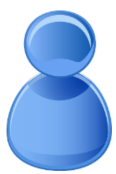
人工智能技术及应用

Artificial Intelligence and Application

Recurrent Neural Network (RNN)

Example Application

- Slot Filling



I would like to arrive Shanghai on June 2nd.

ticket booking system

Slot

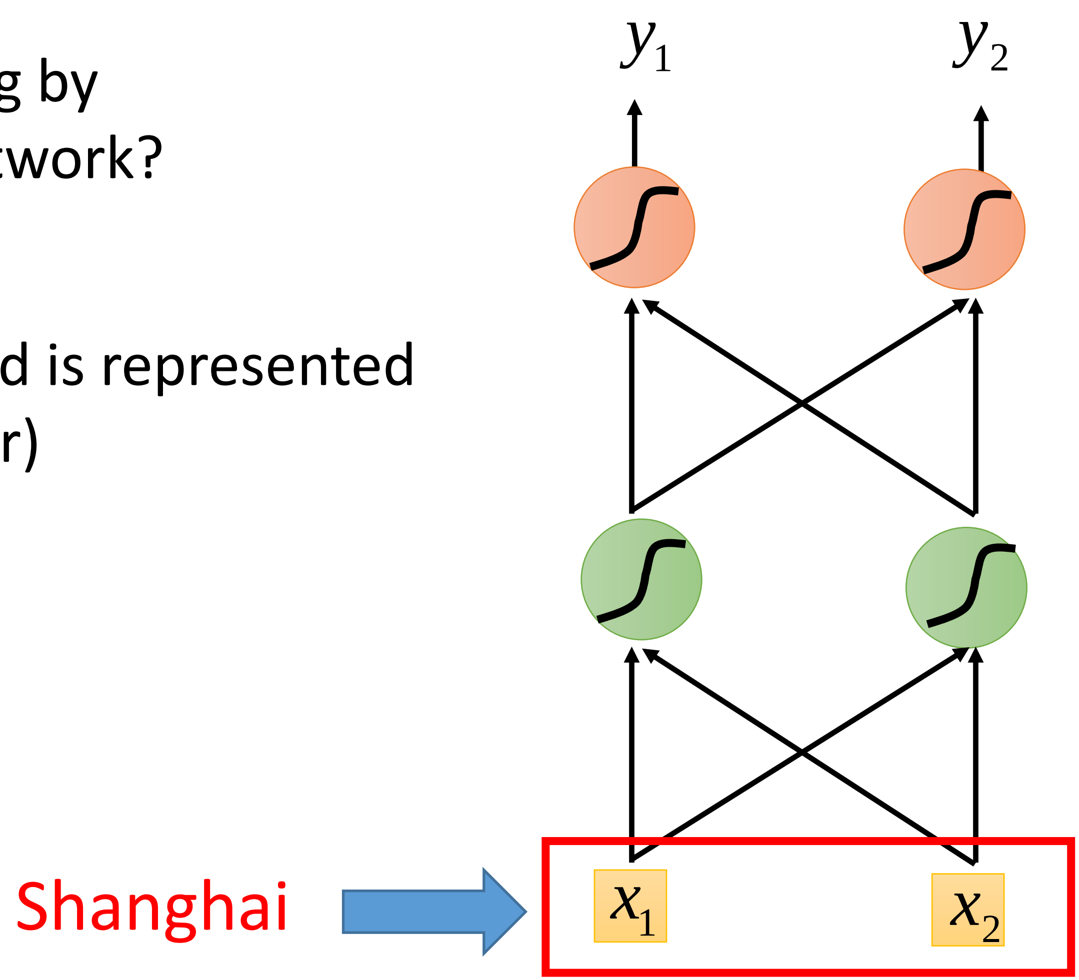
Destination:	Shanghai
time of arrival:	June 2nd

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]

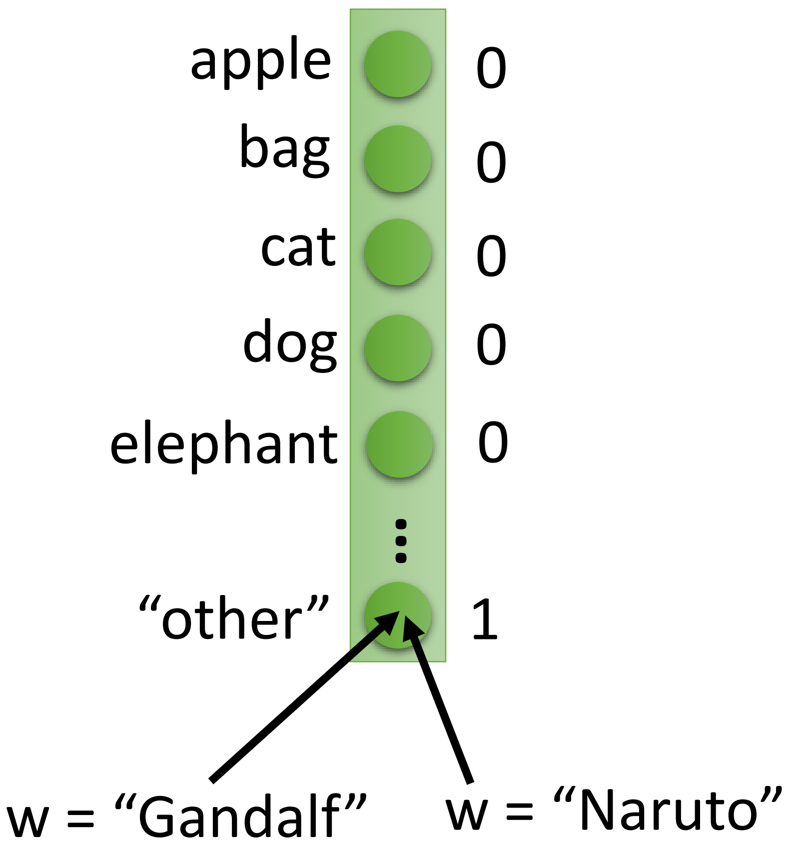
cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

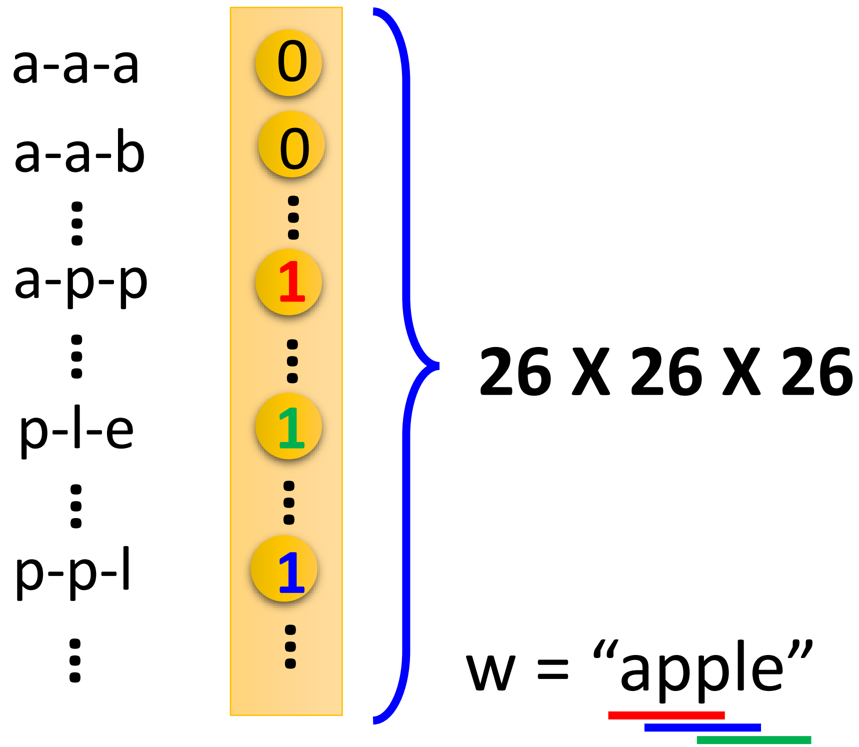
elephant = [0 0 0 0 1]

Beyond 1-of-N encoding

Dimension for "Other"



Word hashing



Example Application

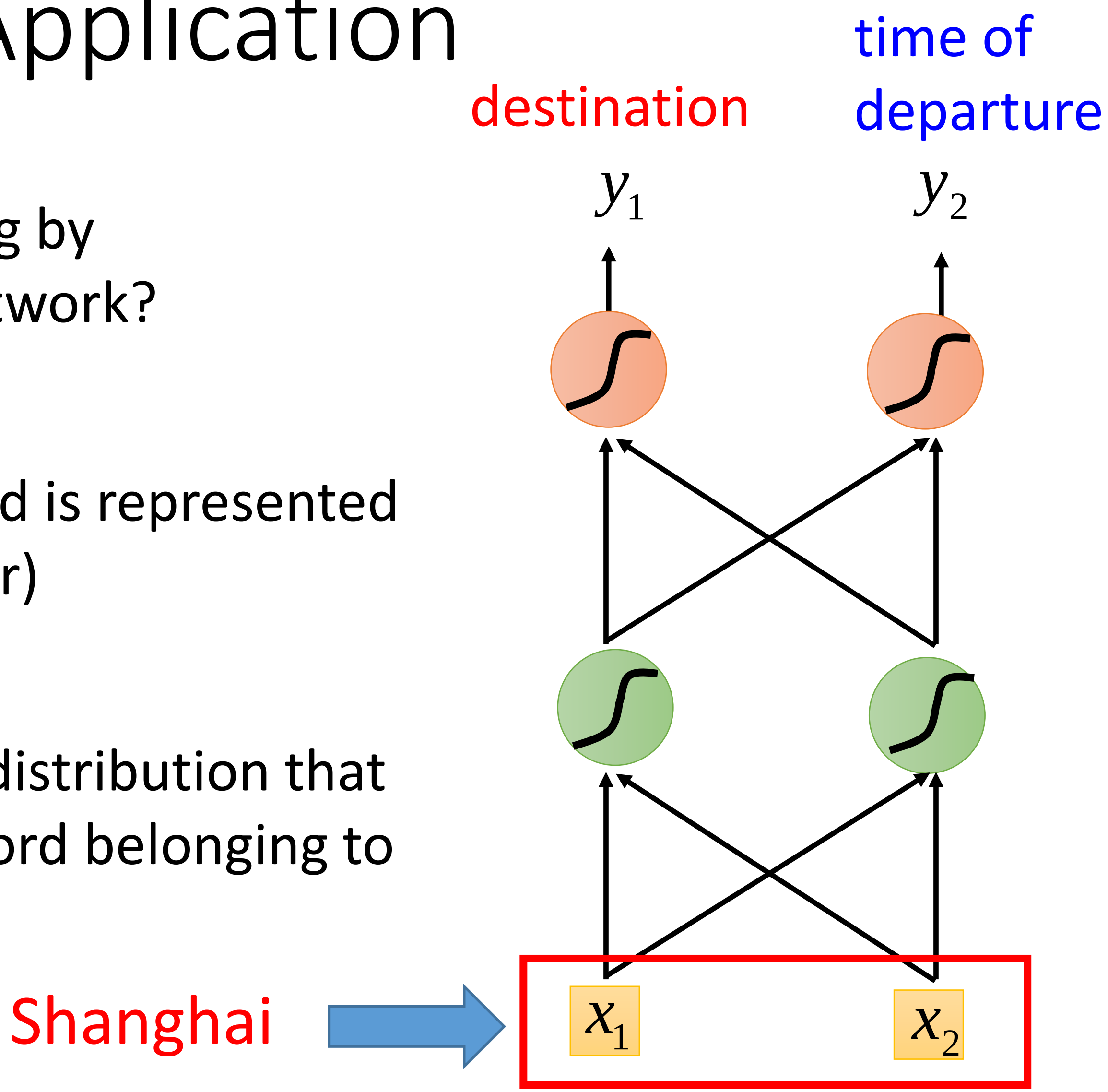
Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)

Output:

Probability distribution that the input word belonging to the slots



Example Application

arrive Shanghai on June 2^{nd}

other destination other time time

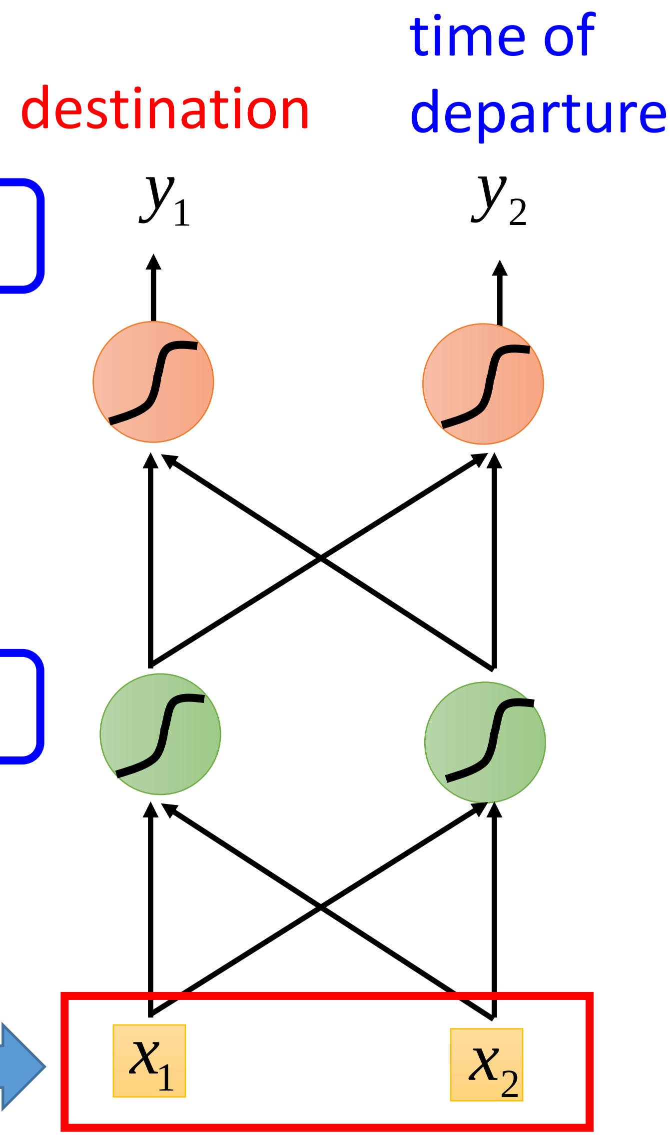
Problem?

leave Shanghai on June 2^{nd}

place of departure

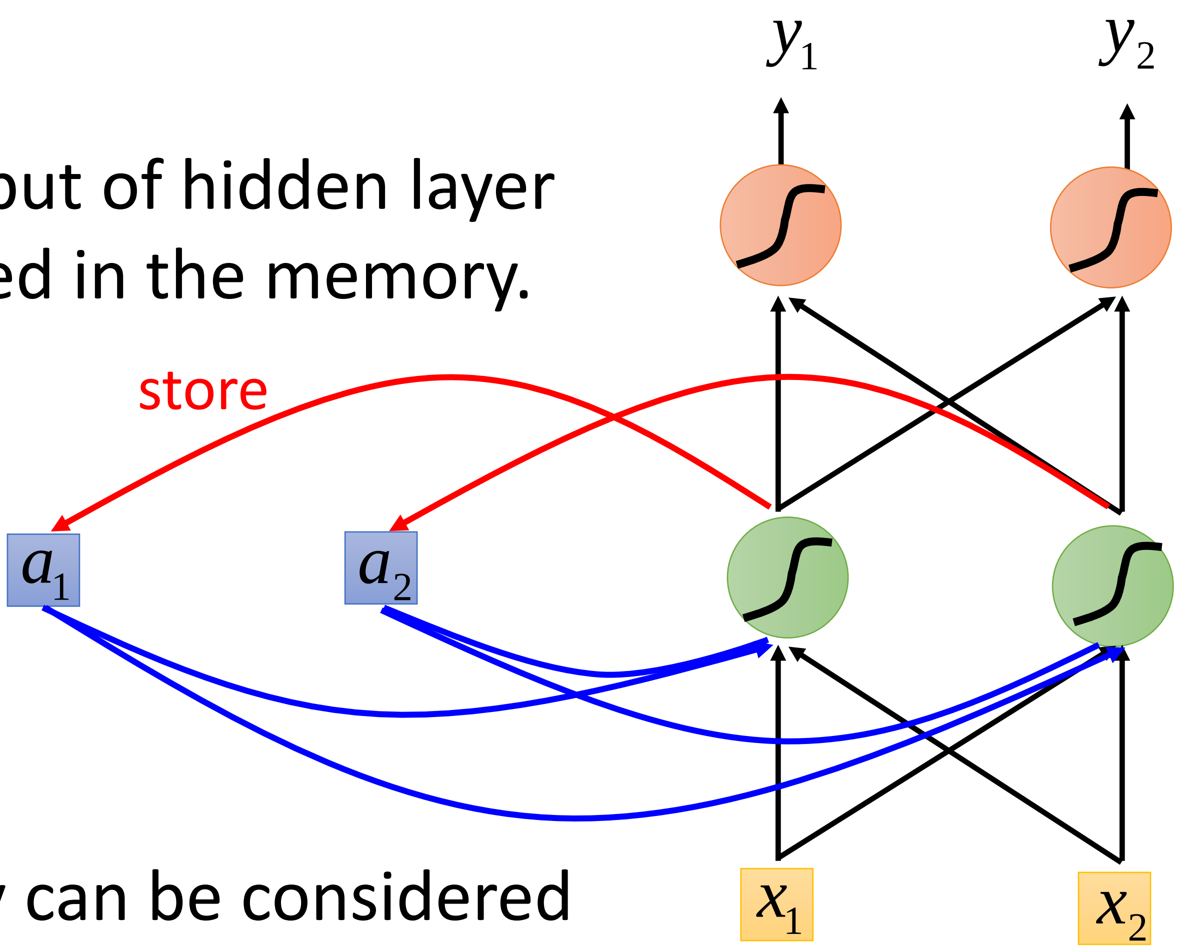
Neural network needs memory!

Shanghai



Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.

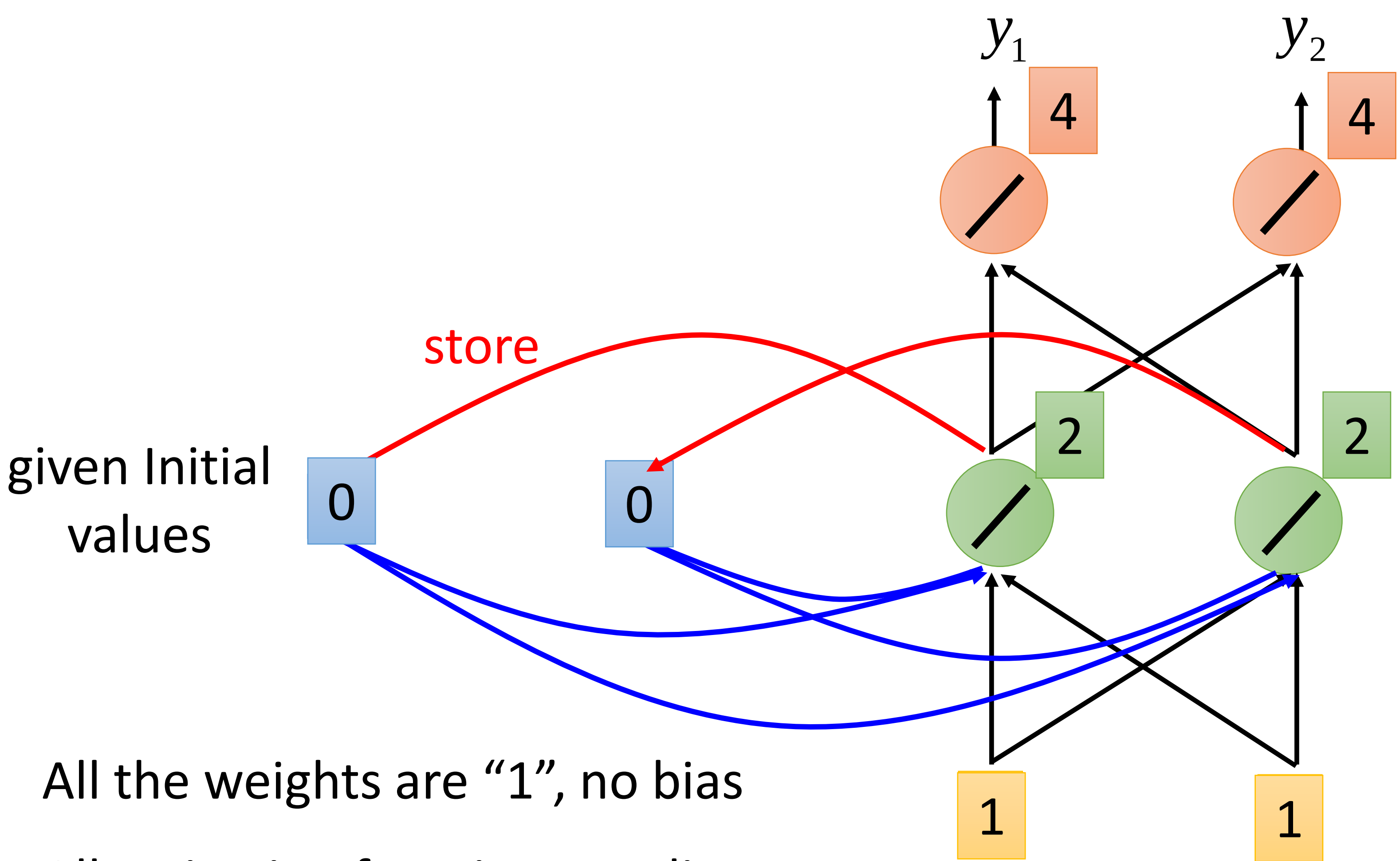


Memory can be considered as another input.

Example

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

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



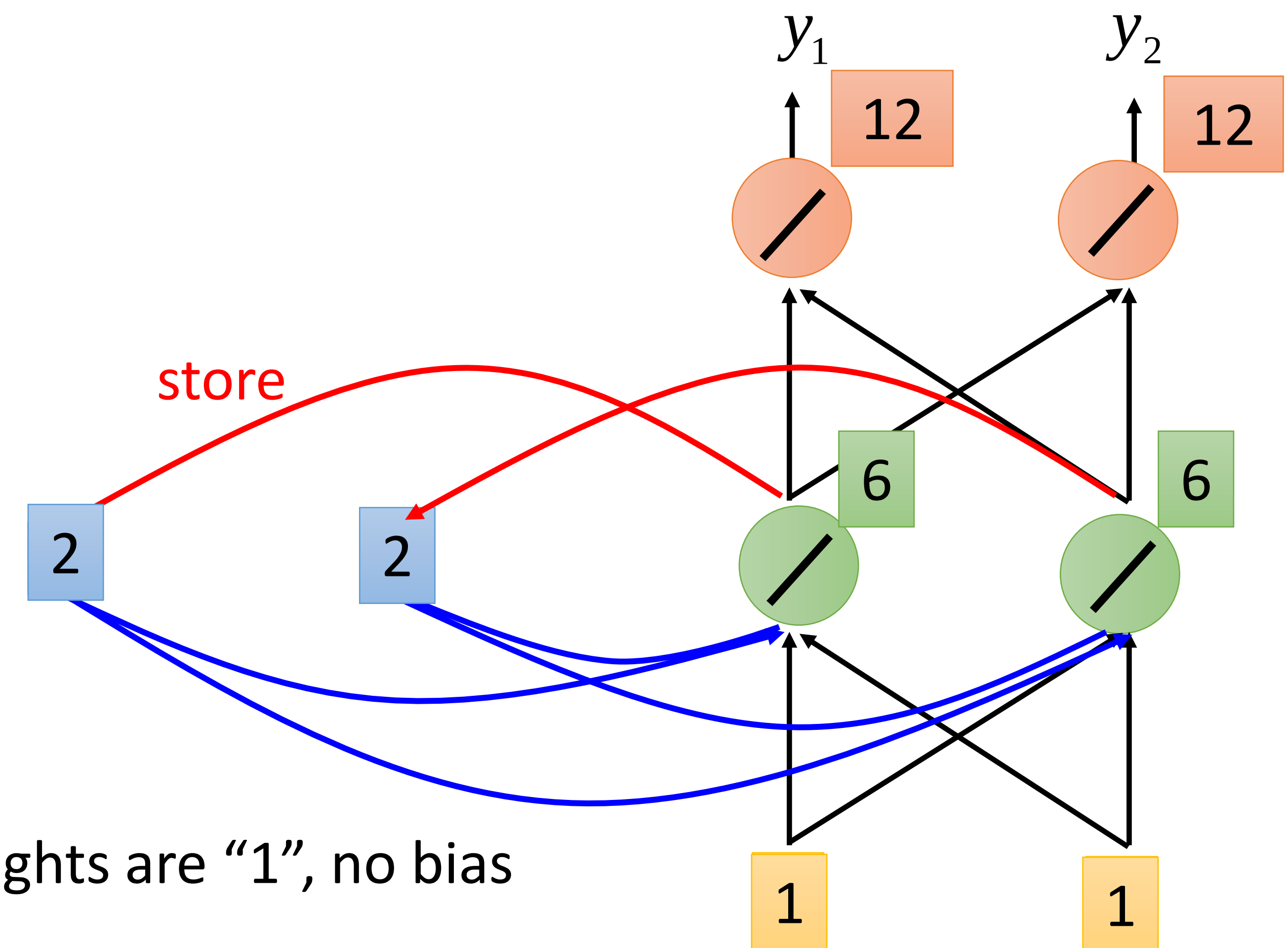
All the weights are "1", no bias

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

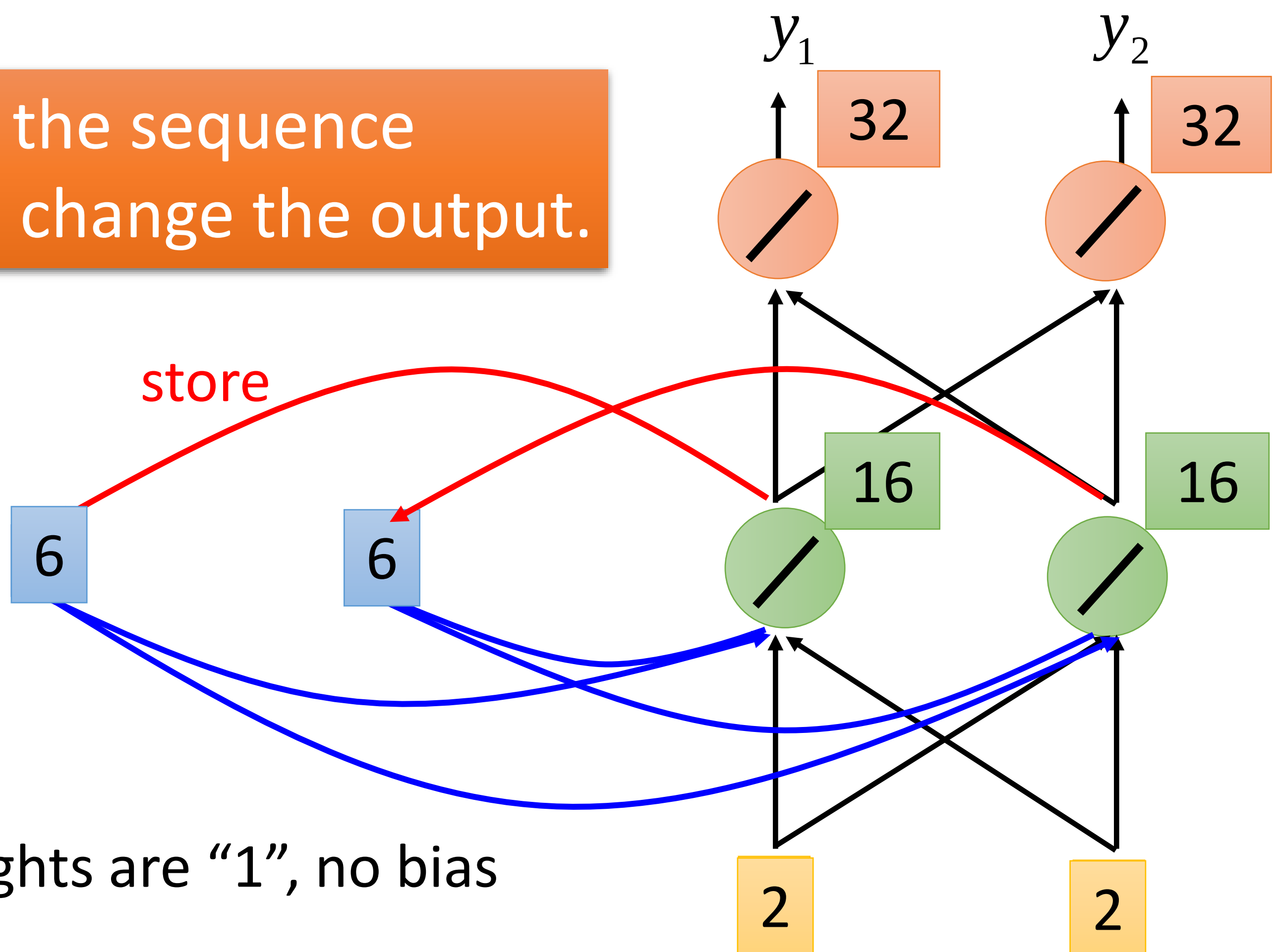
All activation functions are linear

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

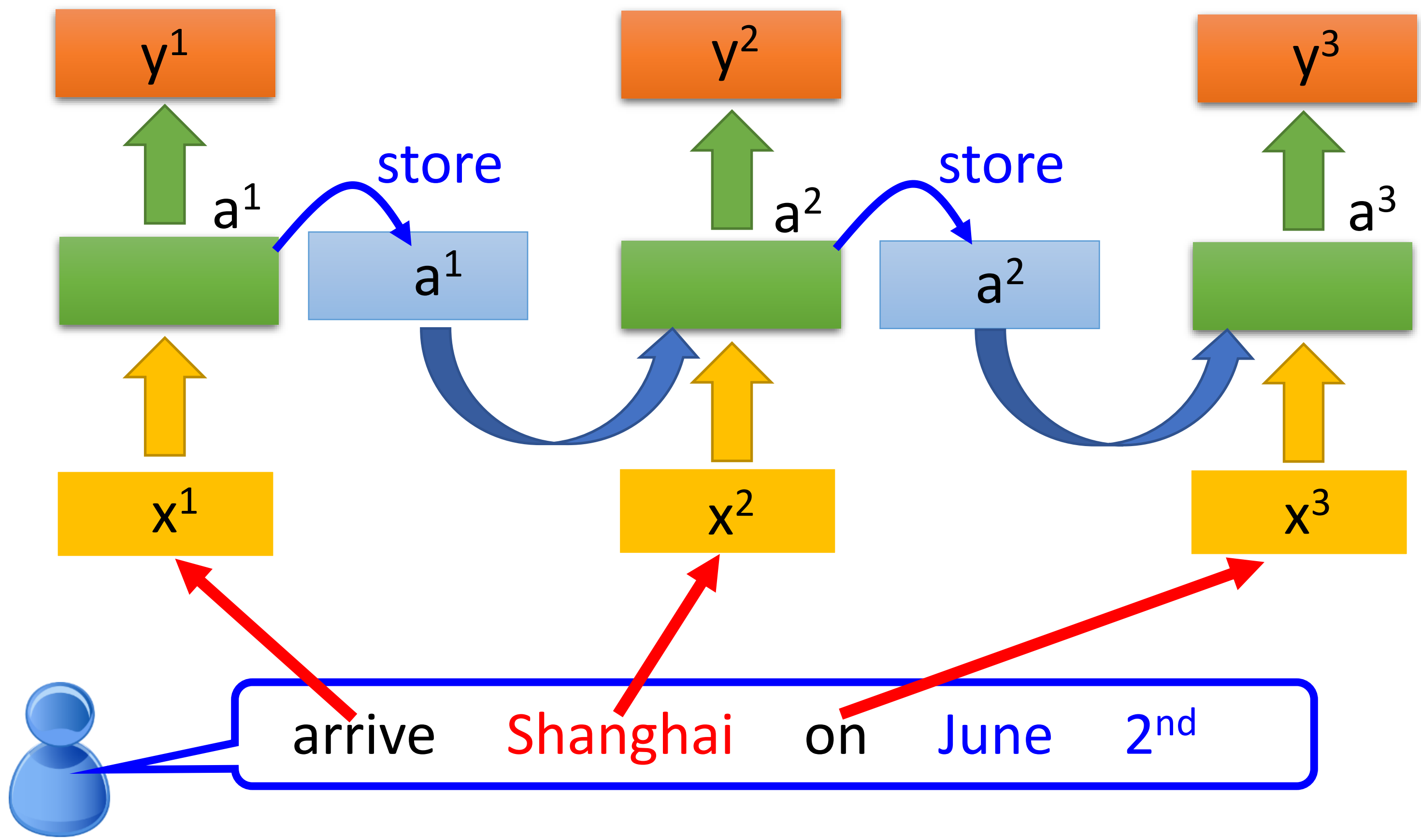
RNN

The same network is used again and again.

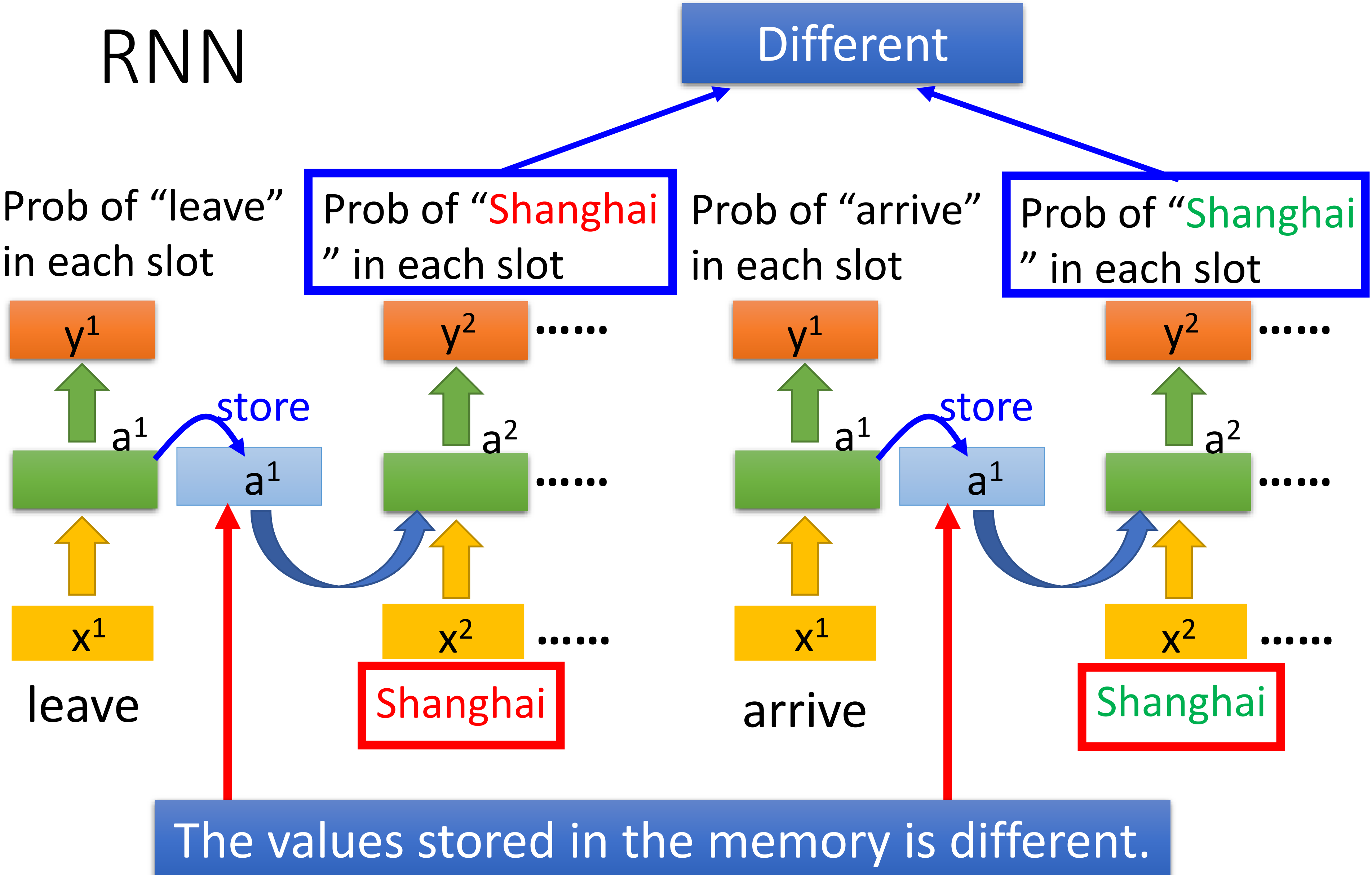
Probability of “arrive” in each slot

Probability of “Shanghai” in each slot

Probability of “on” in each slot

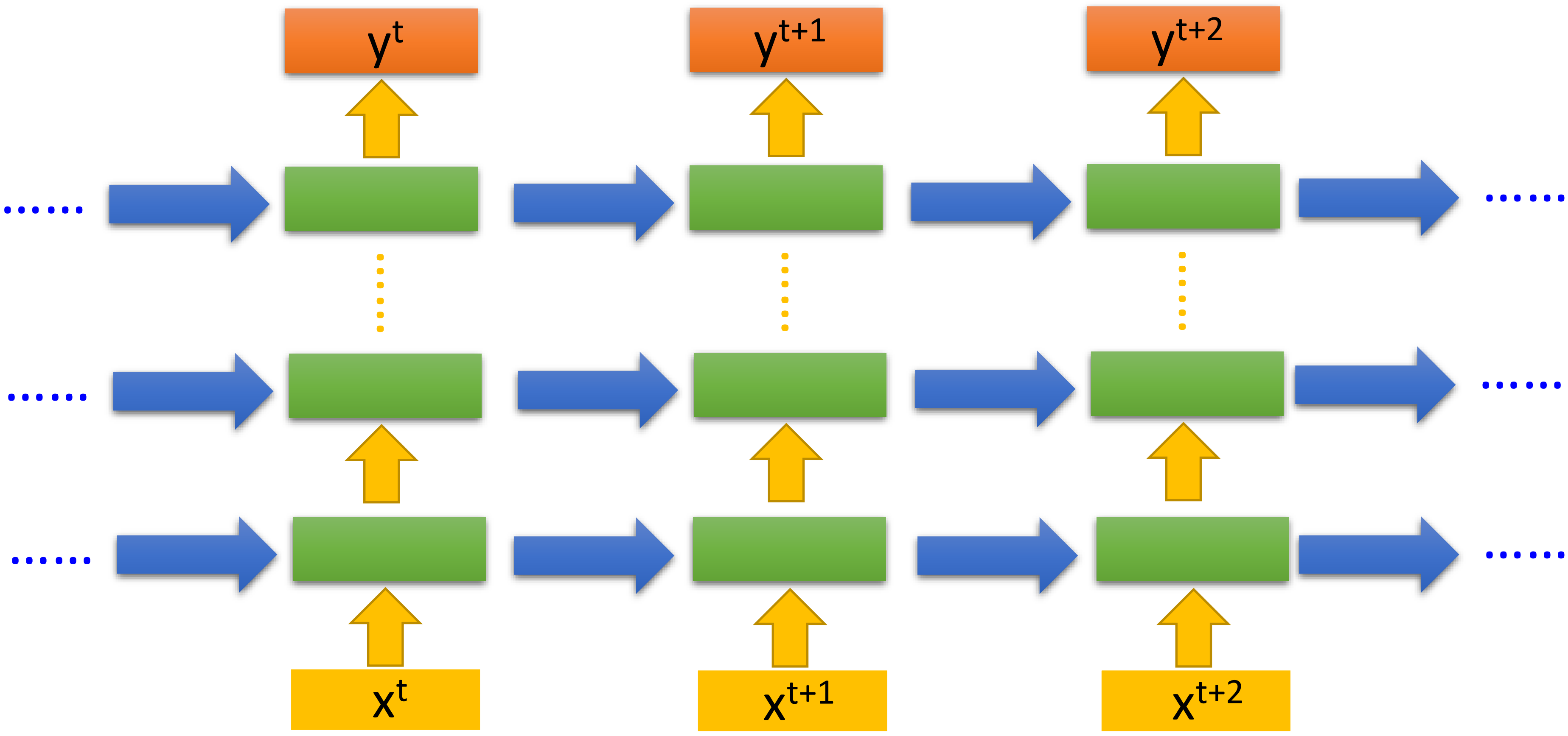


RNN

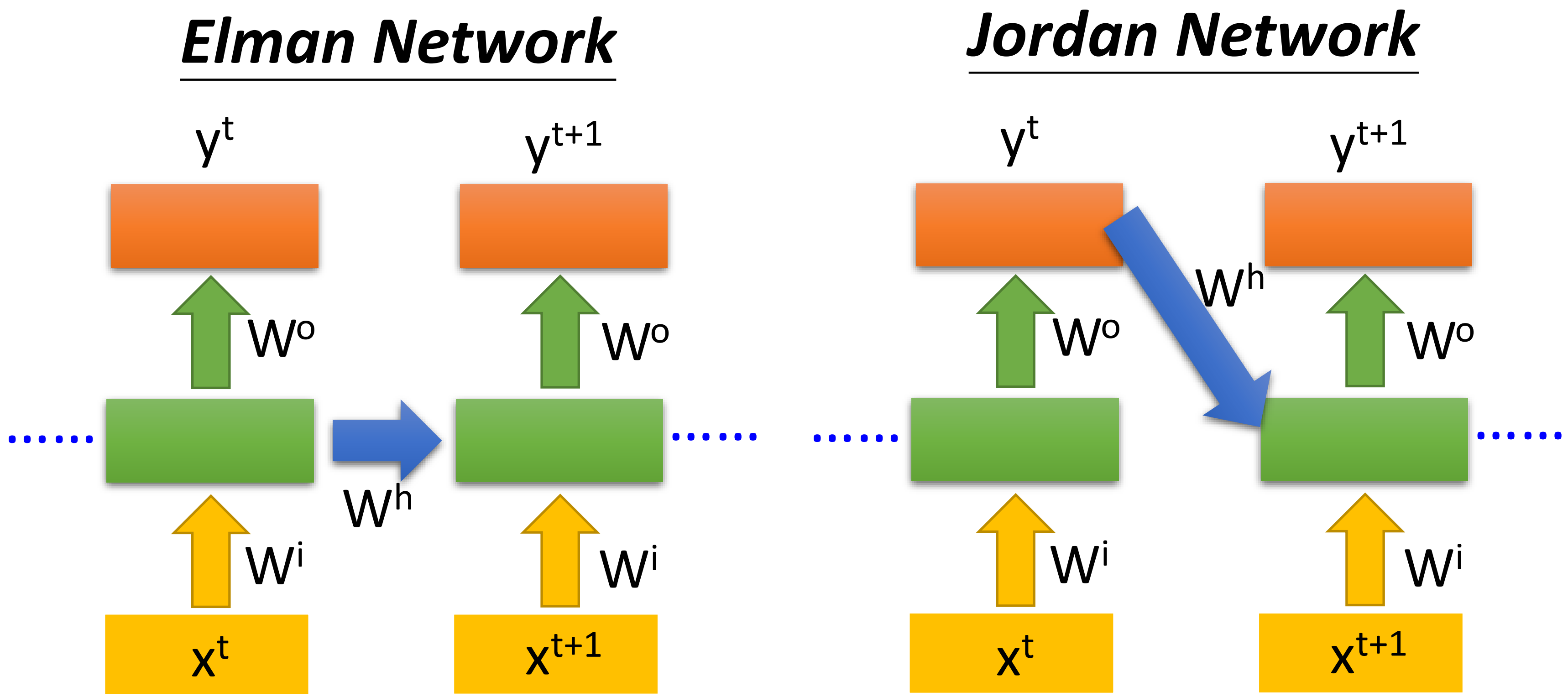


Different
Prob of "leave" in each slot
Prob of "Shanghai" in each slot
Prob of "arrive" in each slot
Prob of "Shanghai" in each slot
y1
y2
a1
store
a1
a2
x1
x2
leave
Shanghai
y1
y2
a1
store
a1
a2
x1
x2
arrive
Shanghai
The values stored in the memory is different.

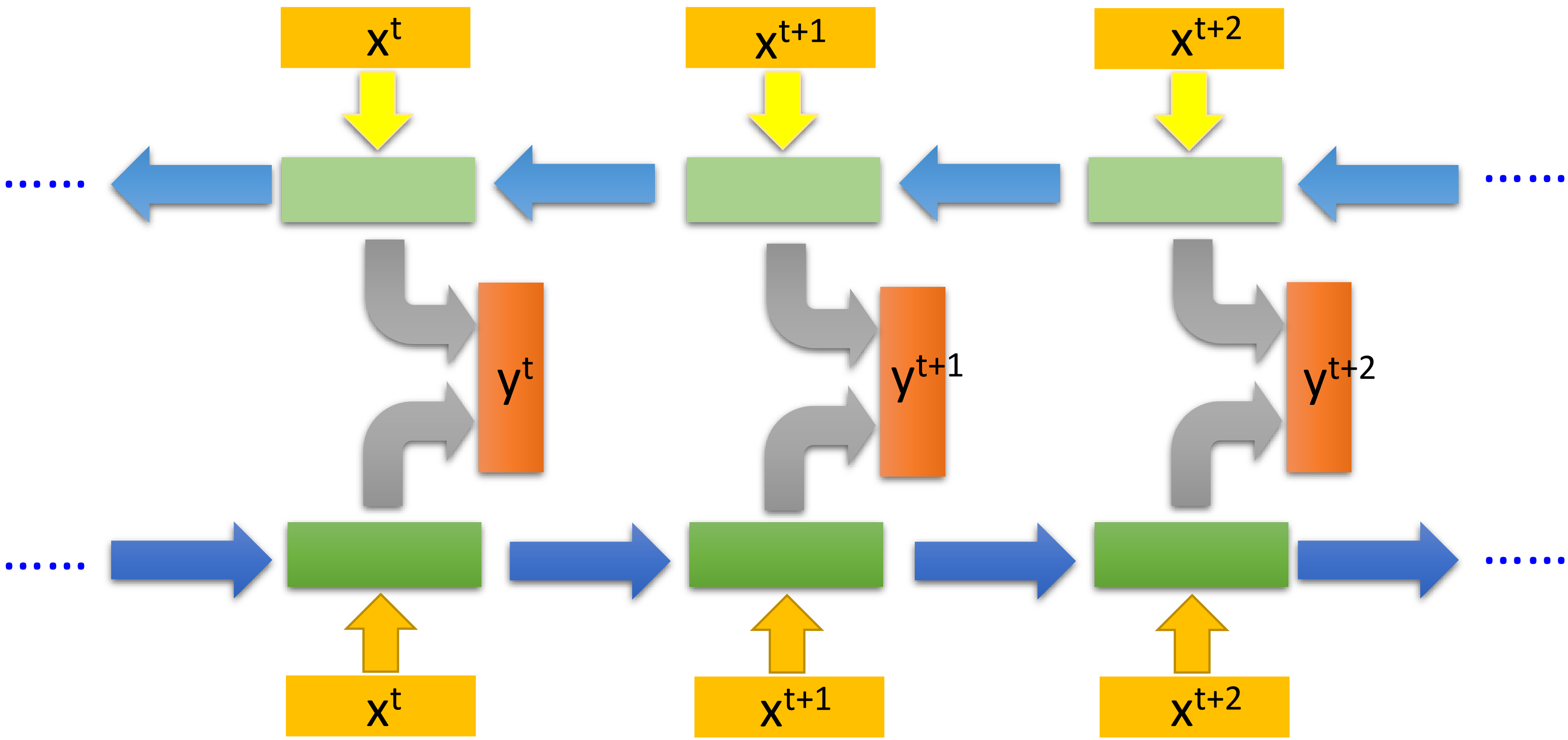
Of course it can be deep …



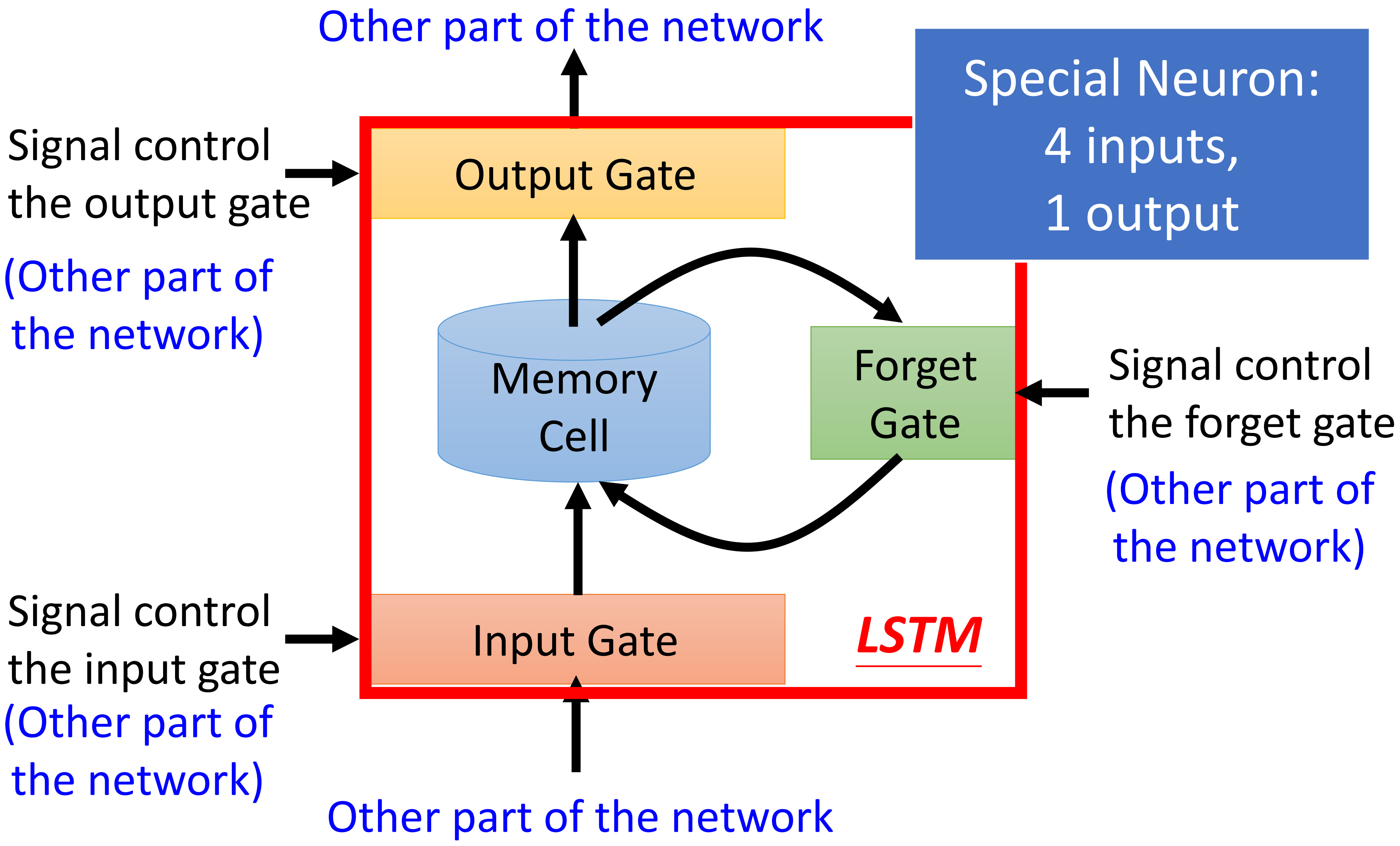
Elman Network & Jordan Network

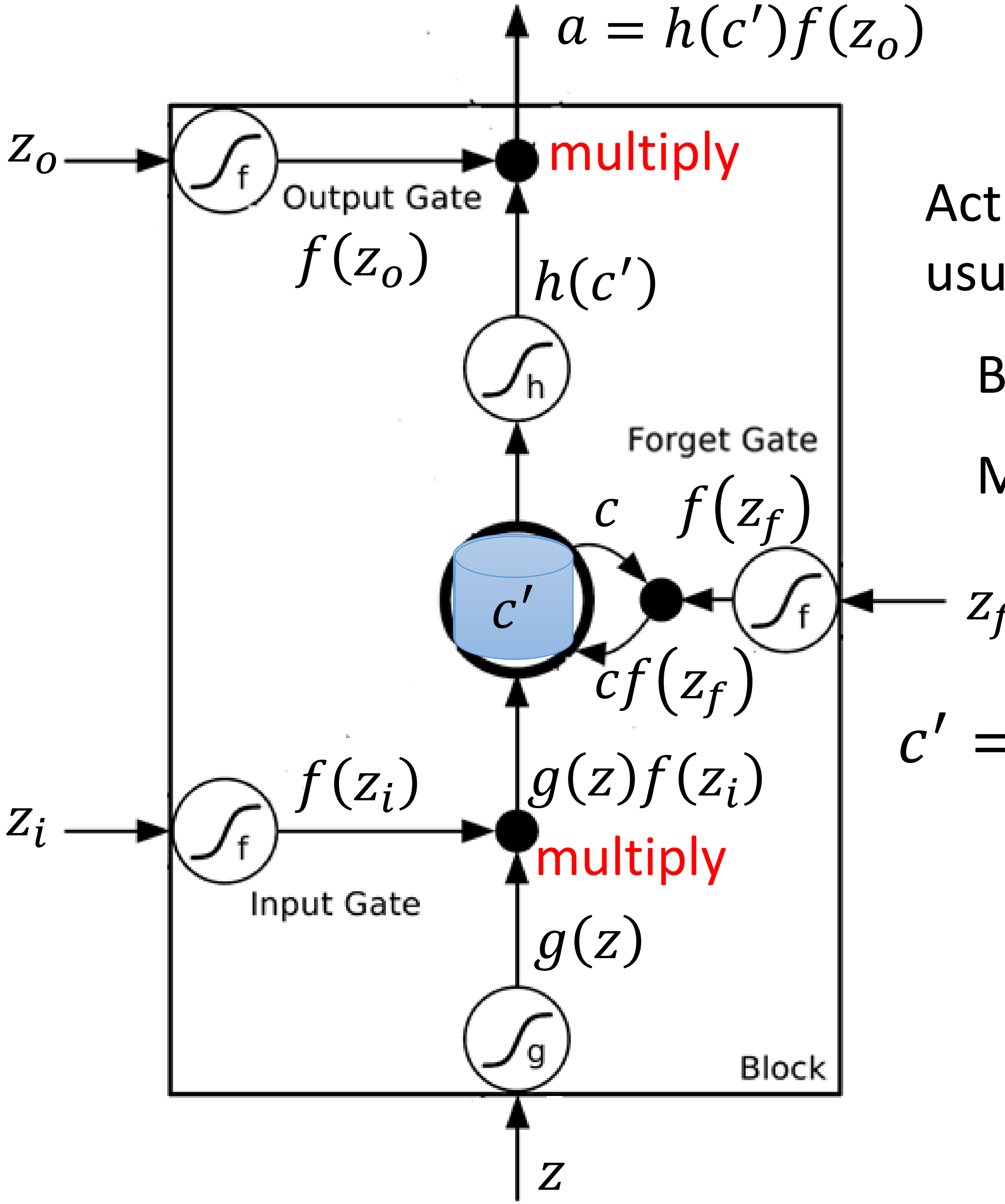


Bidirectional RNN



Long Short-term Memory (LSTM)



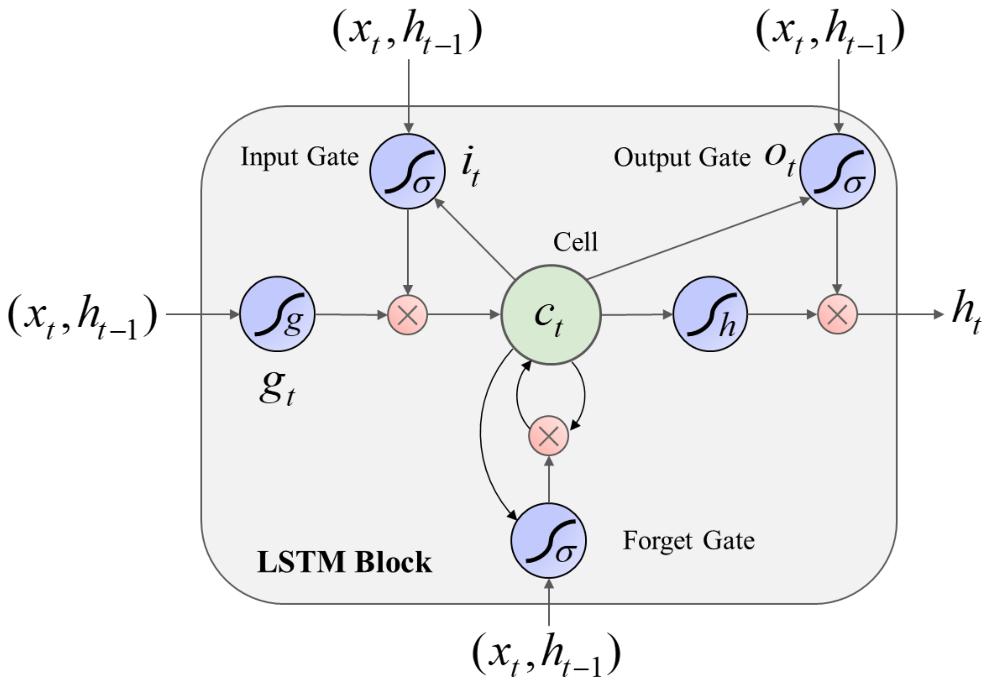


Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

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



$$f_t = sigmoid\left(W_{f,x}x_t + W_{f,h}h_{t-1} + b_f\right)$$

$$i_t = sigmoid\left(W_{i,x}x_t + W_{i,h}h_{t-1} + b_i\right)$$

$$g_t = \tanh\left(W_{g,x}x_t + W_{g,h}h_{t-1} + b_g\right)$$

$$c_t = g_t \cdot i_t + c_{t-1} \cdot f_t$$

$$o_t = sigmoid\left(W_{o,x}x_t + W_{o,h}h_{t-1} + b_o\right)$$

$$h_t = \tanh(c_t) \cdot o_t$$

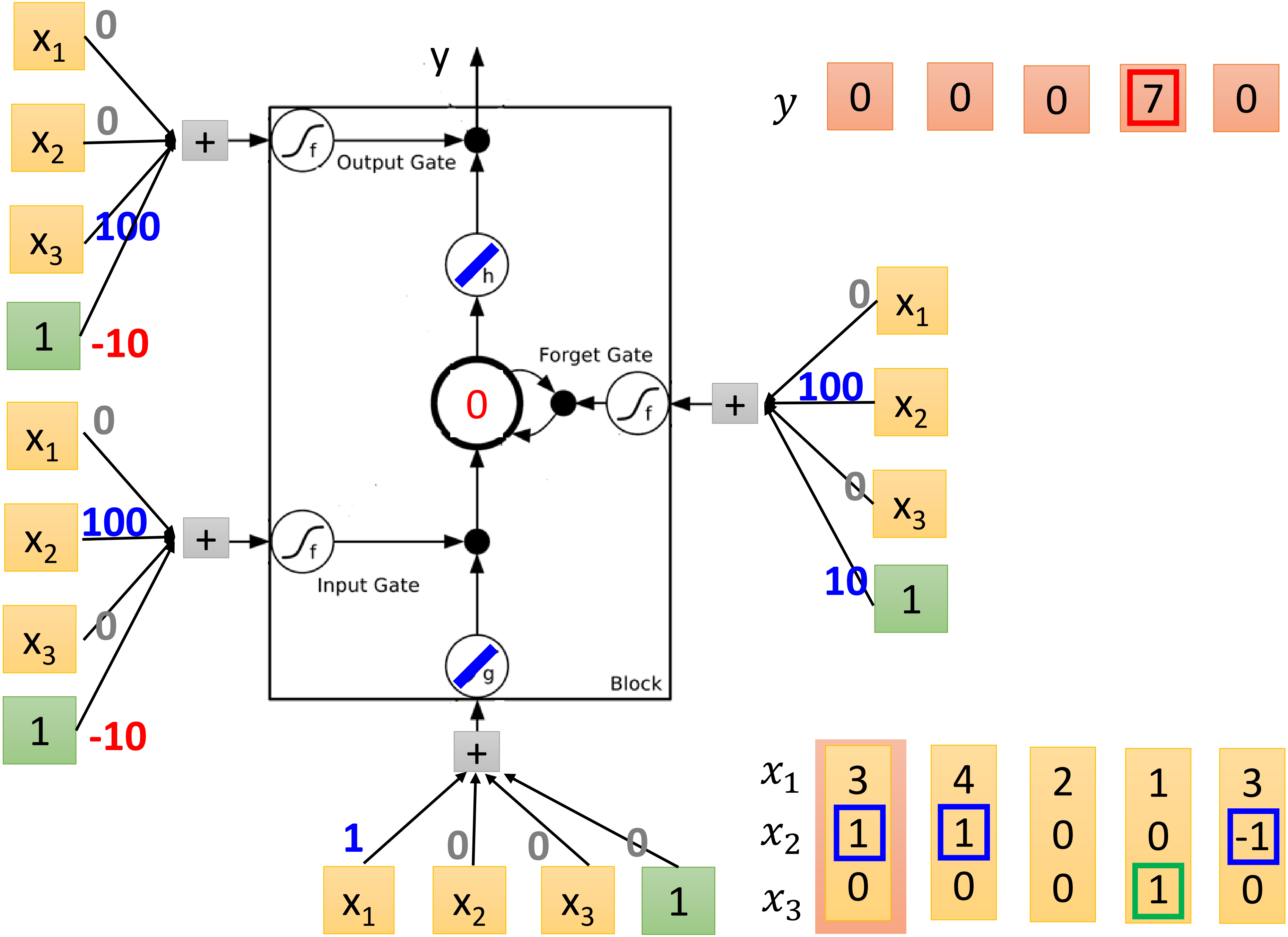
LSTM - Example

	0	0	3	3	7	7	7	0	6
x_1	1	3	2	4	2	1	3	6	1
x_2	0	1	0	1	0	0	-1	1	0
x_3	0	0	0	0	0	1	0	0	1
y	0	0	0	0	0	7	0	0	6

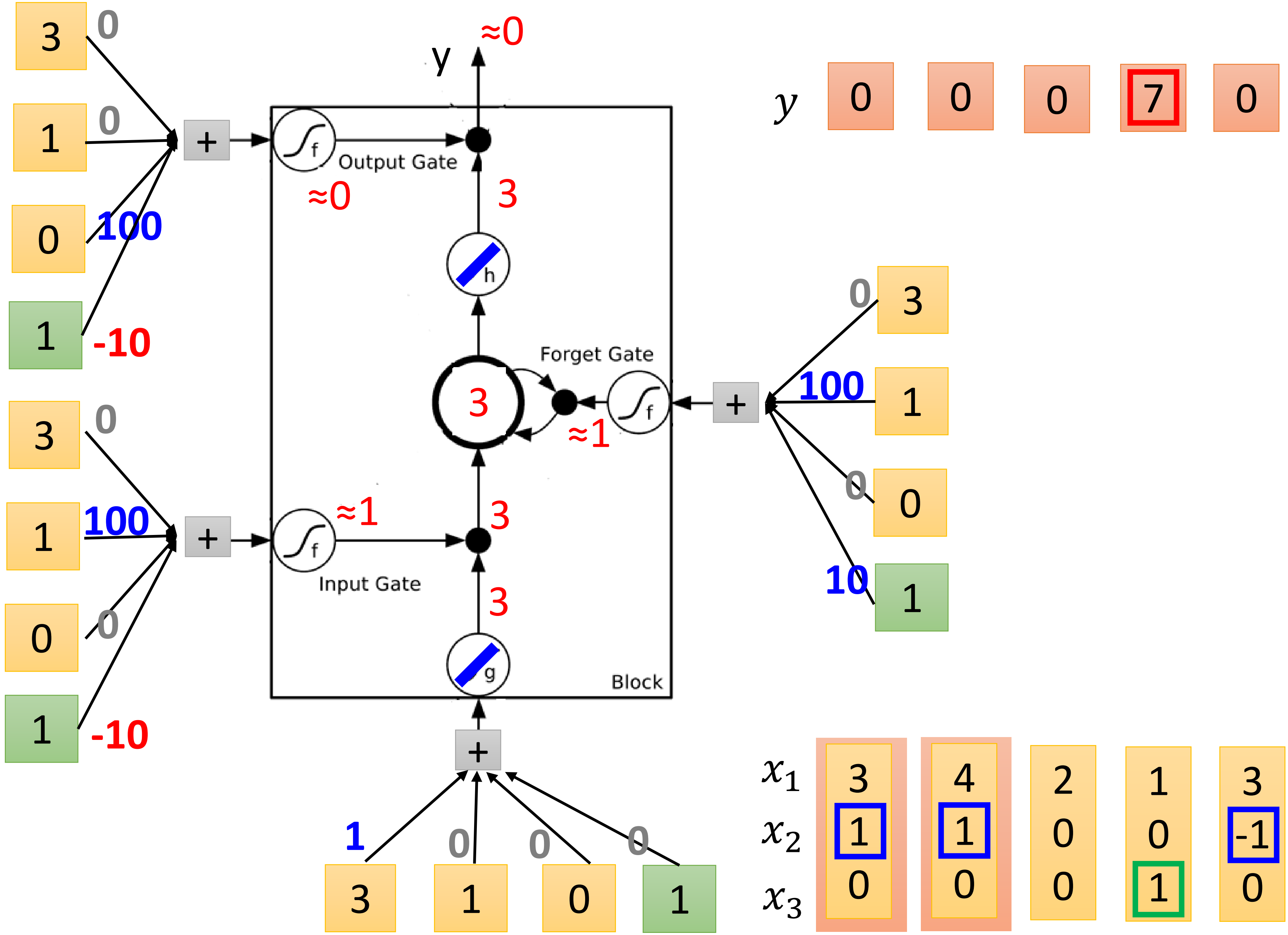
When x_2 = 1, add the numbers of x_1 into the memory

When x_2 = -1, reset the memory

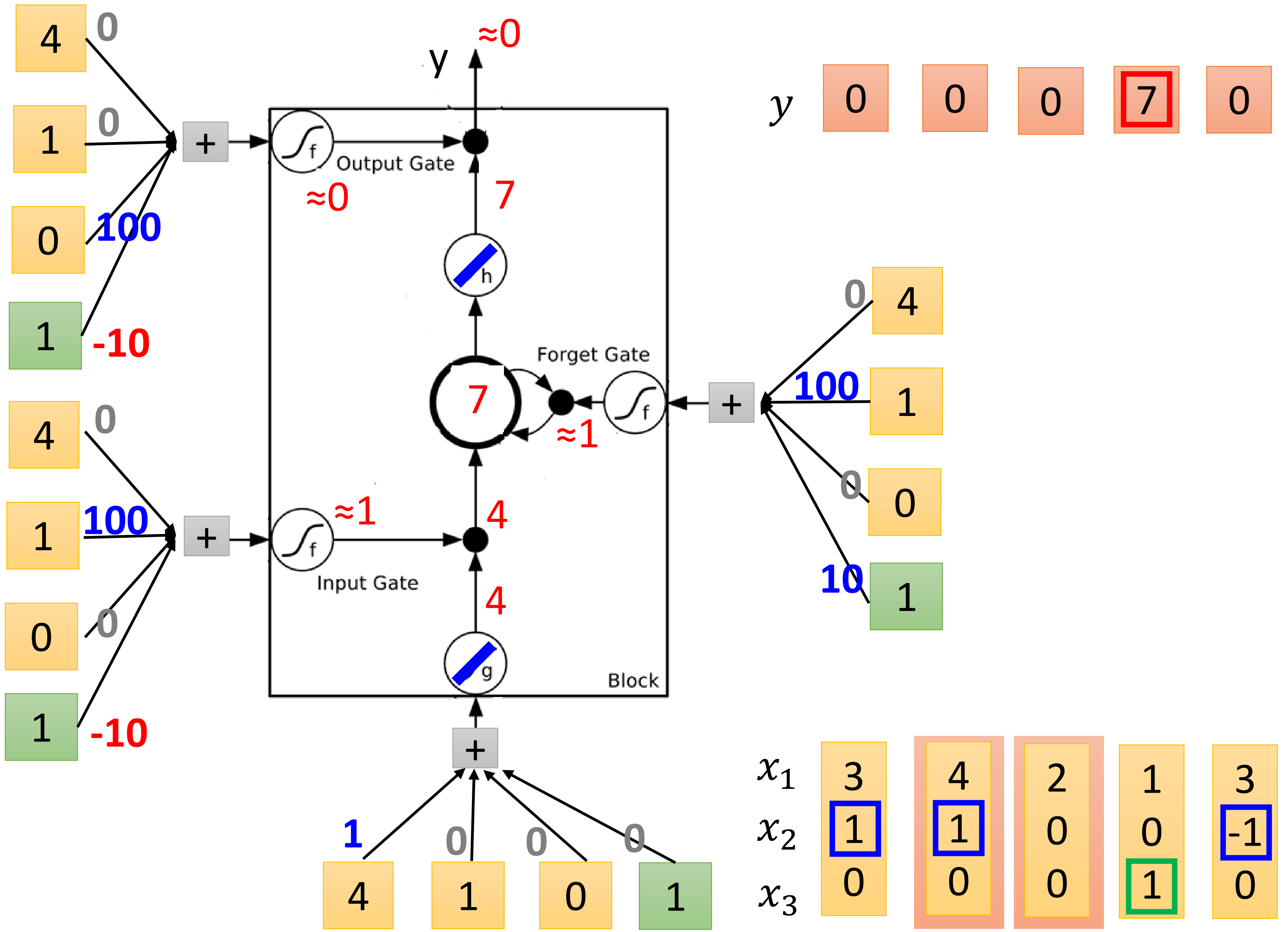
When x_3 = 1, output the number in the memory.



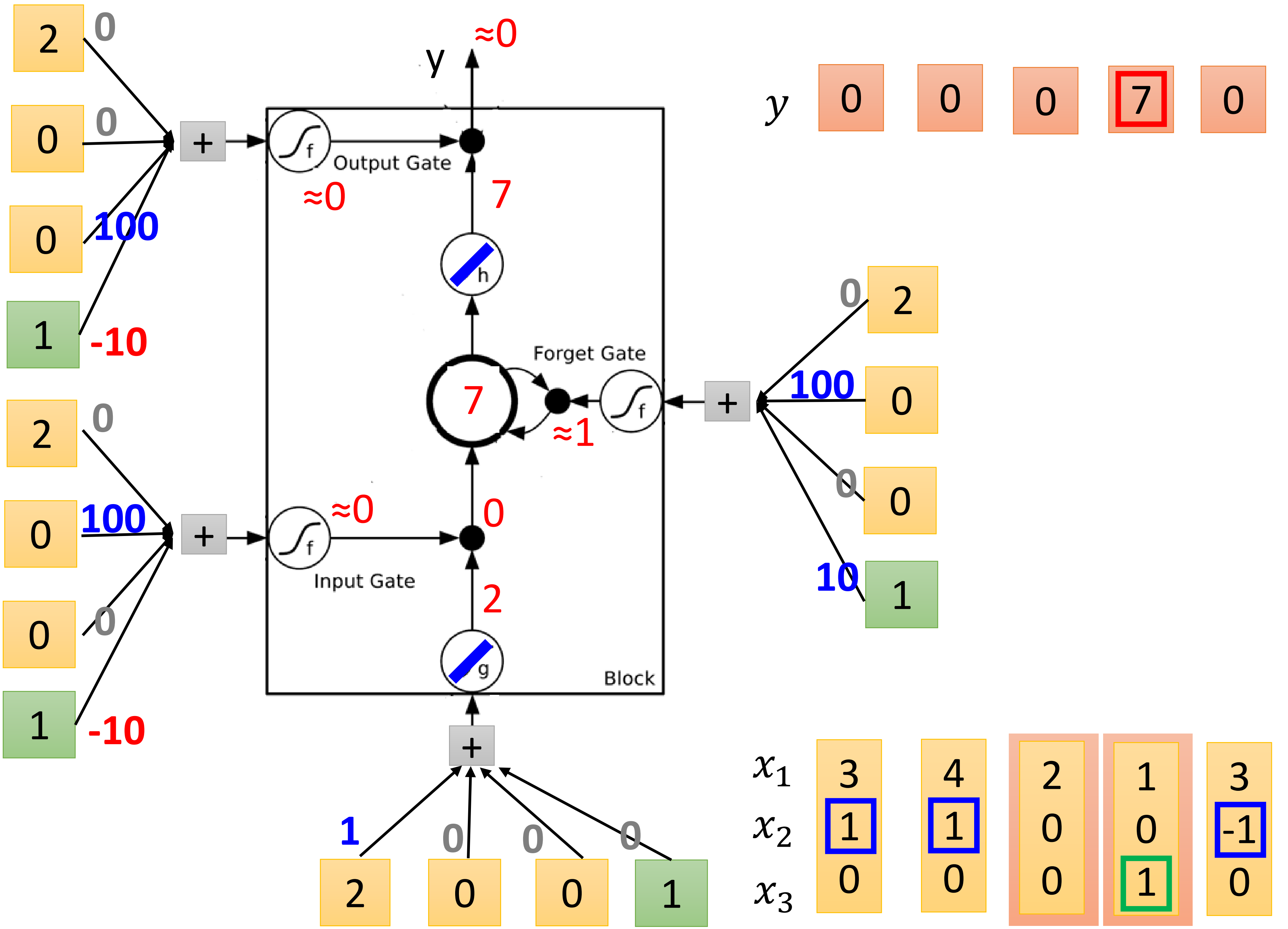
y
Output Gate
Forget Gate
Input Gate
Block
f
h
g
0
x1
x2
x3
1
0
0
100
-10
0
100
0
-10
0
100
0
10
1
0
0
0
y
0
0
0
7
0
x1
x2
x3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0



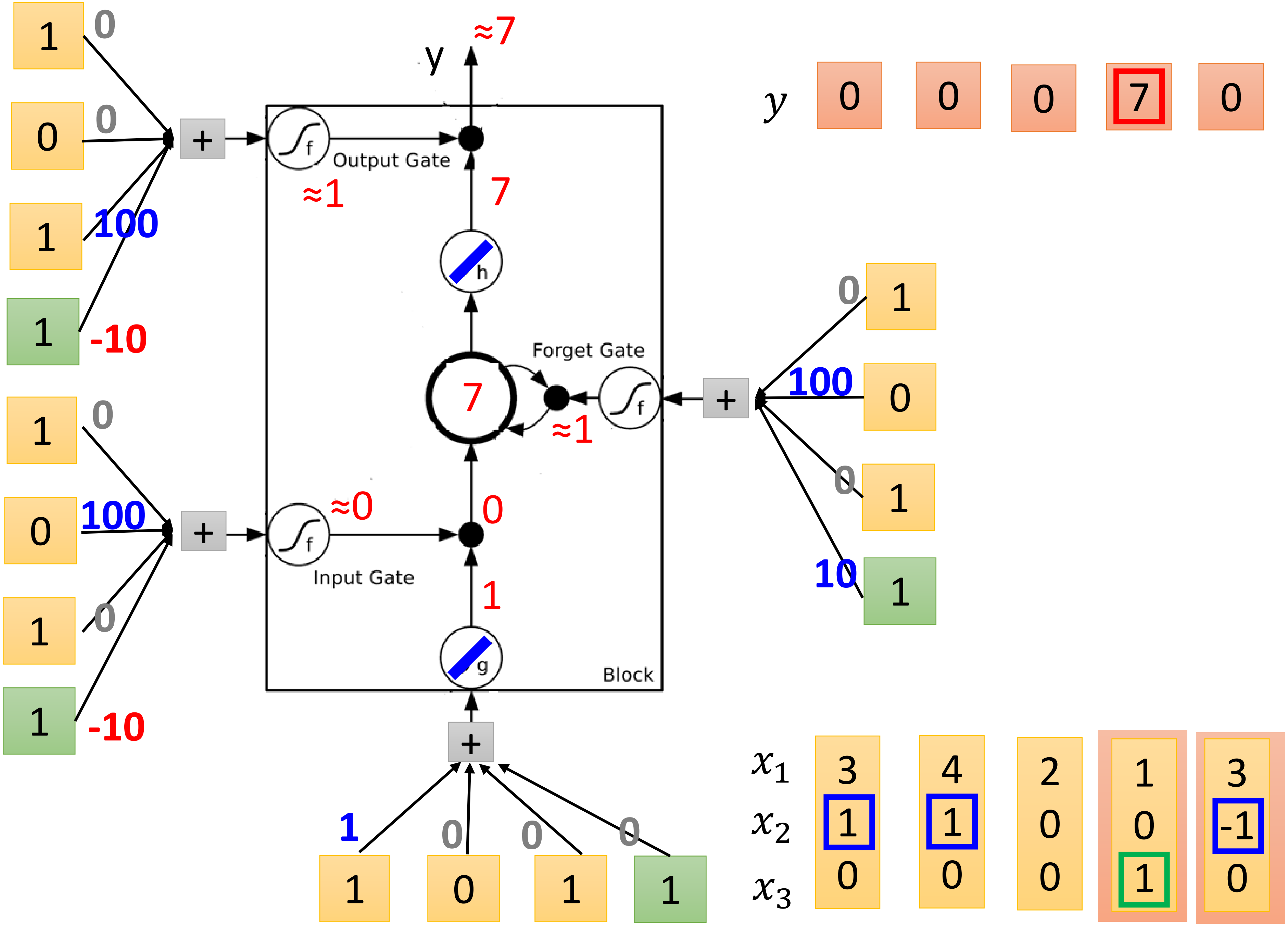
y
≈0
Output Gate
≈0
3
h
Forget Gate
3
≈1
≈1
3
Input Gate
3
g
Block
100
-10
100
-10
100
10
1
y
0
0
0
7
0
x1
x2
x3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0



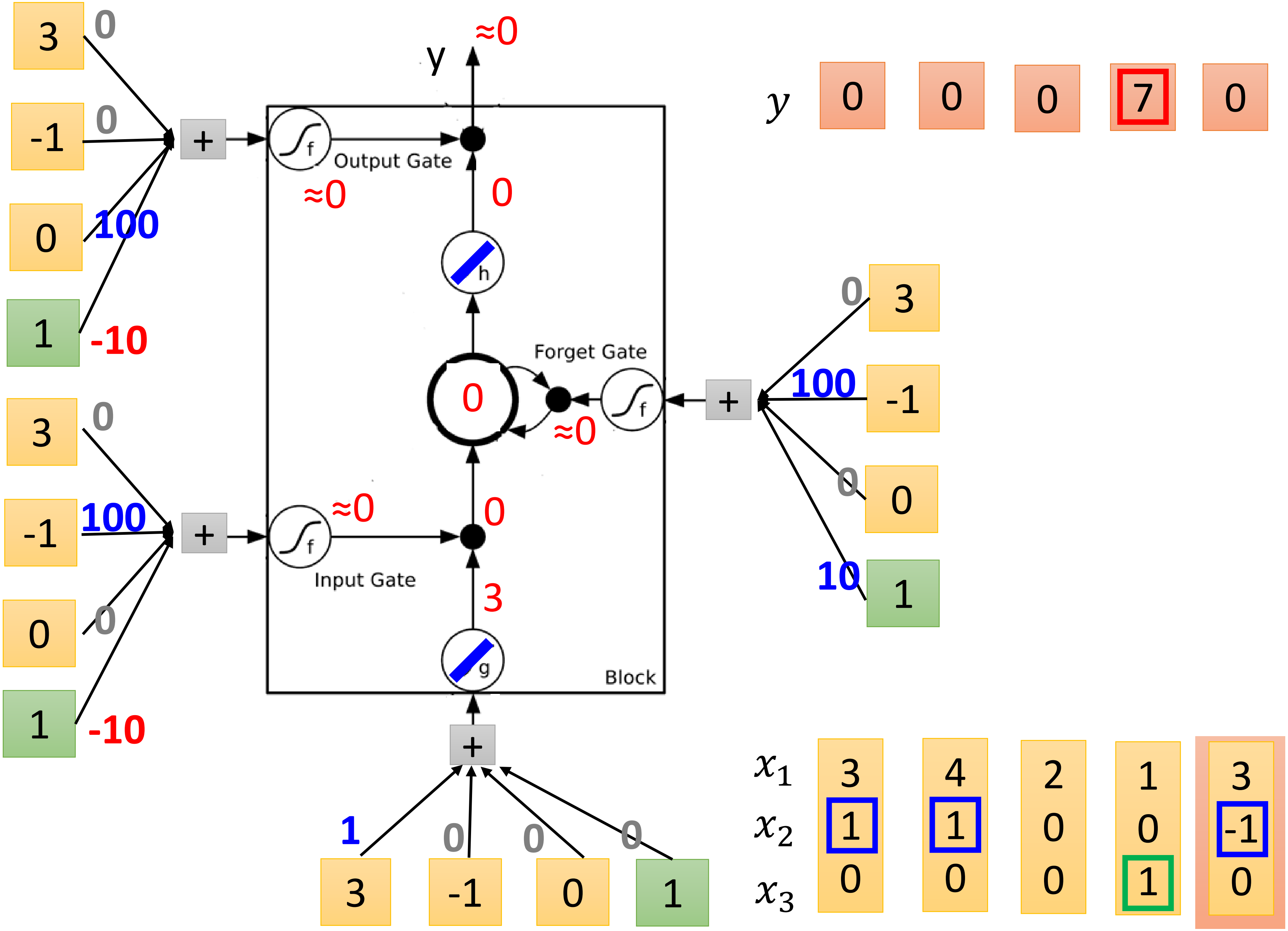
y
≈0
Output Gate
≈0
7
Forget Gate
7
≈1
≈1
4
Input Gate
4
Block
4
0
1
0
0
100
1
-10
4
0
1
100
0
0
1
-10
0
4
100
1
0
0
10
1
1
4
0
1
0
0
0
1
y
0
0
0
7
0
x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0



y
≈0
Output Gate
≈0
7
h
Forget Gate
7
≈1
≈0
0
Input Gate
2
g
Block
2
0
0
1
100
1
-10
2
100
0
0
1
-10
0
2
100
0
0
0
10
1
1
2
0
0
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



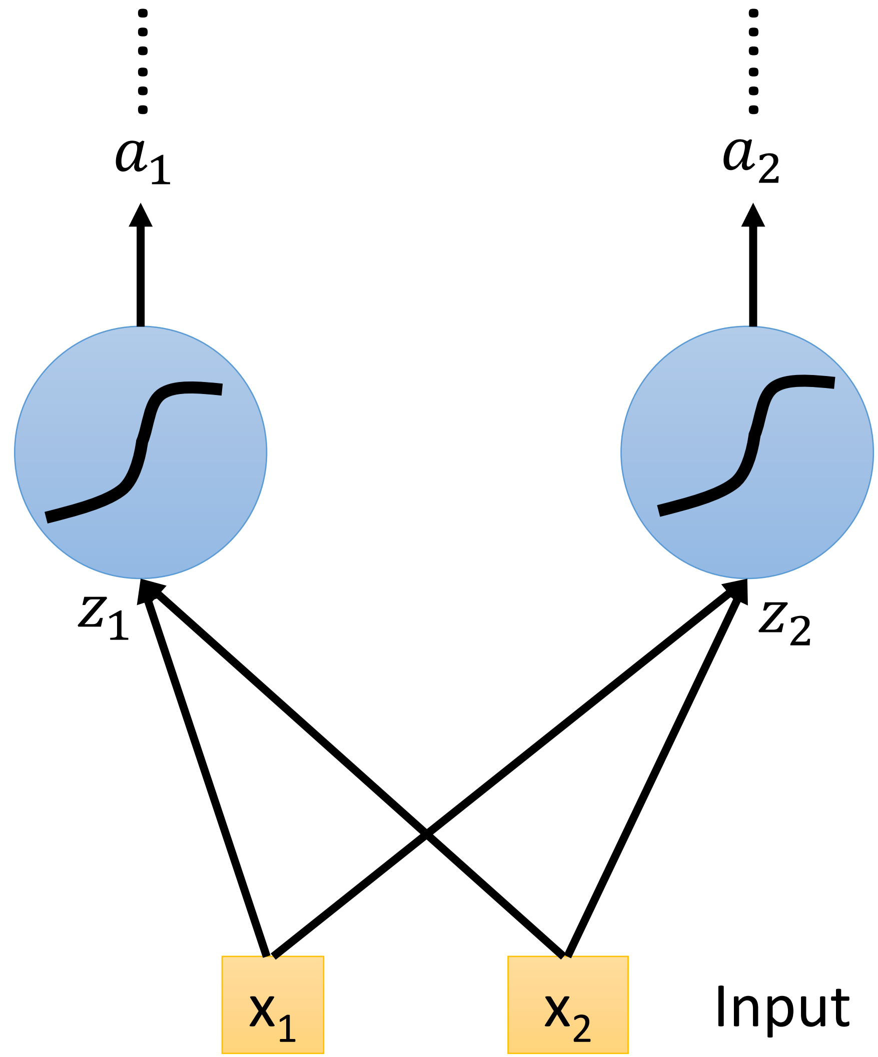
y
≈7
Output Gate
≈1
7
h
Forget Gate
7
≈1
≈0
0
Input Gate
1
g
Block
100
-10
10
y
0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0

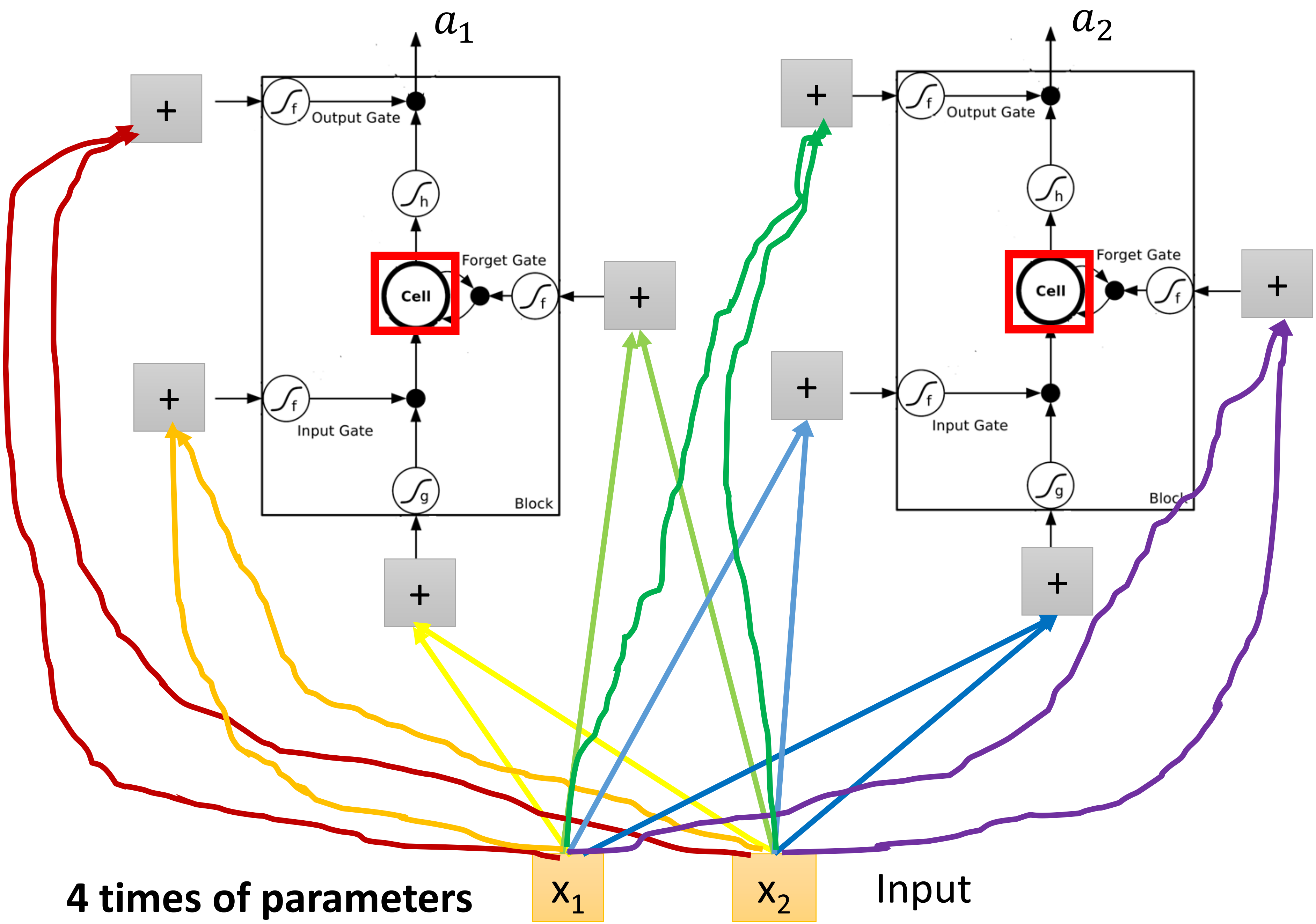


y
≈0
Output Gate
Forget Gate
Input Gate
Block
f
h
g
+
3
-1
0
1
0
100
-10
10
≈0
3
y
0
0
0
7
0
x_1
x_2
x_3
3
1
0
4
1
0
2
0
0
1
0
1
3
-1
0

Original Network:

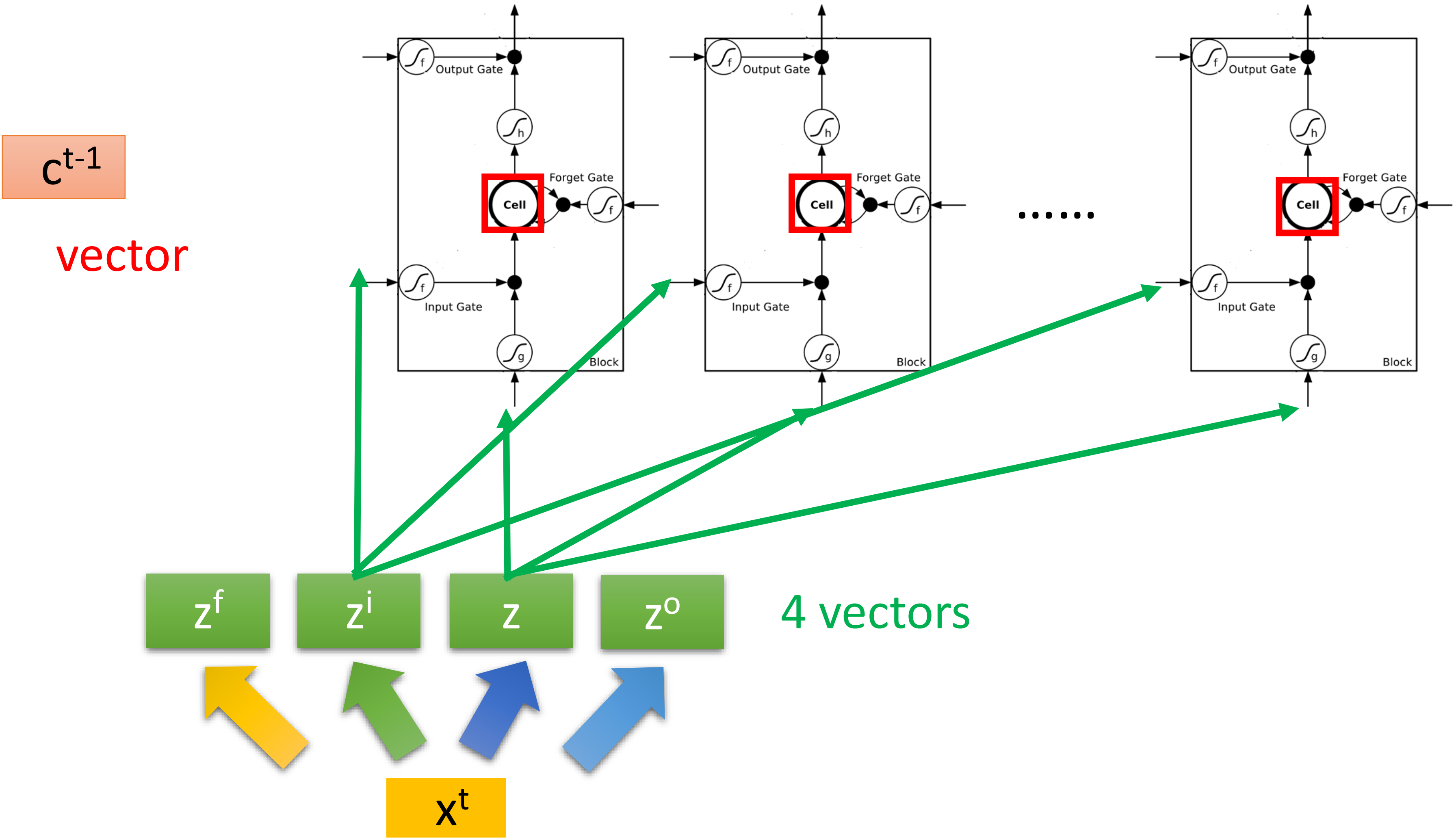
➢ Simply replace the neurons with LSTM





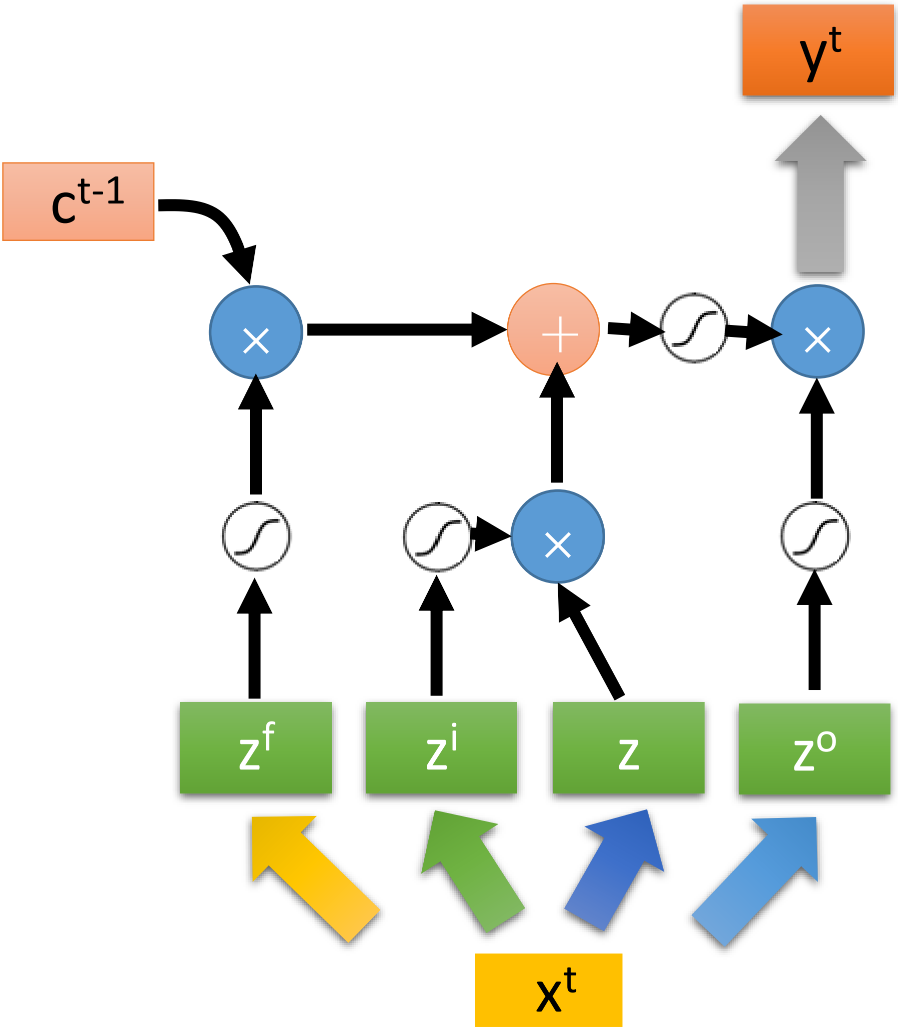
4 times of parameters

LSTM

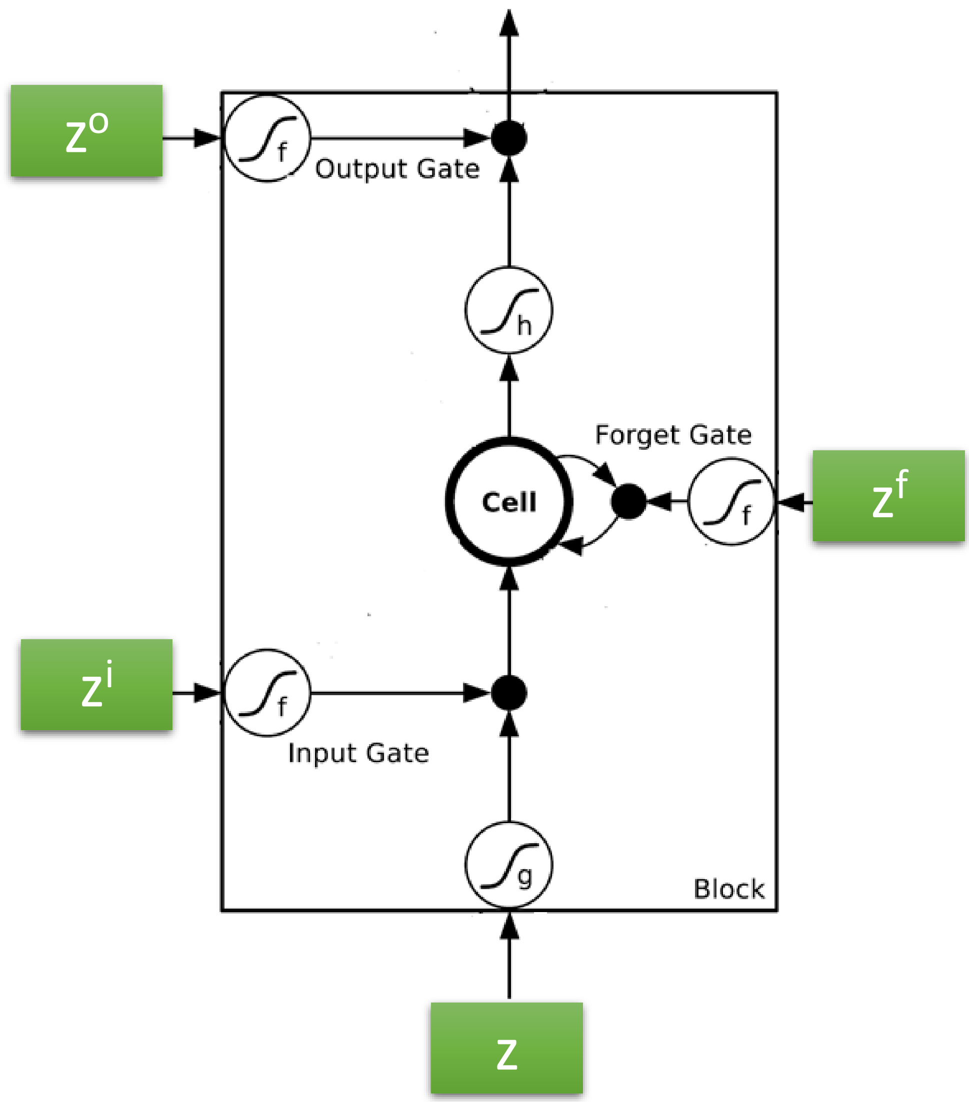


c^{t-1}
vector
Output Gate
Forget Gate
Cell
Input Gate
Block
......
z^f
z^i
z
z^o
4 vectors
x^t

LSTM



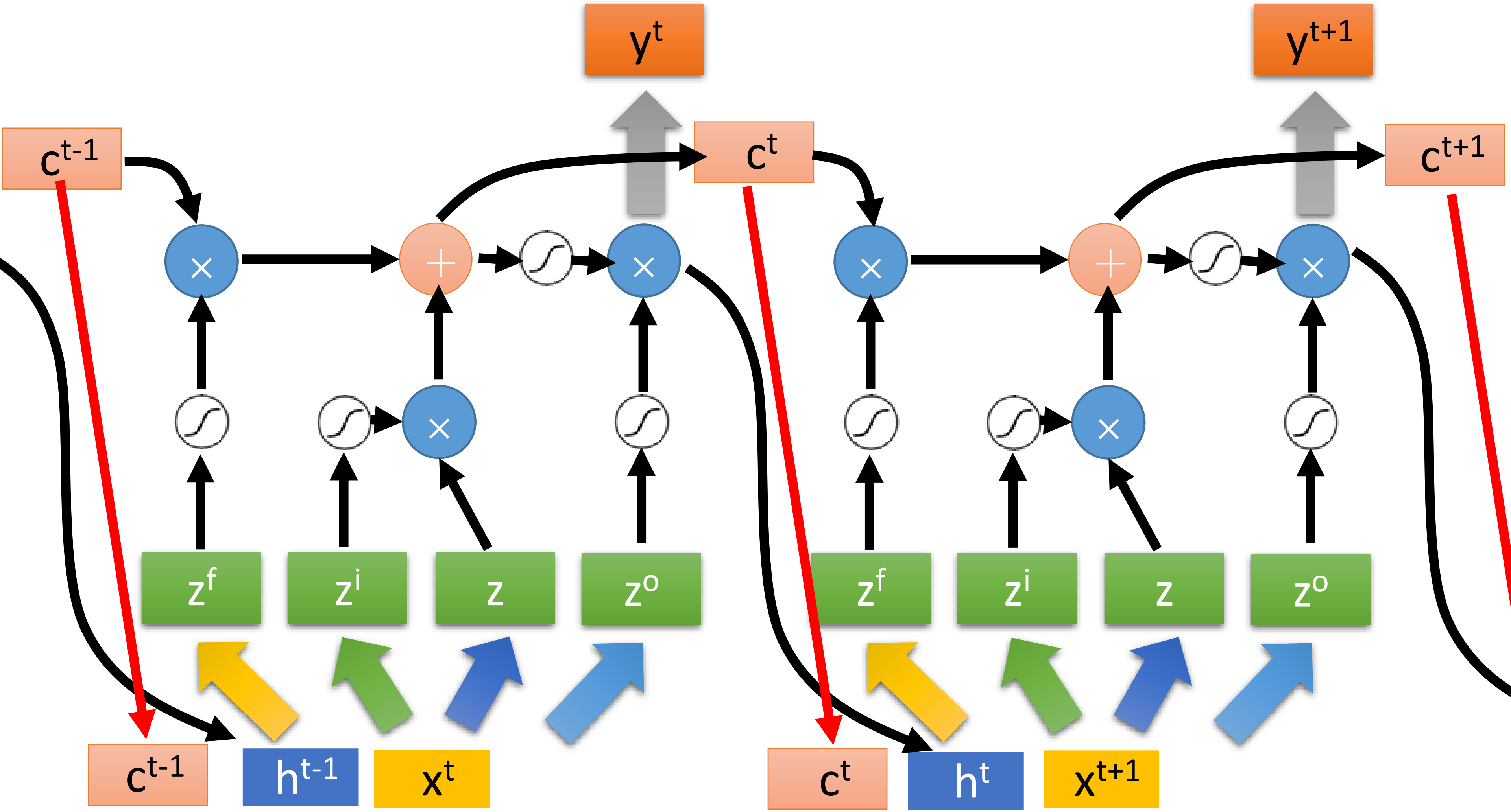
c^{t-1}
y^t
z^f
z^i
z
z^o
x^t



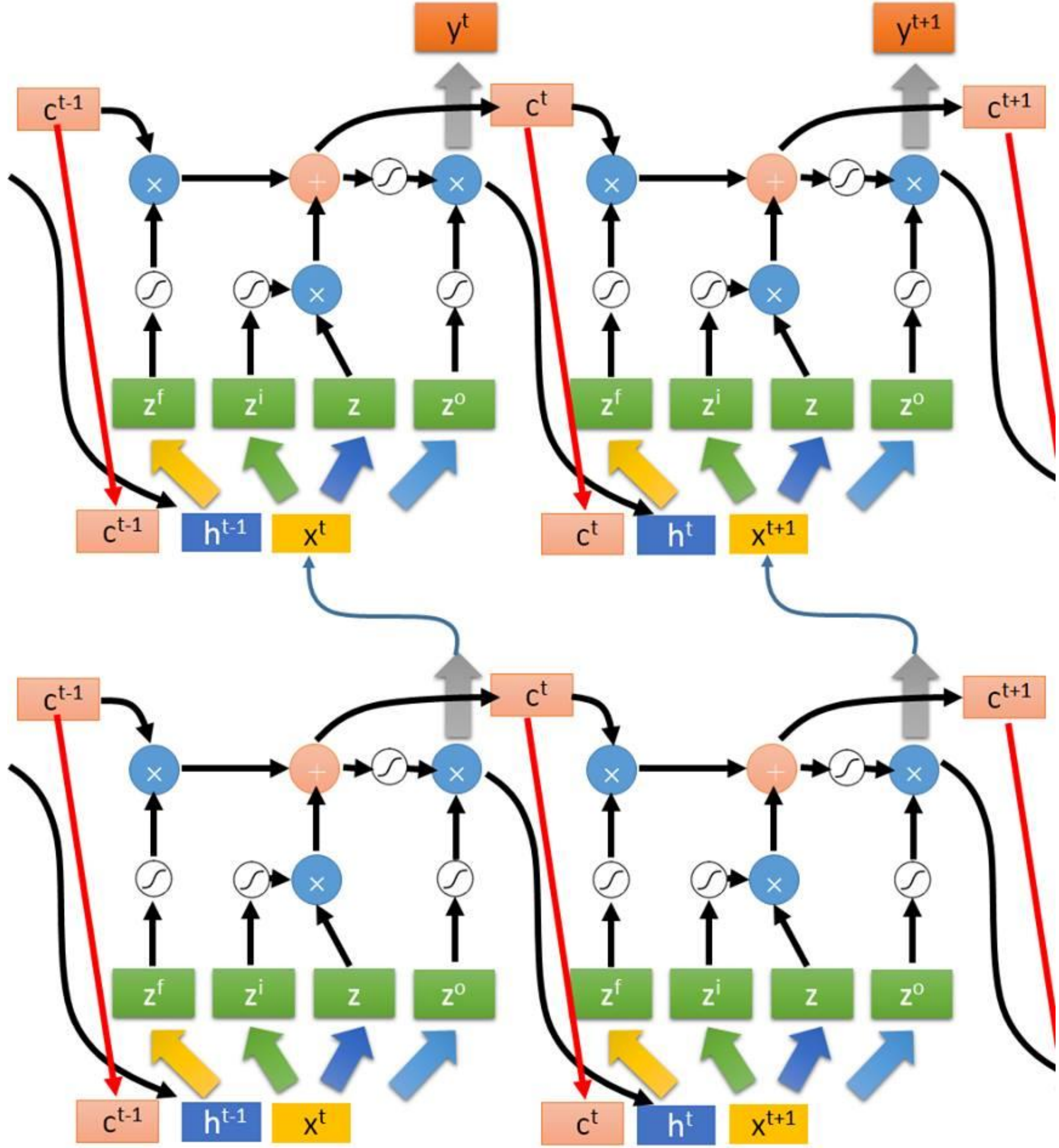
z^o
f
Output Gate
h
Forget Gate
Cell
f
z^f
z^i
f
Input Gate
g
Block
z

LSTM

Extension: “peephole”



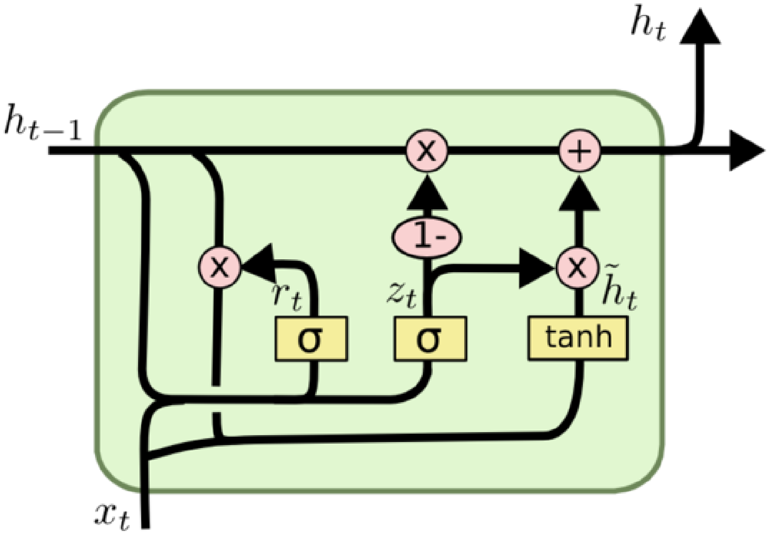
Multiple-layer LSTM



This is quite standard now.

https://img.komicolle.org/2015-09-20/src/14426967627131.gif

Gated Recurrent Unit (GRU)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

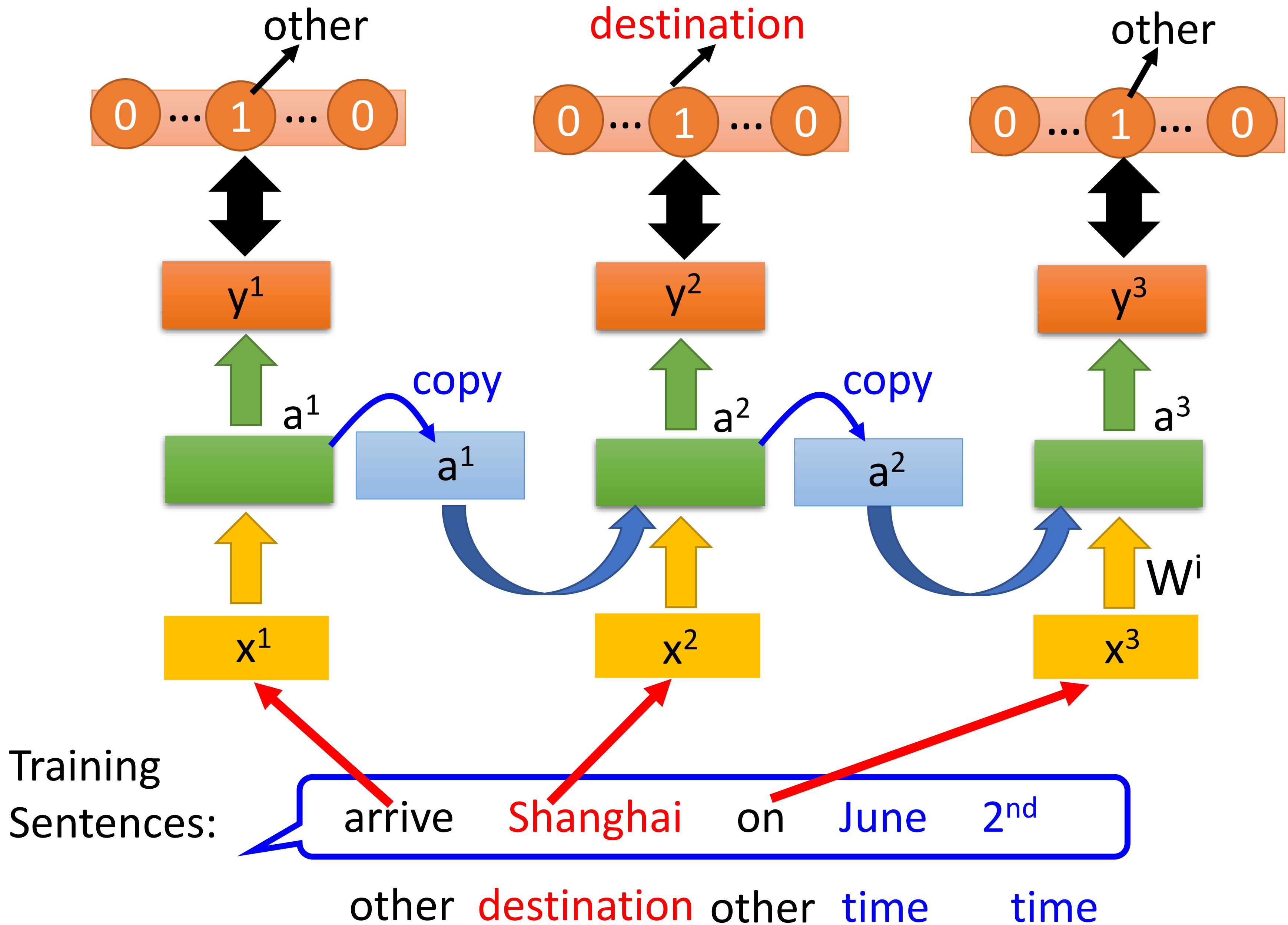
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

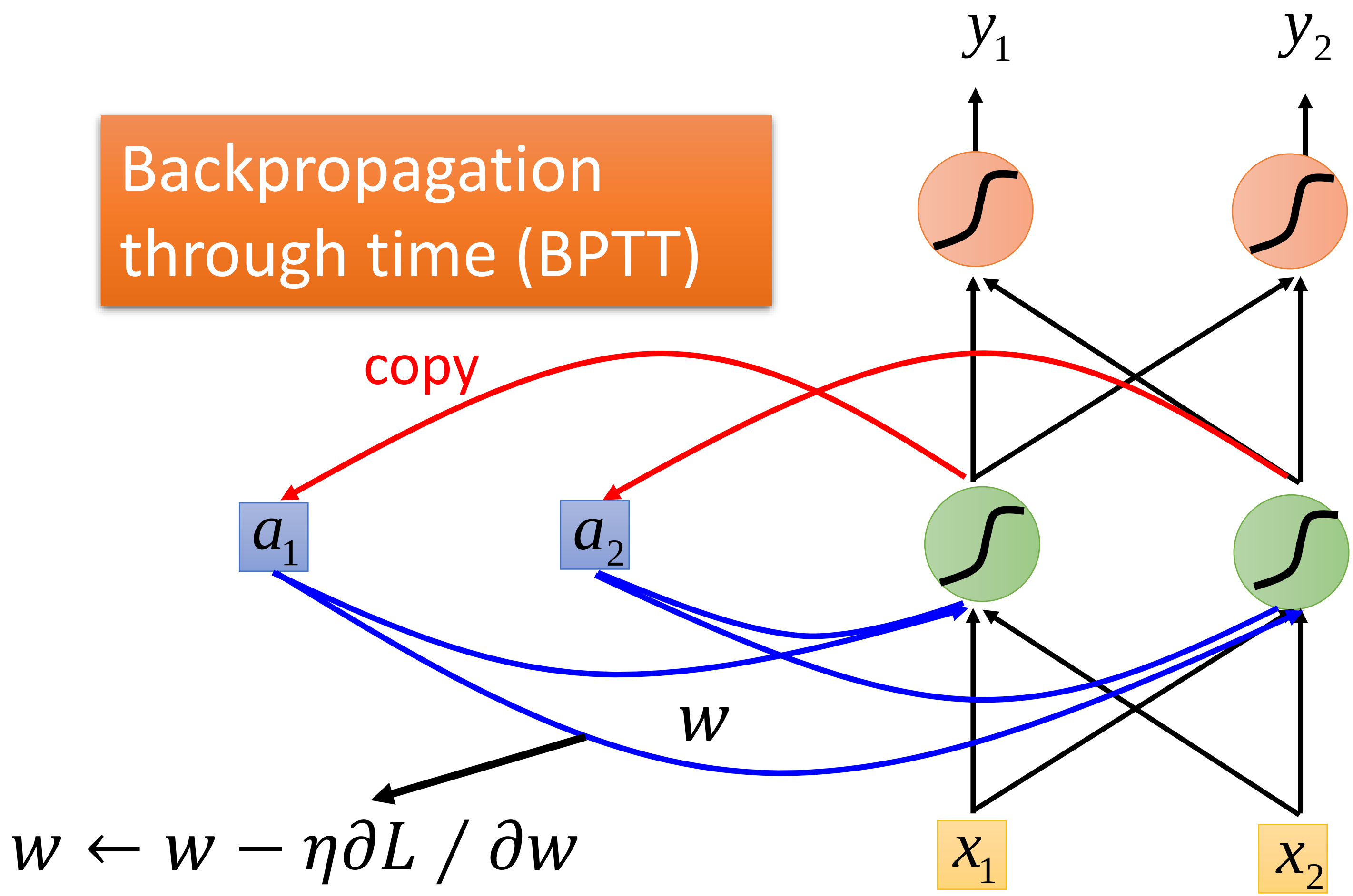
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Only two gates: reset gate and update gate.

Learning Target



Learning

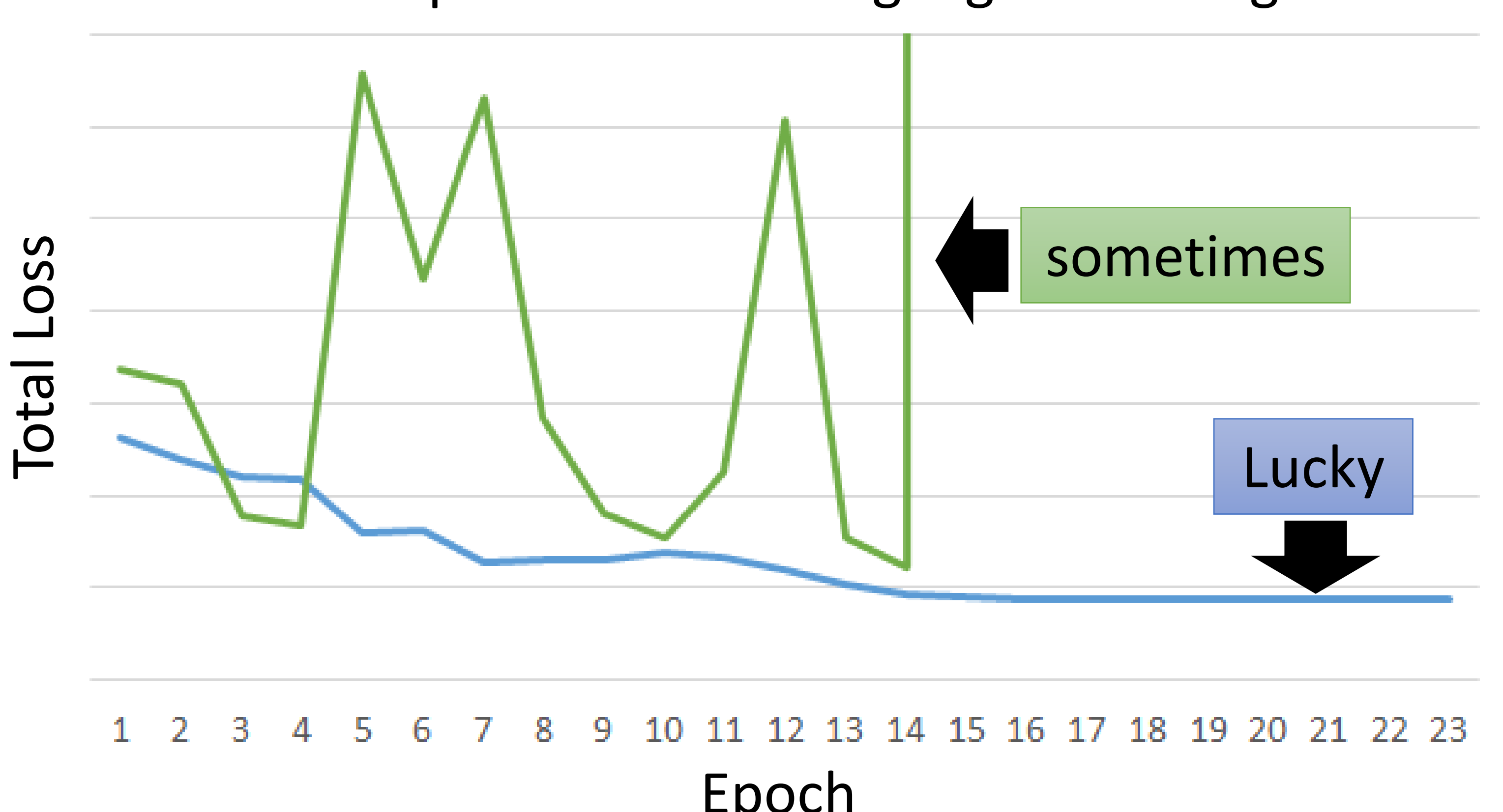


Backpropagation through time (BPTT)
copy
y_1
y_2
a_1
a_2
w
x_1
x_2
$w \leftarrow w - \eta \partial L / \partial w$

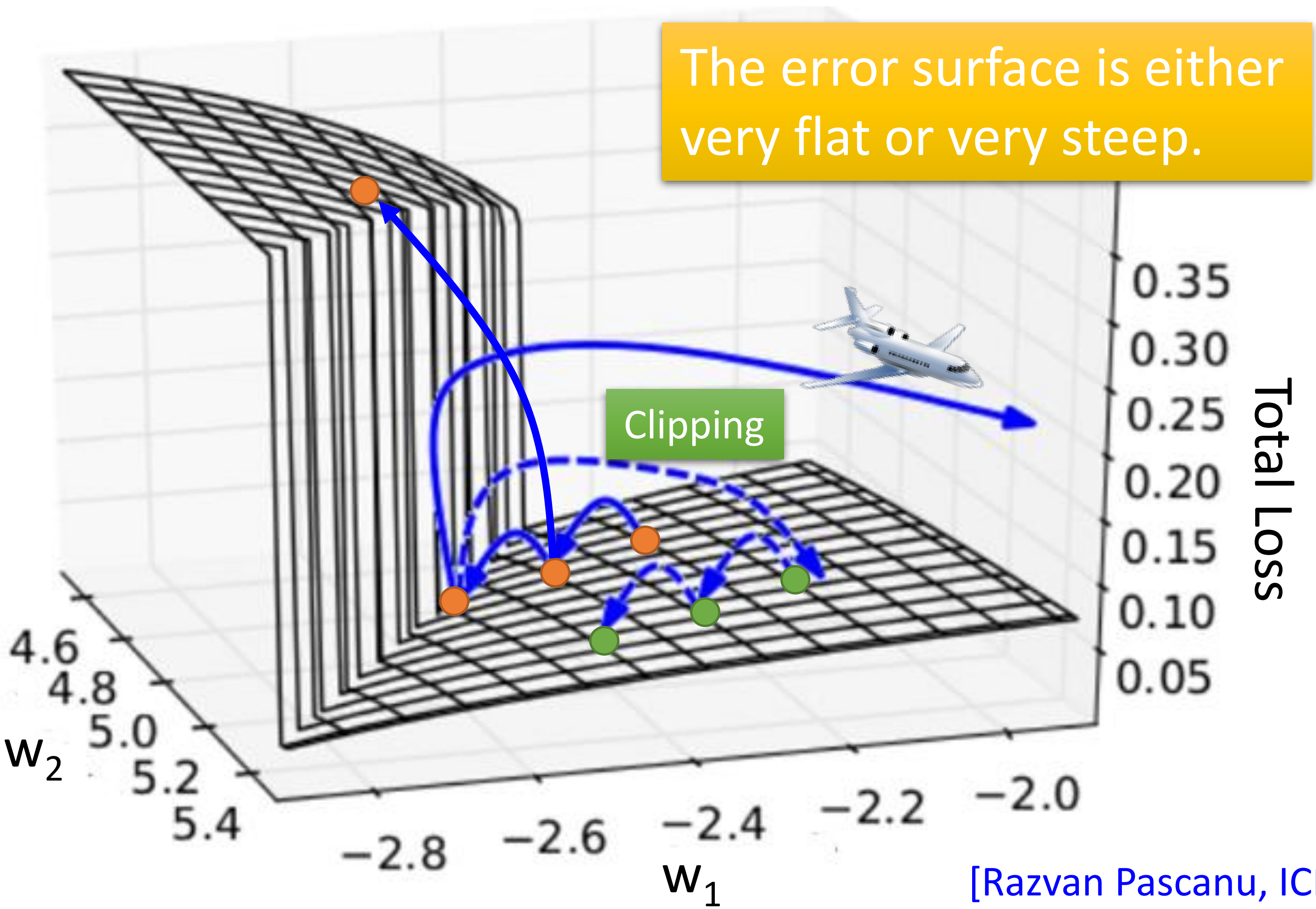
Unfortunately ……

- RNN-based network is not always easy to learn

Real experiments on Language modeling



The error surface is rough.



[Razvan Pascanu, ICML'13]

Why?

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

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

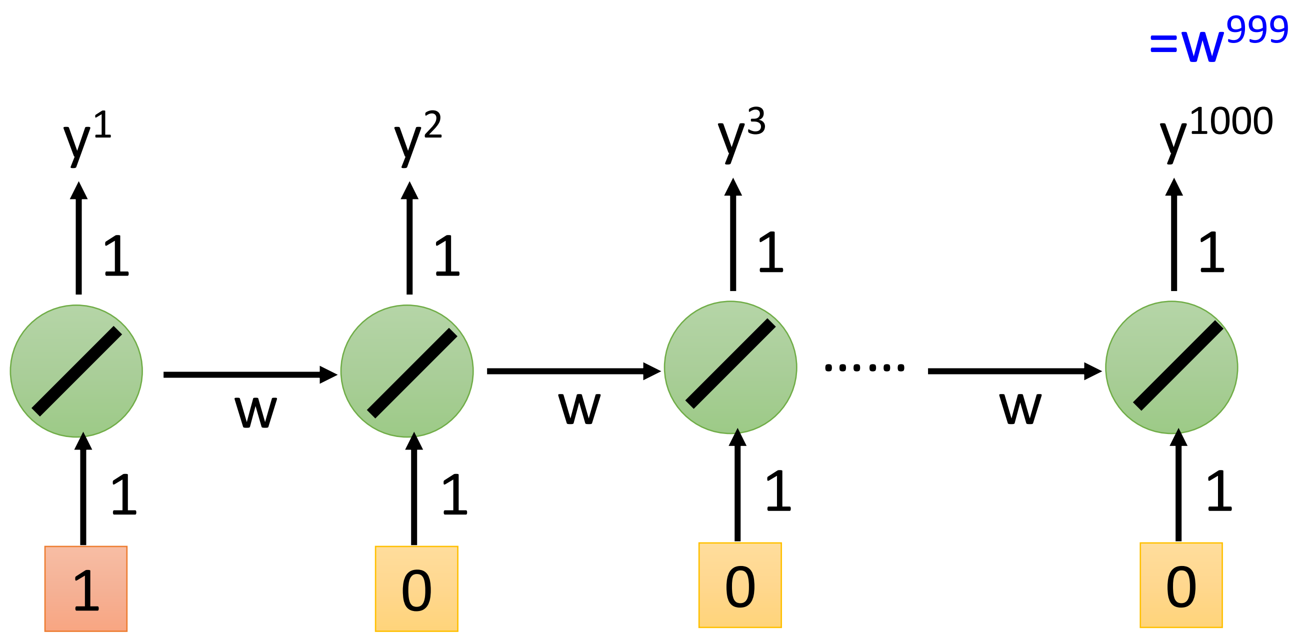
Large $\partial L / \partial w$ → Small Learning rate?

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

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

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

Toy Example



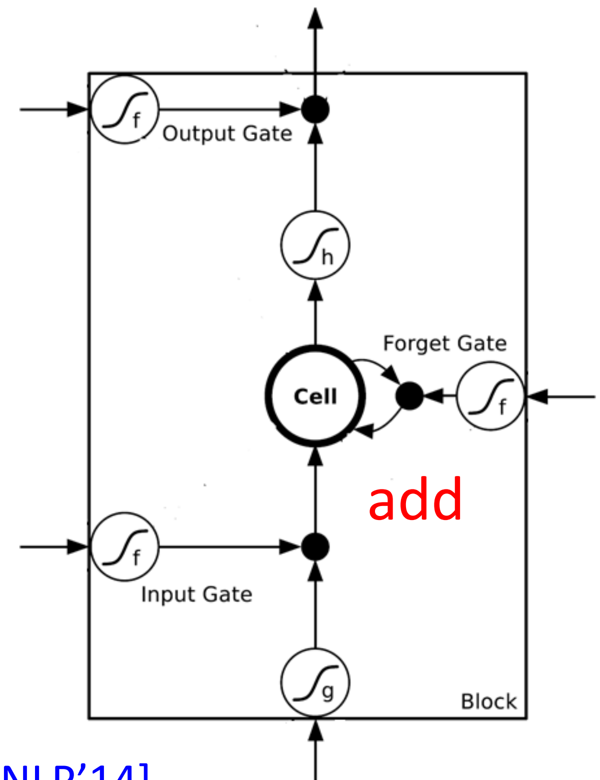
Helpful Techniques

- 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]