

# Language model and RNN



**KOREA**  
UNIVERSITY



Natural Language  
Processing  
& Artificial Intelligence



# Language Model

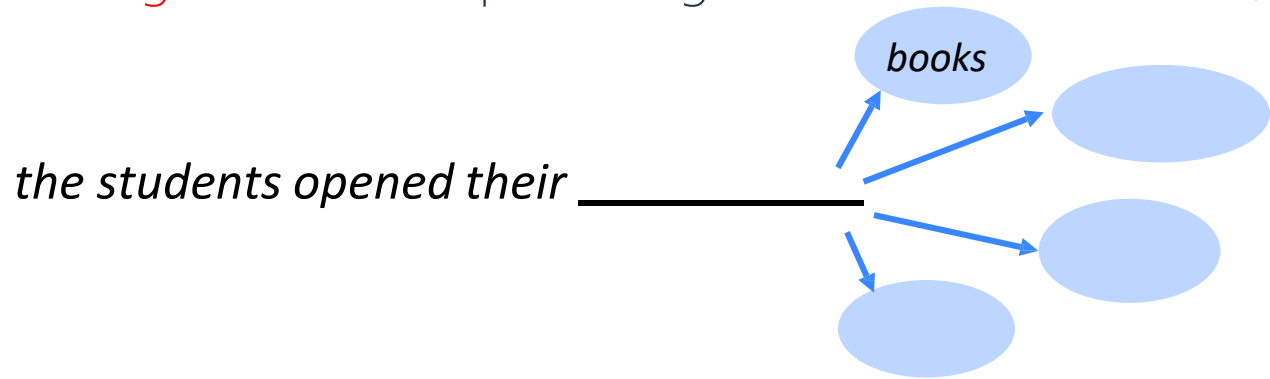
## Overview



# Language Modeling



- **Language Modeling** is the task of predicting what word comes next.



- More formally: given a sequence of word:  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$  compute the probability distribution of the next word  $\mathbf{x}^{(t+1)}$

$$P(\mathbf{x}^{(t+1)} \mid \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$$

where  $\mathbf{x}^{(t+1)}$  can be any word in the vocabulary  $V = \{w_1, \dots, w_{|V|}\}$

- A system that does this is called a **Language Model**.



# Language Modeling

- You can also think of a **Language Model** as a system that assigns probability to a piece of text.
- For example, if we have some text  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$ , then the probability of this text (according to the Language Model) is:

$$\begin{aligned} P(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}) &= P(\mathbf{x}^{(1)}) \times P(\mathbf{x}^{(2)} | \mathbf{x}^{(1)}) \times \dots \times P(\mathbf{x}^{(T)} | \mathbf{x}^{(T-1)}, \dots, \mathbf{x}^{(1)}) \\ &= \prod_{t=1}^T P(\mathbf{x}^{(t)} | \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)}) \end{aligned}$$



This is what our LM provides



# You use Language Models every day!



what is the |



what is the **weather**  
what is the **meaning of life**  
what is the **dark web**  
what is the **xfl**  
what is the **doomsday clock**  
what is the **weather today**  
what is the **keto diet**  
what is the **american dream**  
what is the **speed of light**  
what is the **bill of rights**

Google Search

I'm Feeling Lucky



# n-gram Language Models

*the students opened their\_\_\_\_\_*

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn a n-gram Language Model!
- Definition: A n-gram is a chunk of **n** consecutive words.
  - **uni**grams: “the”, “students”, “opened”, “their”
  - **bi**grams: “the students”, “students opened”, “opened their”
  - **tri**grams: “the students opened”, “students opened their”
  - **4**-grams: “the students opened their”
- Idea: Collect statistics about how frequent different n-grams are, and use these to predict next word.



# n-gram Language Models



- First we make a **simplifying assumption**:  $\mathbf{x}^{(t+1)}$  depends only on the preceding  $n-1$  words.

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)}) = P(\mathbf{x}^{(t+1)} | \overbrace{\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}}^{n-1 \text{ words}}) \quad (\text{assumption})$$

prob of a n-gram

$$\begin{aligned} & P(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \\ &= P(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)}) \end{aligned}$$

prob of a (n-1)-gram

(definition of conditional prob)

$$\approx \frac{\text{count}(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})}{\text{count}(\mathbf{x}^{(t)}, \dots, \mathbf{x}^{(t-n+2)})} \quad (\text{statistical approximation})$$



# n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.



~~as the proctor started the clock, the~~ *students opened their* \_\_\_\_\_  
discard condition on this

$$P(\mathbf{w} | \text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

For example, suppose that in the corpus:

- “students opened their” occurred 1000 times
- “students opened their *books*” occurred 400 times
  - $\rightarrow P(\text{books} | \text{students opened their}) = 0.4$
- “students opened their *exams*” occurred 100 times
  - $\rightarrow P(\text{exams} | \text{students opened their}) = 0.1$

Should we have  
discarded the  
“proctor” context?





# Sparsity Problems with n-gram Language Models



**Problem:** What if “students opened their  $w$ ” never occurred in data? Then  $w$  has probability 0!

**(Partial) Solution:** Add small  $\delta$  to the count for every  $w \in V$ . This is called *smoothing*.

Sparsity Problem 1

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

Sparsity Problem 2

**Problem:** What if “students opened their” never occurred in data? Then we can’t calculate probability for any  $w$ !

**(Partial) Solution:** Just condition on “opened their” instead. This is called *backoff*.

**Note:** Increasing  $n$  makes sparsity problems worse. Typically we can’t have  $n$  bigger than 5.



## Storage Problems with n-gram Language Models

**Storage**: Need to store count for all  $n$ -grams you saw in the corpus.

$$P(\mathbf{w}|\text{students opened their}) = \frac{\text{count}(\text{students opened their } \mathbf{w})}{\text{count}(\text{students opened their})}$$

Increasing  $n$  or increasing corpus increases model size!



# n-gram Language Models in practice

- You can build a simple trigram Language Model over a 1.7 million word corpus (Reuters) in a few seconds on your laptop\*

today the \_\_\_\_\_

Business and financial news

get probability  
distribution

company	0.153
bank	0.153
price	0.077
italian	0.039
emirate ...	0.039

**Sparsity problem:**  
not much granularity  
in the probability  
distribution

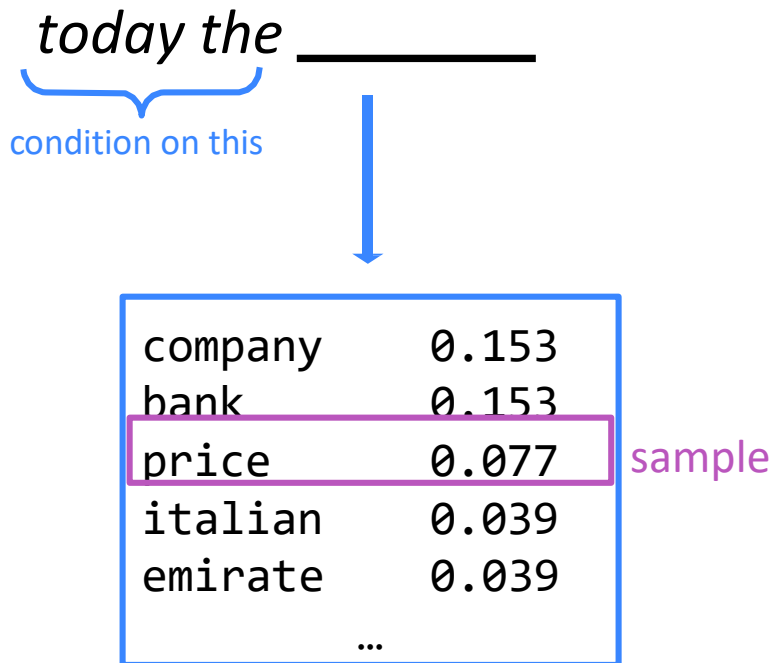
Otherwise, seems reasonable!

\* Try for yourself: <https://nlpforhackers.io/language-models/>



# Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.





# Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

*today the price* \_\_\_\_\_

condition on this



of	0.308	sample
for	... 0.050	
it	0.046	
to	0.046	
is	0.031	



# Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

*today the price of* \_\_\_\_\_

condition on this



the	0.072
18	0.043
oil	0.043
its	0.036
gold	0.018
...	

sample



## Generating text with a n-gram Language Model

- You can also use a Language Model to generate text.

*today the price of gold*



# Generating text with a n-gram Language Model



- You can also use a Language Model to *generate text*.

*today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .*

Surprisingly grammatical!

...but **incoherent**. We need to consider more than three words at a time if we want to model language well.

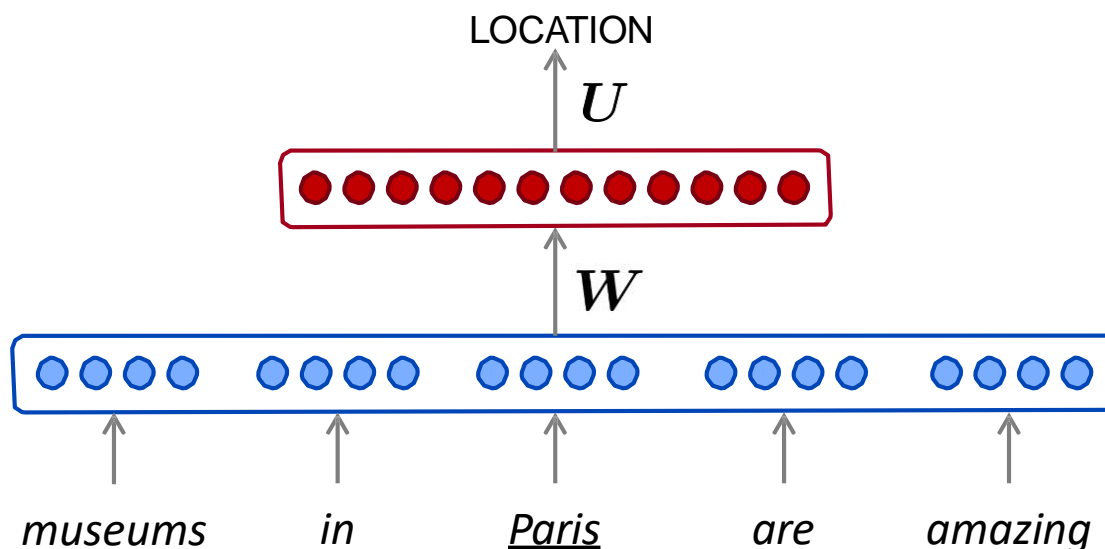
But increasing  $n$  worsens sparsity problem,  
and increases model size...





# How to build a *neural* Language Model?

- Recall the Language Modeling task:
  - Input: sequence of words  $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$
  - Output: prob dist of the next word  $P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$
- How about a **window-based neural model**?





# A fixed-window neural Language Model

~~as the proctor started the clock~~

discard

the students opened their \_\_\_\_\_

fixed window



# A fixed-window neural Language Model

output

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{U}\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden

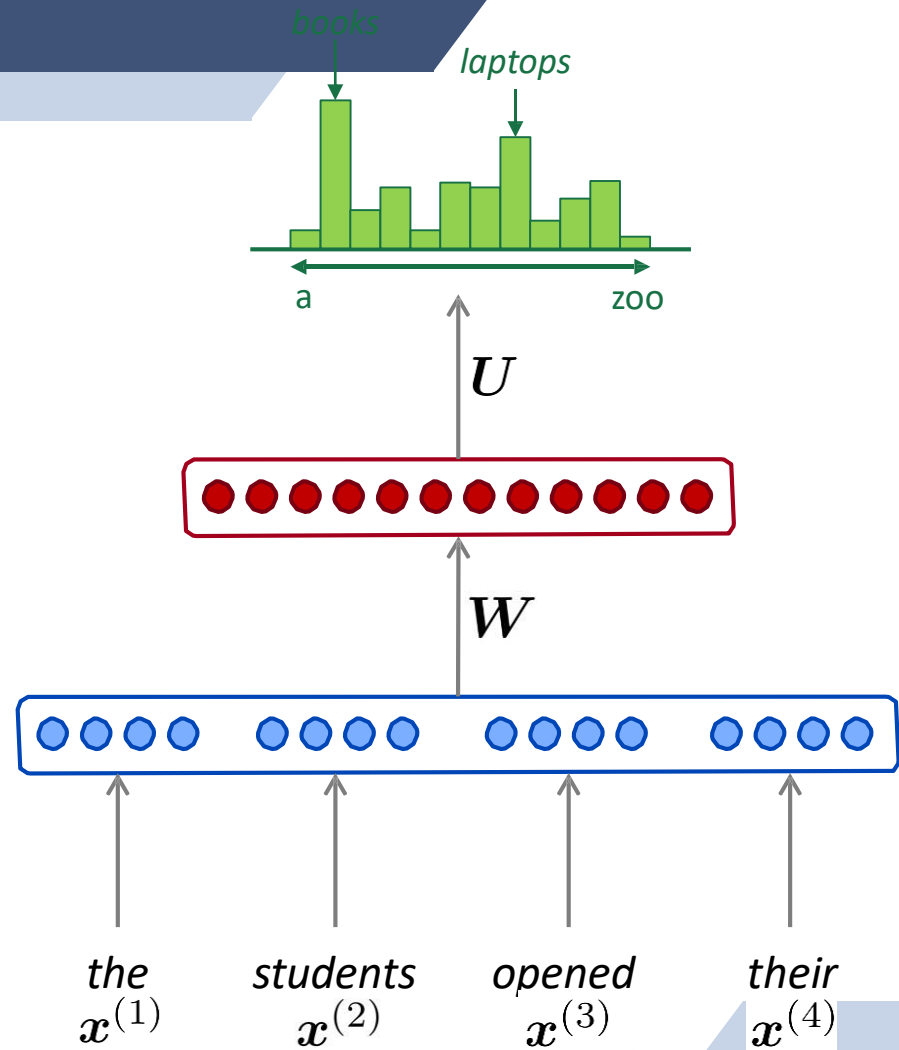
$$\mathbf{h} = f(\mathbf{W}\mathbf{e} + \mathbf{b}_1)$$

concatenated word

$$\mathbf{e} = [\mathbf{e}^{(1)}; \mathbf{e}^{(2)}; \mathbf{e}^{(3)}; \mathbf{e}^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$





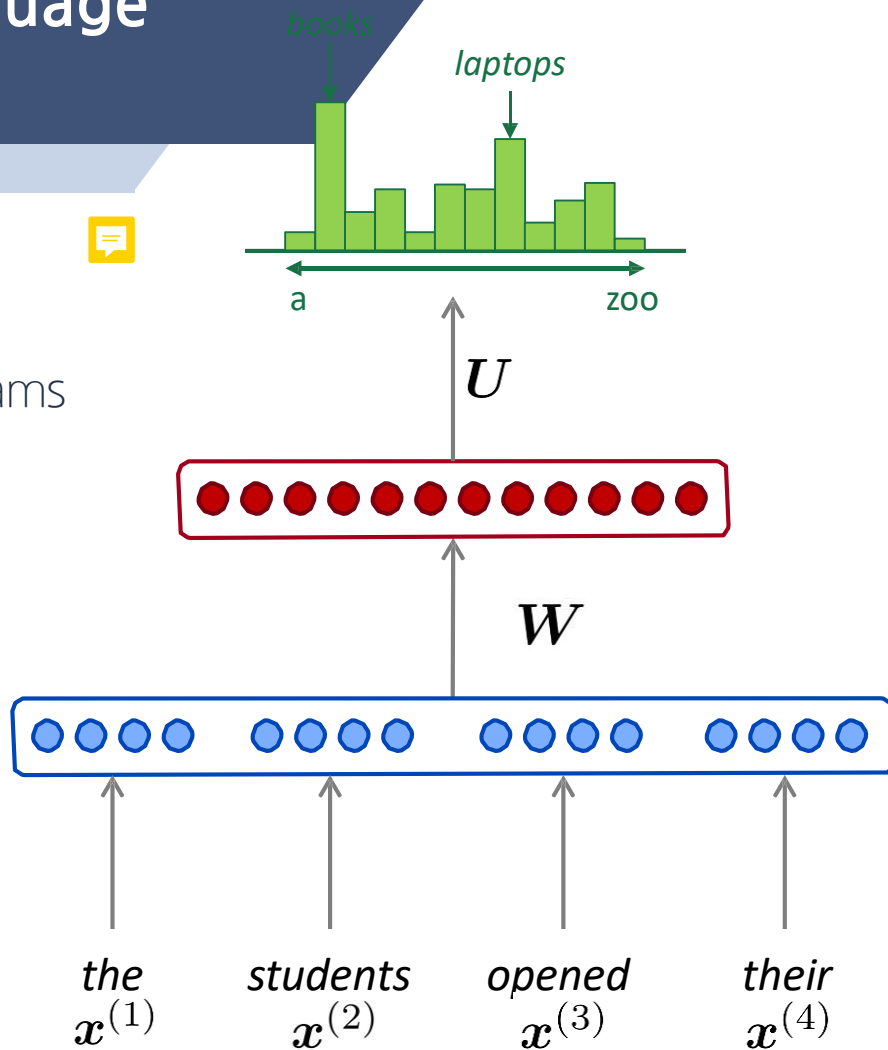
# A fixed-window neural Language Model

Improvements over n-gram LM:

- No sparsity problem
- Don't need to store all observed n-grams

Remaining problems:

- Fixed window is too small
- Enlarging window enlarges  $\mathbf{W}$
- Window can never be large enough!
- $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$  are multiplied by completely different weights in  $\mathbf{W}$ .



# Recurrent Neural Network



# NLP Trends considering RNN



## Recurrent Neural Networks

- ❑ RNNs work around the idea of processing sequential information.
- ❑ Need for Recurrent Networks
  - ❑ Given that a RNN performs sequential processing by modeling units in sequence, it has the ability to capture the inherent sequential nature present in language, where units are characters, words or even sentences.
  - ❑ In a way, RNNs have “memory” over previous computations and use this information in current processing.
    - ➔ And words in a language develop their semantical meaning based on the previous words in the sentence.
  - ❑ It also has the ability to model variable length of text, including very long sentences, paragraphs and even documents.
  - ❑ It is also apt for creating a gist of the sentence in a fixed dimensional hyperspace.
    - ➔ Essential as Many NLP tasks also requires semantic modeling over the whole sentence.
- ❑ RNNs try to create a composition of an arbitrarily long sentence along with unbounded context.



# NLP Trends considering RNN

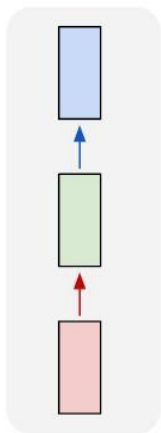


## RNN NLP Applications

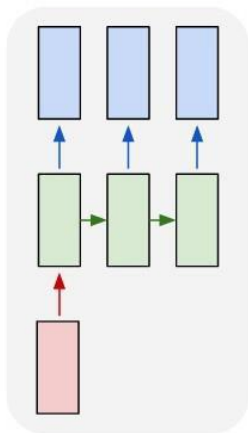
- ❑ RNN for word-level classification:
  - ❑ Has a huge presence in the field of word-level classification.  
→ Lample et al., 2016 proposed to use bidirectional LSTM for NER
  - ❑ Have also shown considerable improvement in language modeling over traditional methods based on count statistics.  
→ Graves, 2013 introduced the effectiveness of RNNs in modeling complex sequences with long range context structures.
- ❑ RNN for sentence-level classification:
  - ❑ Studies show that the dynamics of LSTM gates can capture the reversal effect of the word 'not'.
  - ❑ The hidden state of a RNN can be used for semantic matching between texts.
- ❑ RNN for language generation:
  - ❑ Conditioned on textual or visual data, deep LSTMs have been shown to generate reasonable task specific text in tasks such as machine translation, image captioning and etc.



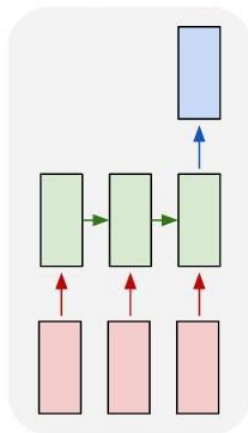
one to one



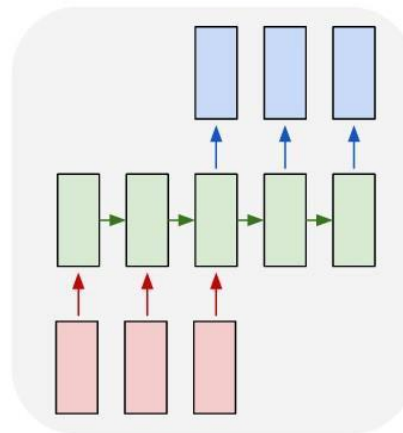
one to many



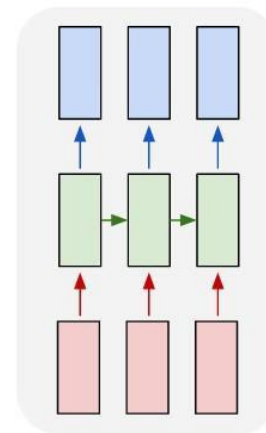
many to one



many to many



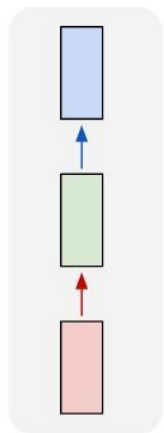
many to many



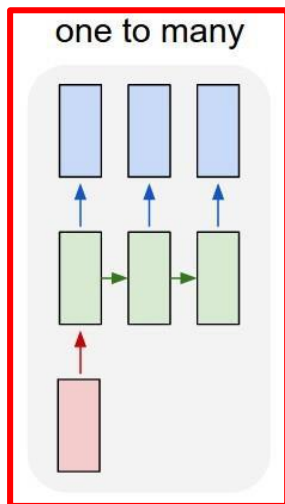




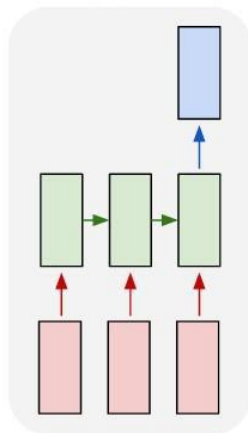
one to one



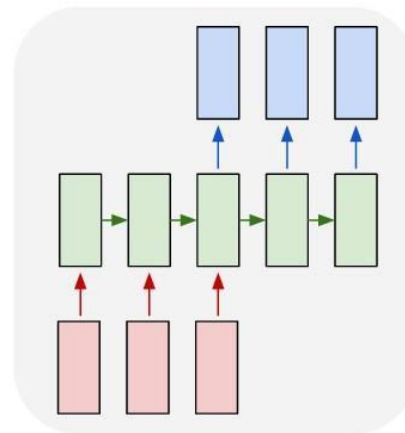
one to many



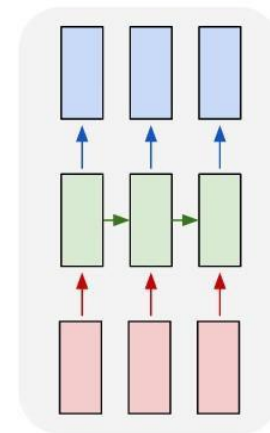
many to one



many to many



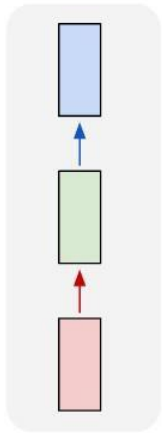
many to many



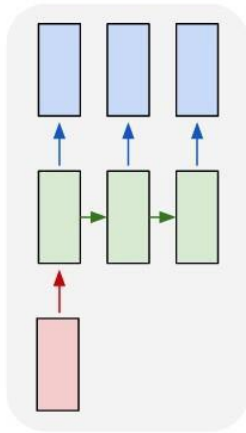
Automatically generate **caption** with the given image



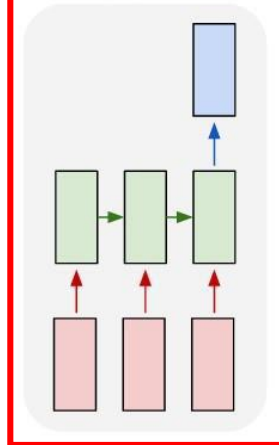
one to one



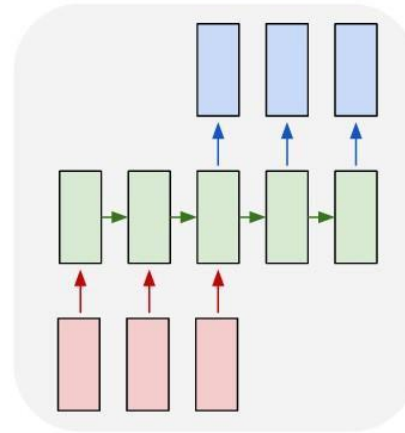
one to many



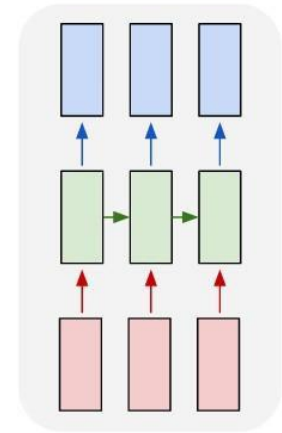
many to one



many to many



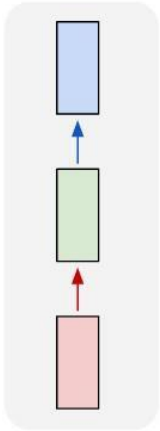
many to many



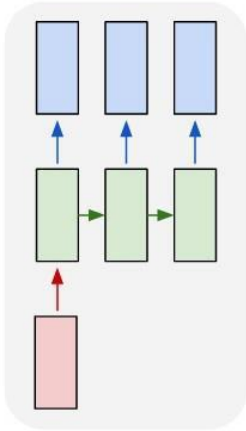
Predict whether a company would be **bankrupted**



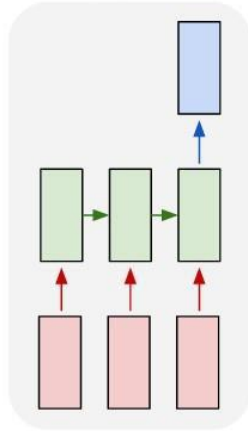
one to one



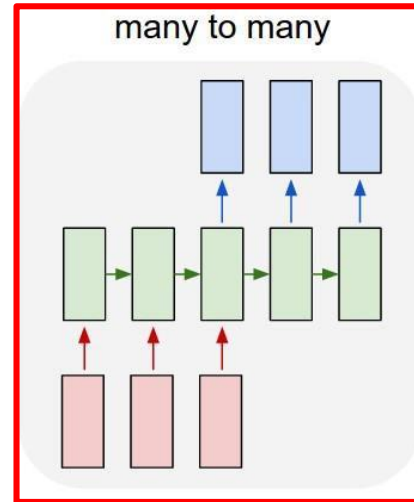
one to many



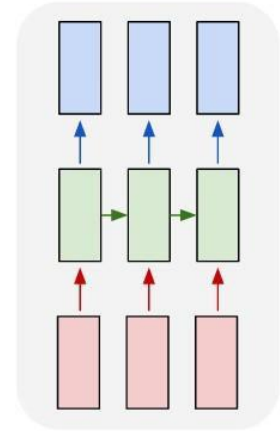
many to one



many to many



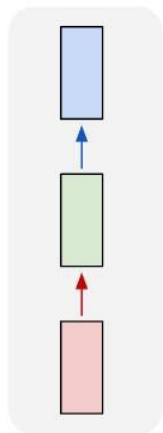
many to many



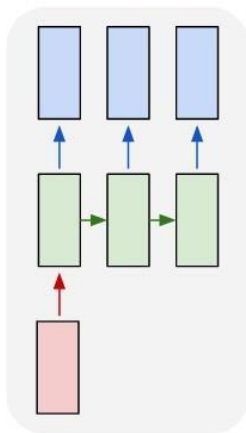
Translate one **sentence** into another language



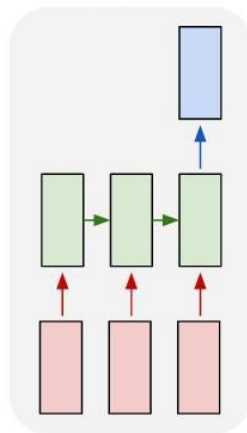
one to one



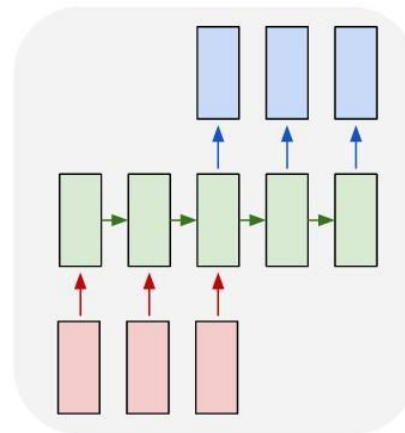
one to many



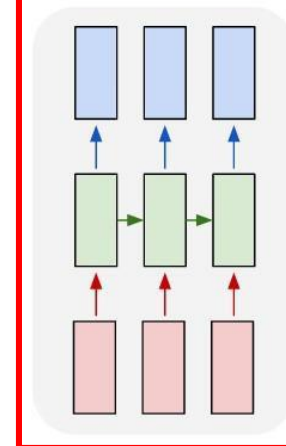
many to one



many to many



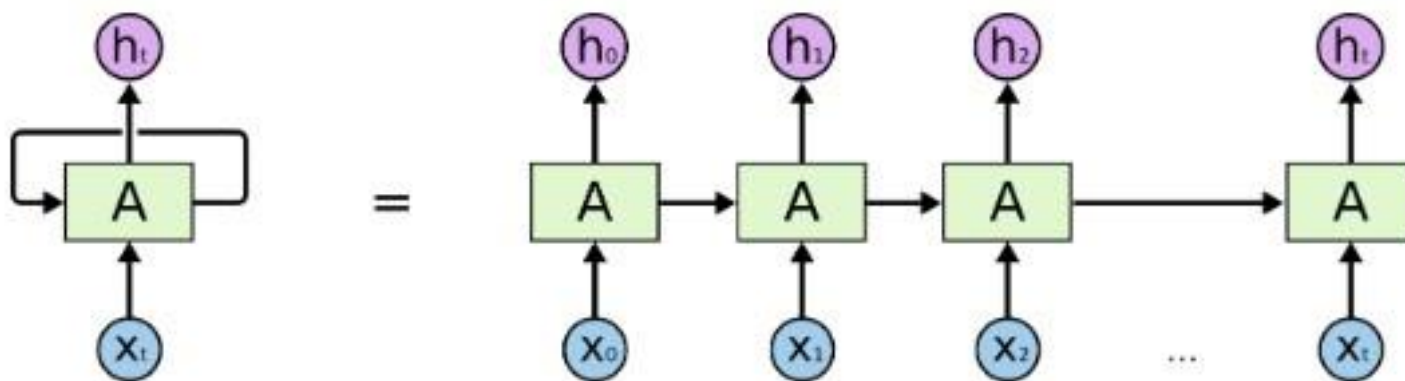
many to many



Classify whether the **word** is owns` name or not



# Recurrent Neural Network



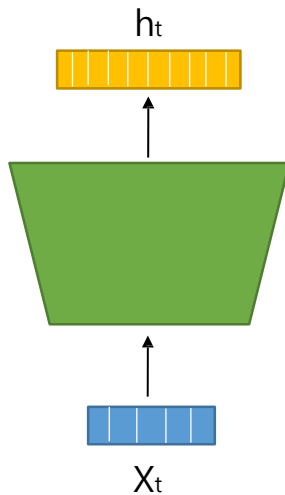
An unrolled recurrent neural network.



# Recurrent Neural Network



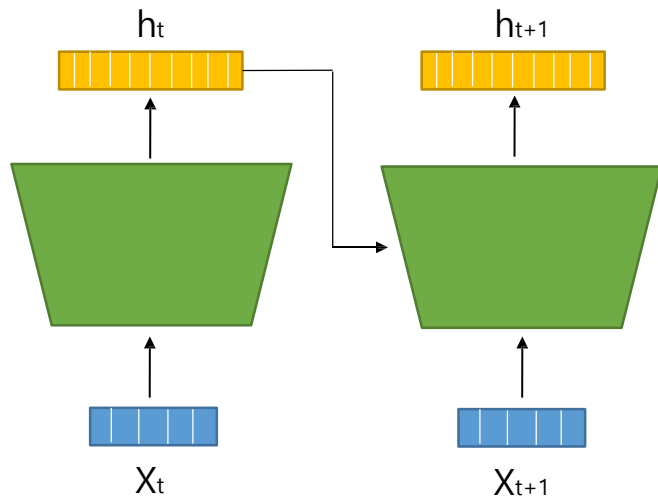
- Process both new inputs and model output of previous input!





# Recurrent Neural Network

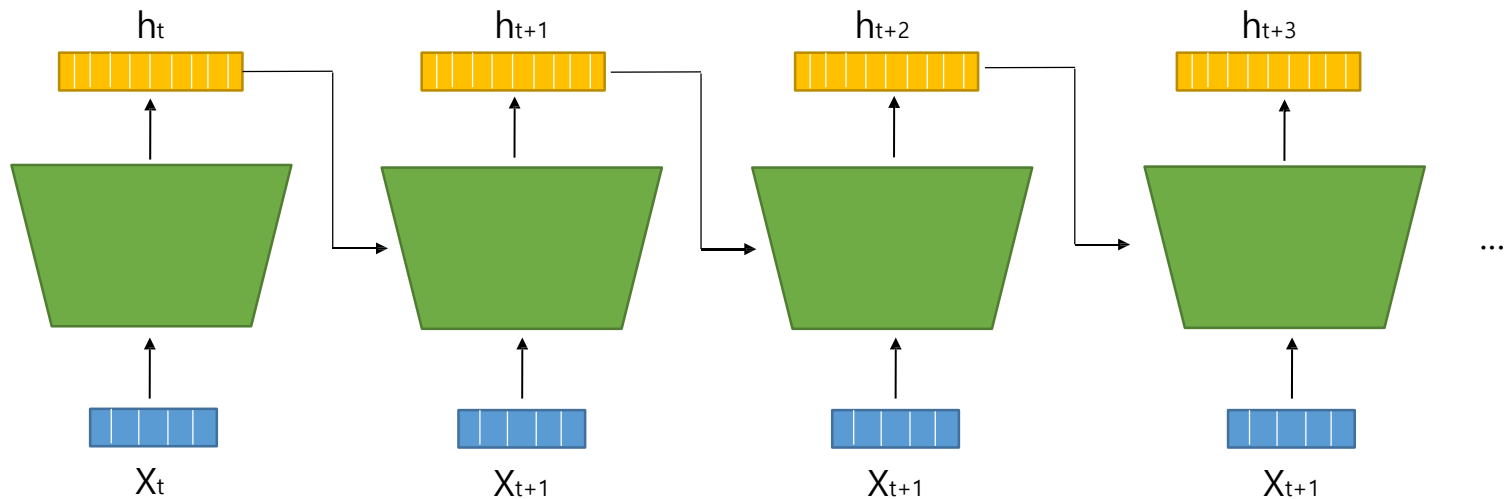
- Process both new inputs and model output of previous input!





# Recurrent Neural Network

- Process both new inputs and model output of previous input!

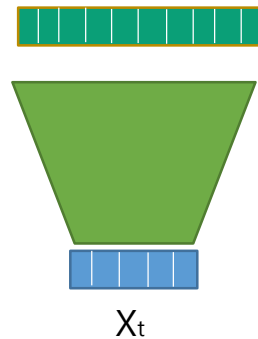






# Recurrent Neural Network

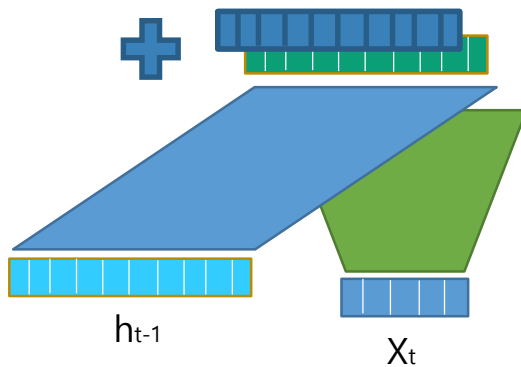
- But exactly, how can we combine new input and previous output?





# Recurrent Neural Network

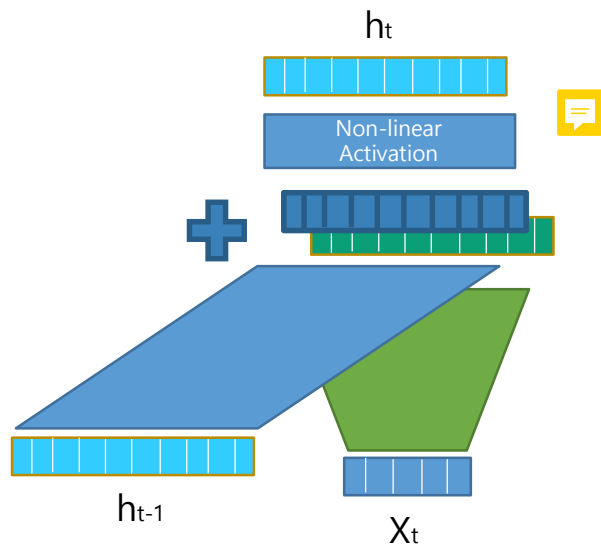
- But exactly, how can we combine new input and previous output?





# Recurrent Neural Network

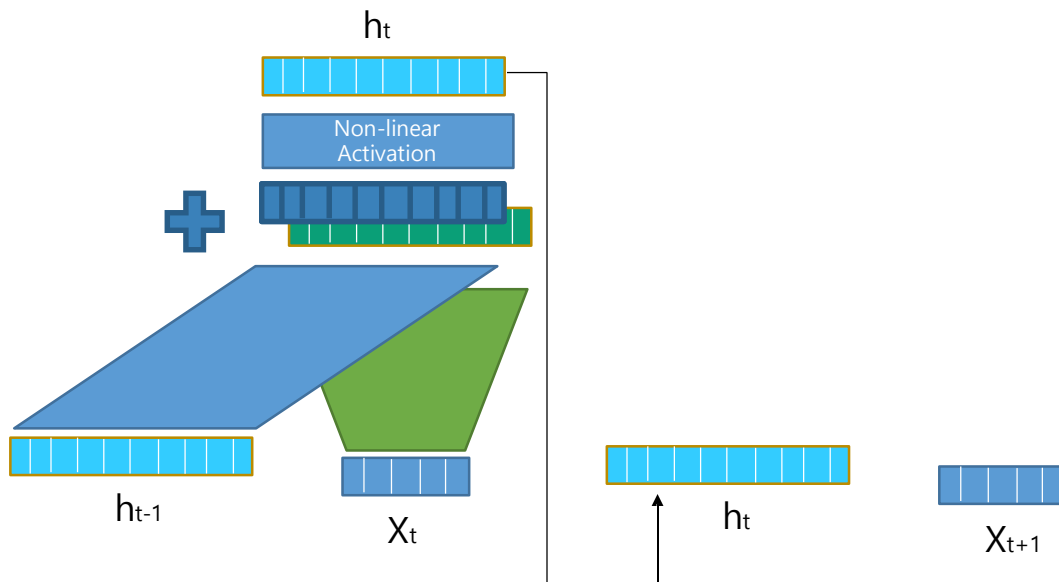
- But exactly, how can we combine new input and previous output?





# Recurrent Neural Network

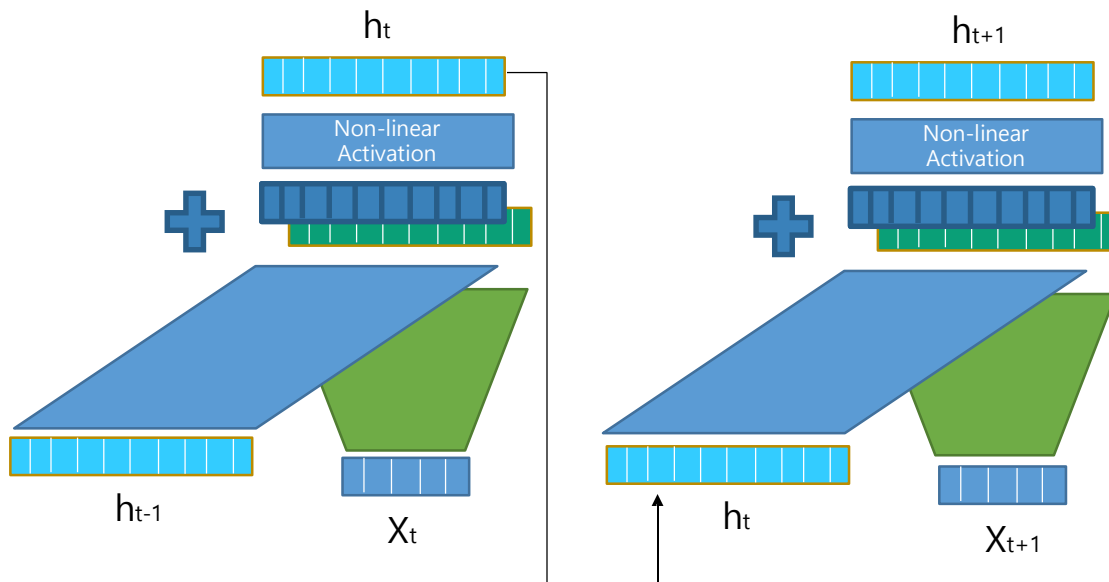
- But exactly, how can we combine new input and previous output?





# Recurrent Neural Network

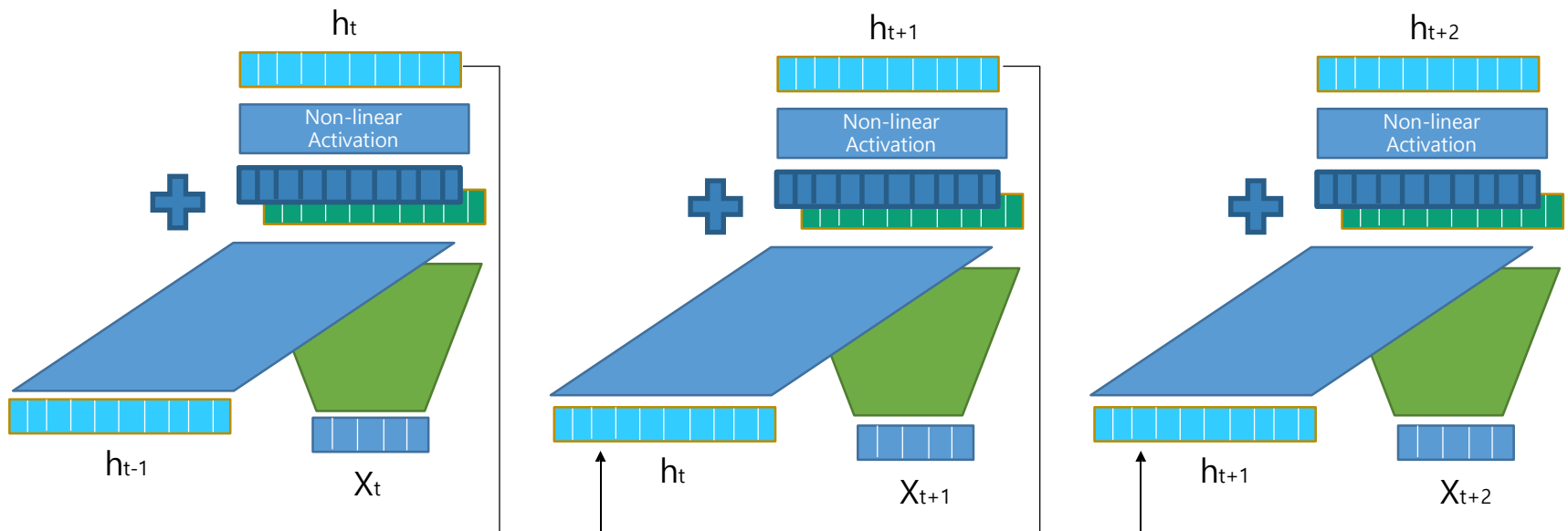
- But exactly, how can we combine new input and previous output?





# Recurrent Neural Network

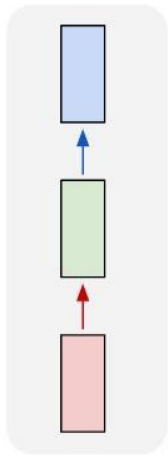
- But exactly, how can we combine new input and previous output?



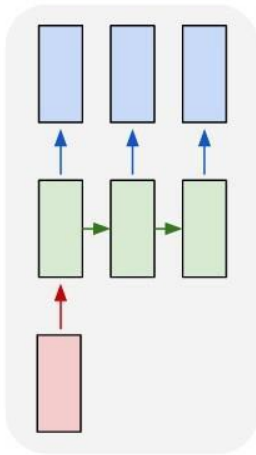


# Types of Task Dealing with Sequential Data

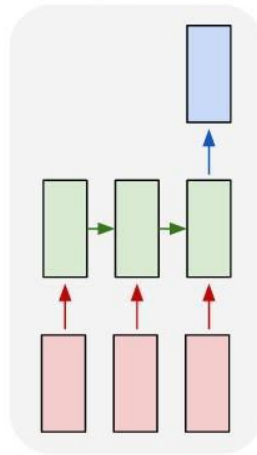
one to one



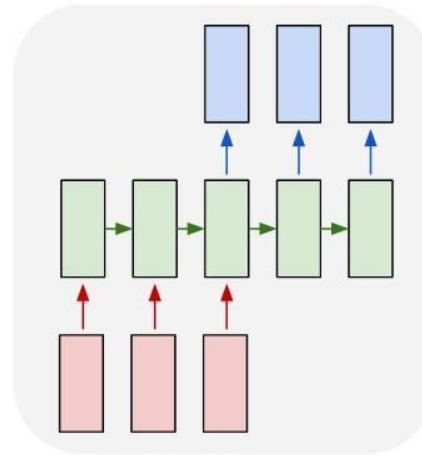
one to many



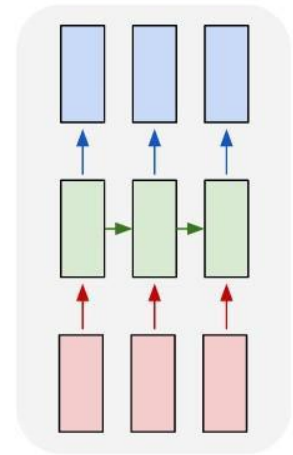
many to one



many to many



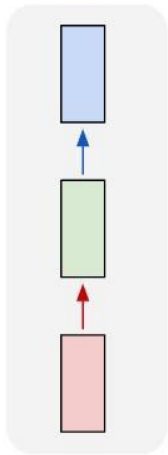
many to many



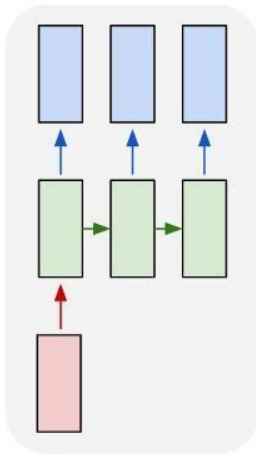


# Types of Task Dealing with Sequential Data

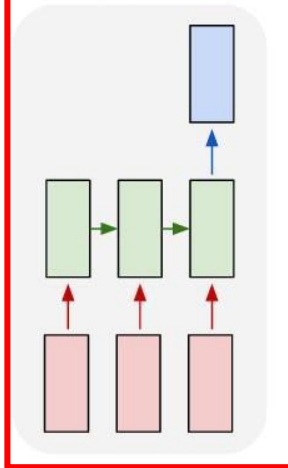
one to one



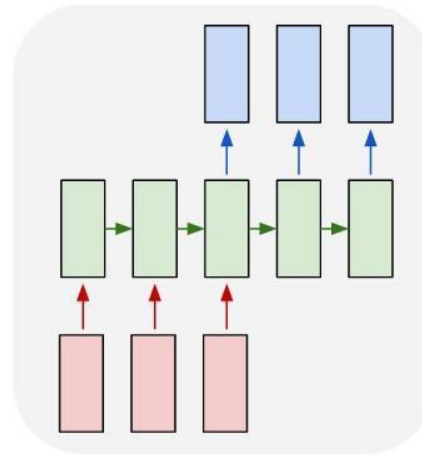
one to many



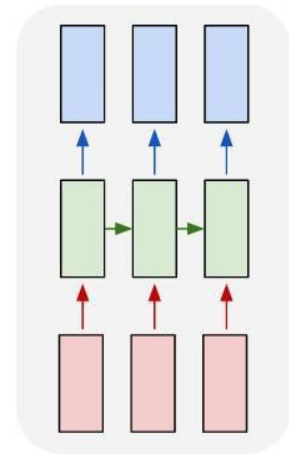
many to one



many to many



many to many

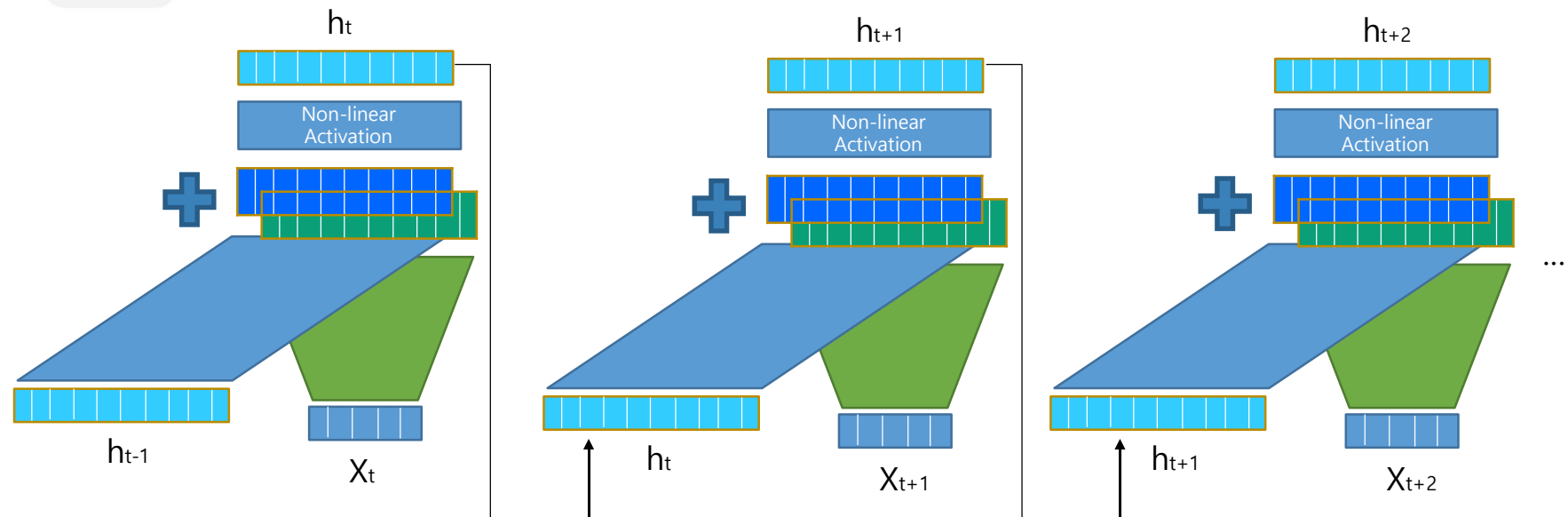
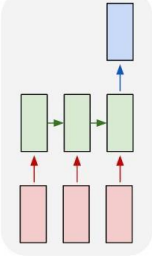






# Recurrent Neural Network

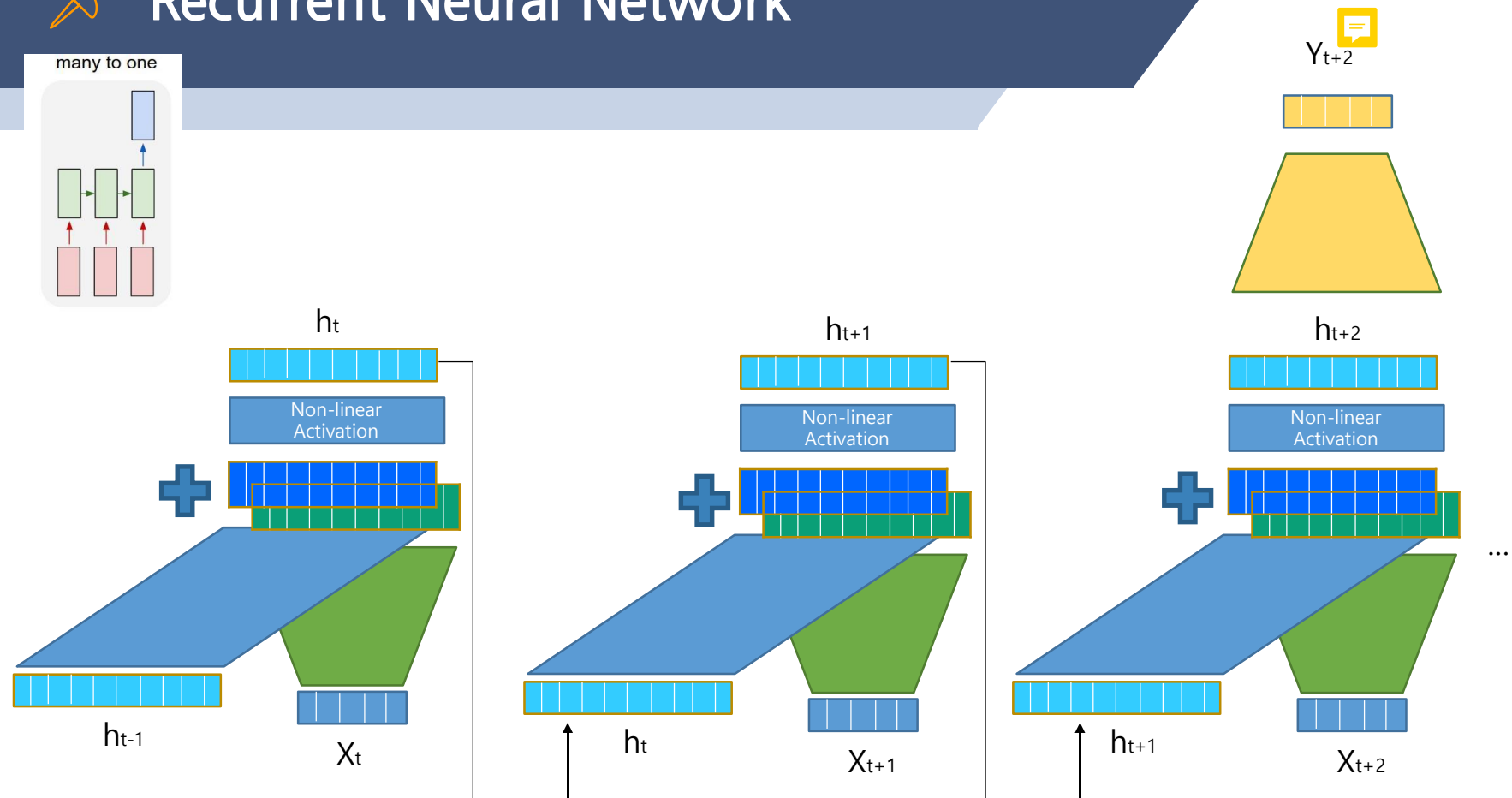
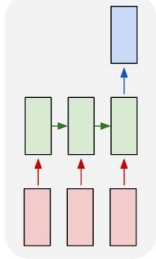
many to one





# Recurrent Neural Network

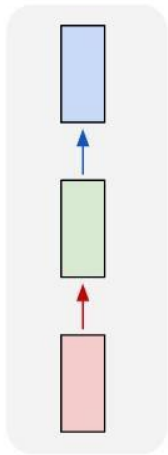
many to one



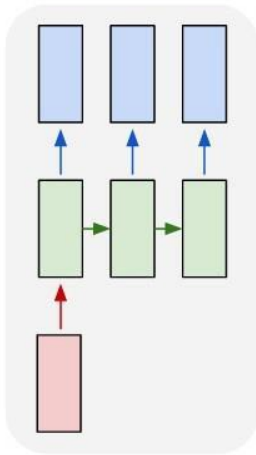


# Types of Task Dealing with Sequential Data

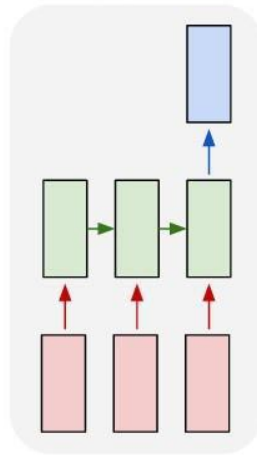
one to one



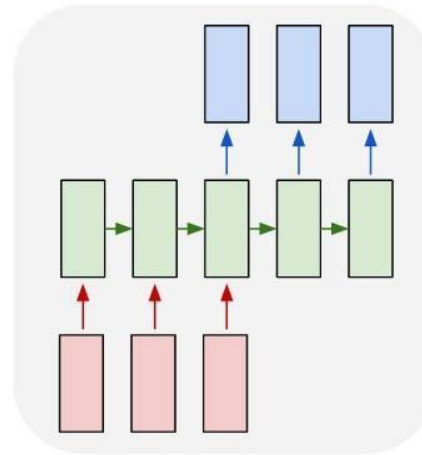
one to many



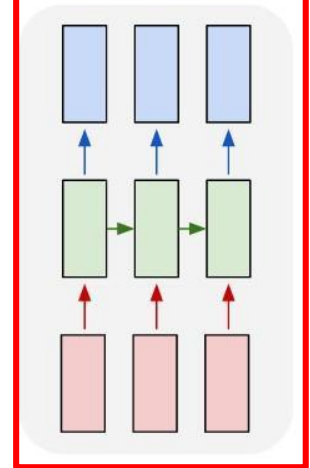
many to one



many to many



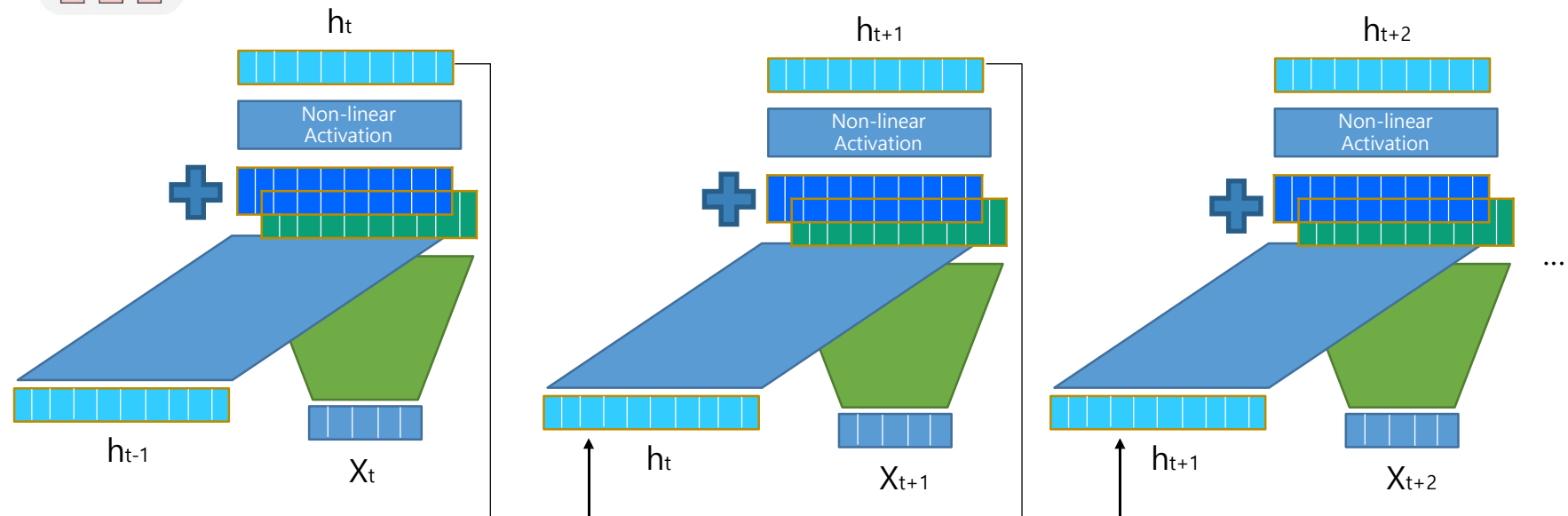
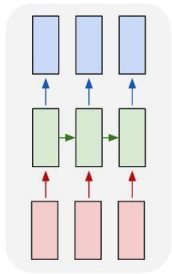
many to many





# Recurrent Neural Network

many to many

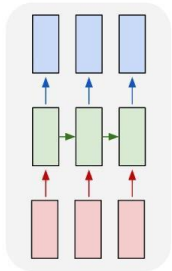




# Recurrent Neural Network



many to many



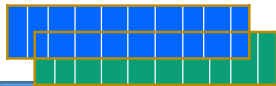
$Y_t$



$h_t$



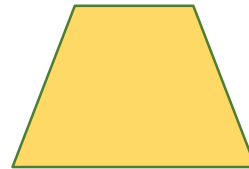
Non-linear  
Activation



$h_{t-1}$

$x_t$

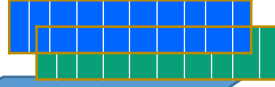
$Y_{t+1}$



$h_{t+1}$



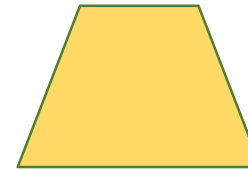
Non-linear  
Activation



$h_t$

$x_{t+1}$

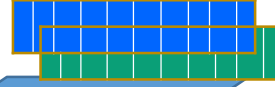
$Y_{t+2}$



$h_{t+2}$



Non-linear  
Activation



$h_{t+1}$

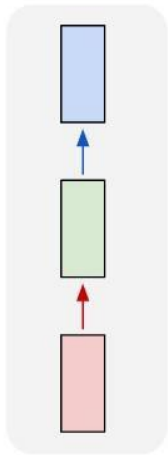
$x_{t+2}$

...

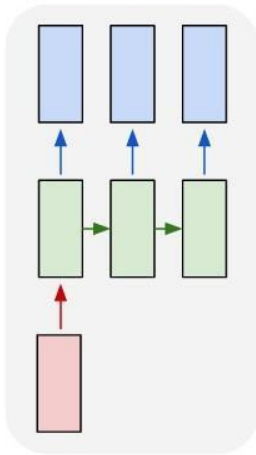


# Types of Task Dealing with Sequential Data

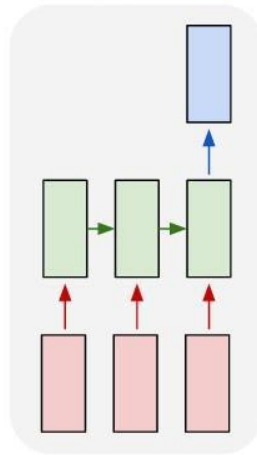
one to one



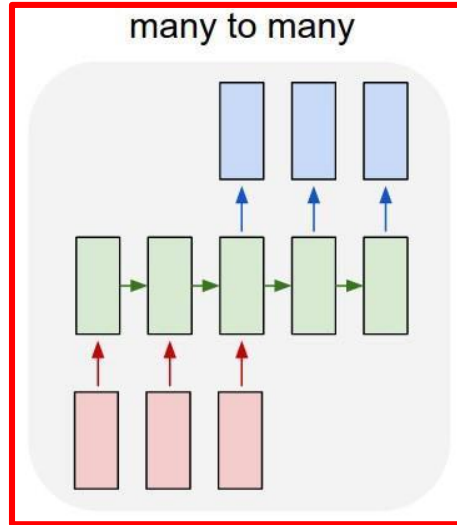
one to many



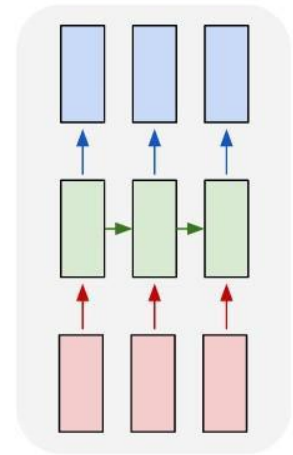
many to one



many to many



many to many

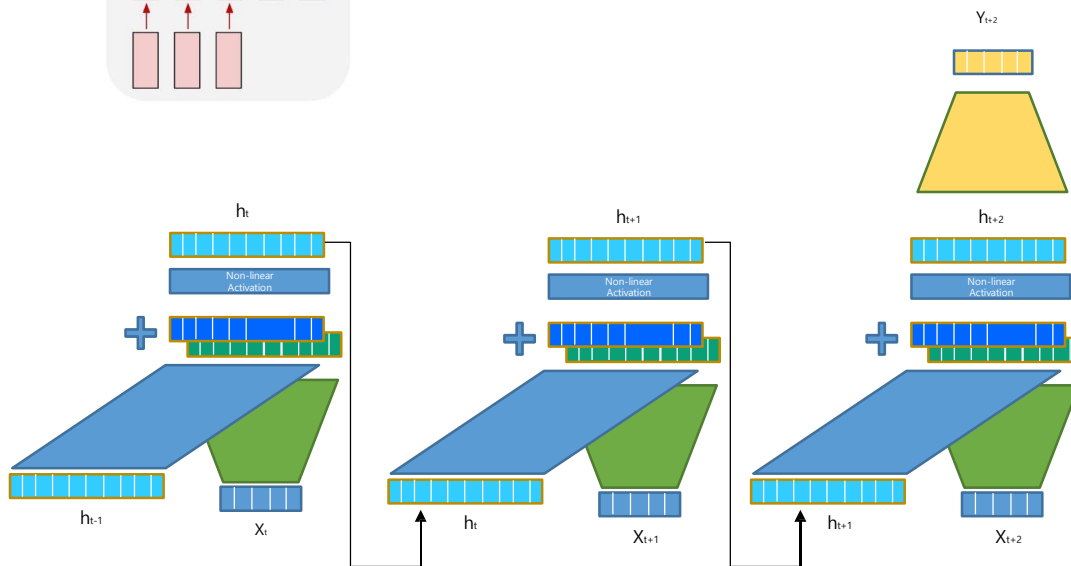
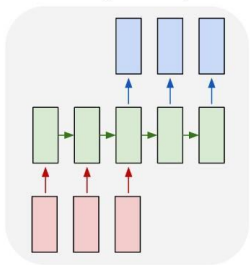






# Recurrent Neural Network

many to many

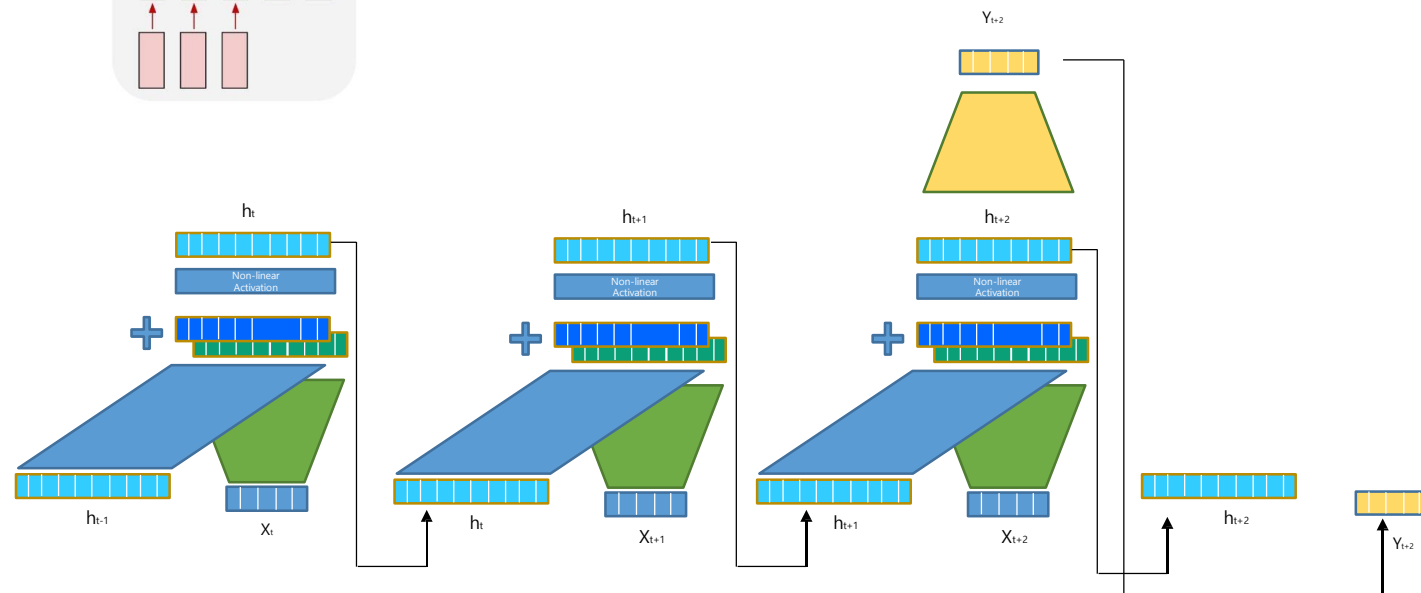
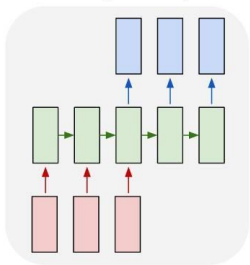






# Recurrent Neural Network

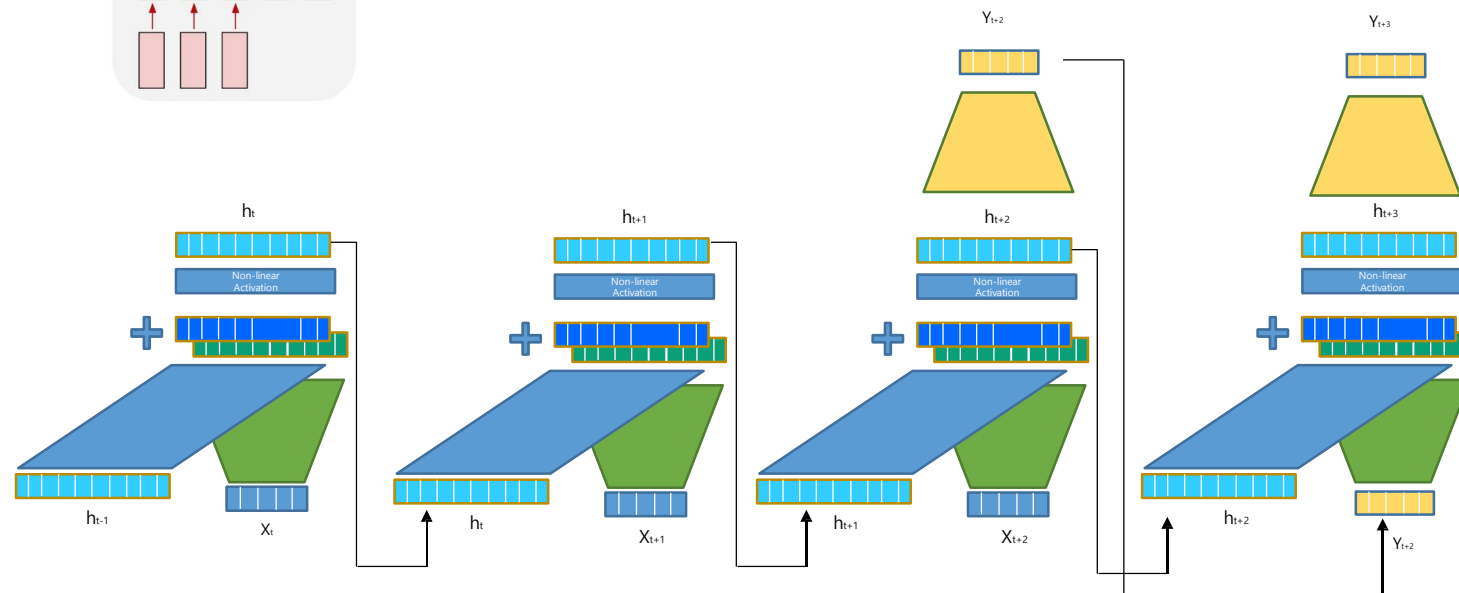
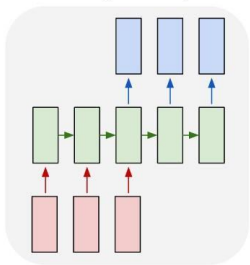
many to many





# Recurrent Neural Network

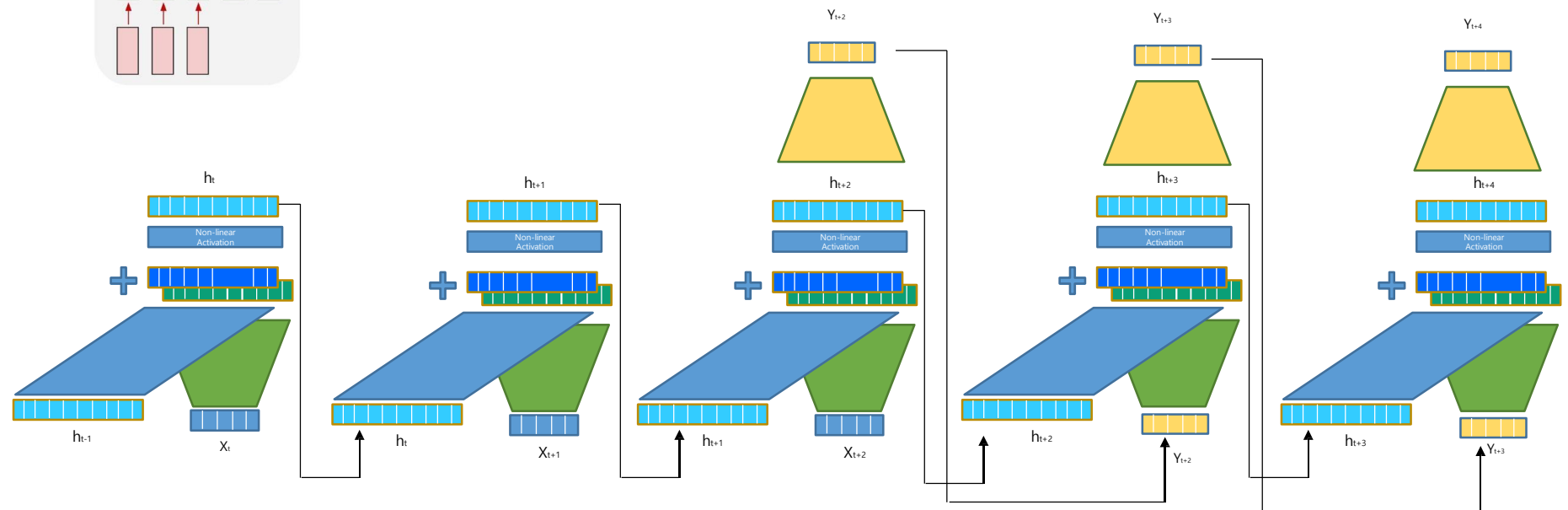
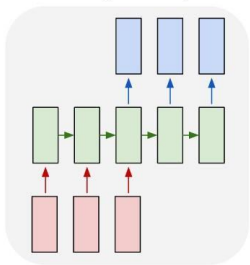
many to many





# Recurrent Neural Network

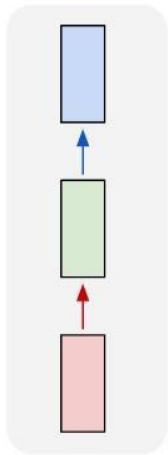
many to many



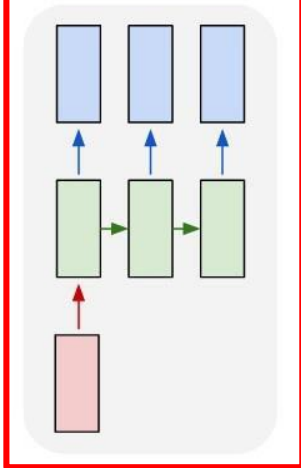


# Types of Task Dealing with Sequential Data

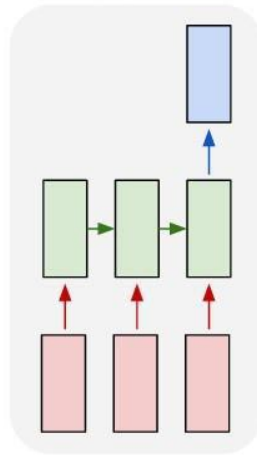
one to one



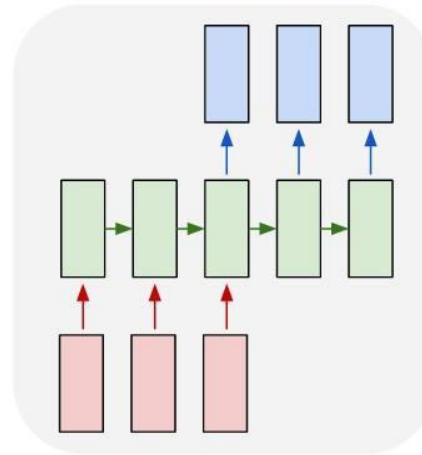
one to many



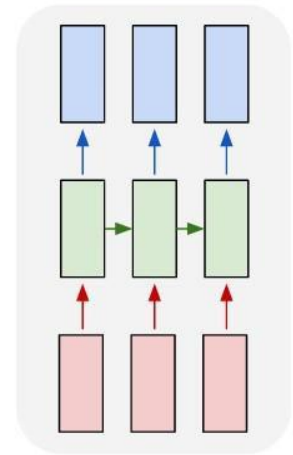
many to one



many to many



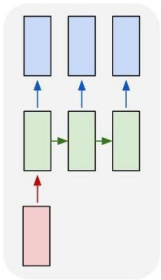
many to many





# Recurrent Neural Network

one to many



$h_{t+1}$

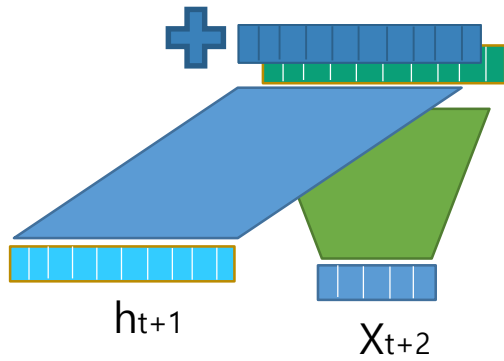
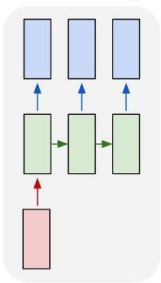


$x_{t+2}$



# Recurrent Neural Network

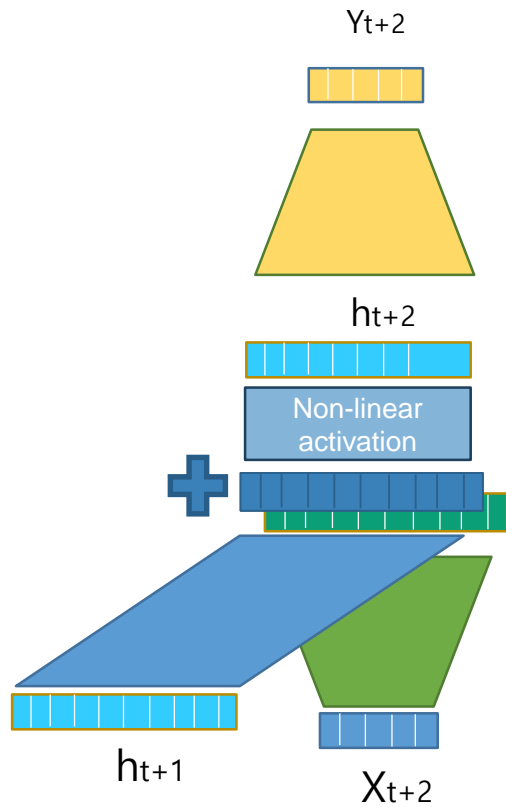
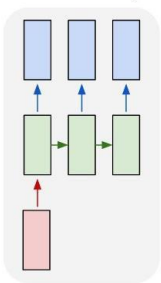
one to many





# Recurrent Neural Network

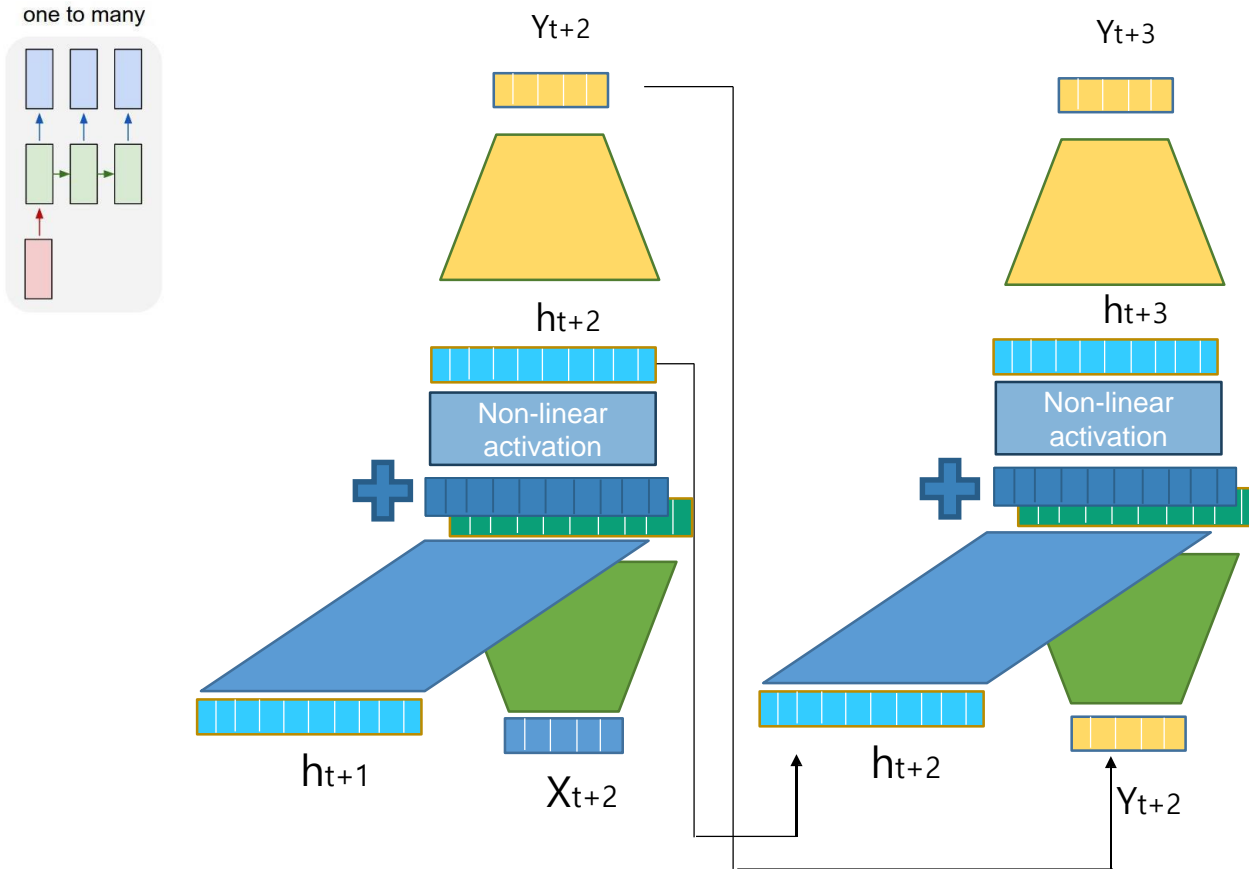
one to many





# Recurrent Neural Network

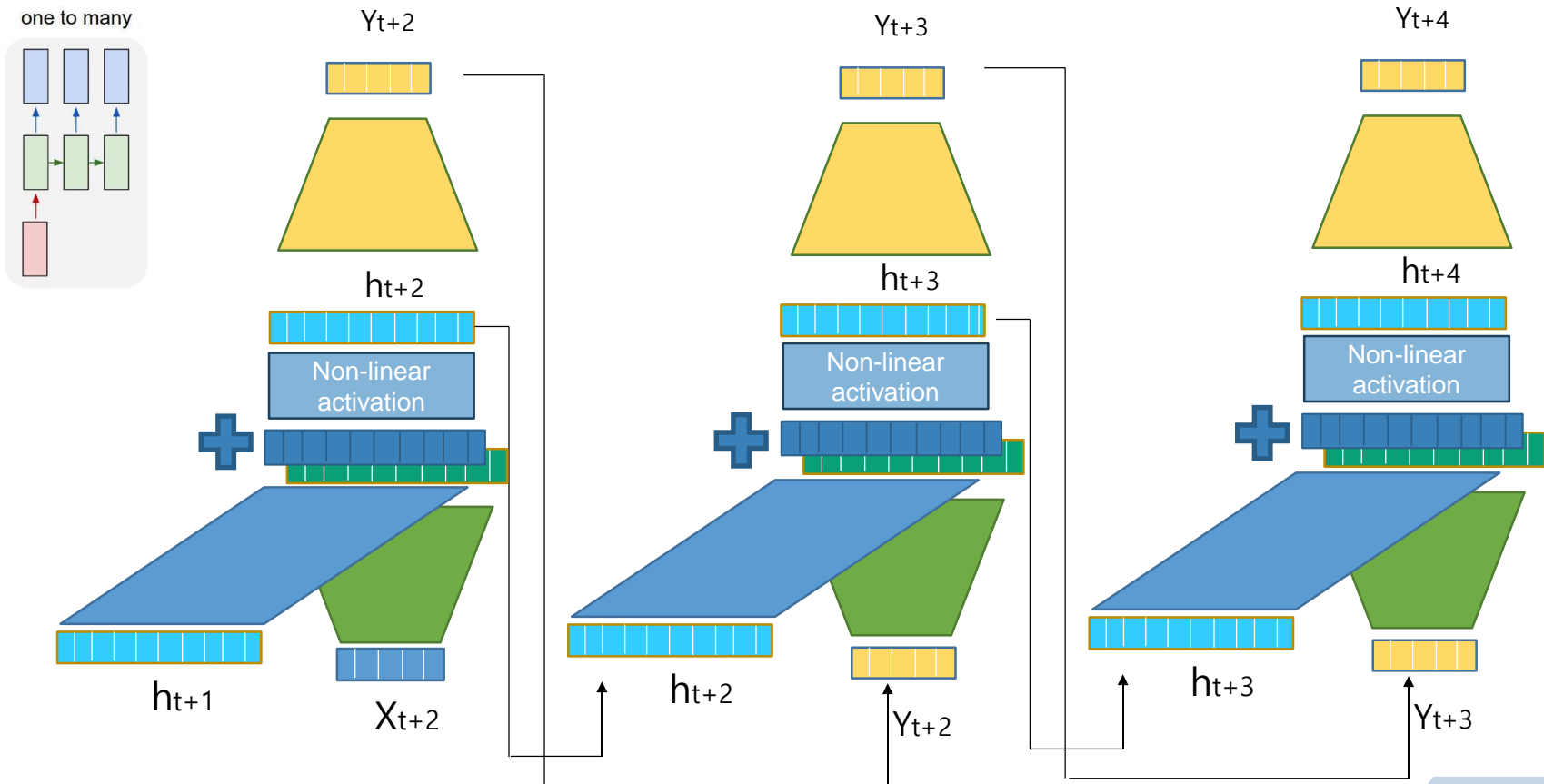
one to many







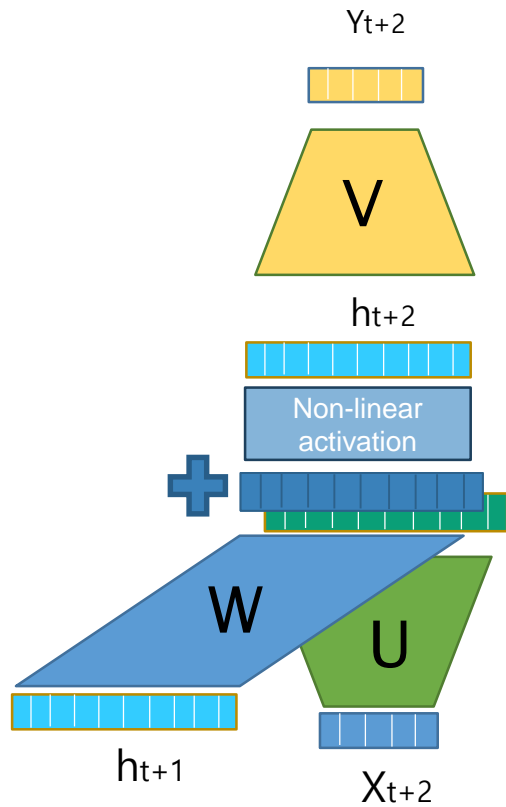
# Recurrent Neural Network



# Recurrent Neural Network with Math

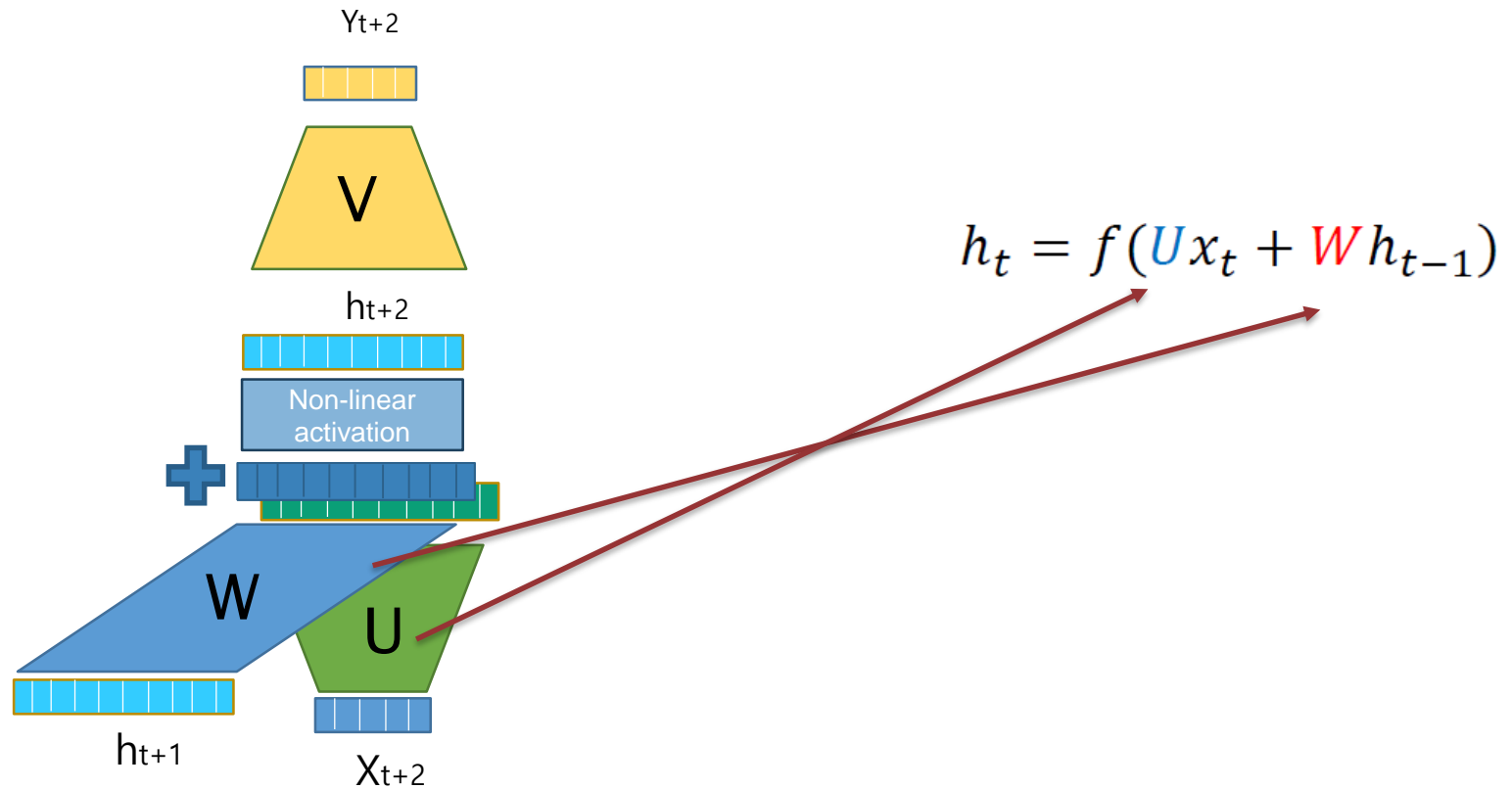


# Recurrent Neural Network with Math



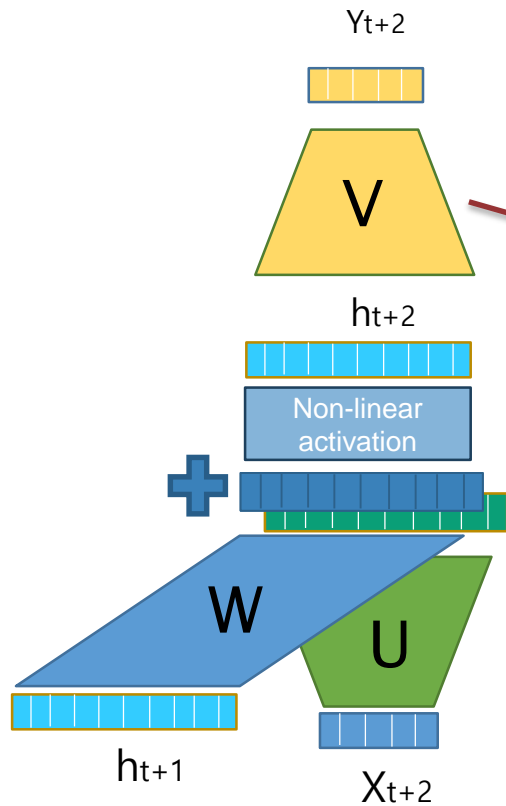


# Recurrent Neural Network with Math





# Recurrent Neural Network with Math

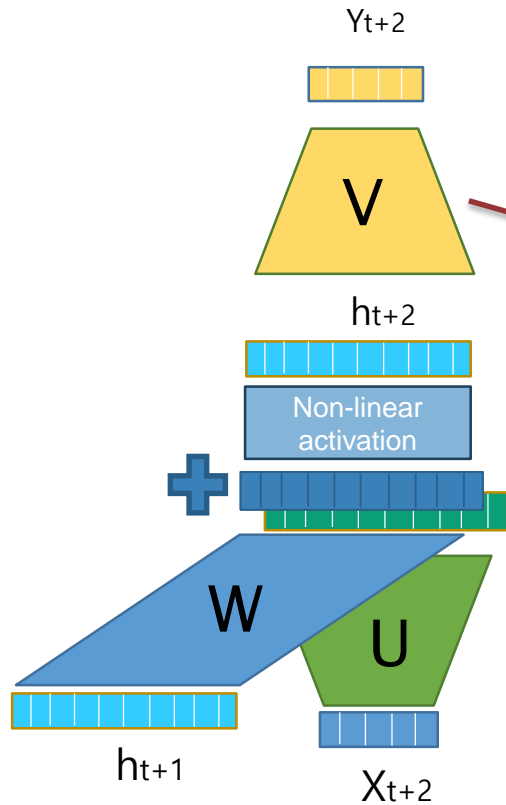


$$h_t = f(Ux_t + Wh_{t-1})$$

$$y_t = f(Vh_t)$$



# Recurrent Neural Network with Math



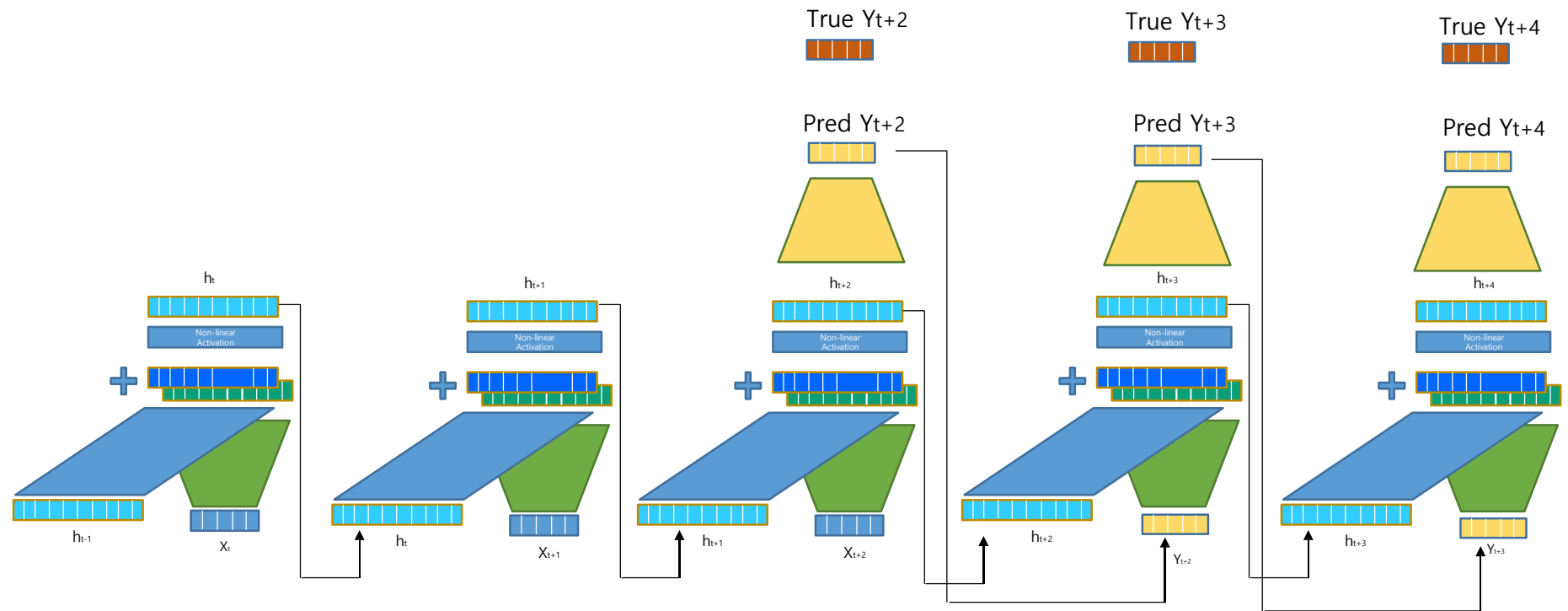
$$h_t = f(Ux_t + Wh_{t-1})$$
$$y_t = f(Vh_t)$$

$f(x) = \tanh(x)$

$f(x) = x$



# Calculate Loss of Recurrent Neural Network





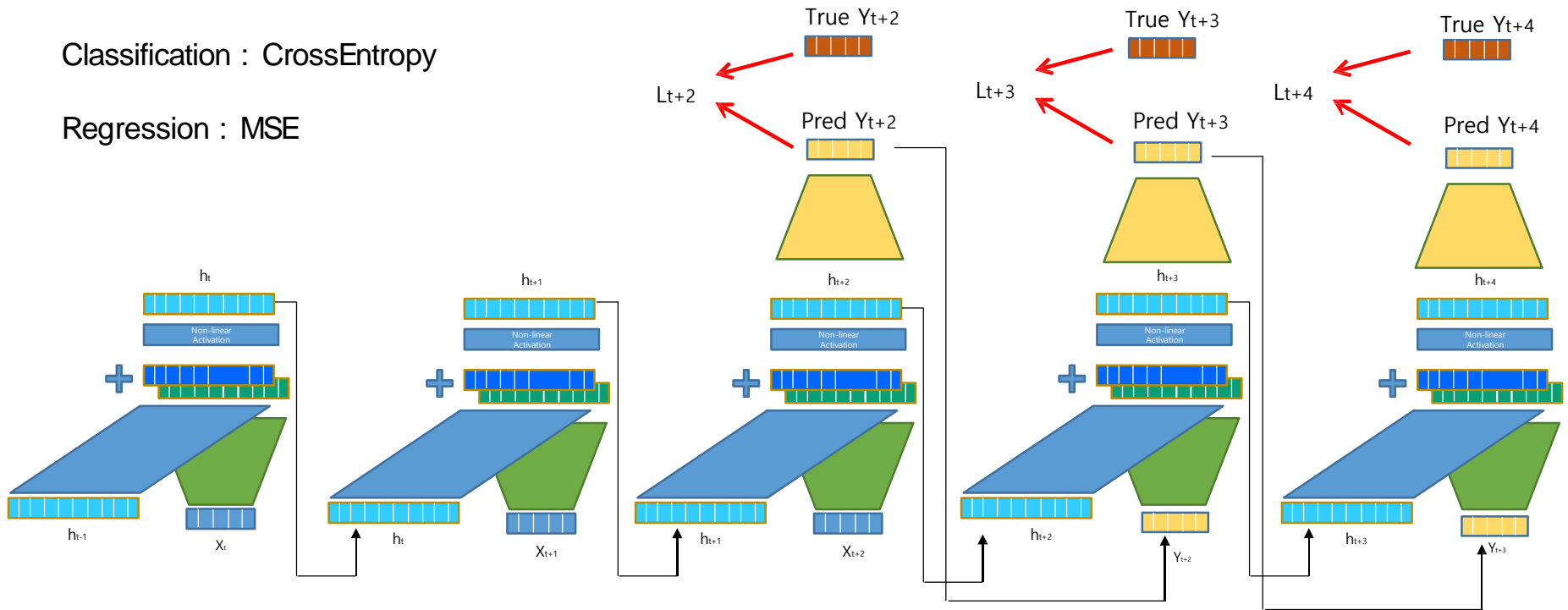
# Calculate Loss of Recurrent Neural Network



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

Classification : CrossEntropy

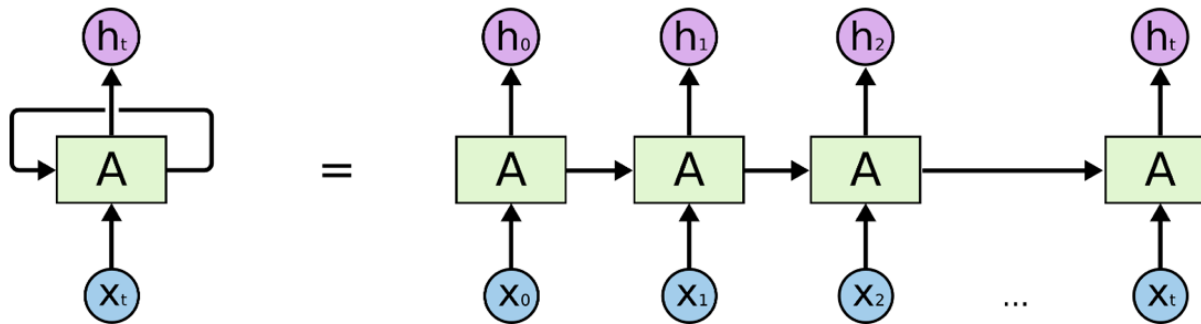
Regression : MSE







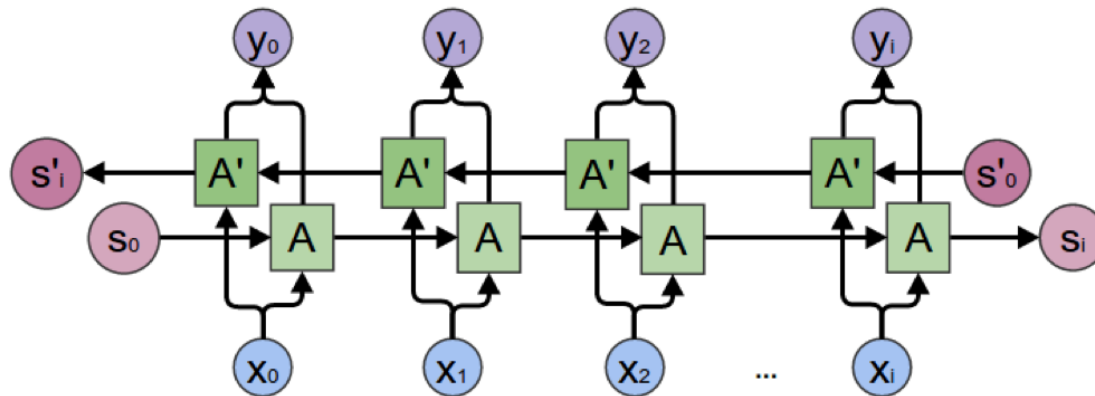
# Recurrent Neural Network (RNN)



- ❑ A kind of neural network with cyclic structure
- ❑ A circular structure results in a state, which can handle sequences of varying lengths.
- ❑ Output  $h_t$ , reflecting the input value  $[x_0, x_1, \dots, x_{t-1}]$



# Bidirectional RNN

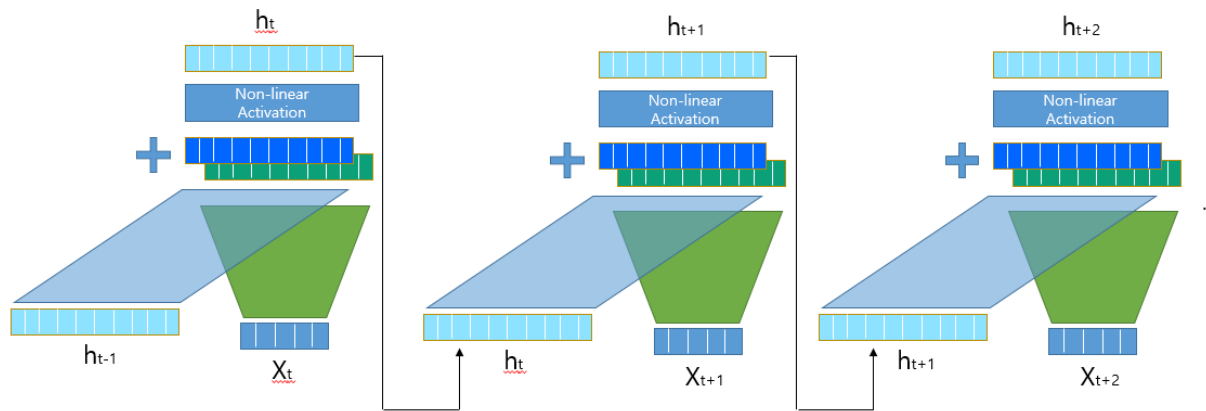


- ❑ RNNs that combine two RNNs in different directions to enable bidirectional dependence
- ❑ Output  $y_t$  has input  $[x_0, x_1, \dots, x_{t-1}]$  and  $[x_{t+1}, x_{t+2}, \dots, x_N]$  is reflected

# Vanishing Gradient Problem



# Vanishing Gradient Problem



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

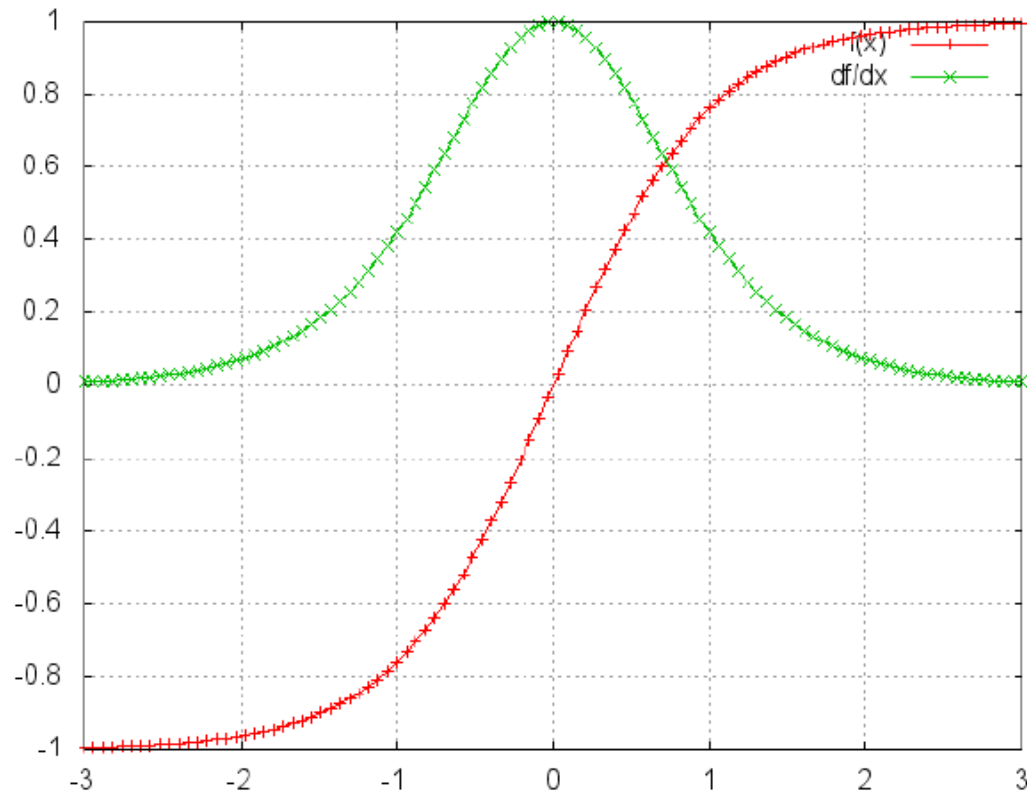
$$h_{t-1} = \tanh(W[h_{t-2}, x_{t-1}])$$

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

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



# Vanishing Gradient Problem



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



# Vanishing Gradient Problem



With long sequence, gradient could be vanished

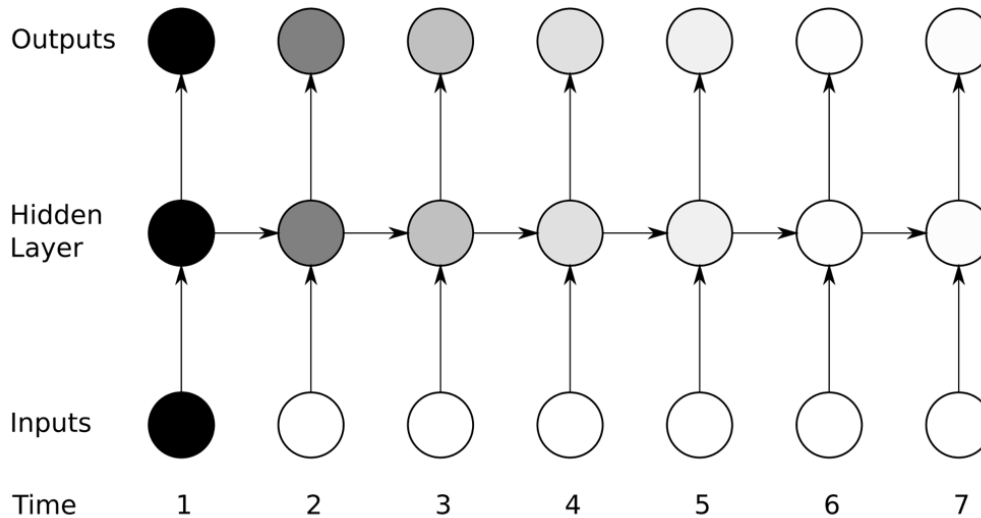
Back- propagation could not be done properly

Vanilla RNN is weak to learn long sequence

# Long Short Term Memory Network



# Long-term Dependency Problem

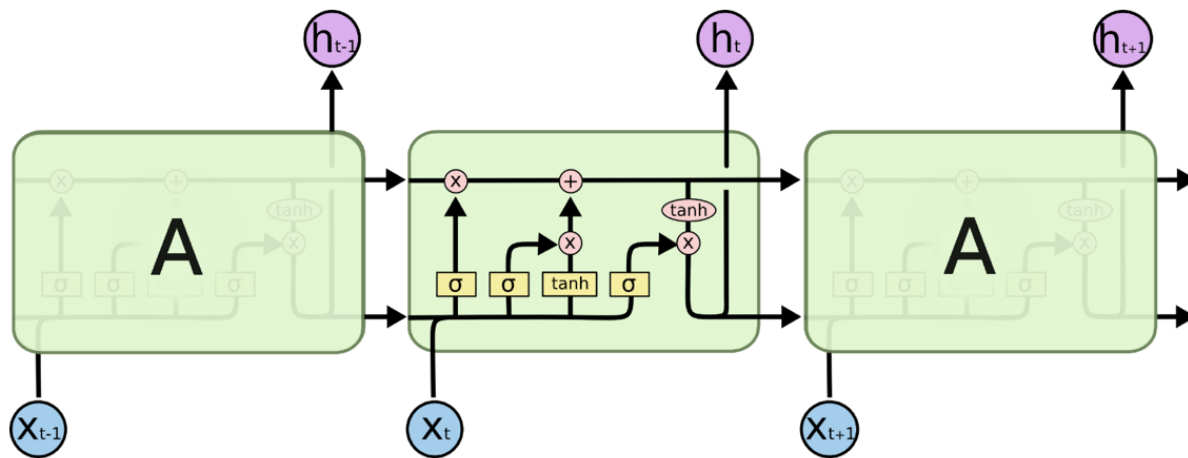


- ❑ The basic RNN is diluted with the state value as the step progresses, making it difficult to reflect long-distance dependency





# Long Short-Term Memory (LSTM)



- ❑ An improved RNN structure to reflect long-distance dependency by adding a gate to control the amount of information transfer without reflecting the input value unconditionally to the state

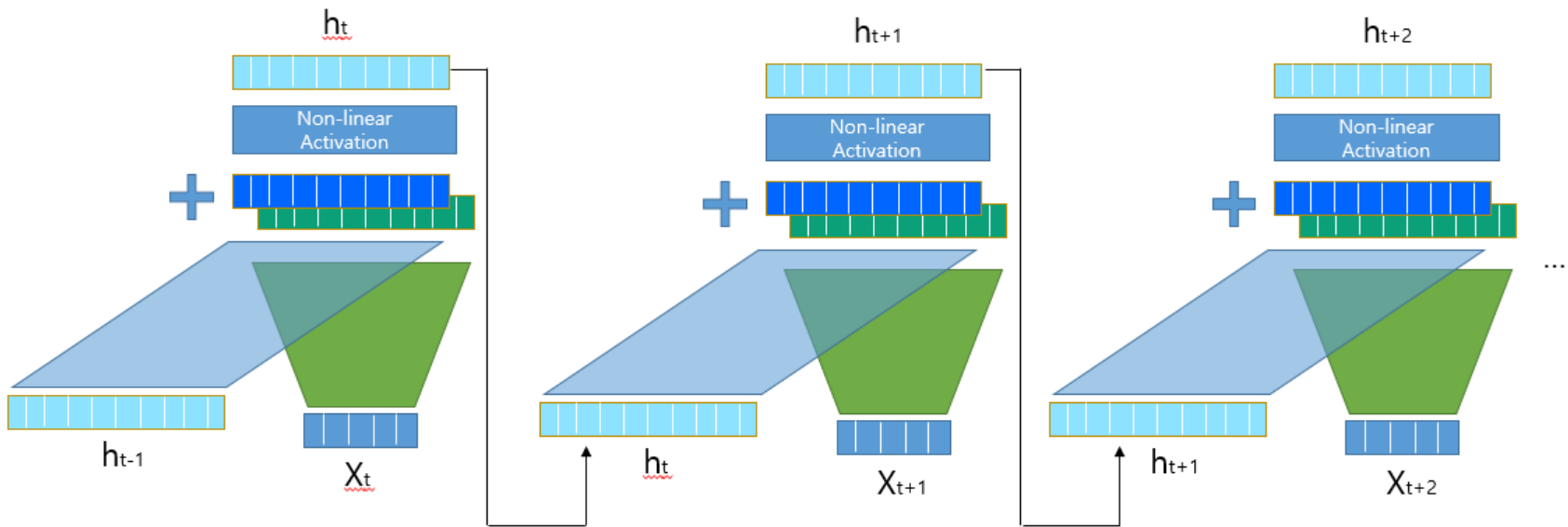


# Long Short-Term Memory (LSTM)

$$\begin{aligned} (0,1) \left\{ \begin{aligned} i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) && \text{Input gate} \\ f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) && \text{Forget gate} \text{🗨️} \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) && \text{Output gate} \end{aligned} \right. \\ l_t &= \tanh(W_l[h_{t-1}, x_t] + b_l) && \text{New input} \\ \tilde{h}_t &= f_t \cdot \tilde{h}_{t-1} + i_t \cdot l_t && \text{Hidden state update} \\ h_t^s &= o_t \cdot \tilde{h}_t && \text{New output} \end{aligned}$$

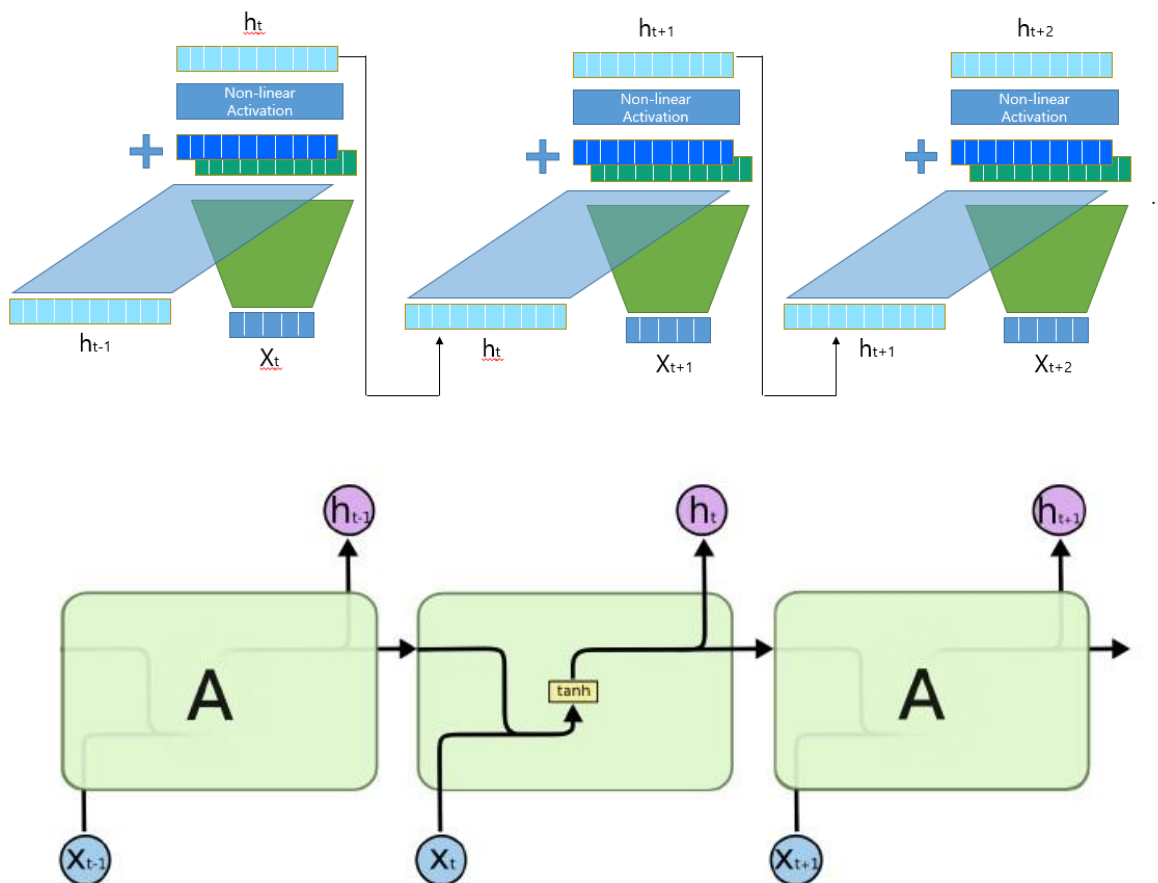


# Long Short Term Memory Network





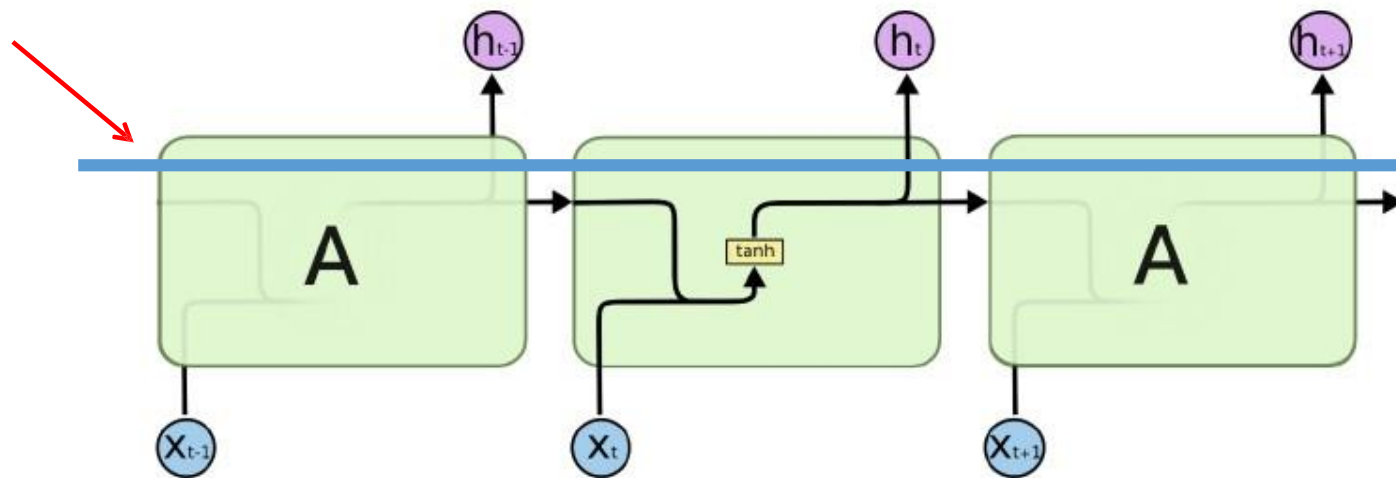
# Long Short Term Memory Network





# Long Short Term Memory Network

Add Information Flow?



Standard RNN

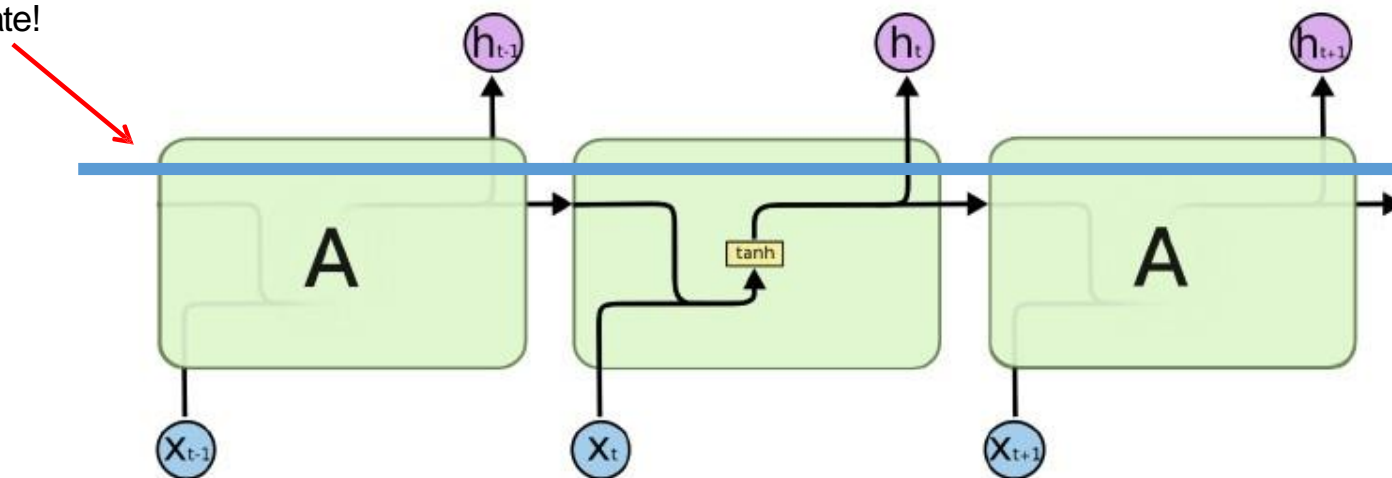


# Long Short Term Memory Network



Add Information Flow?

Cell State!



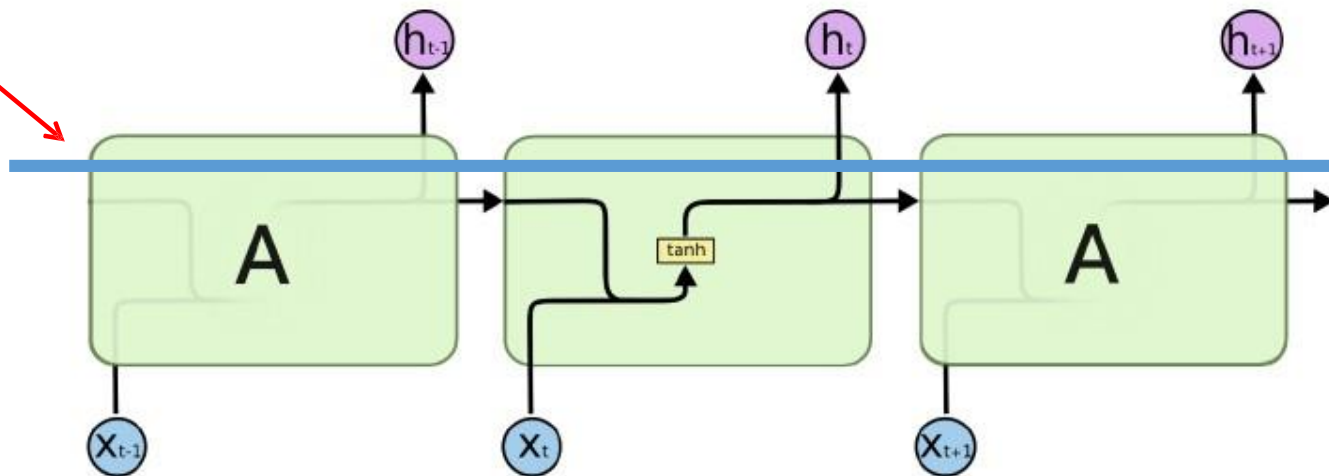
Standard RNN



# Long Short Term Memory Network

Add Information Flow?

Cell State!



남길 건 남기고, 잊어버릴 건 잊어버리고, 새로 추가할 건 추가해서  
Cell State에 중요한 정보만 계속 흘러가도록!

Hidden State는 Cell State를 적당히 가공해서 내보내자!


어떻게? → Using Gate!



# Long Short Term Memory Network

Gate - element wise coefficient multiplication

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

x 	<table><tr><td>0.01</td><td>0.03</td><td>-0.03</td><td>0.19</td><td>0.49</td><td>0.18</td><td>-0.19</td><td>1.8</td><td>-0.11</td><td>0.0</td></tr><tr><td></td><td></td><td></td><td>8</td><td></td><td></td><td></td><td></td><td></td><td>3</td></tr></table>										0.01	0.03	-0.03	0.19	0.49	0.18	-0.19	1.8	-0.11	0.0				8						3	$G \cdot C_{t-1}$
	0.01	0.03	-0.03	0.19	0.49	0.18	-0.19	1.8	-0.11	0.0																					
				8						3																					
<table><tr><td>0.1</td><td>0.1</td><td>0.3</td><td>0.99</td><td>0.98</td><td>0.2</td><td>0.1</td><td>0.9</td><td>0.1</td><td>0.1</td></tr></table>										0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1	$G$											
0.1	0.1	0.3	0.99	0.98	0.2	0.1	0.9	0.1	0.1																						
	<table><tr><td>0.1</td><td>0.3</td><td>-0.1</td><td>0.2</td><td>0.5</td><td>0.9</td><td>-1.9</td><td>2.0</td><td>-1.1</td><td>0.3</td></tr></table>										0.1	0.3	-0.1	0.2	0.5	0.9	-1.9	2.0	-1.1	0.3	$C_{t-1}$										
0.1	0.3	-0.1	0.2	0.5	0.9	-1.9	2.0	-1.1	0.3																						

How to judge value of each gate coefficient?

→ Using small non- linear layer!

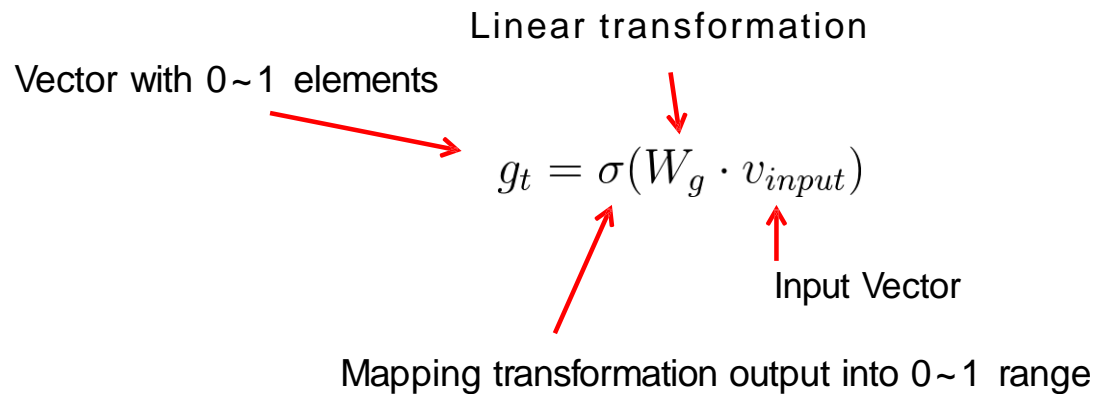




# Long Short Term Memory Network

Gate - element wise coefficient multiplication

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

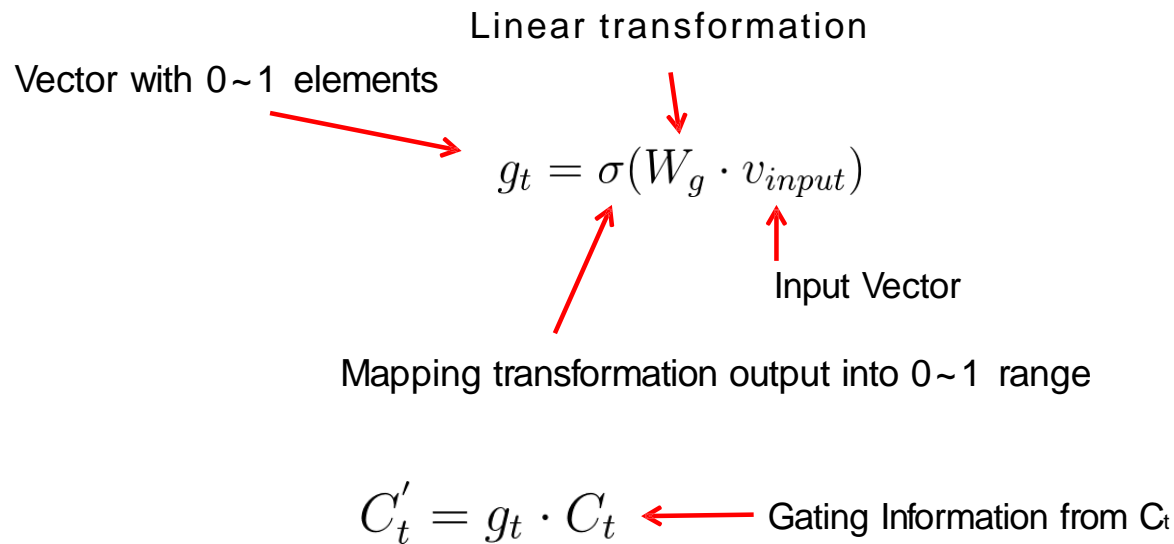




# Long Short Term Memory Network

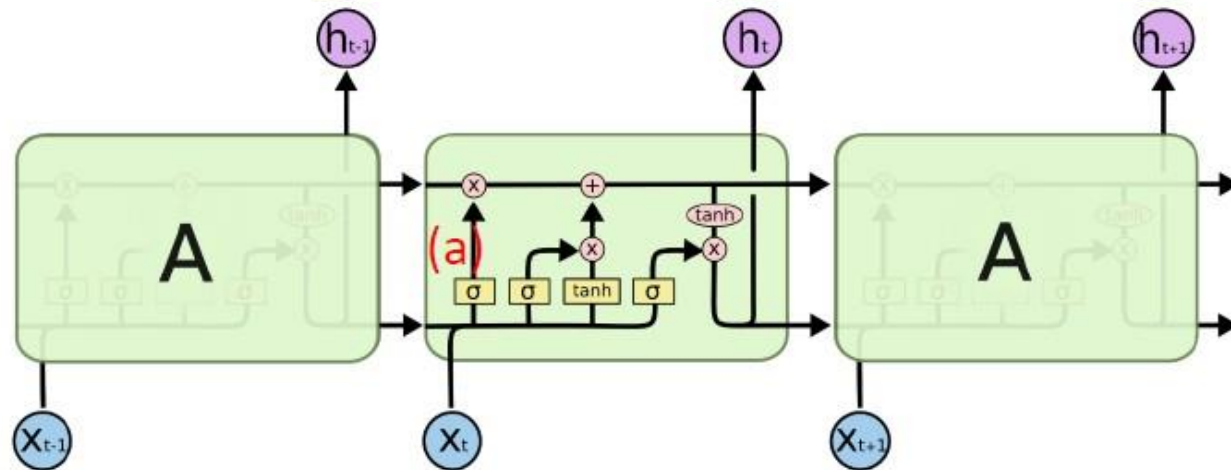
Gate - element wise coefficient multiplication

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





# Long Short Term Memory Network

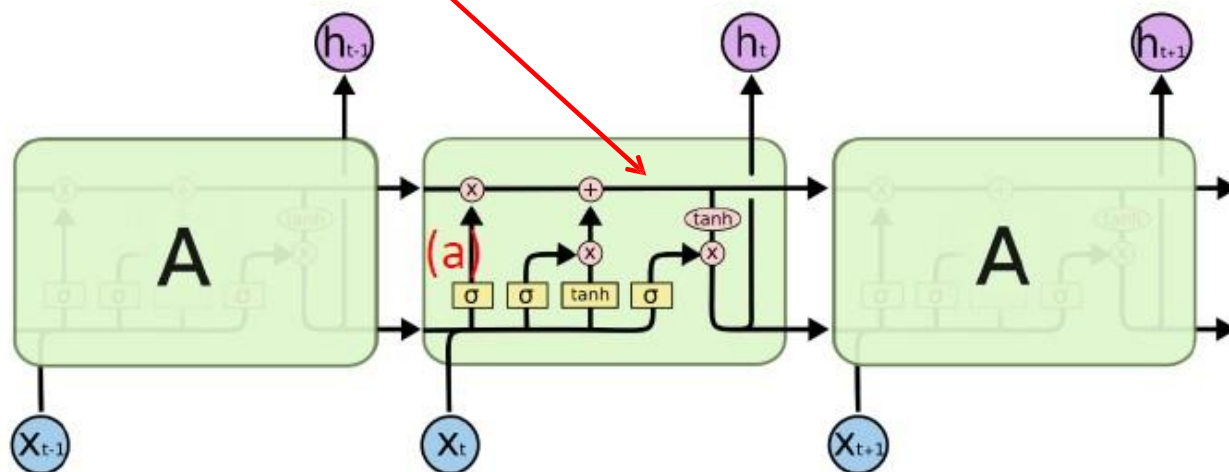


LSTM



# Long Short Term Memory Network

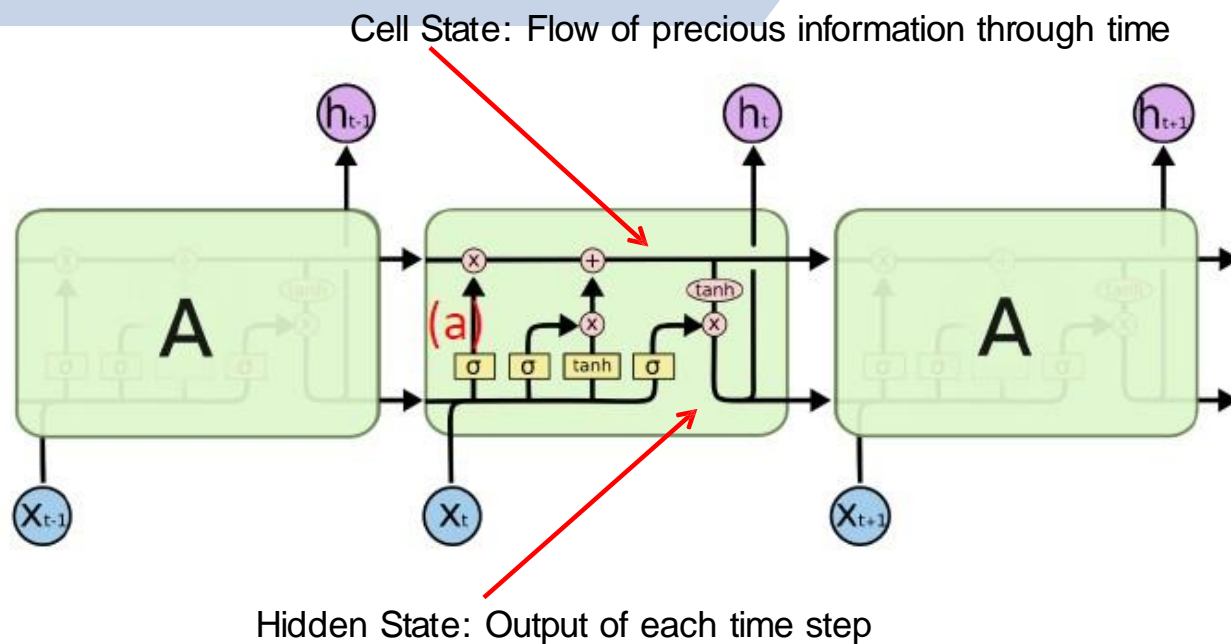
Cell State: Flow of precious information through time



LSTM



# Long Short Term Memory Network



LSTM

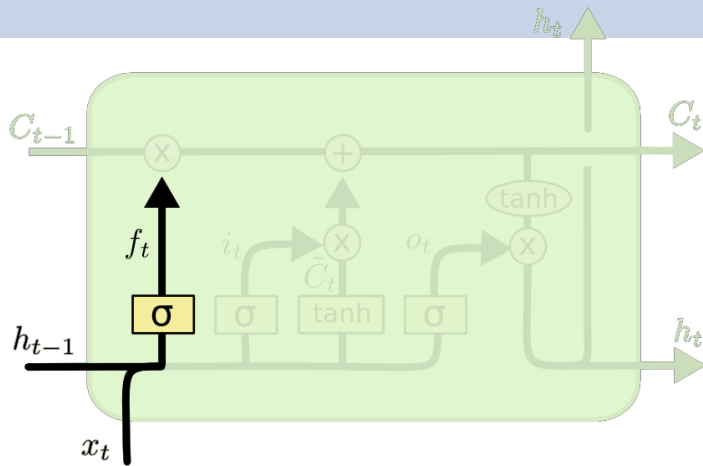


## Long Short Term Memory Network

1.  $C_{t-1}$  에서 불필요한 정보를 지운다.
2.  $C_{t-1}$ 에 새로운 인풋  $x_t$ 와  $h_{t-1}$ 를 보고 중요한 정보를 넣는다.
3. 위 과정을 통해  $C_t$ 를 만든다.
4.  $C_t$ 를 적절히 가공해 해당  $t$ 에서의  $h_t$ 를 만든다.
5.  $C_t$ 와  $h_t$ 를 다음 스텝  $t+1$ 로 전달한다.



# Long Short Term Memory Network



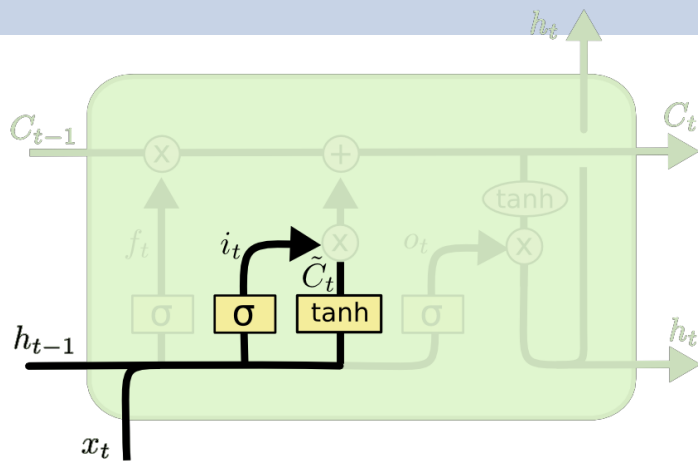
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$



1.  $C_{t-1}$  에서 불필요한 정보를 지운다.  
 $f$ 는 forget gate를 의미



# Long Short Term Memory Network



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



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

2.  $C_{t-1}$ 에 새로운 인풋  $x_t$ 와  $h_{t-1}$ 를 보고 중요한 정보를 넣는다.

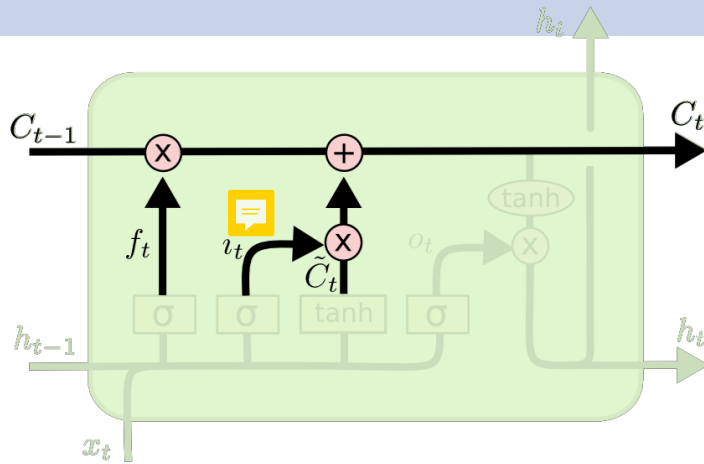
$i$ 는 input gate를 의미

임시  $C_t$ 를 계산





# Long Short Term Memory Network



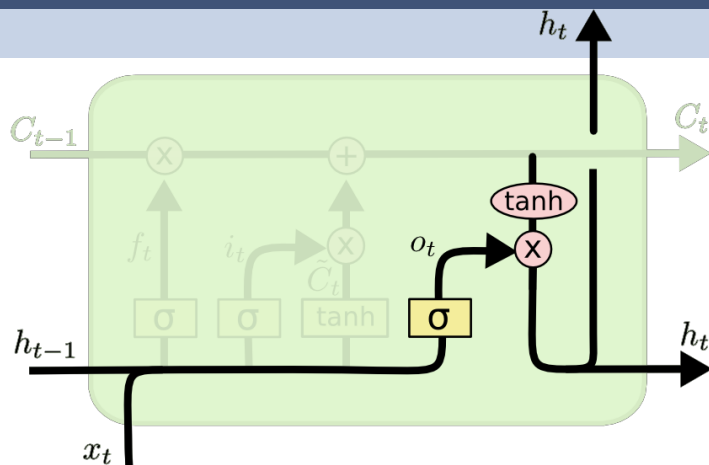
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

3. 위 과정을 통해  $C_t$ 를 만든다.

$f_t$ 를 이용해서  $C_{t-1}$ 의 일부 정보를 날리고 임시  $C_t$  정보를 추가한다



# Long Short Term Memory Network



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

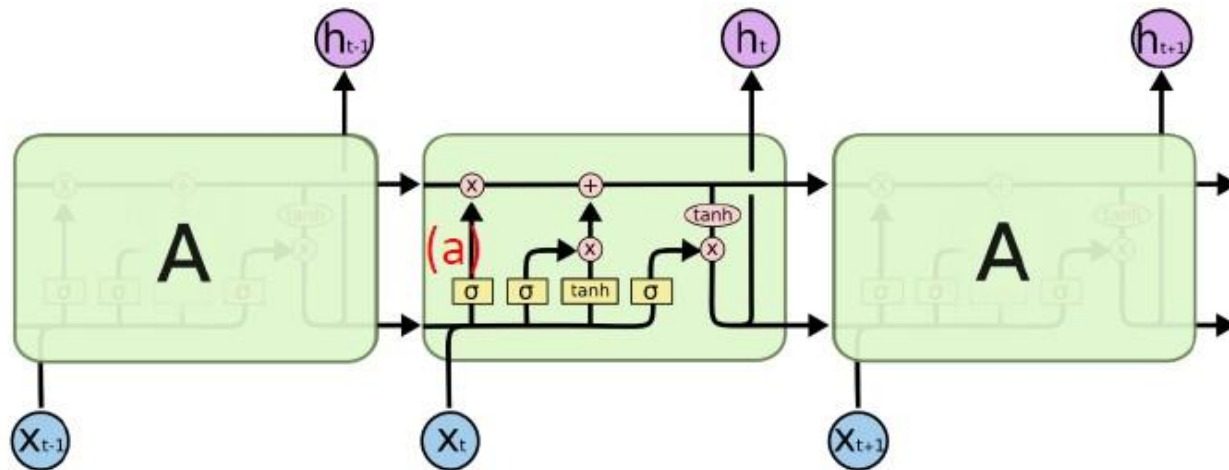
$$h_t = o_t * \tanh(C_t)$$

4.  $C_t$ 를 적절히 가공해 해당 t에서의  $h_t$ 를 만든다.

$C_t$ 를 가공할 output gate  $o_t$ 를 바탕으로  $h_t$ 를 계산



# Long Short Term Memory Network



5.  $C_t$ 와  $h_t$ 를 다음 스텝  $t+1$ 로 전달한다.



# Long Short Term Memory Network

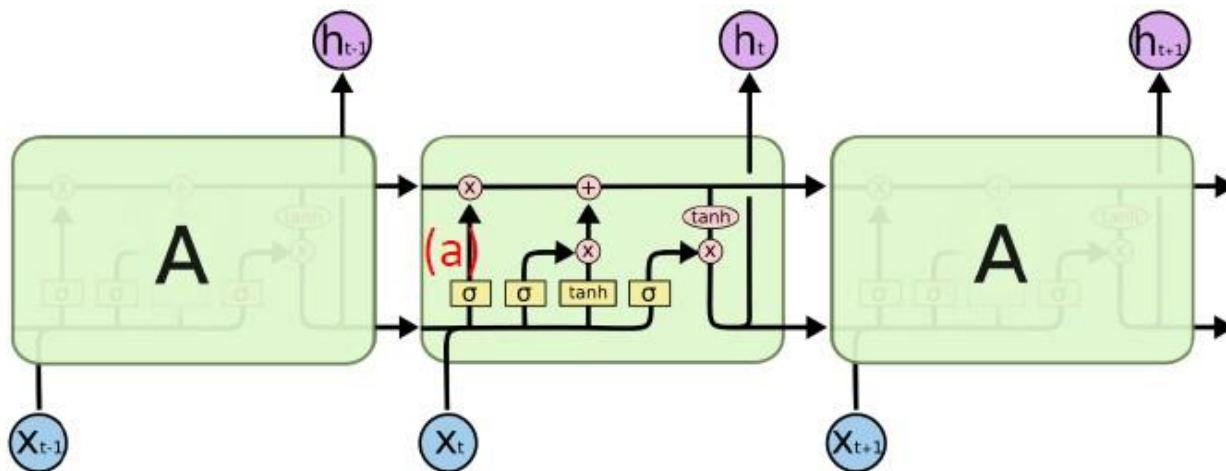
1.  $C_{t-1}$  에서 불필요한 정보를 지운다.  $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
2.  $C_{t-1}$  에 새로운 인풋  $x_t$  와  $h_{t-1}$  를 보고 중요한 정보를 넣는다.  $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
3. 위 과정을 통해  $C_t$  를 만든다.  $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
4.  $C_t$  를 적절히 가공해 해당  $t$  에서의  $h_t$  를 만든다.  $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
5.  $C_t$  와  $h_t$  를 다음 스텝  $t+1$  로 전달한다.  $o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$   
 $h_t = o_t * \tanh(C_t)$



So, How LSTM could overcome vanishing gradient problem?



## So, How LSTM could overcome vanishing gradient problem?

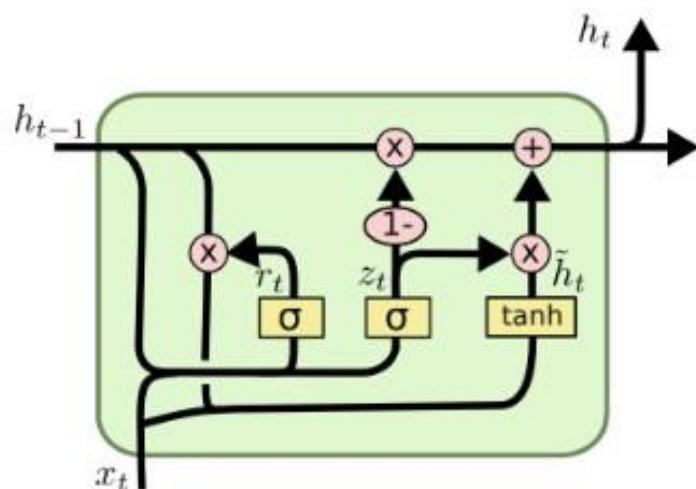


$t$  시점에서 cell state에 기록된 정보가 만약 지워지지 않고  $t+n$ 에서 활용 되었을 때 중간에 non-linear activation function을 거치지 않고  $t+n$  시점까지 흘러오기 때문에 Vanishing Gradient Problem을 상당 부분 해결 할 수 있다.

# Gated Recurrent Unit (GRU)



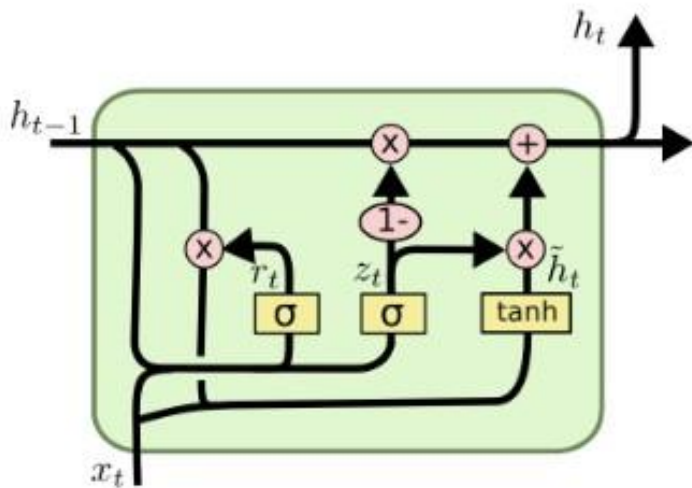
# Gated Recurrent Unit (GRU)







## Gated Recurrent Unit (GRU)



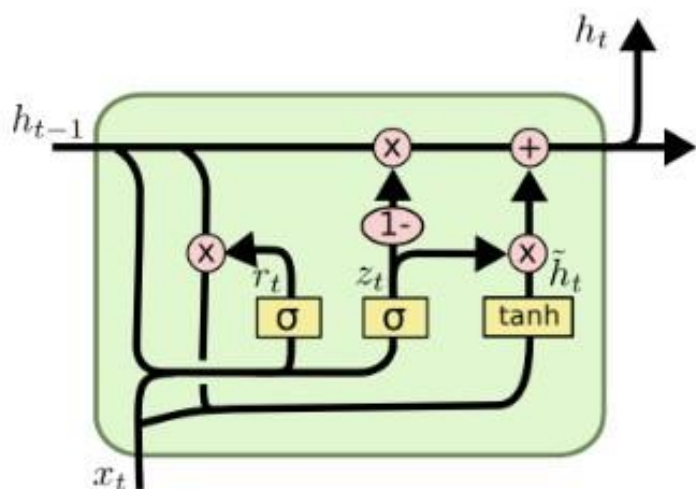
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

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

1. Reset gate를 계산해서 임시  $h_t$ 를 만든다.



## Gated Recurrent Unit (GRU)

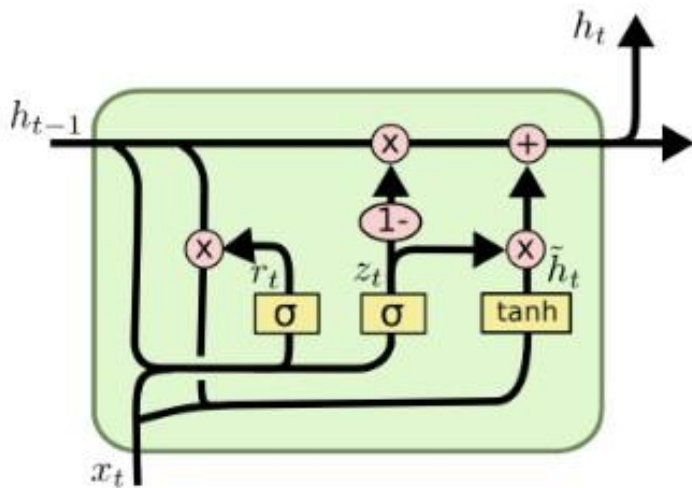


$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

2. Update gate를 통해  $h_{t-1}$ 과  $h_t$ 간의 비중을 결정한다.



## Gated Recurrent Unit (GRU)

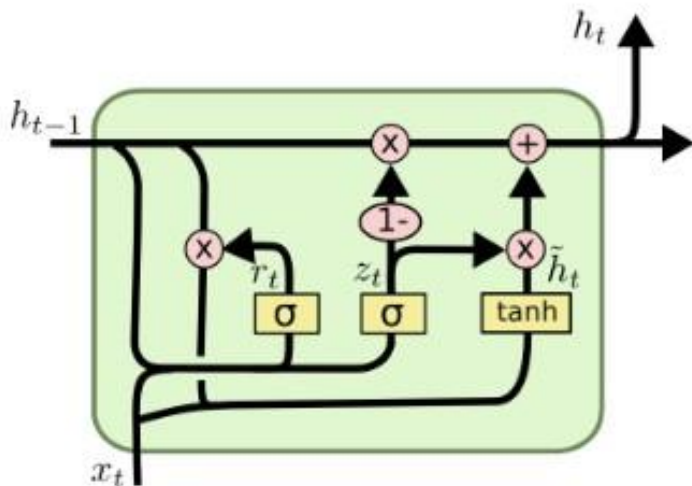


$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

3.  $z_t$ 를 이용해 최종  $h_t$ 를 계산한다.



## Gated Recurrent Unit (GRU)



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

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

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

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

1. Reset gate를 계산해서 임시  $h_t$ 를 만든다.
2. Update gate를 통해  $h_{t-1}$ 과  $h_t$ 간의 비중을 결정한다.
3.  $z_t$ 를 이용해 최종  $h_t$ 를 계산한다.



# THANK YOU

Any questions?