KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information

2022

Deep Learning based Text Processing

Lec 09: Introduction to Recurrent Neural Network

hsyi@kisti.re.kr

Hongsuk Yi (이홍석)

강사소개

- ❖ 성명 : 이홍석
- ❖ 소속
  - ✓KISTI 데이터기반문제해결연구단 책임연구원
  - ✓과학기술연합대학원(UST-KISTI) 응용AI전공 교수
- ❖ 연구내용
  - ✓딥러닝기반 도심지 교통혼잡 예측 (2018~2021)
  - ✓ 지능형 인프라 기술 연구 (2018~2019)
  - ✓도심지 교통예측을 위한 트랜스포머 모델 연구(2022~2025)
- ❖ 컴퓨팅 기술
  - ✓GPU 기반 가속컴퓨팅 기술 : CUDA, OpenCL
  - ✓슈퍼컴퓨팅 병렬처리 기술 : MPI, OpenMP, ManyCore Computing
  - ✓Deep Learning 기술 : 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:

LSTM

Overview of Course (I)

- **Basic knowledge of NLP**
  - ✓Natural language processing concepts and its applications
- **Word Embedding Theory**
  - ✓NPLM, Word2Vec, FastTex
- **Word Embedding - Practice 1**
  - ✓Word embedding based on Word2Vec
- **Word Embedding - Practice 2**
  - ✓Word embedding and its Applications

Overview of Course (II)

- The concept of machine learning and deep learning
  - ✓practice of libraries for deep learning (Numpy, Pandas)
- Understanding regression problems with FCN
  - ✓Regression problem practice (housing price prediction)
- Understanding classification problems with FCN
  - ✓Classification problem practice (diabetes onset prediction)
- Text Analysis Practice with FCN
  - ✓Text classification with movie review

Overview of Course (III)

- **Introduction to Recurrent Neural Network**
  - ✓Simple RNN, BPTT, Memory Cell
  - ✓Code: Implementing an RNN with Keras
- **Introduction to Long-Short Term Memroy**
  - ✓Cell state, LSTM, and GRU, and Applications
  - ✓A Visual Guide to Recurrent Layers in Keras
  - ✓Code: A simple LSTM layers
- **Text generation with RNN**
  - ✓Tokenizer, Character-Level Language model
  - ✓Code: Alice's Adventures in Wonderland
- **Sequence to Sequence Learning model with RNN**
  - ✓Introduction to Seq2Seq and Attention model
  - ✓Code: Character-Level Neural Machine Translation

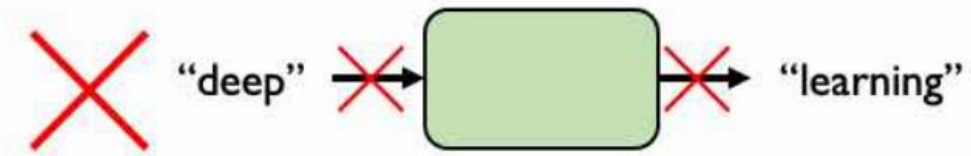
A Sequence Modeling Problem: Predict the Next Word

"This morning I took the dog for a walk."

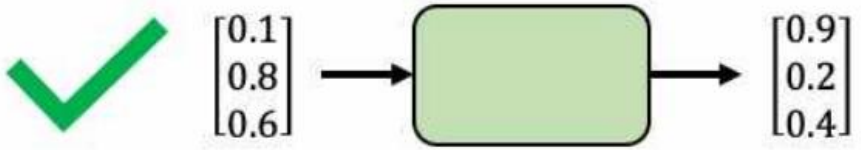
given these words

predict the next word

Representing Language to a Neural Network



Neural networks cannot interpret words



Neural networks require numerical inputs

Trial : Use a Fixed Window

given these two words

"This morning I took the dog **for a** walk." *predict the next word*

One hot feature encoding : tells us what each word is

for a [1 0 0 0 0 0 1 0 0 0] ⟶ prediction

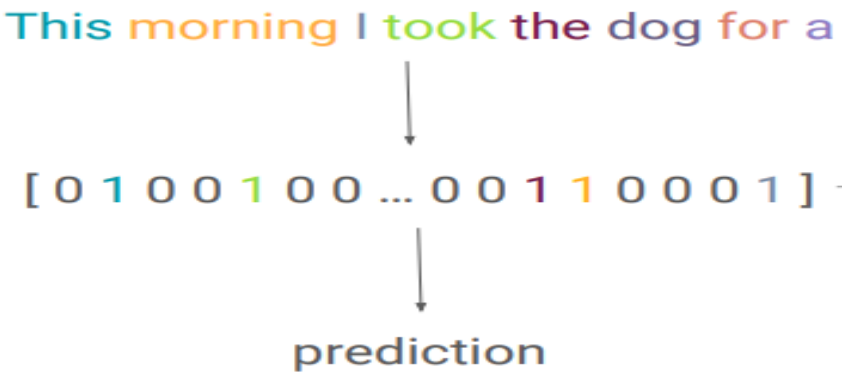
Problem : Can't Model Long-Term Dependencies

"France is where I grew up, but I now live in Boston. I speak fluent _____."

We need information from the distant past to accurately predict the current word.

Trial: Use Entire Sequence as Set of Counts

- Bag-of-words model: a text is represented as the bag of its words.



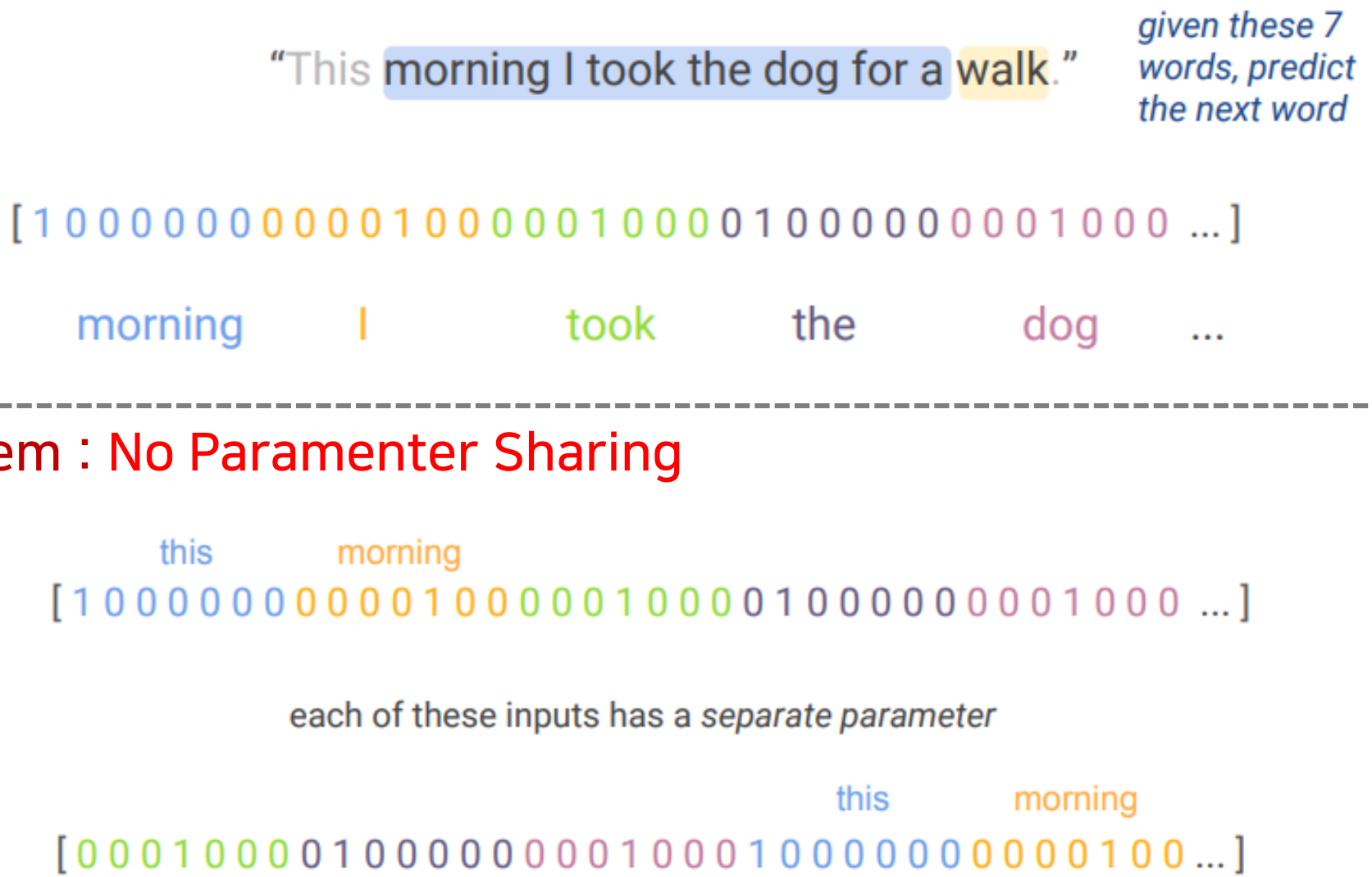
Problem : Counts Don't Preserve Order

"The food was good, not bad at all."

vs

"The food was bad, not good at all."

Trial : Use a Really Big Fixed Window



Problem : No Paramenter Sharing

Sequence Modeling: Design Criteria

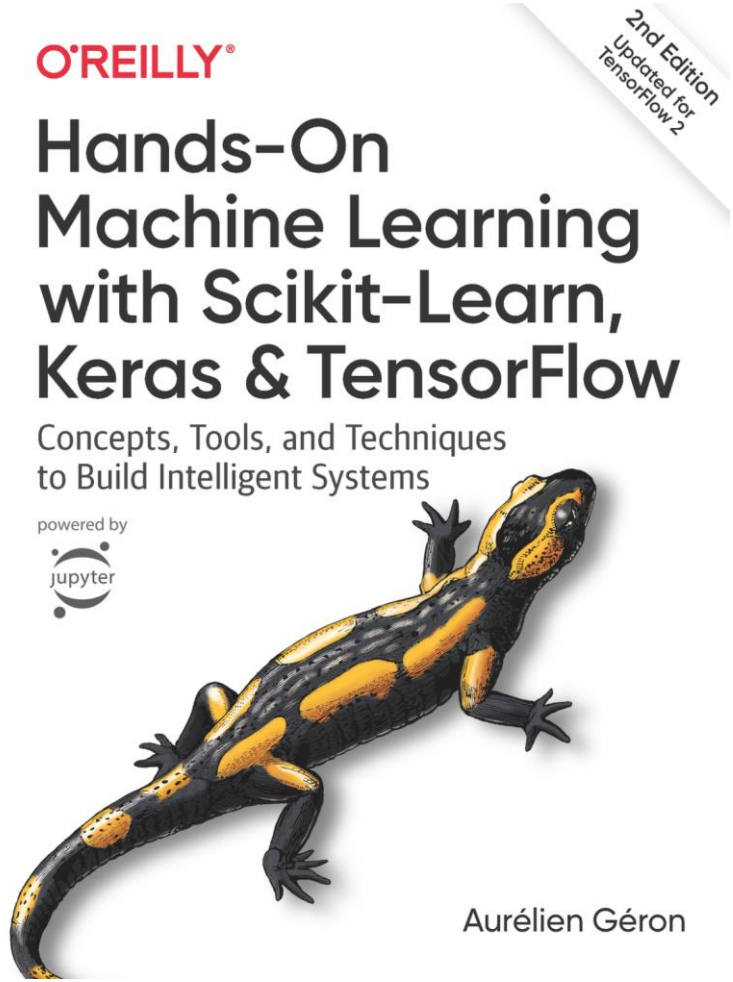
- Design criteria to model sequences, we need:

1. to deal with variable-length sequences
2. to maintain sequence order
3. to keep track of long-term dependencies
4. to share parameters across the sequence

RNN meet these sequence modeling design critiera

Introduction to Recurrent Neural Network

Reference Materials

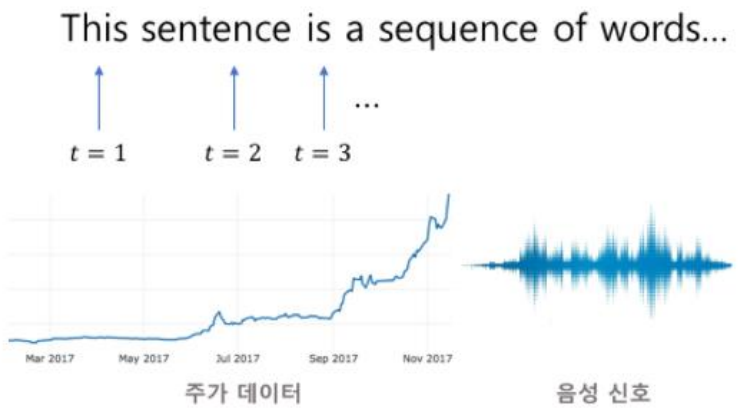


딥 러닝을 이용한 자연어 처리 입문

https://wikidocs.net/book/2155

What is a time series problem?

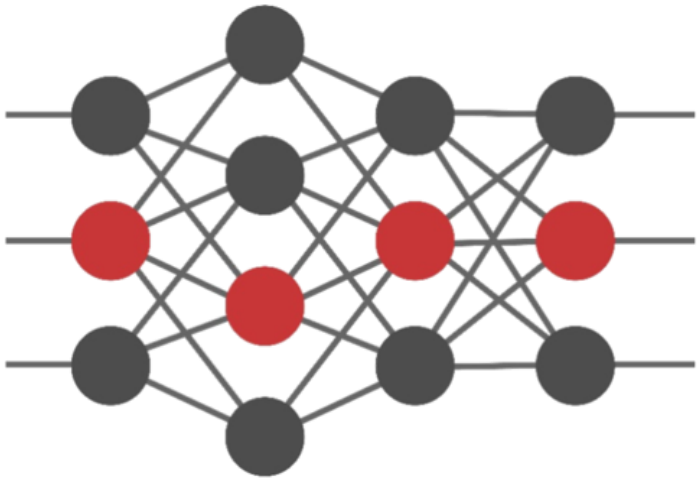
- Time series deals with data over time:
  - ✓Predicting
    - weather (short term forecasts), climate (long term forecasts)
    - sales accross different geographies for different products in the catalog



Why Not Feedforward Networks

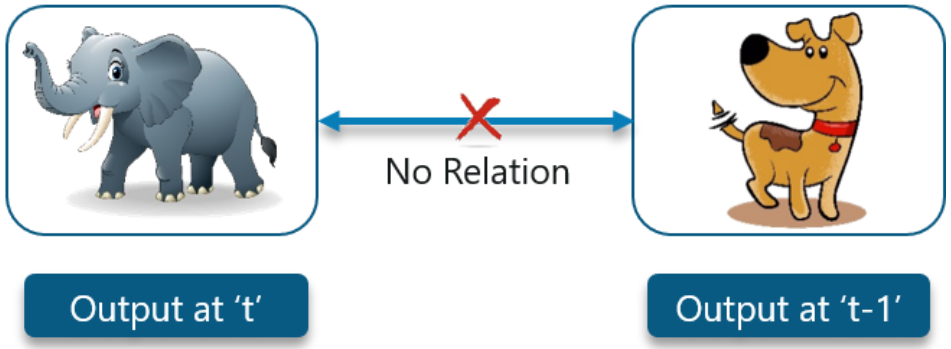
In the existing FeedForWared Networks,

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.

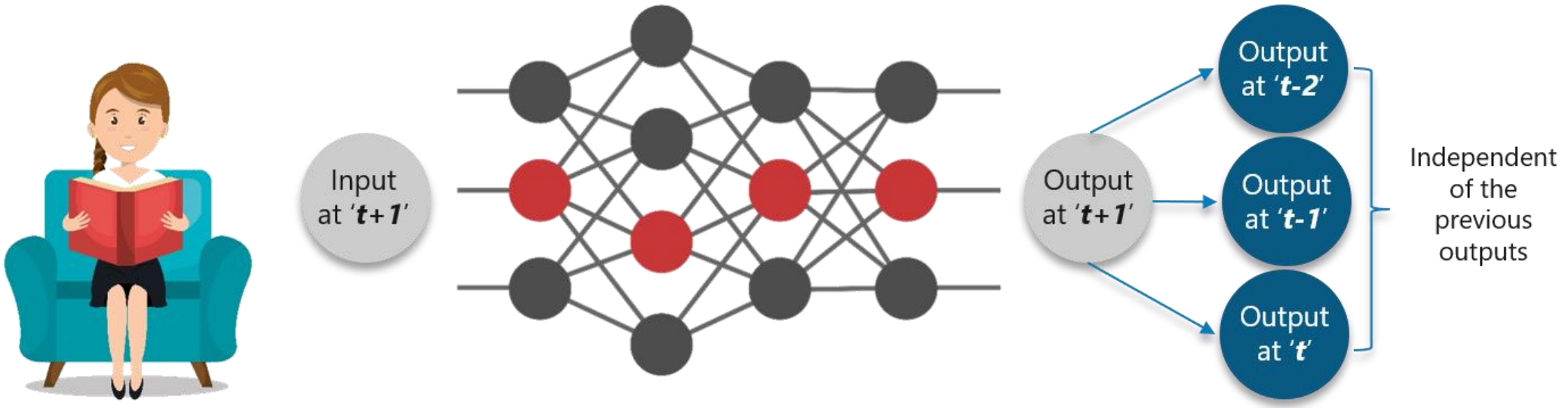


(2) The next elephant image is input to output an elephant (label).

Why Not Feedforward Networks

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

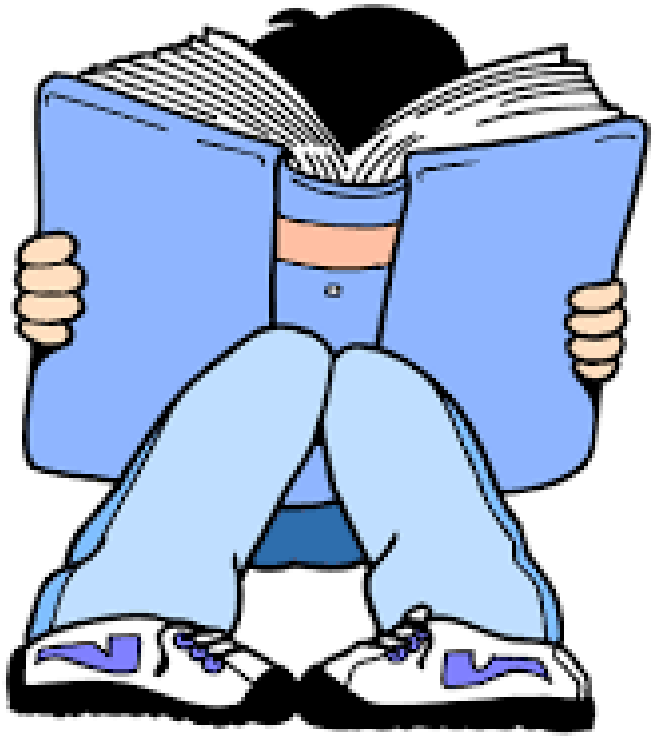
If you are reading a book, you should be well aware of the previous page.



RNN Memory Size

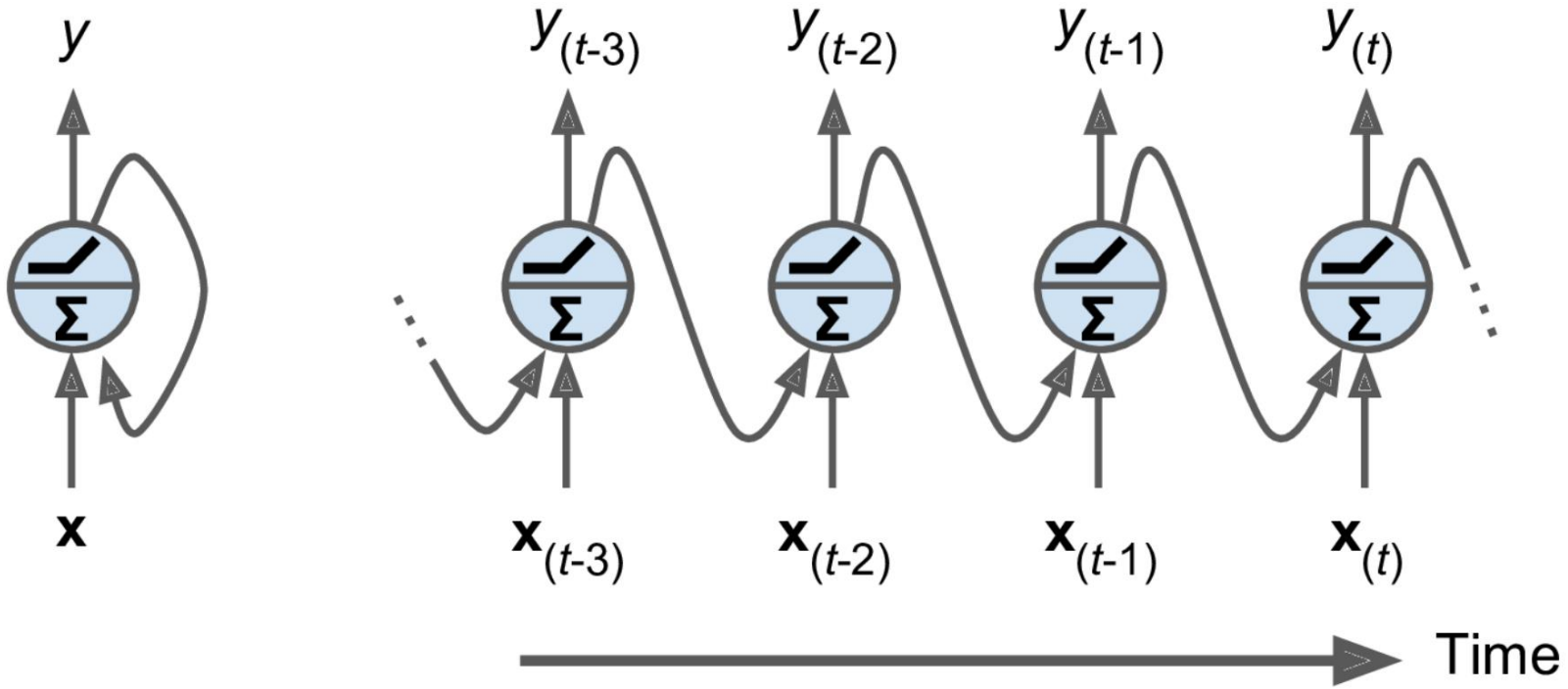
❖ What is Memory Size?

- ✓1 page
- ✓10 pages
- ✓100 pages
- ✓200 pages



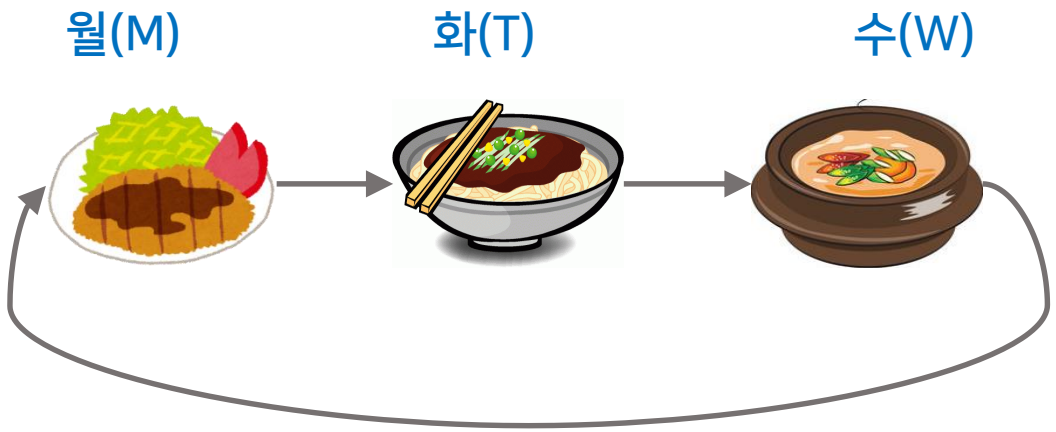
❖ Memory Cells ~ a form of memory

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



A simple of RNN (1/3)

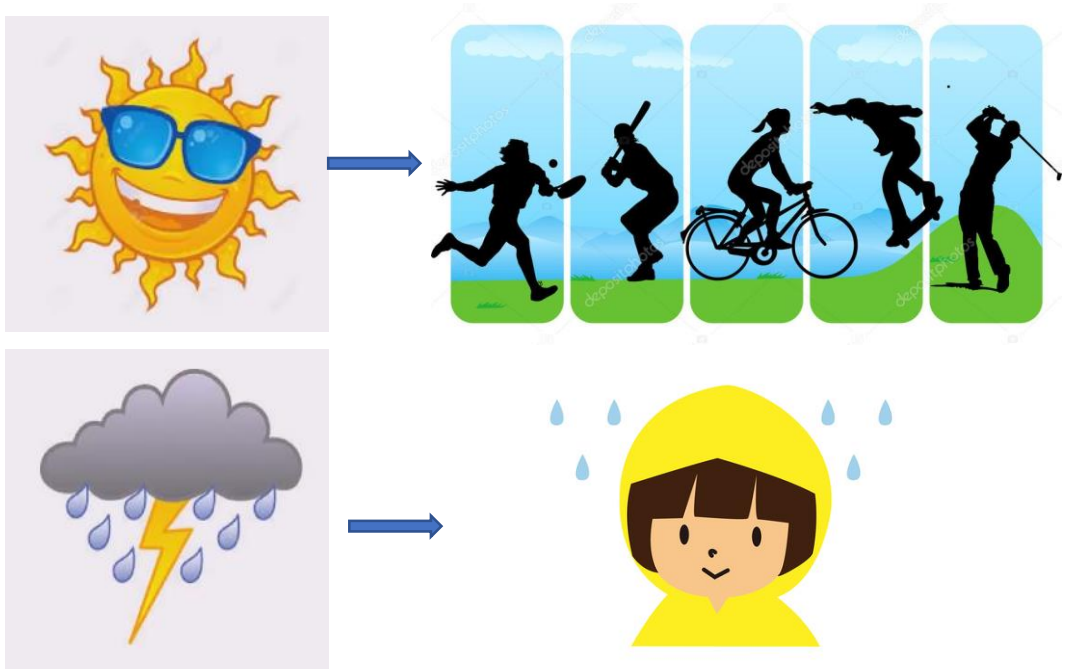
- There is a pattern in dinner.
  - ✓Only three types of meals are available.



RNN = Recurrent Neural Network

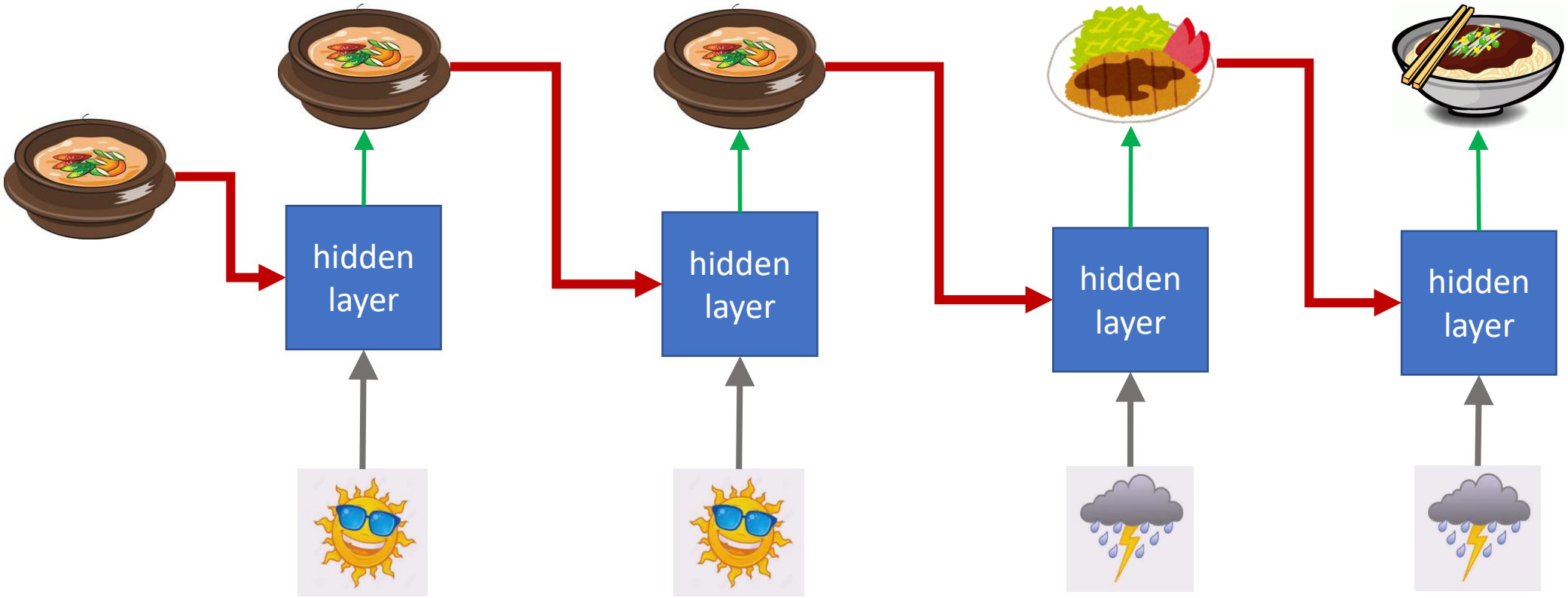
A simple of RNN (2/3)

- 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



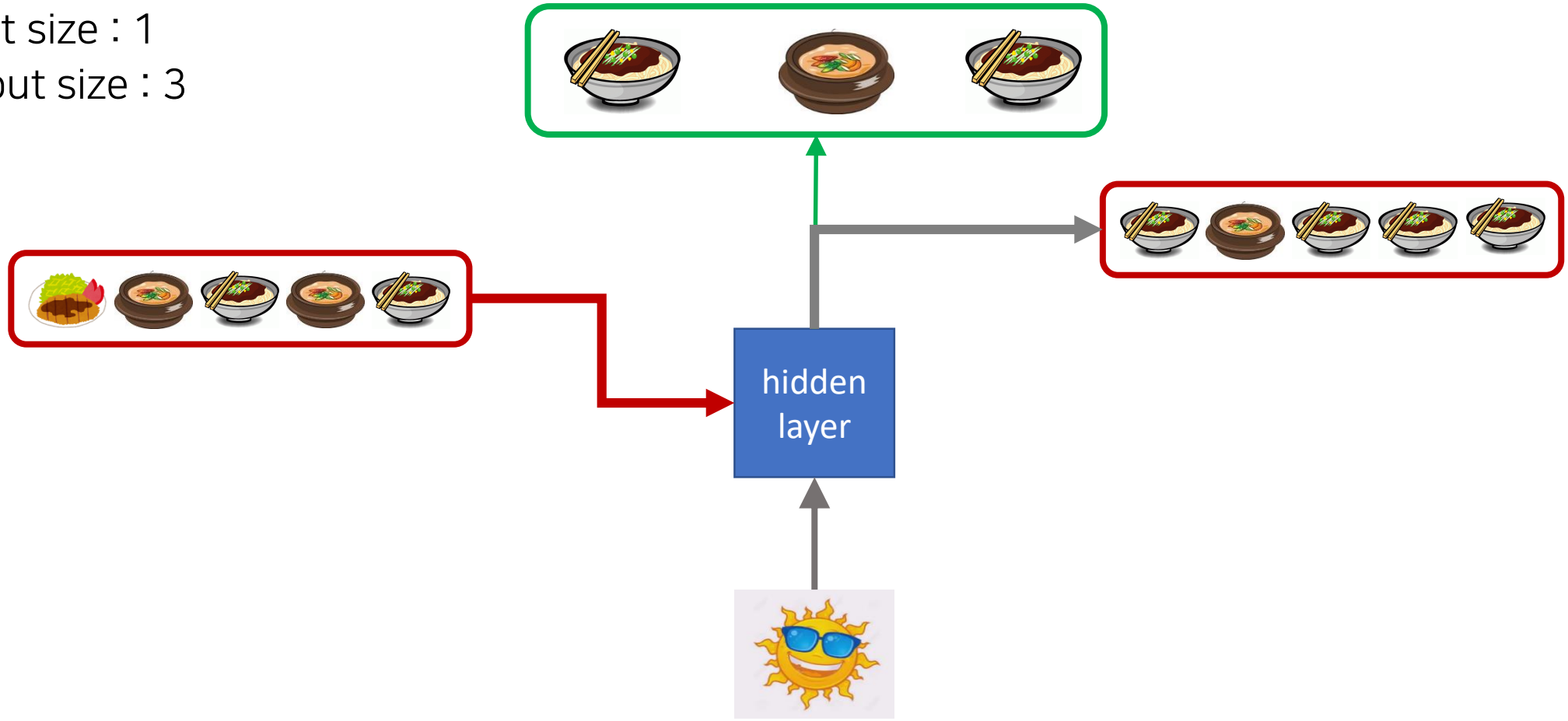
A simple of RNN (3/3)

- You can predict food depending on the weather



❖ A example of Length of memory

- ✓memory size: 5
- ✓input size : 1
- ✓output size : 3



Output of a recurrent layer for a single instance

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

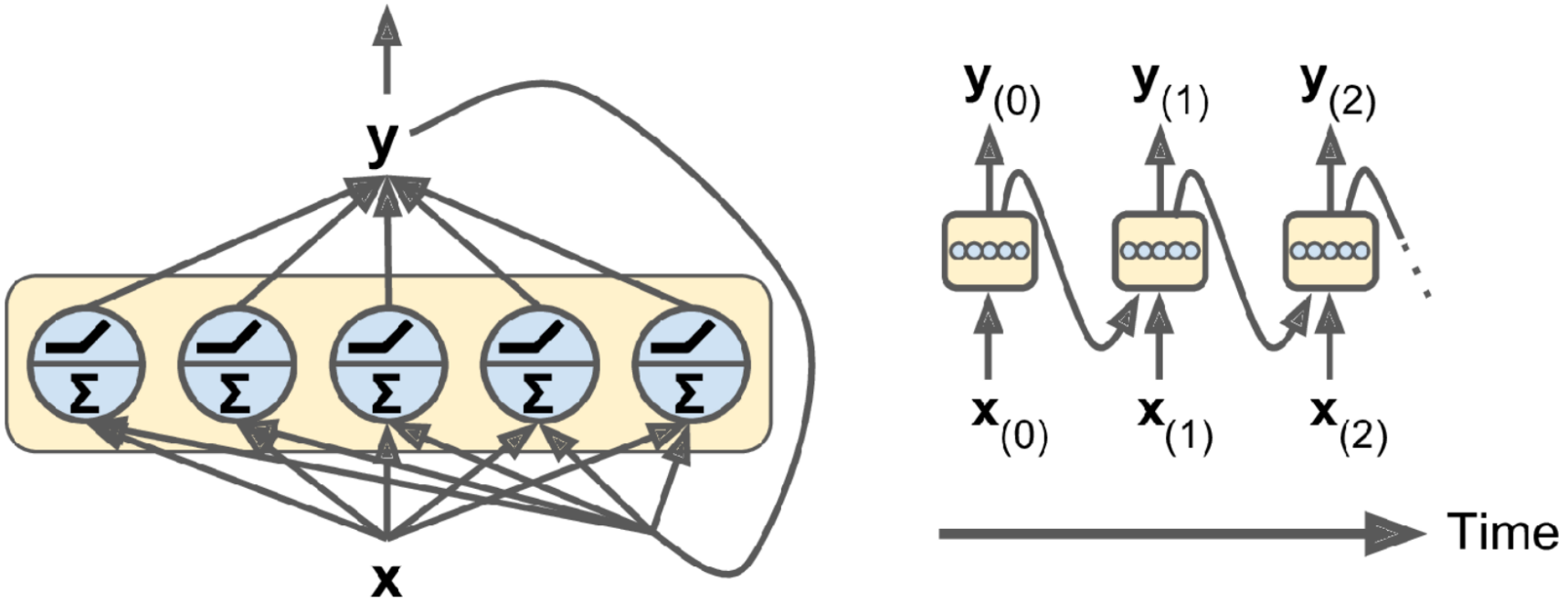
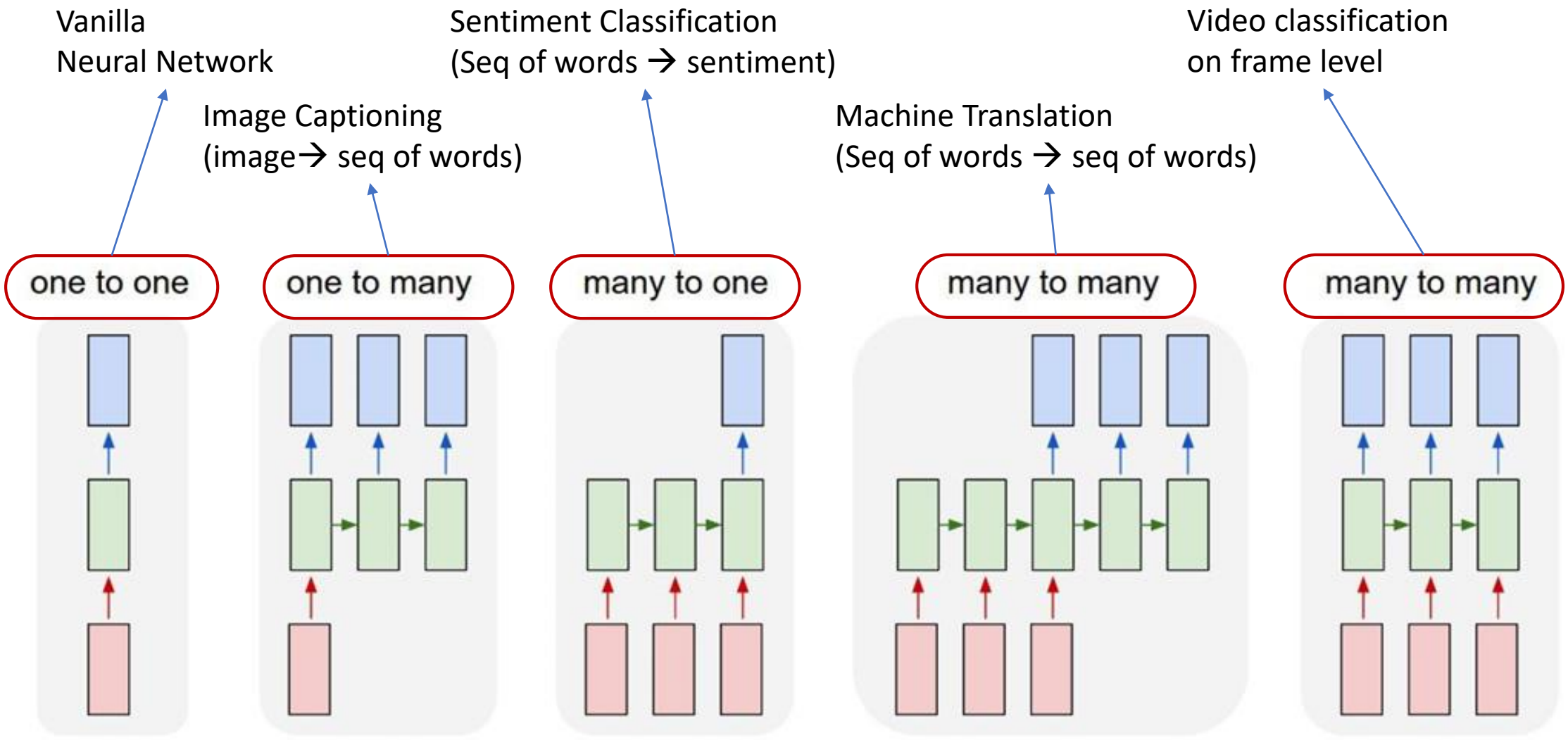


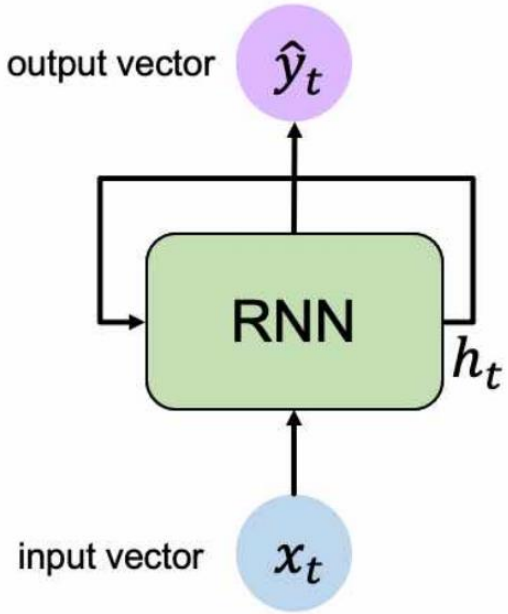
Figure 15-2. A layer of recurrent neurons (left) unrolled through time (right)

Recurrent Neural Networks: Process Sequences



How to train RNN?

RNN State Update and Output



Output Vector

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

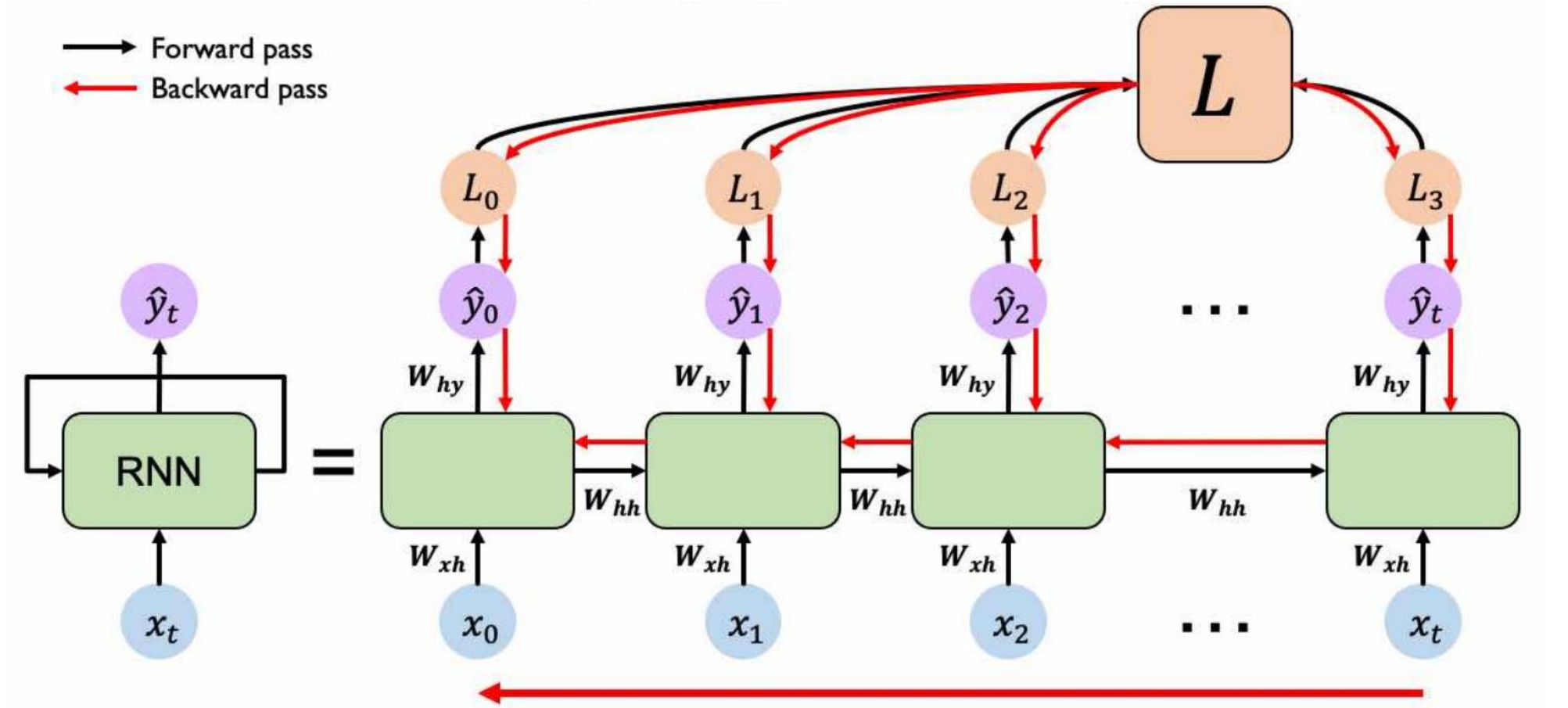
Update Hidden State

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

Input Vector

$$x_t$$

BPTT: Backpropagation Through Time



Standard RNN Gradient Flow

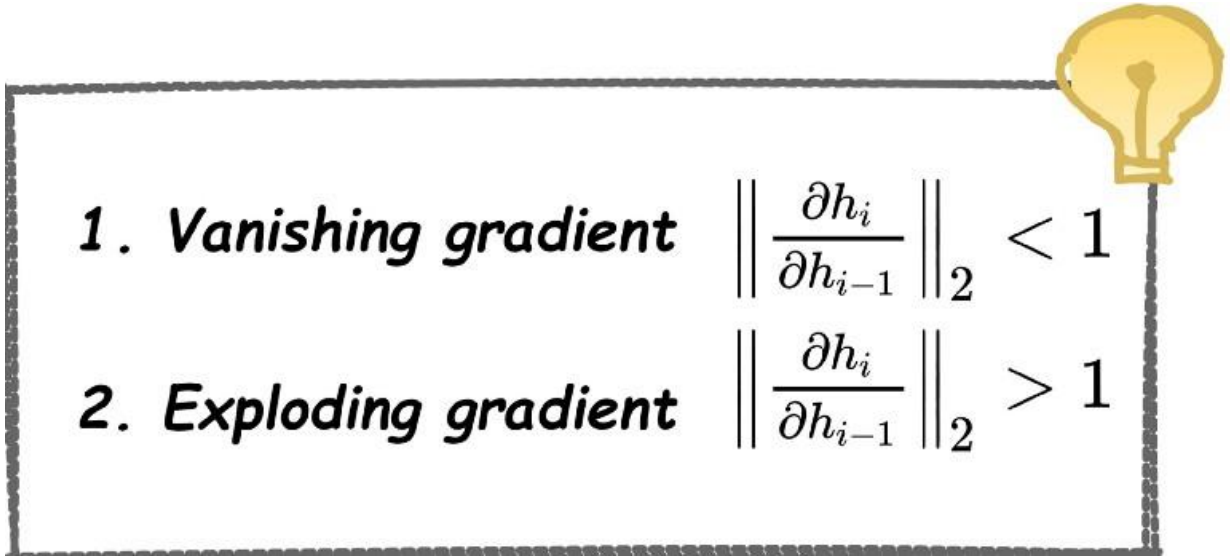
$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left(\sum_{k=1}^{t} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}}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

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

- VGP(Vanishing Gradient Problem)
  - ✓ when the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.
- EGP(Exploding Gradient Problem)
  - ✓ If the slope tends to grow exponentially instead of decaying



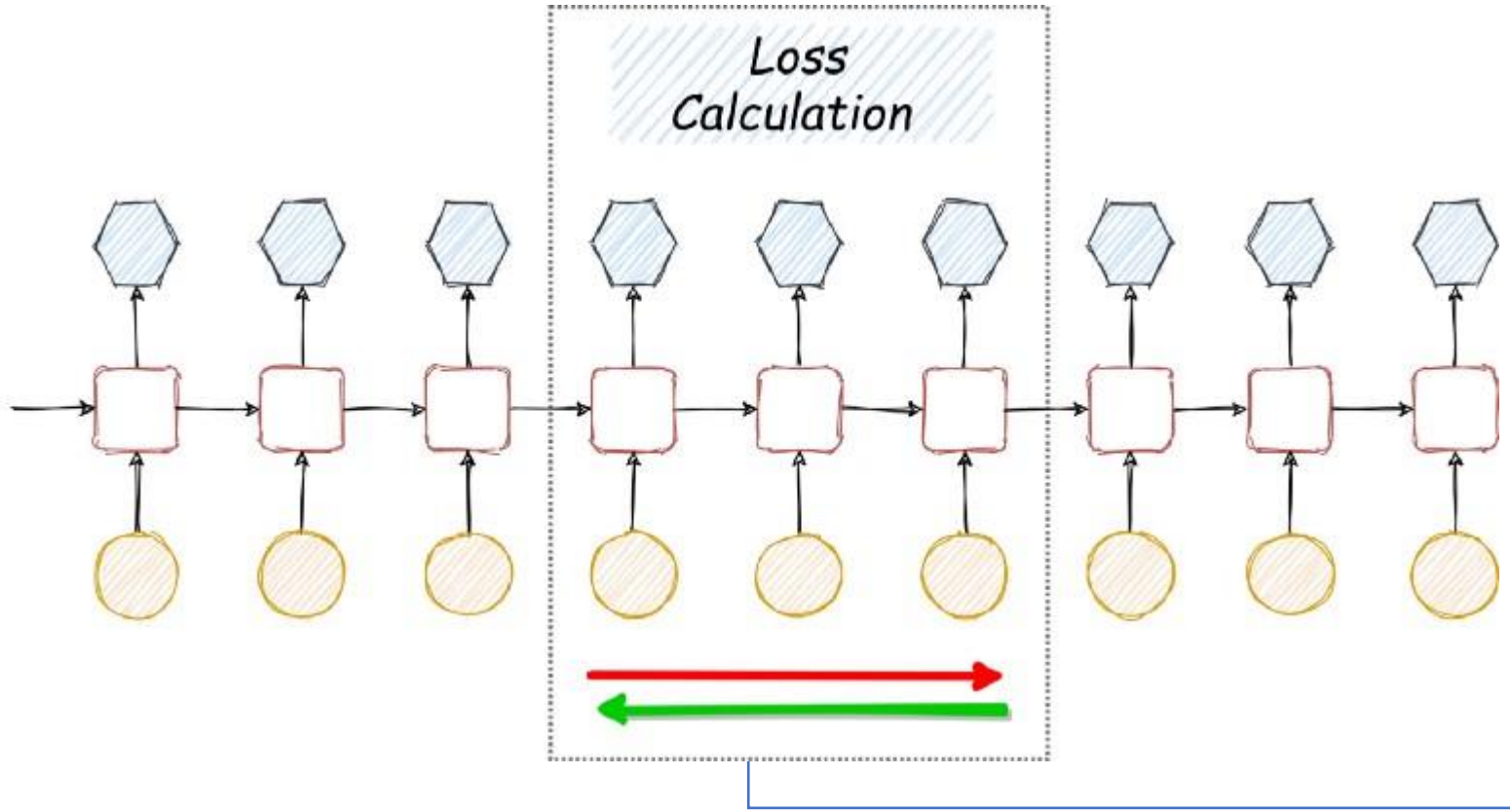
Solutions to the vanishing/exploding gradient problems

- **Truncated backpropagation through time (TBPTT)**
  - ✓ simply limits the number of time steps the signal can backpropagate after each forward pass.
    - E.g., even if the sequence has 100 elements/steps, we may only backpropagate through 20 or so
- **Long short-term memory (LSTM)**
  - ✓ uses a memory cell for modeling long-range dependencies and avoid vanishing gradient problems
- **Gradient Clipping**
  - ✓ set a max value for gradients if they grow to large
    - solves only exploding gradient problem)

(1) TBPTT: Truncated Backpropagation Through Time

❖ TBPTT

✓ Truncated BPTT trick tries to overcome the VGP by considering a moving window through the training process

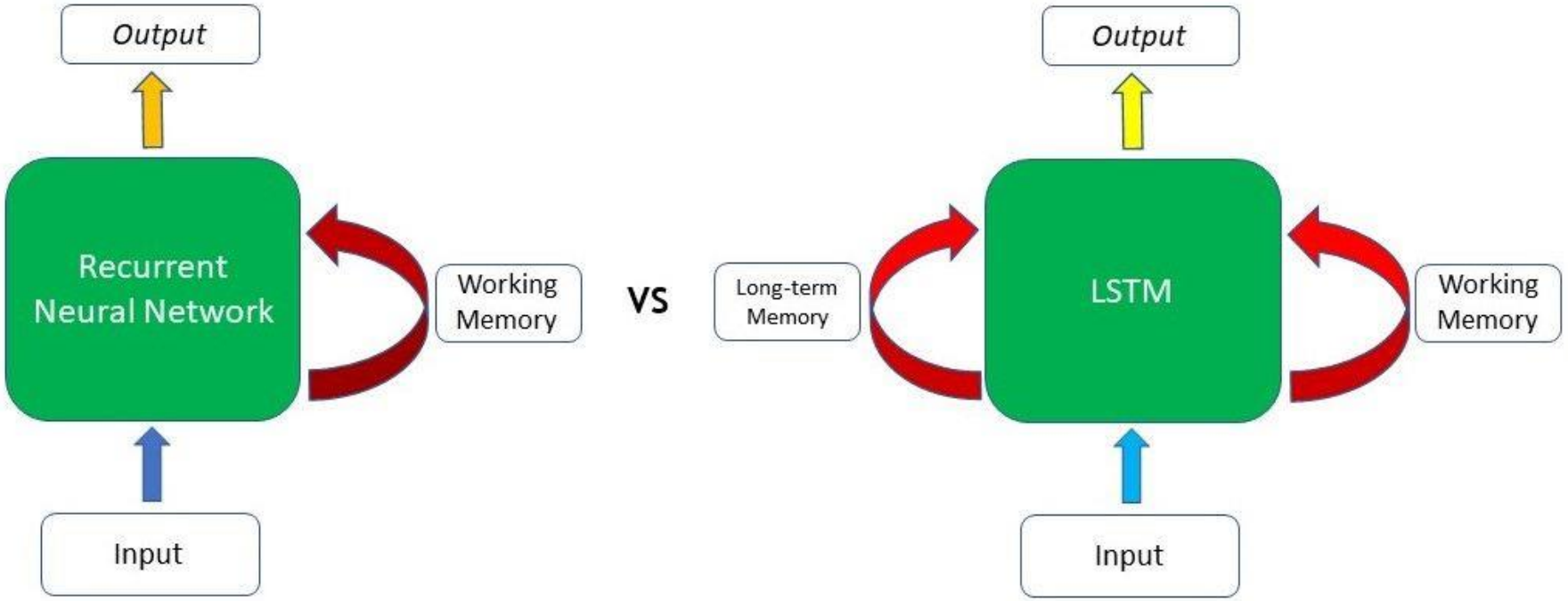


Run forward and backward through chuncks of the sequence instead of whole sequence

Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps.

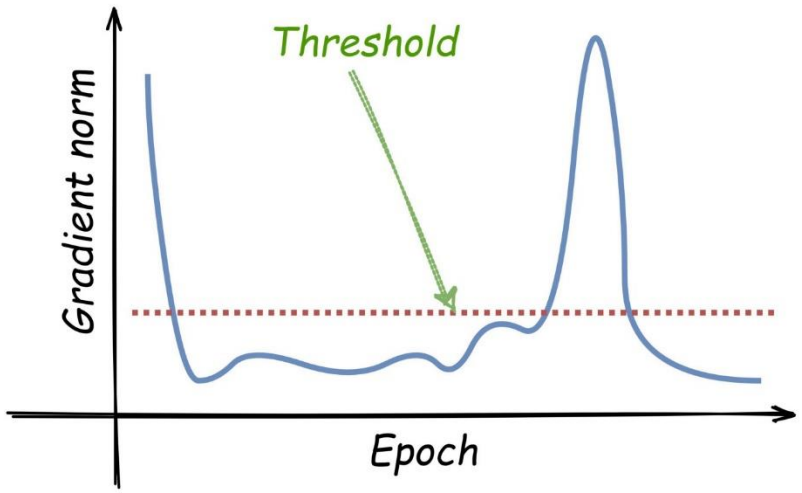
(2) Long Short-Term Memory networks (LSTMs)

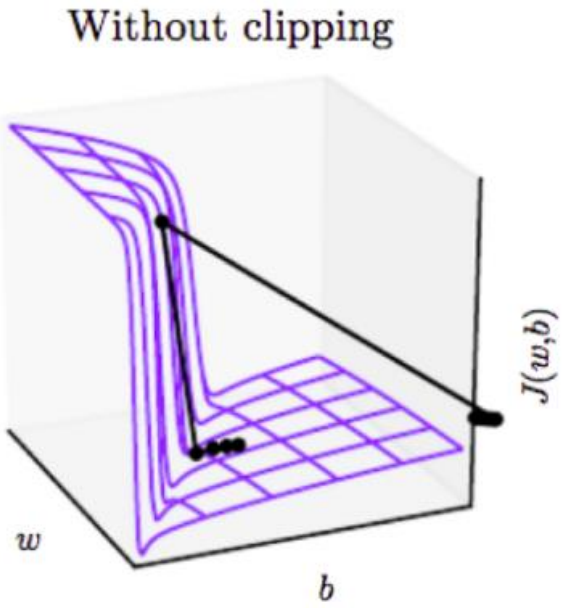
- Allows learning of long-term dependencies
  - ✓A RNN addresses the vanishing/exploding gradient problem
  - ✓Capable of learning long-term dependencies by remembering information

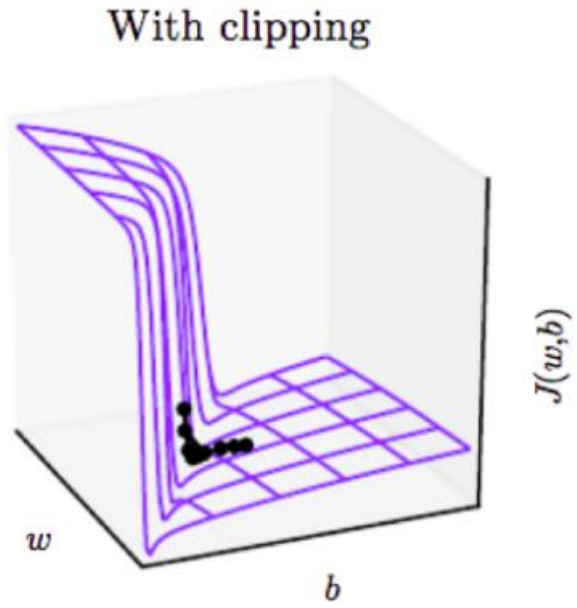


(3) Gradient Clipping

$$
\begin{aligned}
&\textbf{if } \|\hat{\mathbf{g}}\| \geq threshold \textbf{ then} \\
&\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}} \\
&\textbf{end if}
\end{aligned}
$$





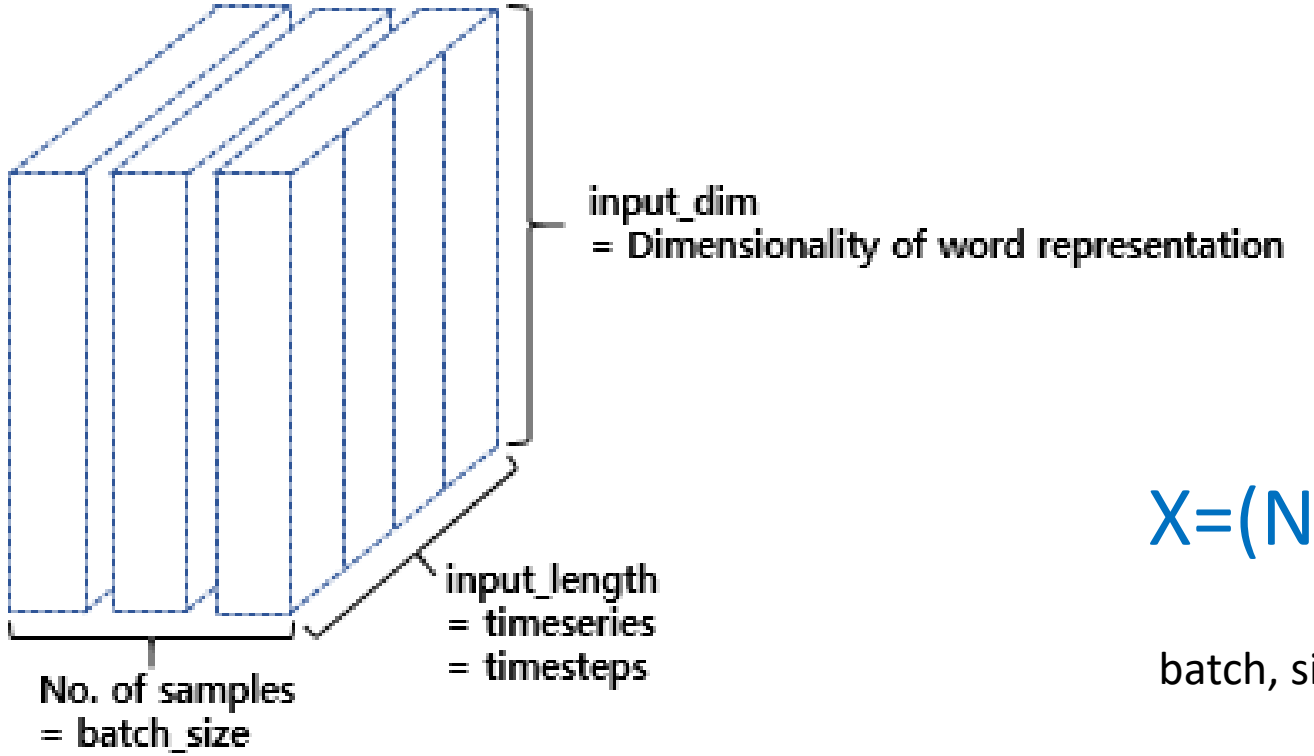


source: https://eehoeskrap.tistory.com/582

Implementing an RNN with Keras

RNN Input : X Dimension

- The RNN layer receives a 3D tensor of size as input
  - ✓batch_size, timesteps, and input_dim

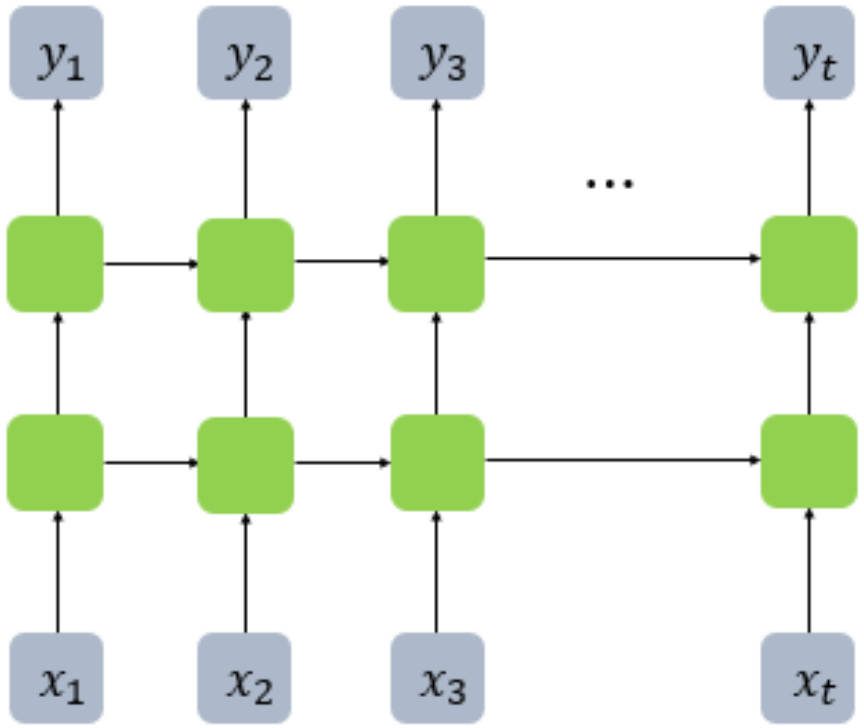


X=(None, 2, 10)

batch, size of memory, input_dim)

A example: Deep Recurrent Neural Network

model.**add**(SimpleRNN(hidden_units, input_length=10, input_dim=5, return_sequences=True))



Deep RNN with 2 hidden layers

SimpleRNN without Batch (1/4)

output_dim : (batch_size, hidden_units)

```
model.add(SimpleRNN(3, input_shape=(2,10)))
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 3)                 42
=================================================================
Total params: 42
Trainable params: 42
Non-trainable params: 0
```

X=(None, 2, 10)

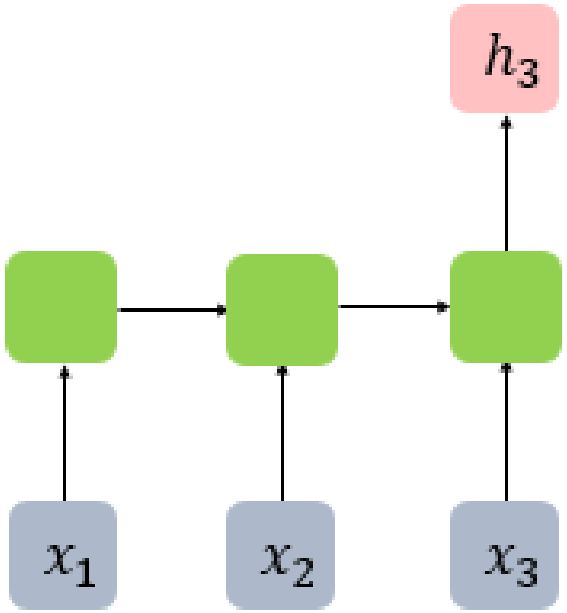
SimpleRNN with batch size (2/4)

```
# with Batch size = 8
model.add(SimpleRNN(3, batch_input_shape=(8,2,10)))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (8, 3)                    42
=================================================================
```

❖ Setting the return_sequences of the RNN layer

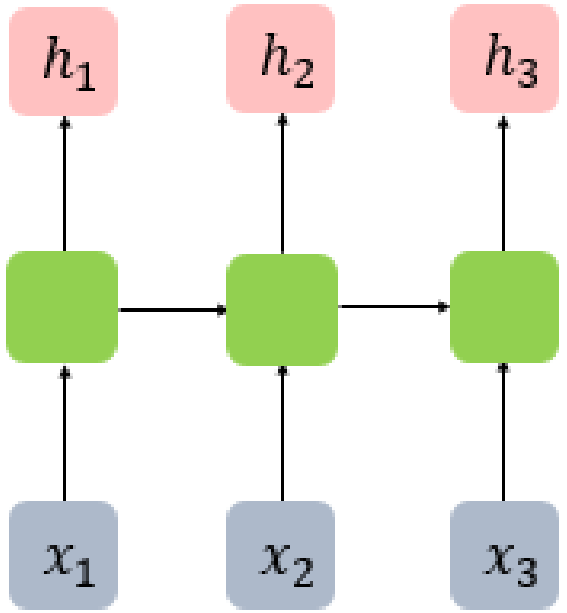
return_sequences=False



다음층으로 마지막 은닉 상태만 전달

many-to-one task

return_sequences=True



다음층으로 모든 은닉 상태 전달

many-to-many task

SimpleRNN : with batch and return_sequence=True (4/4)

```
model.add(SimpleRNN(3, batch_input_shape=(8,2,10), return_sequences=True))
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_2 (SimpleRNN)    (8, 2, 3)                 42

=================================================================
```

[HW] Let's Code!
Simple RNN

Input Data and Train Data Shape

```
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)
```

SimpleRNN and Hidden states

```
1 rnn = SimpleRNN(3)
2 # rnn = SimpleRNN(3, return_sequences=False, return_state=False)와 동일.
3 hidden_state = rnn(train_X)
4
5 print('hidden state : {}, shape: {}'.format(hidden_state, hidden_state.shape))
```

```
hidden state : [[ 0.9206875  -0.9963834  -0.99492633]], shape: (1, 3)
```

```
1 rnn = SimpleRNN(3, return_sequences=True)
2 hidden_states = rnn(train_X)
3
4 print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
```

```
hidden states : [[[-0.91097903 -0.99876505 -0.95570076]
  [ 0.9831649  -0.52204293 -0.94400156]
  [-0.7801549   0.03199867 -0.44760188]
  [ 0.8878632  -0.20588721  0.7986243 ]]], shape: (1, 4, 3)
```

SimpleRNN and Return Sequence

```
1 rnn = SimpleRNN(3, return_sequences=True, return_state=True)
2 hidden_states, last_state = rnn(train_X)
3
4 print('hidden states : {}, shape: {}'.format(hidden_states, hidden_states.shape))
5 print('last hidden state : {}, shape: {}'.format(last_state, last_state.shape))
```

```
hidden states : [[[0.99741    0.5095164  0.9999336 ]
  [0.97162277 0.32083237 0.99995947]
  [0.95783615 0.86253214 0.99940664]
  [0.99748254 0.7658603  0.9743646 ]]], shape: (1, 4, 3)
last hidden state : [[0.99748254 0.7658603  0.9743646 ]], shape: (1, 3)
```

Simple RNN and Return_Sequence

```
1 rnn = SimpleRNN(3, return_sequences=False, return_state=True)
2 hidden_state, last_state = rnn(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))
```

```
hidden state : [[0.9096651 0.9990103 0.7855833]], shape: (1, 3)
last hidden state : [[0.9096651 0.9990103 0.7855833]], shape: (1, 3)
```

KISTI SINCE 1962

2022 TRUST KISTI

Korea Institute of Science and Technology Information