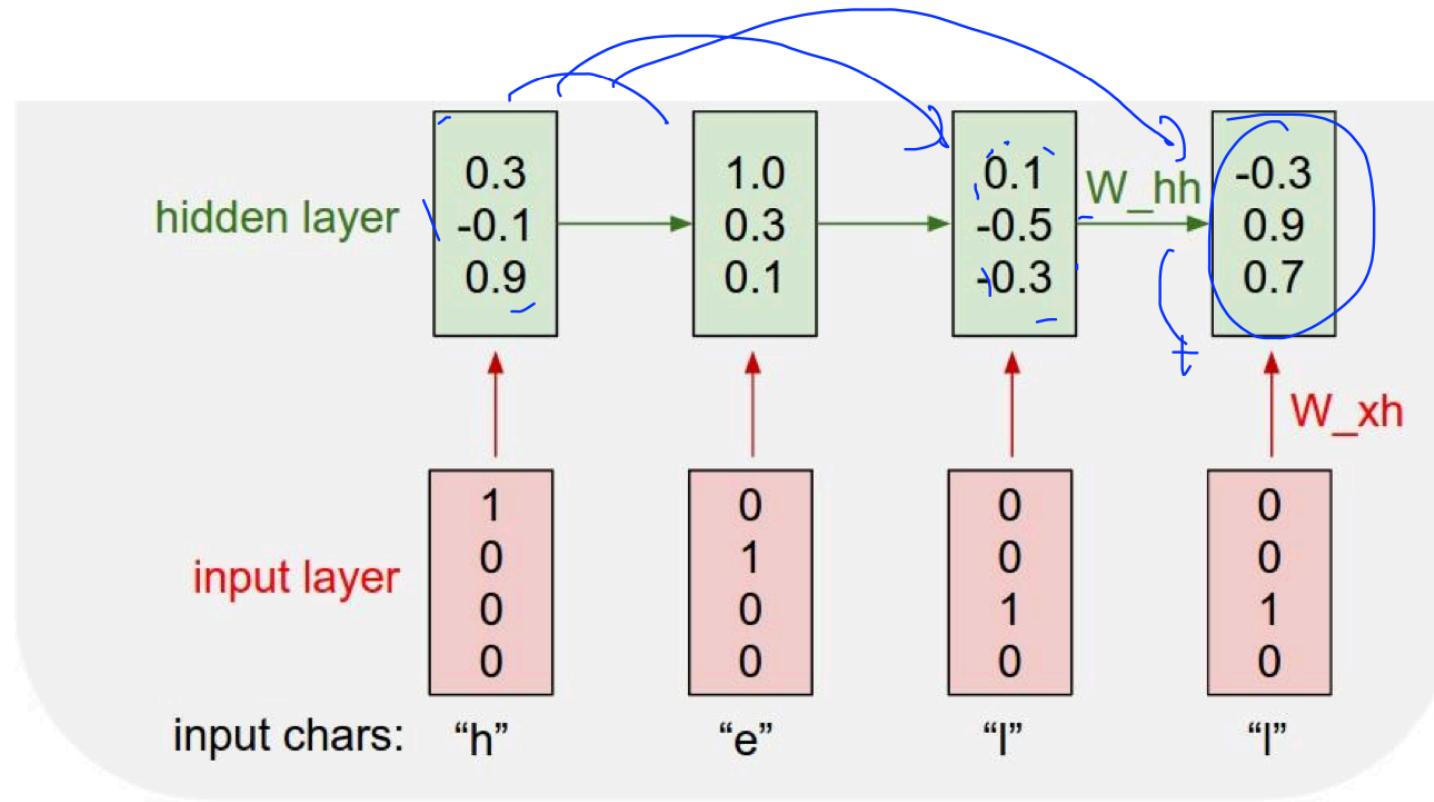
new state      /      old state      input vector at some time step  
 some function with parameters W



## Output of Recurrent Neural Network

이전 timestep의 hidden state를 hidden-hidden weight랑 곱하고,  
현재 timestep의 input vector를 input-hidden weight랑 곱해서, 둘을 더함.

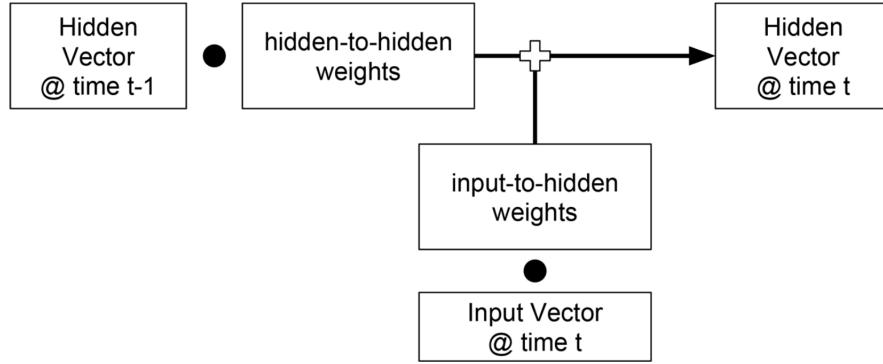
## Elman RNN



## Output of Recurrent Neural Network

이전 timestep의 hidden state를 hidden-hidden weight랑 곱하고,  
현재 timestep의 input vector를 input-hidden weight랑 곱해서, 둘을 더함.

# Elman RNN



## ElmanRNN PyTorch

- RNNCell은 Input Vector와 이전 Timestep의 Hidden Vector를 받아 현재 Hidden Vector를 내보내는 PyTorch 기본 모듈이다.
- 오른쪽 모델의 Output은 각 TimeStep의 Hidden State들을 담는다.

```
def forward(self, x_in, initial_hidden=None):
    """ ElmanRNN의 정방향 계산

    매개변수:
        x_in (torch.Tensor): 입력 데이터 텐서
            If self.batch_first: x_in.shape = (batch_size, seq_size, feat_size)
            Else: x_in.shape = (seq_size, batch_size, feat_size)
        initial_hidden (torch.Tensor): RNN의 초기 은닉 상태

    반환값:
        hiddens (torch.Tensor): 각 타임 스텝에서 RNN 출력
            If self.batch_first:
                hiddens.shape = (batch_size, seq_size, hidden_size)
            Else: hiddens.shape = (seq_size, batch_size, hidden_size)
    """

    if self.batch_first:
        batch_size, seq_size, feat_size = x_in.size()
        x_in = x_in.permute(1, 0, 2)
    else:
        seq_size, batch_size, feat_size = x_in.size()

    hiddens = []

    if initial_hidden is None:
        initial_hidden = self._initial_hidden(batch_size)
        initial_hidden = initial_hidden.to(x_in.device)

    hidden_t = initial_hidden

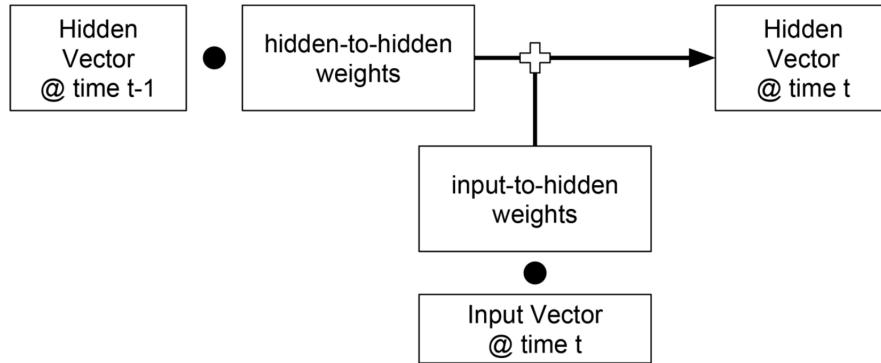
    for t in range(seq_size):
        hidden_t = self.rnn_cell(x_in[t], hidden_t)
        hiddens.append(hidden_t)

    hiddens = torch.stack(hiddens)

    if self.batch_first:
        hiddens = hiddens.permute(1, 0, 2)

    return hiddens
```

# Elman RNN



## ElmanRNN PyTorch

- RNNCell은 Input Vector와 이전 Timestep의 Hidden Vector를 받아 현재 Hidden Vector를 내보내는 PyTorch 기본 모듈이다.
- 오른쪽 모델의 Output은 각 TimeStep의 Hidden State들을 담는다.

## RNNCELL

CLASS `torch.nn.RNNCell(input_size, hidden_size, bias=True, nonlinearity='tanh', device=None, dtype=None)` [\[SOURCE\]](#)

An Elman RNN cell with tanh or ReLU non-linearity.

$$h' = \tanh(W_{ih}x + b_{ih} + W_{hh}h + b_{hh})$$

If `nonlinearity` is 'relu', then ReLU is used in place of tanh.

### Parameters

- input\_size** – The number of expected features in the input  $x$
- hidden\_size** – The number of features in the hidden state  $h$
- bias** – If `False`, then the layer does not use bias weights  $b_{ih}$  and  $b_{hh}$ . Default: `True`
- nonlinearity** – The non-linearity to use. Can be either '`tanh`' or '`relu`'. Default: '`tanh`'

### Inputs: input, hidden

- input** of shape  $(batch, input\_size)$ : tensor containing input features
- hidden** of shape  $(batch, hidden\_size)$ : tensor containing the initial hidden state for each element in the batch. Defaults to zero if not provided.

### Outputs: h'

- h'** of shape  $(batch, hidden\_size)$ : tensor containing the next hidden state for each element in the batch

# Elman RNN

## RNN

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

[SOURCE]

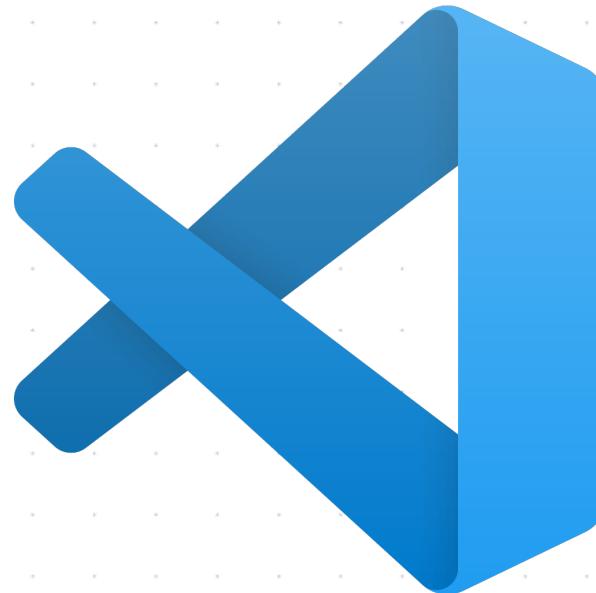
Applies a multi-layer Elman RNN with tanh or ReLU non-linearity to an input sequence.

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

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

where  $h_t$  is the hidden state at time  $t$ ,  $x_t$  is the input at time  $t$ , and  $h_{(t-1)}$  is the hidden state of the previous layer at time  $t-1$  or the initial hidden state at time 0. If `nonlinearity` is `'relu'`, then ReLU is used instead of tanh.

## | 실습 : Surname Classification



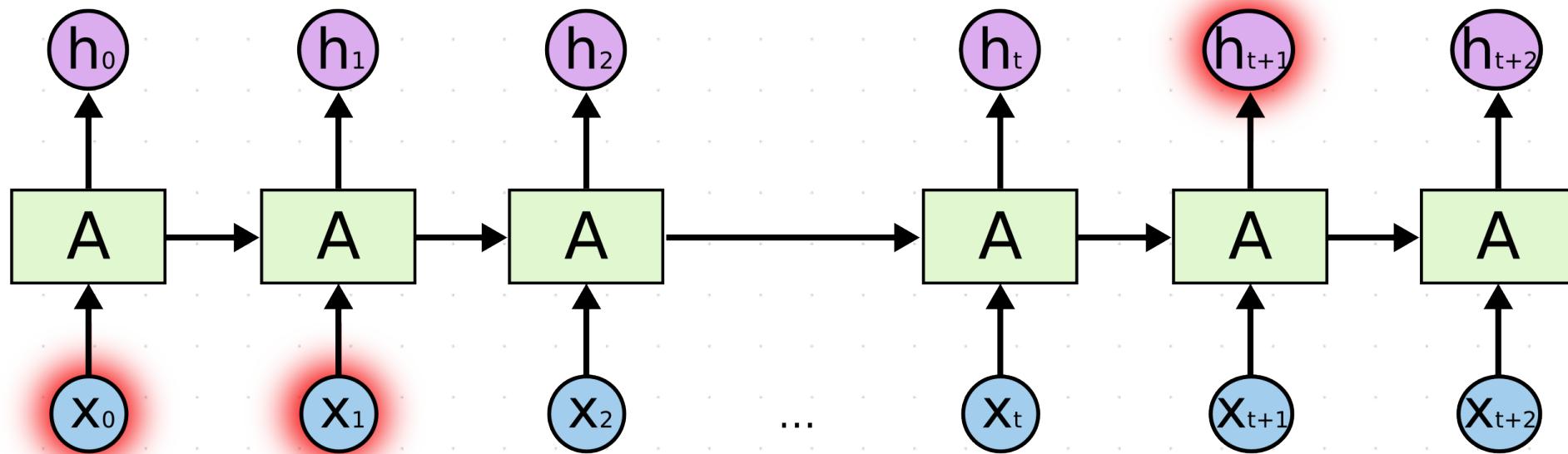
**RNN으로 Surname Classification 풀기**  
[With Colab](#)

## Gating : LSTM and GRU

### Issues in ElmanRNN

지금 ElmanRNN이 잘 안쓰이는 이유가 있겠죠?

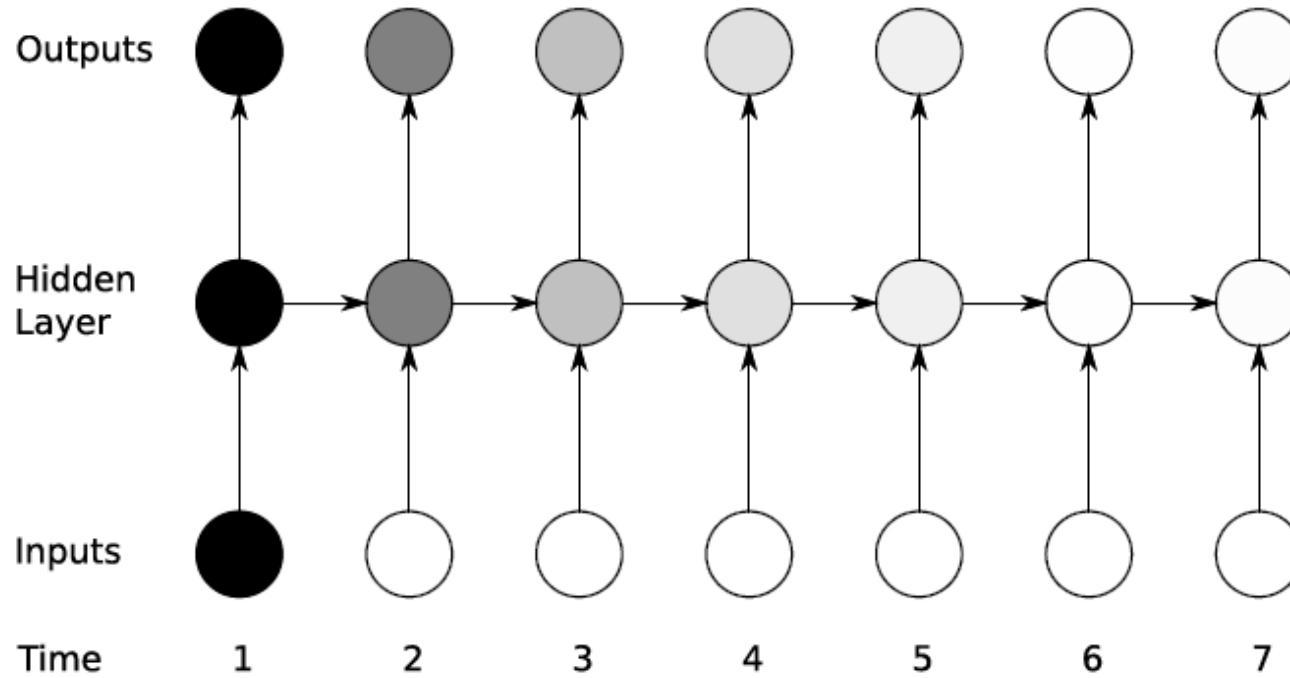
# Gating : LSTM and GRU



## Issue 1 : Long-Term Dependency

Hidden State를 따로 관리하지 않아 오래된 정보는 쉽게 잊어버린다.

# Gating : LSTM and GRU



## Issue 2 : Vanishing Gradient

Gradient가 0에 가깝게 되면 최적화 과정을 불안하게 한다.

## Gating : LSTM and GRU

**How can we control Hidden State?**

Hidden State를 만들 때 어떻게 필요한 정보만 담도록 할 수 있을까?

## Gating : LSTM and GRU

$$a + \lambda b$$

Gating

a와 새로운 값 b를 더할 때  $\lambda$ 값으로 b가 포함되는 정도를 결정

## Gating : LSTM and GRU

$$h_t = h_{t-1} + F(h_{t-1}, x_t)$$

### RNN without Gating

다음 Hidden State를 만들 때, Network의 결과가 그대로 반영.

## Gating : LSTM and GRU

$$h_t = h_{t-1} + \lambda(h_{t-1}, x_t) F(h_{t-1}, x_t)$$

### RNN with Gating : Input Gating

$\lambda$ 를 입력과 hidden state의 Function으로, Context에 따라  
현재 Input의 반영 정도를 결정!

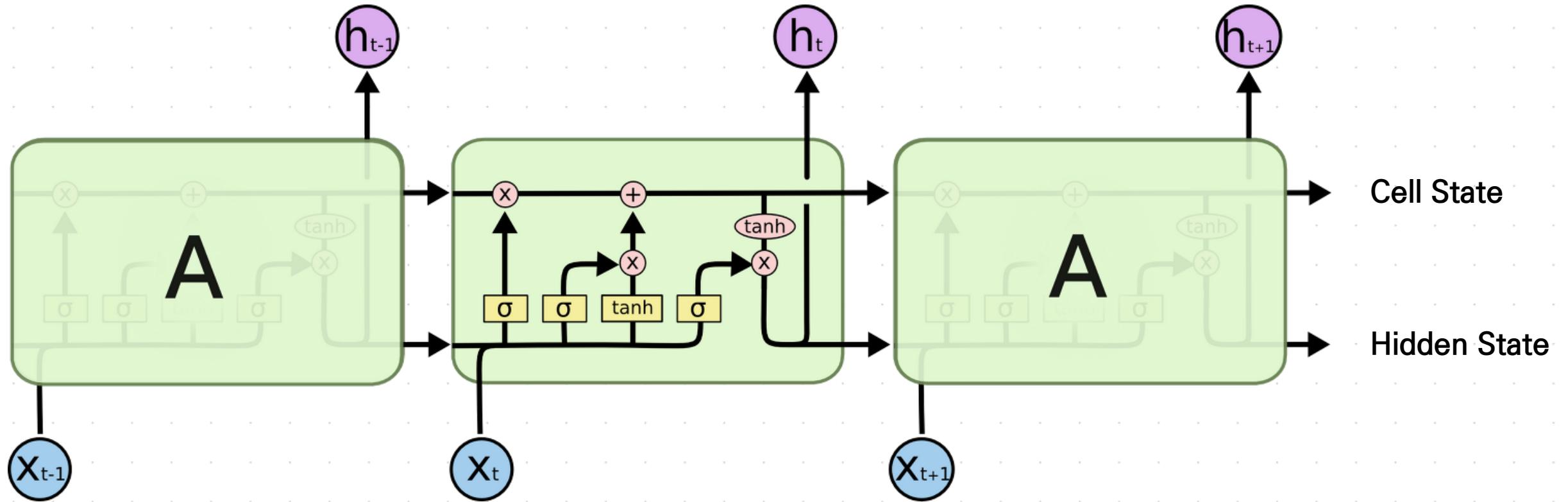
## Gating : LSTM and GRU

$$h_t = \mu(h_{t-1}, x_t) h_{t-1} + \lambda(h_{t-1}, x_t) F(h_{t-1}, x_t)$$

### RNN with Gating : Hidden State Gating

이전 Time step의 hidden state를 현재의 hidden state에  
얼마나 반영할지를 조절

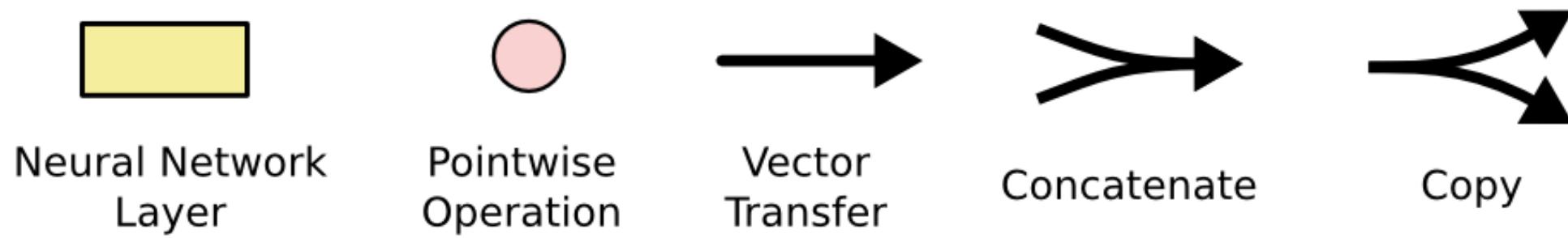
## Gating : LSTM and GRU



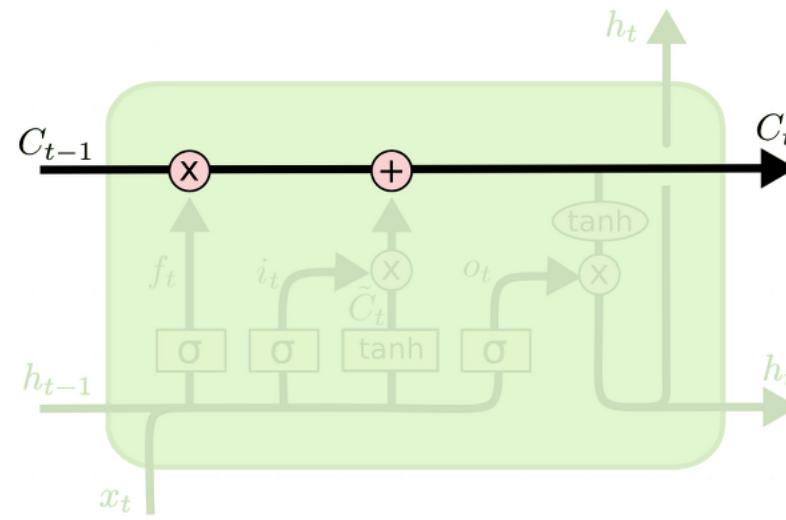
## LSTM : Long Short-Term Memory

Gating을 사용하여 현재 Input State와 이전 step의 Hidden State의 반영 정도를 결정하고, Cell State를 따로 두어서 장기적으로 정보를 가지고 있을 수 있도록 만든다.

# Gating : LSTM and GRU



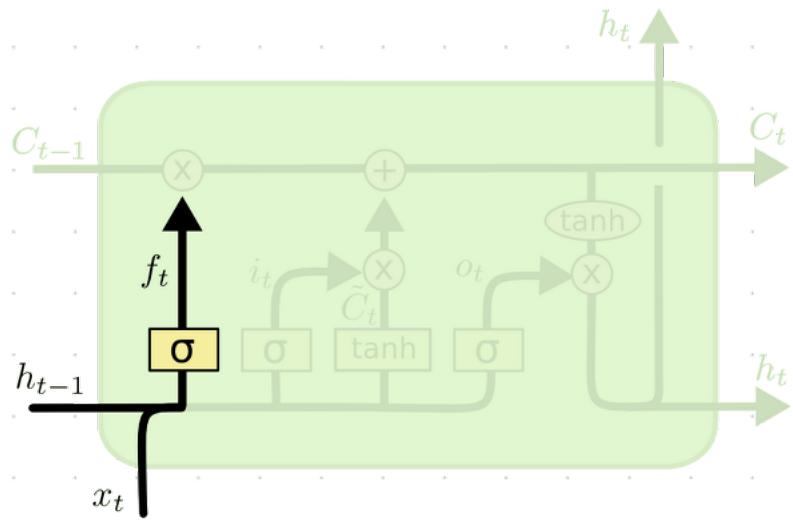
# Gating : LSTM and GRU



## Cell State

Cell State가 LSTM의 장기기억을 담당. 추후 Gating을 통해 이전 step의 Cell State를 얼마나 잊어버릴지, 새로운 입력을 얼마나 반영할지를 결정.

# Gating : LSTM and GRU

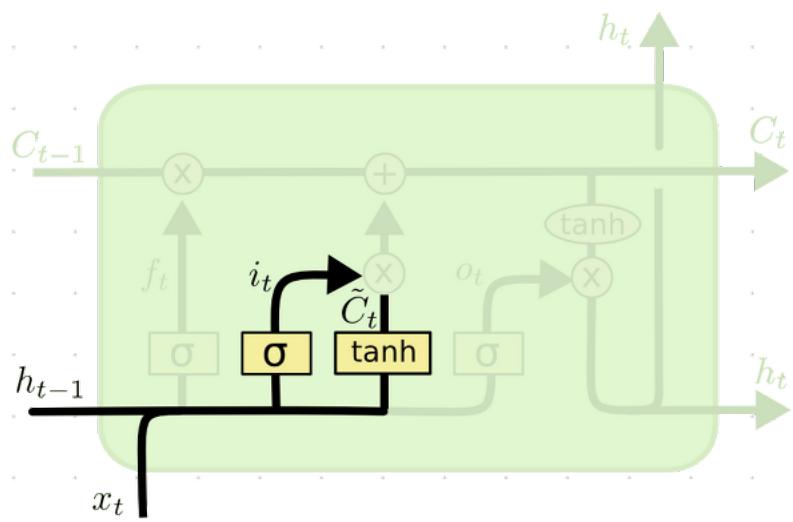


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

## Gate 1 : Forget Gate

현재의 입력과 이전 step의 Hidden State를 바탕으로 (Context를 바탕으로)  
이전 step의 Cell State의 각 요소를 다음 Cell State에 얼마나 반영할지를 결정

# Gating : LSTM and GRU



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

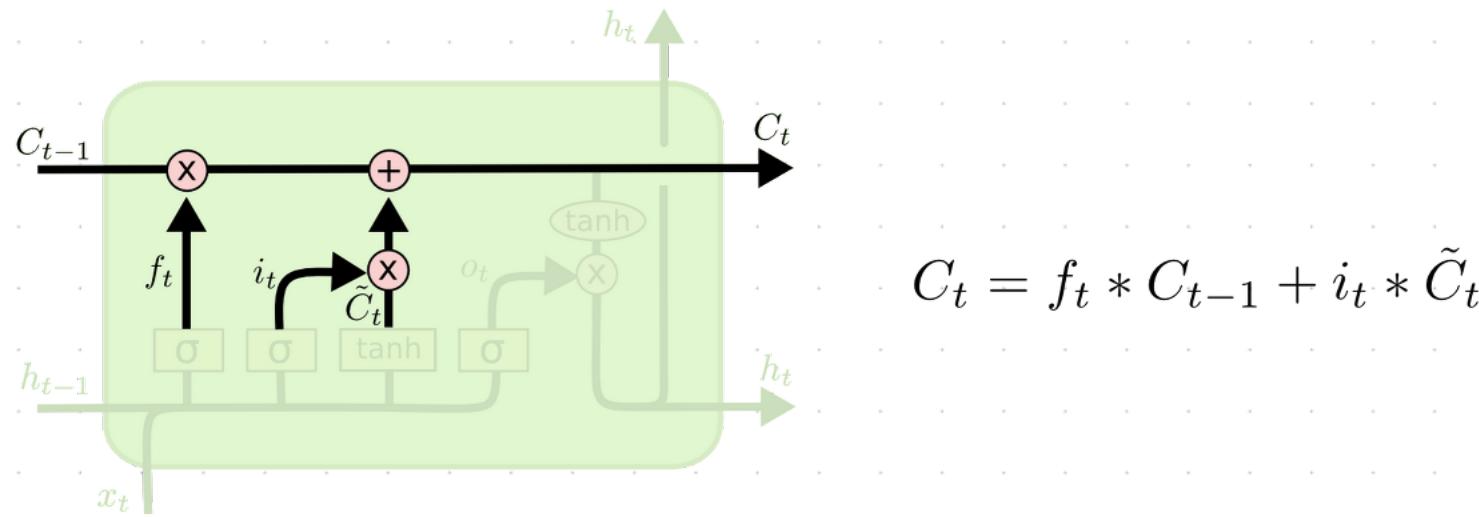
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

## Gate 2 : Input Gate

새로 들어오는 정보를 Cell State에 어떤걸 얼마나 반영할지를 결정.

tanh레이어로 Hidden State 후보를 만들고, sigmoid레이어로 얼마나 반영할지 결정.

# Gating : LSTM and GRU

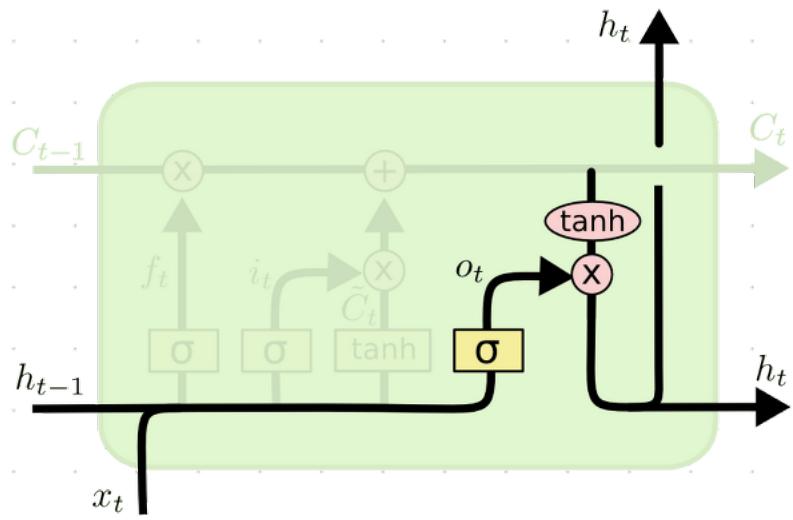


## Gate 2 : Input Gate

새로 들어오는 정보를 Cell State에 어떤걸 얼마나 반영할지를 결정.

tanh레이어로 Hidden State 후보를 만들고, sigmoid레이어로 얼마나 반영할지 결정.

# Gating : LSTM and GRU

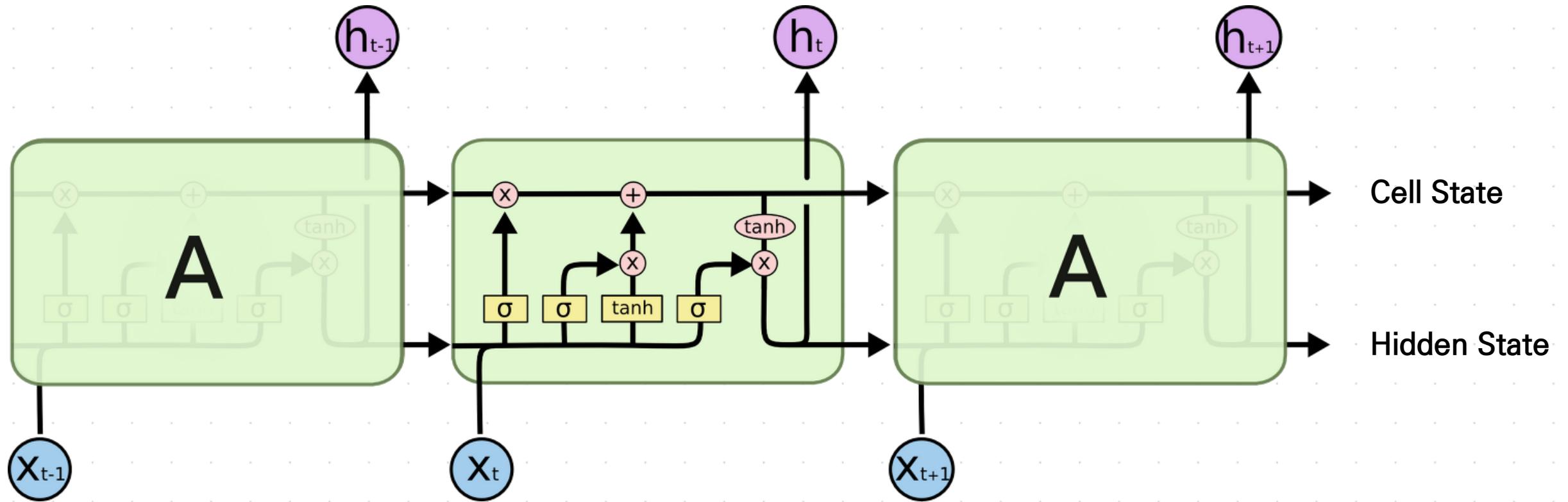


$$o_t = \sigma (W_o [ h_{t-1}, x_t ] + b_o)$$
$$h_t = o_t * \tanh (C_t)$$

## Gate 3 : Output Gate

Cell State를 만드는 과정과 비슷. Input과 이전 step의 Hidden State를 입력으로  $\tanh$ 레이어로 Output후보를 만들고, sigmoid레이어로 얼마나 반영할지 결정.

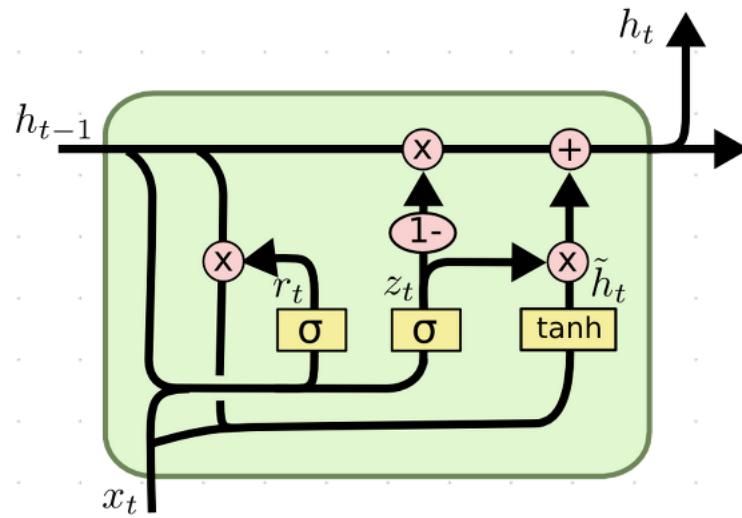
# Gating : LSTM and GRU



## LSTM Full Architecture

결국, LSTM이 하고자 하는 것은 Cell State에 이전 정보를 얼마나 잊어버리고,  
새로운 정보를 얼마나 반영할지를 매 Step마다 반복하여 ElmanRNN의 문제점을 해결.

## Gating : LSTM and GRU



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

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

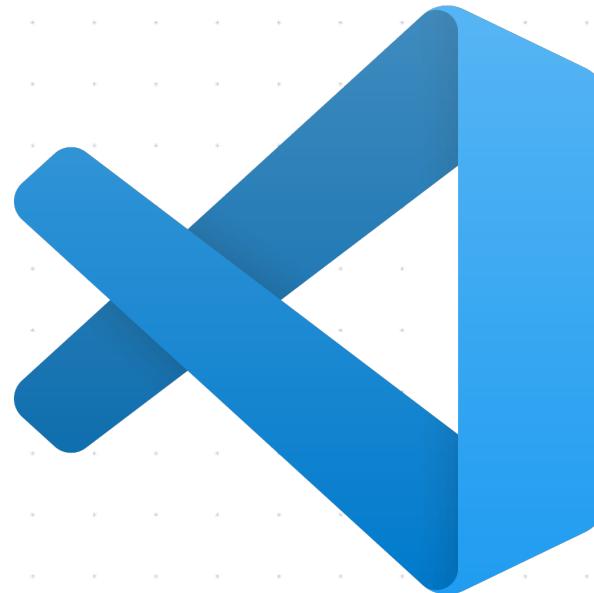
$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

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

## GRU : Gated Recurrent Unit

LSTM의 경량화 버전. Forget Gate와 Input Gate를 합쳐 Update Gate로, Cell State를 따로 두지 않고 Hidden State 하나로만 관리.

## 실습 : Surname Generation



# LSTM으로 Surname Generation 풀기

실습 1. Unconditioned Generation (국적 정보를 활용하지 않음)

실습 2. Conditioned Generation (국적 정보를 활용)



Thank You