KiSTi 한국과학기술정보연구원 Korea Institute of Science and Technology Information

2022

Deep Learning based Text Processing

Lec 10: Introduction to Recurrent Neural Network

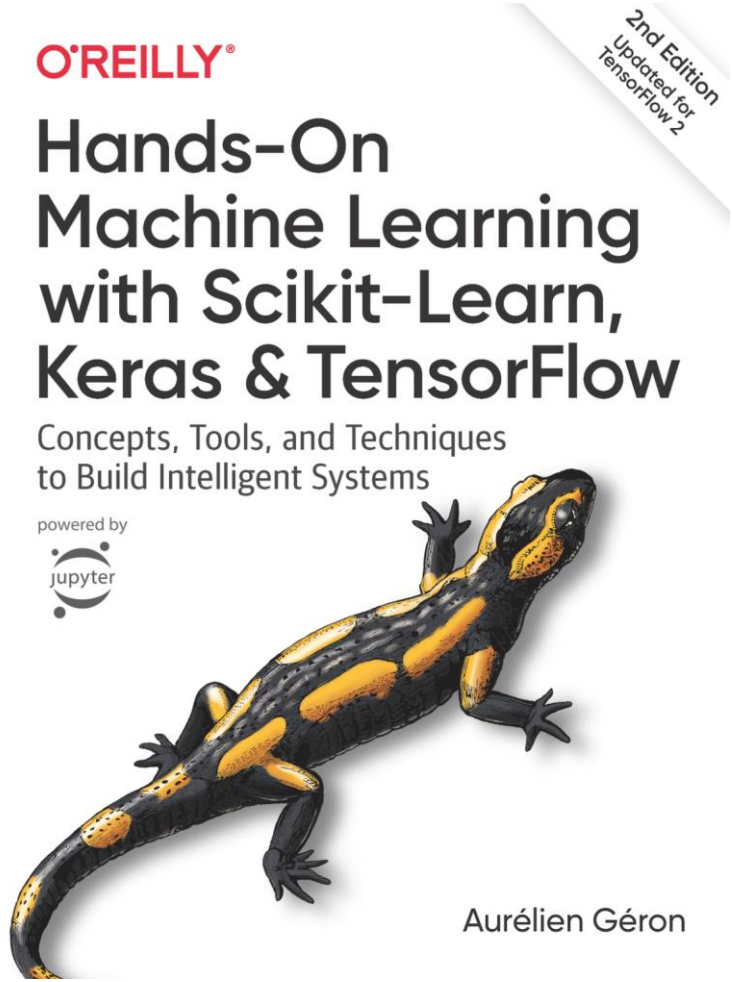
hsyi@kisti.re.kr

Hongsuk Yi (이홍석)

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

Reference Materials



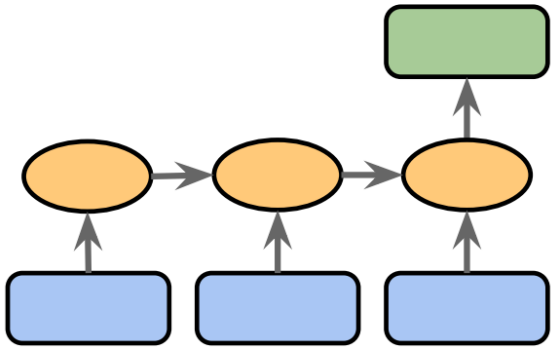
딥 러닝을 이용한 자연어 처리 입문

https://wikidocs.net/book/2155

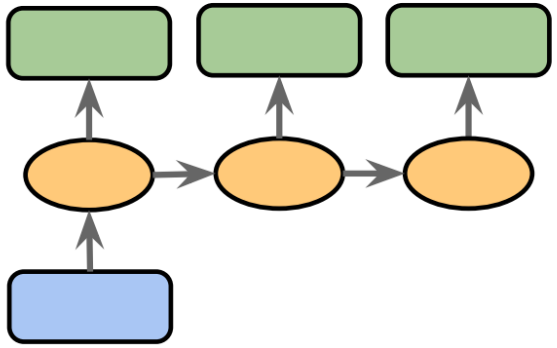
Reviewing the last class:

LSTM

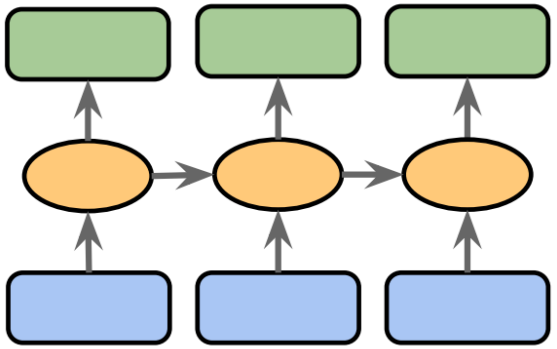
Different Types of Sequence Modeling Tasks



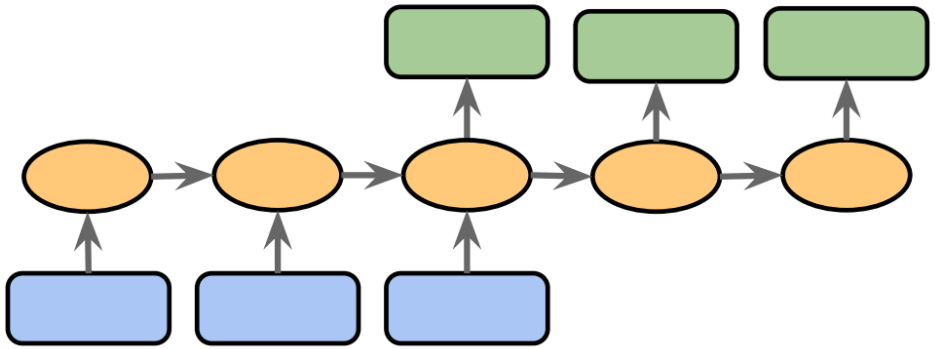
many-to-one



one-to-many



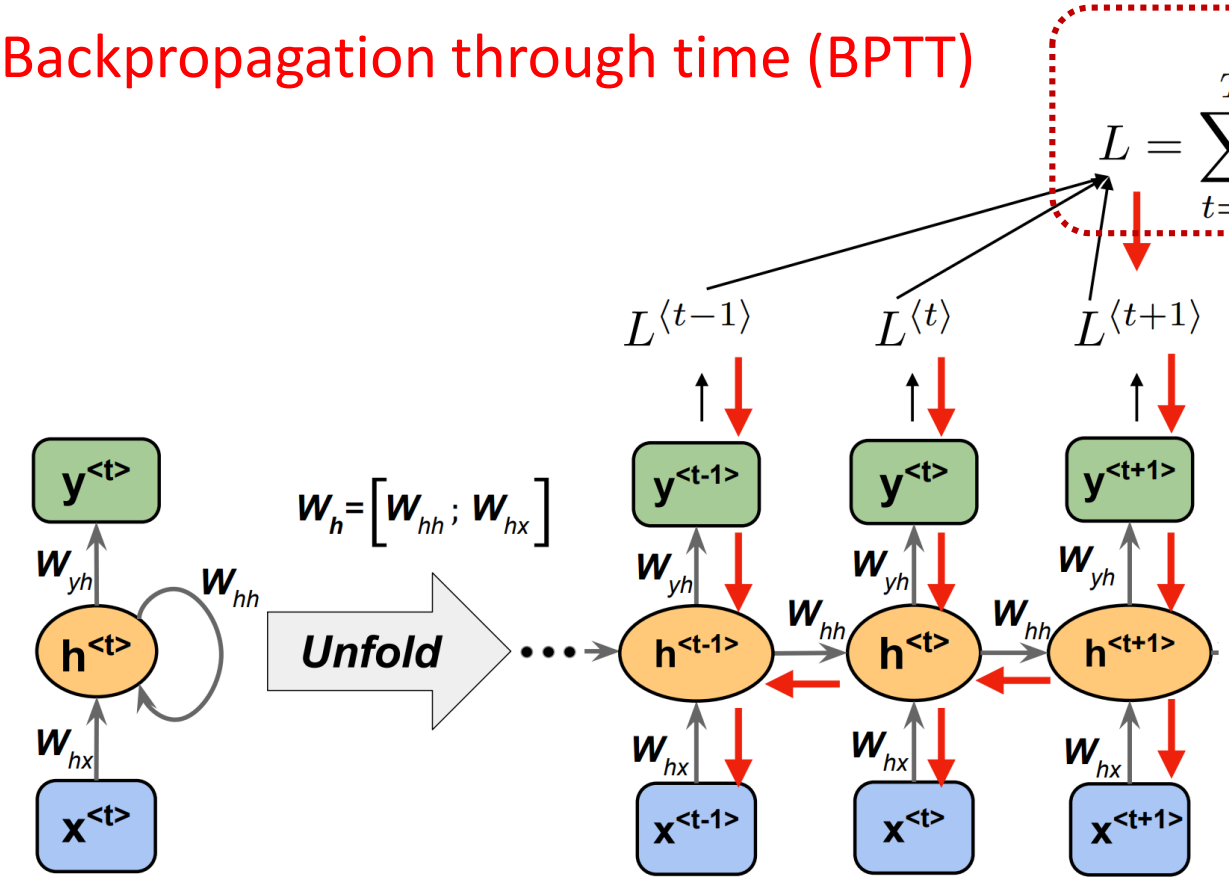
many-to-many



many-to-many

- ✓To train an RNN the trick is to unroll it through time and then simply use regular backpropagation.

Backpropagation through time (BPTT)



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

computed as a multiplication of adjacent time steps:

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

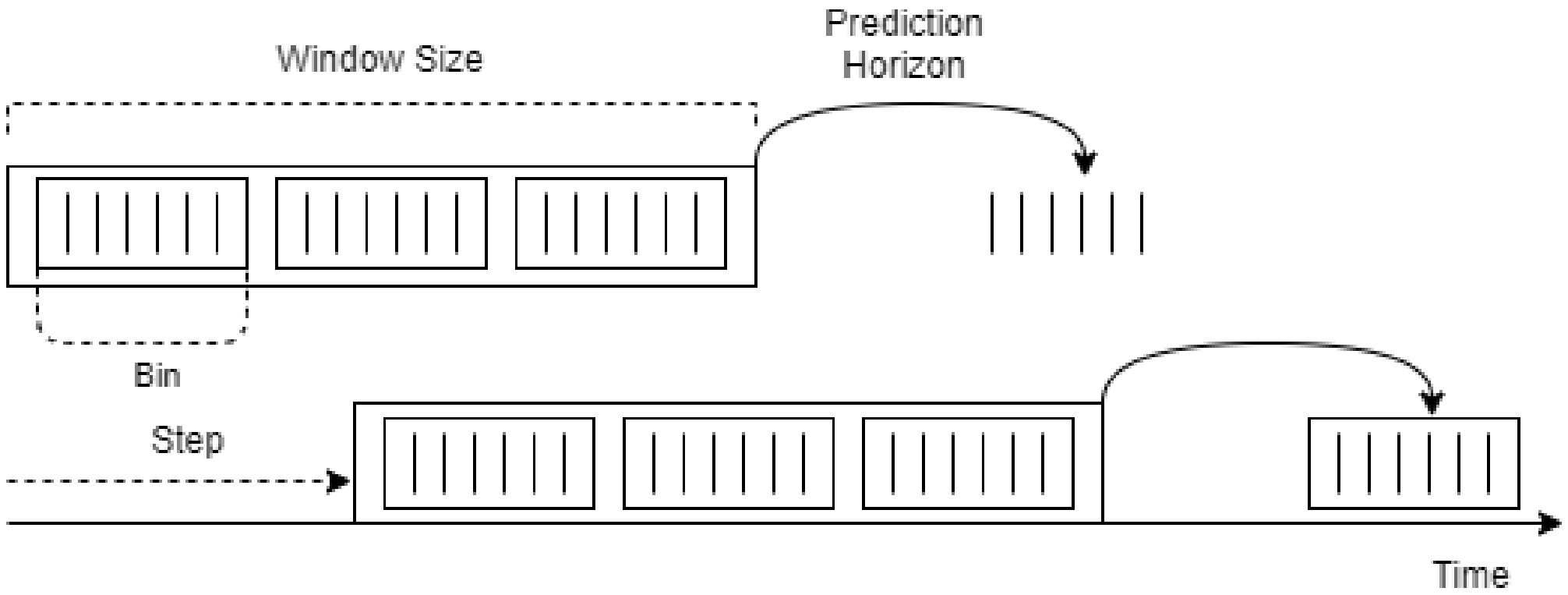
Solutions to the vanishing/exploding gradient problems

KISTI SINCE 1962

- **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)

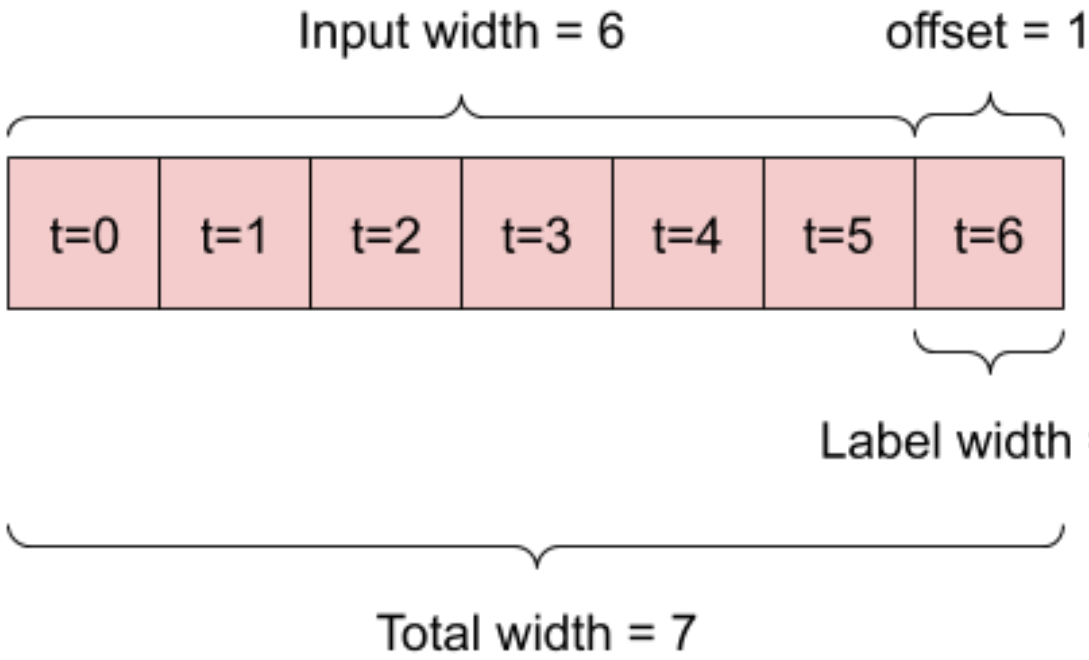
Modelling Experiments

- 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.



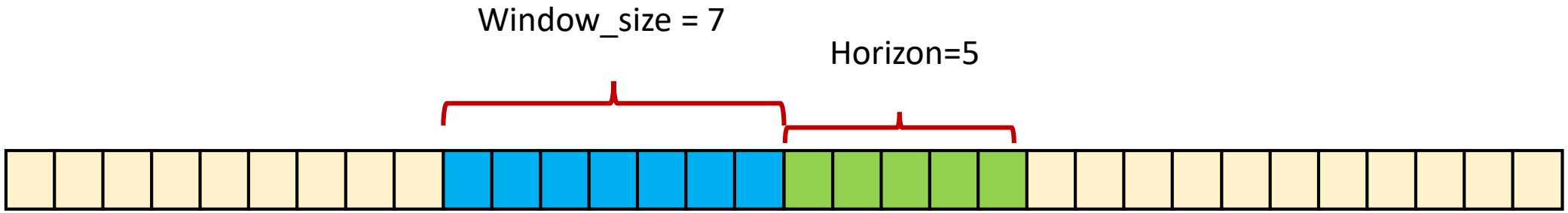
Data windowing

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

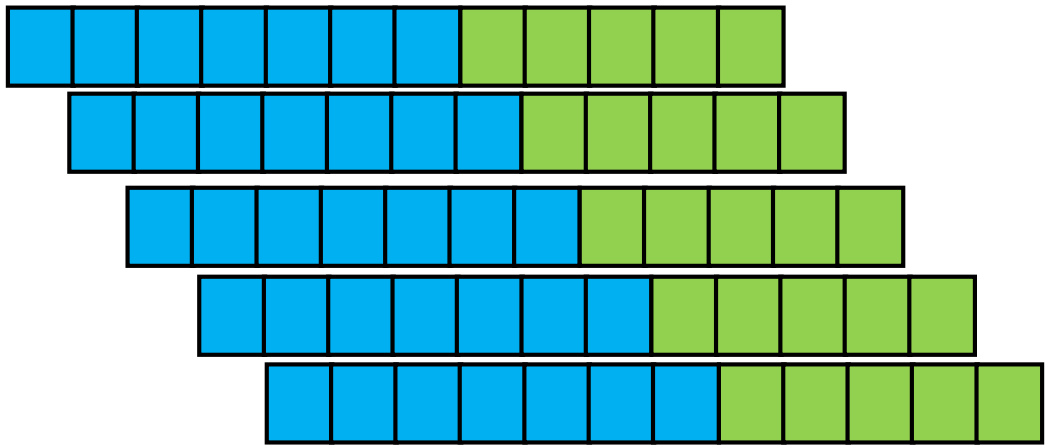


Many-to-One RNN Data Structure

Step1: set the number of window_size, horizon

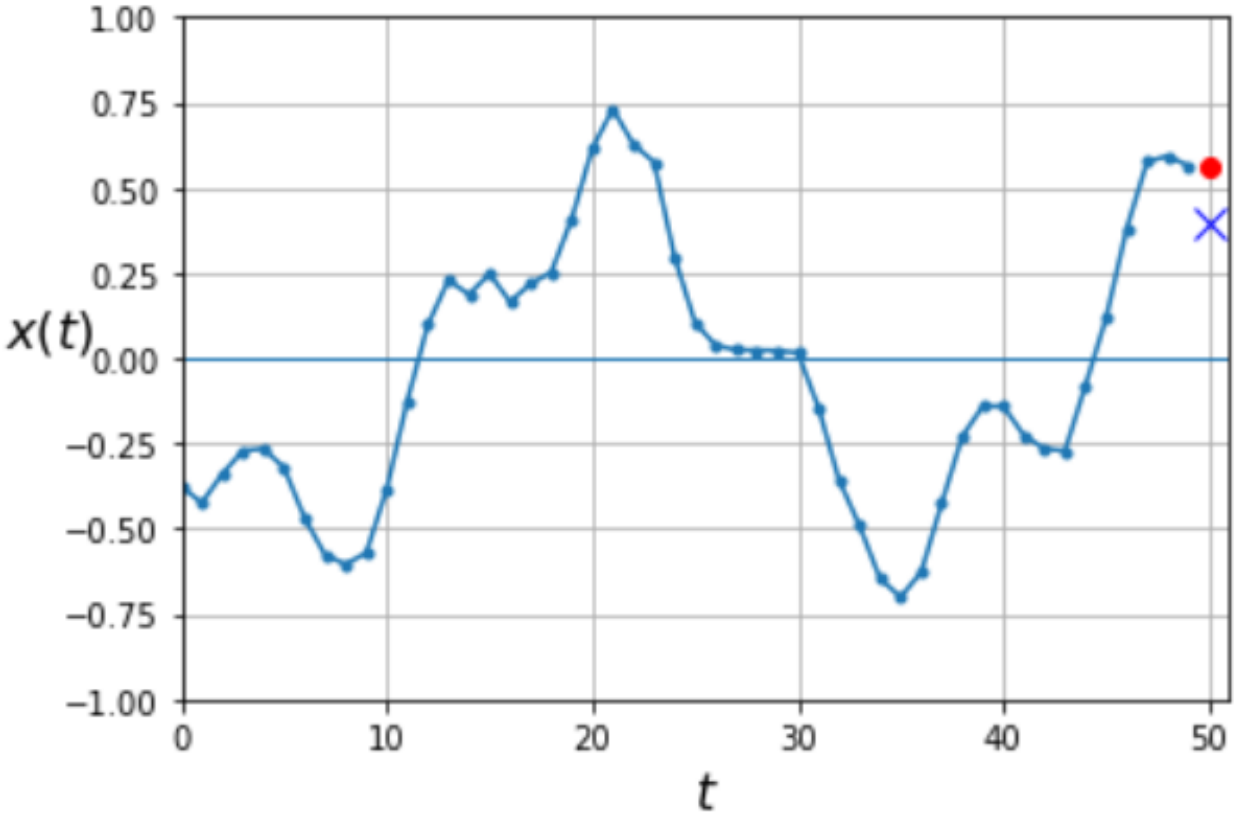


Step2: set the number of window_size, horizon



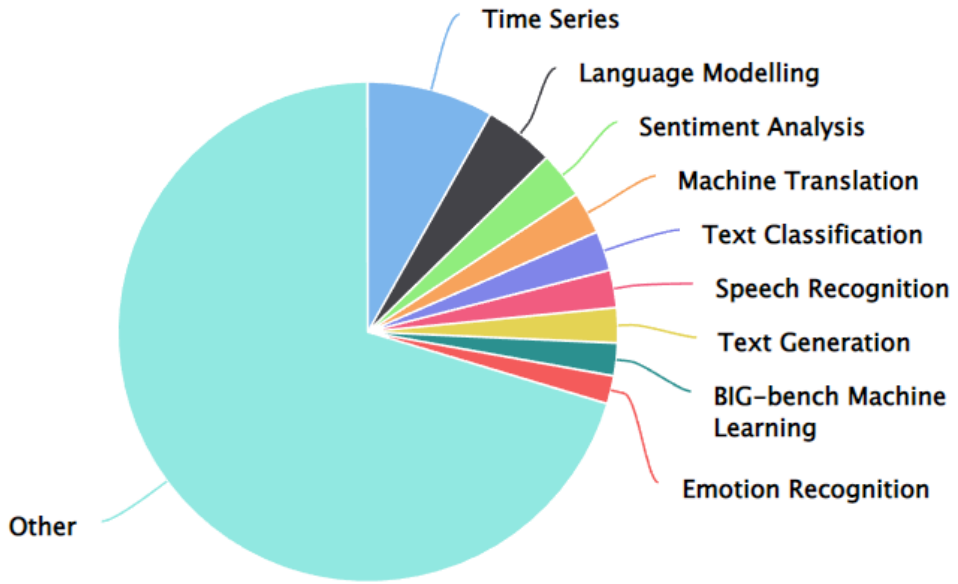
An example of sine function data

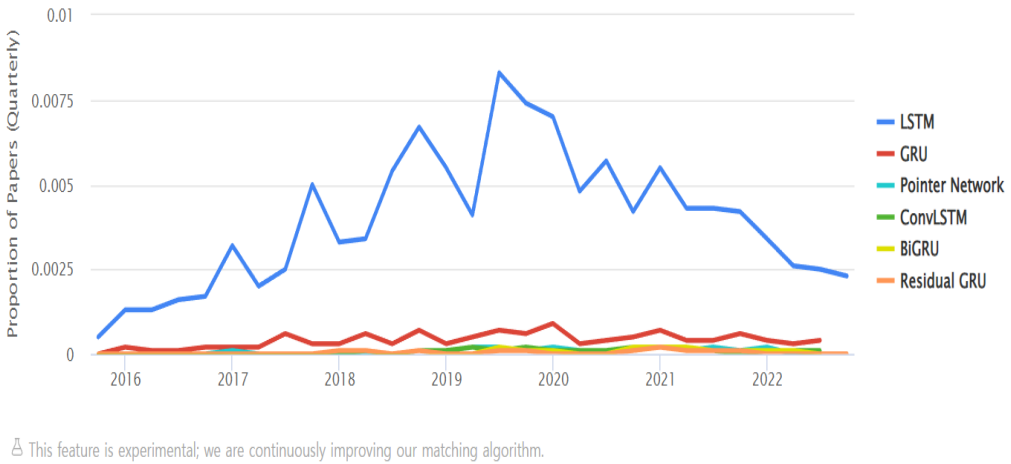
- Quizz
  - ✓window_size ?
  - ✓horizon ?



Long-Short Term Memory
(장단기 메모리)

LSTM Tasks and Usage Over Time : 2022





<source> https://paperswithcode.com/method/lstm

Overview of RNN

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

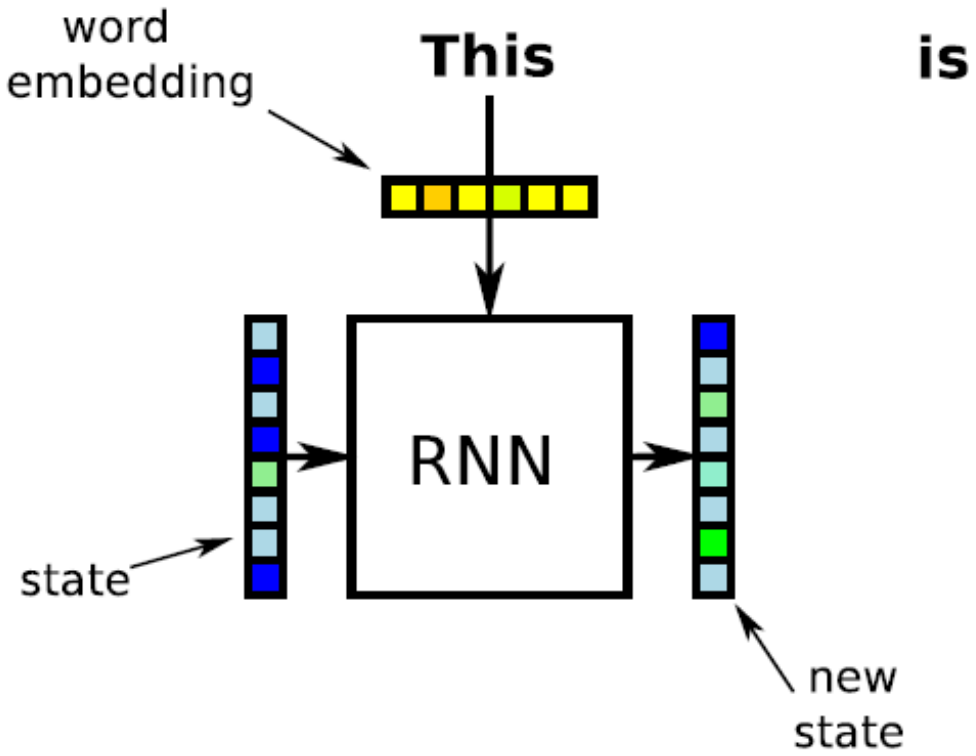


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

training RNNs: backpropagation through time

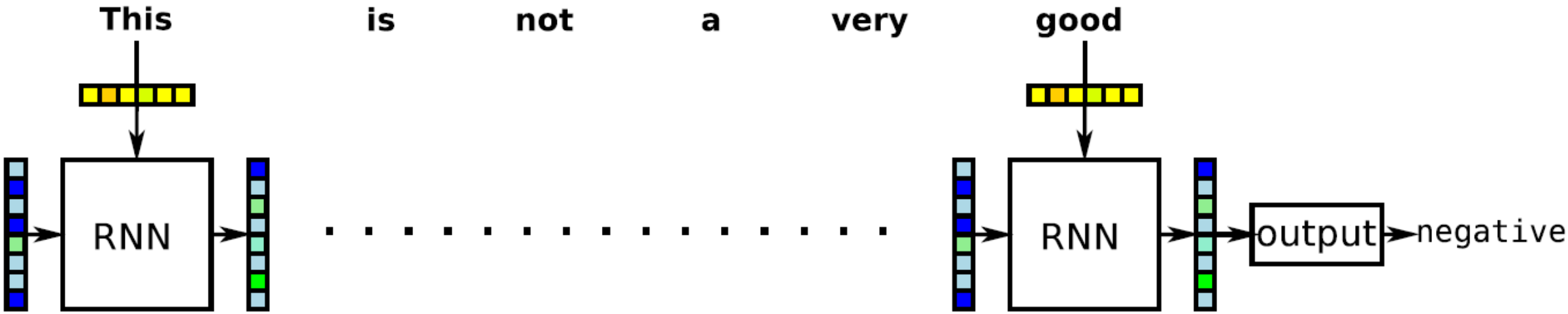


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

simple RNN implementation

- 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

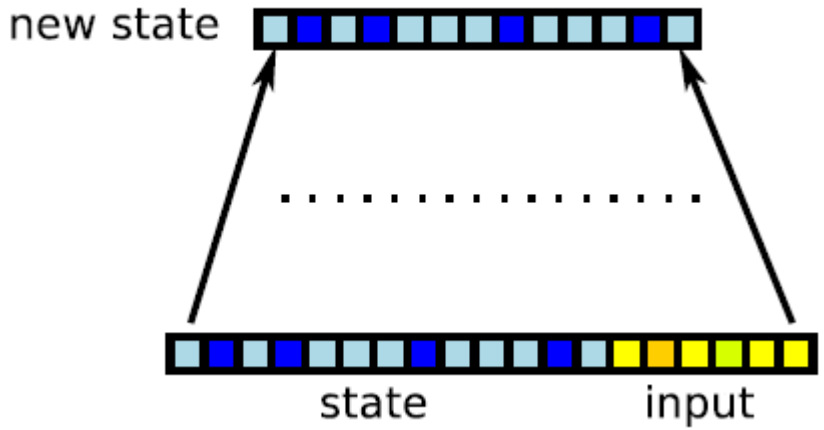


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

simple RNNs have a drawback

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

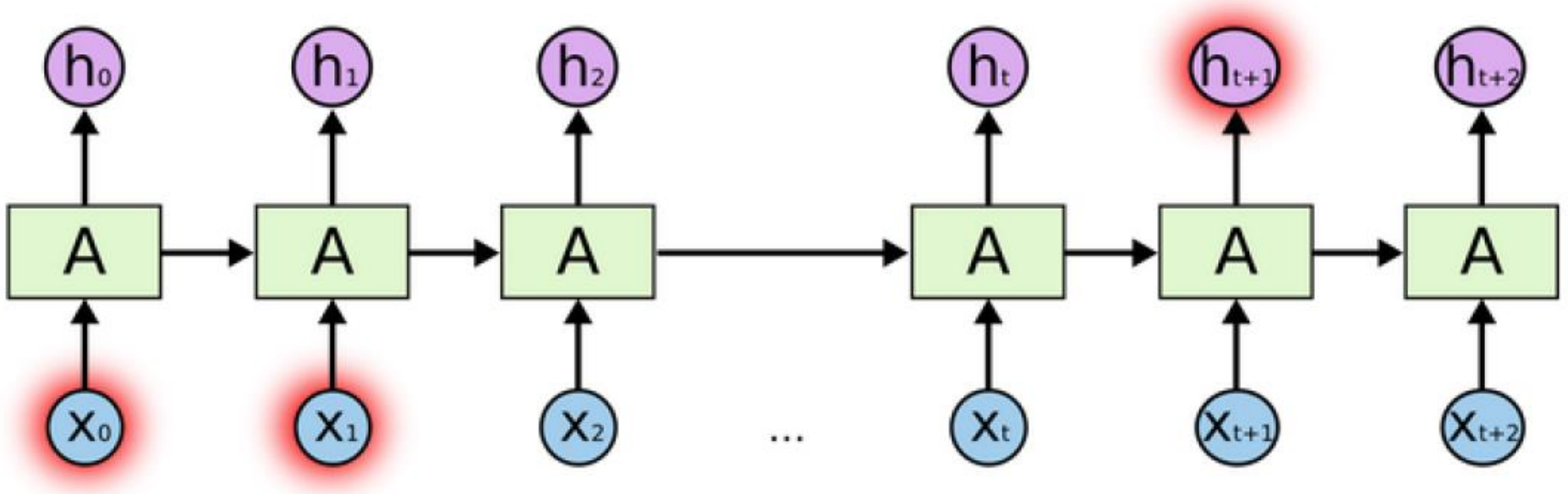


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

- 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

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

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

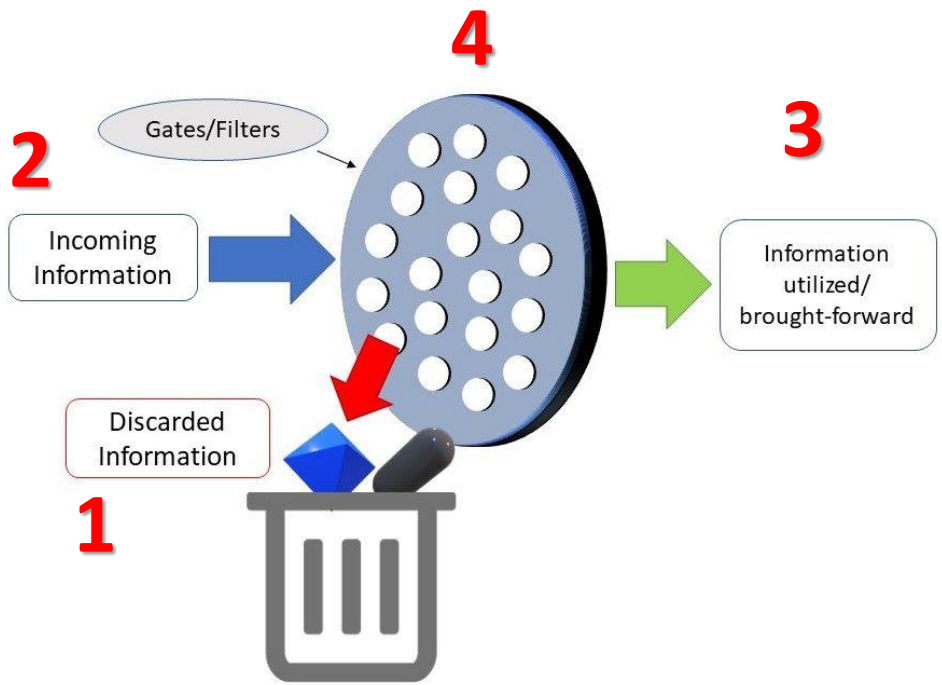
image from Goldberg's book

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

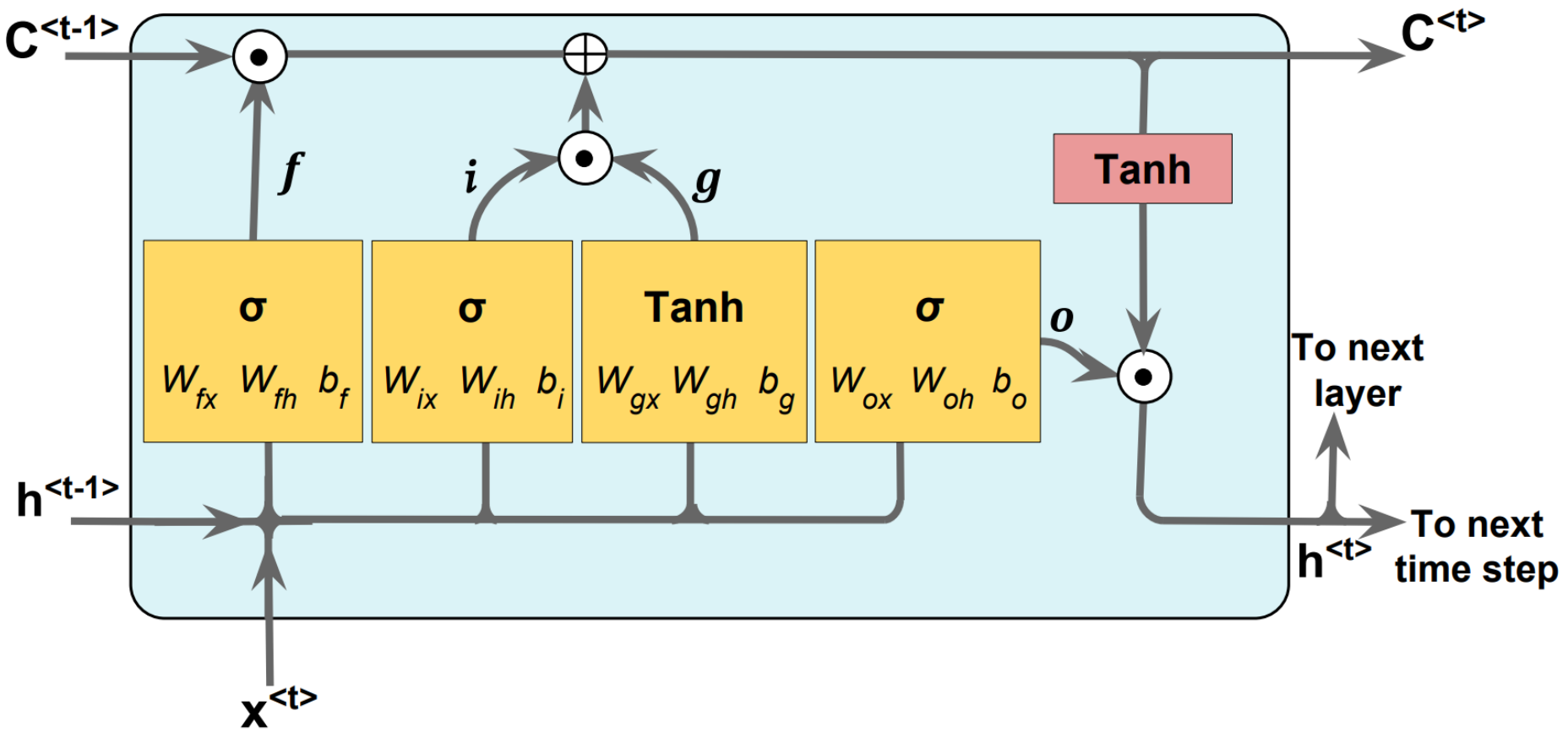
Cell states and Gates

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.



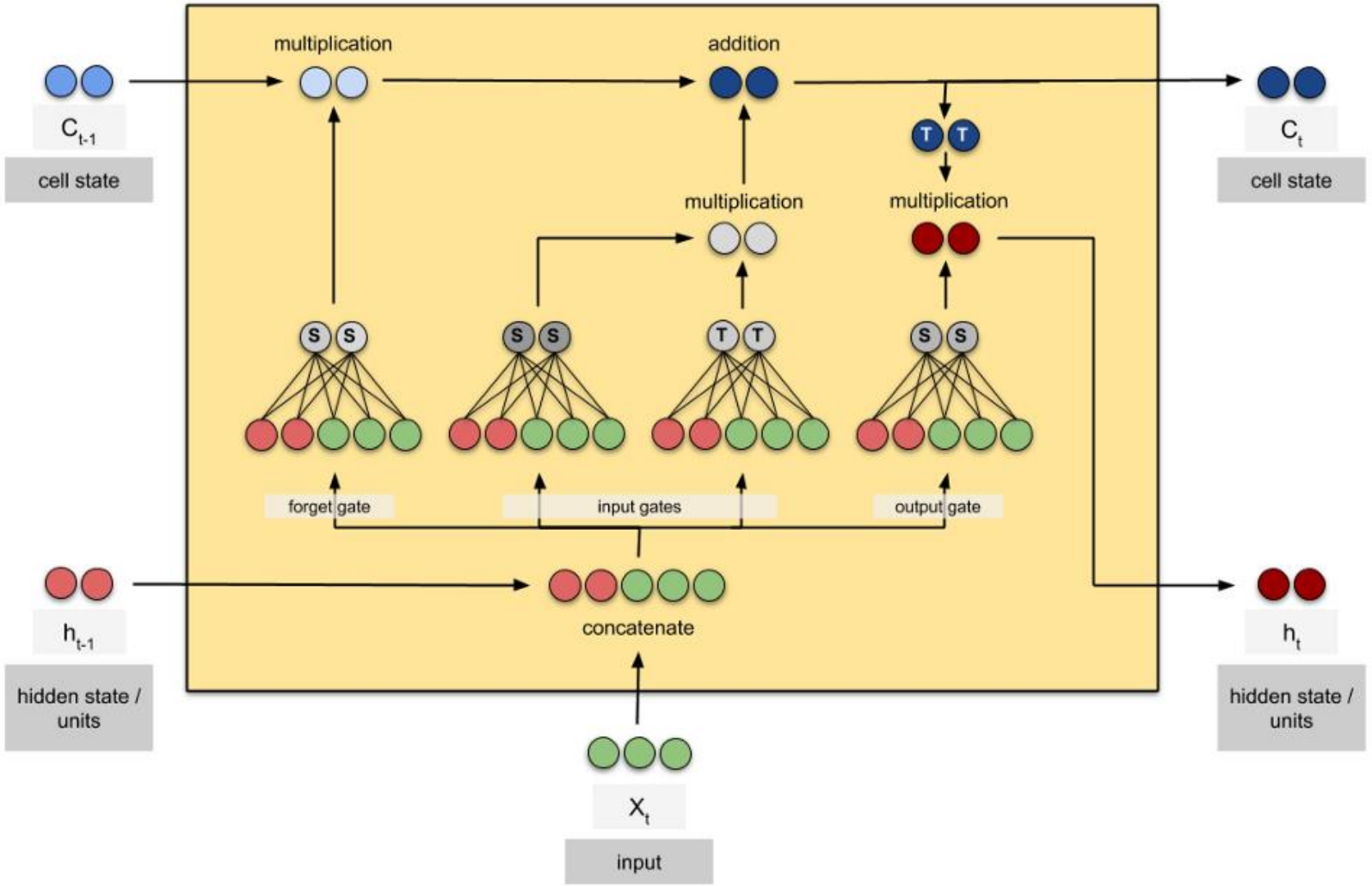
Last time: Simple RNN and LSTM



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

limitations of RNNs

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

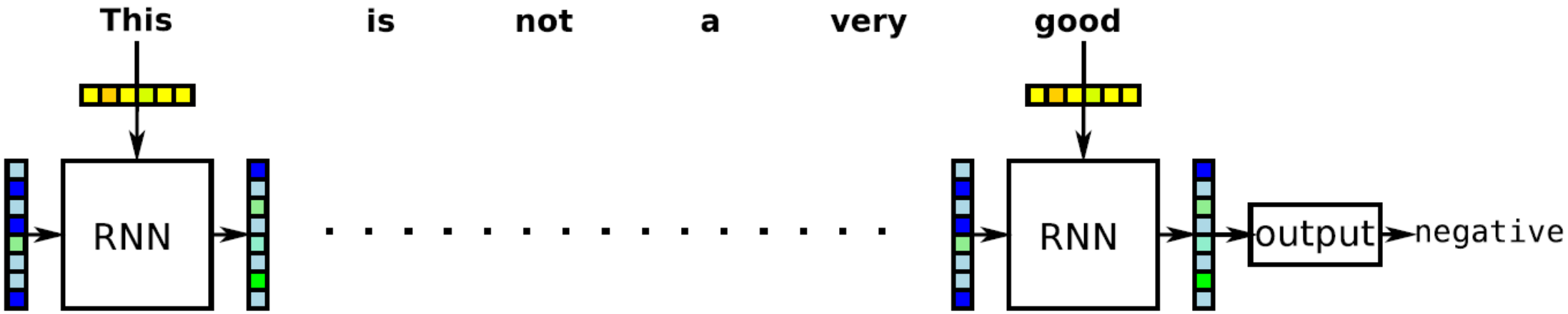
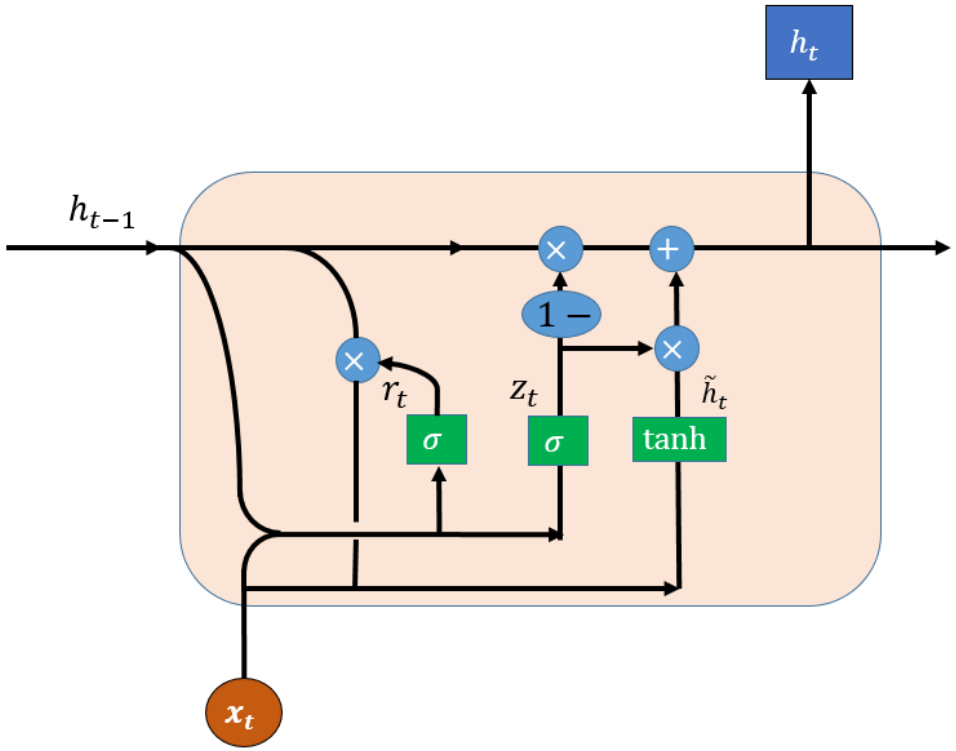


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

Gated Recurrent Units (GRU)

- 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

A Visual Guide to Recurrent Layers in Keras

source: https://amitness.com/2020/04/recurrent-layers-keras/

RNN: Single Output

❖ Let's take a simple example of encoding

For simplicity, let's assume we used some word embedding to convert each word into 2 numbers.

I am groot

Credits: Marvel Studios

Word	E1	E2
I	0.5	0.4
am	0.3	0.1
groot	0.7	0.5

We could either use one-hot encoding, pretrained word vectors, or learn word embeddings from scratch

https://amitness.com/2020/04/recurrent-layers-keras/

SimpleRNN (1)

❖ SimpleRNN with a Dense layer

✓ to build an architecture for something like sentiment analysis or text classification.

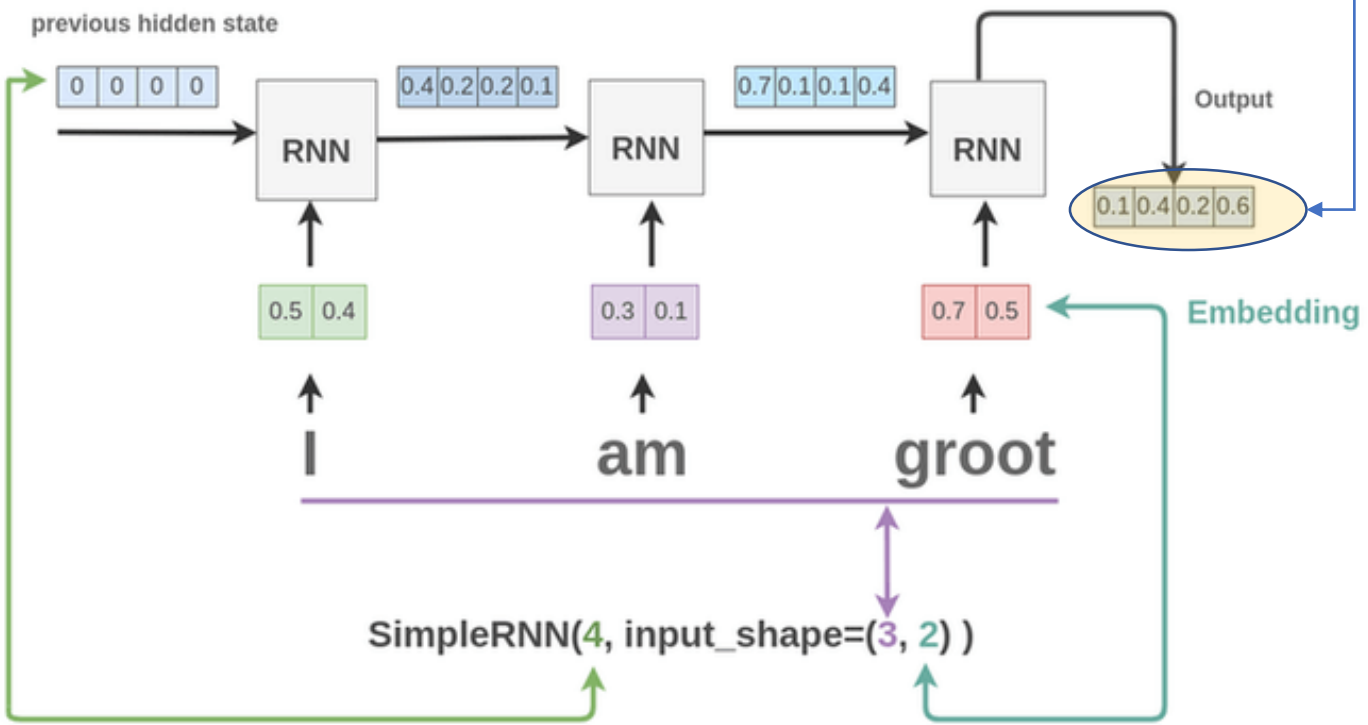
```
import tensorflow as tf
from tensorflow.keras.layers import SimpleRNN   #Dense, LSTM
# from tensorflow.keras.models import Sequential

x = tf.random.normal((1, 3, 2))

layer = SimpleRNN(4, input_shape=(3, 2))
output = layer(x)

print(output.shape)
print(x)
```

```
(1, 4)
tf.Tensor(
[[[ 0.6887584   1.3883604 ]
  [ 0.01564607 -1.4314882 ]
  [-0.05214449 -0.65099174]]], shape=(1, 3, 2), dtype=float32)
```

SimpleRNN : Many-to-One

- we treat each word as a time-step and the embedding as features.

```
model.add(layers.SimpleRNN(4,input_shape=(3,2)))
```



SimpleRNN (3) : return_sequences=True

```
# multiple output
layer = SimpleRNN(4, input_shape=(3, 2), return_sequences=True )
output = layer(x)
print(output.shape)
print(output)
```

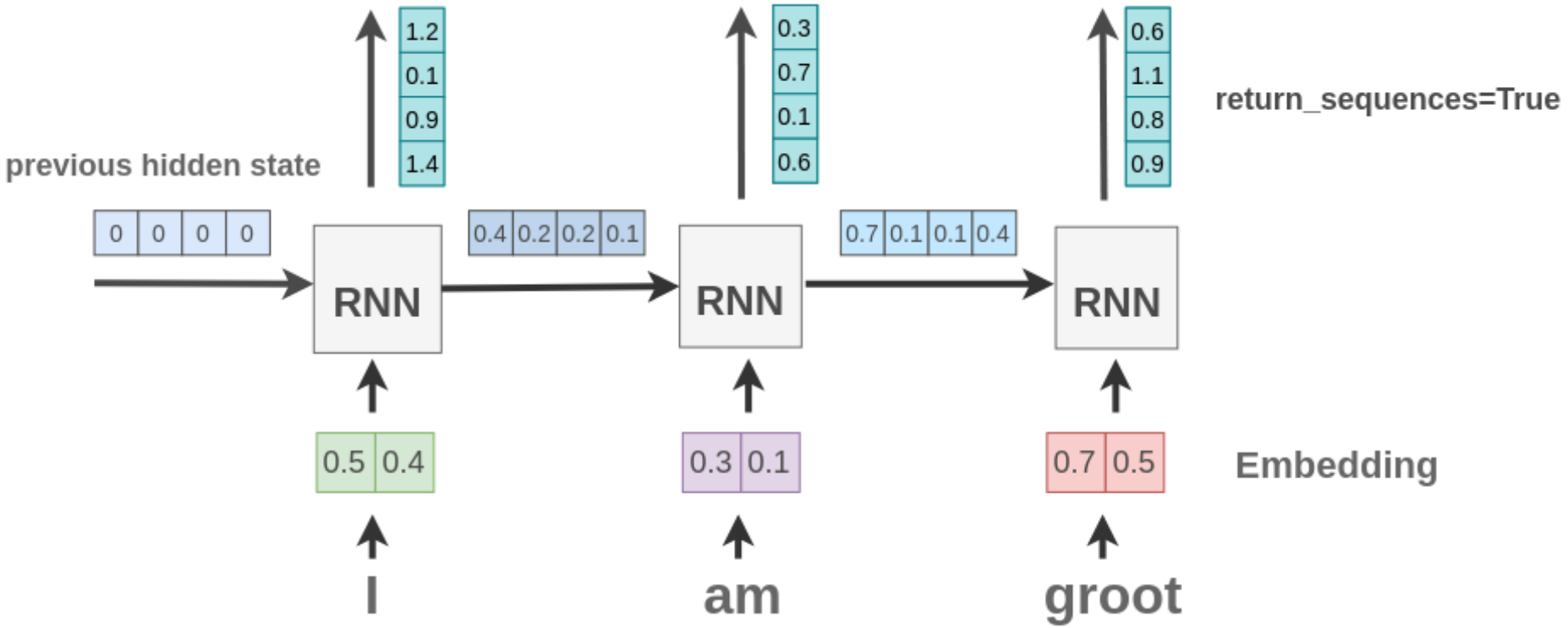
```
(1, 3, 4)
tf.Tensor(
[[[-0.6854385   0.08265962  0.30888444 -0.30752325]
  [ 0.4584542  -0.1935767  -0.91095936 -0.2416075 ]
  [ 0.7241105  -0.49960855 -0.5059616   0.7261468 ]]], shape=(1, 3, 4), dtype=float32)
```

RNN with return_sequences : Many-to-Many

❖ return_sequences = True

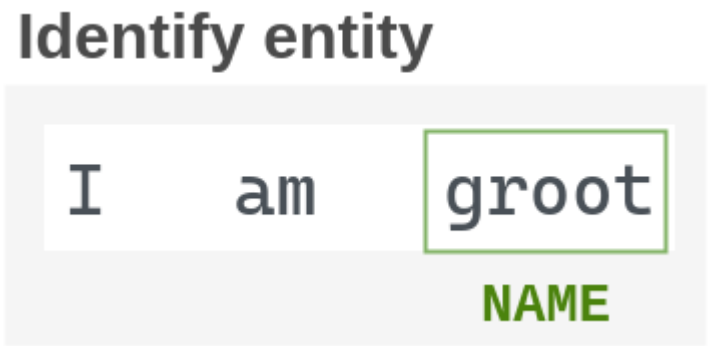
✓ True : the output from each unfolded RNN cell is returned instead of only the last cell.

```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
```



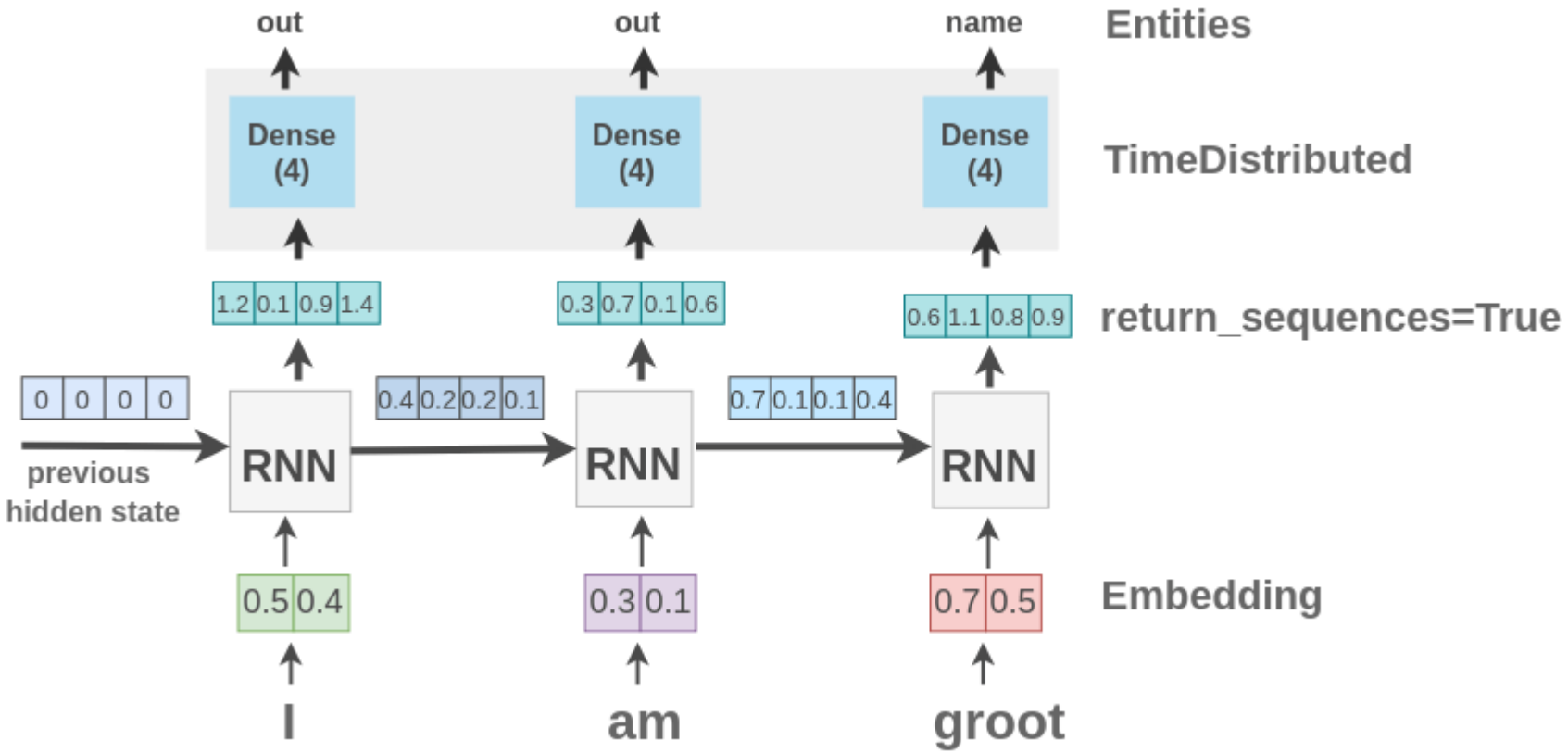
RNN with TimeDistributed : Many-to-Many

- Suppose we want to recognize entities in a text.
  - ✓For example, in our text “I am Groot”, we want to identify “Groot” as a name.



(3) RNN: TimeDistributed Layer

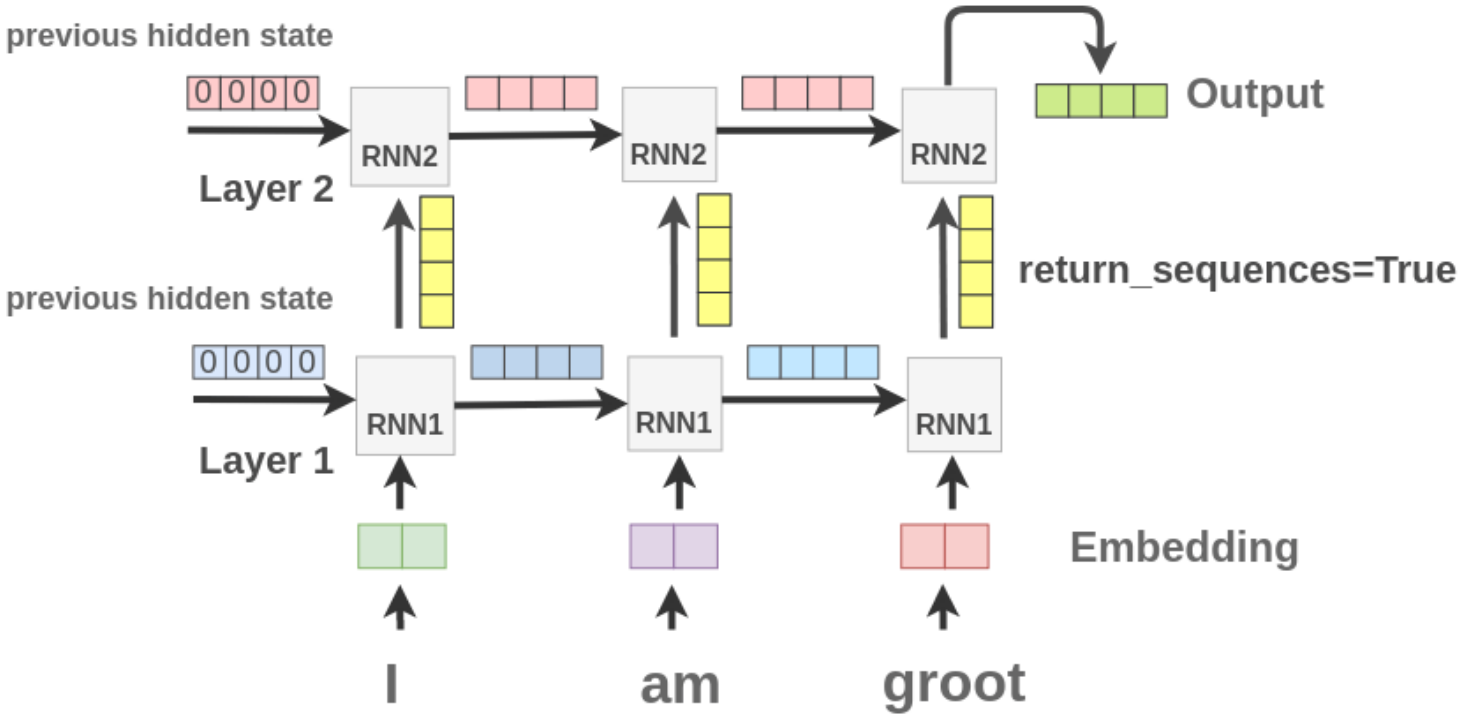
```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
model.add(TimeDistributed(Dense(4, activation='softmax')))
```



RNN Stacking Layer : Deep but Many-to-One

- We can also stack multiple recurrent layers one after another in Keras

```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
model.add(SimpleRNN(4))
```



[HW] Let's Code!

A Simple LSTM layer

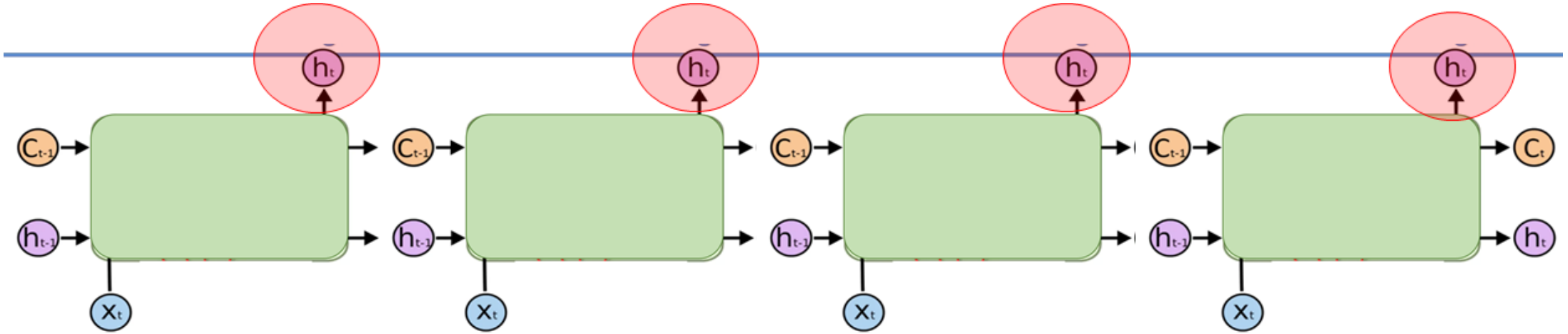
LSTM Parameters

❖ 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

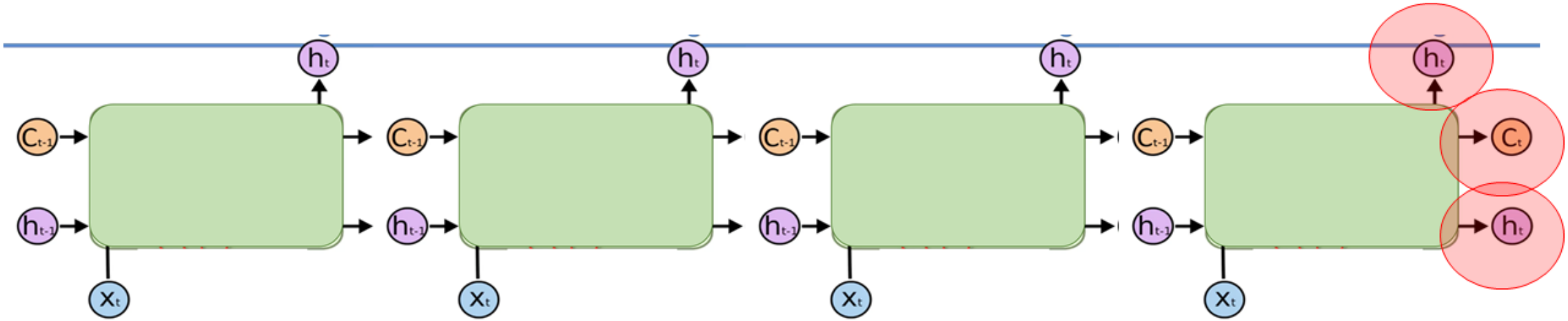
- return_sequences = True
  - ✓all hidden states



LSTM : return_state=True

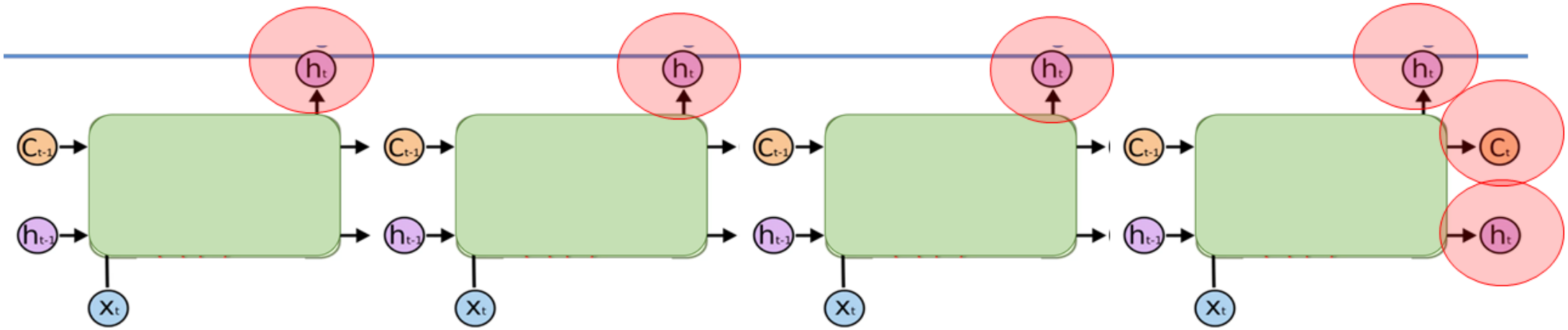
❖ return_state=True

✓last hidden state + last hidden state (again!) + last cell state

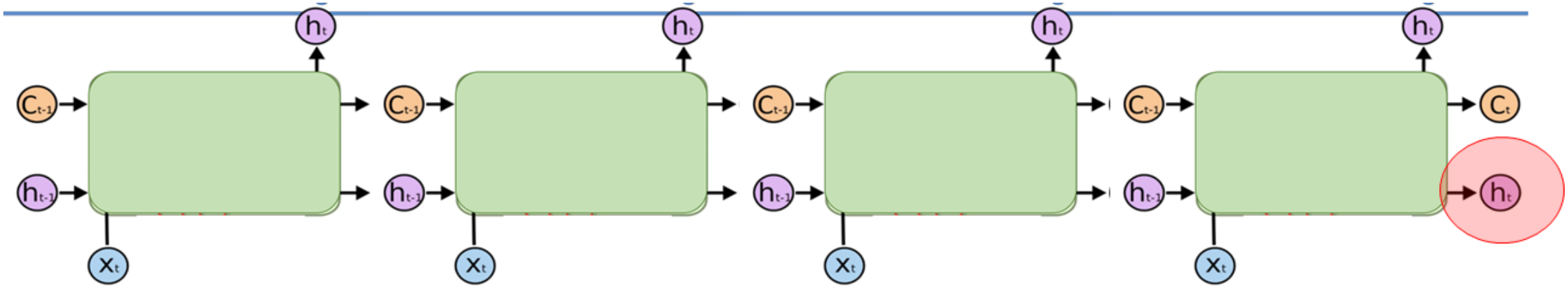


LSTM: return_sequences=True and return_state=True

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

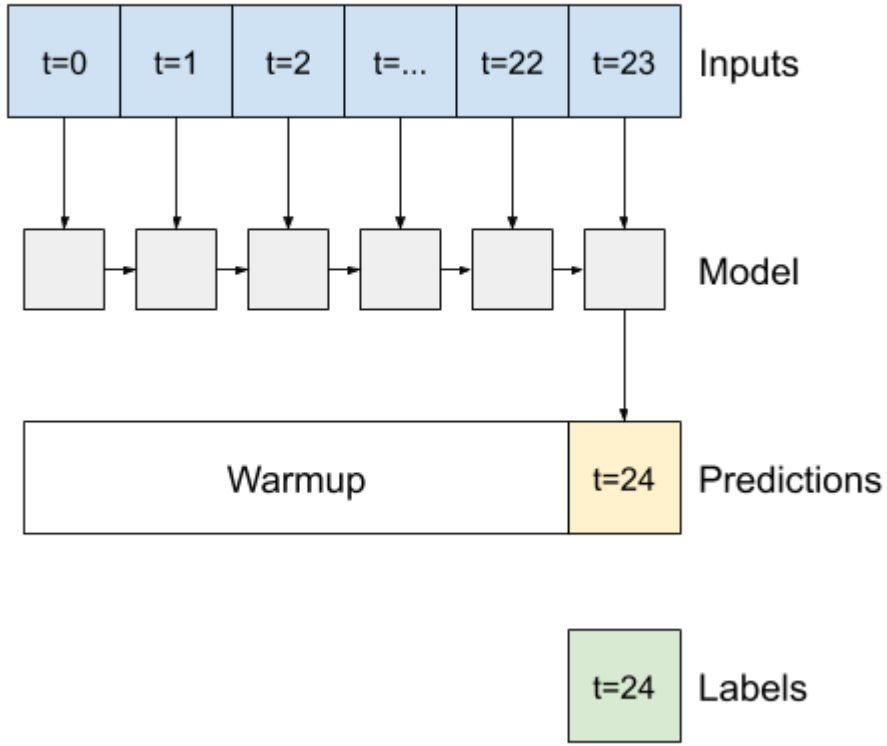


① Default: Last Hidden State (Hidden State of the last time step)



(Examples) return_sequences=False (the default)

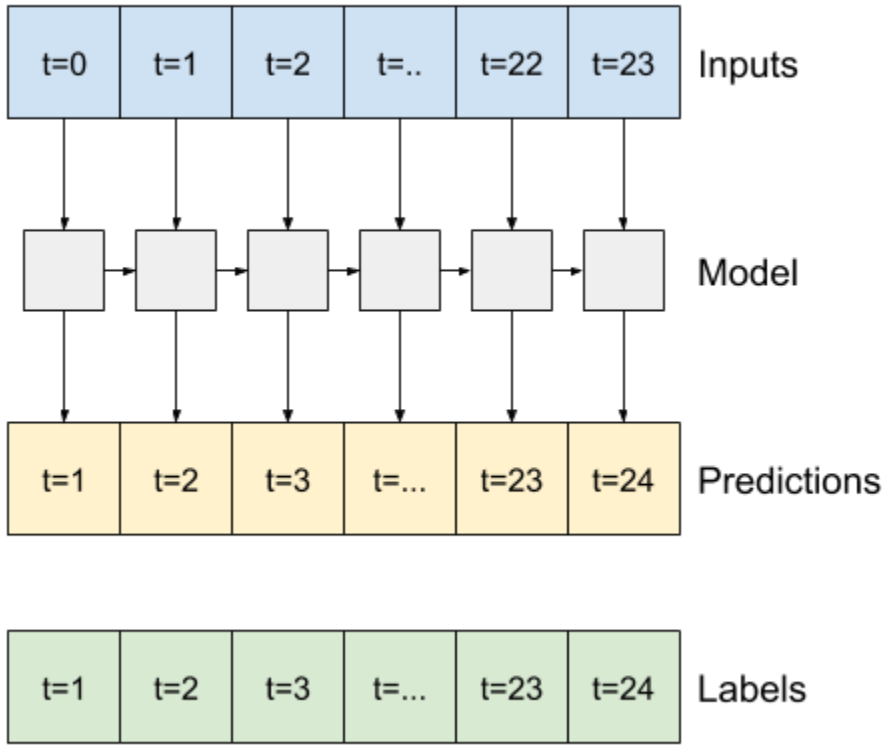
- the layer only returns the output of the final time step, giving the model time to warm up its internal state before making a single prediction:



(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



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)
```

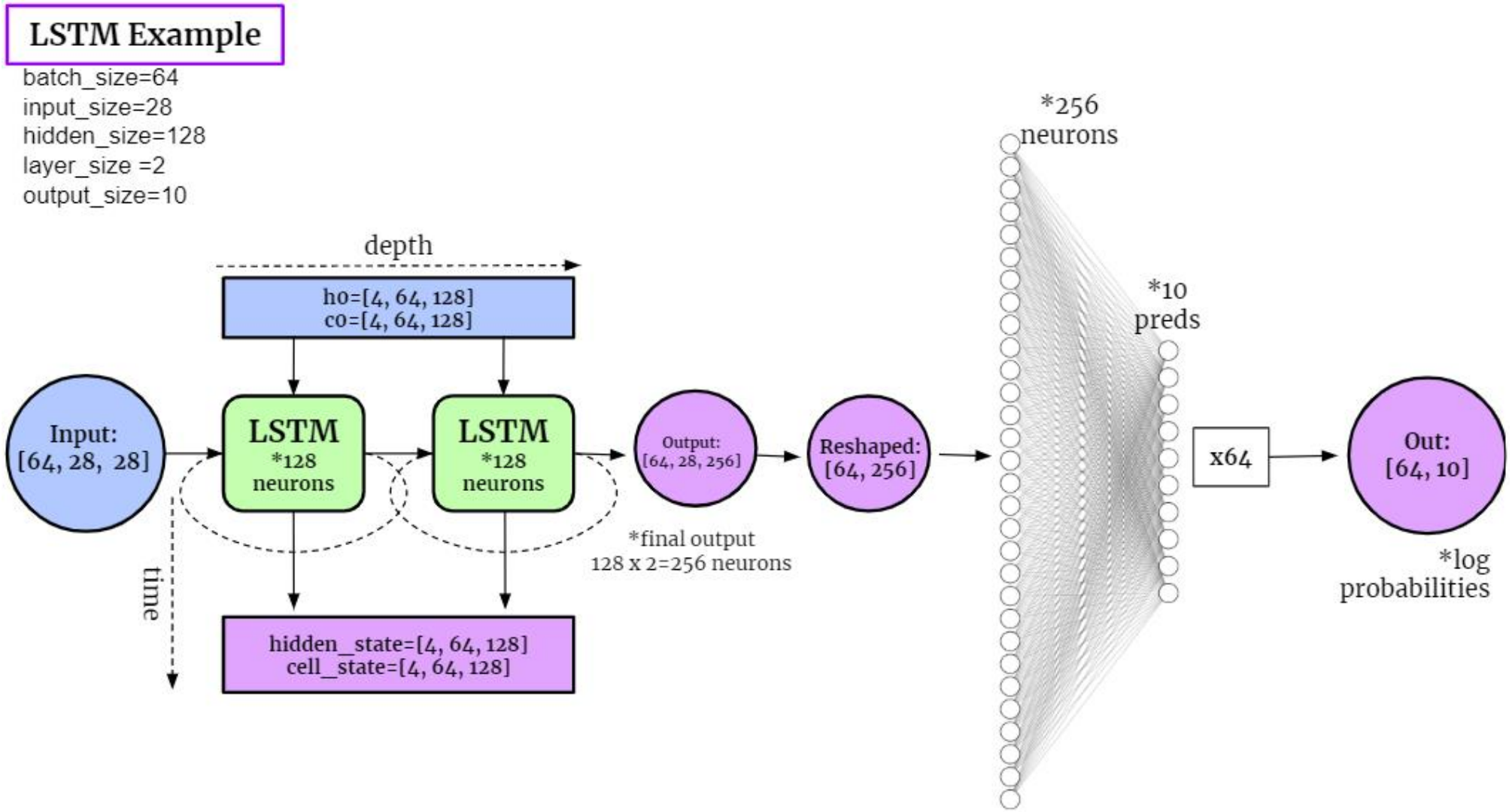
GRU 이해하기

```
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)
```

Let's Code 2: LSTM

LSTM Example : Parameters



LSTM Example : many-to-one

```
from keras.models import Model
from keras.layers import Input, Dense, LSTM
import numpy as np

x = np.array([[[1.], [2.], [3.], [4.], [5.]]])
y = np.array([[6.]])

xInput = Input(batch_shape=(None, 5, 1))
xLstm = LSTM(3)(xInput)
xOutput = Dense(1)(xLstm)

model = Model(xInput, xOutput)
model.compile(loss='mean_squared_error', optimizer='adam')
print(model.summary())

model.fit(x, y, epochs=50, batch_size=1, verbose=0)
model.predict(x, batch_size=1)
```

time series input

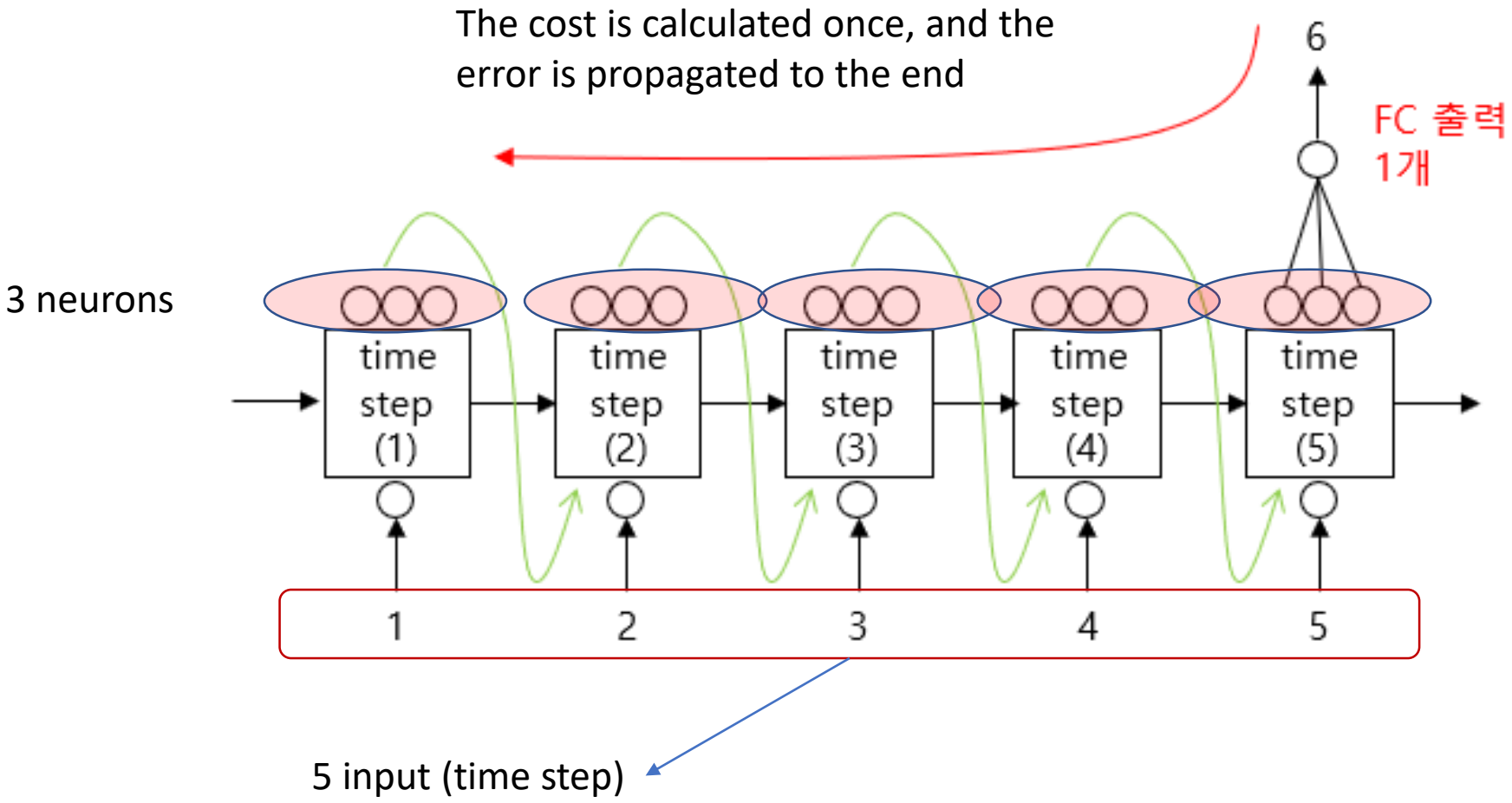
Inputs =[batch, time_step, features]

neurons in LSTM layers

```
Model: "model_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 5, 1)]            0
_________________________________________________________________
lstm_6 (LSTM)                (None, 3)                 60
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 4
=================================================================
Total params: 64
Trainable params: 64
Non-trainable params: 0
```

Unfolded LSTM : Many to One

return_sequences=False



LSTM many-to-many with TimeDistributed Layer

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
x = np.array([[[1.], [2.], [3.], [4.], [5.]]])
y = np.array([[[2.], [3.], [4.], [5.], [6.]]])
```

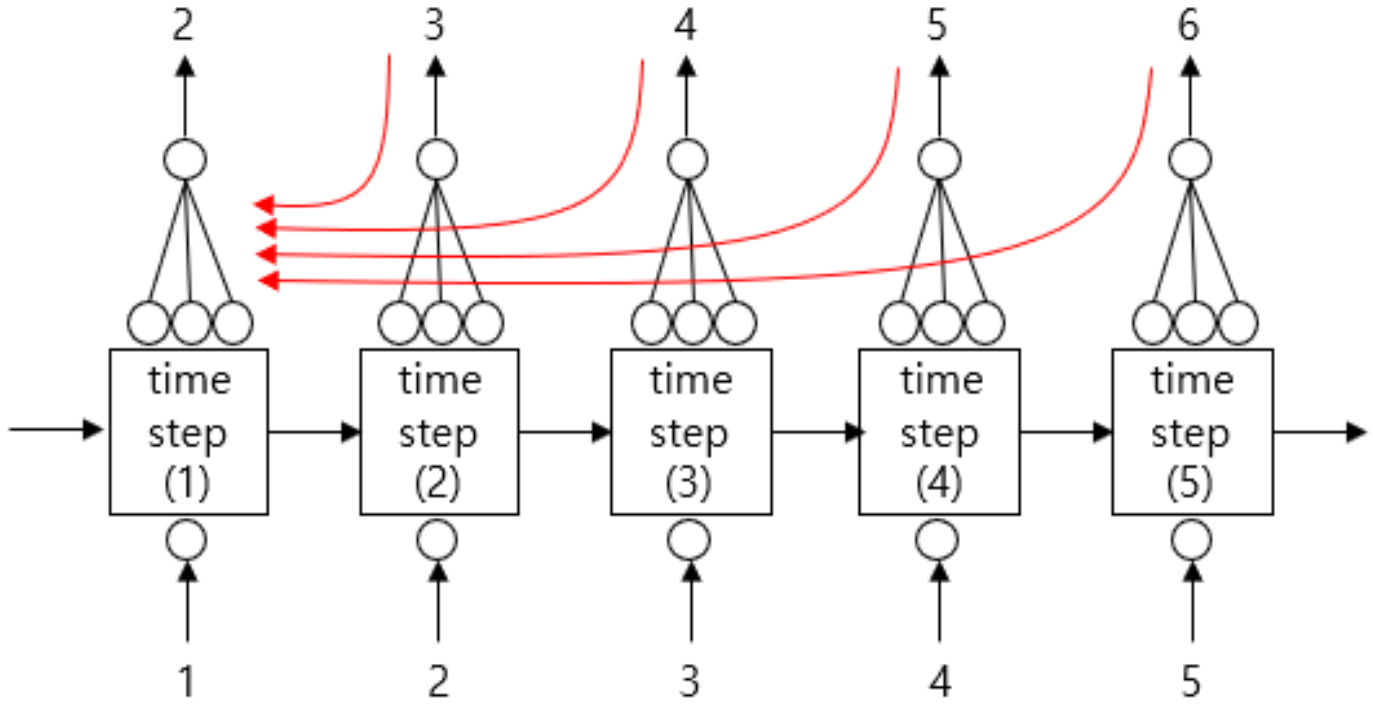
```
model2 = keras.models.Sequential([
    keras.layers.LSTM(3, return_sequences=True, input_shape=[5, 1]),
    keras.layers.TimeDistributed(keras.layers.Dense(1) )                ])
model2.compile(loss='mean_squared_error', optimizer='adam')
model2.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_18 (LSTM)               (None, 5, 3)              60

time_distributed_8 (TimeDis  (None, 5, 1)              4
tributed)

=================================================================
Total params: 64
Trainable params: 64
Non-trainable params: 0
_________________________________________________________________
```

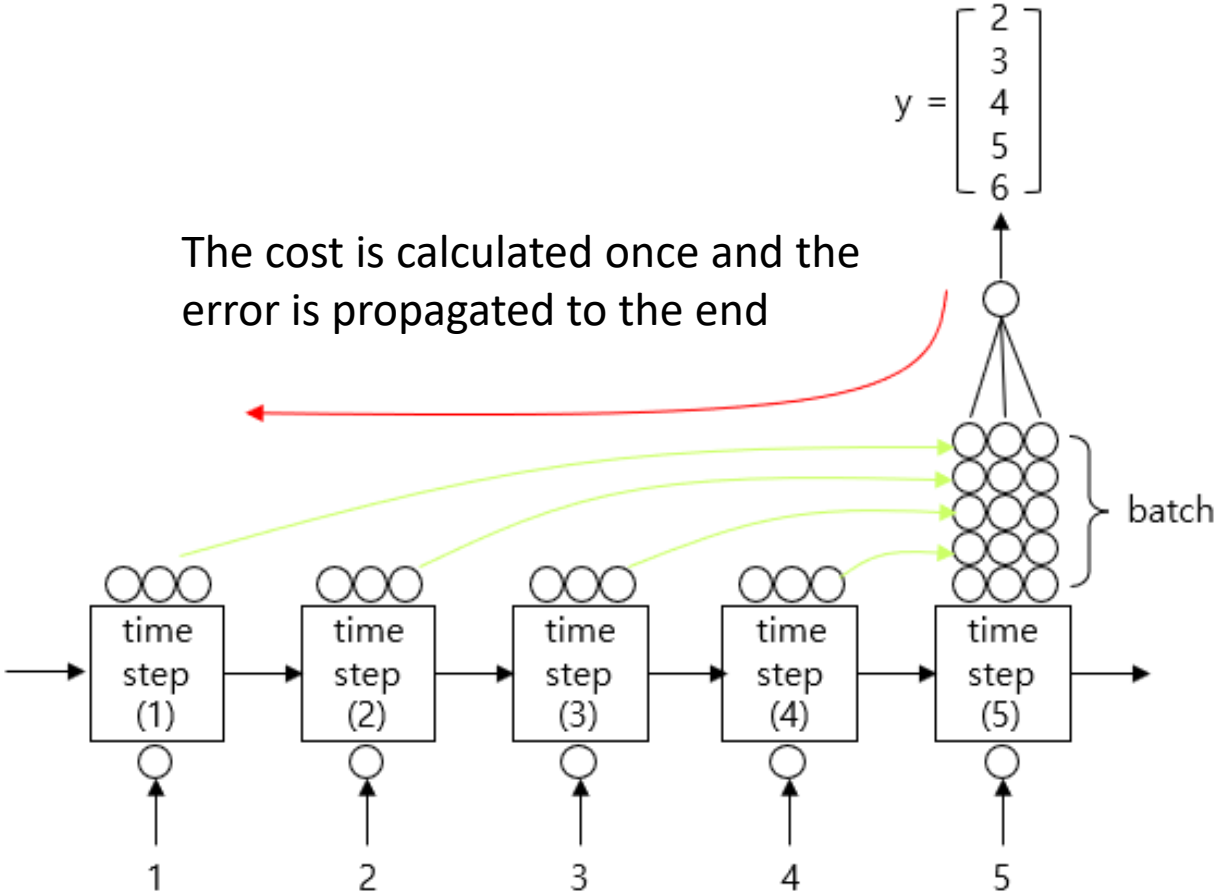
LSTM Output is 5x3(neurons)

Final Output is 5x1

Backpropagation Through Time and Loss



Without keras.layers.TimeDistributed()



KISTI SINCE 1962

2022 TRUST KISTI

Korea Institute of Science and Technology Information