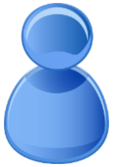
Recurrent Neural Network (RNN)

Example Application

- Slot Filling



I would like to arrive Taipei on November 2nd.

ticket booking system

Slot

Destination: Taipei

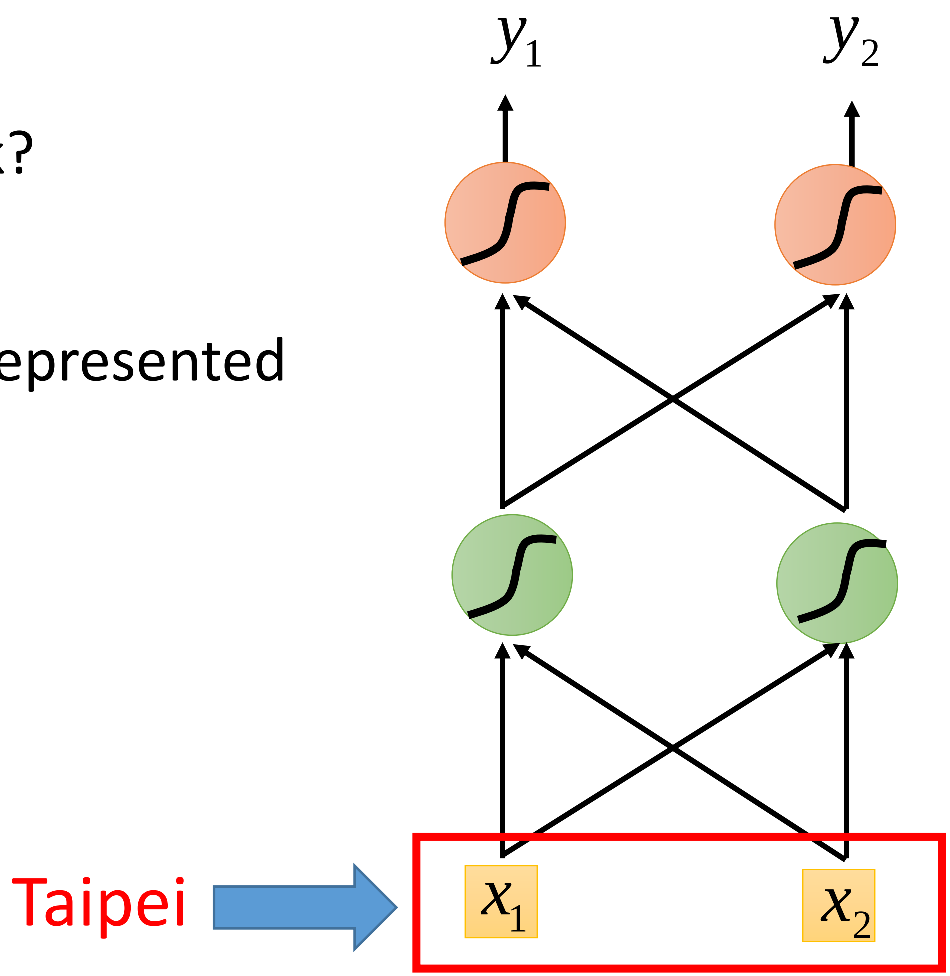
time of arrival: November 2nd

Example Application

Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)



1-of-N encoding

How to represent each word as a vector?

1-of-N Encoding lexicon = {apple, bag, cat, dog, elephant}

The vector is lexicon size.

Each dimension corresponds to a word in the lexicon

The dimension for the word is 1, and others are 0

apple = [1 0 0 0 0]

bag = [0 1 0 0 0]

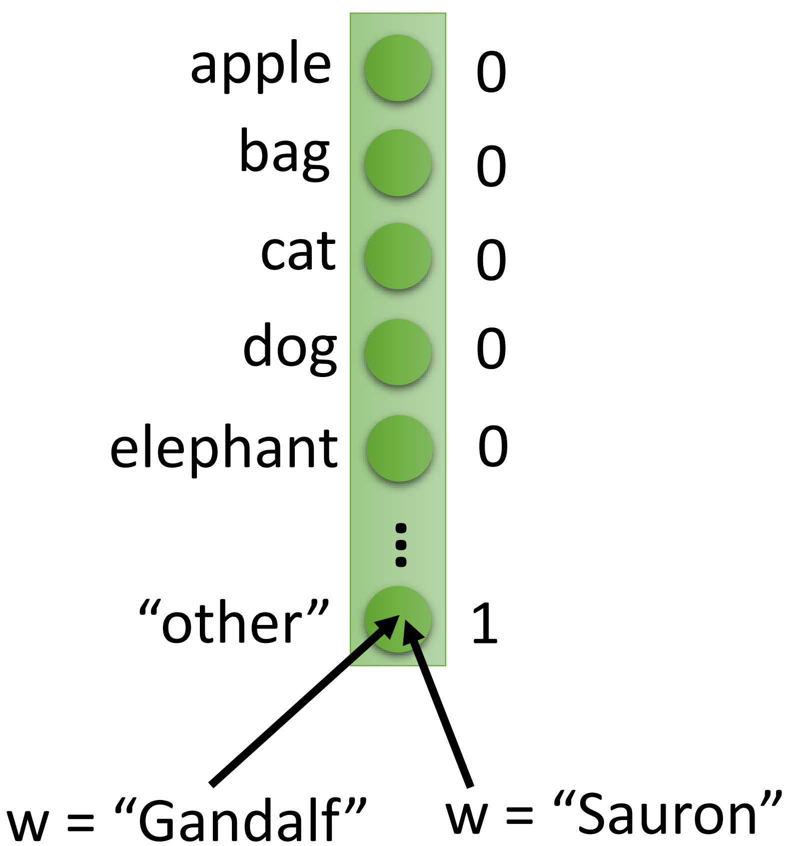
cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

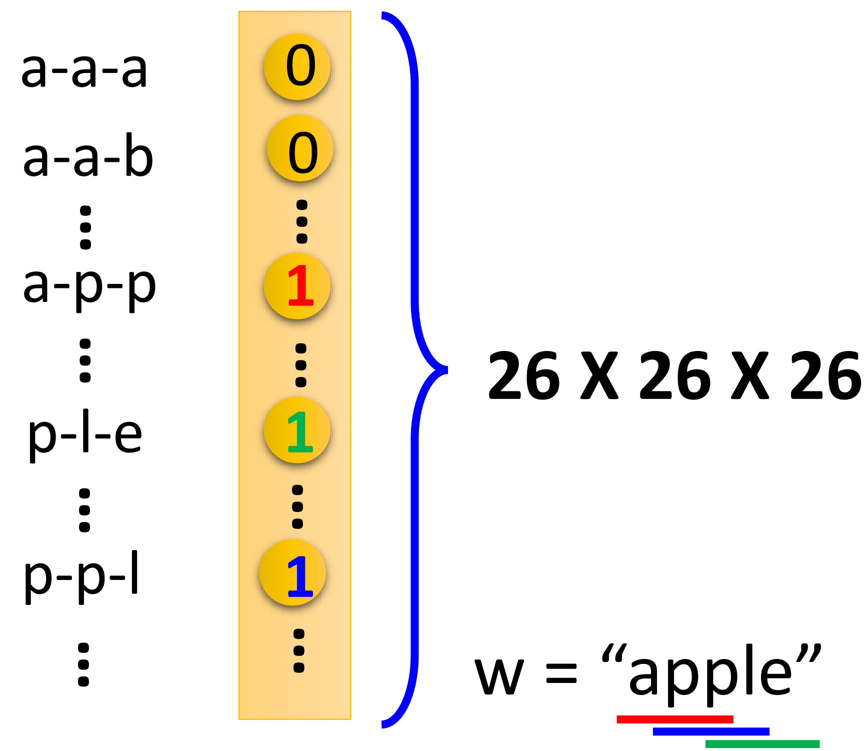
elephant = [0 0 0 0 1]

Beyond 1-of-N encoding

Dimension for "Other"



Word hashing



Example Application

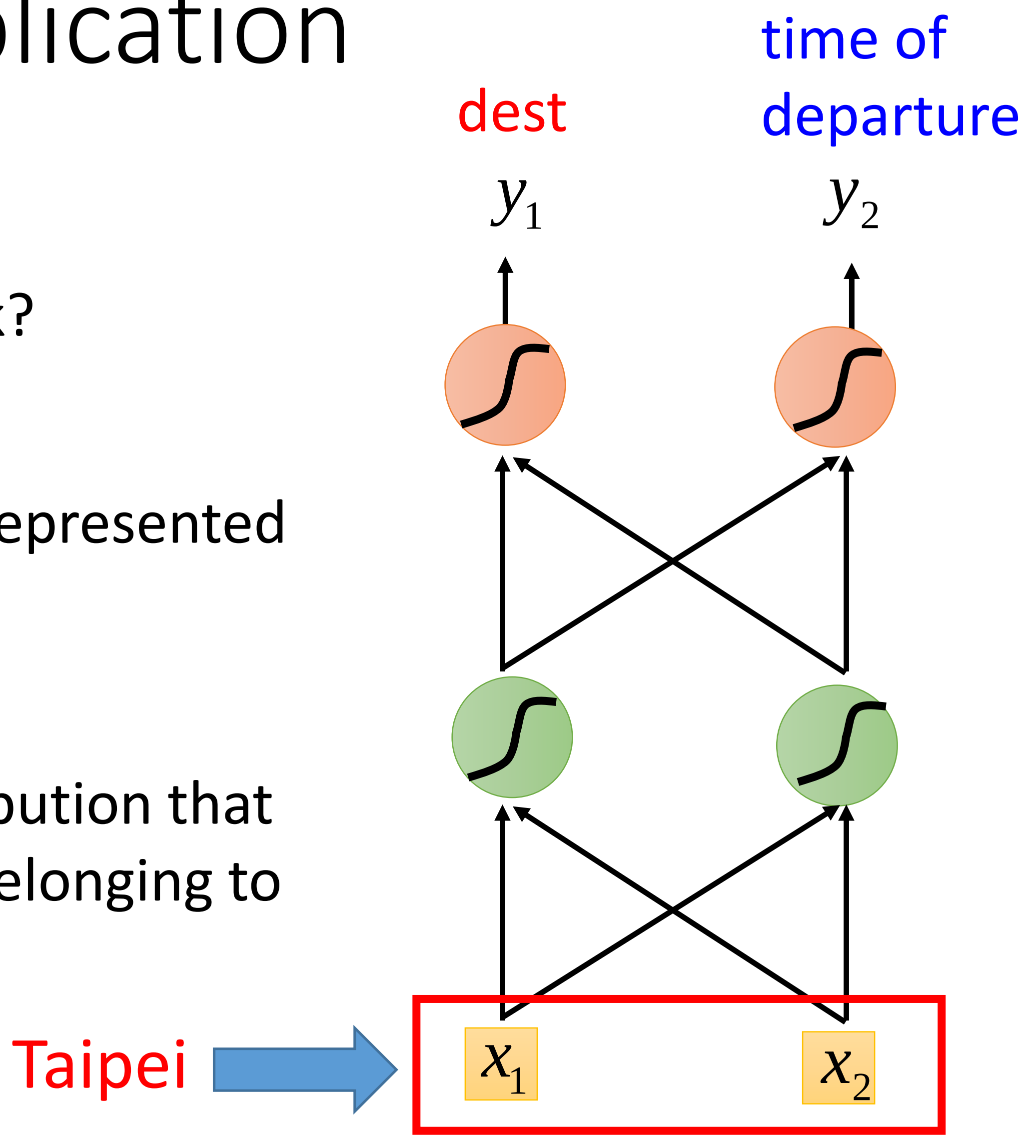
Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)

Output:

Probability distribution that the input word belonging to the slots



Example Application

arrive Taipei on November 2^{nd}

other | dest | other | time | time

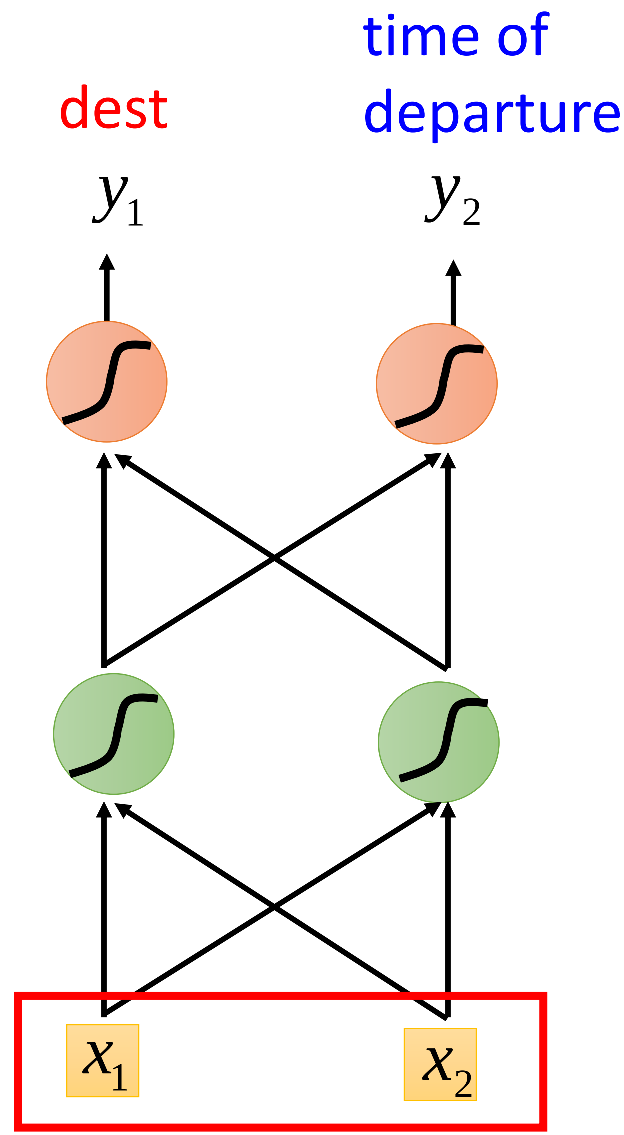
Problem?

leave Taipei on November 2^{nd}

place of departure

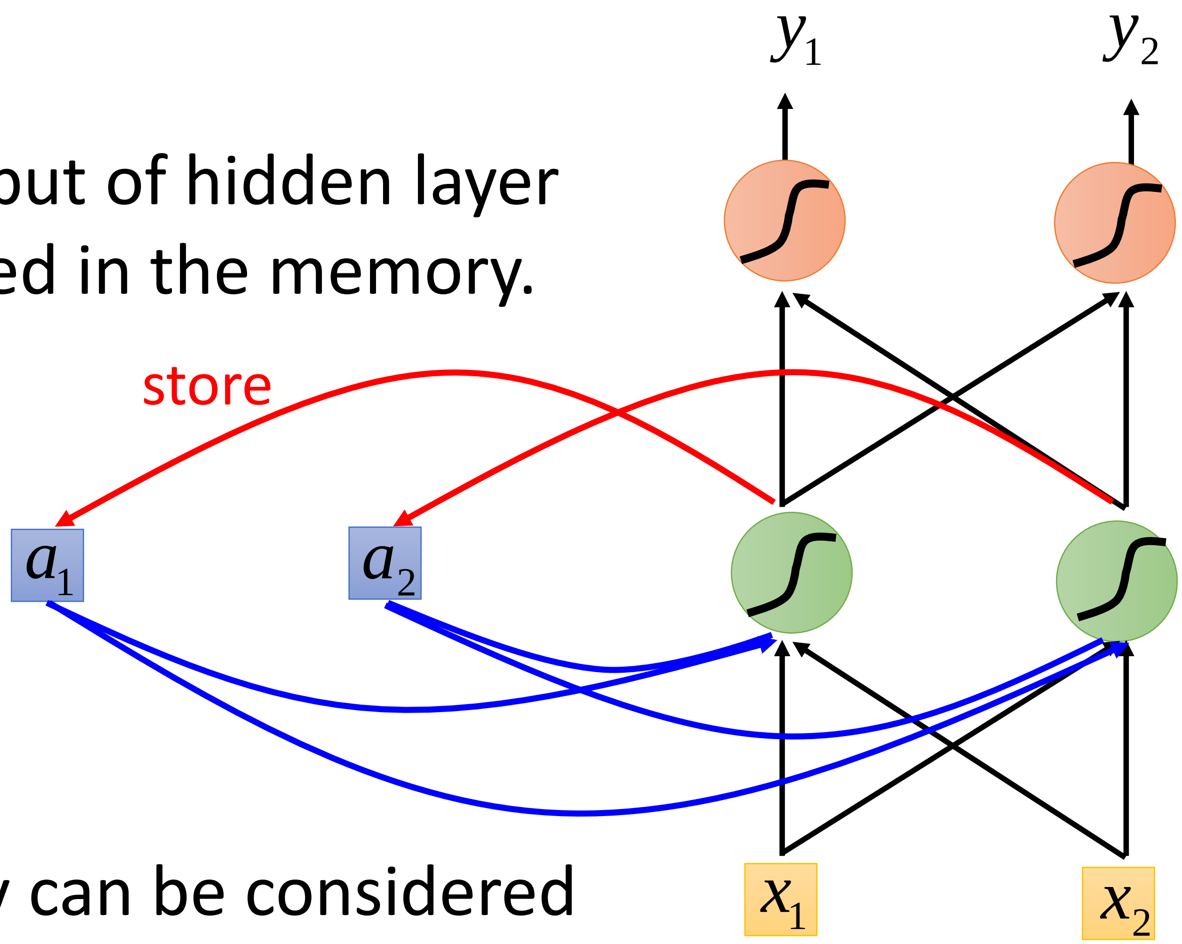
Neural network needs memory!

Taipei



Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.

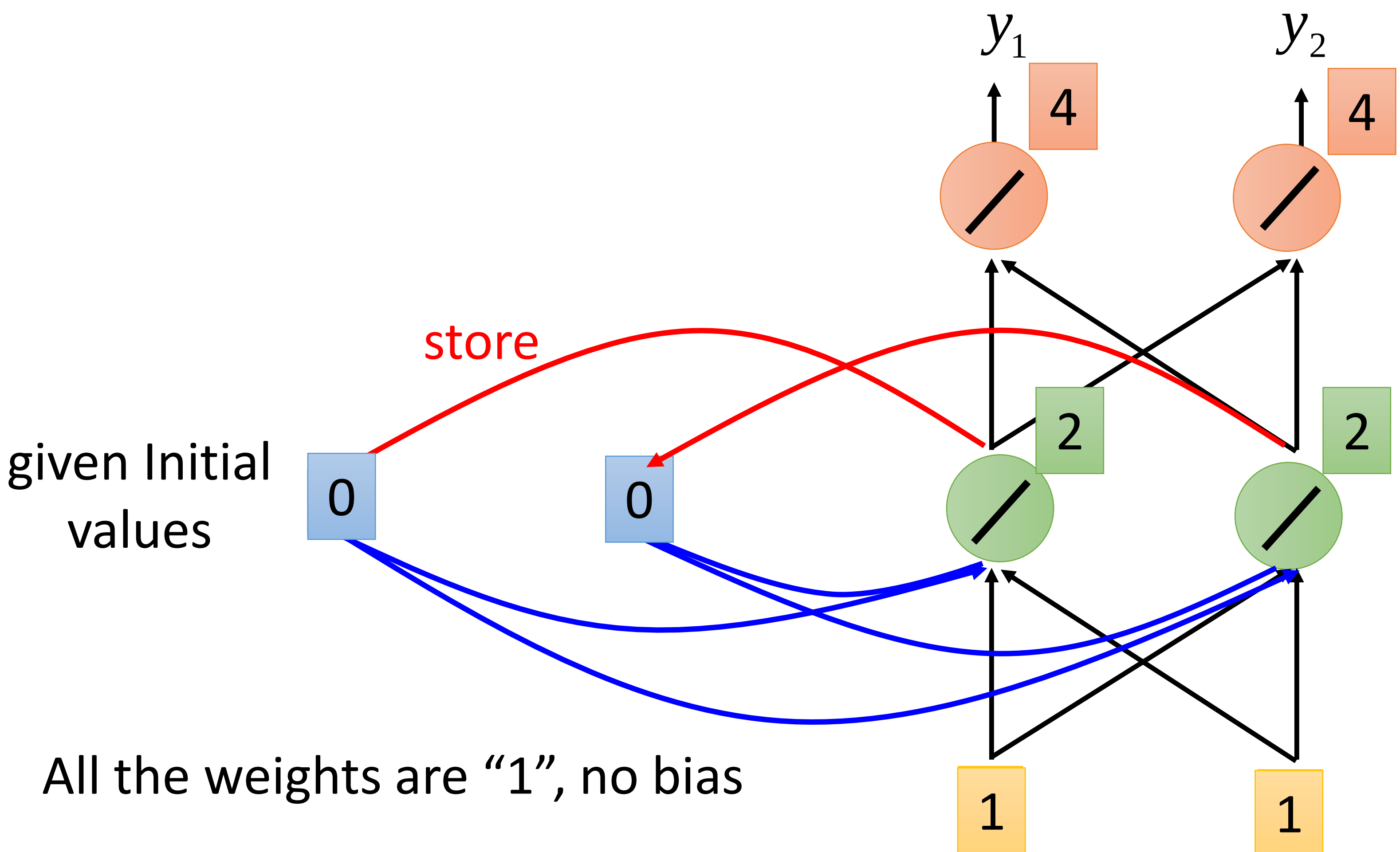


Memory can be considered as another input.

Example

Input sequence: $\begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}2\\2\end{bmatrix}$ ……

output sequence: $\begin{bmatrix}4\\4\end{bmatrix}$



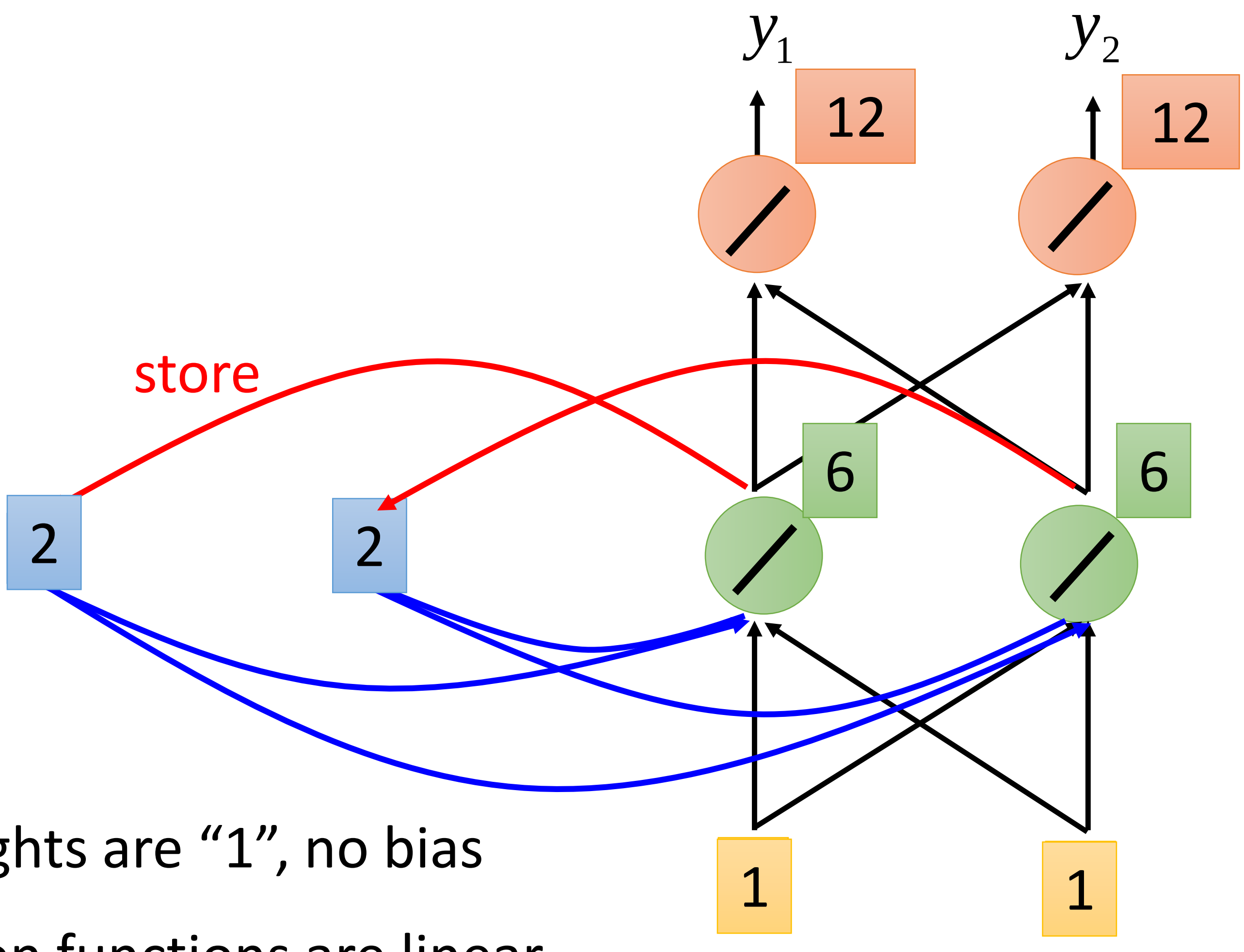
All the weights are "1", no bias

All activation functions are linear

Example

Input sequence: $\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}$ ……

output sequence: $\begin{bmatrix}4\\4\end{bmatrix}\begin{bmatrix}12\\12\end{bmatrix}$



All the weights are “1”, no bias

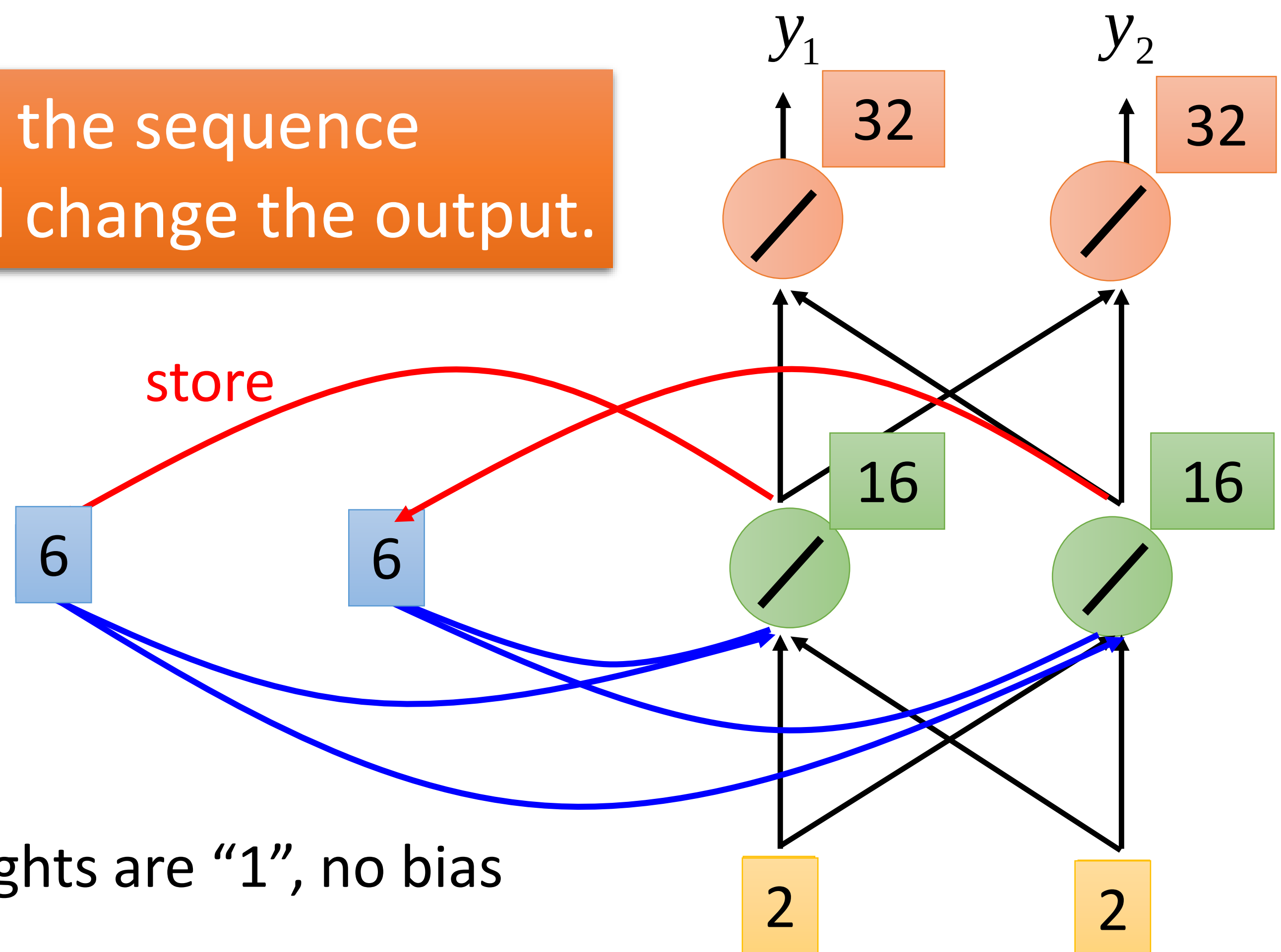
All activation functions are linear

Example

Input sequence: $\begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}2\\2\end{bmatrix}$ ……

output sequence: $\begin{bmatrix}4\\4\end{bmatrix} \begin{bmatrix}12\\12\end{bmatrix} \begin{bmatrix}32\\32\end{bmatrix}$

Changing the sequence order will change the output.

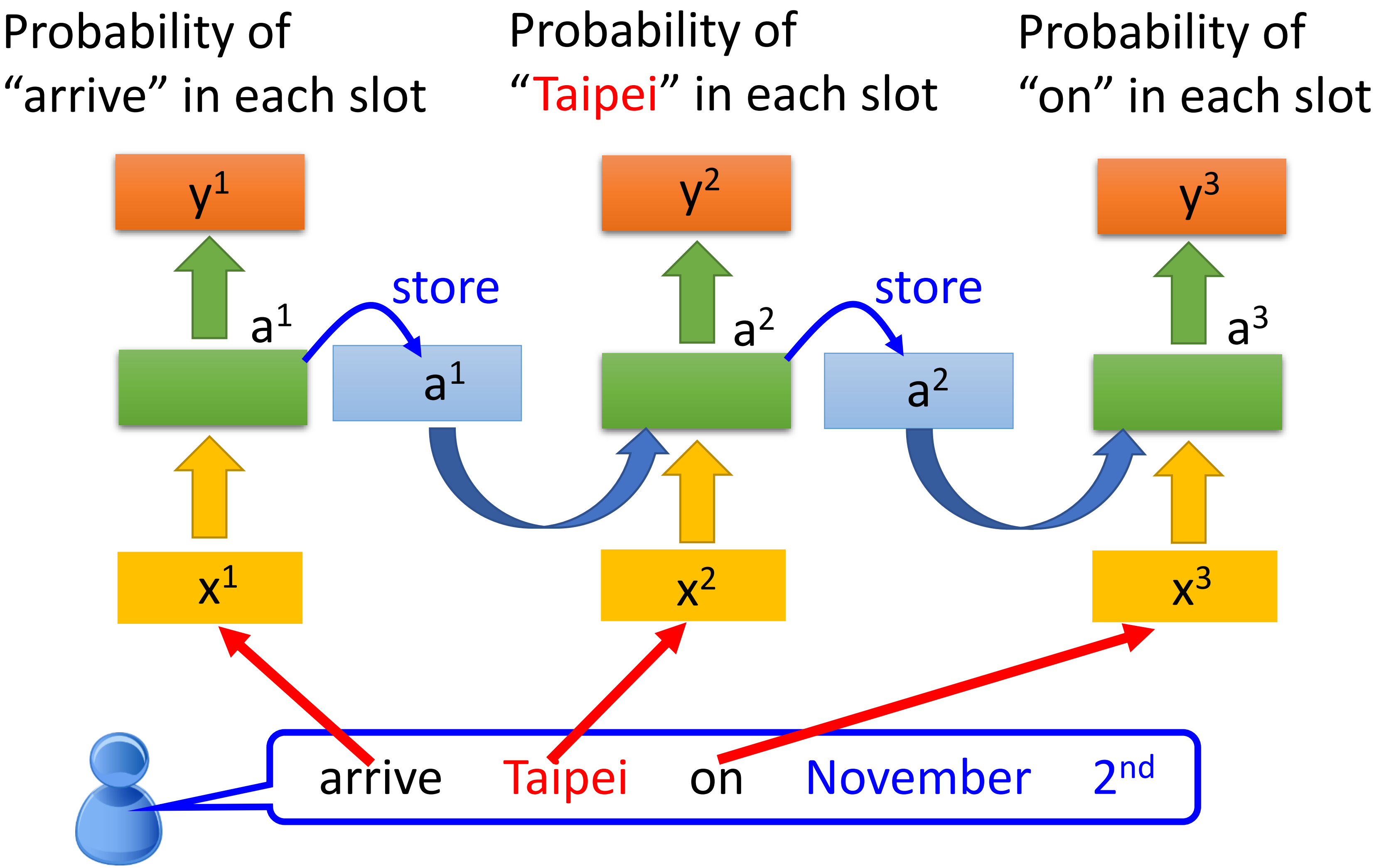


All the weights are "1", no bias

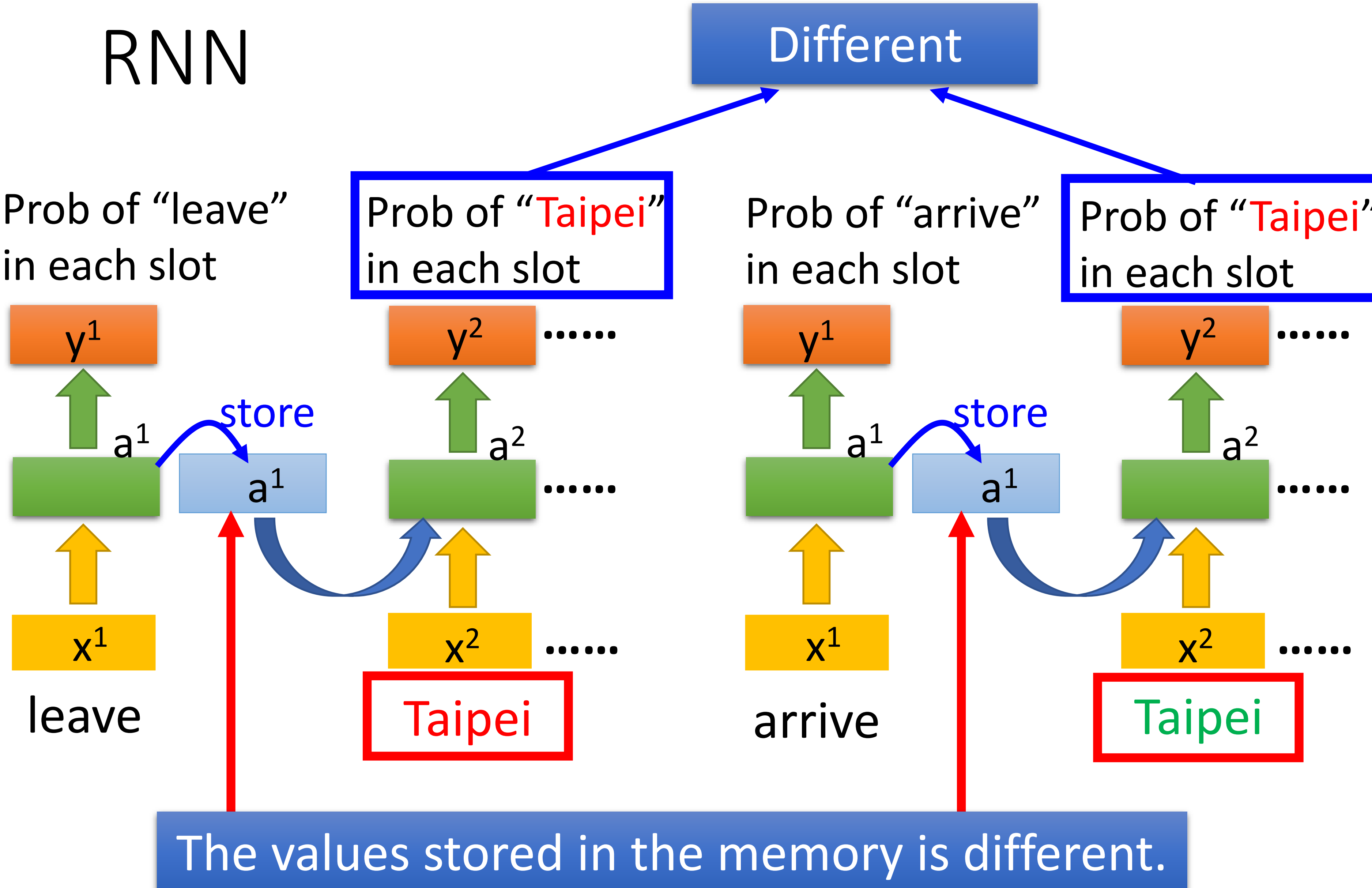
All activation functions are linear

RNN

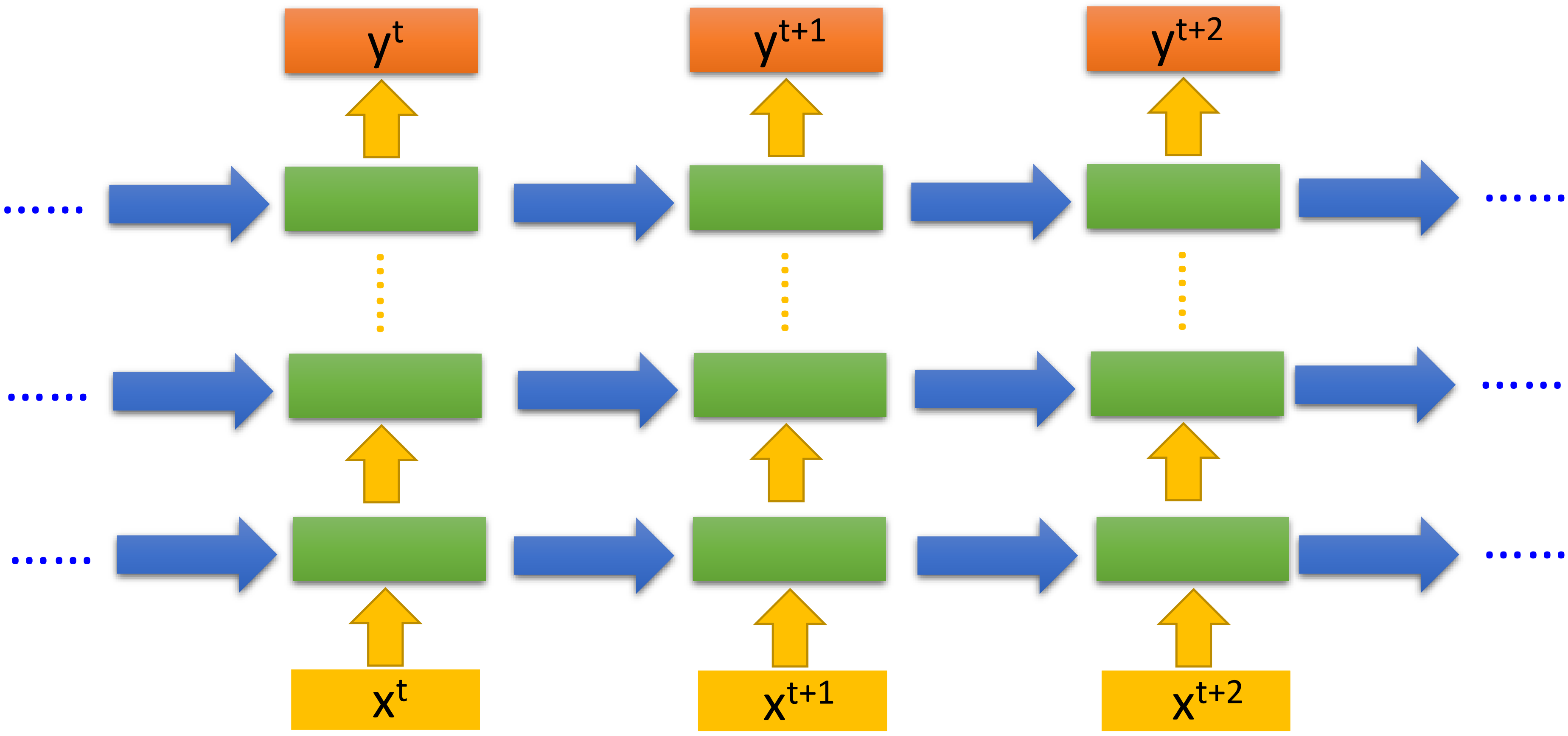
The same network is used again and again.



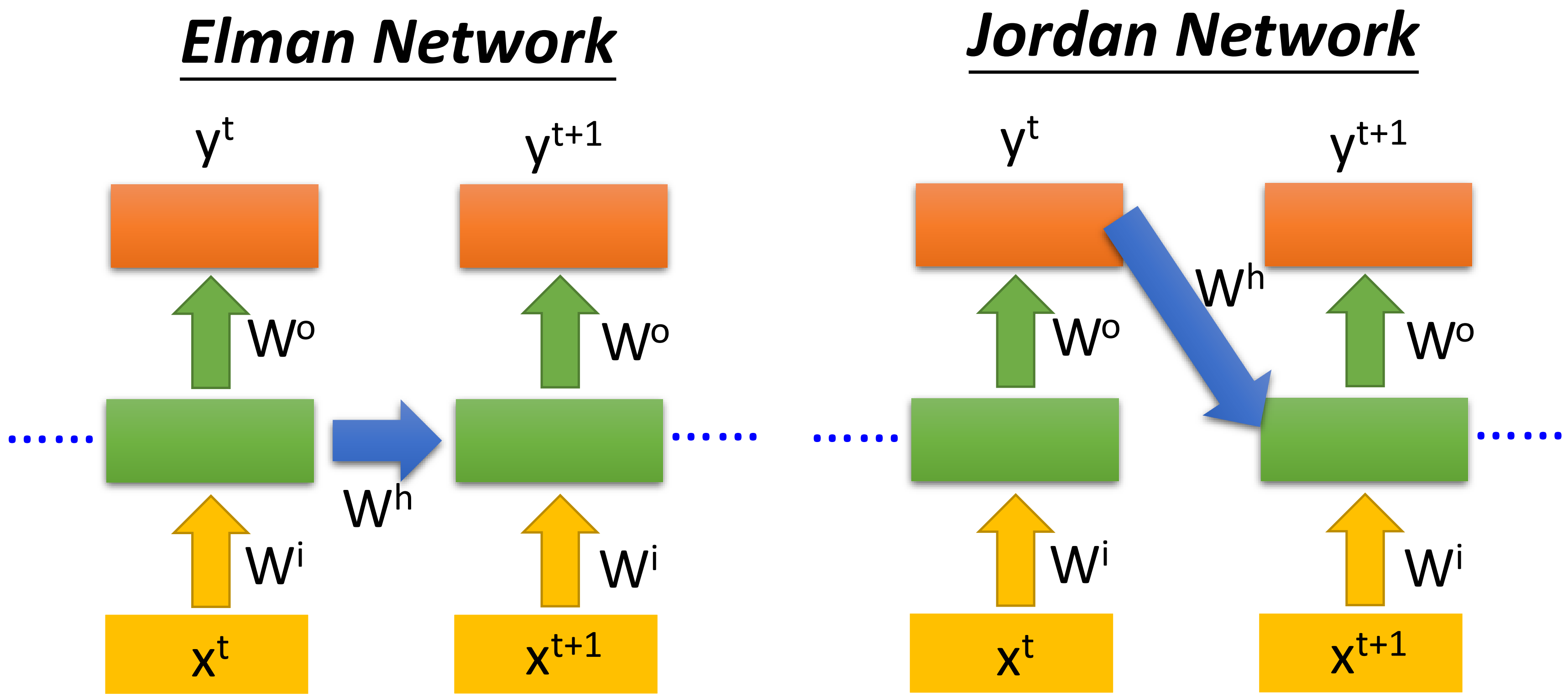
RNN


Different
Prob of "leave" in each slot
Prob of "Taipei" in each slot
Prob of "arrive" in each slot
Prob of "Taipei" in each slot
y1
y2
a1
a2
store
a1
x1
x2
leave
Taipei
arrive
Taipei
The values stored in the memory is different.

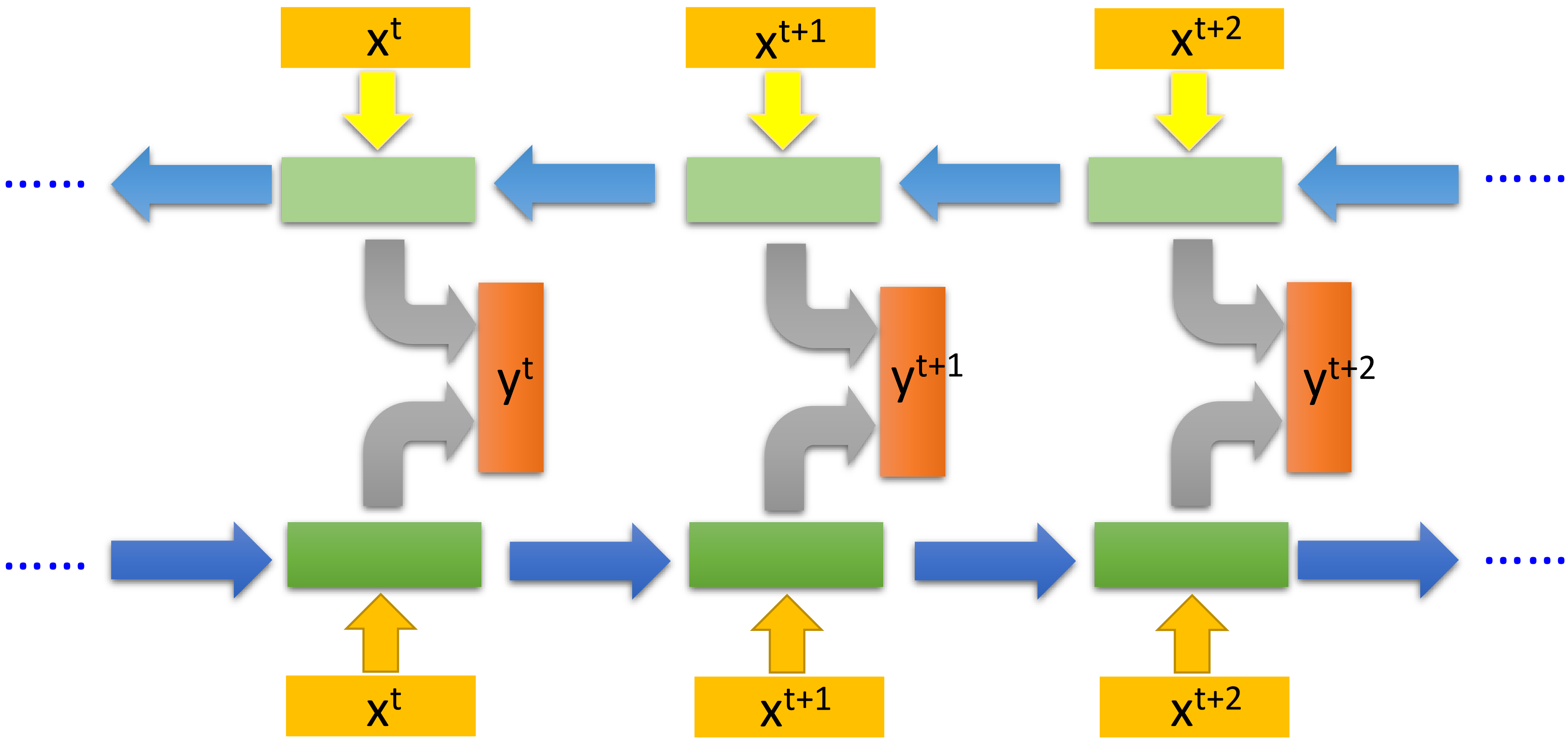
Of course it can be deep …



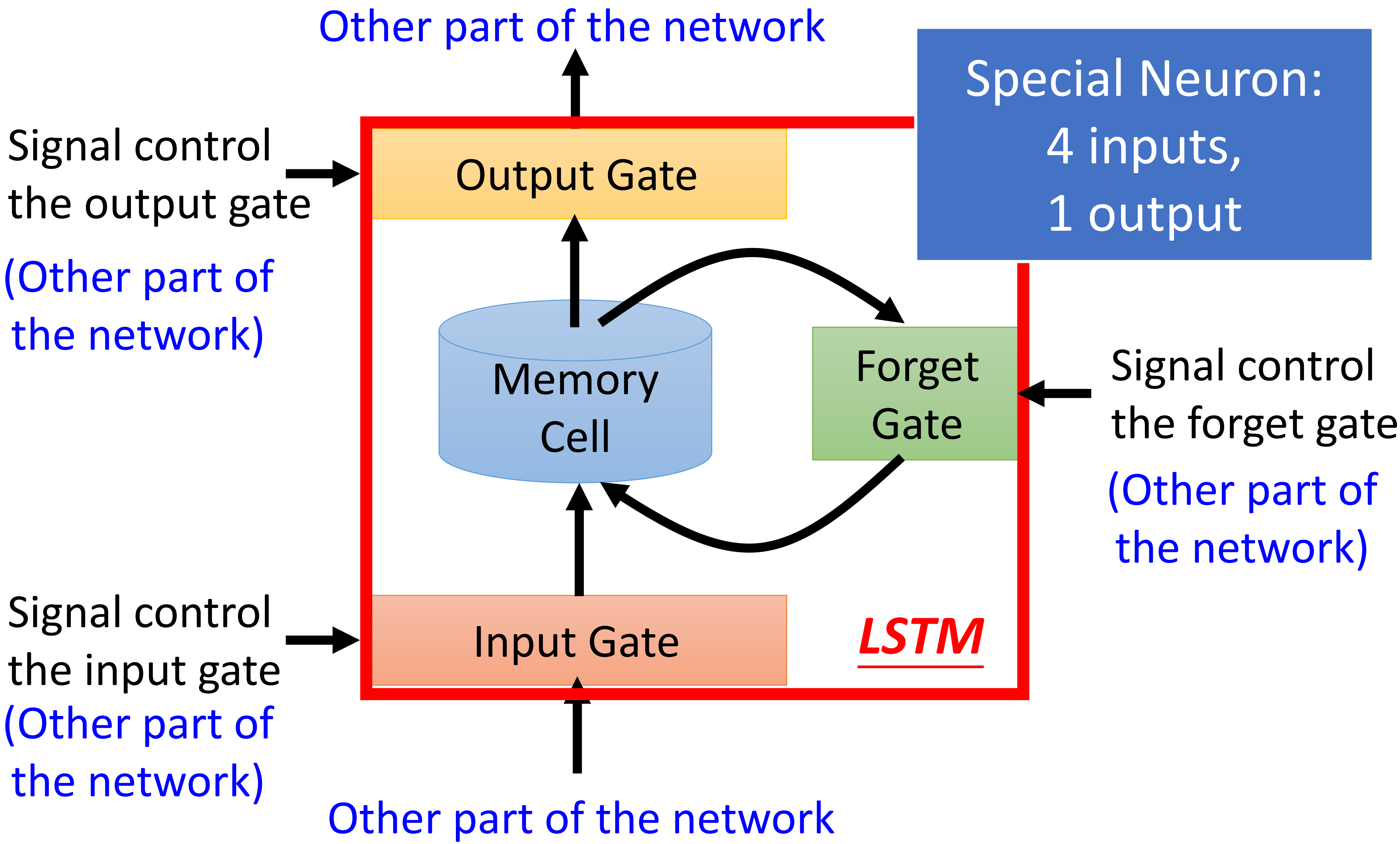
Elman Network & Jordan Network

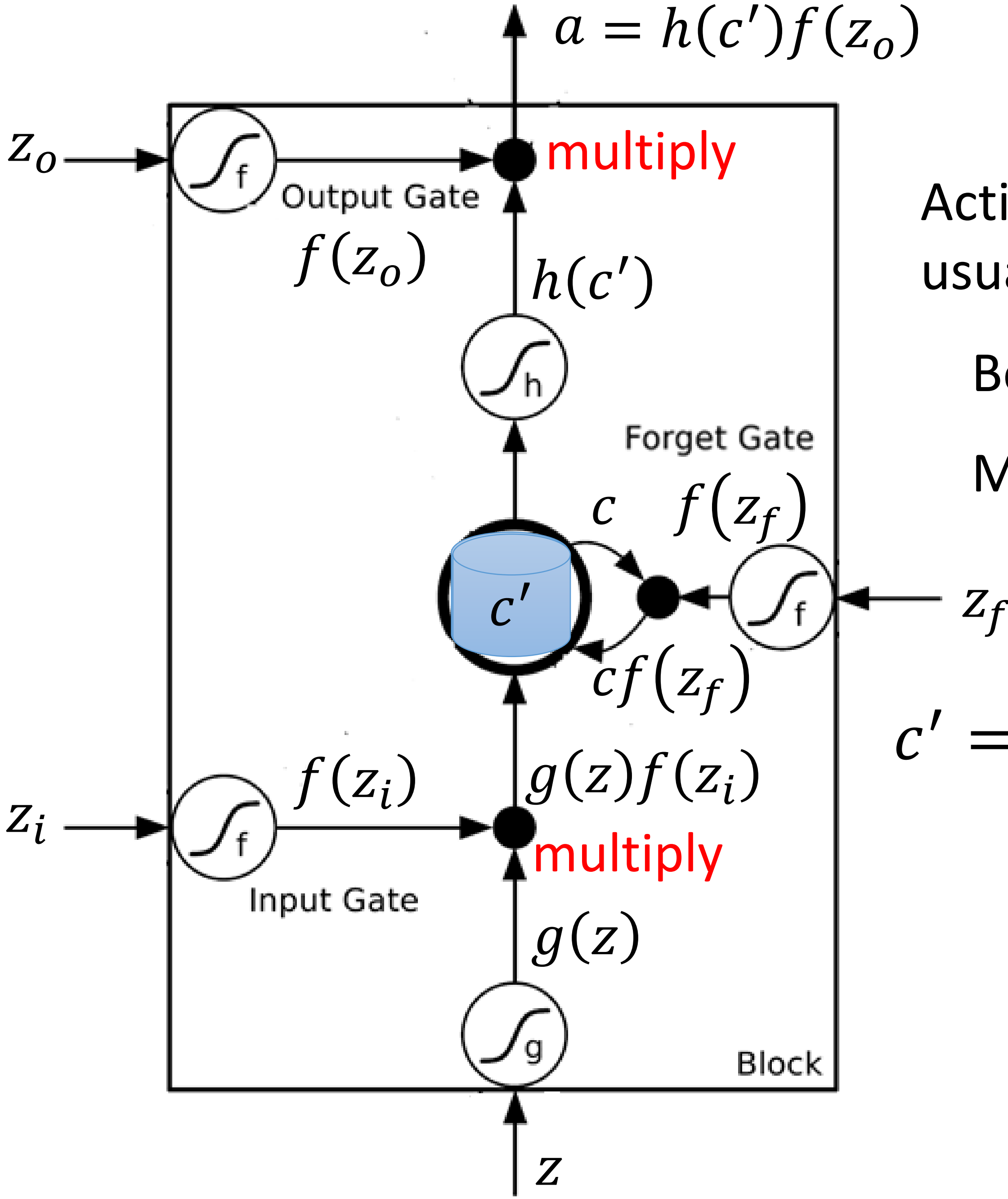


Bidirectional RNN



Long Short-term Memory (LSTM)





Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

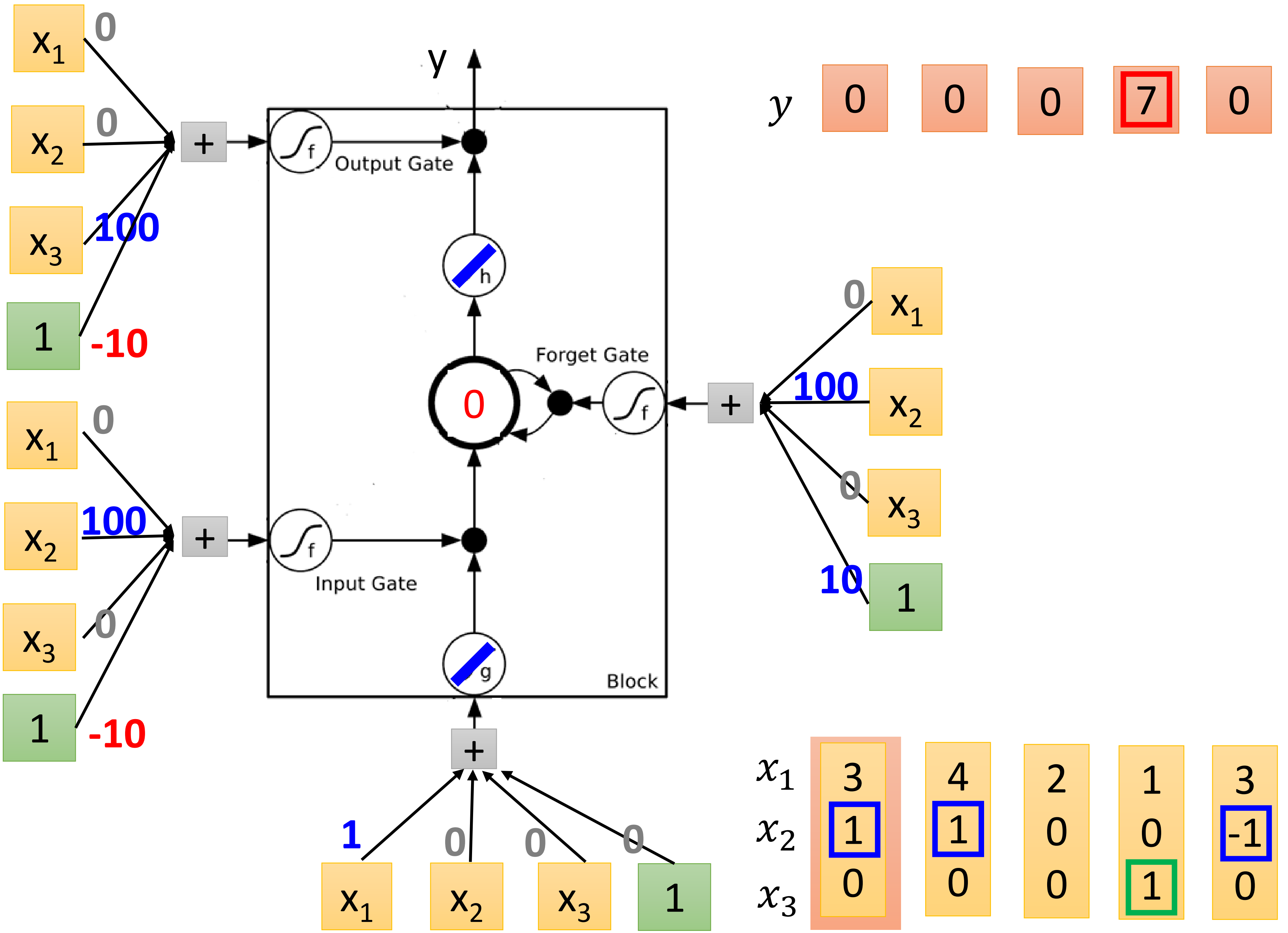
LSTM - Example

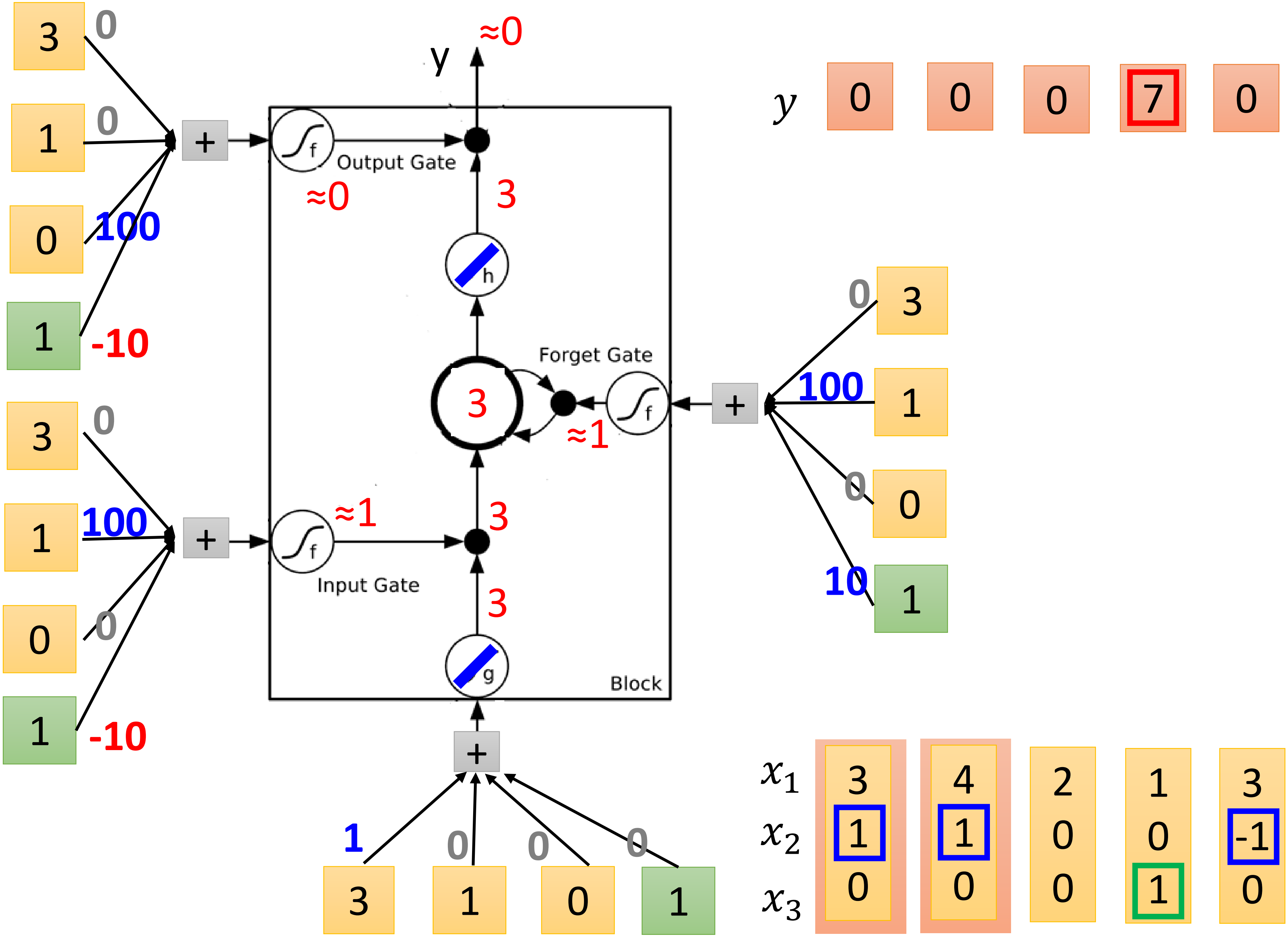
	0	0	3	3	7	7	7	0	6
x_1	1	3	2	4	2	1	3	6	1
x_2	0	$\boxed{1}$	0	$\boxed{1}$	0	0	$\boxed{-1}$	$\boxed{1}$	0
x_3	0	0	0	0	0	$\boxed{1}$	0	0	$\boxed{1}$
y	0	0	0	0	0	$\boxed{7}$	0	0	$\boxed{6}$

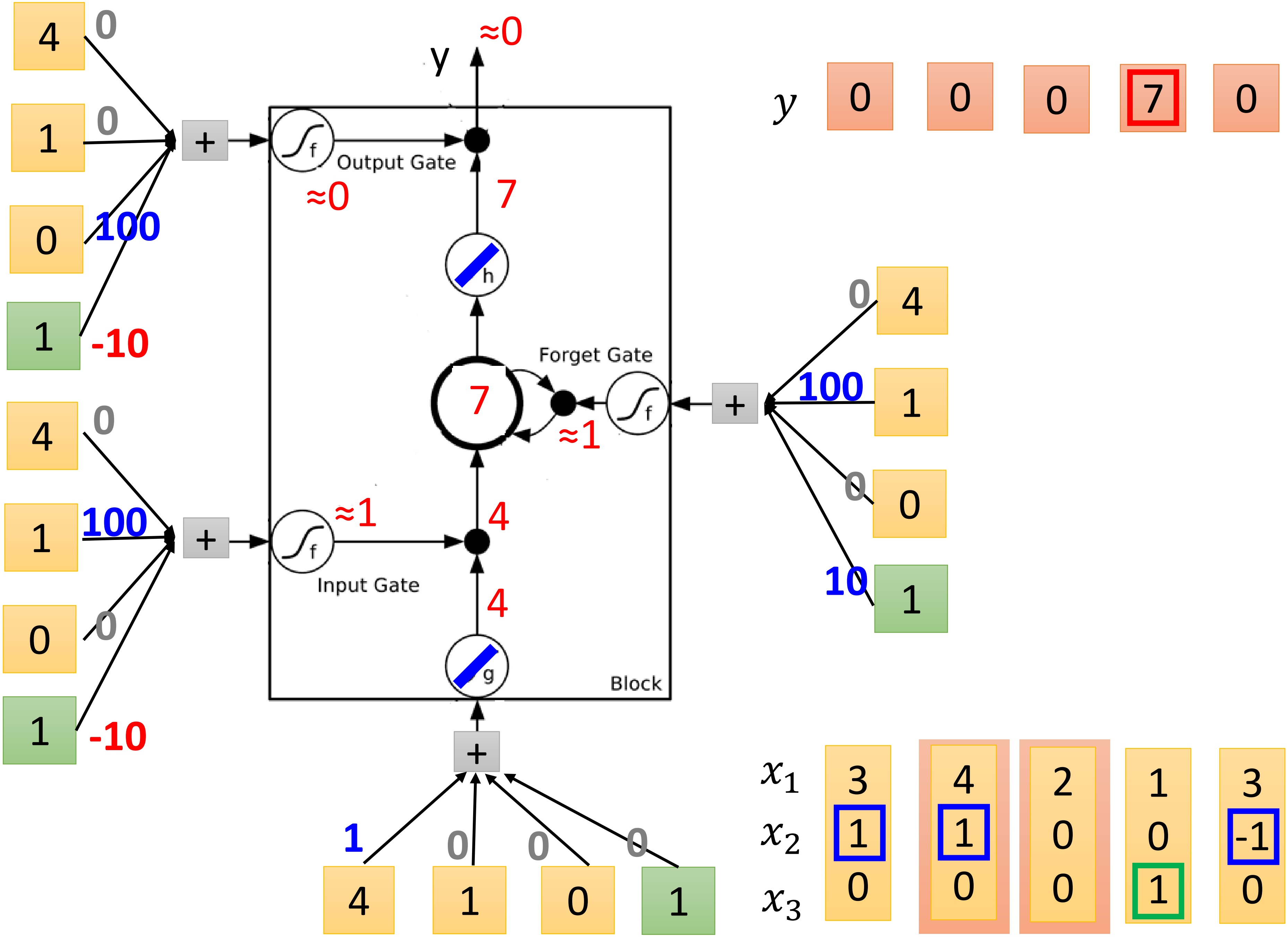
When x_2 = 1, add the numbers of x_1 into the memory

When x_2 = -1, reset the memory

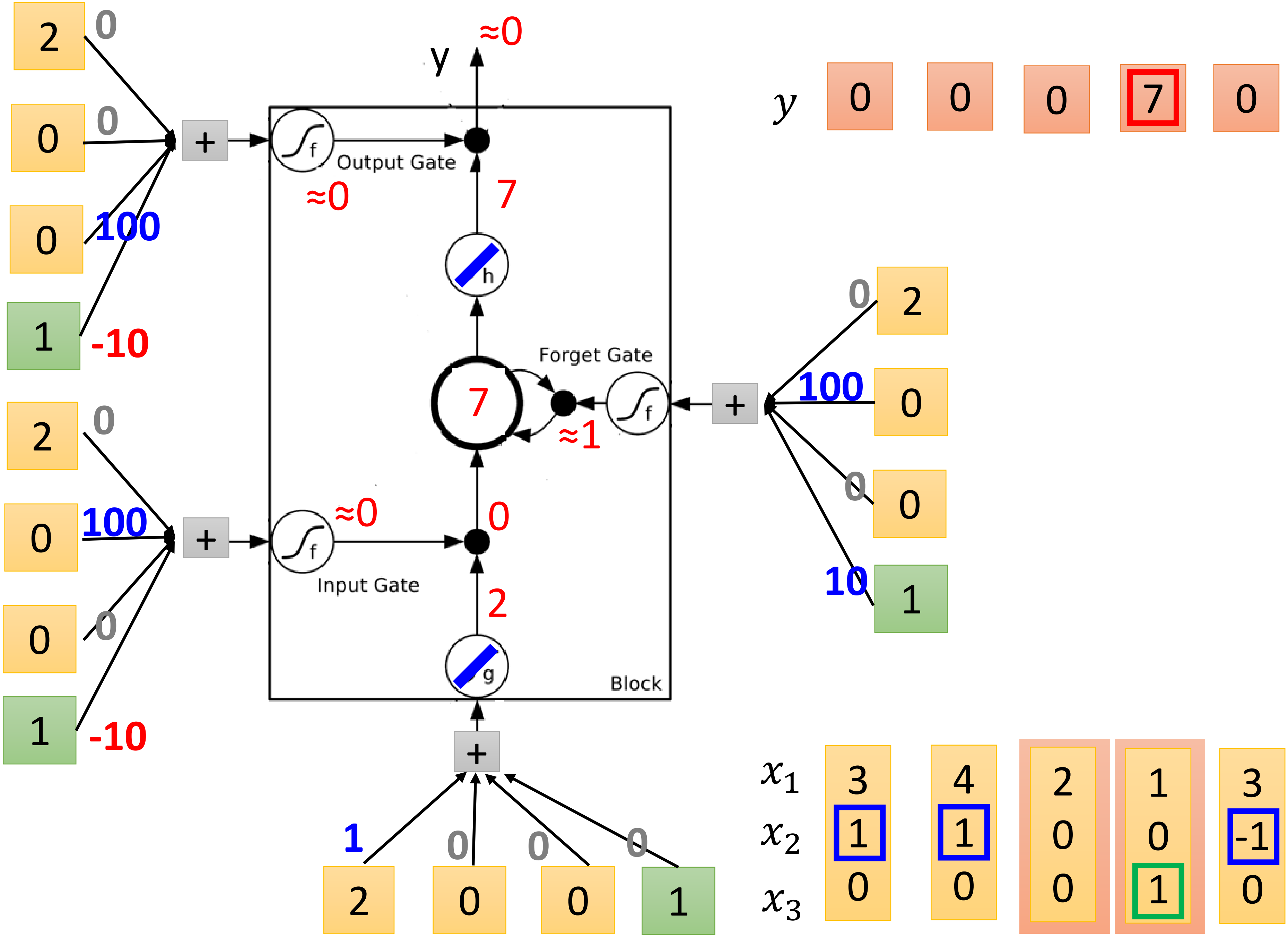
When x_3 = 1, output the number in the memory.

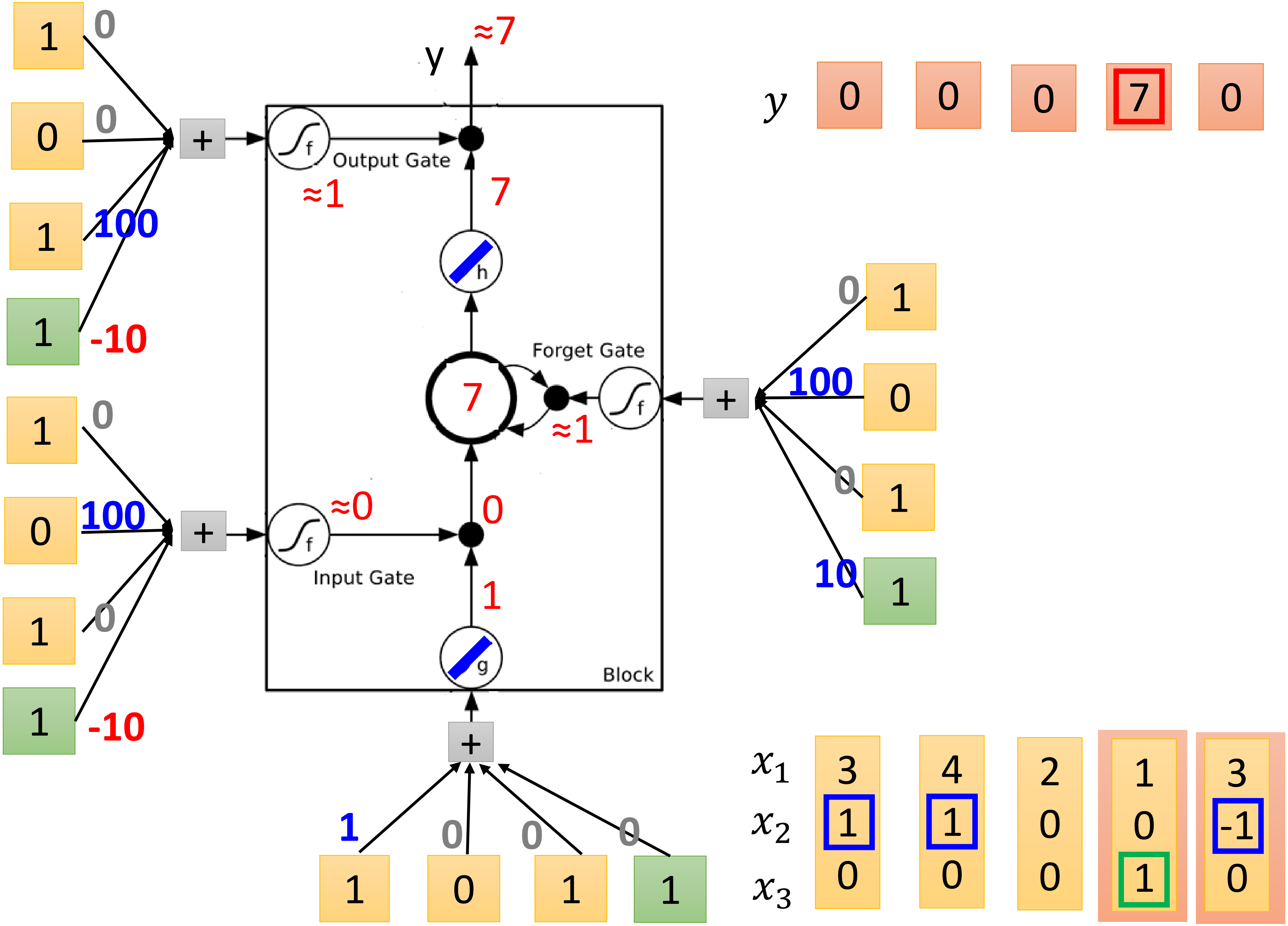

y
Output Gate
Forget Gate
Input Gate
Block
x1 0
x2 0
x3 100
1 -10
x1 0
x2 100
x3 0
1 -10
x1 0
x2 100
x3 0
1 10
x1 1
x2 0
x3 0
1 0
0
y 0 0 0 7 0
x1 3 4 2 1 3
x2 1 1 0 0 -1
x3 0 0 0 1 0


y
≈0
Output Gate
≈0
3
Forget Gate
3
≈1
≈1
3
Input Gate
3
Block
3
0
1
0
0
100
1
-10
3
0
1
100
0
0
1
-10
0
3
100
1
0
0
10
1
1
3
0
1
0
0
0
1
y
0
0
0
7
0
x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0

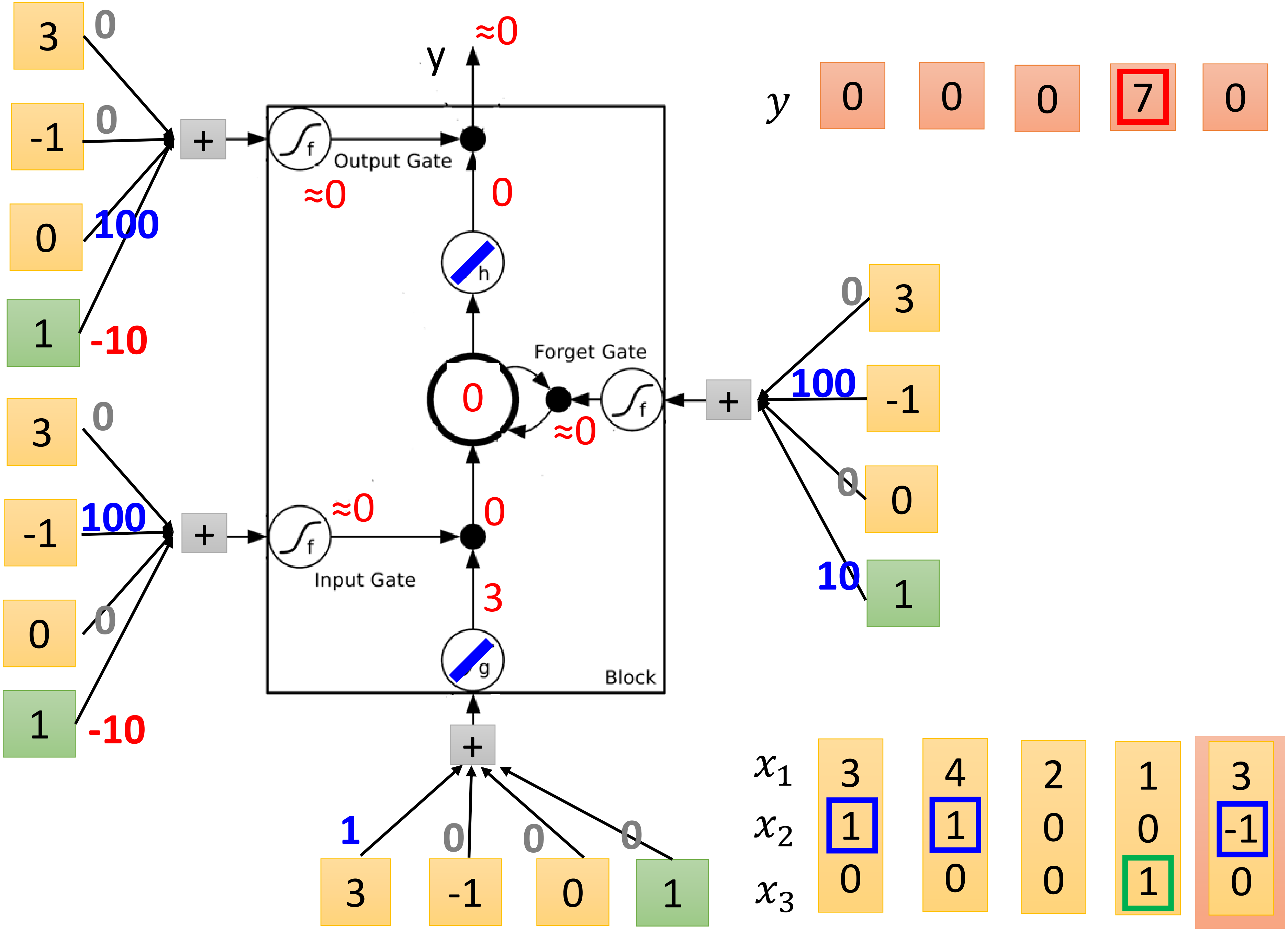


y
≈0
Output Gate
≈0
7
h
Forget Gate
7
≈1
f
≈1
4
Input Gate
4
g
Block
4
0
1
0
0
100
1
-10
4
0
1
100
0
0
1
-10
0
4
100
1
0
0
10
1
1
4
0
1
0
0
0
1
y
0
0
0
7
0
x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0


y
≈0
Output Gate
≈0
7
Forget Gate
≈1
7
Input Gate
≈0
0
2
Block
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0



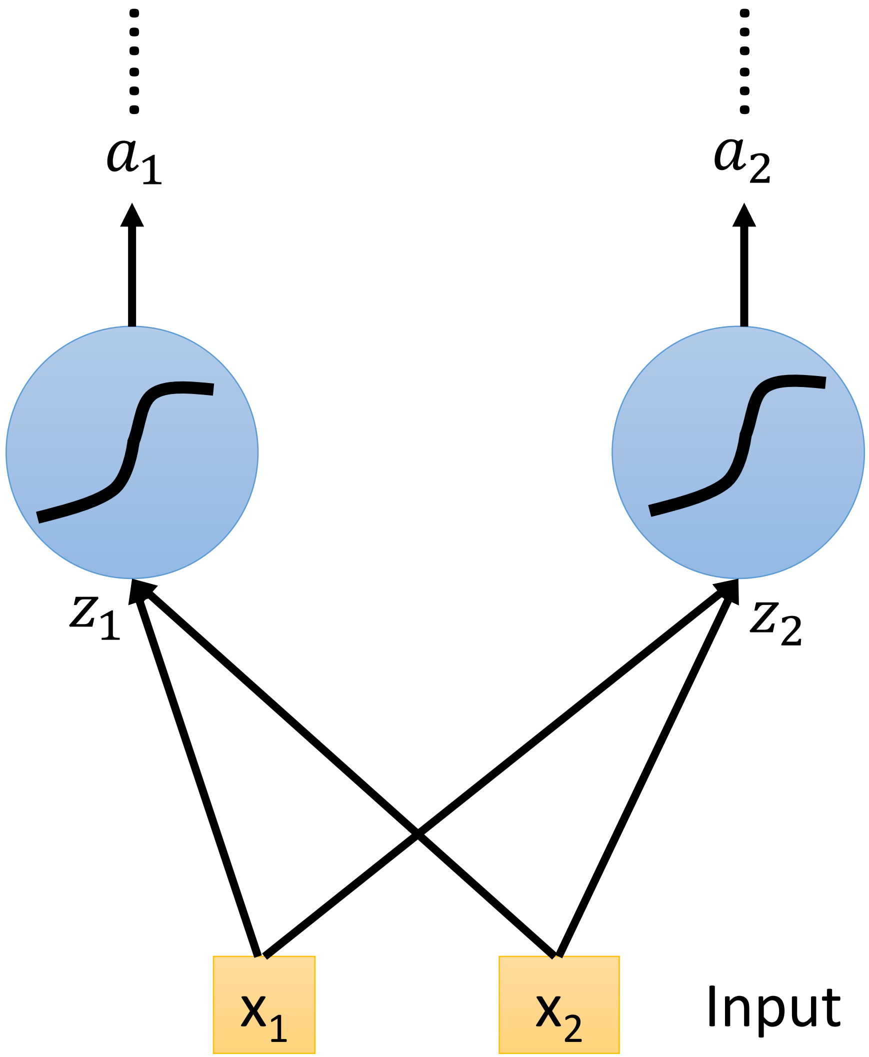
y
≈7
Output Gate
≈1
7
h
Forget Gate
7
≈1
≈0
0
Input Gate
1
g
Block
100
-10
100
-10
100
10
1
0
0
0
+
y
0 0 0 7 0
x_1
x_2
x_3
3 1 0
4 1 0
2 0 0
1 0 1
3 -1 0

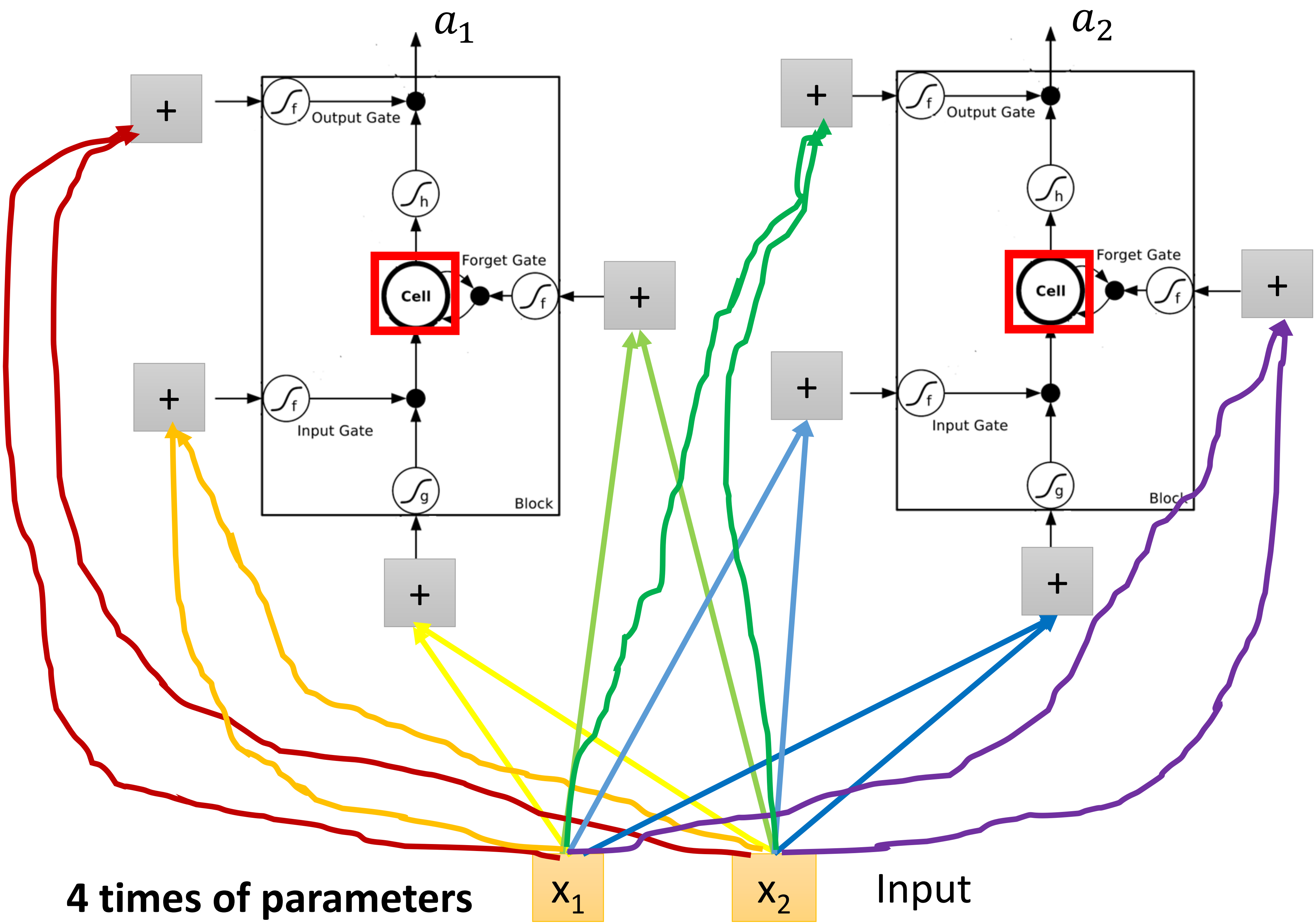


y
≈0
Output Gate
Forget Gate
Input Gate
Block
100
-10
10
y 0 0 0 7 0
x1 3 4 2 1 3
x2 1 1 0 0 -1
x3 0 0 0 1 0

Original Network:

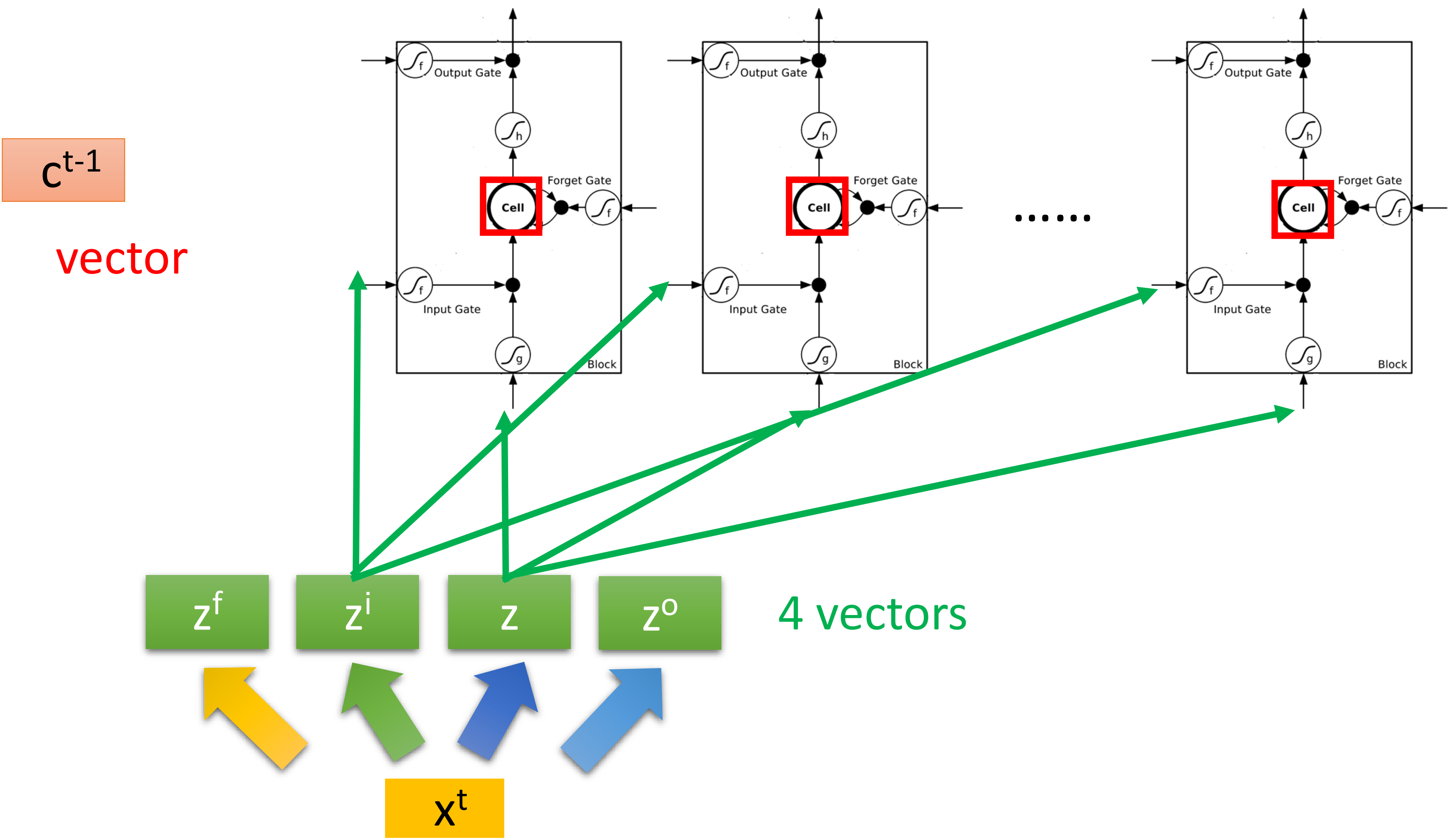
➢ Simply replace the neurons with LSTM





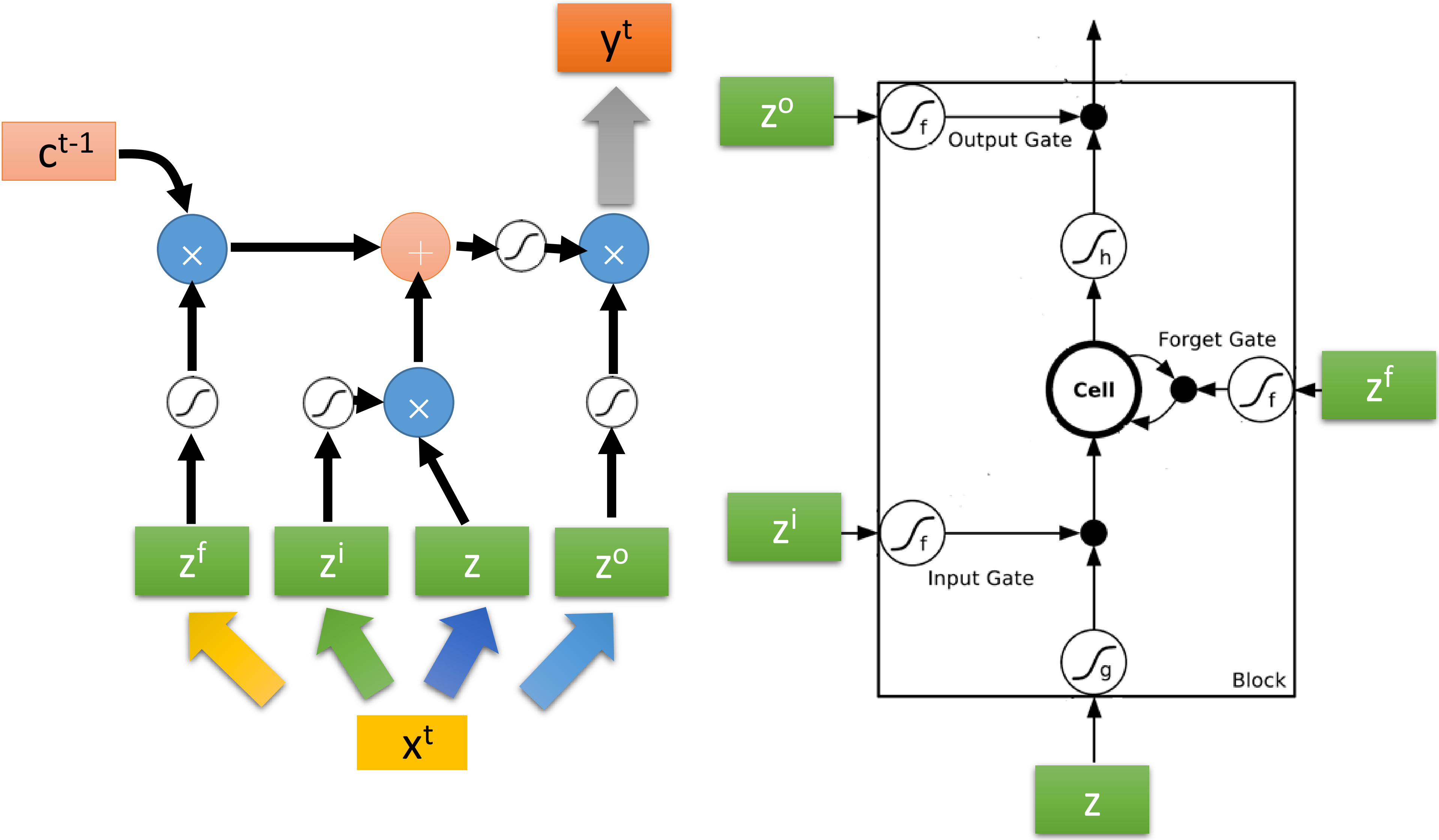
4 times of parameters

LSTM



c^{t-1}
vector
Output Gate
Forget Gate
Cell
Input Gate
Block
......
z^f
z^i
z
z^o
4 vectors
x^t

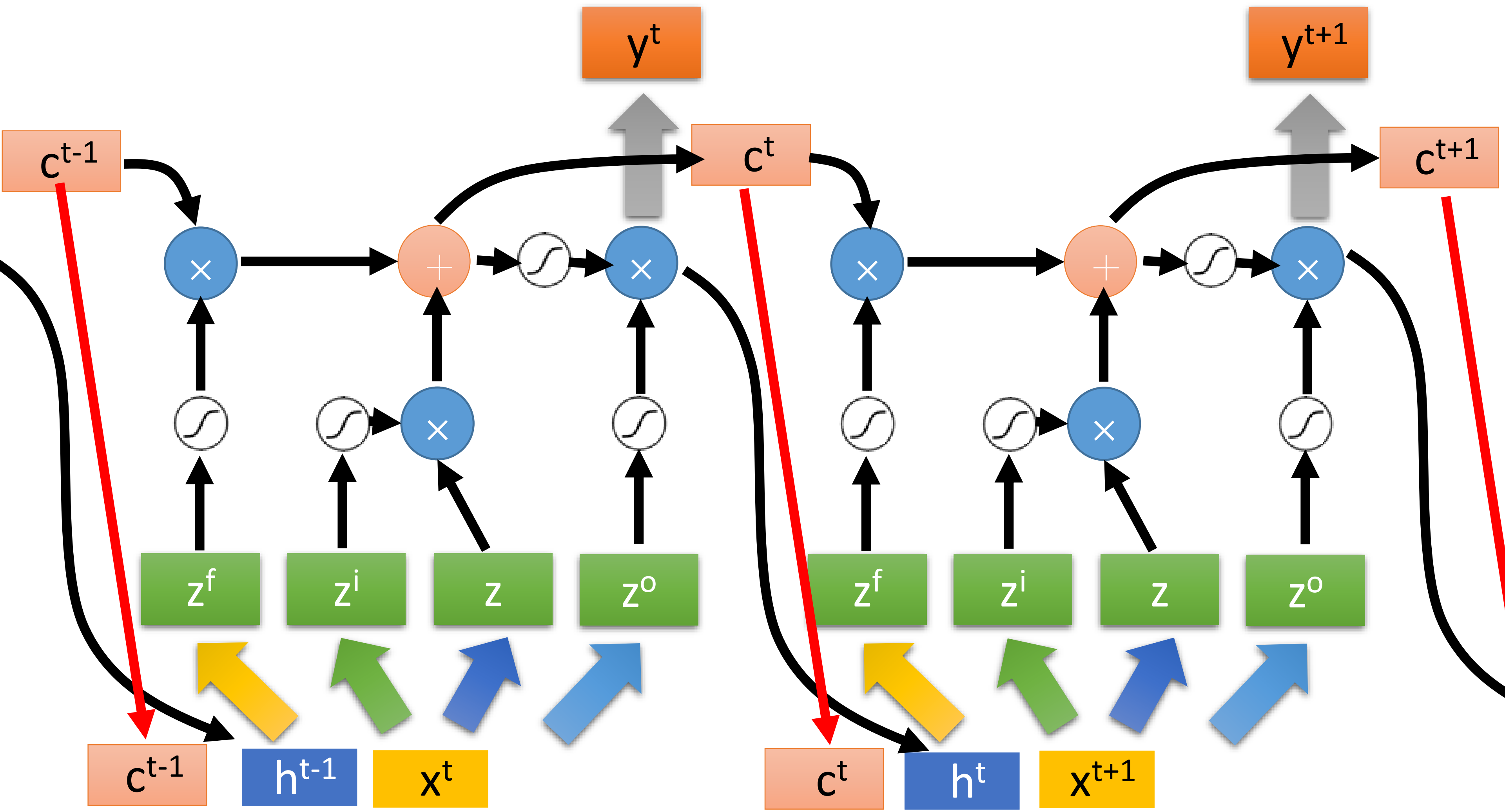
LSTM



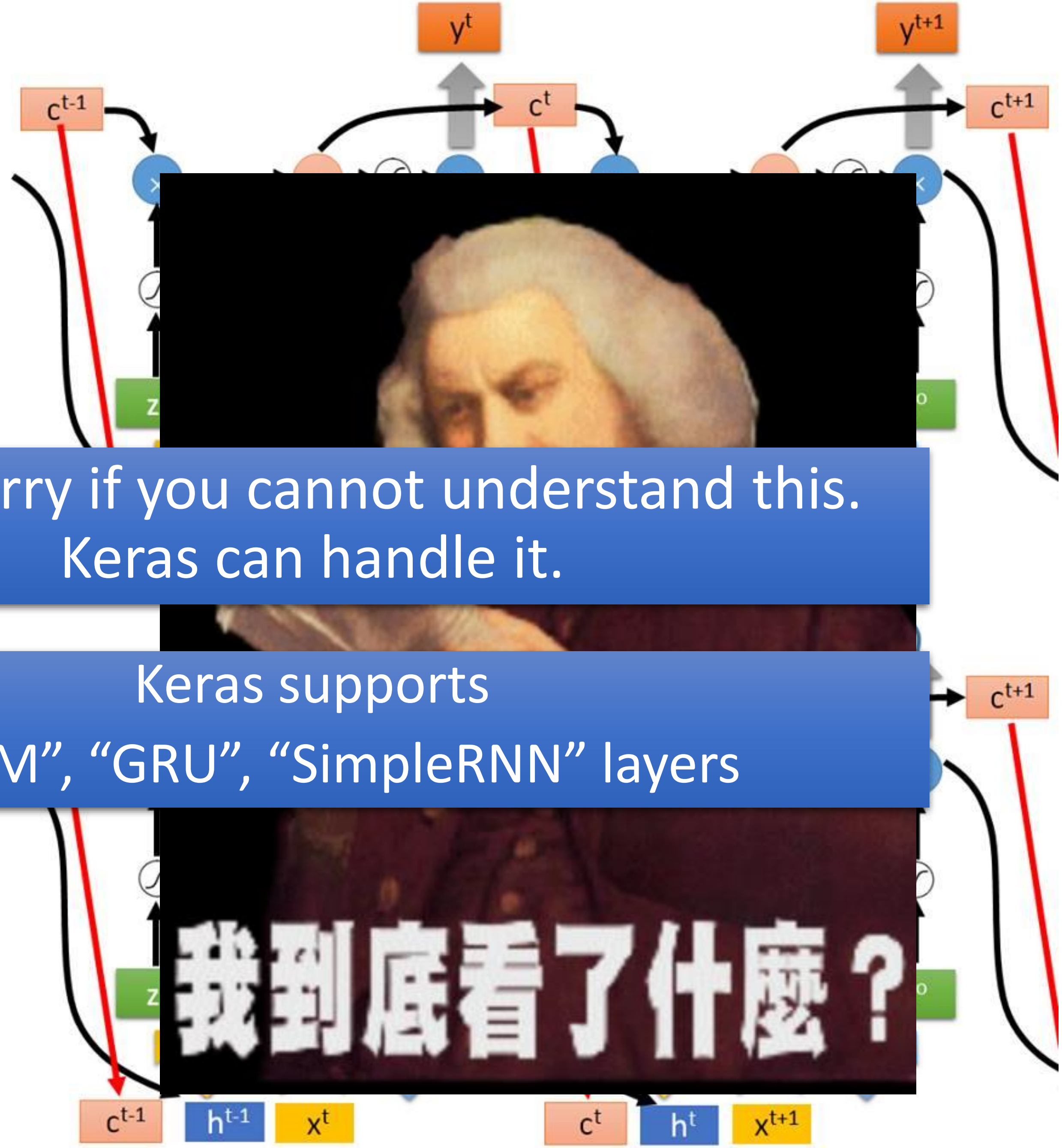
y^t
c^{t-1}
z^f
z^i
z
z^o
x^t
z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

LSTM

Extension: “peephole”



Multiple-layer LSTM



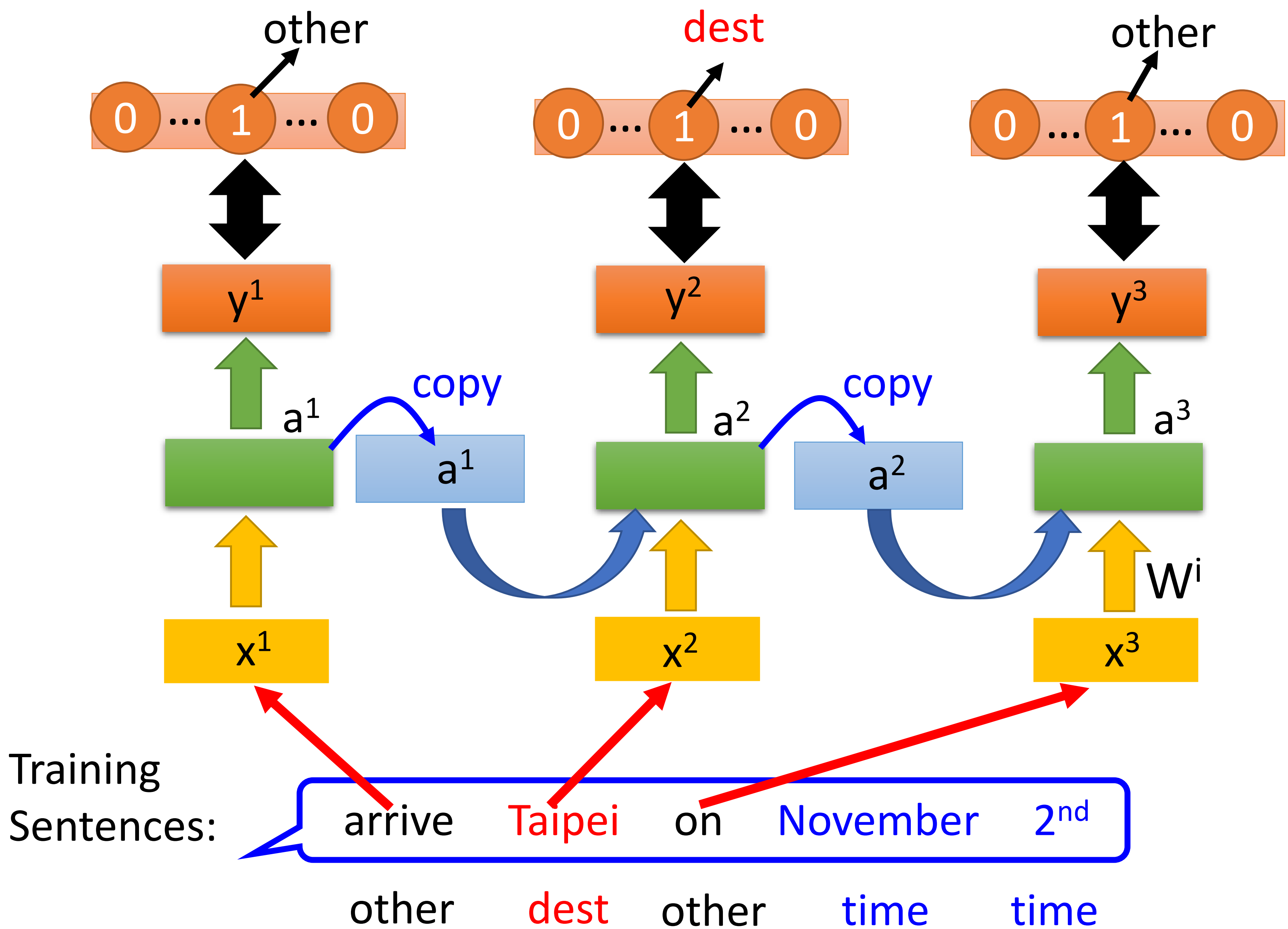
Don’t worry if you cannot understand this.
Keras can handle it.

Keras supports
“LSTM”, “GRU”, “SimpleRNN” layers

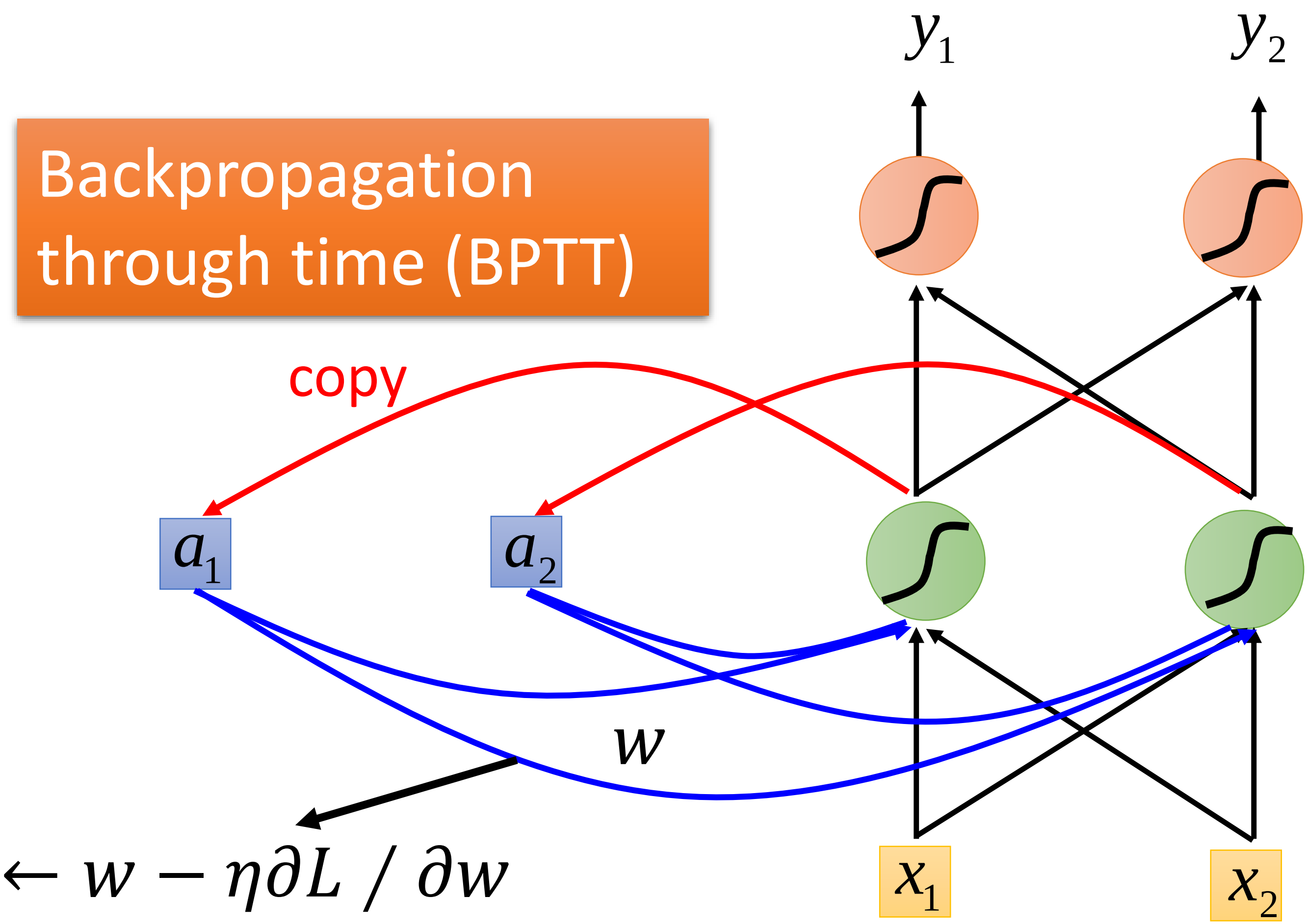
This is quite standard now.

https://img.komicolle.org/2015-09-20/src/1442696762713l.gif

Learning Target



Learning

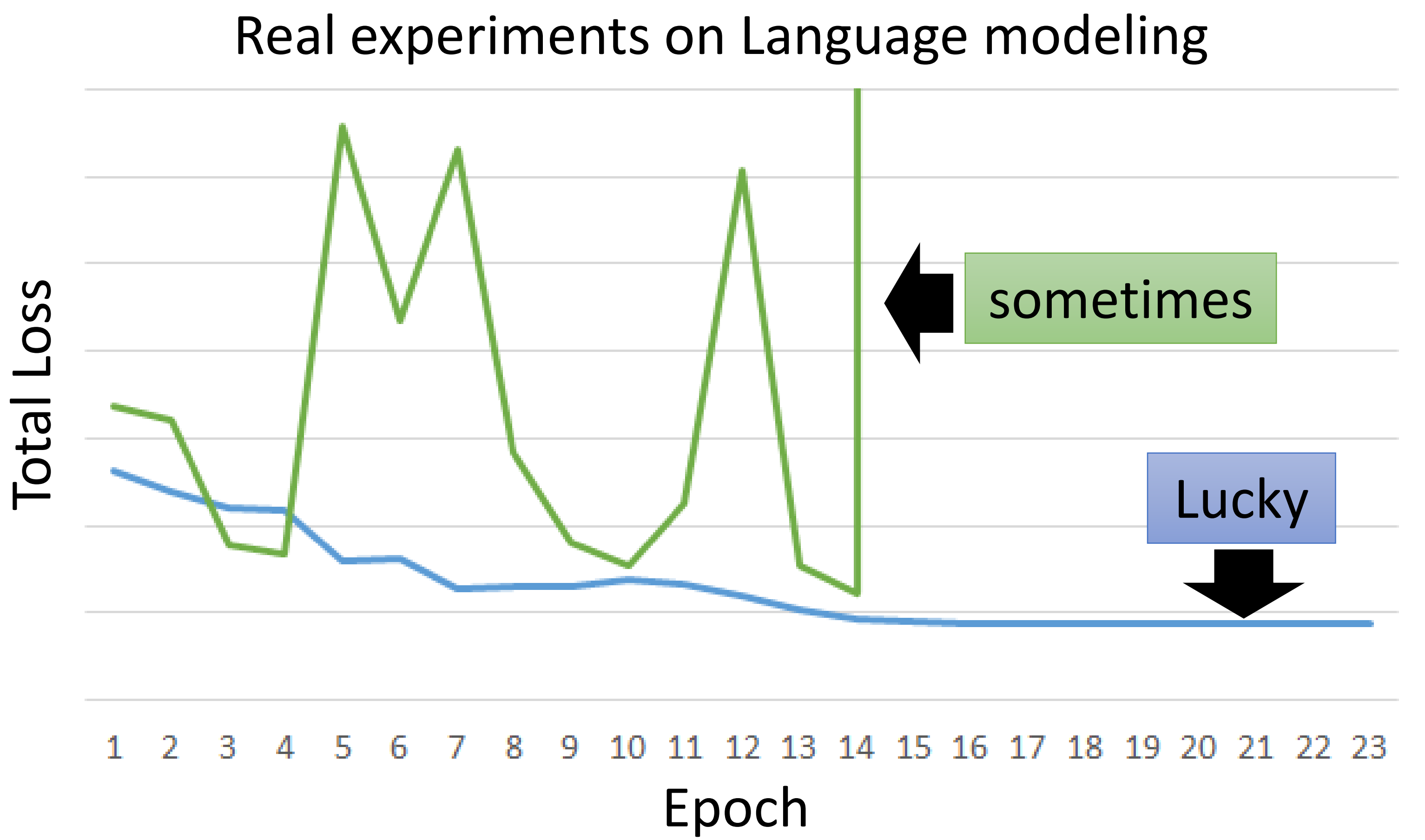


$$w \leftarrow w - \eta \partial L / \partial w$$

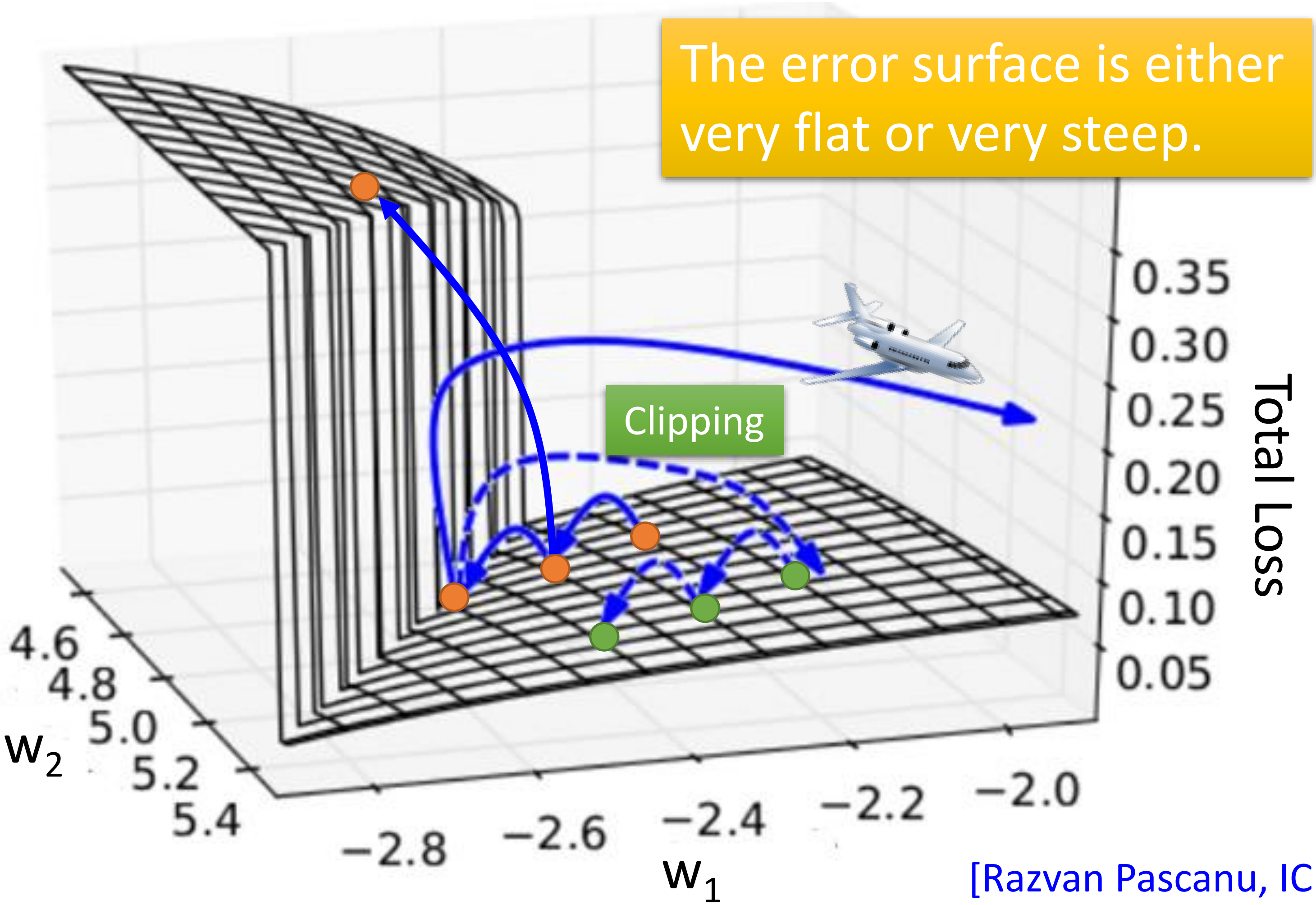
感謝 曾柏翔 同學
提供實驗結果

Unfortunately ……

- RNN-based network is not always easy to learn



The error surface is rough.



[Razvan Pascanu, ICML'13]

Why?

$w = 1 \Rightarrow y^{1000} = 1$

$w = 1.01 \Rightarrow y^{1000} \approx 20000$

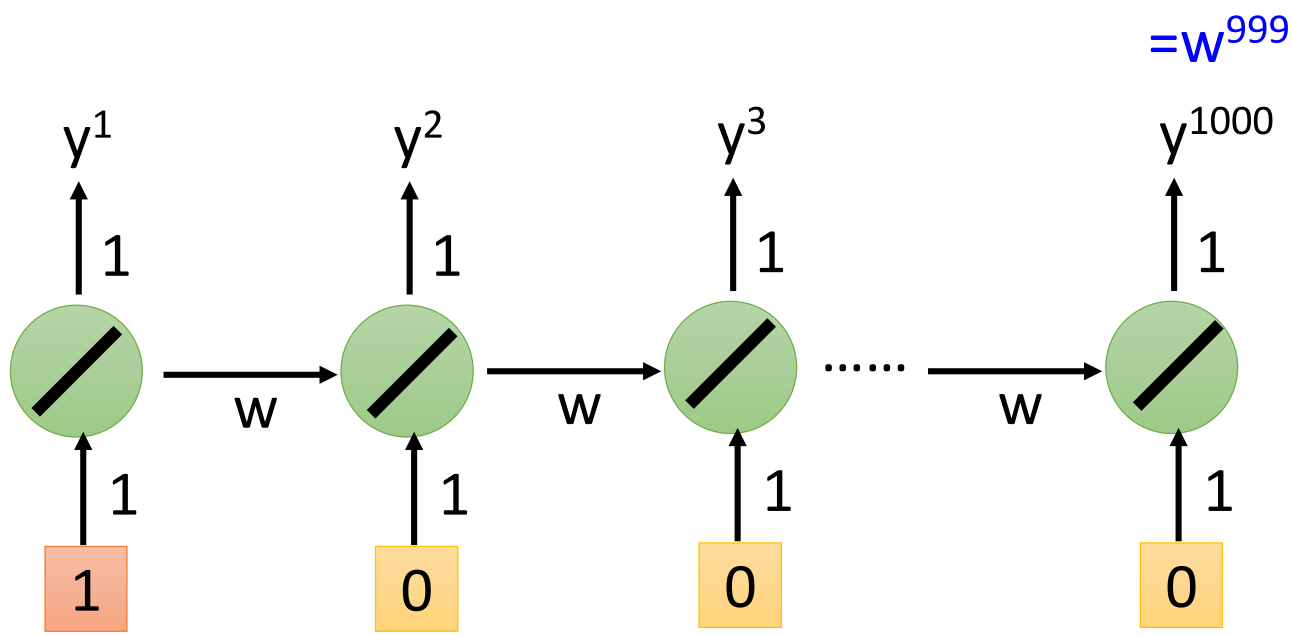
Large $\partial L / \partial w$ → Small Learning rate?

$w = 0.99 \Rightarrow y^{1000} \approx 0$

$w = 0.01 \Rightarrow y^{1000} \approx 0$

small $\partial L / \partial w$ → Large Learning rate?

Toy Example



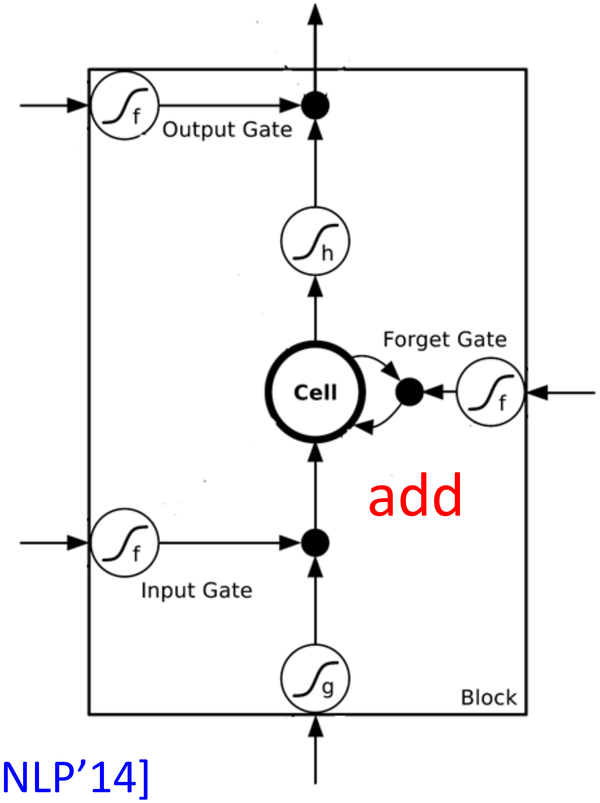
Helpful Techniques

- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)

➢ Memory and input are ***added***

➢ The influence never disappears unless forget gate is closed

No Gradient vanishing
(If forget gate is opened.)



Gated Recurrent Unit (GRU): simpler than LSTM

[Cho, EMNLP'14]