

Recurrent Neural Network

Dr Chang Xu

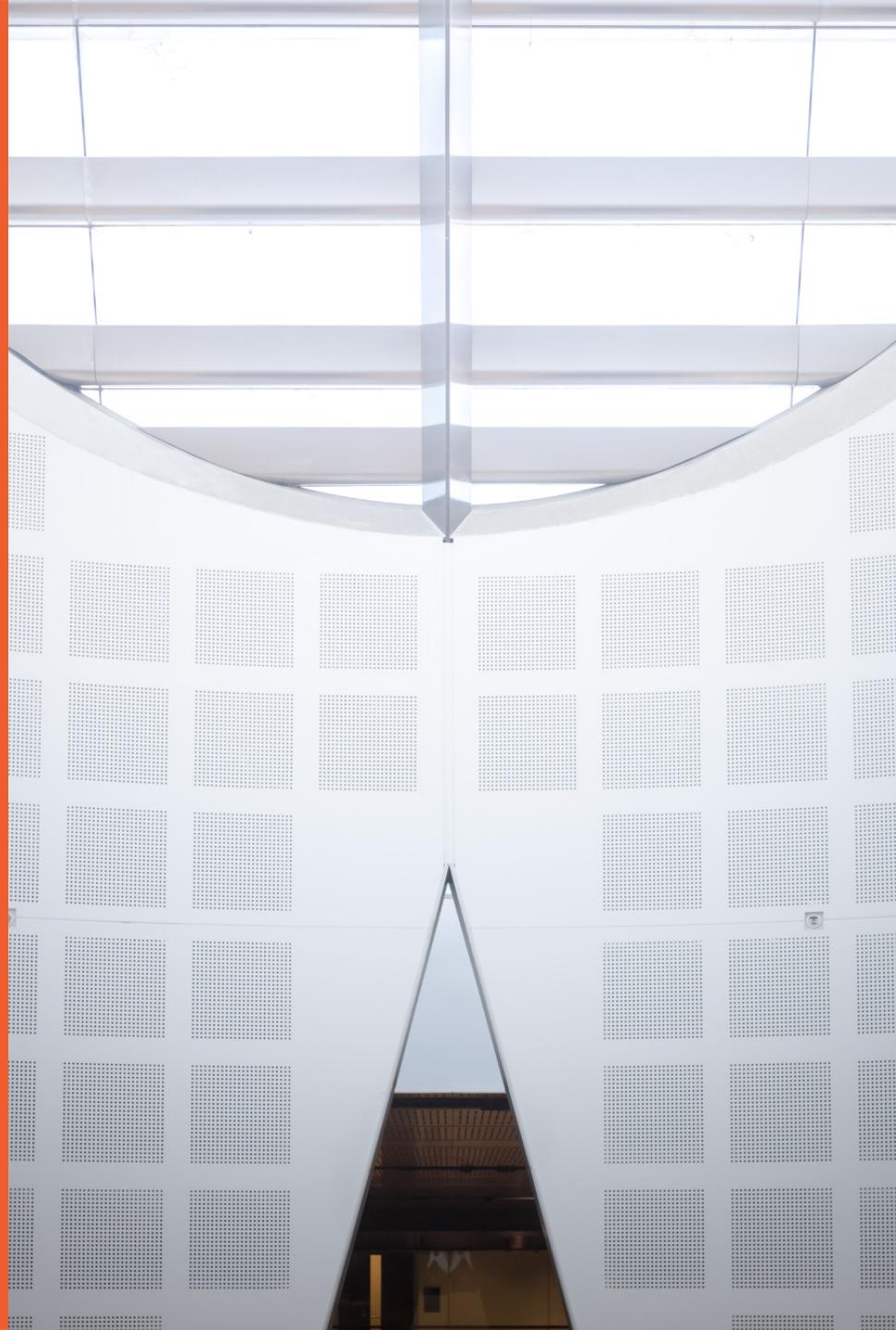
UBTECH Sydney AI Centre

Acknowledgements:

Dalu Guo



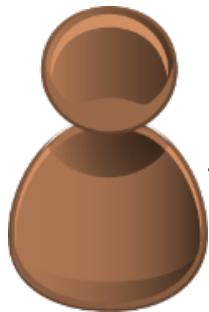
THE UNIVERSITY OF
SYDNEY



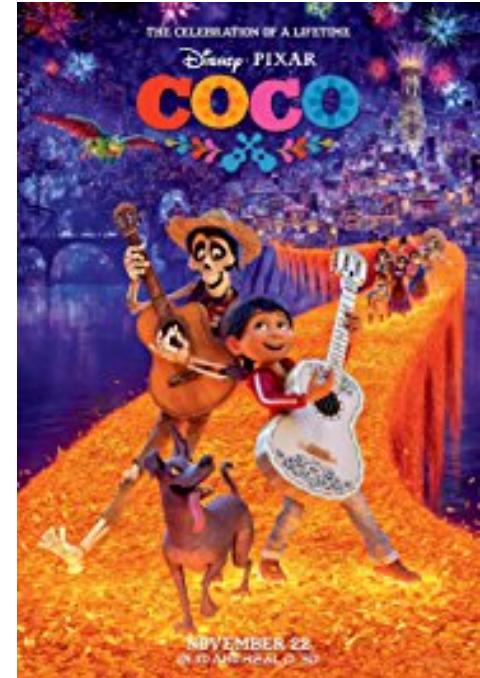
Quick Review

Example Application

□ Sentiment analysis



How about the COCO ?

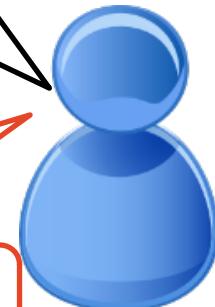


Complicated? Yes, and a little slow, too.

Rank 3/5



But the animation is as colorful as the story and emotions do eventually reach the desired level.



Example Application

Samsung Mobile @SamsungMobile

关注

There's a new star in town. And it's going to change the way you experience everything. Introducing the #GalaxyS9. Find out more.

翻译自英语

Introducing the
Galaxy S9+
The Camera. Reimagined.

上午10:55 - 2018年2月26日

Samsung Mobile US @SamsungMobileUS · 2月25日
Nothing wrong with sticking to what you know.
翻译自英语
4 4 19

Emma Driver @emmalb1988 · 2月25日
Absolutely. Have nothing **but** love for my Samsung. Spent 7 years as an iPhone user **but** now I'd never look back.
翻译自英语
7

Shubham Kumar @iamshubhmKr · 2月25日
回复 @SamsungMobile
Same old boring Samsung.
翻译自英语
1 1 5

Image credit to <https://twitter.com/SamsungMobile/status/967807667463958531>

Recurrent Neural Network

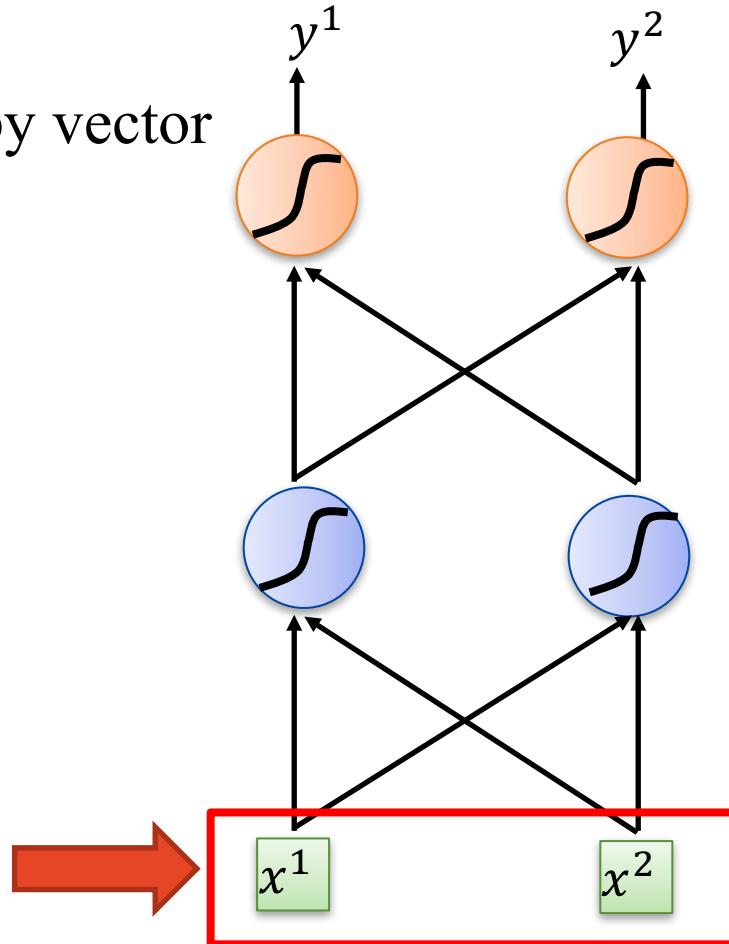
- Input

- Word sequence
 - Each word is represented by vector

- Methods

- One hot encoding
 - Word hash
 - Word embedding

Complicated



Recurrent Neural Network

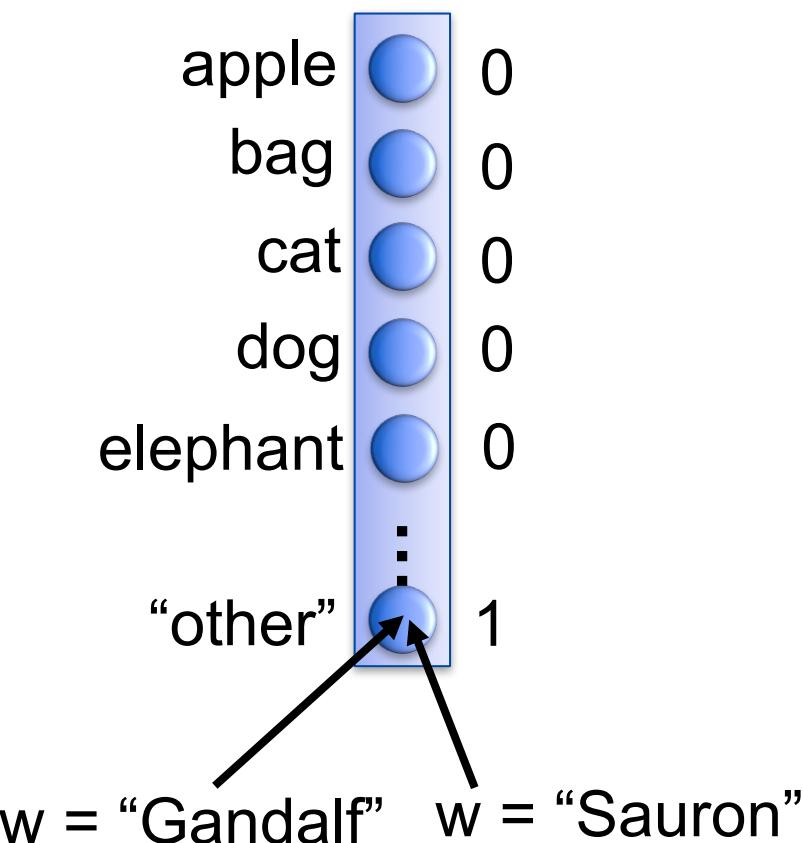
□ One hot encoding

- The length of vector is lexicon size
 - Each dimension corresponds to a word in the lexicon
 - The dimension for the word is 1, and others are 0
- lexicon = {apple, bag, cat, dog, elephant}

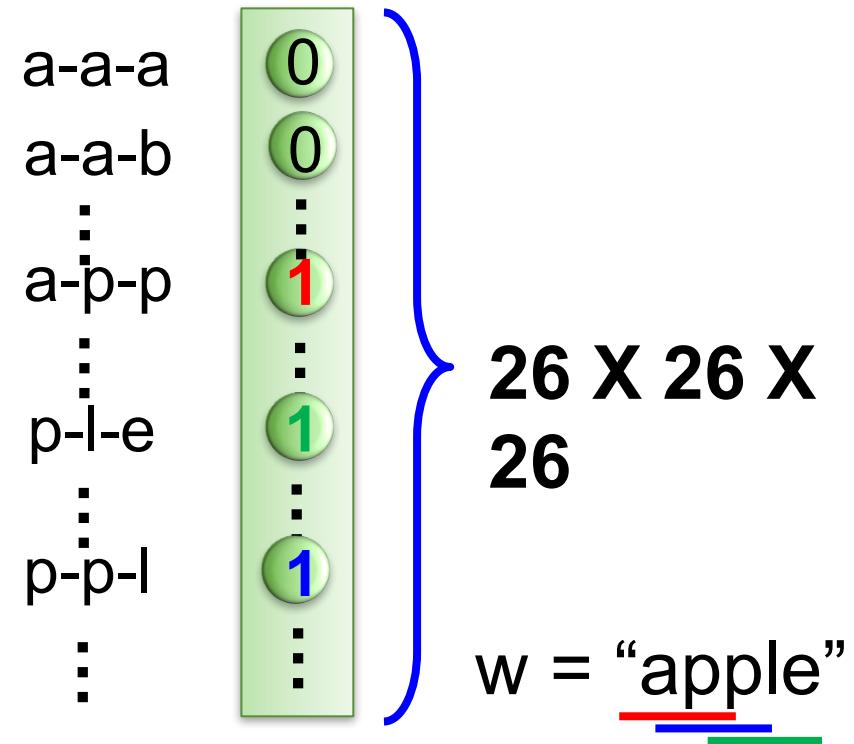
Word	D_1	D_2	D_3	D_4	D_5
Apple	1	0	0	0	0
Bag	0	1	0	0	0
Cat	0	0	1	0	0
Dog	0	0	0	1	0
Elephant	0	0	0	0	1

Recurrent Neural Network

Dimension for “Other”



Word hashing



Recurrent Neural Network

□ Output

□ Probability distribution that the sentence belongs to negative or positive.

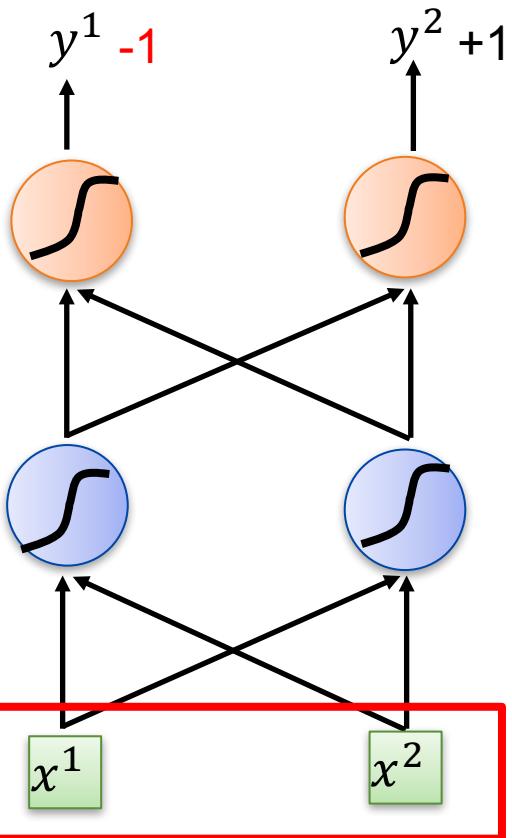
Complicated? Yes,
and a little slow, too.

$$\begin{aligned}y^1 &= 0.9 \\y^2 &= 0.1\end{aligned}$$

But the animation is as
colorful as the story and
emotions do eventually reach
the desired level.

$$\begin{aligned}y^1 &= 0.3 \\y^2 &= 0.7\end{aligned}$$

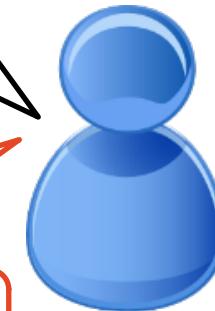
Complicated? Yes,
and a little slow, too.



Recurrent Neural Network

Complicated? Yes, and a little slow, too.

But the animation is as colorful as the story and emotions do eventually reach the desired level.



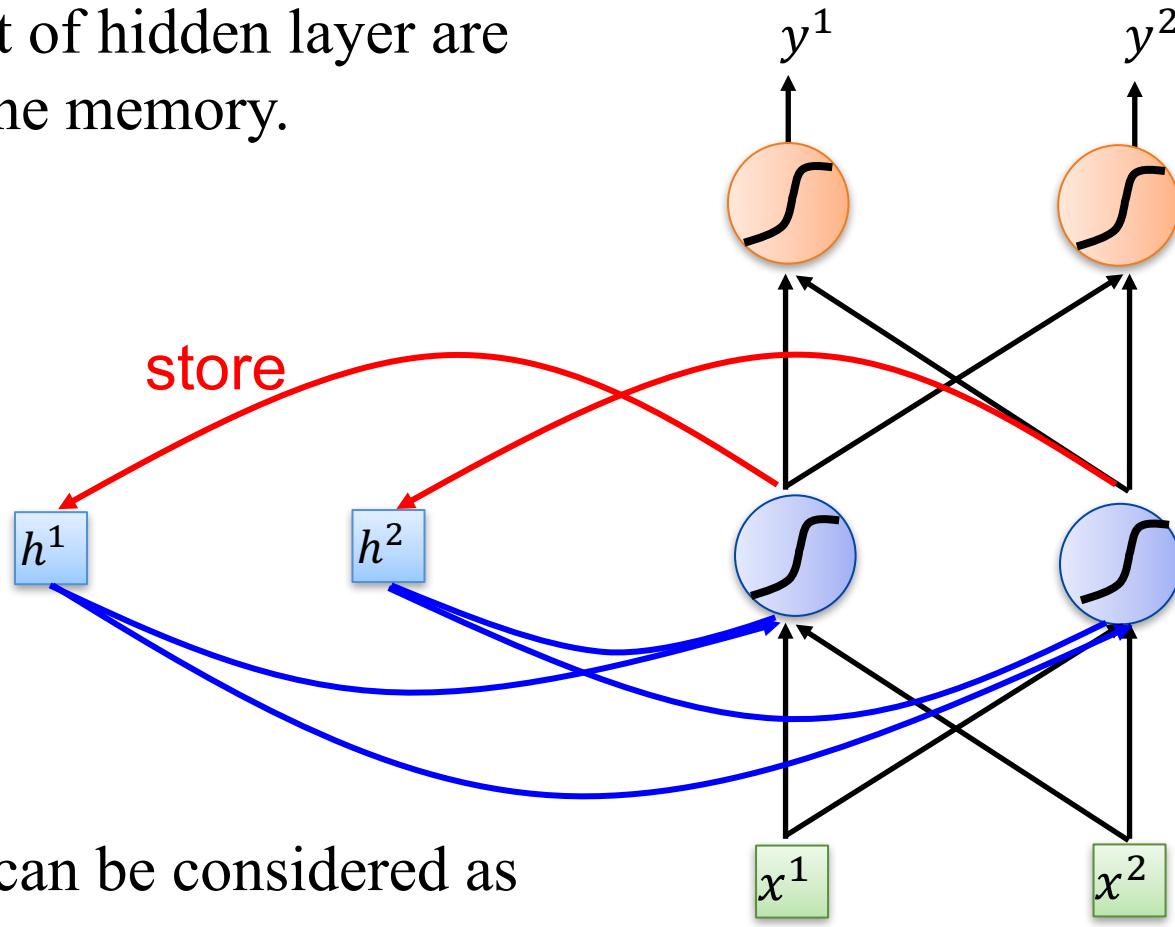
Negative OR Positive



Need
Memory

Recurrent Neural Network

The output of hidden layer are stored in the memory.

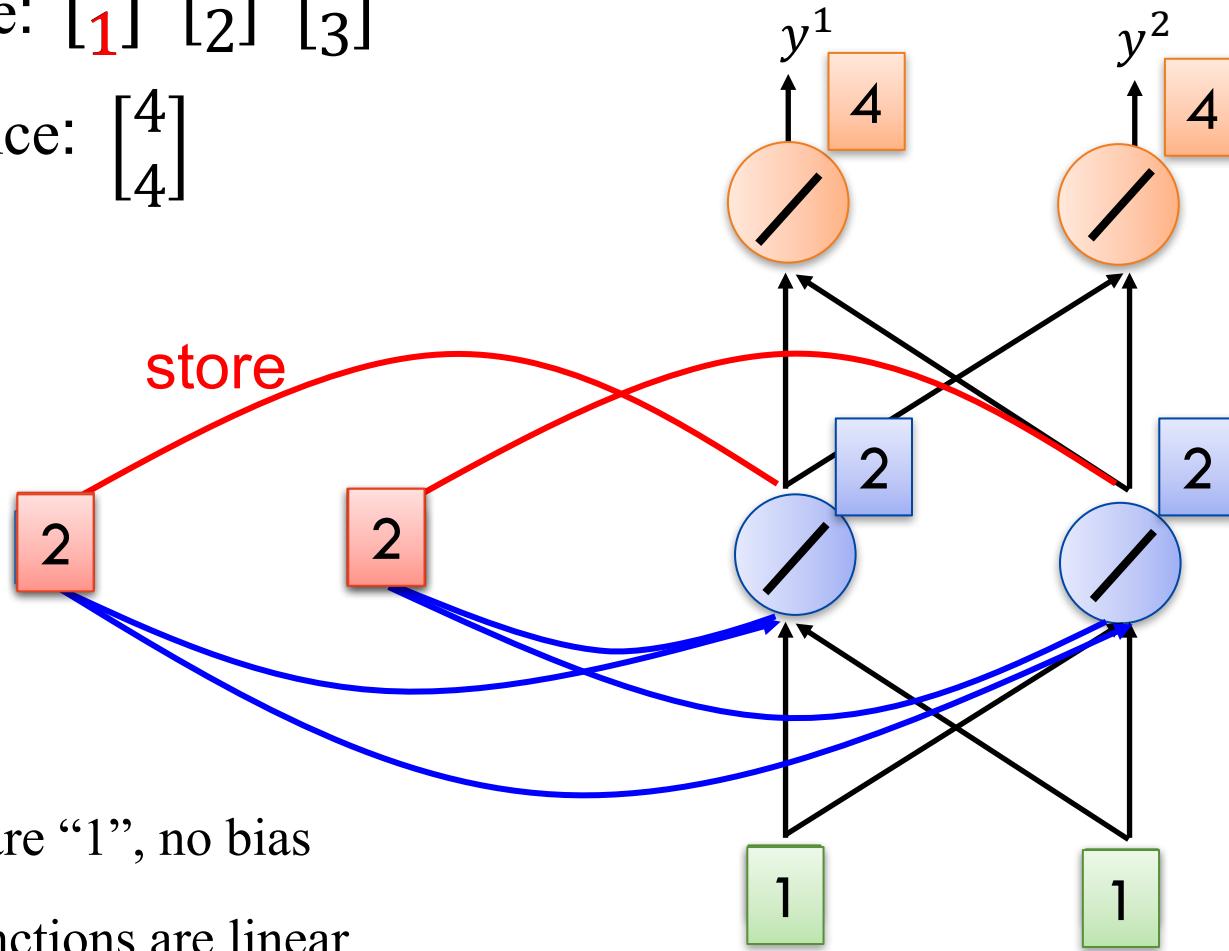


Memory can be considered as another input.

Recurrent Neural Network

Input sequence: $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

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



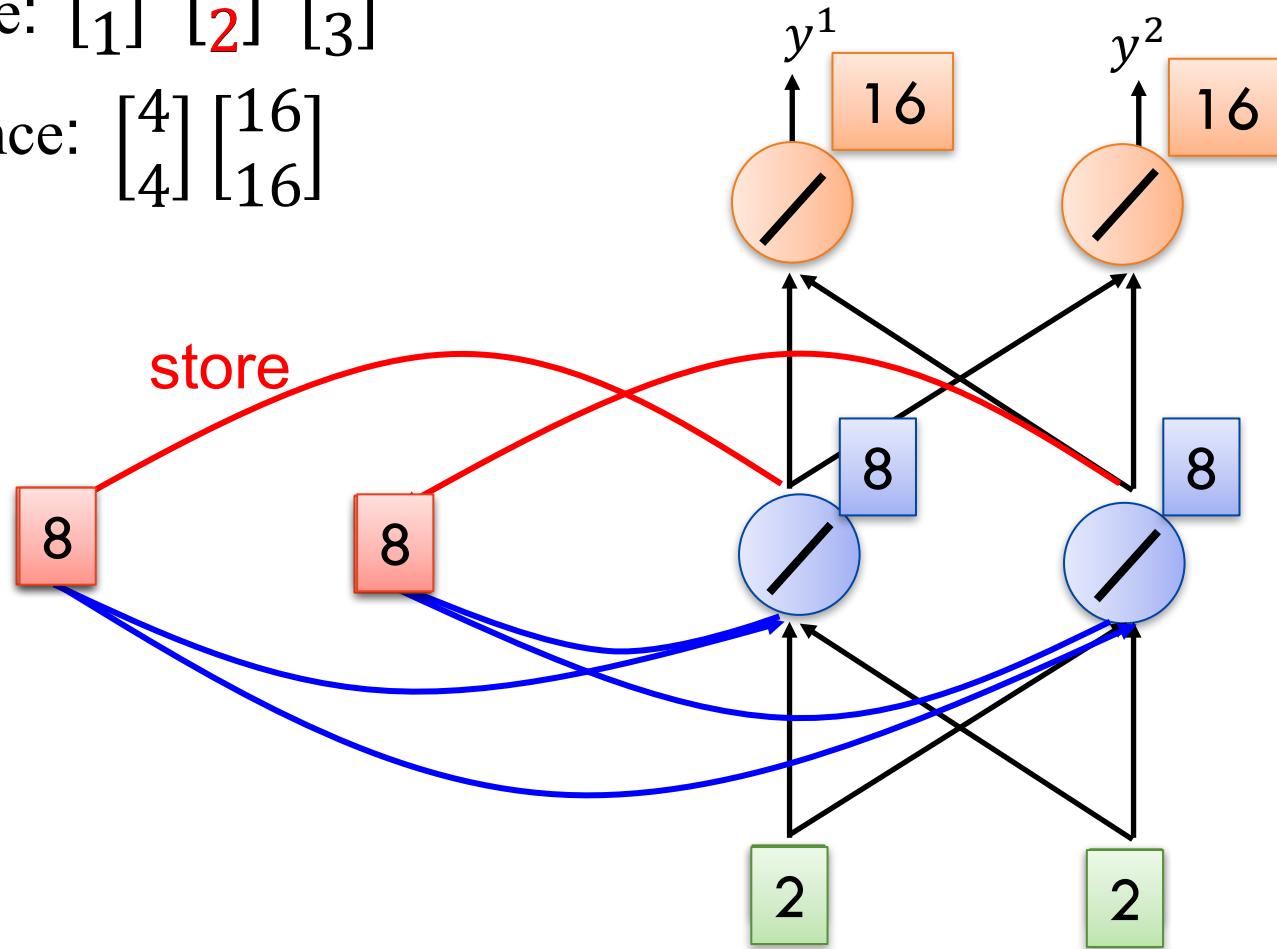
All the weights are “1”, no bias

All activation functions are linear

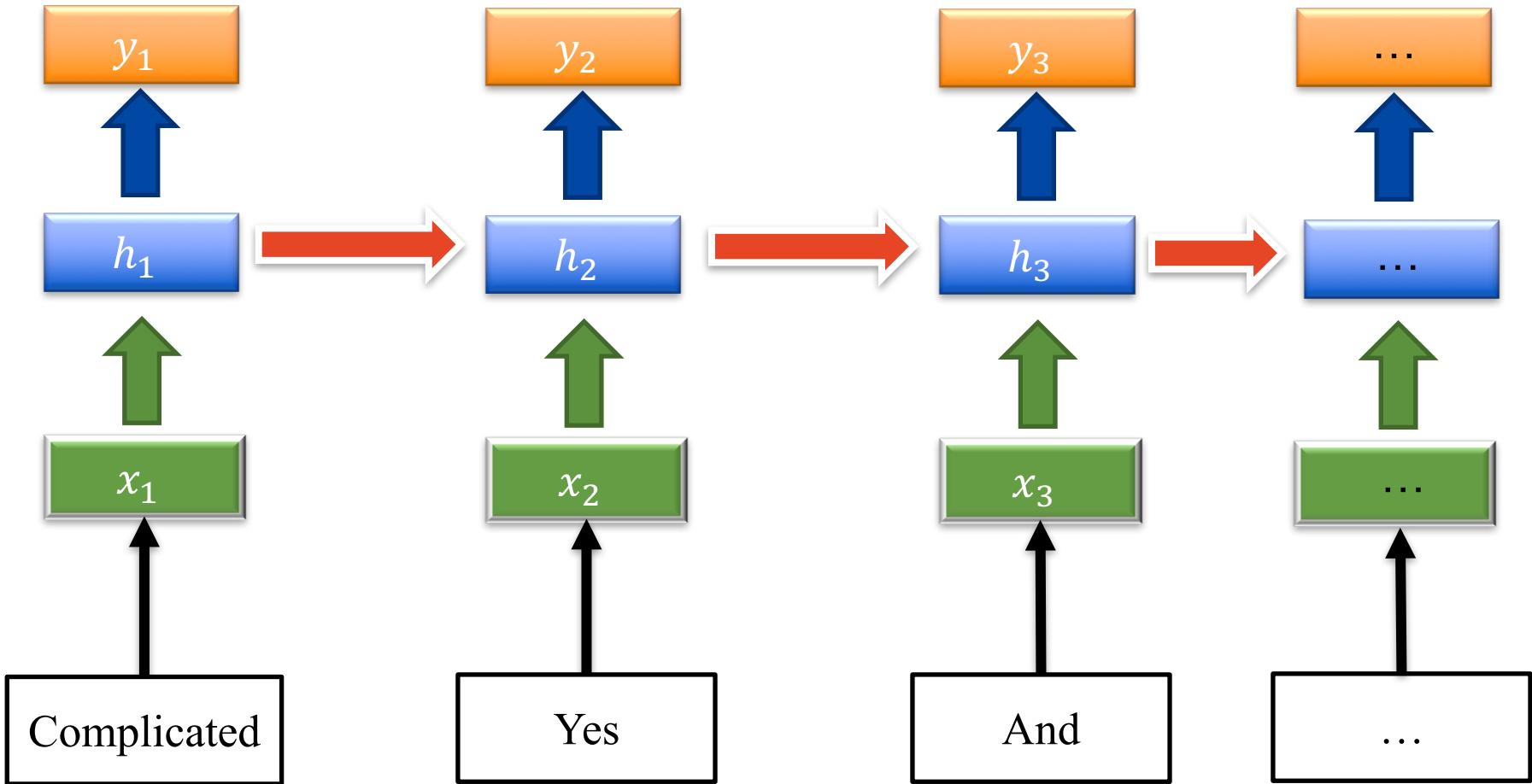
Recurrent Neural Network

Input sequence: $\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix}$

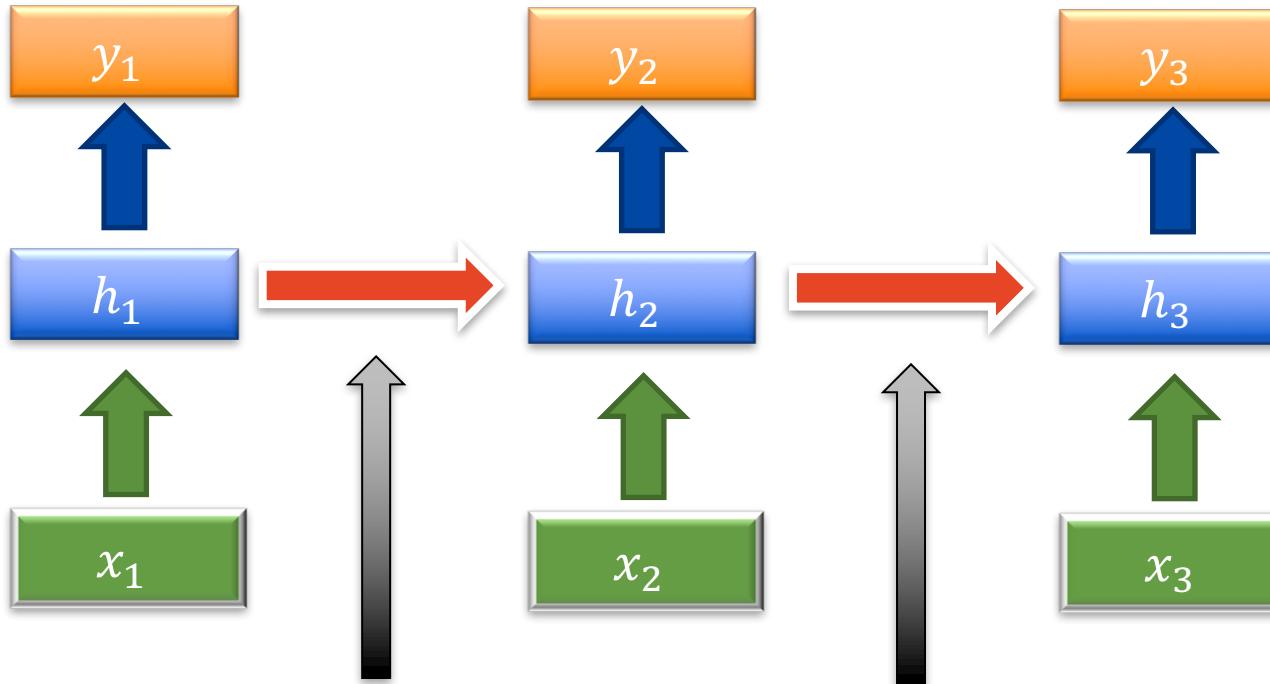
Output sequence: $\begin{bmatrix} 4 \\ 4 \end{bmatrix} \begin{bmatrix} 16 \\ 16 \end{bmatrix}$



Recurrent Neural Network



Recurrent Neural Network



Memory can be considered as another input.

Recurrent Neural Network

□ Formally

□ x_t is the input

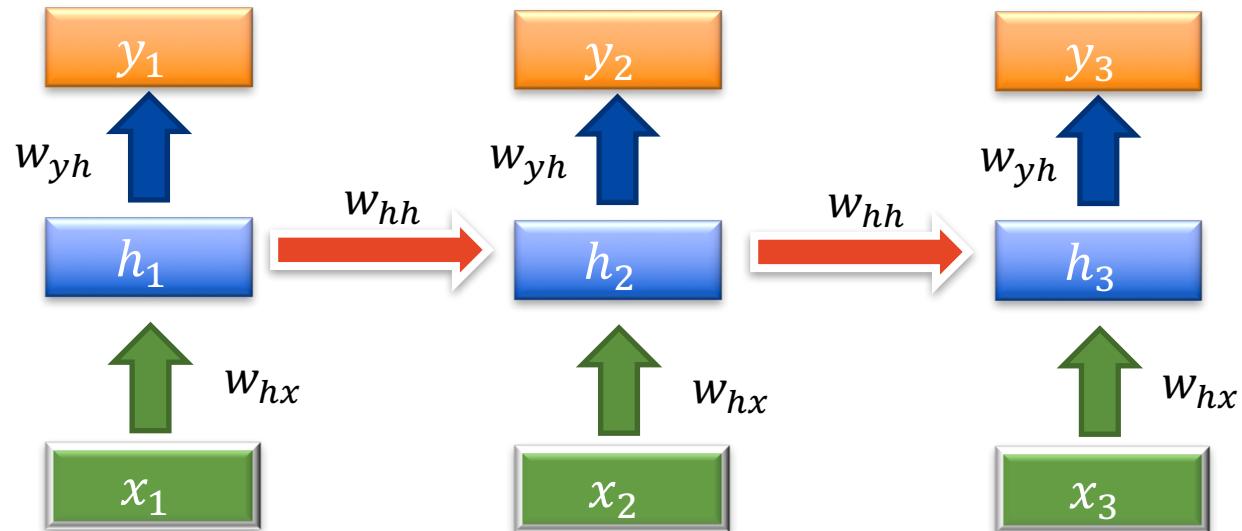
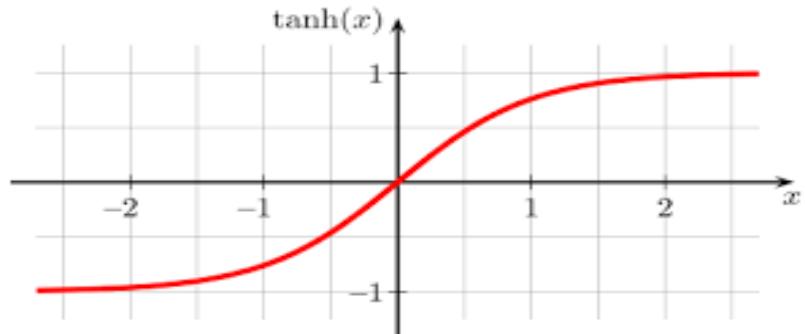
□ h_t is the hidden state

□ y_t is the output

□ $h_0 = \vec{0}$

□ $h_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$

□ $y_t = w_{yh}h_t$



Recurrent Neural Network

□ Vanishing gradient problem

$$\square h_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$$

$$\square y_t = w_{yh}h_t$$

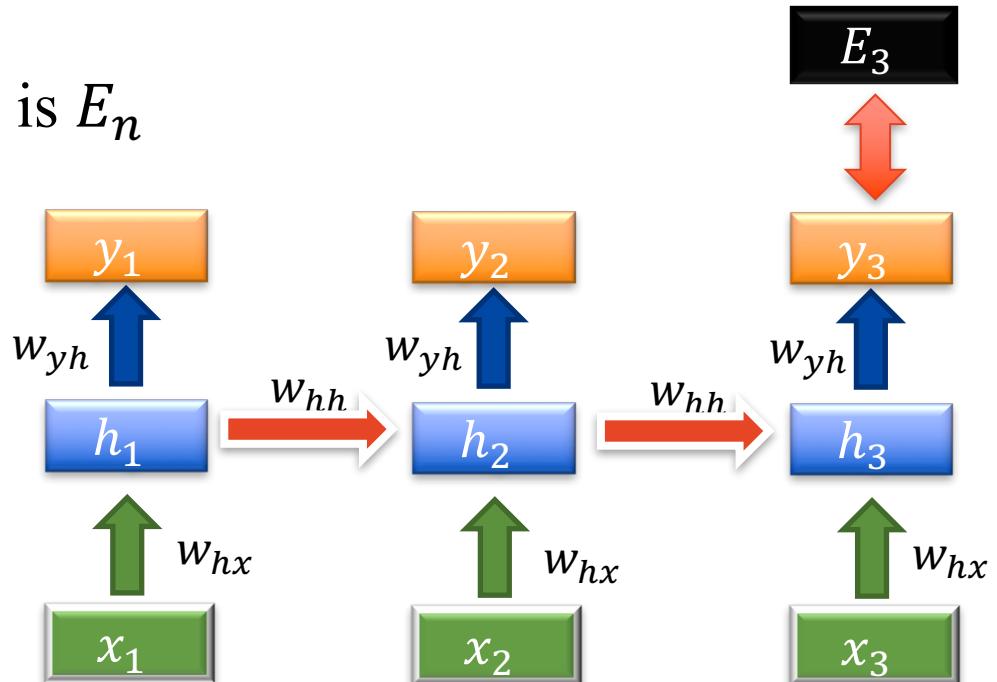
□ For every step, its loss is E_n

$$\frac{\partial E_3}{\partial w_{hh}} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial w_{hh}}$$

$$h_3 = \tanh(w_{hh}h_2 + w_{hx}x_3 + b_h)$$

$$\frac{\partial h_3}{\partial w_{hh}} = (\tanh)' \left(h_2 + w_{hh} \frac{\partial h_2}{\partial w_{hh}} \right)$$

$$\frac{\partial h_2}{\partial w_{hh}} = (\tanh)' \left(h_1 + w_{hh} \frac{\partial h_1}{\partial w_{hh}} \right)$$



$$(\tanh)'w_{hh} (\tanh)'w_{hh} (\tanh)'w_{hh} \dots$$

Recurrent Neural Network

□ Vanishing gradient problem

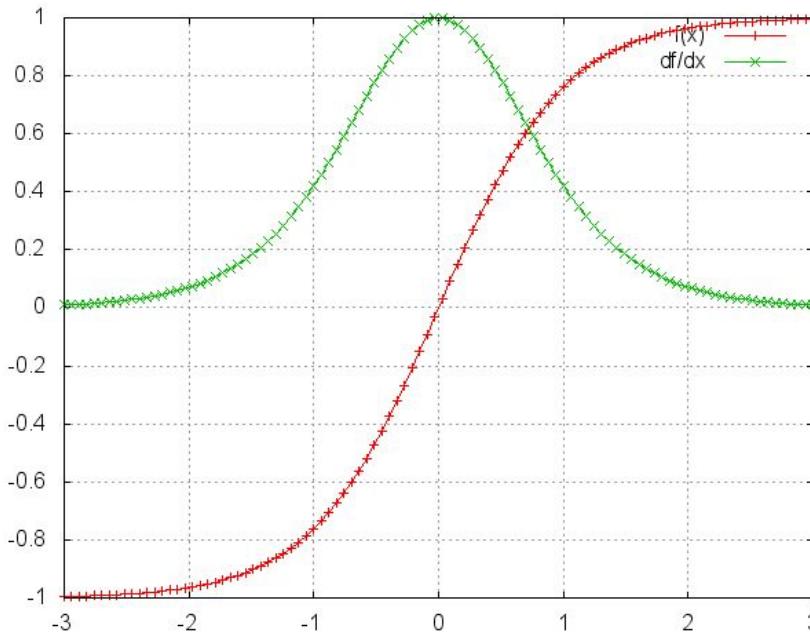
$$\square h_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$$

$$\square y_t = w_{yh}h_t$$

□ For every step, its loss is E_n

$$\frac{\partial E_3}{\partial w_{hh}} = \frac{\partial E_3}{\partial y_3} \frac{\partial y_3}{\partial h_3} \frac{\partial h_3}{\partial w_{hh}}$$

$$(\tanh)'w_{hh}(\tanh)'w_{hh}(\tanh)'w_{hh} \dots$$



$$(\tanh)' \leq 1$$

If $0 < w_{hh} < 1$, **Vanishing Gradients**

$$(\tanh)'w_{hh}(\tanh)'w_{hh}(\tanh)'w_{hh} \dots \rightarrow 0$$

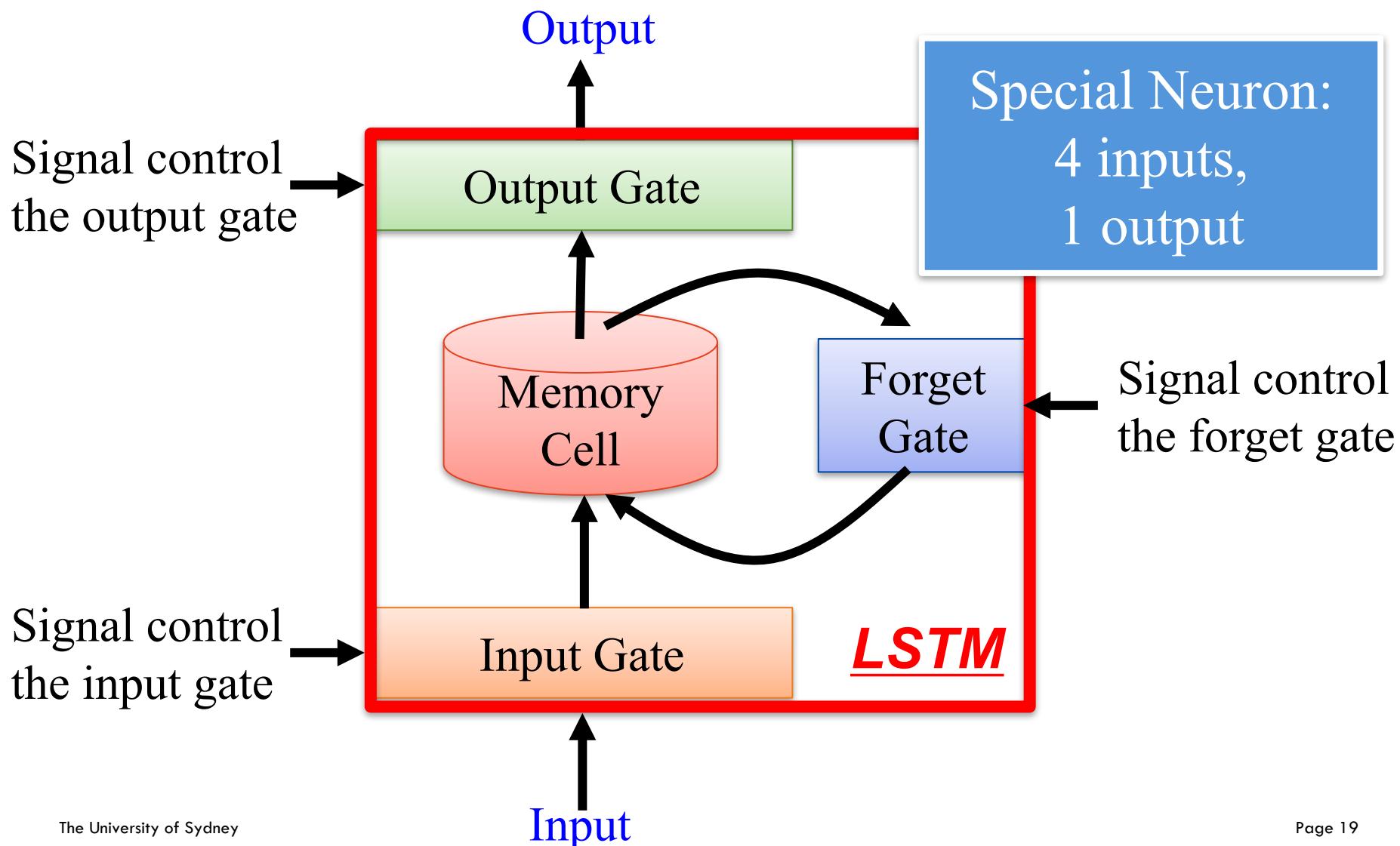
If w_{hh} is larger, **Exploding Gradients**

$$(\tanh)'w_{hh}(\tanh)'w_{hh}(\tanh)'w_{hh} \dots \rightarrow \infty$$

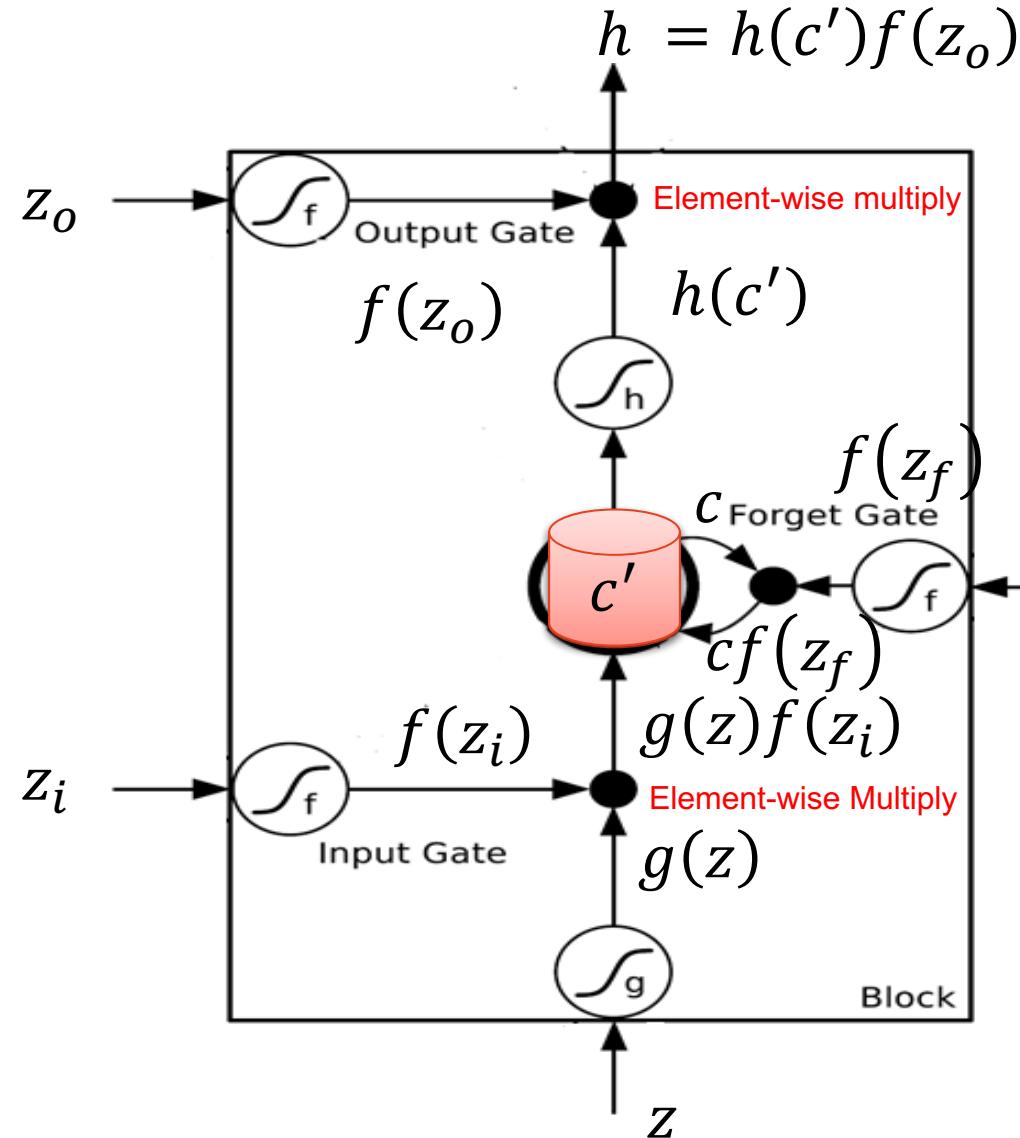
$$\begin{aligned}0.9^{1000} &\approx 0 \\1.01^{1000} &\approx 21000\end{aligned}$$

Long Short-Term Memory

Long Short-Term Memory

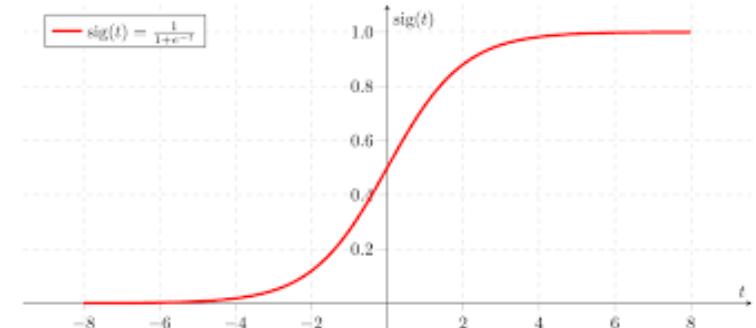


Long Short-Term Memory



f is usually a sigmoid function
Value From 0 to 1
Mimic open and close gate

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



Long Short-Term Memory

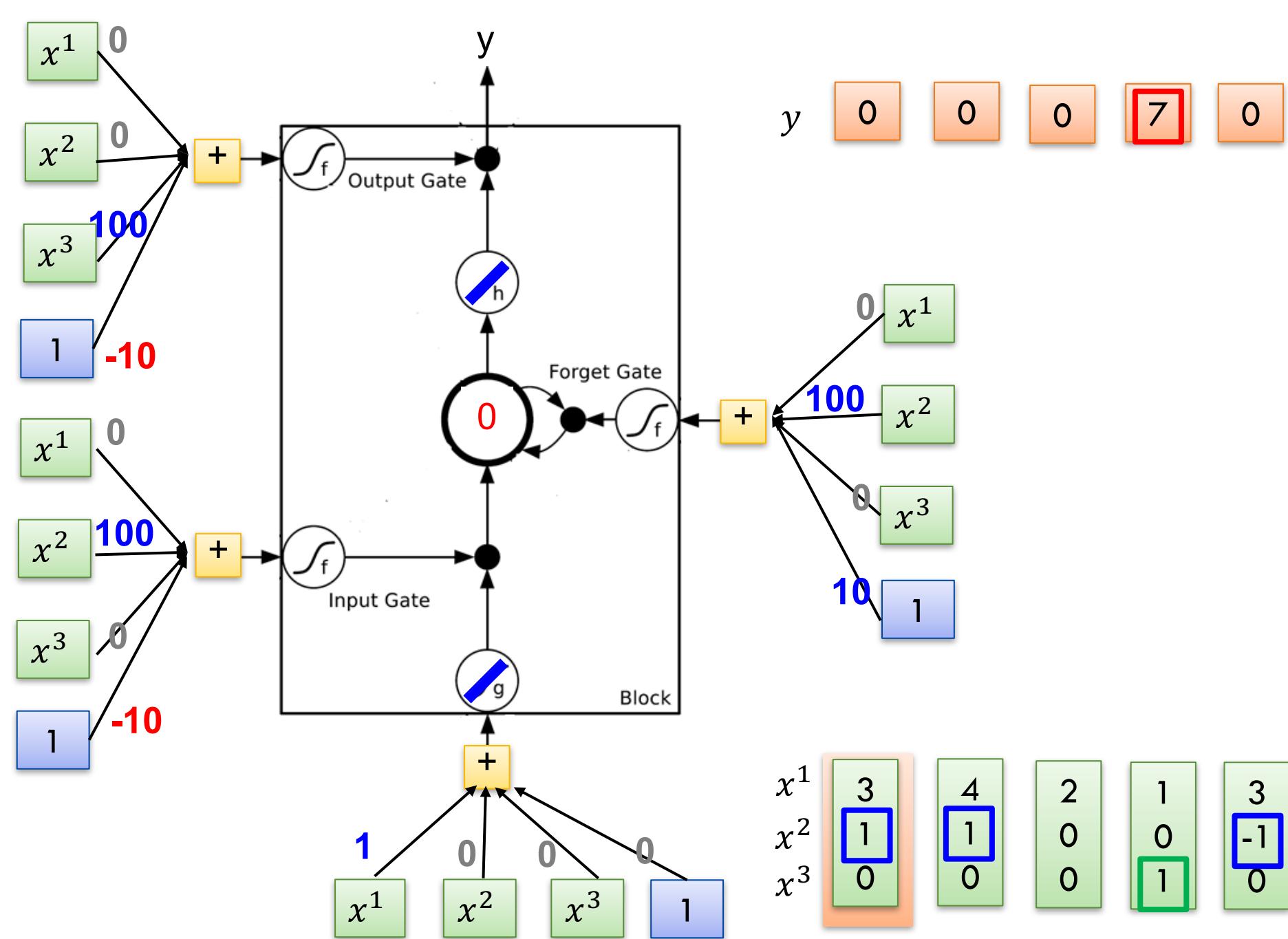
C	0	0	3	3	7	7	7	0	6
x^1	1	3	2	4	2	1	3	6	1
x^2	0	1	0	1	0	0	-1	1	0
x^3	0	0	0	0	0	1	0	0	1

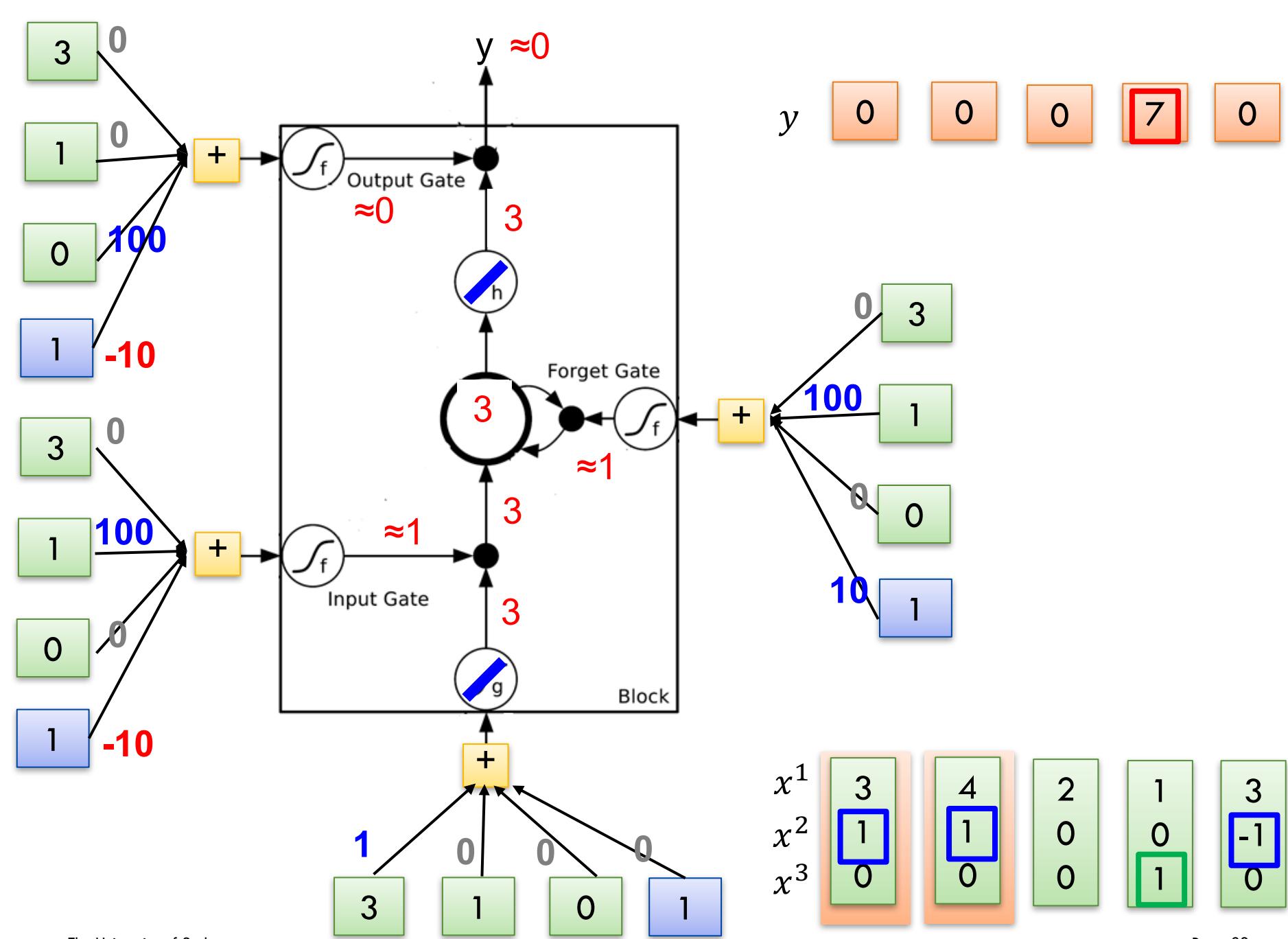
y	0	0	0	0	0	7	0	0	6
-----	---	---	---	---	---	---	---	---	---

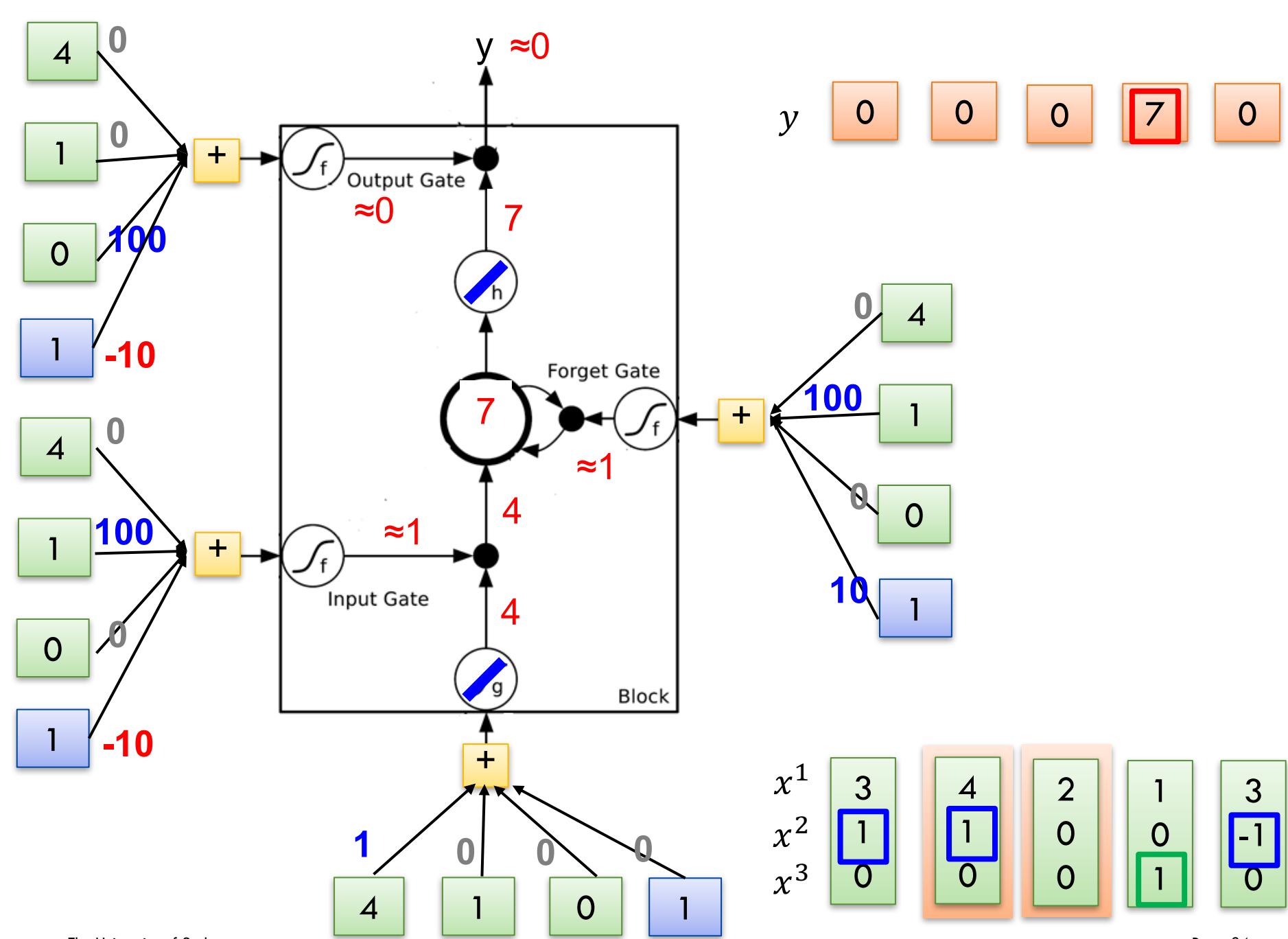
When $x^2 = 1$, add the numbers of x_1 into the memory

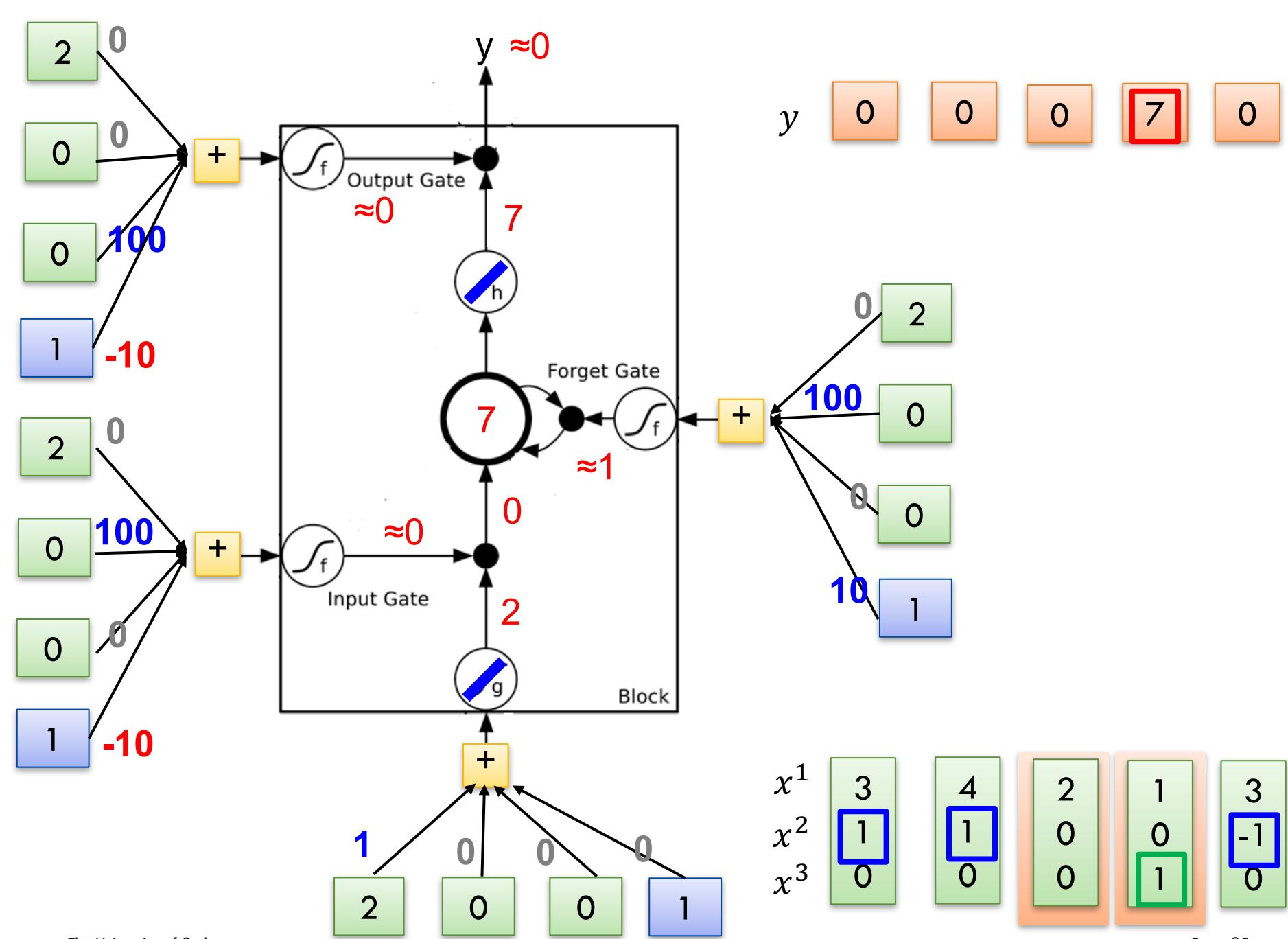
When $x^2 = -1$, reset the memory

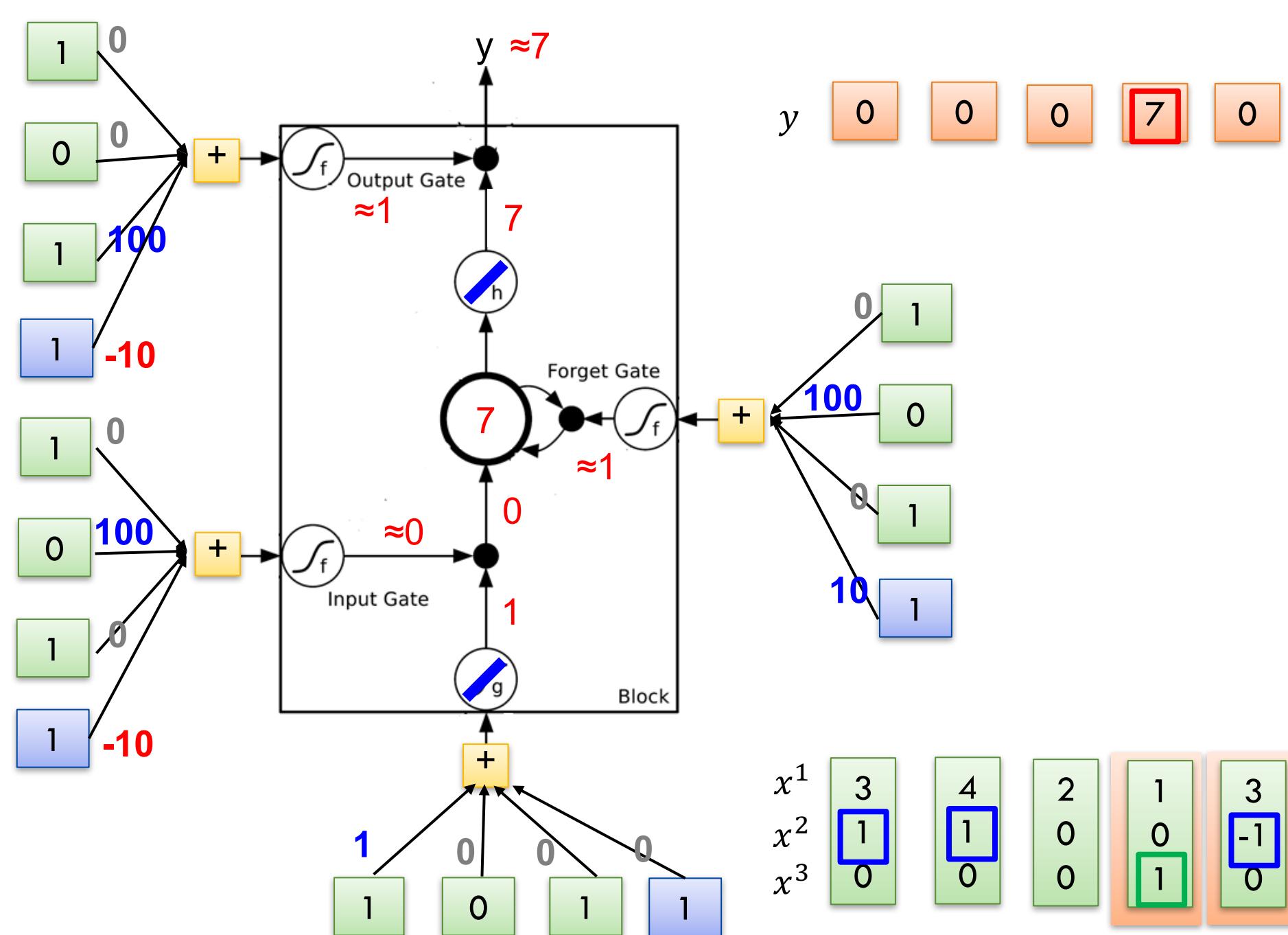
When $x^3 = 1$, output the number in the memory.

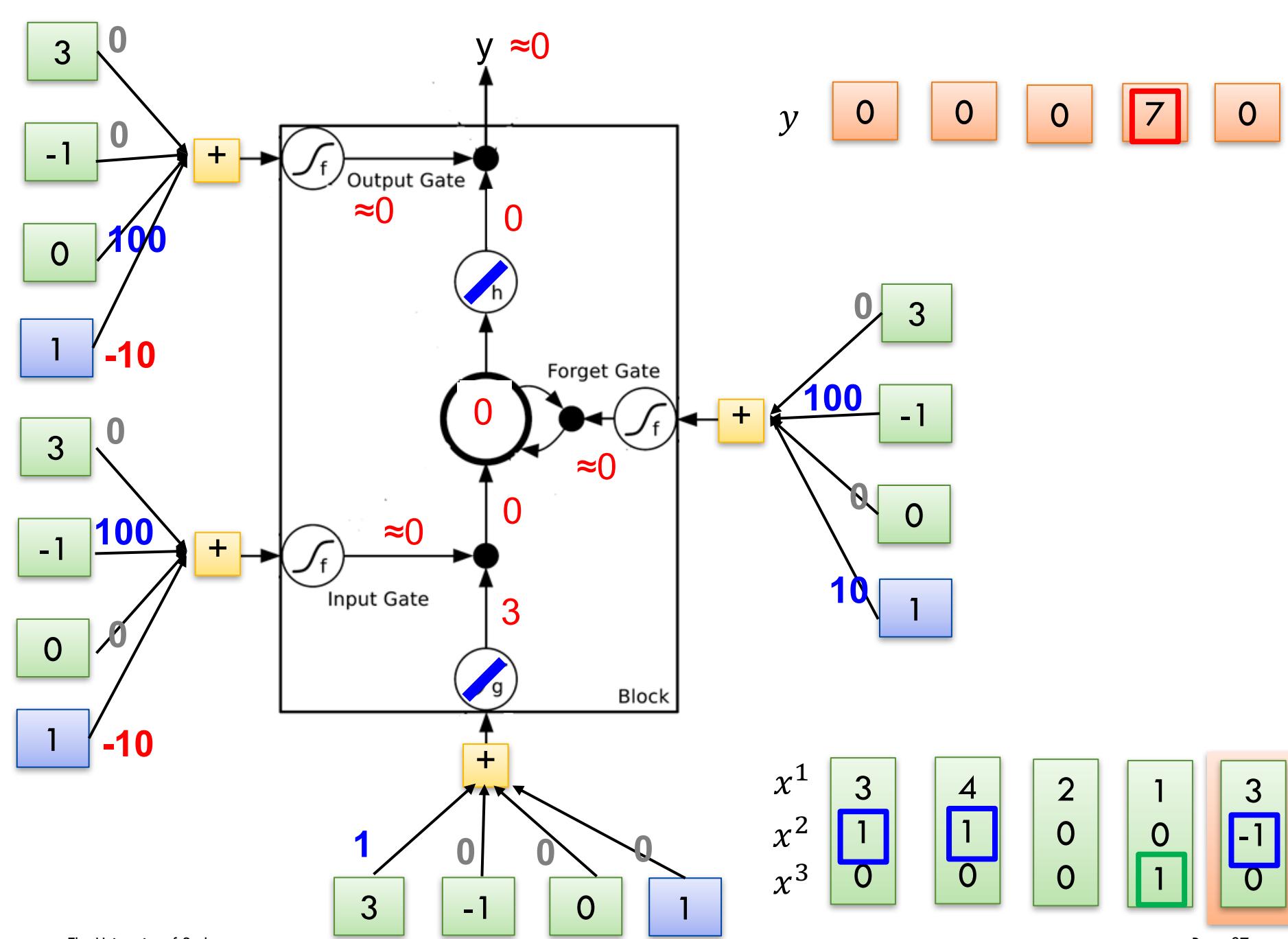




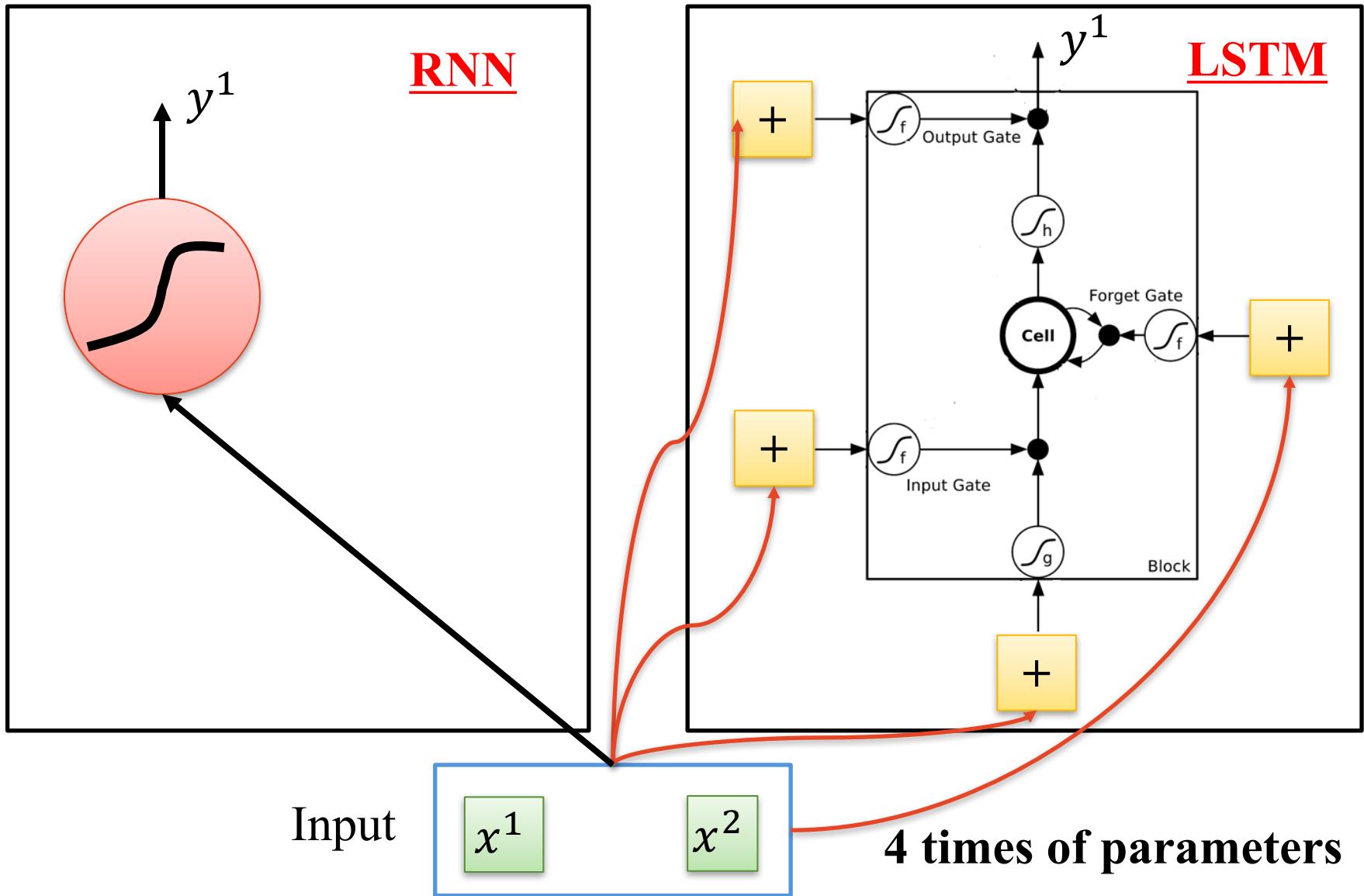








Long Short-Term Memory



Long Short-Term Memory

□ Forget gate

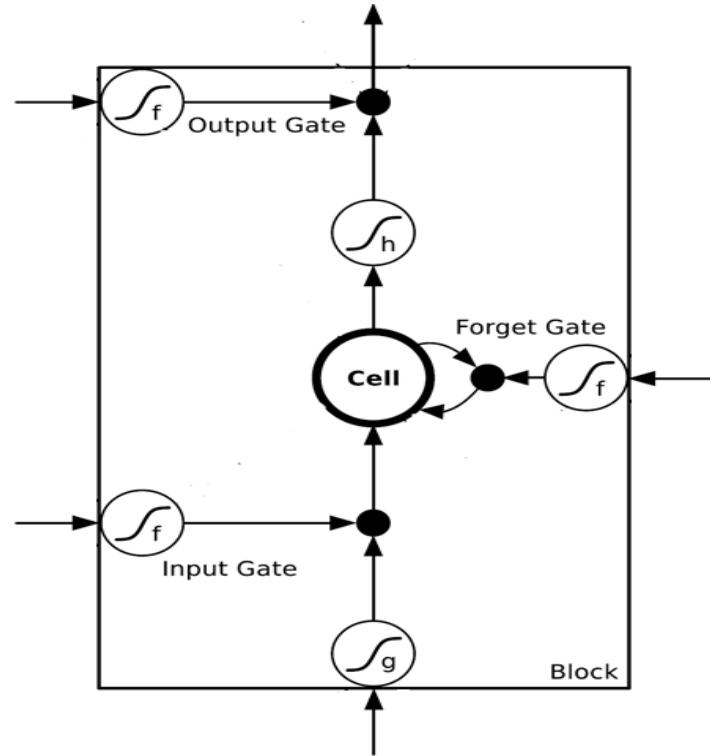
- $f_t = \sigma(w_{fh}h_{t-1} + w_{fx}x_t + b_f)$

□ Input gate

- $i_t = \sigma(w_{ih}h_{t-1} + w_{ix}x_t + b_i)$

□ Output gate

- $o_t = \sigma(w_{oh}h_{t-1} + w_{ox}x_t + b_o)$



Long Short-Term Memory

□ Candidate cell state

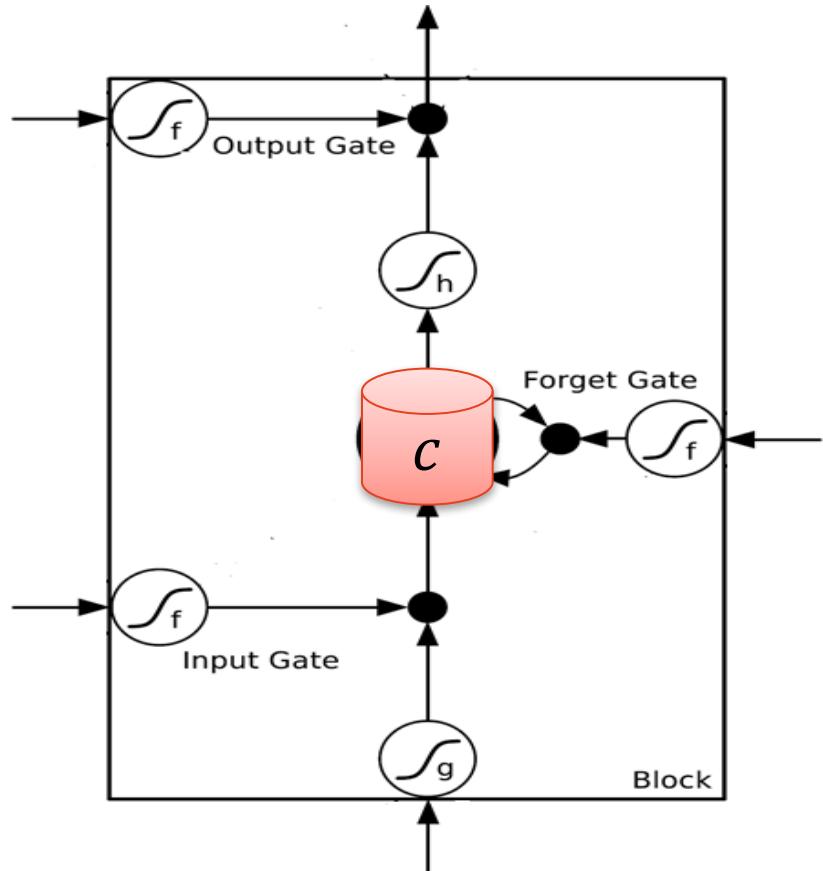
- $\tilde{c}_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$

□ Update cell state

- $c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$

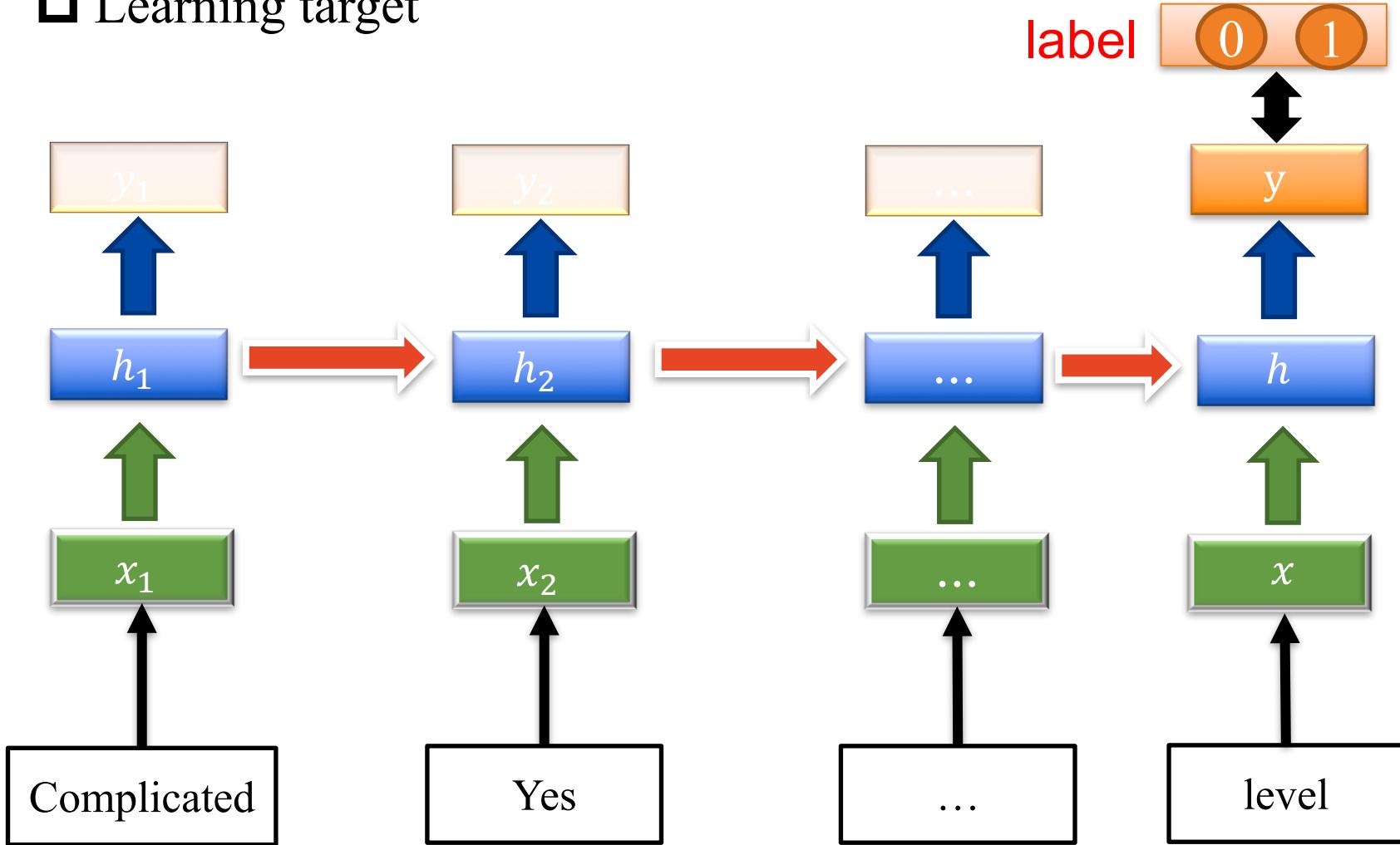
□ Output state

- $h_t = o_t * \tanh(c_t)$



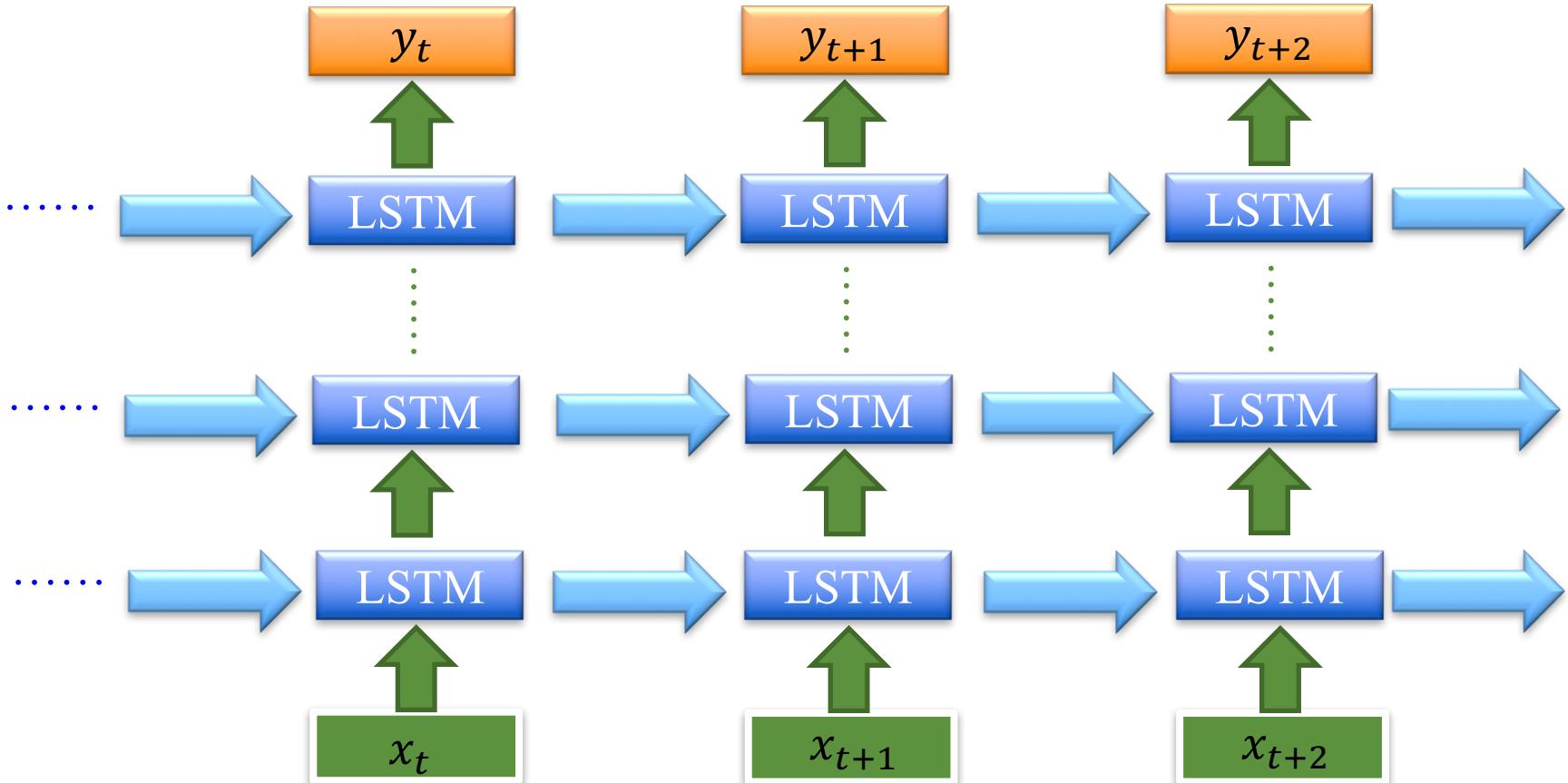
Long Short-Term Memory

□ Learning target



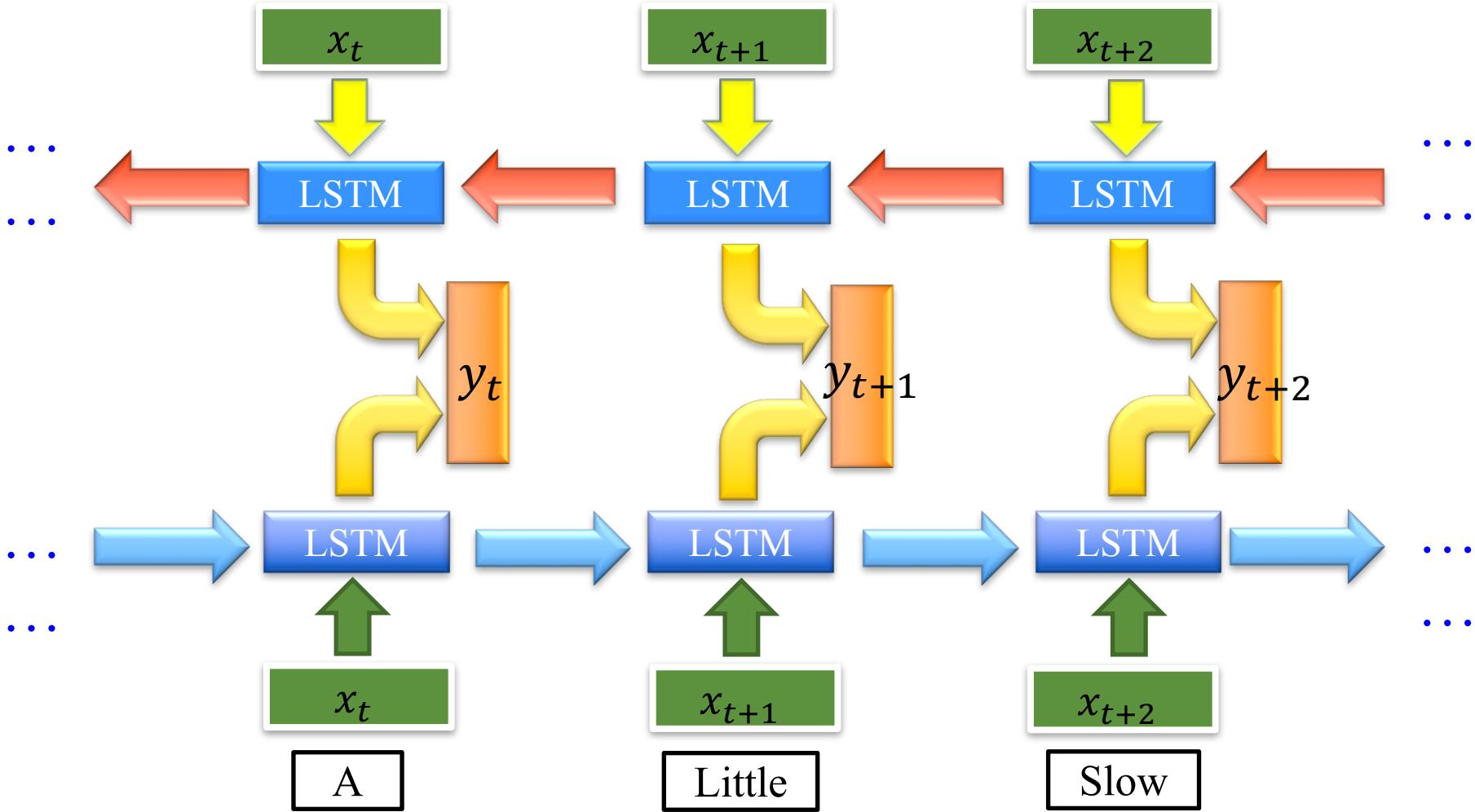
Long Short-Term Memory

□ Stacked LSTM

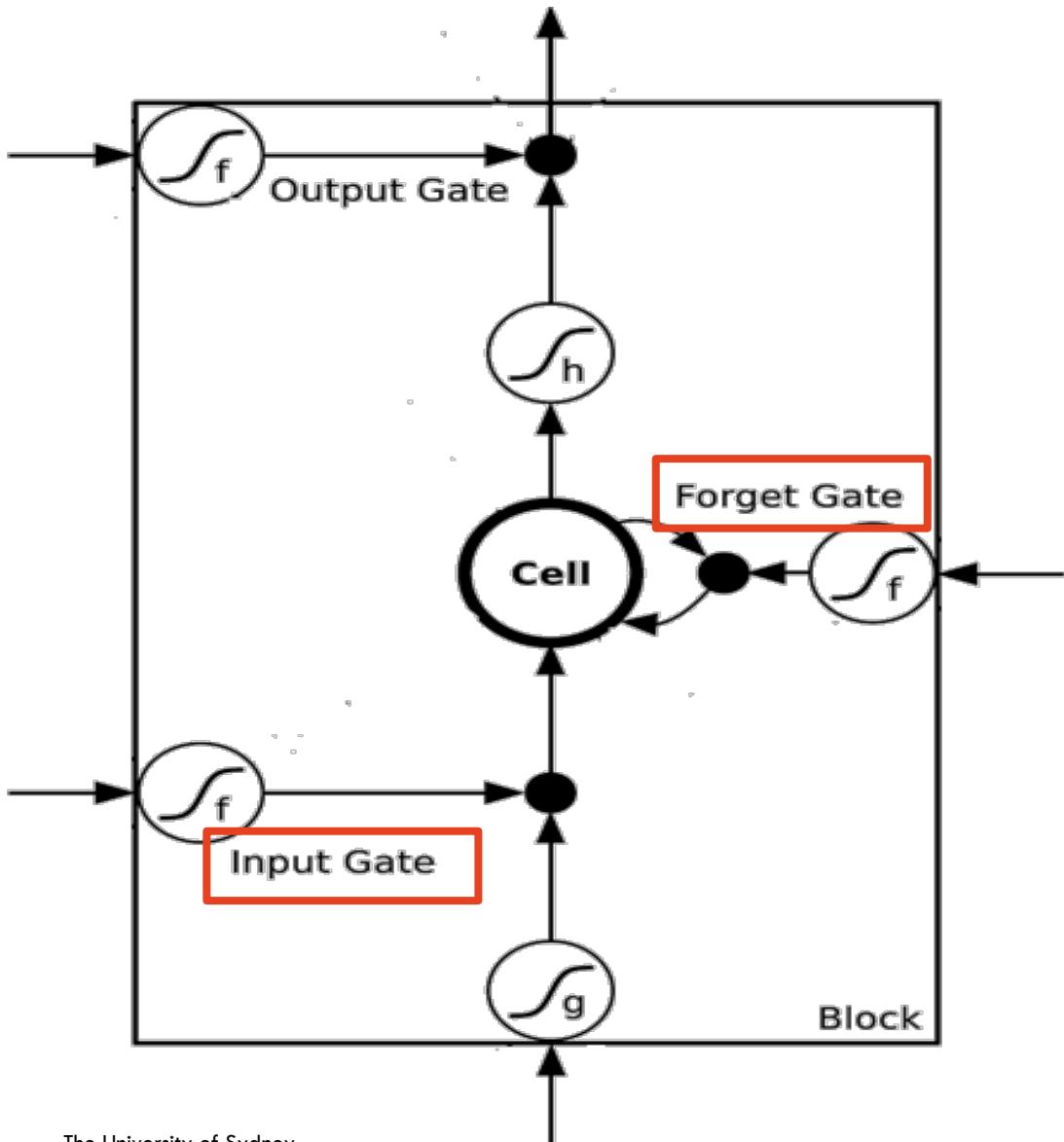


Long Short-Term Memory

□ Bidirectional RNN



LSTM V.S. RNN



Prevent vanishing gradient

□ RNN

$$\bullet h_t = \tanh(w_{hh}h_{t-1} + w_{hx}x_t + b_h)$$

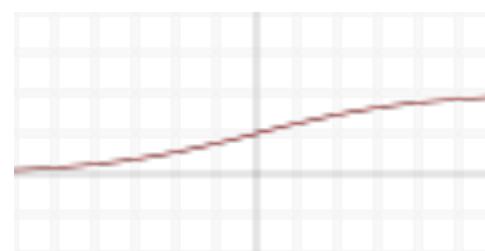
□ LSTM

$$\bullet c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

Back propagation from c_t to c_{t-1} only involves elementwise multiplication with f_t , no matrix multiplication with w_{hh} or tanh.



$$f(x) = \tanh(x)$$
$$f'(x) = 1 - f(x)^2 \leq 1$$



$$f(x) = \sigma(x)$$
$$f'(x) = f(x)(1 - f(x)) \leq 0.25$$