KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information

2022

Advanced Topic in Research Data-centric Deep Learning

Lec 09: Introduction to Recurrent Neural Network

hsyi@kisti.re.kr

Hongsuk Yi (이홍석)

Lecturer

- Name: Hongsuk Yi
- affiliation
  - ✓Principal Researcher, Dept. of Data-centric Problem Solving Research, KISTI
  - ✓Professor, Dept. of Applied AI, UST
- Research details
  - ✓Traffic congestion prediction in urban areas based on deep learning (2018-2021)
  - ✓Research on Intelligent Infrastructure Technology (2018~2019)
  - ✓Study on the Transformer Model for Traffic Prediction in Urban Areas (2022-2025)
- Computing technology
  - ✓GPU-based accelerated computing technology: CUDA, OpenCL
  - ✓Supercomputing Parallel Processing Technology: MPI, OpenMP, ManyCore Computing
  - ✓Deep Learning Technology: Tensorflow, PyTorch, Keras, Theano

A personal website: sites.google.com/view/deepai/home

Course Materials : github.com/hongsuk.yi
Information : sites.google.com/view/deepai/home
publication :https://www.researchgate.net/profile/Hongsuk-Yi/publications

Reviewing the last class:

❖ Course Information

- ✓ we learn the ability to develop deep learning applications by using various deep learning techniques like RNN, transformer, CNN, GAN, and deep reinforcement learning.
- ✓ This course meets for in-class lecture TUE 15:00PM - 18:00PM (UST Lecture room at KISTI).
- ✓ For all inquiries related to this course, please contact hongsuk.yi AT gmail DOT com
- ✓ Collaborative Lecture by Two Professors: Prof. Seongchan Kim, Prof. Hongsuk Yi
- ✓ Some papers in a reading list will be reviewed and presented by students

Syllabus

Materials Deep Learning with Tensorflow 2 and Keras (Second Edition, Packt Pub.)
(https://www.packtpub.com/product/deep-learning-with-tensorflow-2-and-keras/9781838823412)

Lecture 9	11/01	Recurrent Neural Networks and Long-Short Term Memory	Prof. Hongsuk Yi
Lecture 10	11/08	Attention mechanism and Transformer model	Prof. Hongsuk Yi
Lecture 11	11/15	Generative Adversarial Networks (1)	presentation 1, presentation 2
Lecture 12	11/22	Generative Adversarial Networks (2)	presentation 3, presentation 4
Lecture 13	11/29	Introduction to Deep Reinforcement Learning	presentation 5, presentation 6
Lecture 14	12/06	Deep Q-Networks	presentation 7, presentation 8
Lecture 15	12/13	Policy Gradients	presentation 9, presentation 10
Lecture 16	12/20	Final Exam	-

Reading list for further discussion

- https://github.com/hongsukyi/lec22B_DeepLearning

Attention and Transformers

- [Attention] Attention is All You Need, In Proceedings of NeurIPS 2017
- [BERT] BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, In Proceedings of ACL 2019

Generative Adversarial Networks

- [Original GAN] Generative Adversarial Networks, Goodfellow et al. (2014)
- [StyleGAN] A Style-Based Generator Architecture for Generative Adversarial Networks Karras et al. (2019).

Reading list for further discussion

- https://github.com/hongsukyi/lec22B_DeepLearning

Deep Reinforcement Learning

- Deep Q-Learning
  - [DQN] Playing Atari with Deep Reinforcement Learning, arXiv preprint arXiv:1312.5602
  - [PER] Prioritized Experience Replay arXiv preprint arXiv:1511.05952 (2015)
  - [Dueling DQN] Dueling Network Architectures for Deep Reinforcement Learning arXiv:1511.06581 (2015).
  - [Double DQN] Deep Reinforcement Learning with Double Q-Learning (2015)
  - [Rainbow DQN] Rainbow: Combining Improvements in Deep Reinforcement Learning arXiv:1710.02298 (2017).

Reading list for further discussion

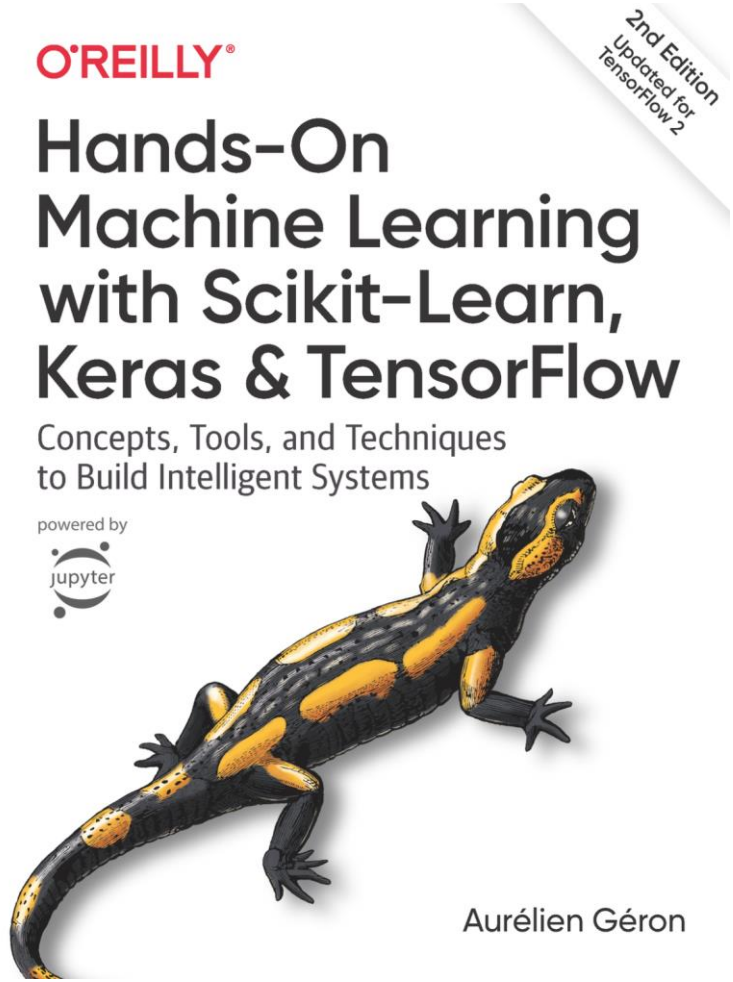
KISTI SINCE 1962

❖ https://github.com/hongsukyi/lec22B_DeepLearning

- Policy Gradient and Deterministic Policy Gradients
  - [A3C] < a href="https://arxiv.org/pdf/1602.01783.pdf"> Asynchronous Methods for Deep Reinforcement Learning Mnih et al, 2016.
  - [DPG] < a href="http://proceedings.mlr.press/v32/silver14.pdf"> Deterministic Policy Gradient Algorithms Silver et al, 2014
  - [DDPG] < a href="https://arxiv.org/pdf/1509.02971.pdf"> Continuous Control With Deep Reinforcement Learning Lillicrap et al, 2015

Introduction to Recurrent Neural Network

Reference Materials

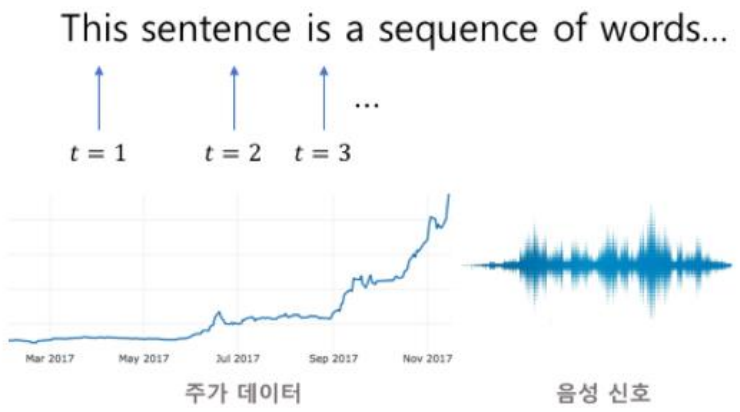


딥 러닝을 이용한 자연어 처리 입문

https://wikidocs.net/book/2155

What is a time series problem?

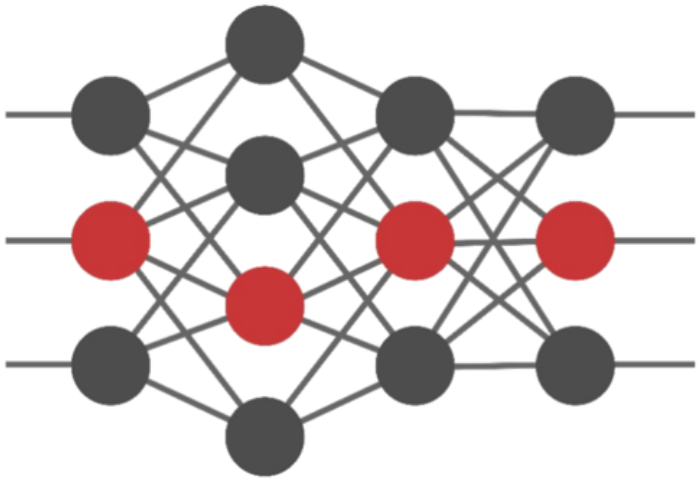
- Time series deals with data over time:
  - ✓Predicting
    - weather (short term forecasts), climate (long term forecasts)
    - sales accross different geographies for different products in the catalog



Why Not Feedforward Networks

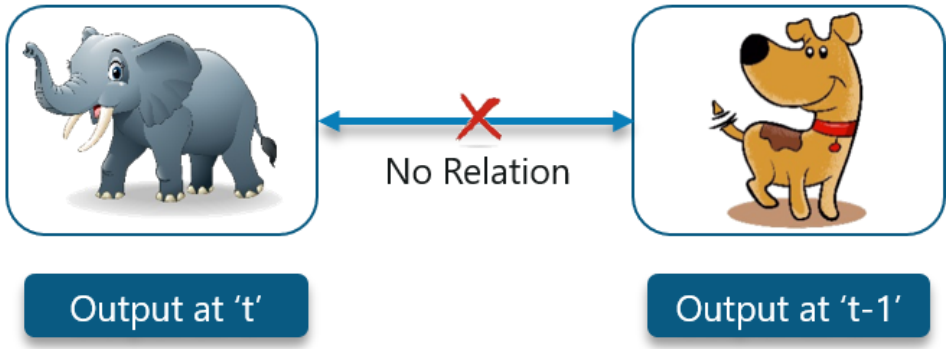
In the existing FeedForWared Networks,

Input at t-1



1) The input is a dog image,
and a well-learned neural network
outputs a dog label.

(3) However, elephants and dogs are not related to each other and are independent.

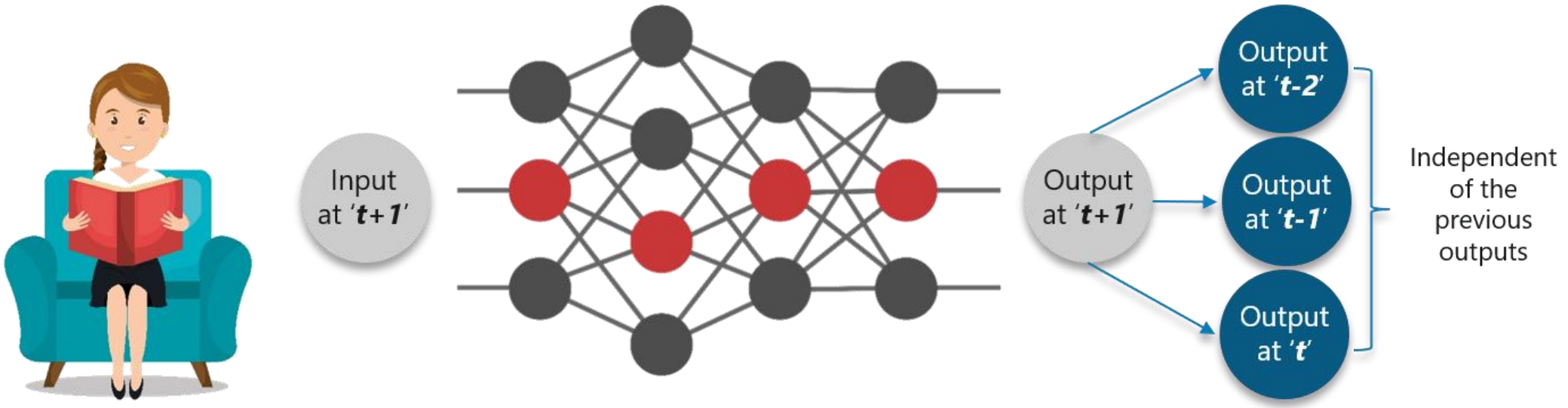


(2) The next elephant image is input to output an elephant (label).

Why Not Feedforward Networks

- FN cannot predict the relationship between the following words with previous words.

If you are reading a book, you should be well aware of the previous page.

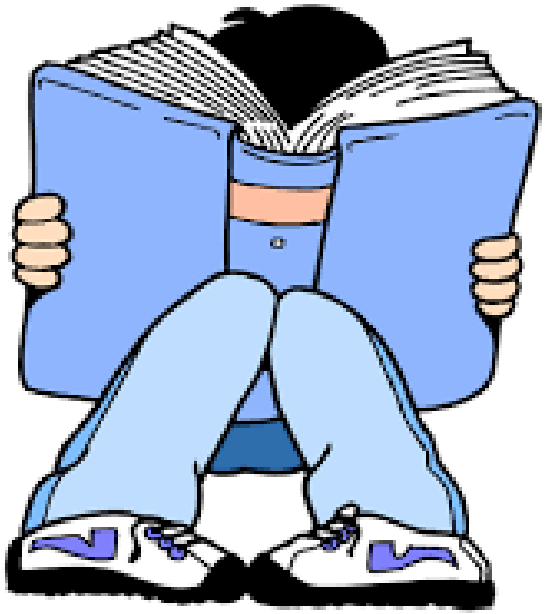


No order when reading a book

- a 10 page picture book

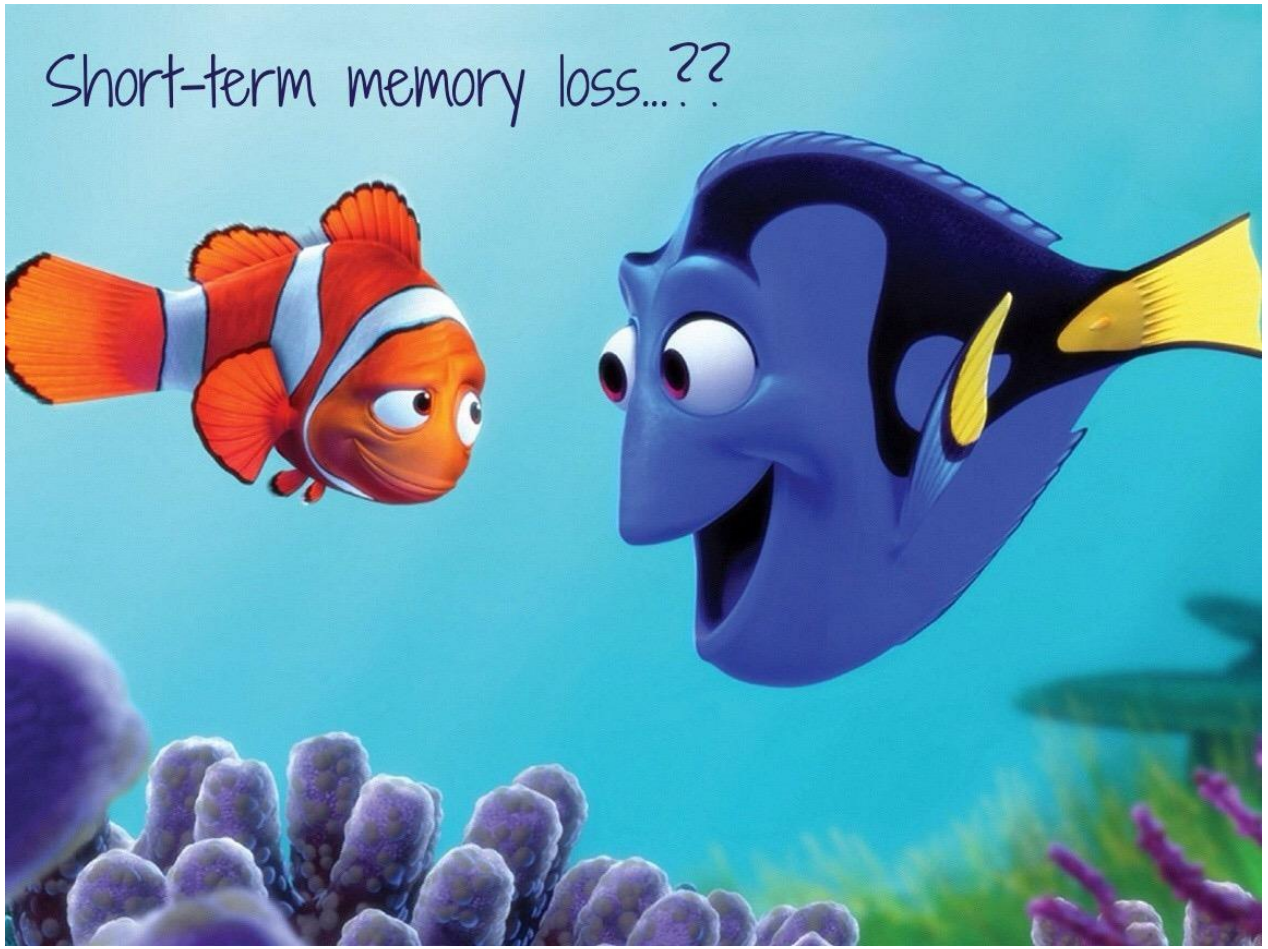
What is Memory

- How much content do you remember?
  - ✓A sime example: History Book
- What is the memory when reading a book
  - ✓Sequence Data Pattern?
  - ✓Memory Cell



When reading a full-length novel

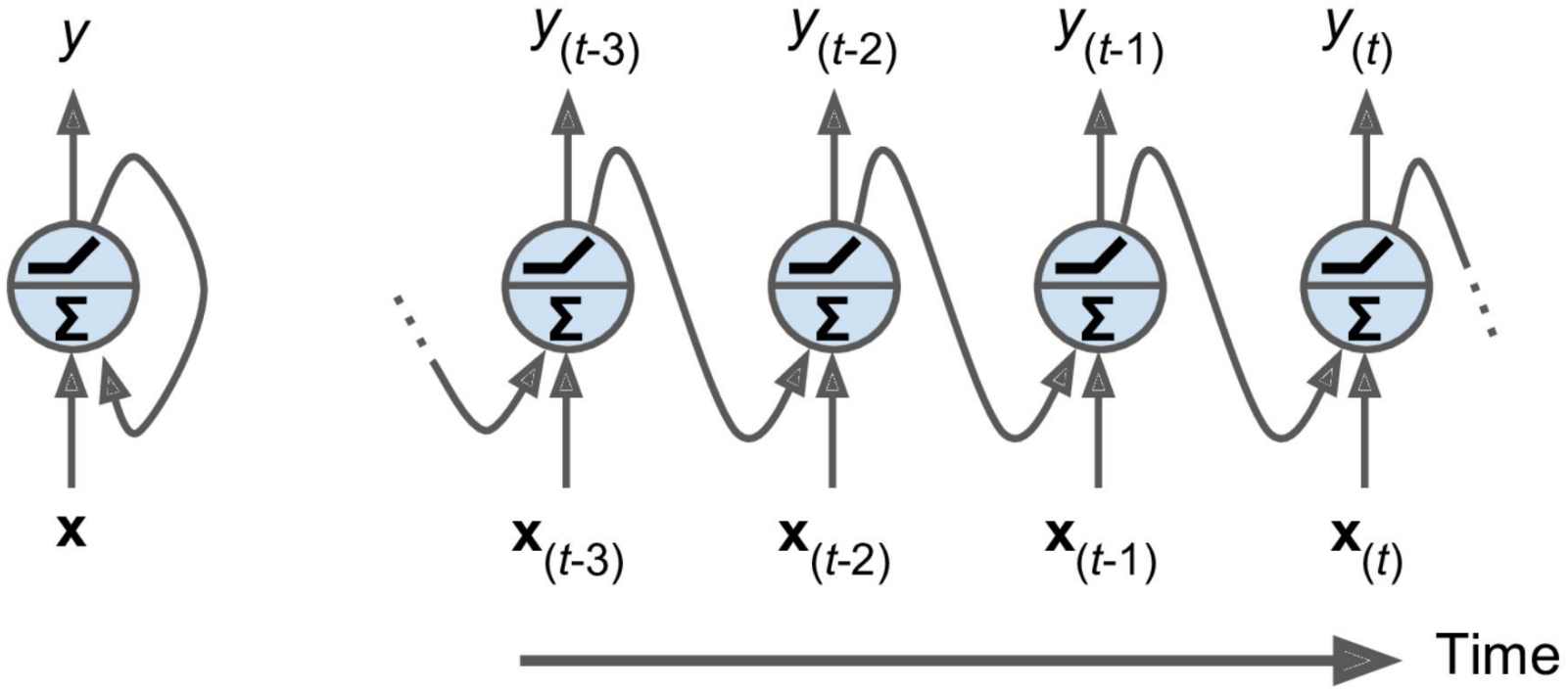
❖ Short-term and Long Term Memory Loss



❖ Attention : No order?

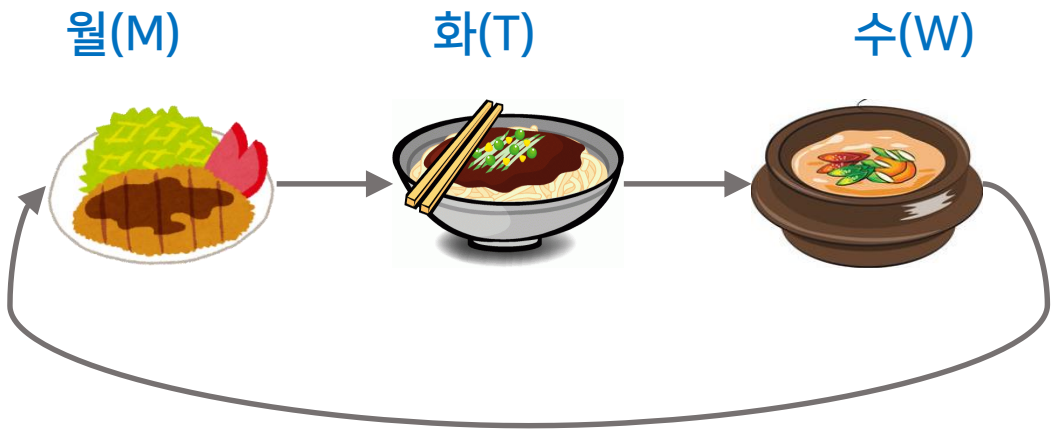
❖ Memory Cells ~ a form of memory

✓ A RNN looks like a feedforward NN, except it also has connections pointing backward.



A simple of RNN (1/3)

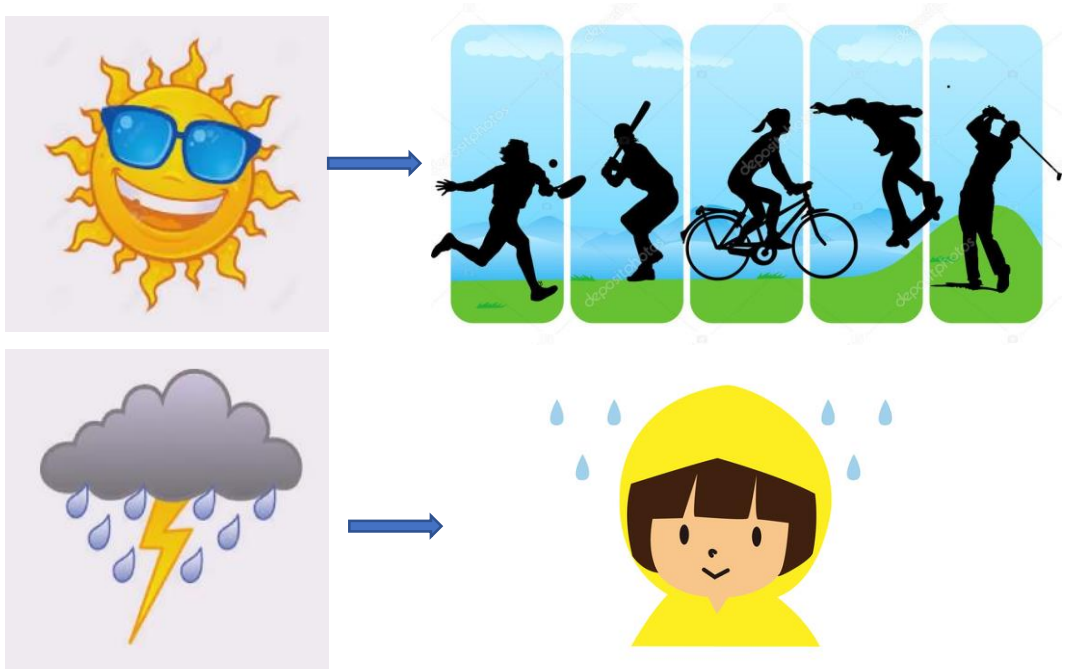
- There is a pattern in dinner.
  - ✓Only three types of meals are available.



RNN = Recurrent Neural Network

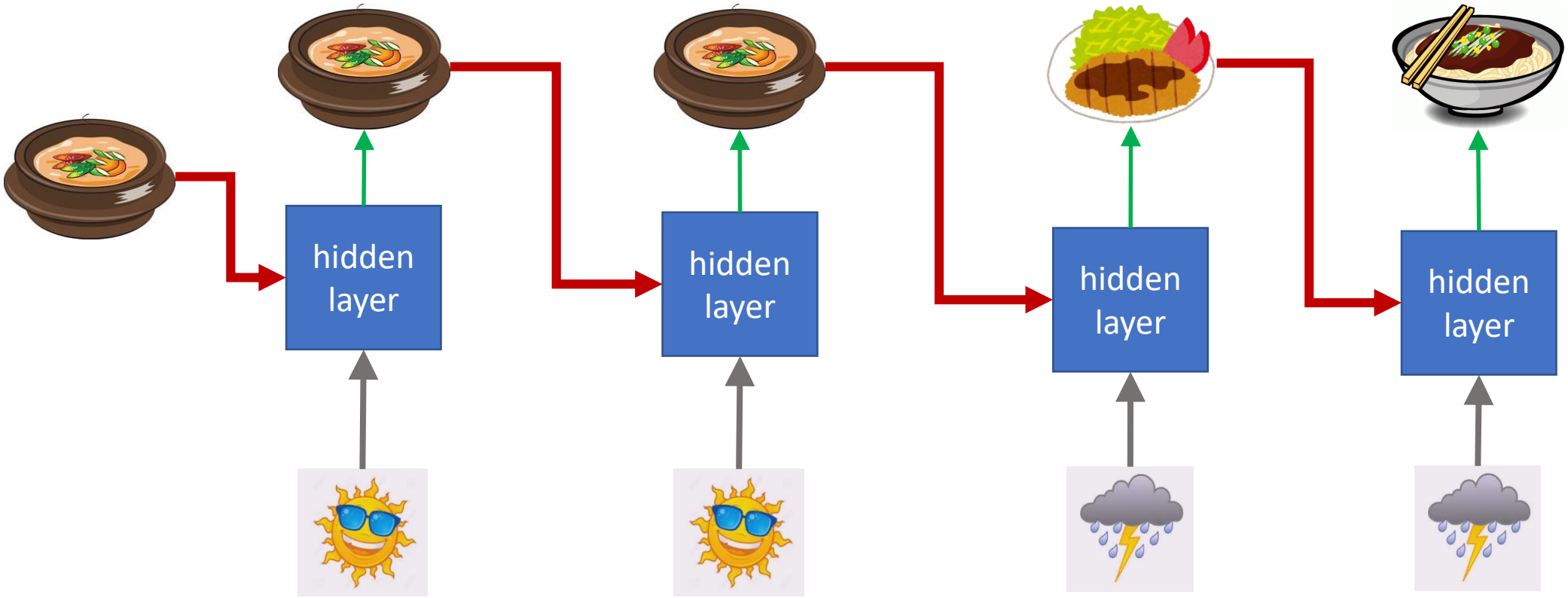
A simple of RNN (2/3)

- The menu varies depending on the weather
  - ✓If the weather is nice, I eat food the day before.
  - ✓When it rains, I eat food according to the pattern



A simple of RNN (3/3)

- You can predict food depending on the weather



Neurons in RNN

Output of a recurrent layer for a single instance

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{W}_x{}^{\intercal}\mathbf{x}_{(t)} + \mathbf{W}_y{}^{\intercal}\mathbf{y}_{(t-1)} + \mathbf{b}\right)$$

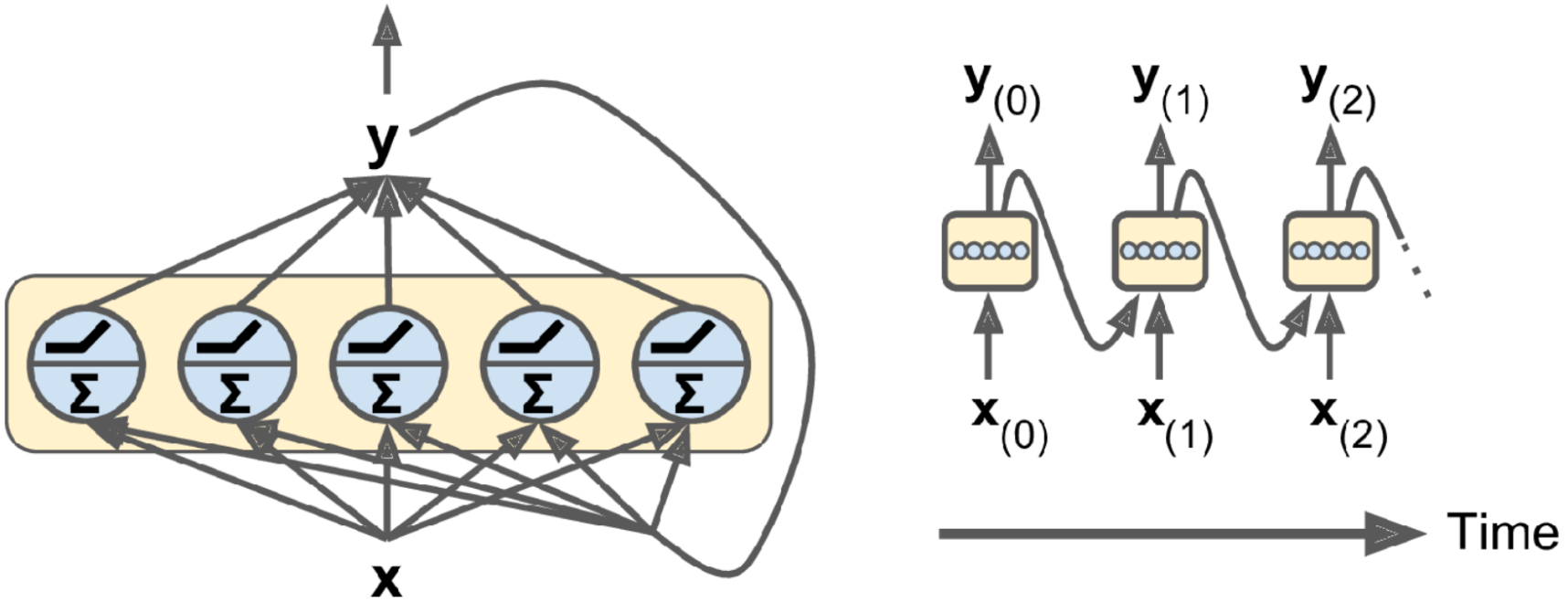
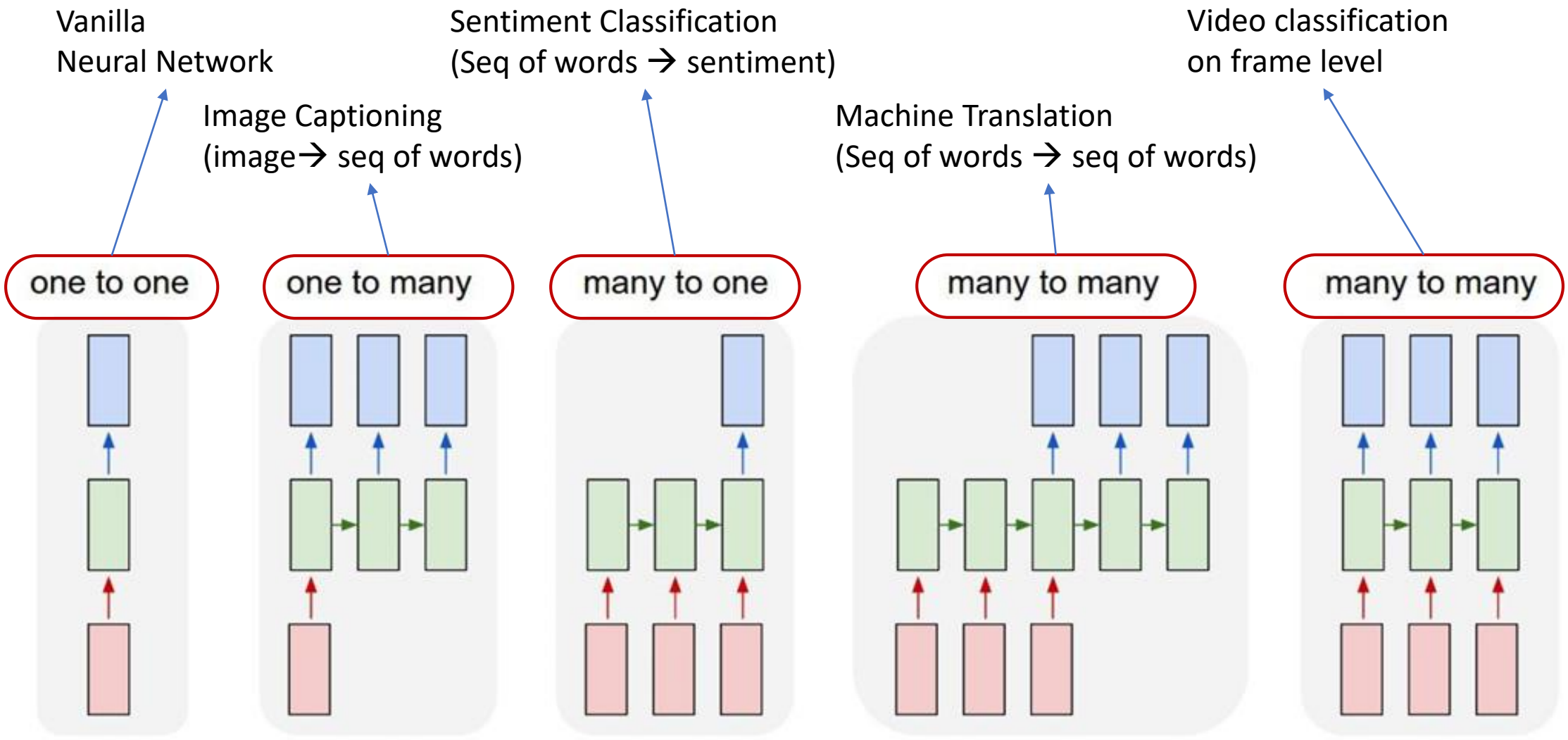


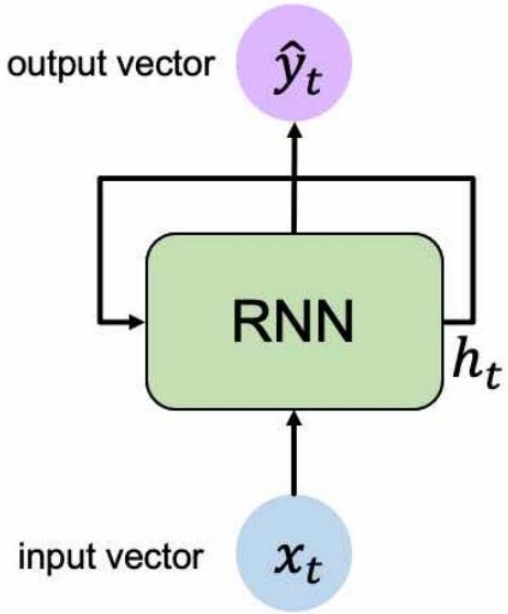
Figure 15-2. A layer of recurrent neurons (left) unrolled through time (right)

Recurrent Neural Networks: Process Sequences



How to train RNN?

RNN State Update and Output



Output Vector

$$\hat{y}_t = \boldsymbol{W}_{hy}^{T} h_t$$

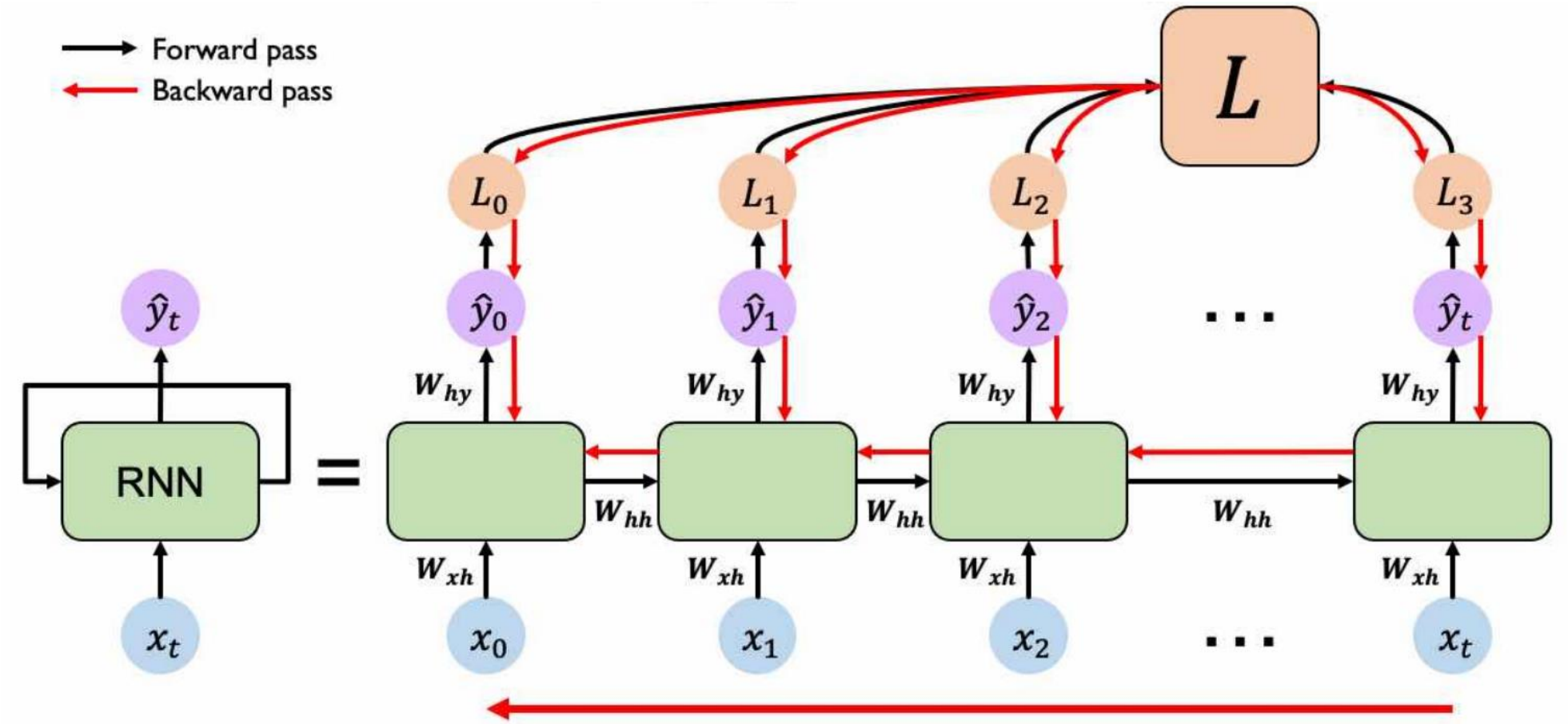
Update Hidden State

$$h_t = \tanh(\boldsymbol{W}_{hh}^{T} h_{t-1} + \boldsymbol{W}_{xh}^{T} x_t)$$

Input Vector

$$x_t$$

BPTT: Backpropagation Through Time



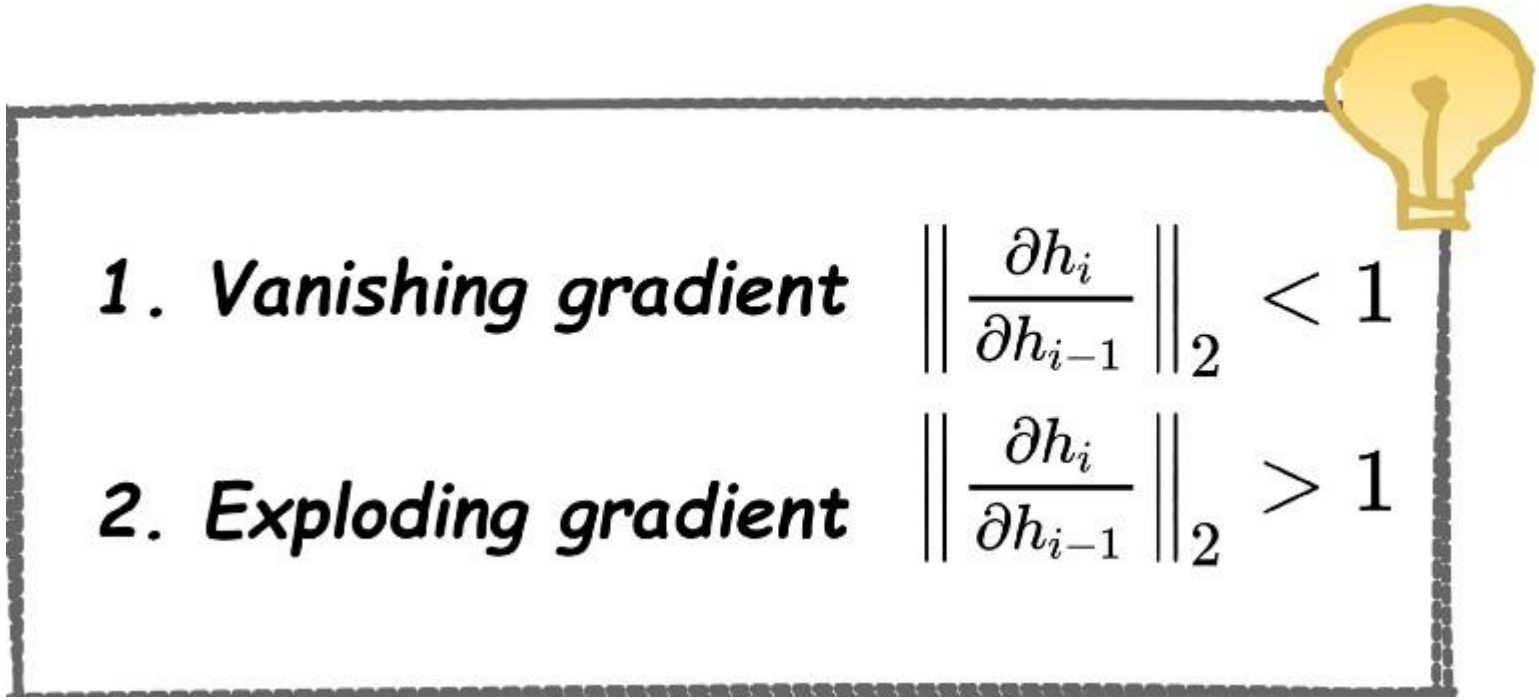
Standard RNN Gradient Flow

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

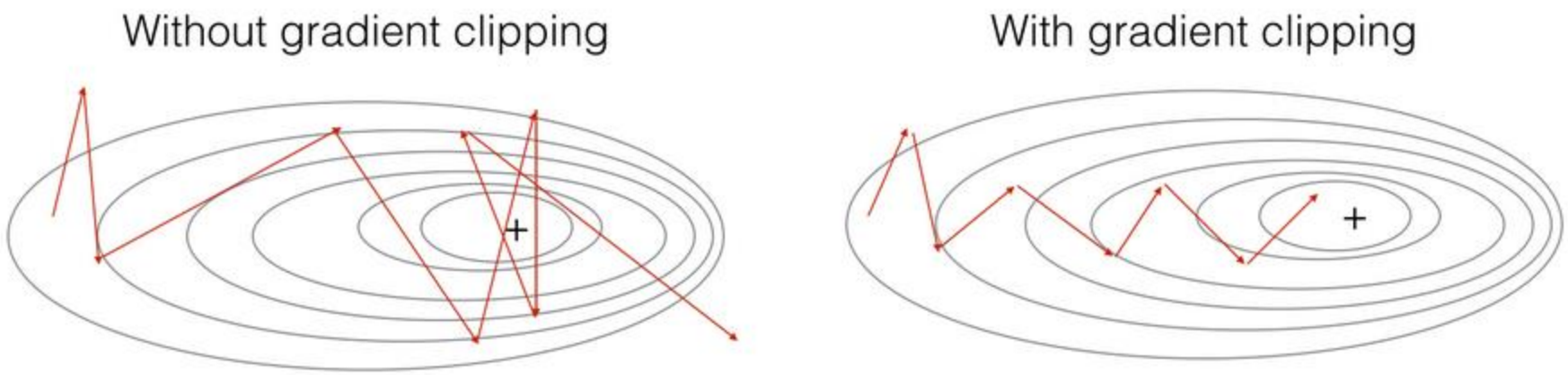
$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

RNN suffer from the problem of VGP and EGP



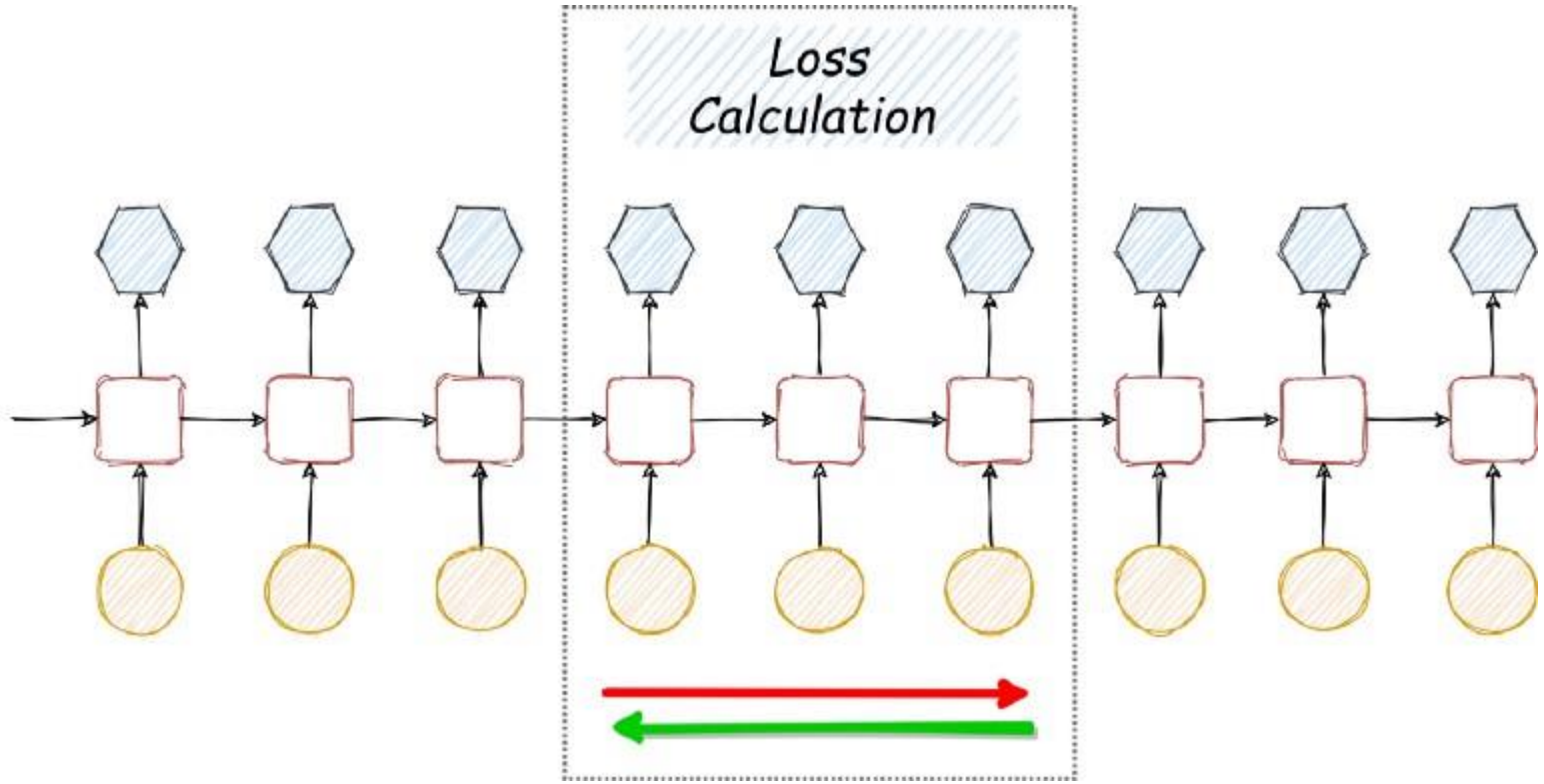
Solutions

- Truncated backpropagation through time (TBPTT)
- Long short-term memory (LSTM)
- Gradient Clipping



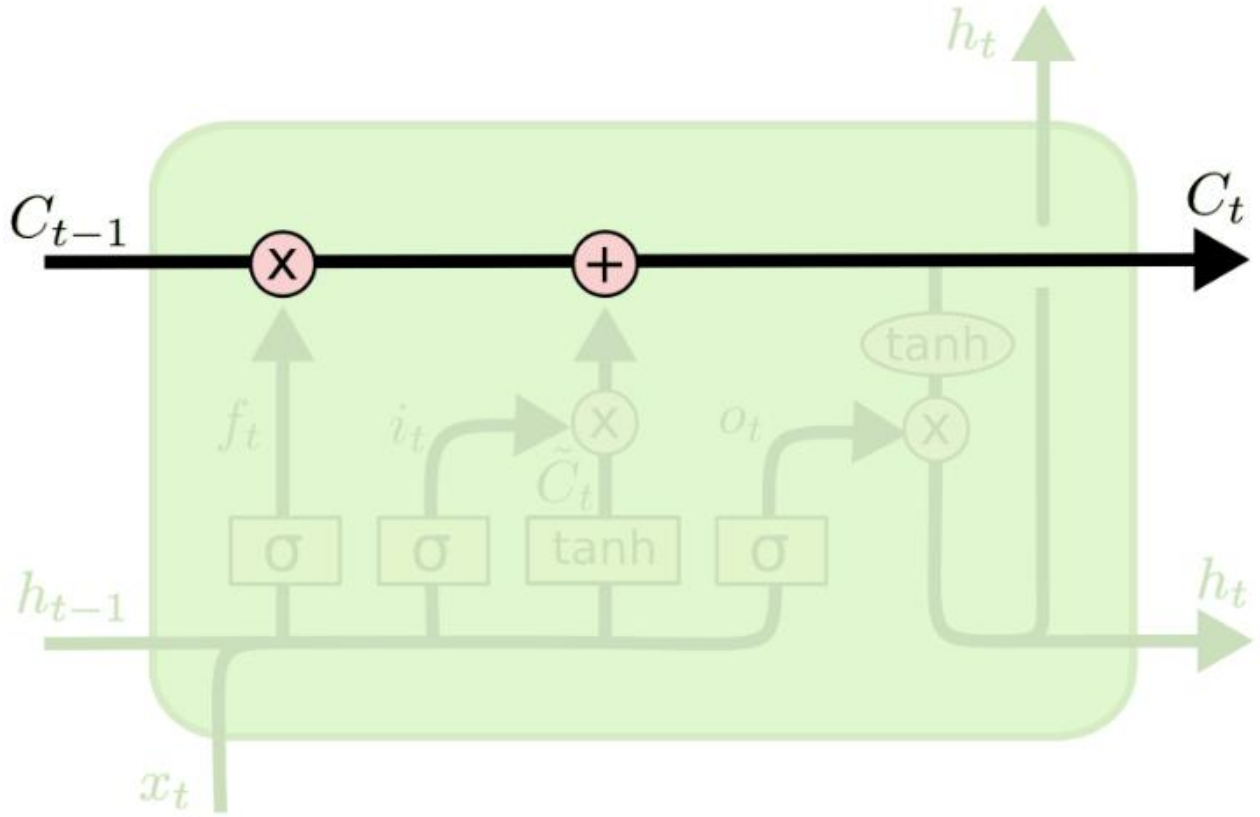
Truncated Backpropagation Through Time

- TBPTT: Truncated BPTT trick by considering a moving window through the training process

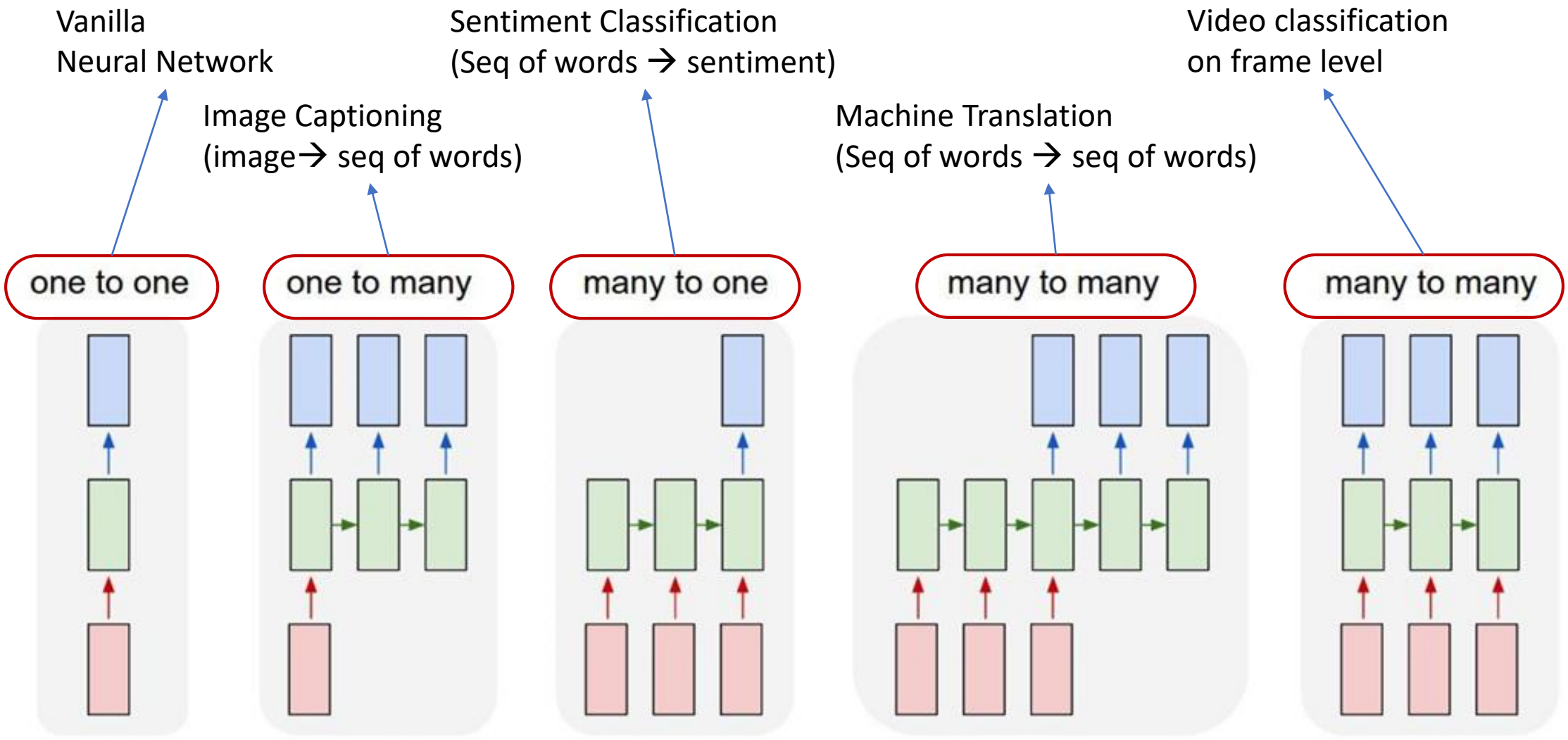


LSTMs

- Capable of learning long-term dependencies by remembering information
- Cell state



Recurrent Neural Networks: Process Sequences



Long-Short Term Memory
(장단기 메모리)

Recurrent NN for processing sequences

- RNN can handle interactions more flexibly
- they are applied in a step-by-step fashion to a sequential input
  - ✓ a sequence of words, characters, or something else
- RNNs use a state representing what has happened previously
  - ✓ after each step, a new state is computed

RNN example

https://chalmers.instructure.com/courses/16100

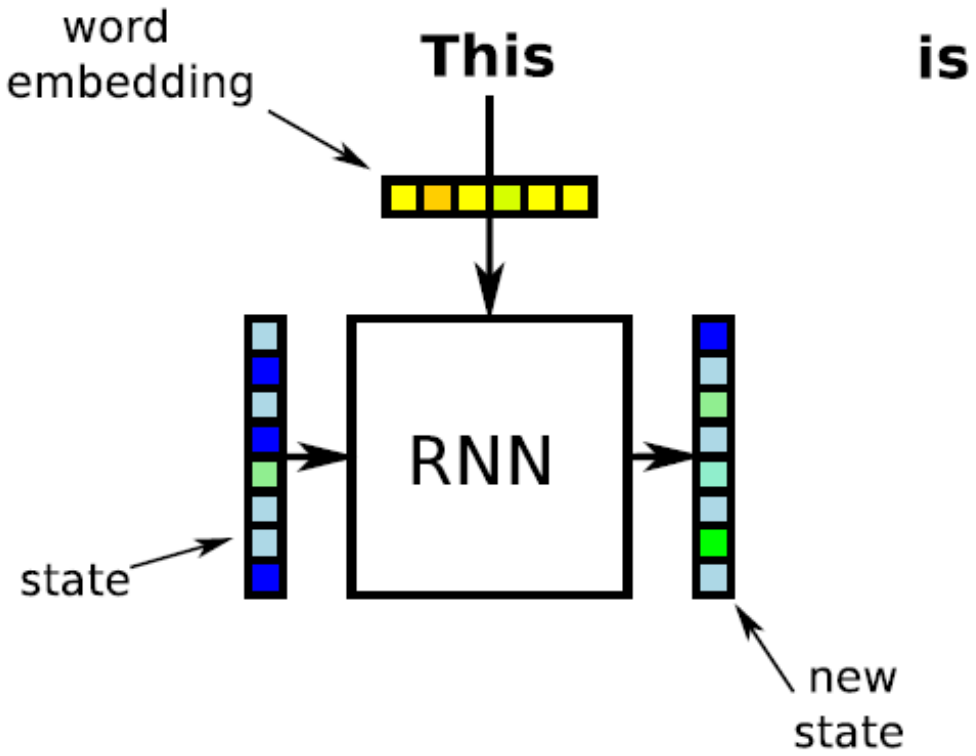


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

training RNNs: backpropagation through time

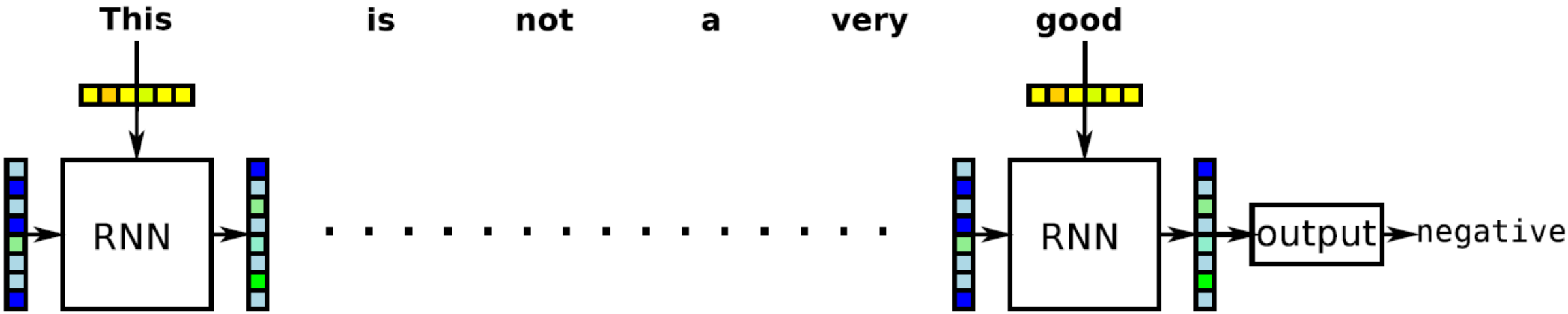


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

- the simple RNN looks similar to a feedforward NN
  - ✓the next state is computed like a hidden layer in a feedforward NN
  - ✓the output is identical to the state representation:

$$y_t = s_t$$

$$s_t = g(\boldsymbol{W} \cdot (s_{t-1} \oplus x_t) + \boldsymbol{b})$$

actication is typically tanh

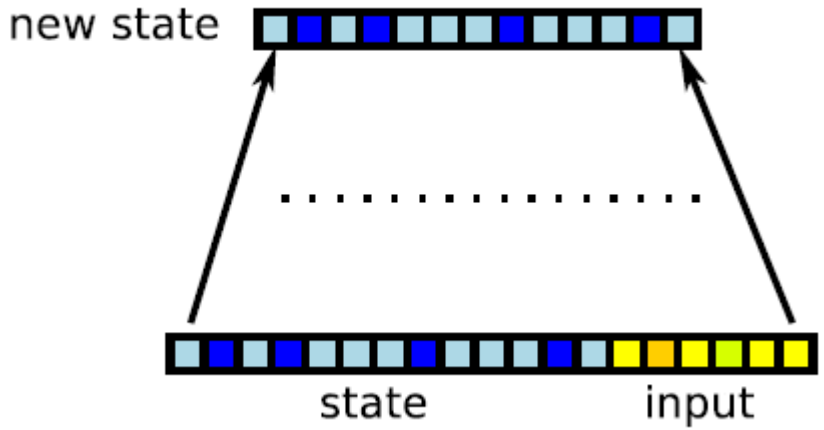


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

simple RNNs have a drawback

- simple RNNs suffer from the problem of vanishing gradients (Hochreiter, 1998)

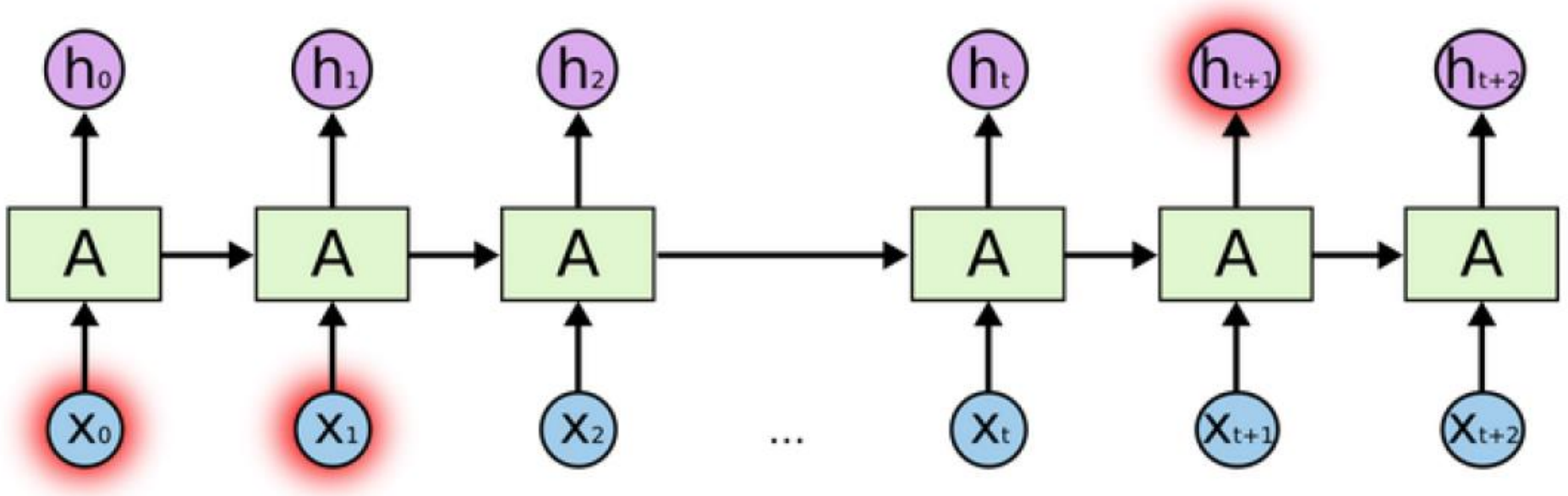


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

- gating architectures allow information flow to be controlled more carefully
  - ✓ should we copy the previous state, or replace it?
  - ✓ the "gates" are controlled by their own parameters

$$\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix} \leftarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}$$

$\mathbf{s}'$ $\mathbf{g}$ $\mathbf{x}$ $\mathbf{(1-g)}$ $\mathbf{s}$

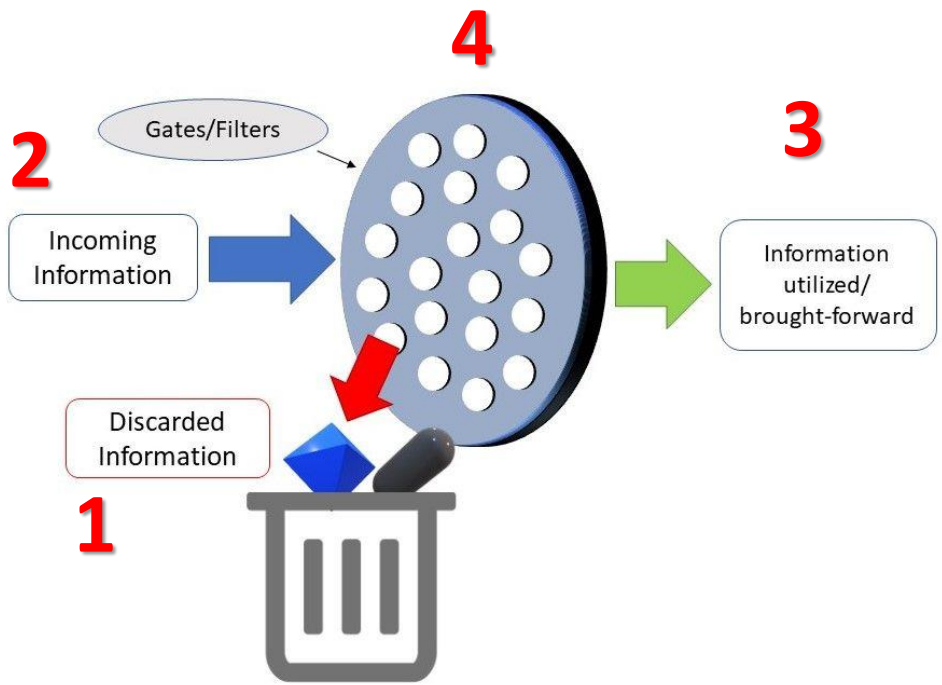
image from Goldberg's book

image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

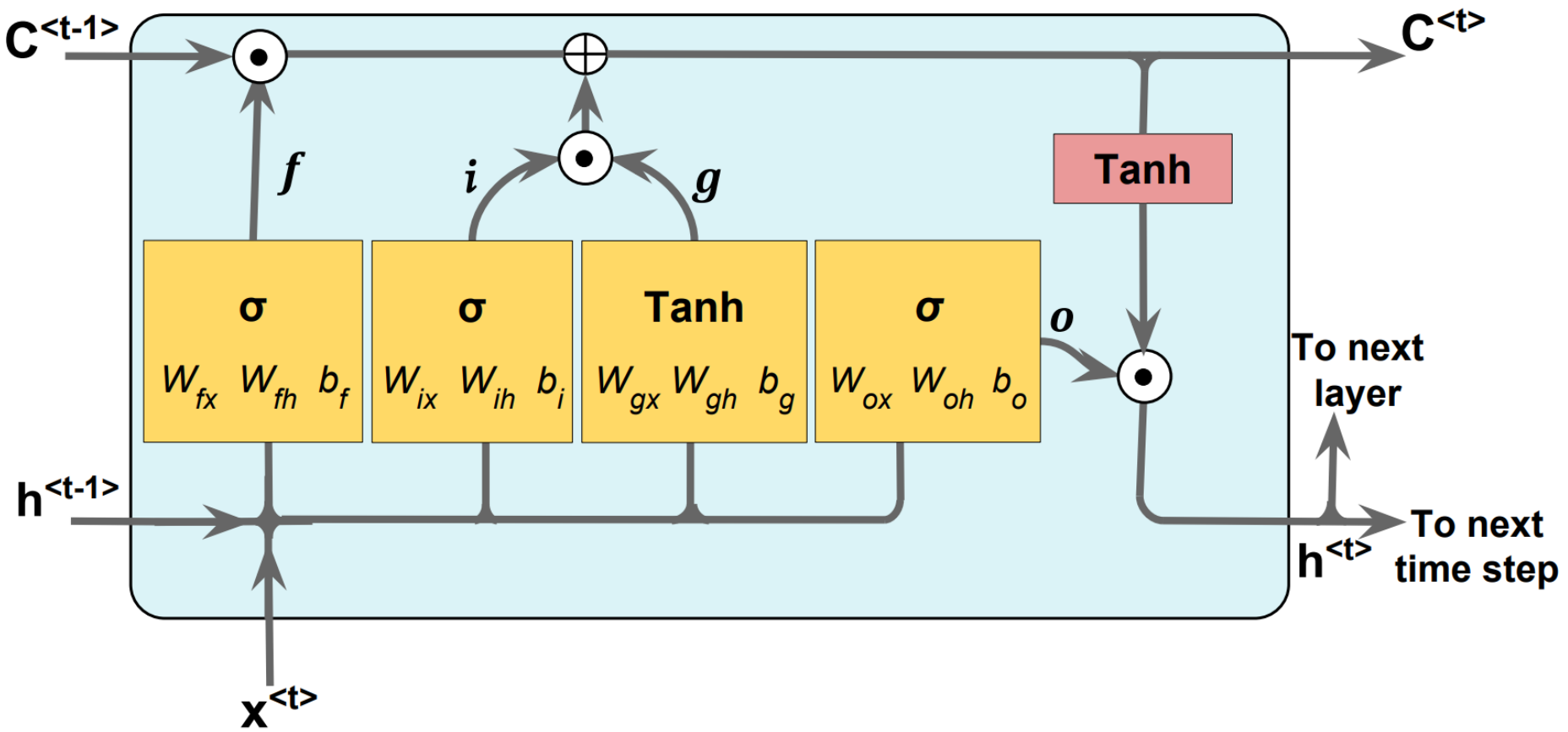
Cell states and Gates

Cell state

- ✓ Gates control the flow of information to/from the memory
- ✓ Gates are controlled by a concatenation of the output from the previous time step and the current input and optionally the cell state vector.



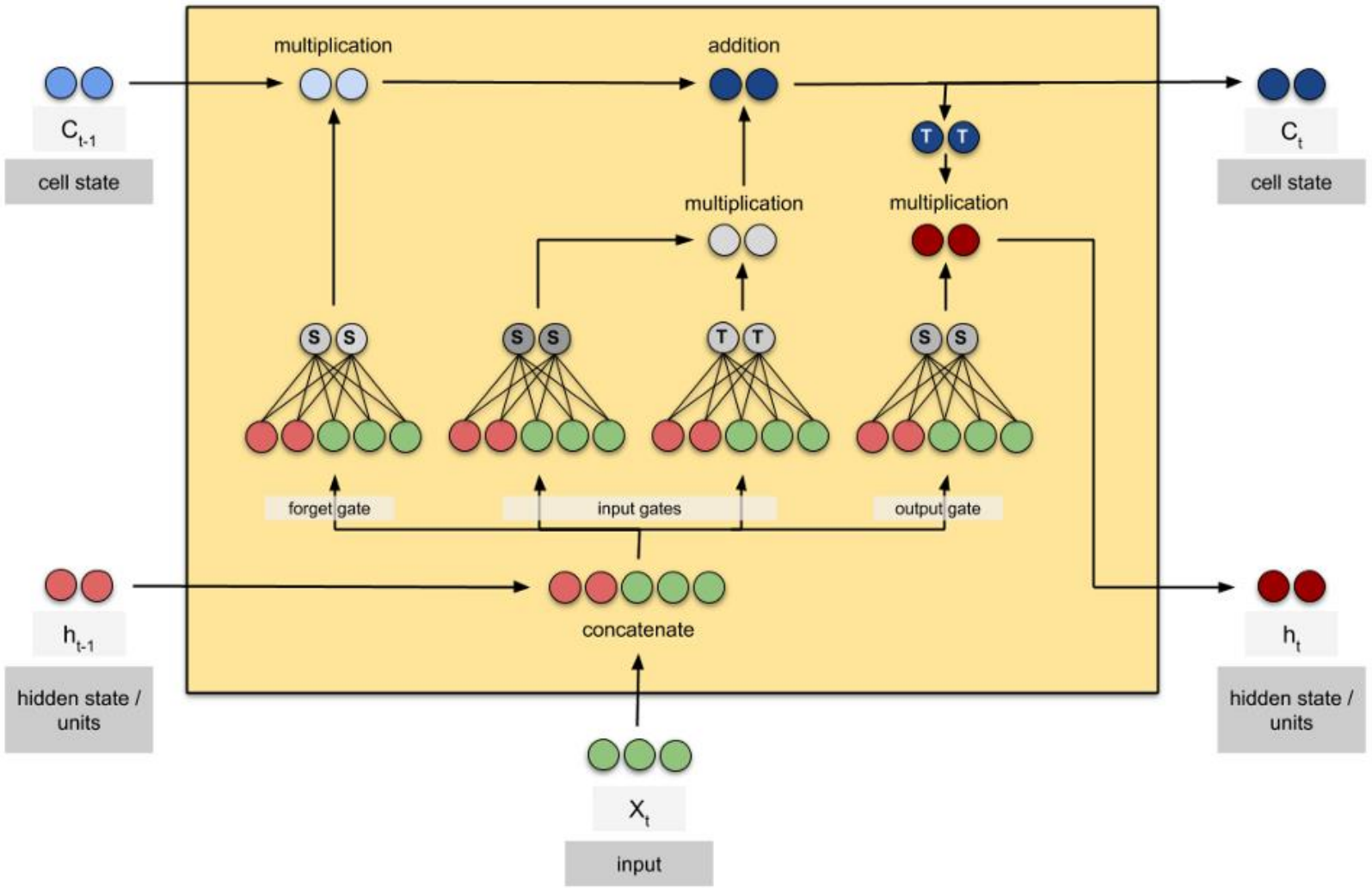
Last time: Simple RNN and LSTM



Understanding the roles played by gates in LSTM

- **Forget Gate**
  - ✓ whether we should keep the information from the previous timestamp or forget it
- **Input Gate**
  - ✓ Decide how much this unit adds to the current state
- **New information: Memory Upgate**
  - ✓ The cell state vector aggregates the two components (old memory via the forget gate and new memory via the input gate)
- **Output Gate**
  - ✓ Decide what part of the current cell state makes it to the output

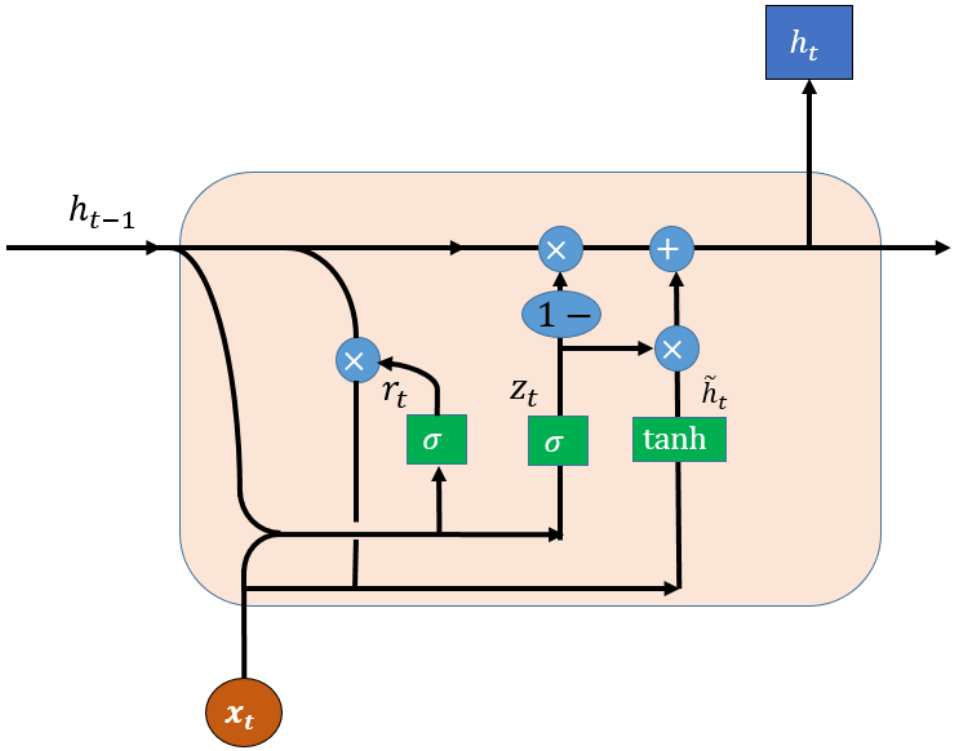
LSTM : Cell state and hidden state



S: Sigmoid
T: tanh

Gated Recurrent Units (GRU, 게이트 순환 유닛)

- Just like LSTM, GRU uses gates to control the flow of information.
  - ✓GRU simplifies the architecture of LSTM, which was similar in performance to LSTM and was complex



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$g_t = tanh(W_{hg}(r_t \circ h_{t-1}) + W_{xg}x_t + b_g)$$
$$h_t = (1 - z_t) \circ g_t + z_t \circ h_{t-1}$$

source: https://www.researchgate.net/figure/Structure-of-a-GRU-cell_fig1_334385520

limitations of RNNs

- even with gated RNNs, it can be hard to cram the useful information into the last state

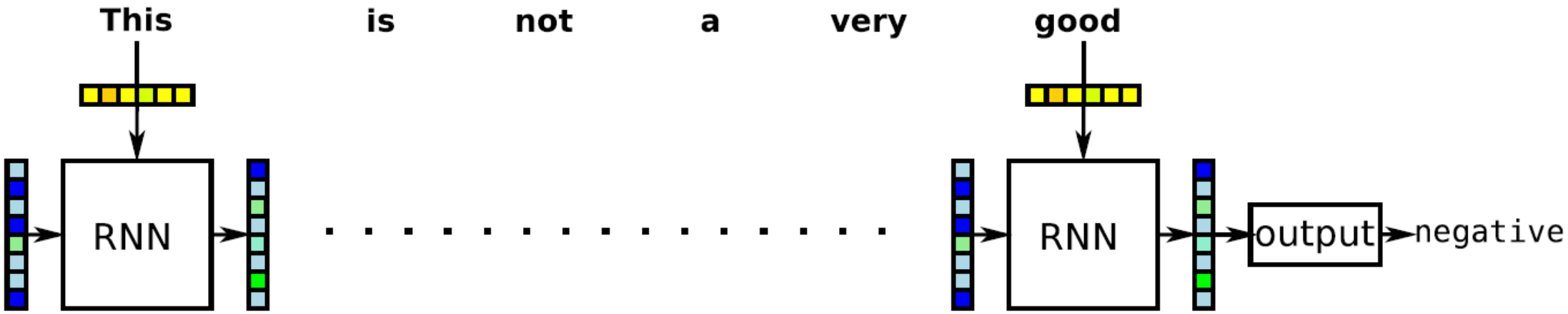


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

RNN, LSTM Hyperparameters

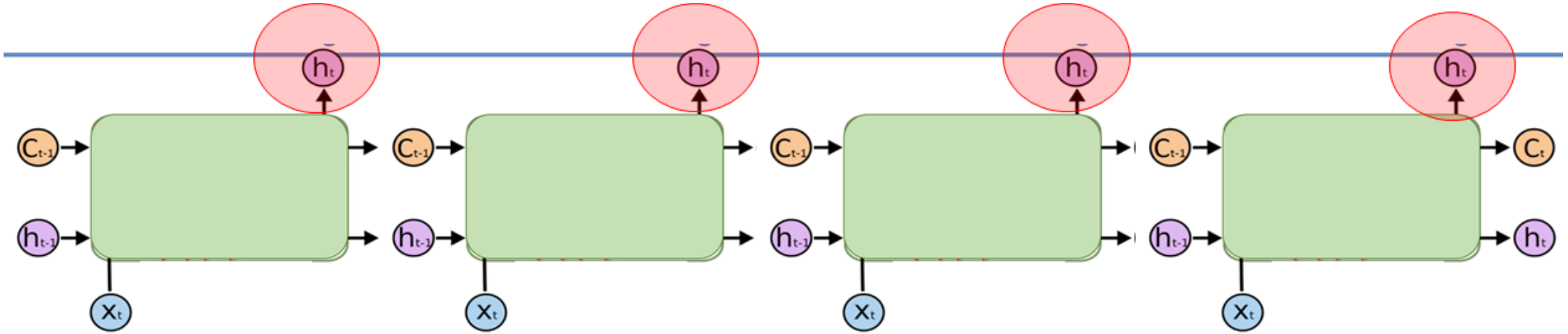
LSTM Parameters

❖ LSTM has 3 important parameters

- ✓ **neurons :** dimensionality of the output space
- ✓ **return_sequences**: whether to return the last output. (hidden state, memory cell,h)
  - in the output sequence, or the full sequence.
  - Default: False.
- ✓ **return_state**: whether to return the last state in addition to the output. (cell state, c)
  - Default: False.

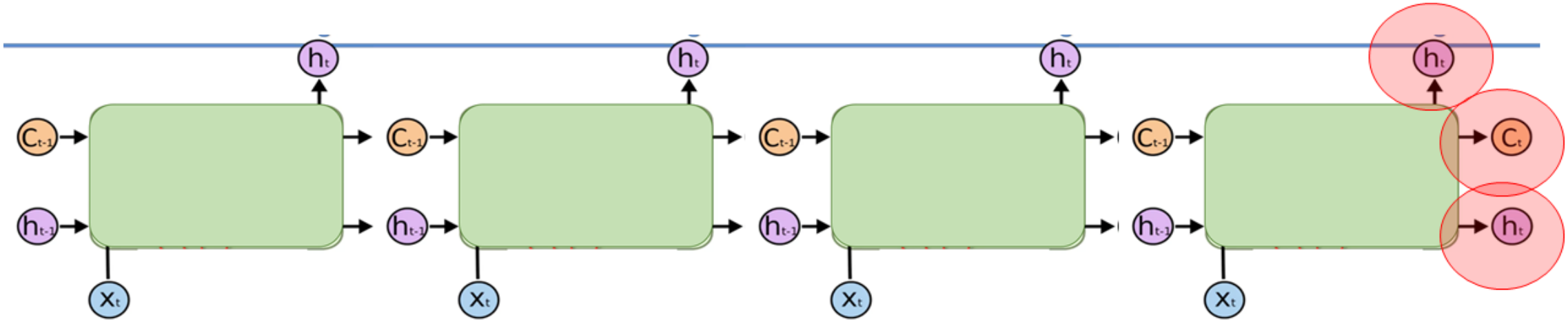
LSTM: return_sequences=True

- return_sequences = True
  - ✓all hidden states



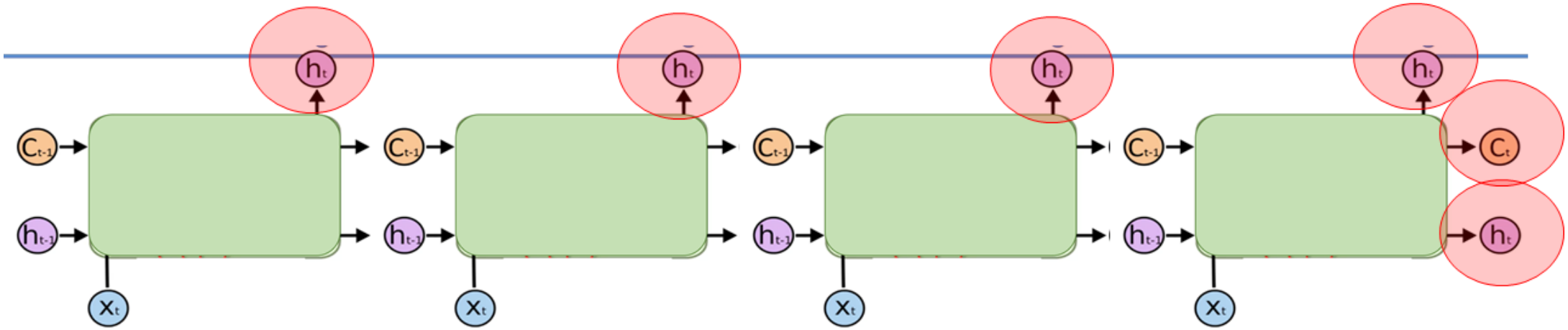
LSTM : return_state=True

- return_state=True
  - ✓last hidden state + last hidden state (again!) + last cell state

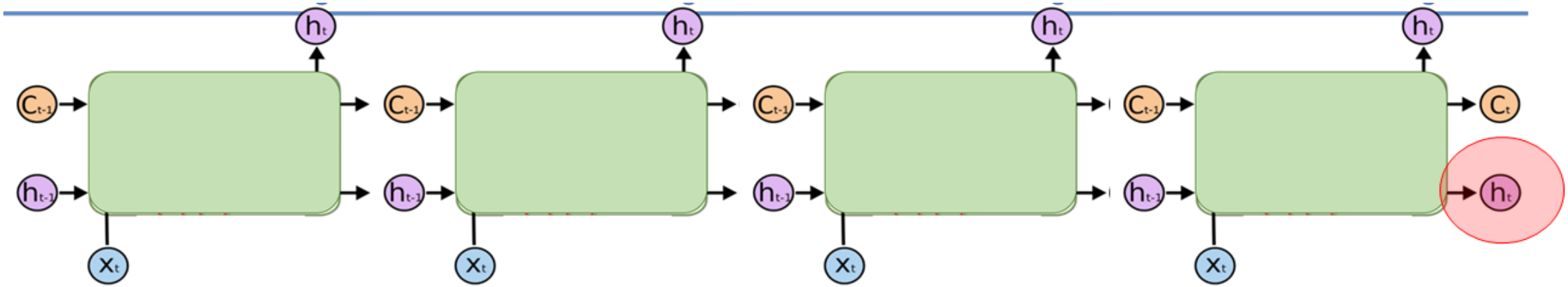


LSTM: return_sequences=True and return_state=True

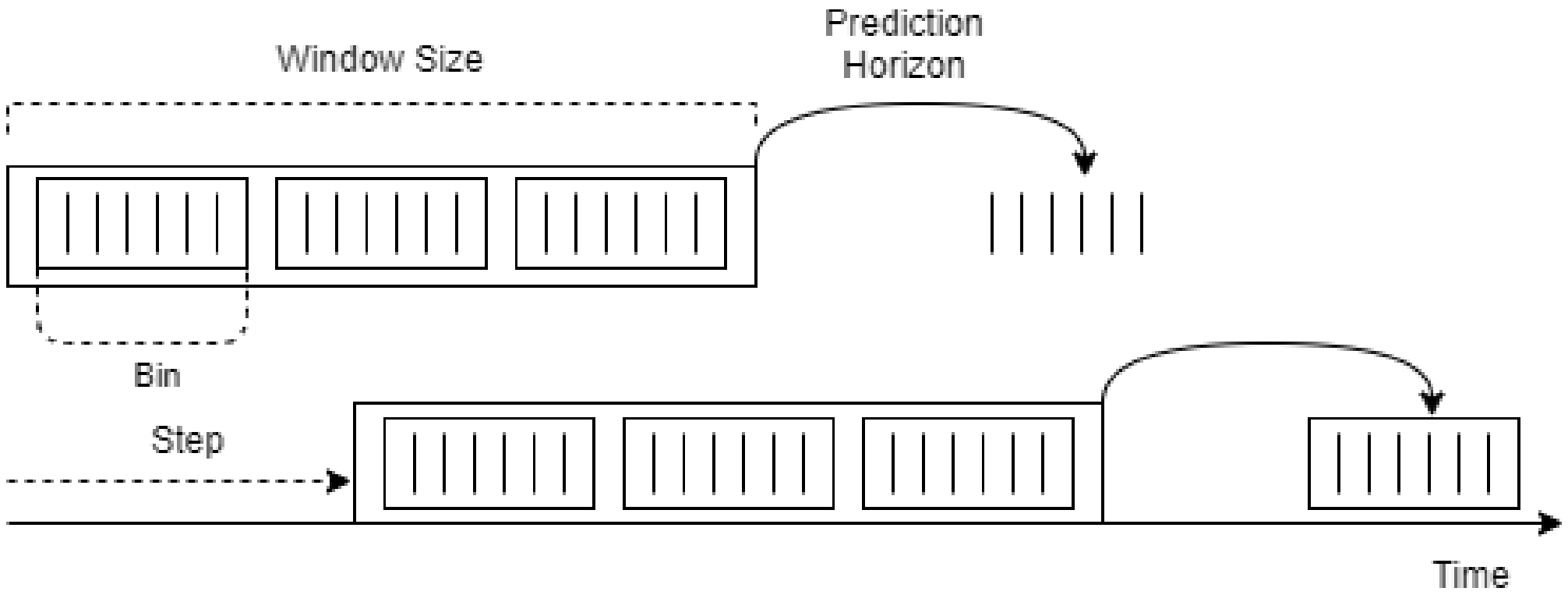
- return_sequences=True + return_state=True:
  - ✓all hidden states + last hidden state + last cell state



① Default: Last Hidden State (Hidden State of the last time step)

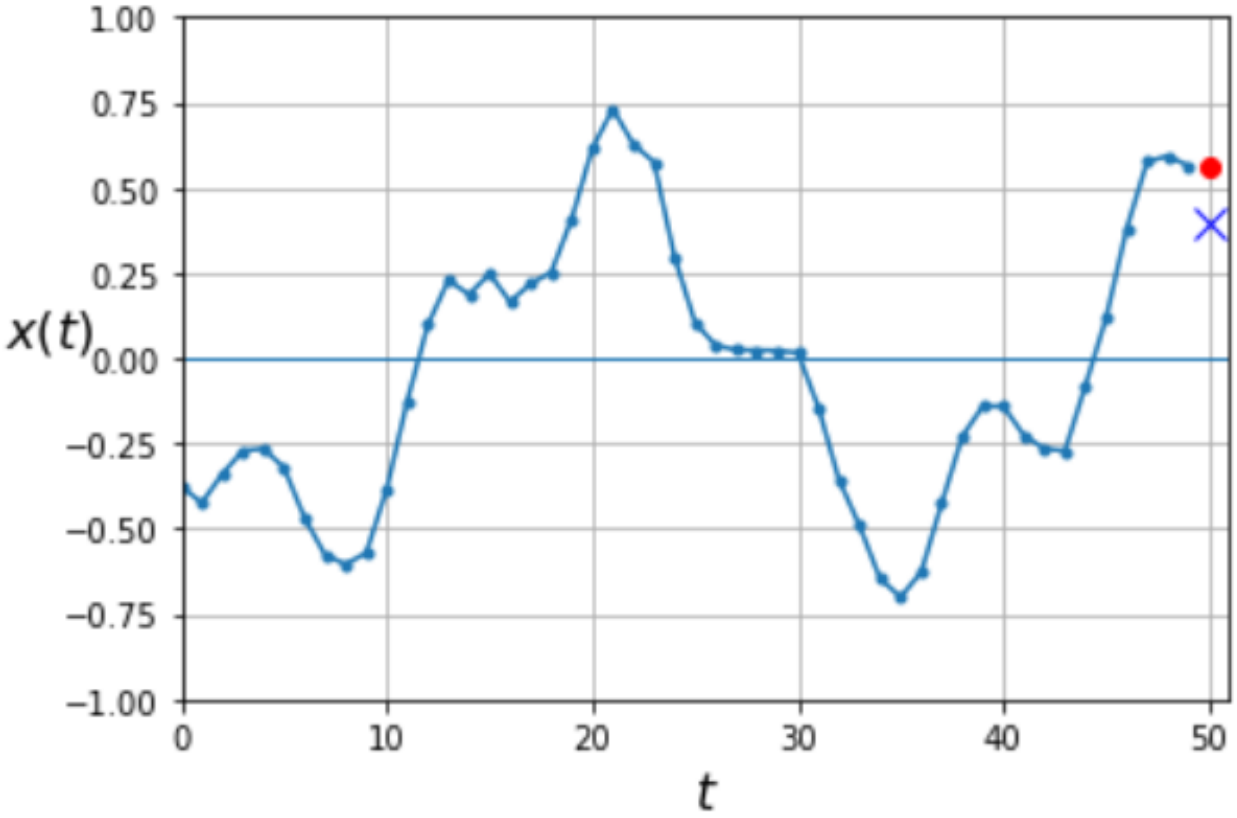


- Two terms are really important in the type of forecasting model :
  - ✓**Window Size** :The number of timesteps we take to predict into the future
  - ✓**Horizon** : The number of timesteps ahead into the future we predict.

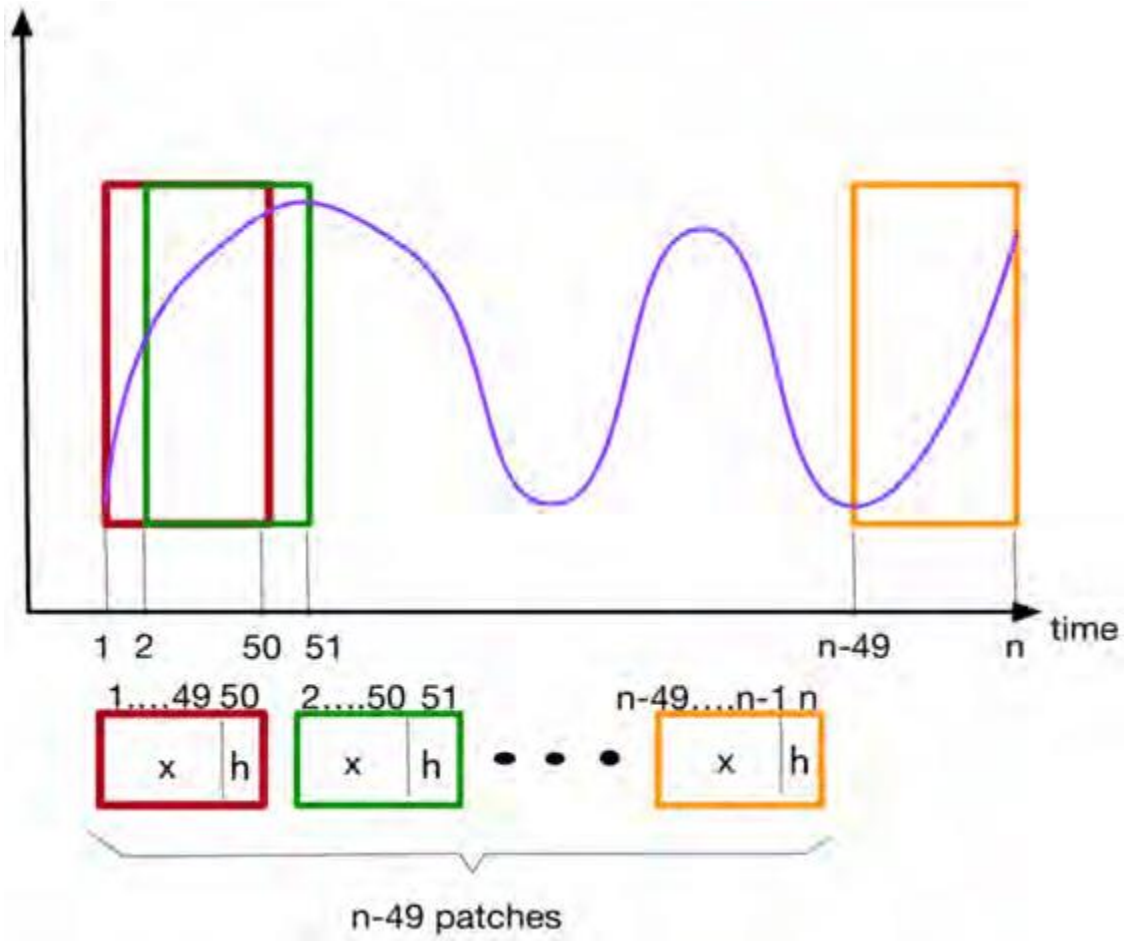


An example of sine function data

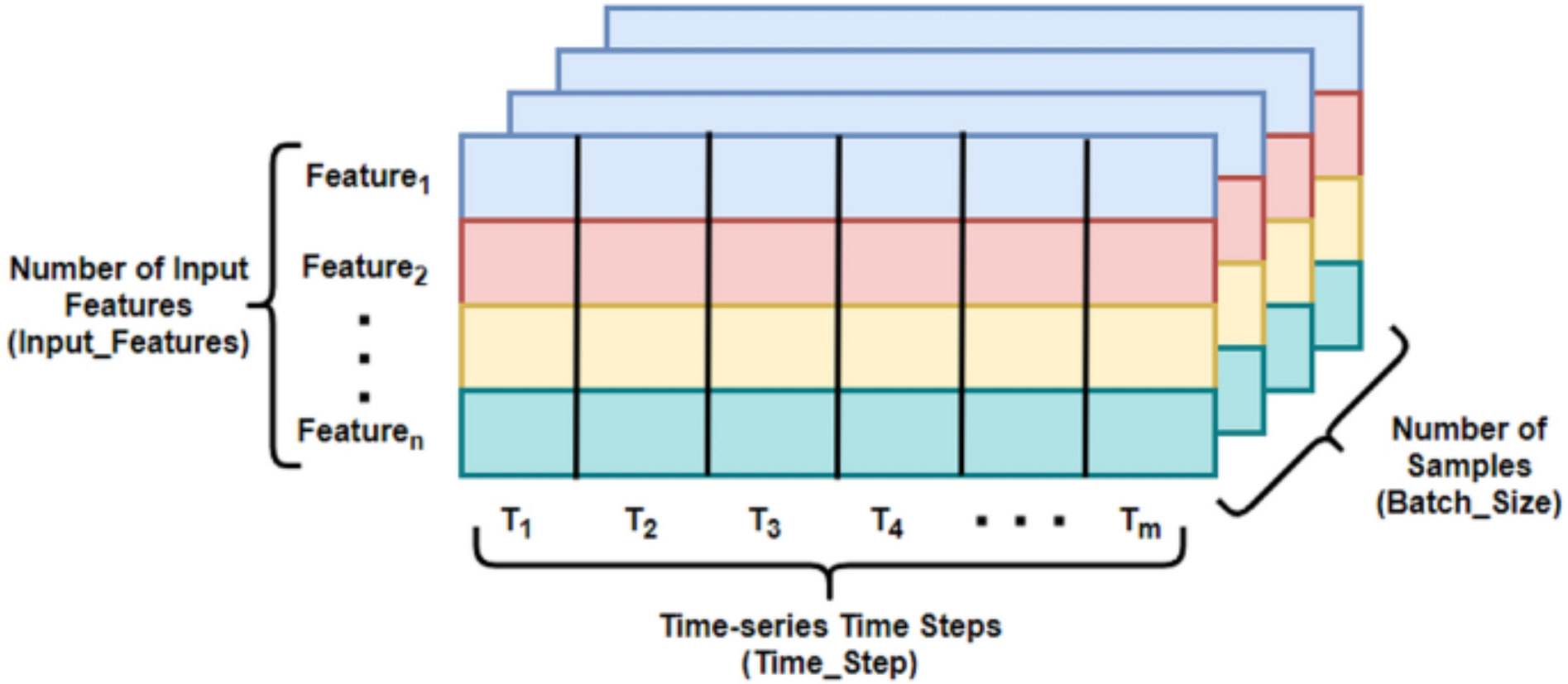
- Quizz
  - ✓window_size ?
  - ✓horizon ?



RNN Sliding window size and horizon

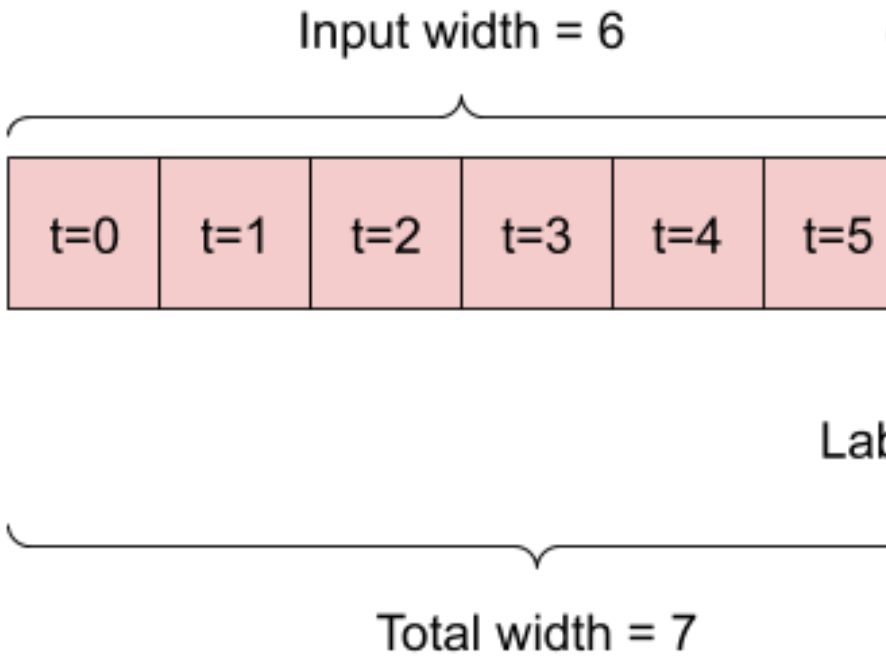


❖ RNN Input Data



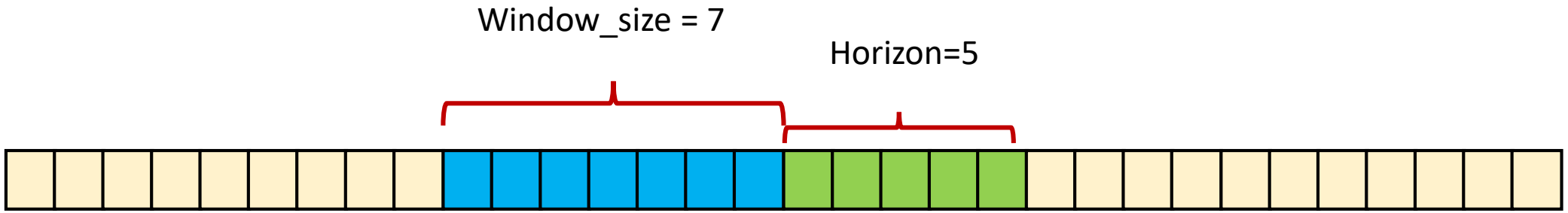
Data windowing

- A model : prediction one hour into the future, given six hours of history

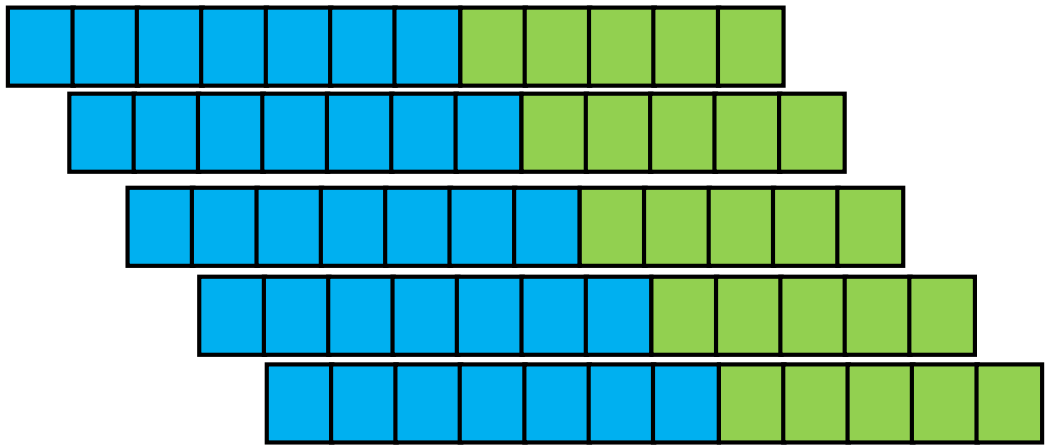


Many-to-One RNN Data Structure

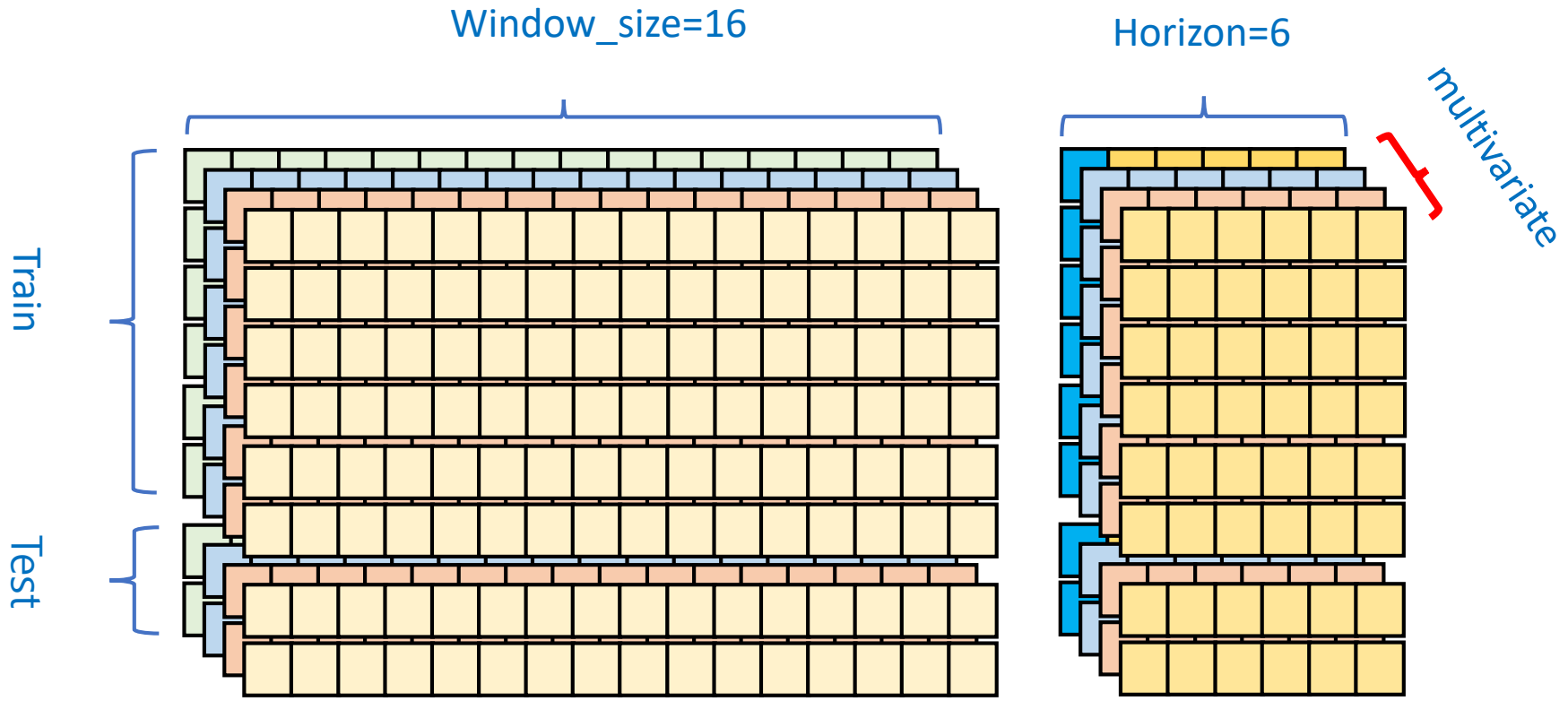
Step1: set the number of window_size, horizon



Step2: set the number of window_size, horizon

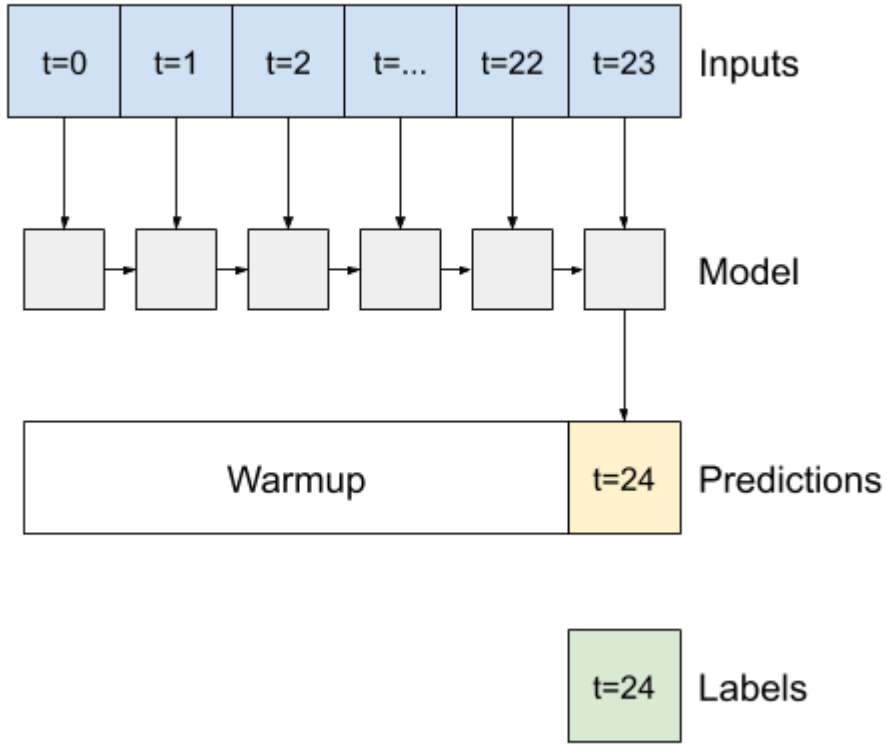


RNN Input-Output Data Structure



(Examples) return_sequences=False (the default)

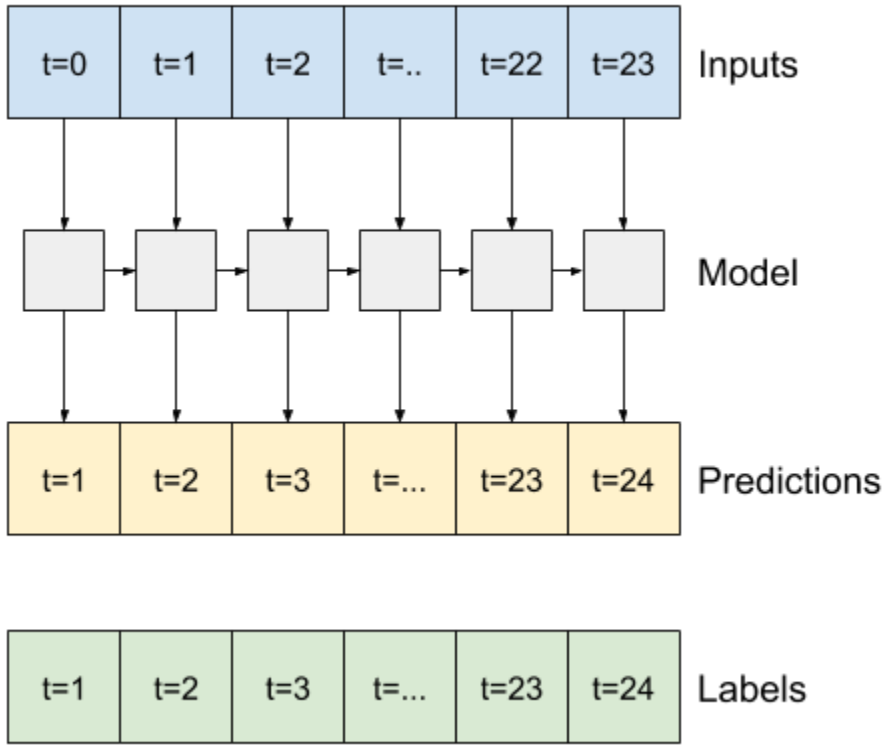
- the layer only returns the output of the final time step, giving the model time to warm up its internal state before making a single prediction:



(Examples) RNN: return_sequences=True

❖ Return_sequences=True

- ✓the layer returns an output for each input. This is useful for:Stacking RNN layers.
- ✓Training a model on multiple time steps simultaneously



[HW] Let's Code!

A Simple LSTM layer

임의의 입력 생성하기

```
1 import numpy as np
2 import tensorflow as tf
3 from tensorflow.keras.layers import SimpleRNN, GRU, LSTM, Bidirectional
4 train_X = [[0.1, 4.2, 1.5, 1.1, 2.8], [1.0, 3.1, 2.5, 0.7, 1.1], [0.3, 2.1, 1.5, 2.1, 0.1], [2.2, 1.4, 0.5, 0.9, 1.1]]
5 print(np.shape(train_X))
```

```
1 train_X=np.reshape(train_X, (-1,4,5))
2 print(train_X.shape)
```

```
(4, 5)
(1, 4, 5)
```

LSTM 이해

```
1 lstm = LSTM(3, return_sequences=False, return_state=True)
2 hidden_state, last_state, last_cell_state = lstm(train_X)
3
4 print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
5 print('last hidden state : {}, shape: {}'.format(last_state, last_state.shape))
6 print('last cell state : {}, shape: {}'.format(last_cell_state, last_cell_state.shape))
```

```
hidden state : [[ 0.00358916  0.41983268 -0.32303718]], shape: (1, 3)
last hidden state : [[ 0.00358916  0.41983268 -0.32303718]], shape: (1, 3)
last cell state : [[ 0.00439484  0.67286074 -0.4374103 ]], shape: (1, 3)
```

LSTM 이해

```
1 lstm = LSTM(3, return_sequences=True, return_state=True)
2 hidden_states, last_hidden_state, last_cell_state = lstm(train_X)
3
4 print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
5 print('last hidden state : {}, shape: {}'.format(last_hidden_state, last_hidden_state.shape))
6 print('last cell state : {}, shape: {}'.format(last_cell_state, last_cell_state.shape))
```

```
hidden states : [[[ 3.7191942e-02 -2.2934976e-03 -3.9808936e-03]
  [ 1.2157261e-01  1.6258408e-04 -1.3819224e-03]
  [ 2.6604503e-01 -1.4512378e-02  8.7545715e-02]
  [ 3.3192644e-01  6.2860921e-02  4.5289420e-02]]], shape: (1, 4, 3)
last hidden state : [[0.33192644 0.06286092 0.04528942]], shape: (1, 3)
last cell state : [[1.3158145 0.2671117 0.1030734]], shape: (1, 3)
```

GRU 이해하기

```
1  gru = GRU(3, return_sequences=True, return_state=True)
2  output, state = gru(train_X)
3
4  print(output)
5  print(state)
```

```
tf.Tensor(
[[[ 0.54427785 -0.8467631   0.56460524]
  [ 0.75815636 -0.9231671   0.7140679 ]
  [ 0.83681715 -0.8887498   0.74824893]
  [ 0.17147721 -0.43287683  0.7910696 ]]], shape=(1, 4, 3), dtype=float32)
tf.Tensor([[ 0.17147721 -0.43287683  0.7910696 ]], shape=(1, 3), dtype=float32)
```

KISTI SINCE 1962

2022 TRUST KISTI

Korea Institute of Science and Technology Information