KISTI 한국과학기술정보연구원
2022

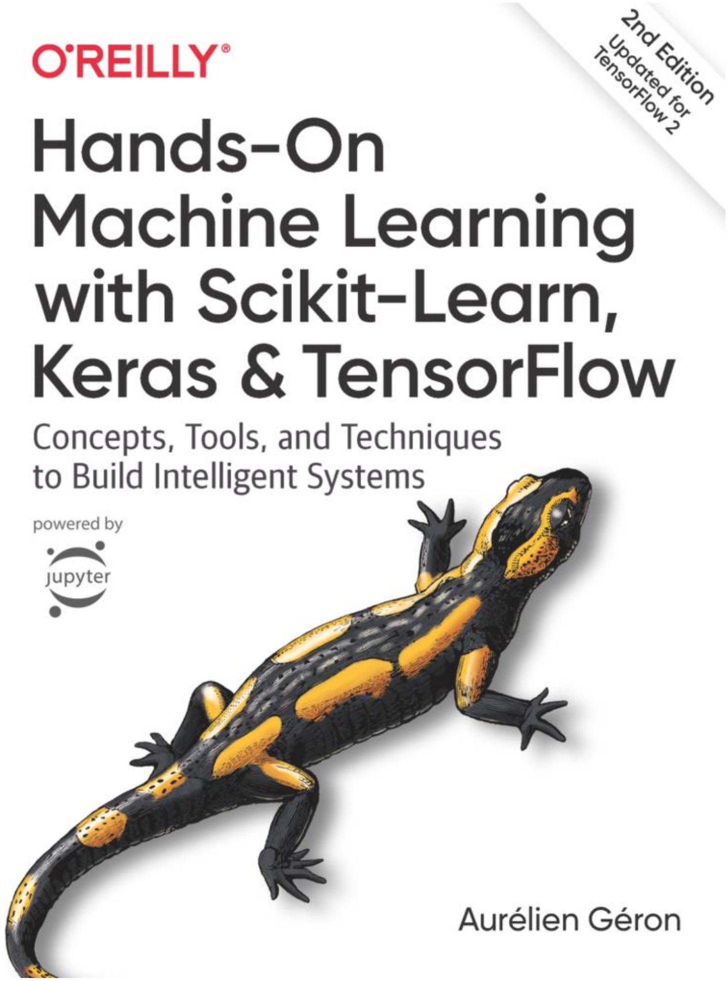
Deep Learning based Text Processing

Lec 10: Introduction to Long-Short Term Memory

hsyi@kisti.re.kr
Hongsuk Yi (이홍석)

Overview of Course

- 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



딥 러닝을 이용한 자연어 처리 입문

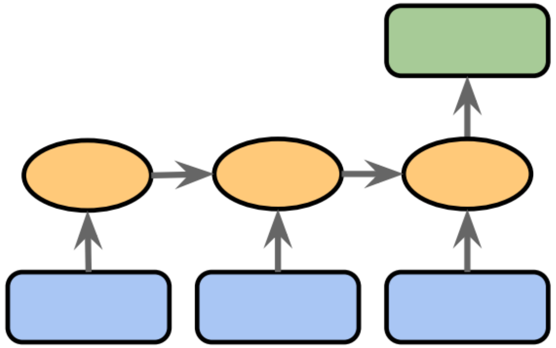
https://wikidocs.net/book/2155

Reviewing the last class: RNN

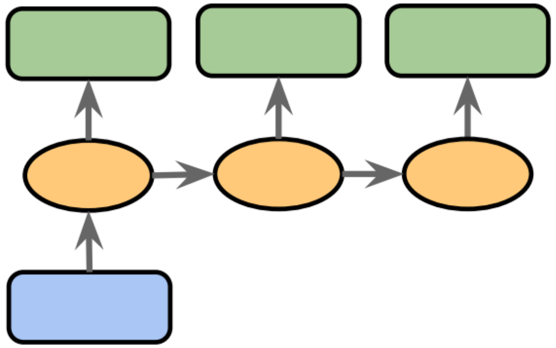
Recurrent NN for processing sequences

- 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

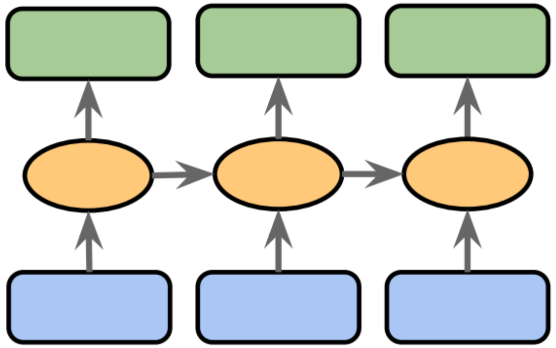
Different Types of Sequence Modeling Tasks



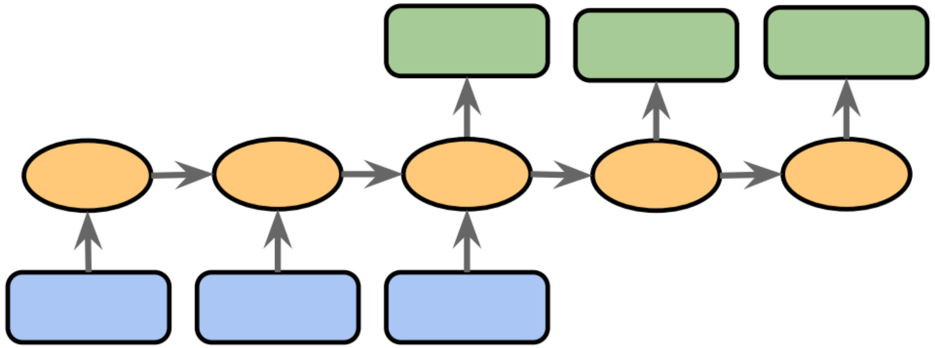
many-to-one



one-to-many



many-to-many



many-to-many

RNN example

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

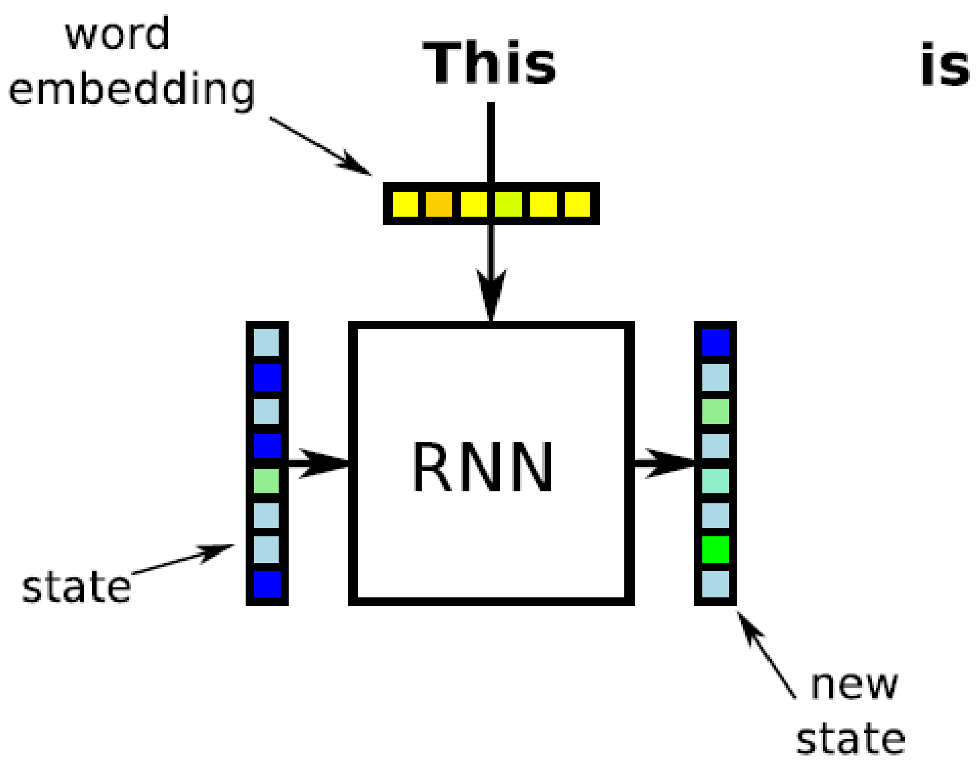


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

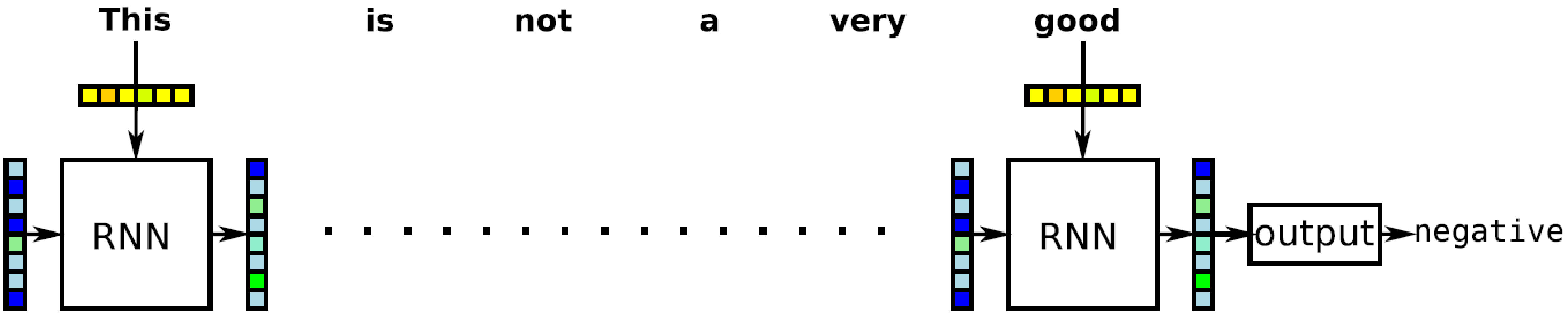


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

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:

$$y_t = \boxed{s_t}$$

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

actication is typically tanh

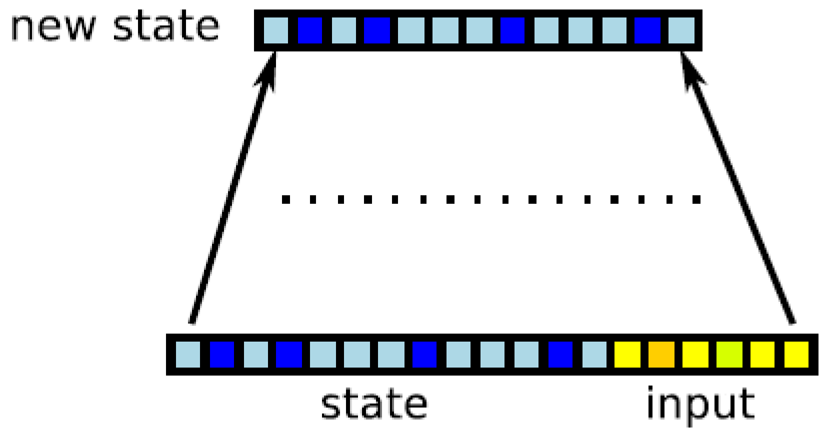
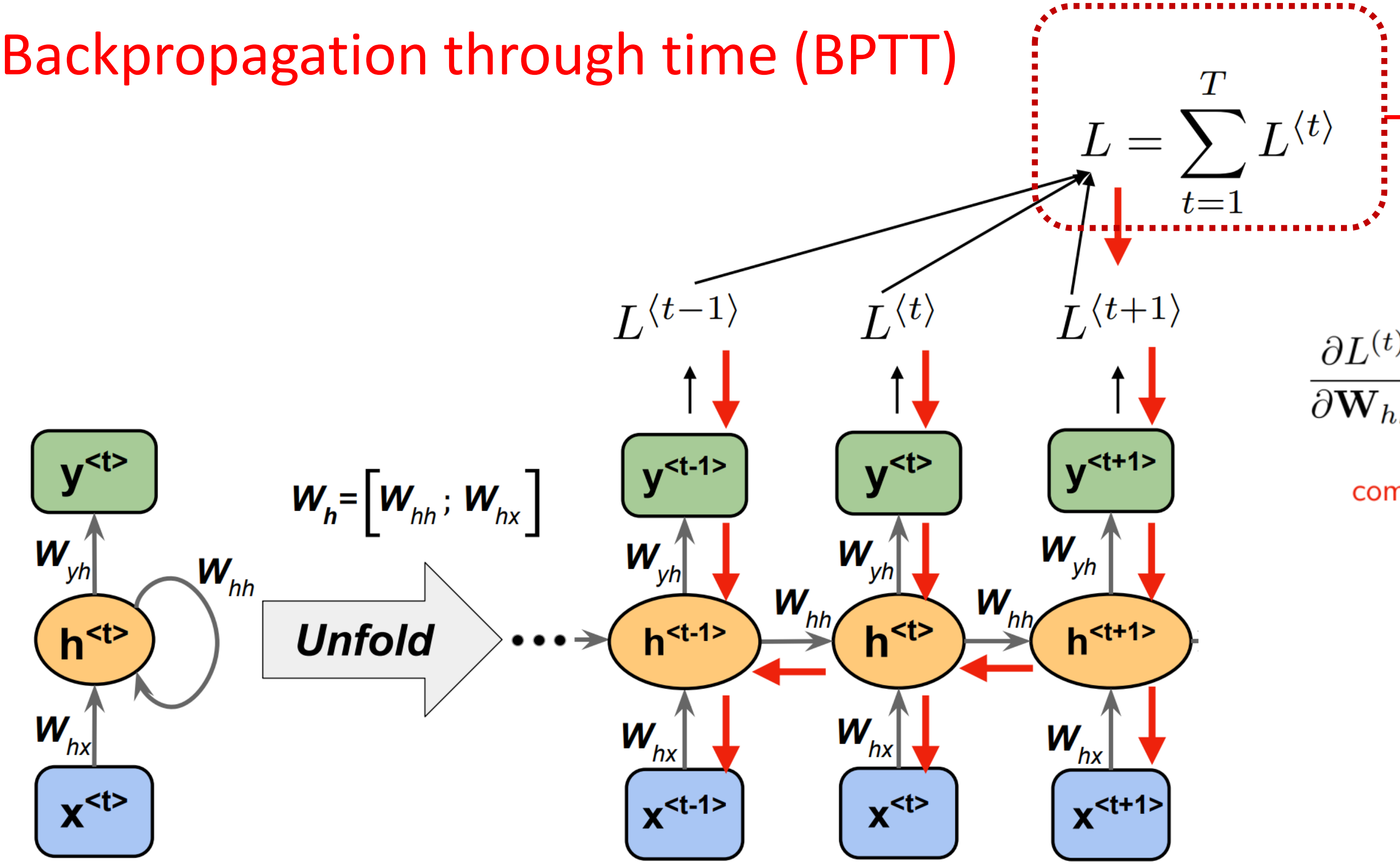


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

backpropagation through time

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

Backpropagation through time (BPTT)



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

computed as a multiplication of adjacent time steps:

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

simple RNNs have a drawback

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

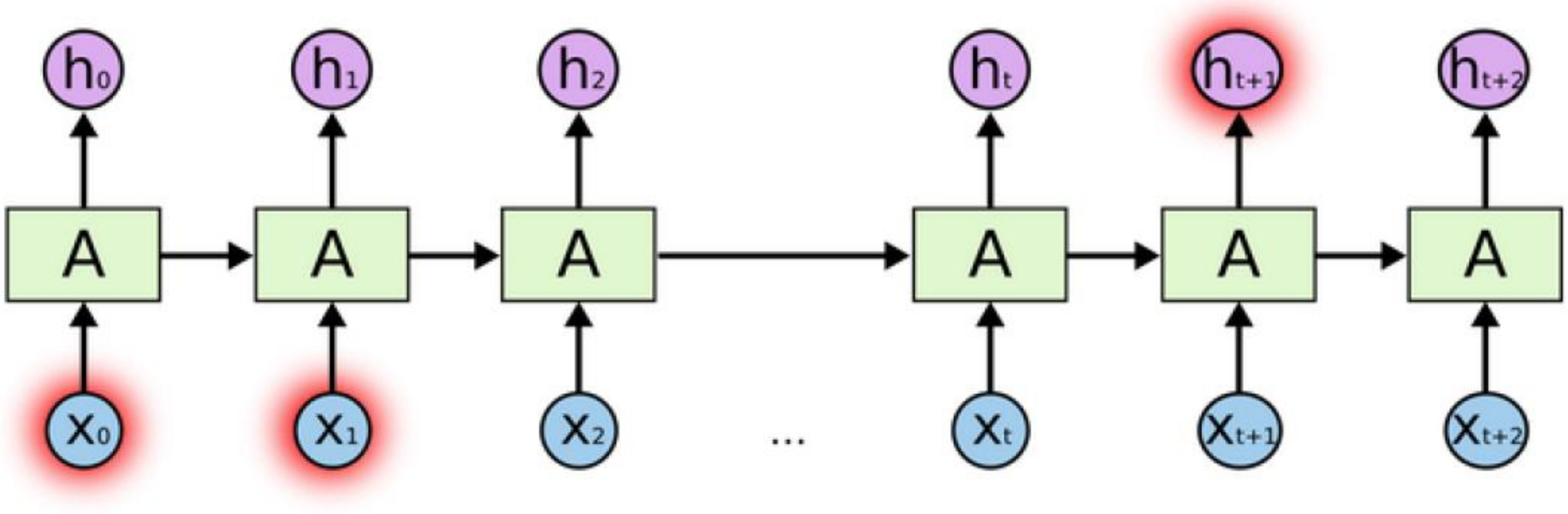


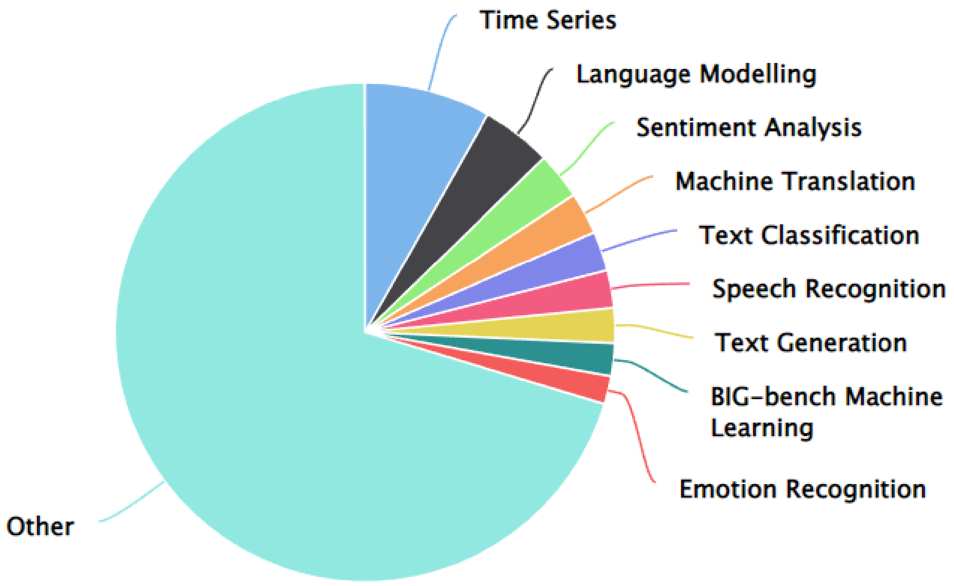
image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

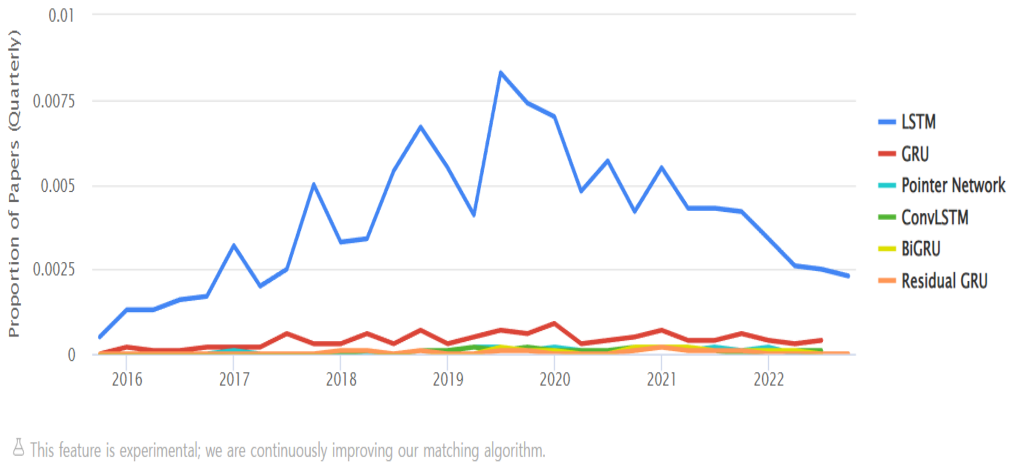
Solutions to the vanishing/exploding gradient problems

- **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)

Long-Short Term Memory
(장단기 메모리)

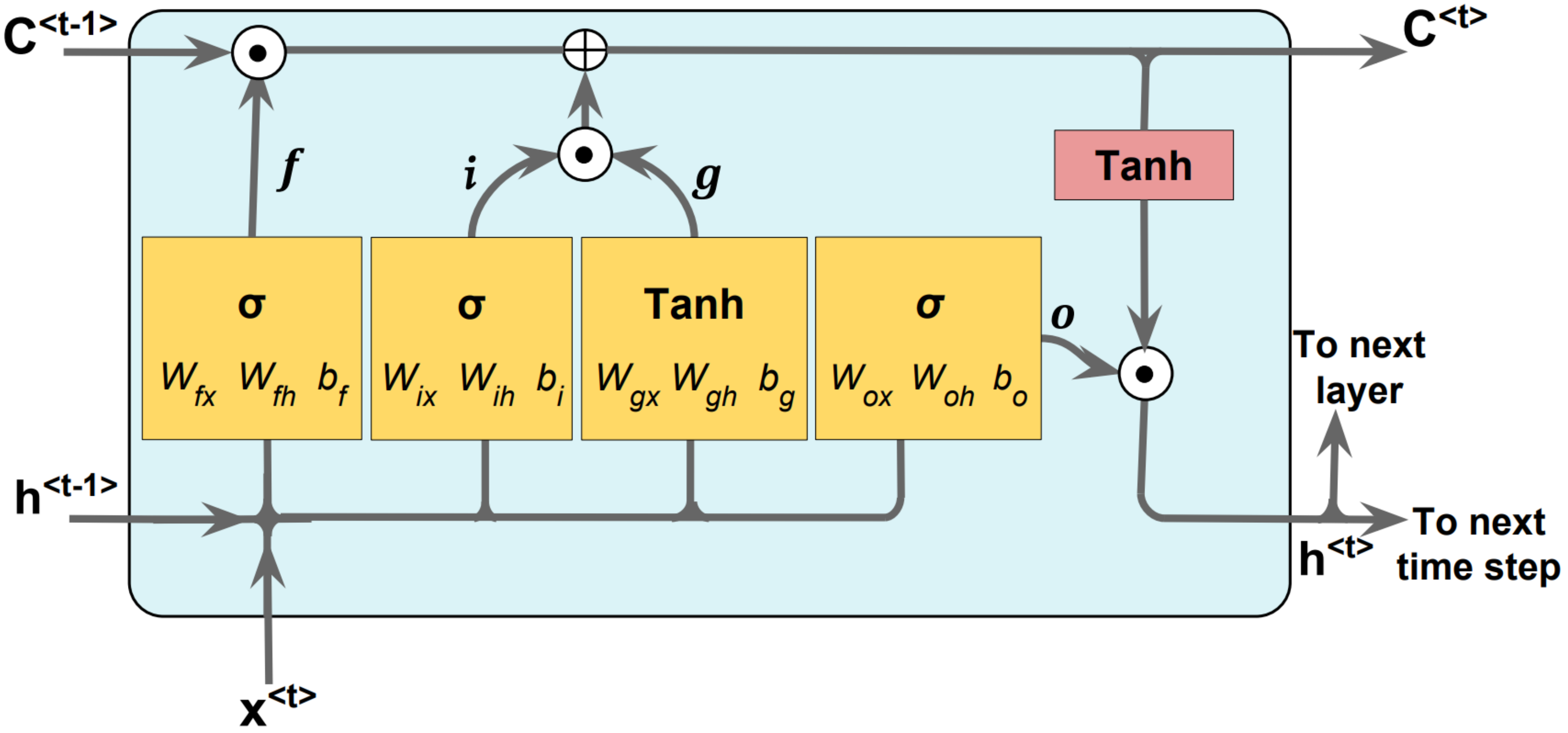
LSTM Tasks and Usage Over Time : 2022





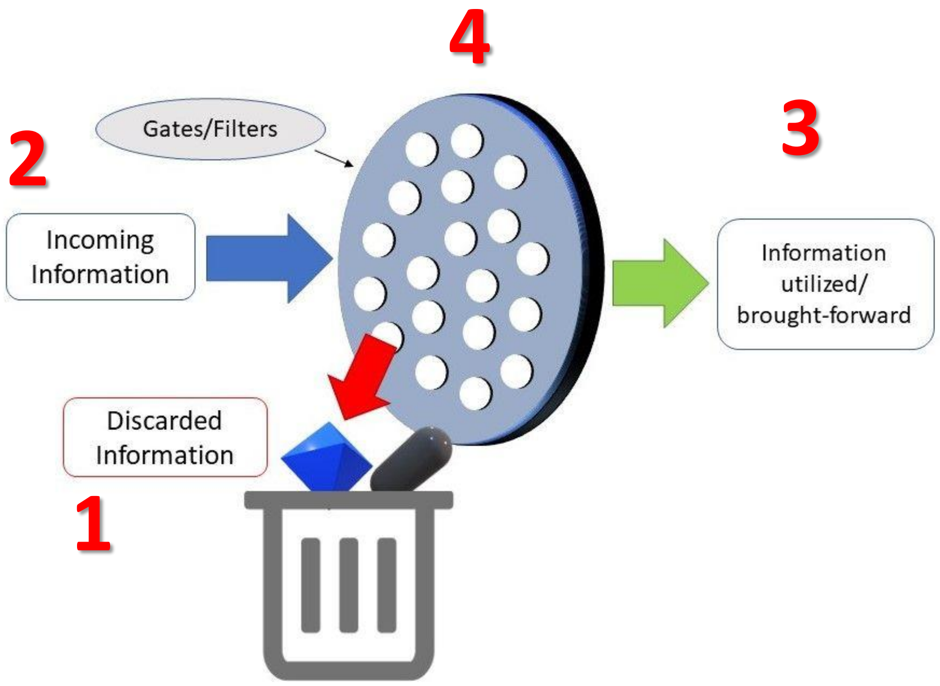
<source> https://paperswithcode.com/method/lstm

Long-short term memory (LSTM)



❖ Cell state

✓ Gates control the flow of information to/from the memory



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

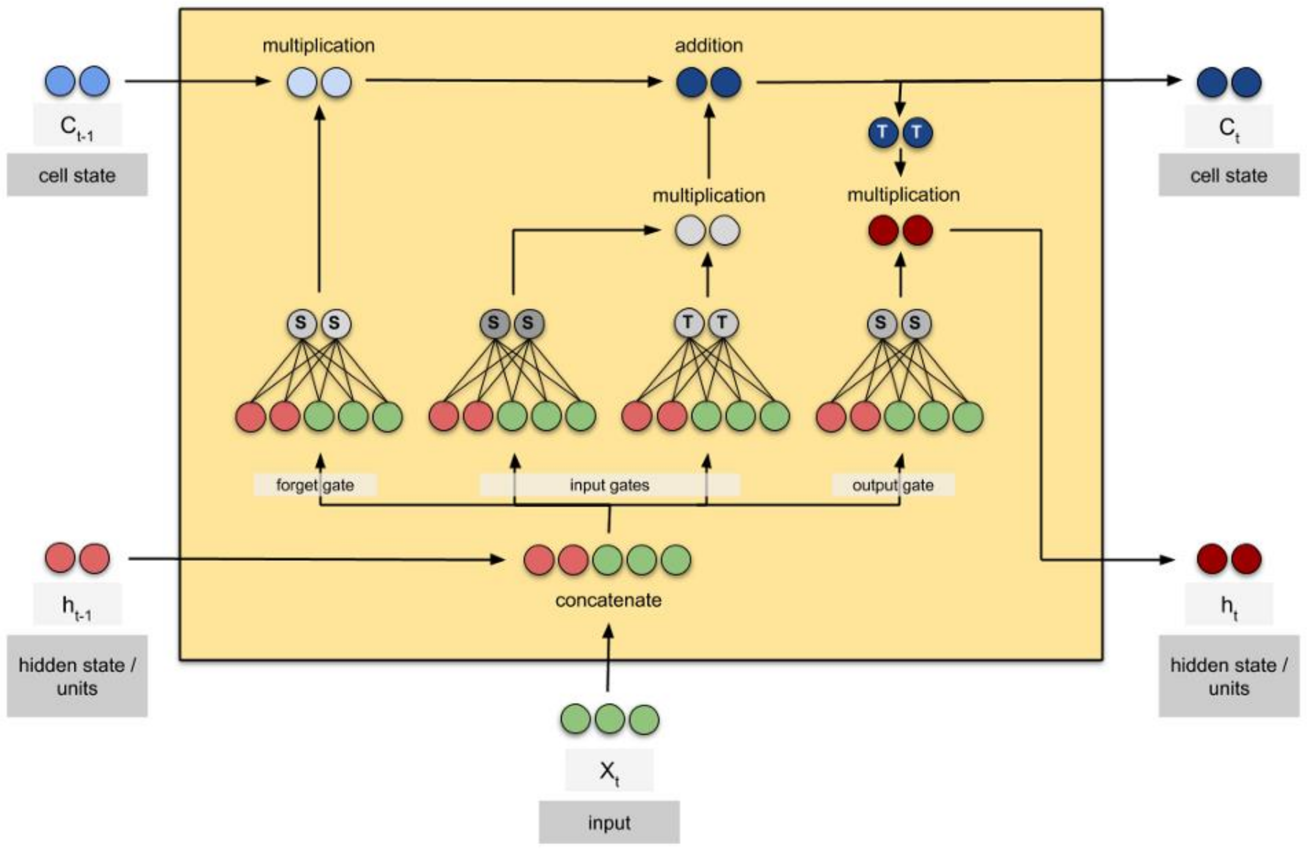
- gating architectures allow information flow to be controlled more carefully
  - ✓ should we copy the previous state, or replace it?

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

$$\underset{\mathbf{s}'}{\begin{bmatrix} 8 \\ 11 \\ 3 \\ 7 \\ 5 \\ 15 \end{bmatrix}} \leftarrow \underset{\mathbf{g}}{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}} \odot \underset{\mathbf{x}}{\begin{bmatrix} 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ 15 \end{bmatrix}} + \underset{(\mathbf{1}-\mathbf{g})}{\begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}} \odot \underset{\mathbf{s}}{\begin{bmatrix} 8 \\ 9 \\ 3 \\ 7 \\ 5 \\ 8 \end{bmatrix}}$$

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

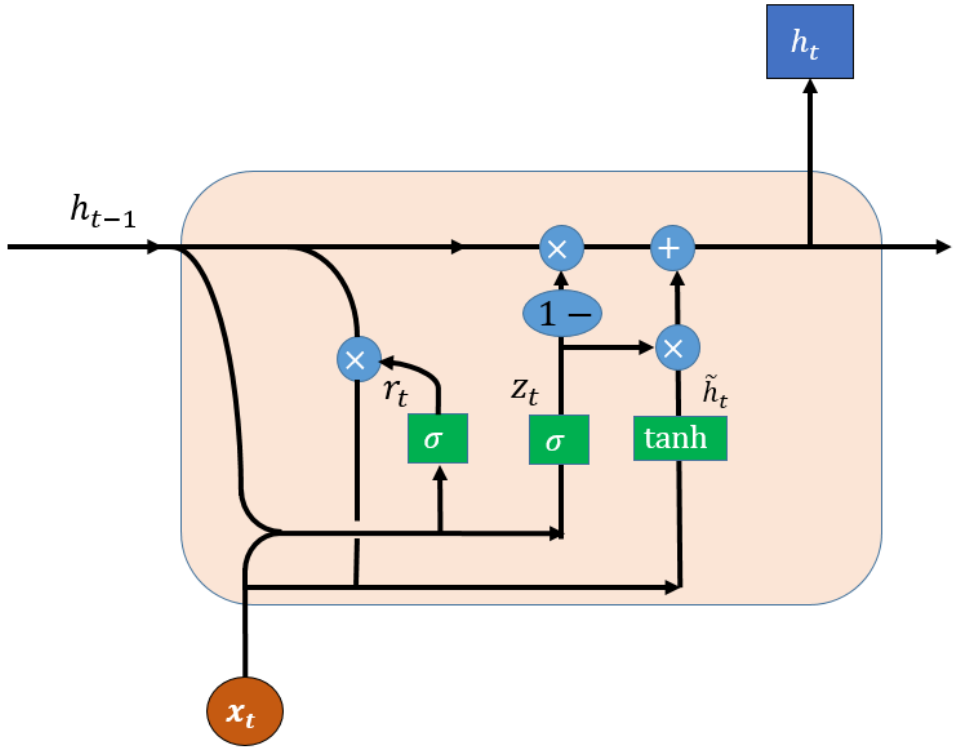
- Gates are controlled by a concatenation of the output from the previous time step and the current input and optionally the cell state vector.



S: Sigmoid
T: tanh

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

limitations of RNNs

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

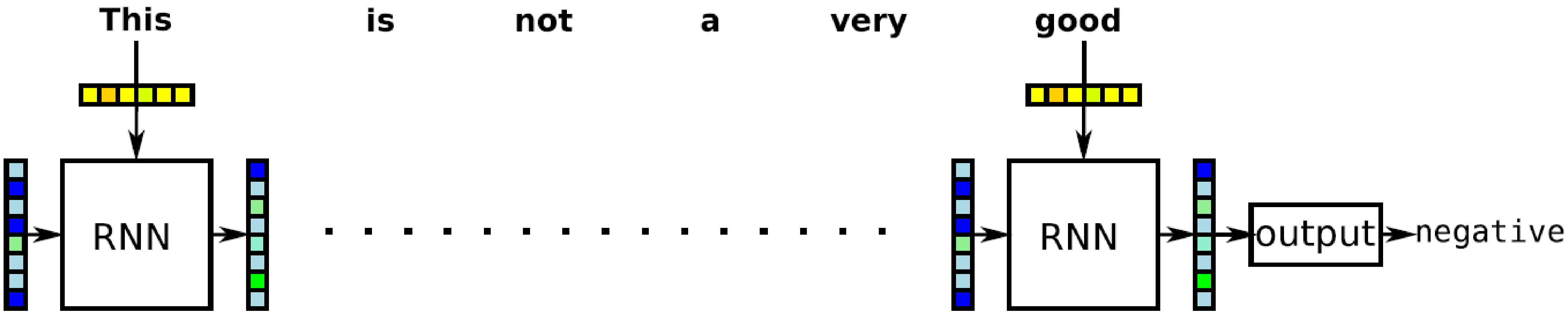
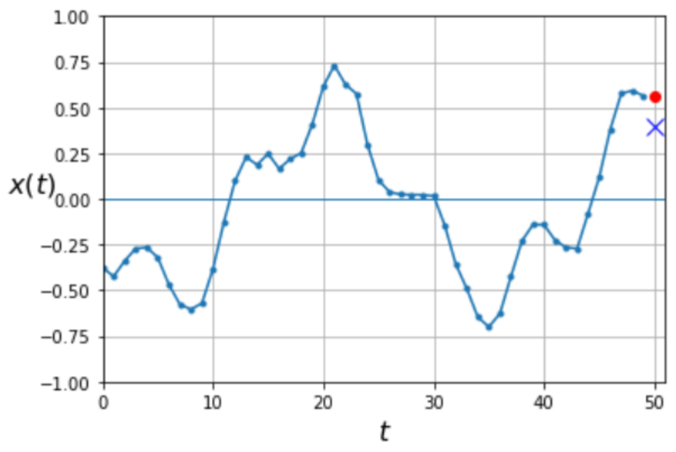
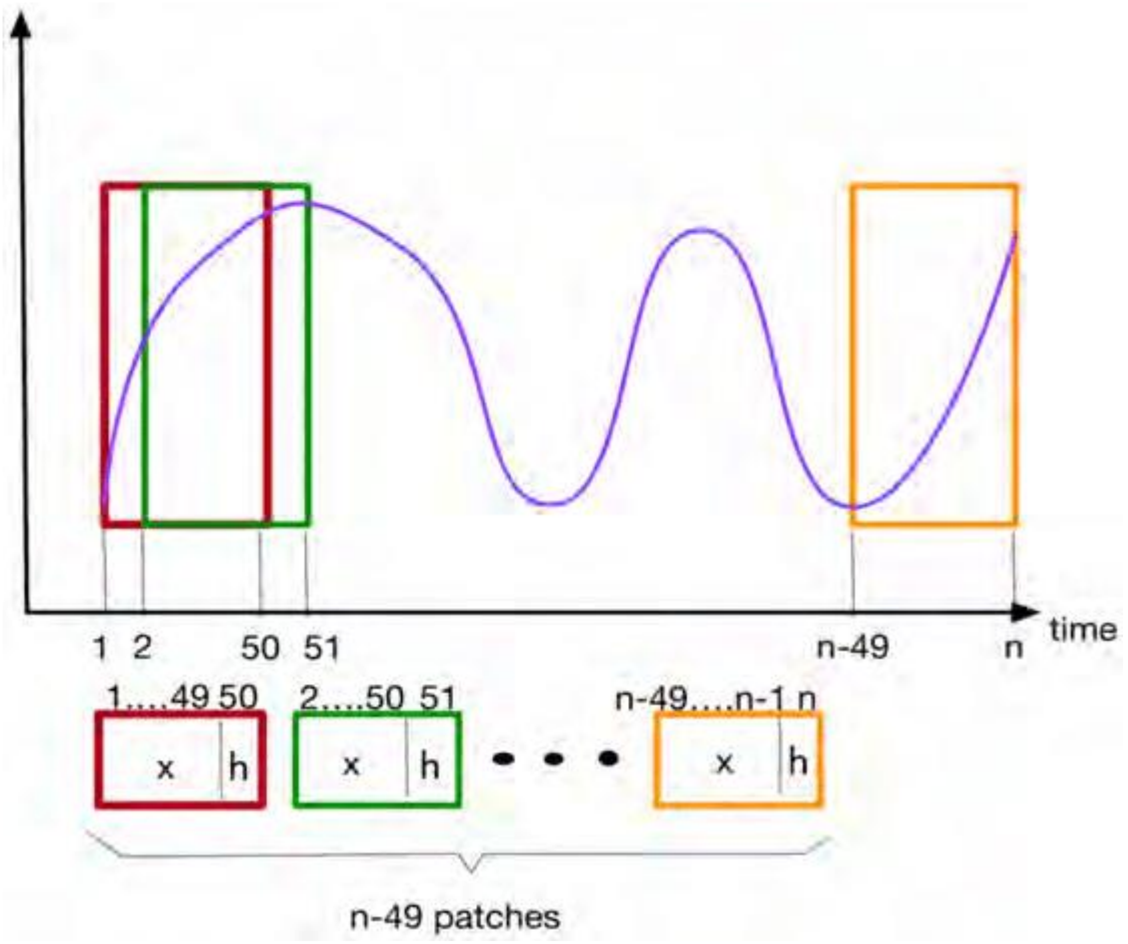


image borrowed from Richard Johansson (*Chalmers Technical University and University of Gothenburg)*

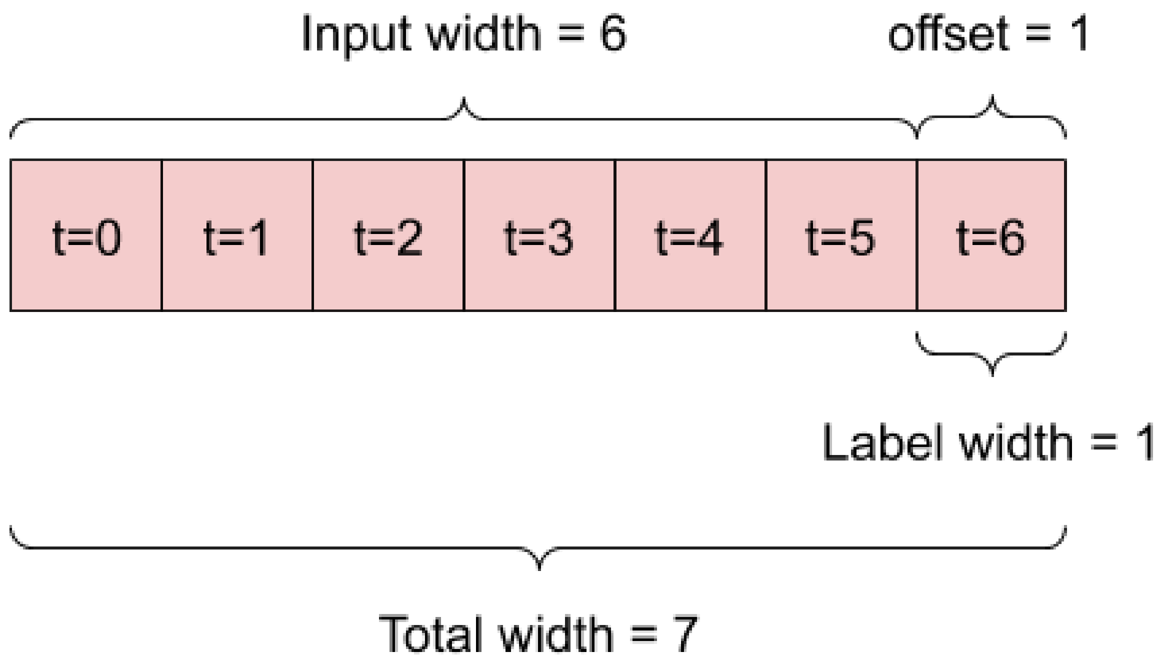
Simple LSTM layer

RNN Sliding window size and horizon



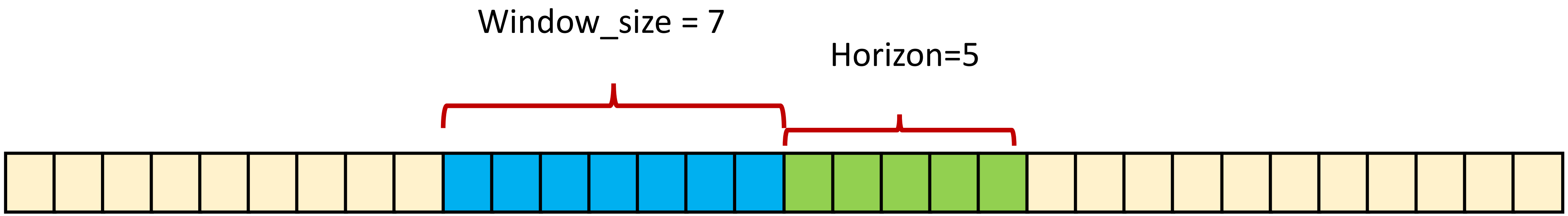


❖ A model : prediction one hour into the future, given six hours of history

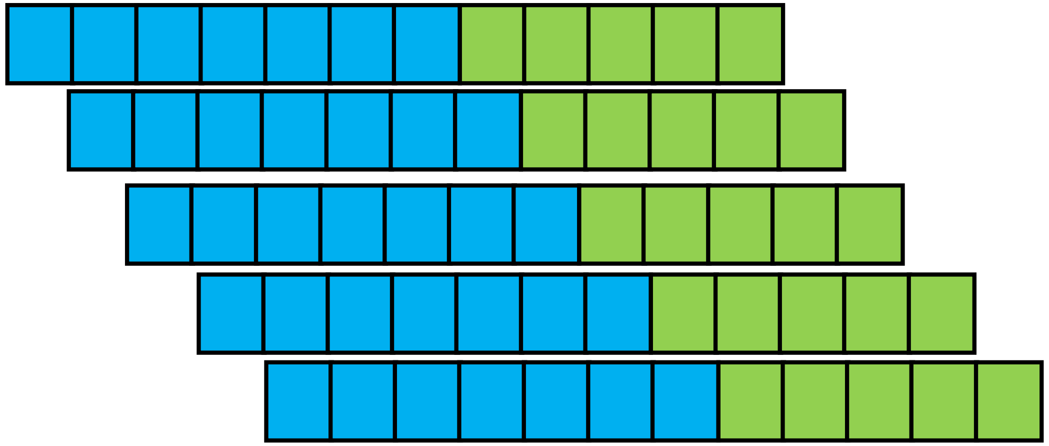


Many-to-One RNN Data Structure

Step1: set the number of window_size, horizon



Step2: set the number of window_size, horizon



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.

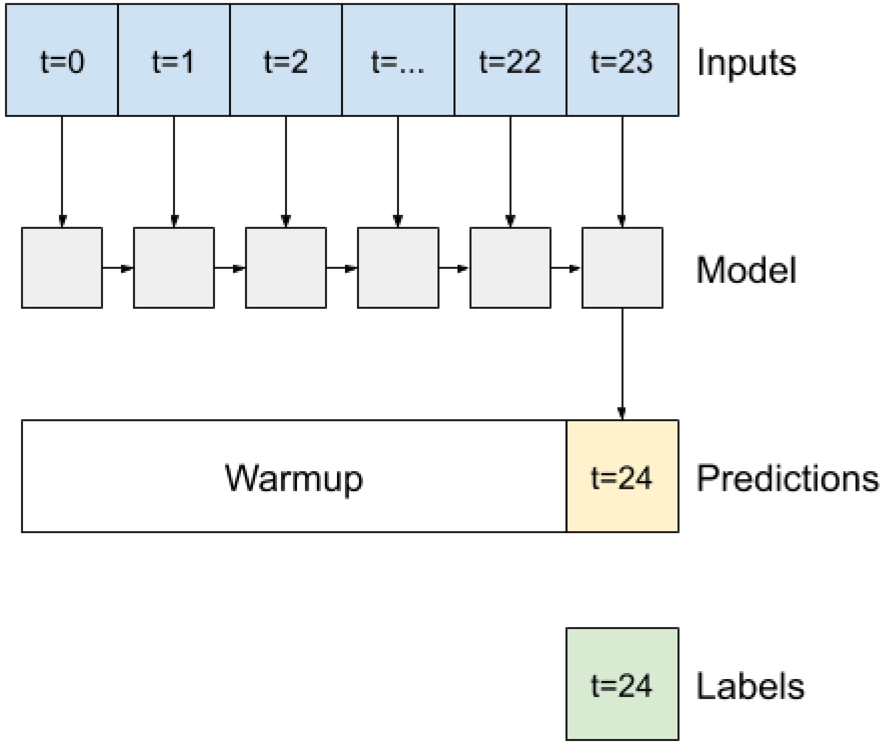
✓ **return_state**: whether to return the last state in addition to the output. (cell state, c)

- Default: False.

return_sequences=False (the default)

❖ Return_sequences=False

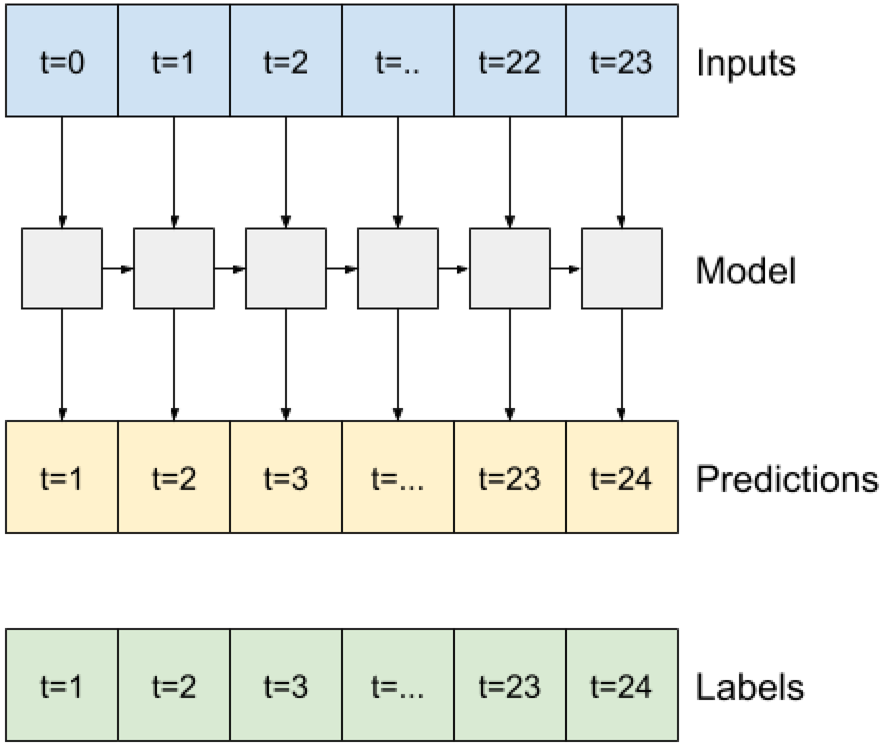
- ✓ the layer only returns the output of the final time step
- ✓ giving the model time to warm up its internal state before making a single prediction:



❖ 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



a sequence modeling problem :

predict the next word

a sequence modeling problem

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

given these words

predict what comes next?

❖ idea: use a fixed window

“This morning I took the dog for a walk.”

given these 2 words,
predict what comes next?

[1 0 0 0 0 0 1 0 0 0]

for a

prediction

One hot feature vector
indicates what each word is

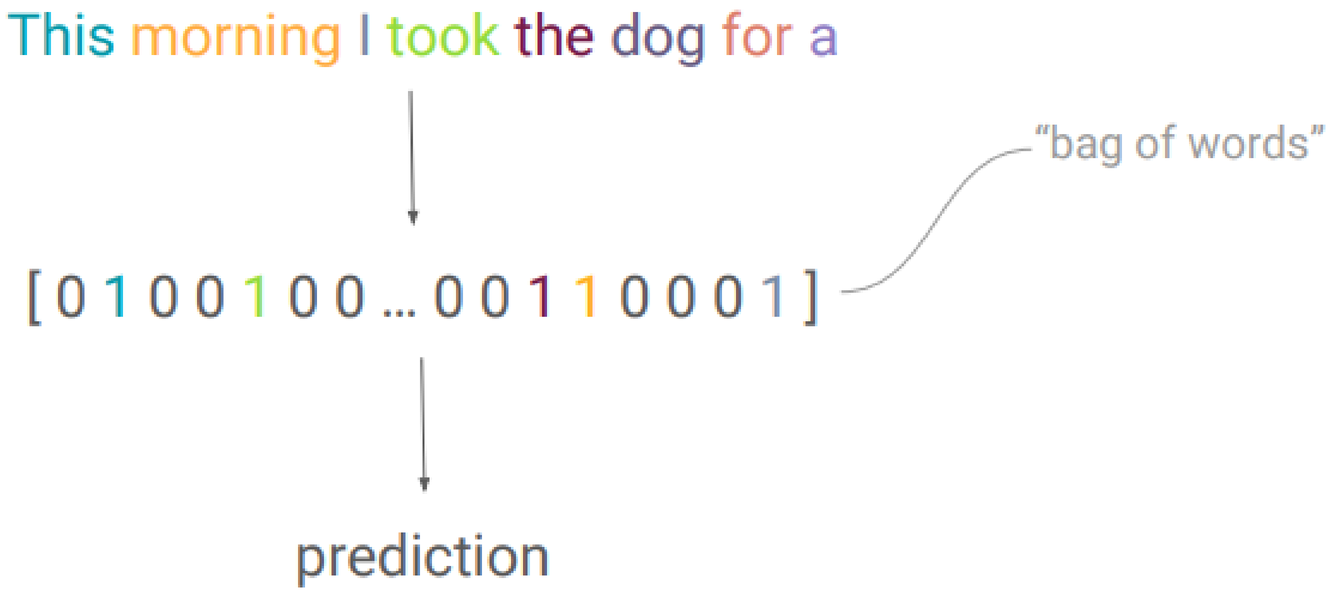
problem: we can’t model long-term dependencies

“In France, I had a great time and I learnt some of the _____ language.”

We need information from the far past and future to accurately guess the correct word.

idea: use entire sequence, as a set of counts

Bag-of-words model:
In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.



problem: counts don't preserve order

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

vs

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

idea: use a really big fixed window

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

given these 7 words, predict the next word

[1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 ...]

morning I took the dog ...

prediction

problem: no parameter sharing

this morning

[1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 ...]

each of these inputs has a *separate parameter*

this morning

[0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 ...]

things we learn about the sequence **won't transfer** if they appear **at different points** in the sequence.

Why Sequence Models?

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

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/

KISTI SINCE 1962

❖ SimpleRNN with a Dense layer

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

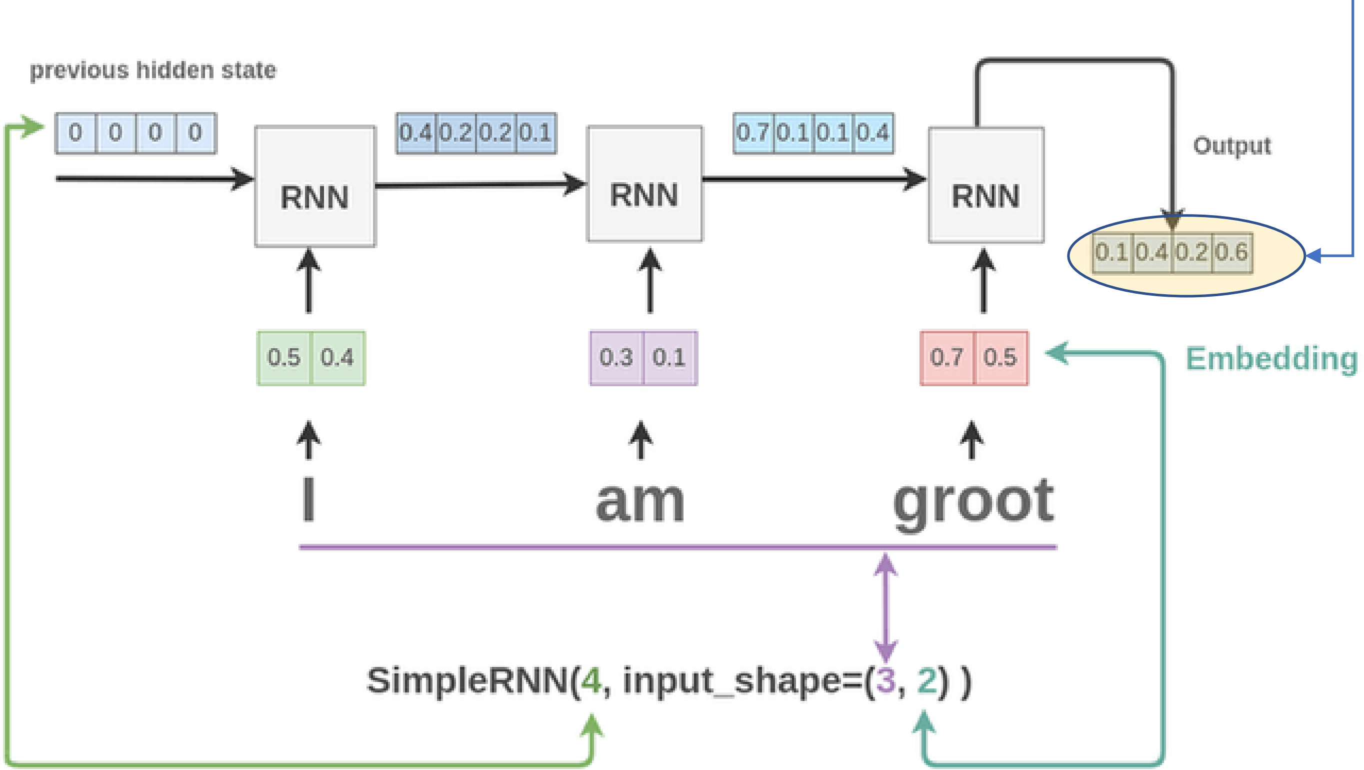
```
import tensorflow as tf
from tensorflow.keras.layers import SimpleRNN   #Dense, LSTM
# from tensorflow.keras.models import Sequential

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)
```

KISTI SINCE 1962

- 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)
```

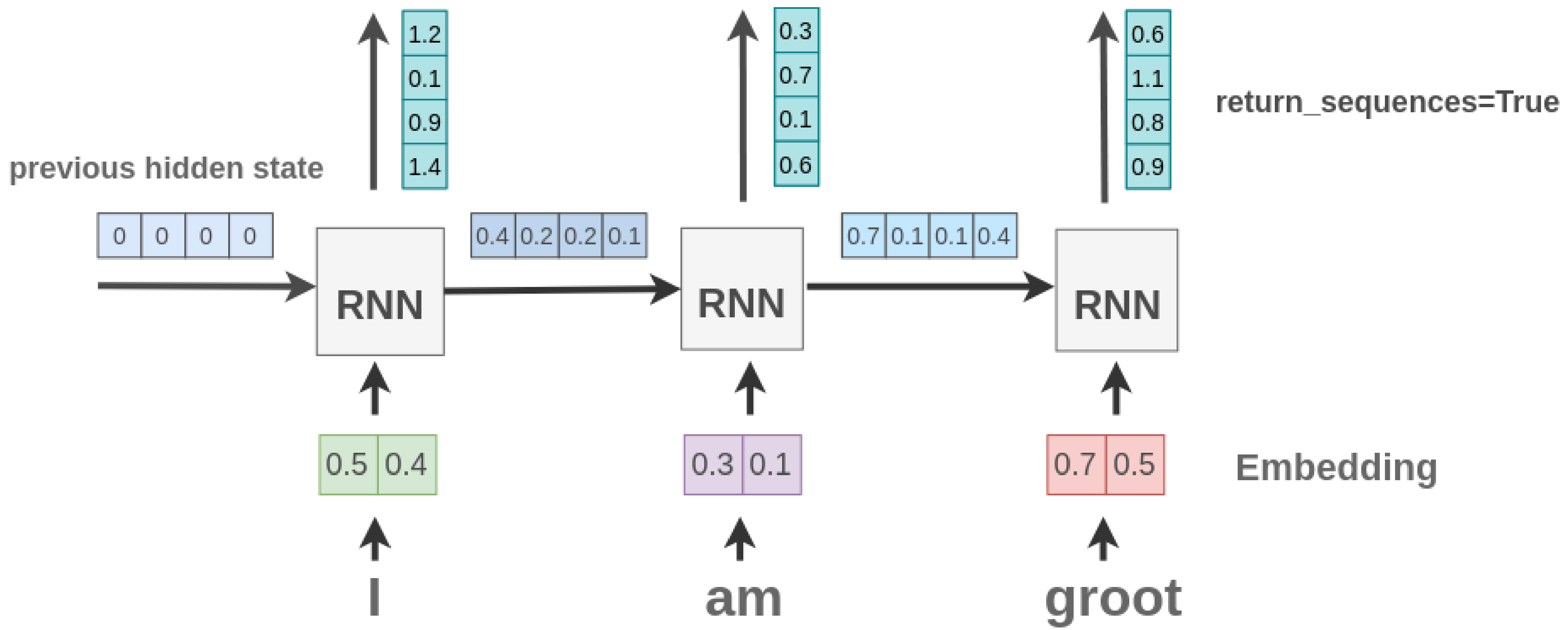
```
(1, 3, 4)
tf.Tensor(
[[[-0.6854385   0.08265962  0.30888444 -0.30752325]
  [ 0.4584542  -0.1935767  -0.91095936 -0.2416075 ]
  [ 0.7241105  -0.49960855 -0.5059616   0.7261468 ]]], shape=(1, 3, 4), dtype=float32)
```

KISTI SINCE 1962

❖ return_sequences = True

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

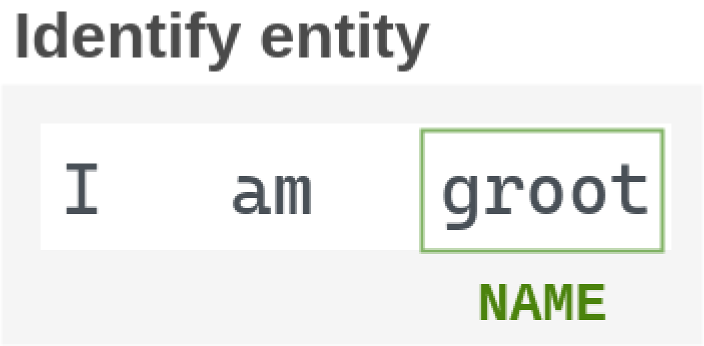
```
model.add(SimpleRNN(4, input_shape=(3, 2), return_sequences=True))
```



RNN with TimeDistributed : Many-to-Many

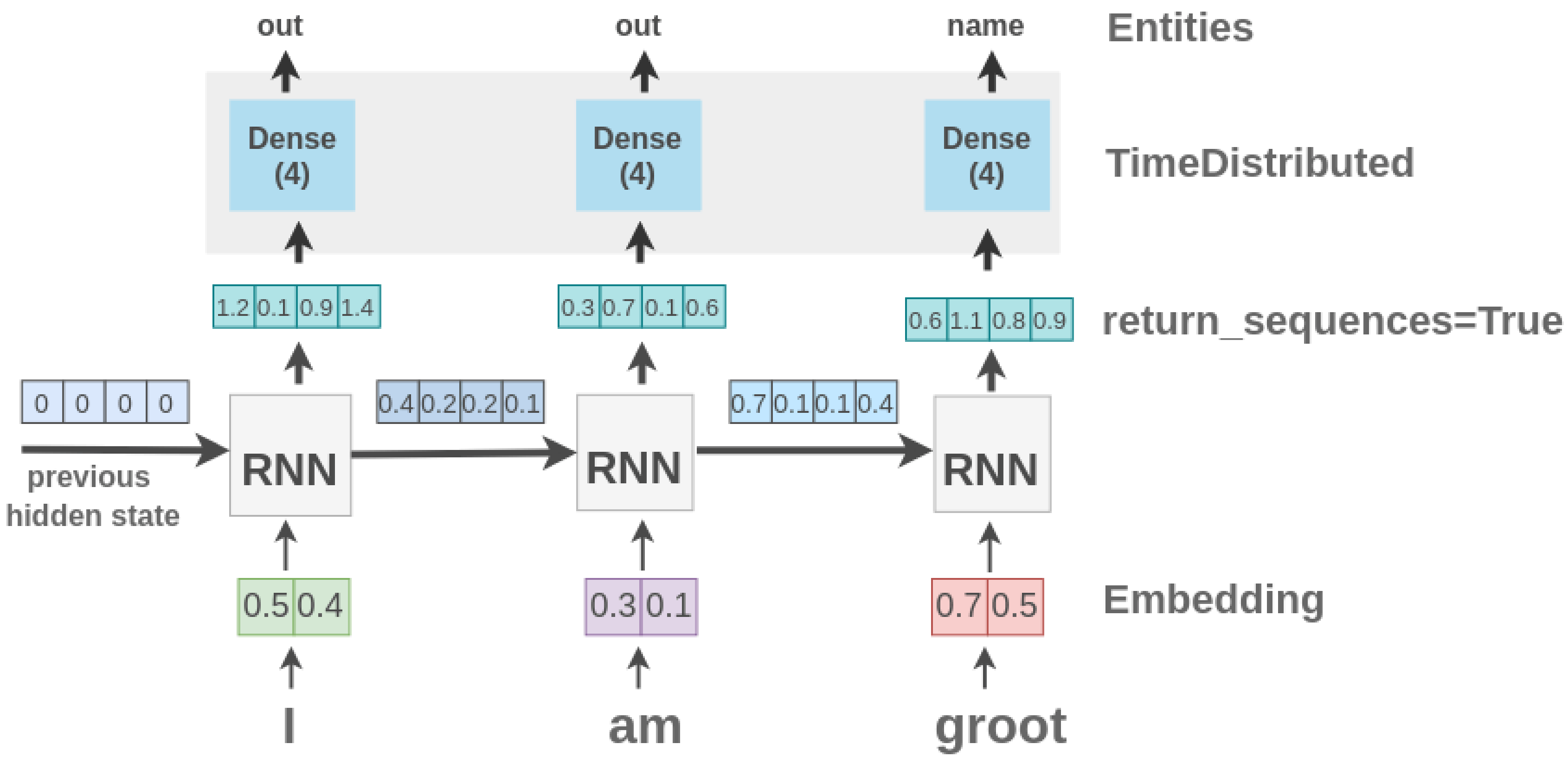
❖ Suppose we want to recognize entities in a text.

✓ For example, in our text "I am Groot", we want to identify "Groot" as a name.



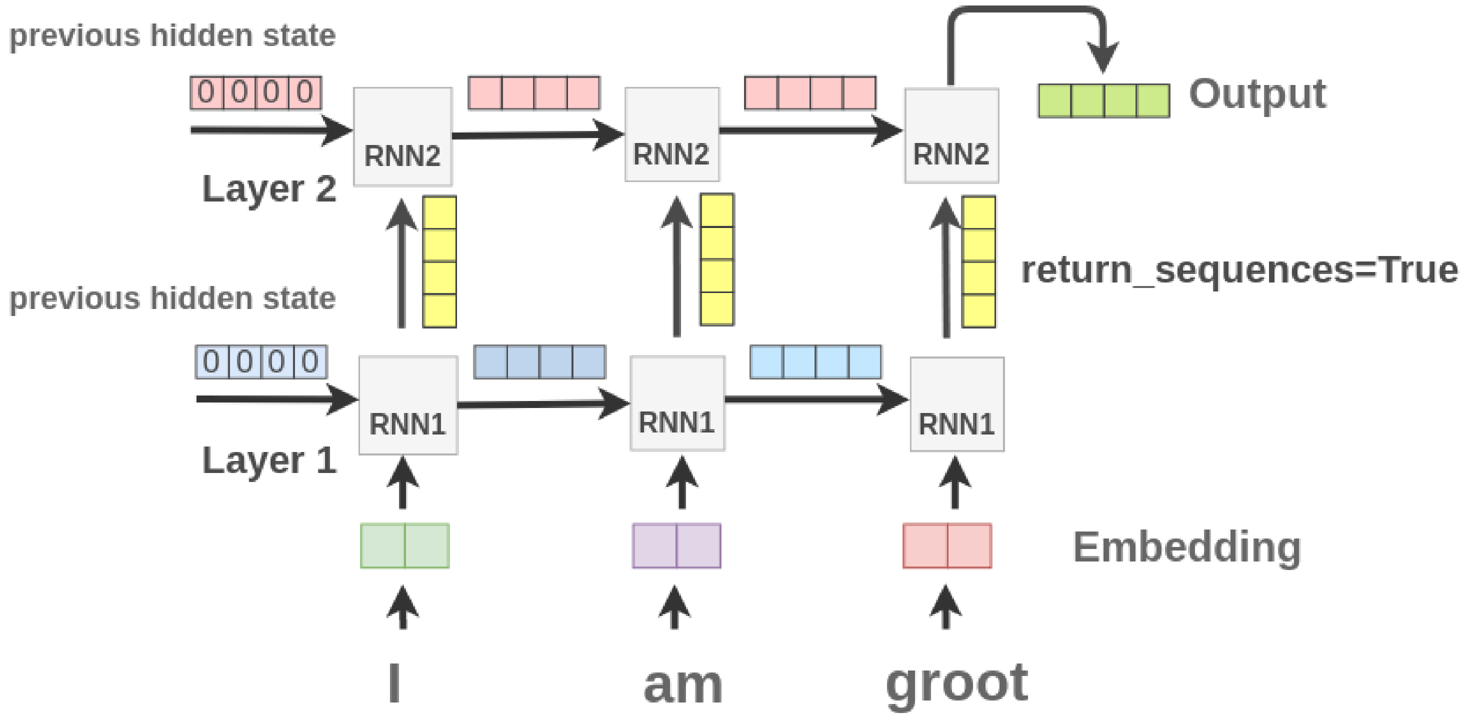
(3) RNN: TimeDistributed Layer

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



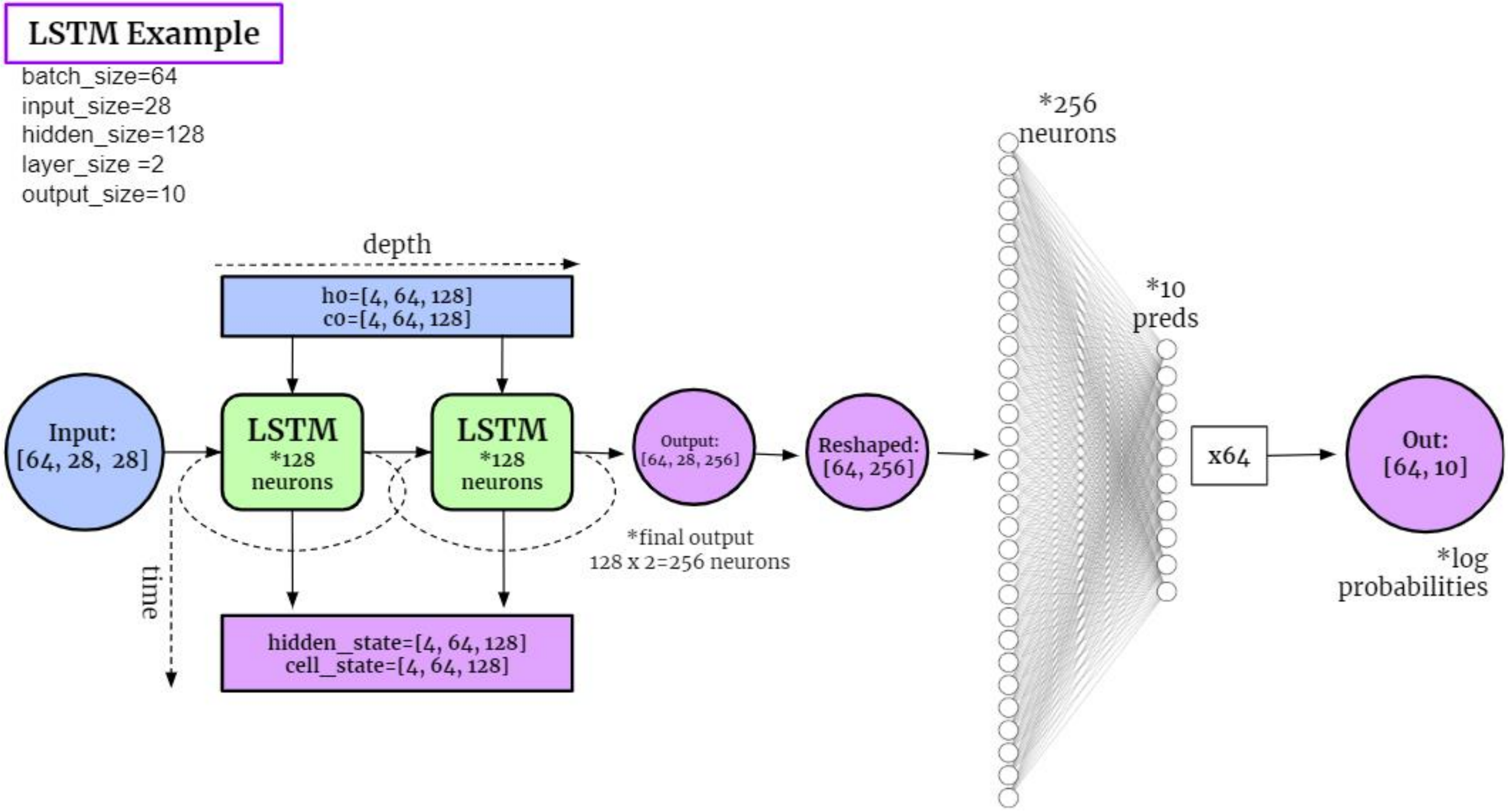
- We can also stack multiple recurrent layers one after another in Keras

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



Let’s Code 2:
LSTM

KISTI SINCE 1962



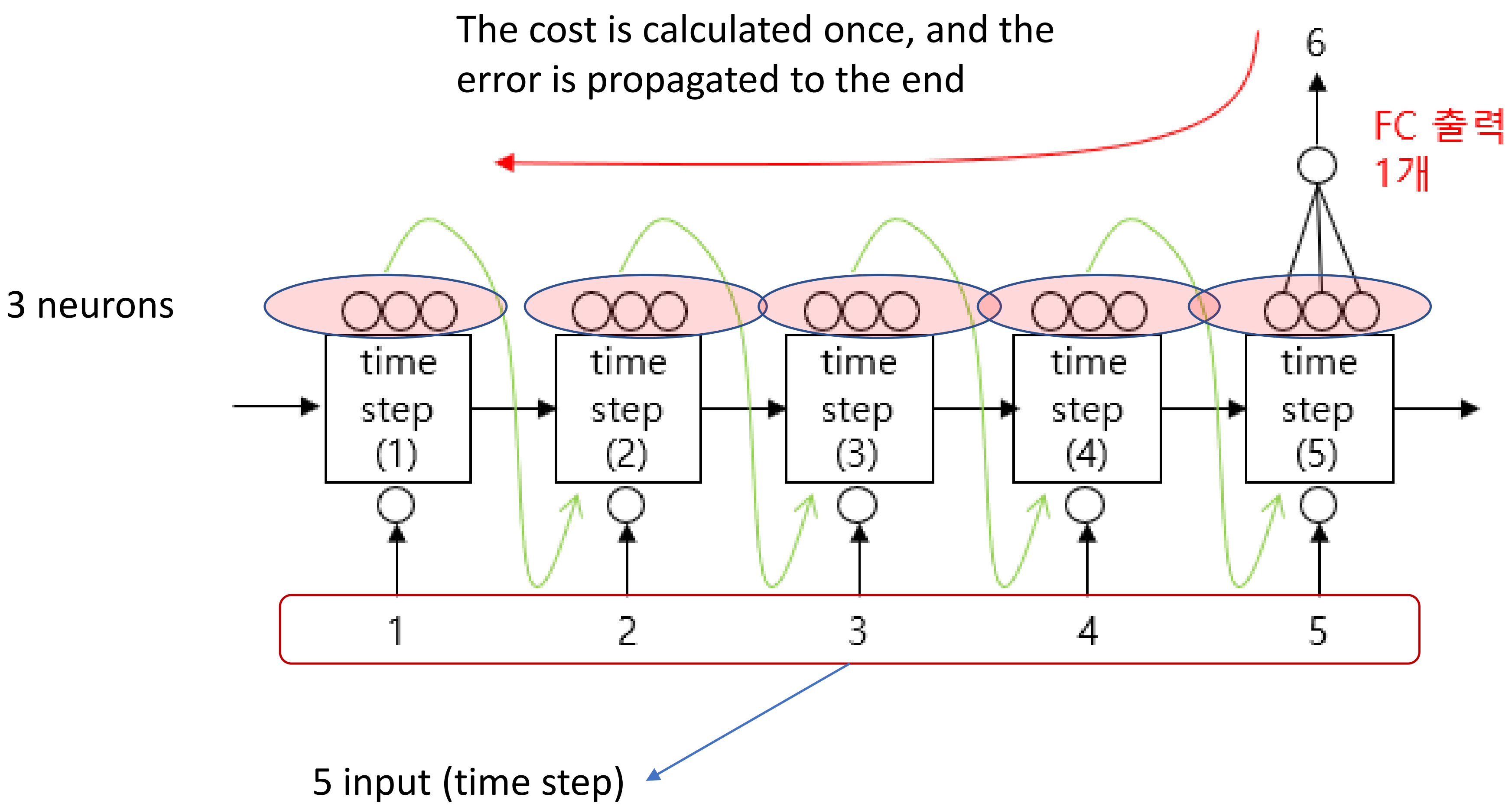
LTSM Example : many-to-one

```
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)
```

```
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

KISTI SINCE 1962

2022

Korea Institute of Science and Technology Information

TRUST KISTI