CSCE 636 Neural Networks (Deep Learning)

Lecture 8: Deep Learning for Text and Sequences

Anxiao (Andrew) Jiang

Based on the interesting lecture of Prof. Hung-yi Lee "Recurrent Neural Network"
https://www.youtube.com/watch?v=xCGidAeyS4M&list=PLJV_el3uVTsPy9oCRY30oBPNLCo89yu49&index=30

Recurrent Neural Network (RNN)

Example Application

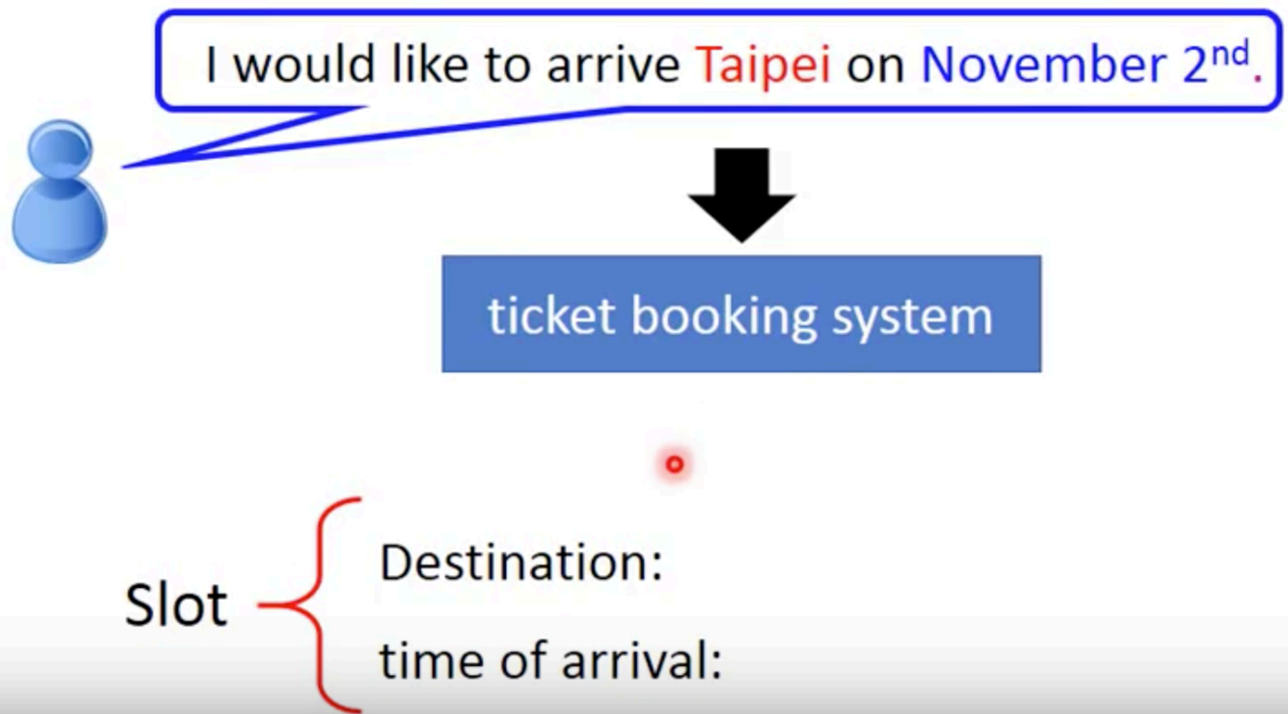
- Slot Filling

Example Application

- Slot Filling

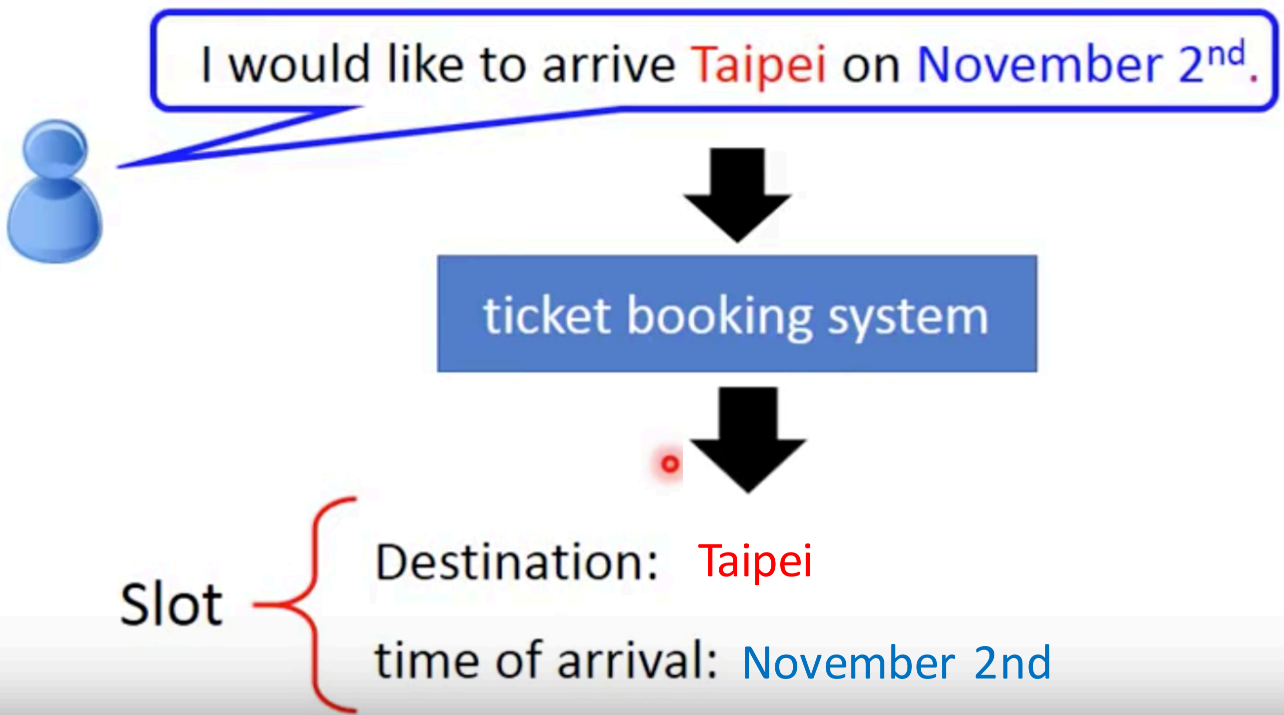
Example Application

- Slot Filling



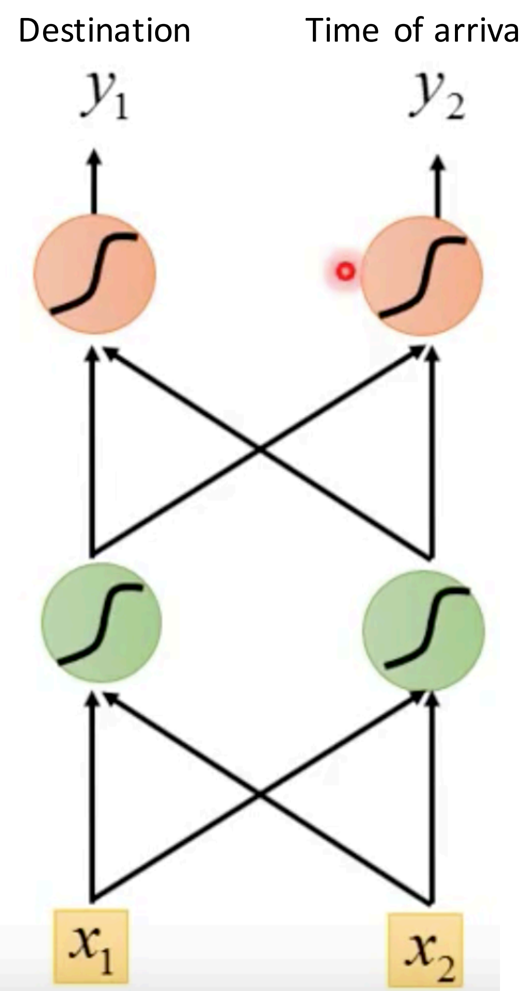
Example Application

- Slot Filling



Example Application

Solving slot filling by Feedforward network?

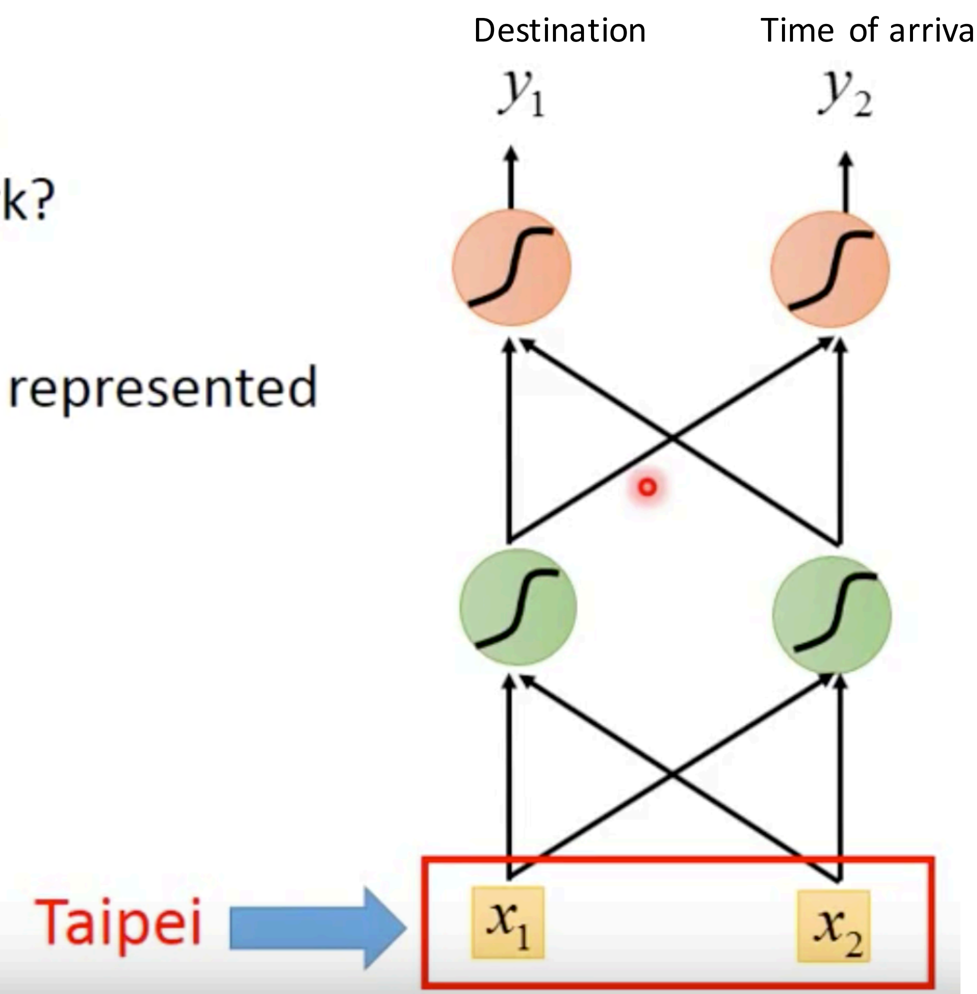


Example Application

Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)



1-of-N encoding (that is, one-hot 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]

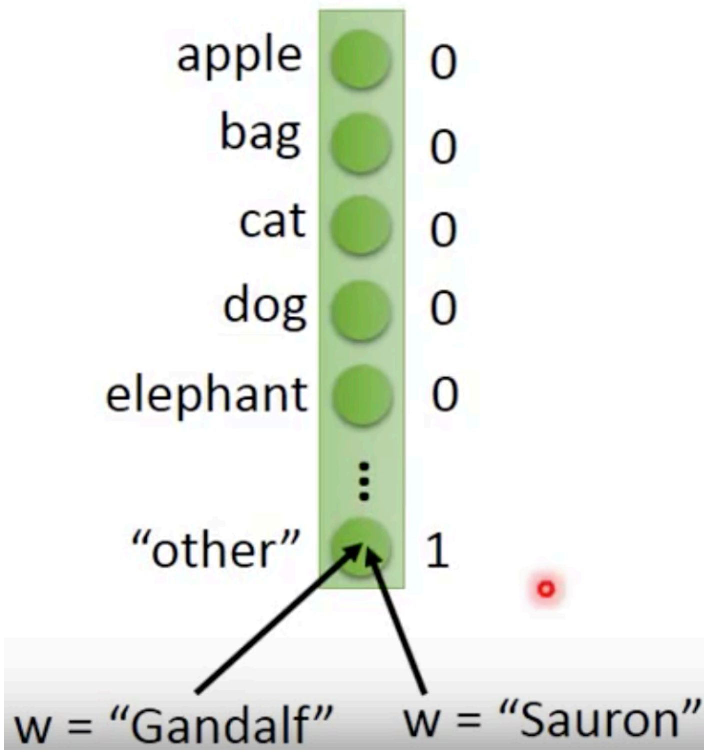
cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

elephant = [0 0 0 0 1]

Beyond 1-of-N encoding

Dimension for "Other"



Dense word embedding

Word2Vec (pre-trained)

GloVe (pre-trained)

Train your own embedding

Example Application

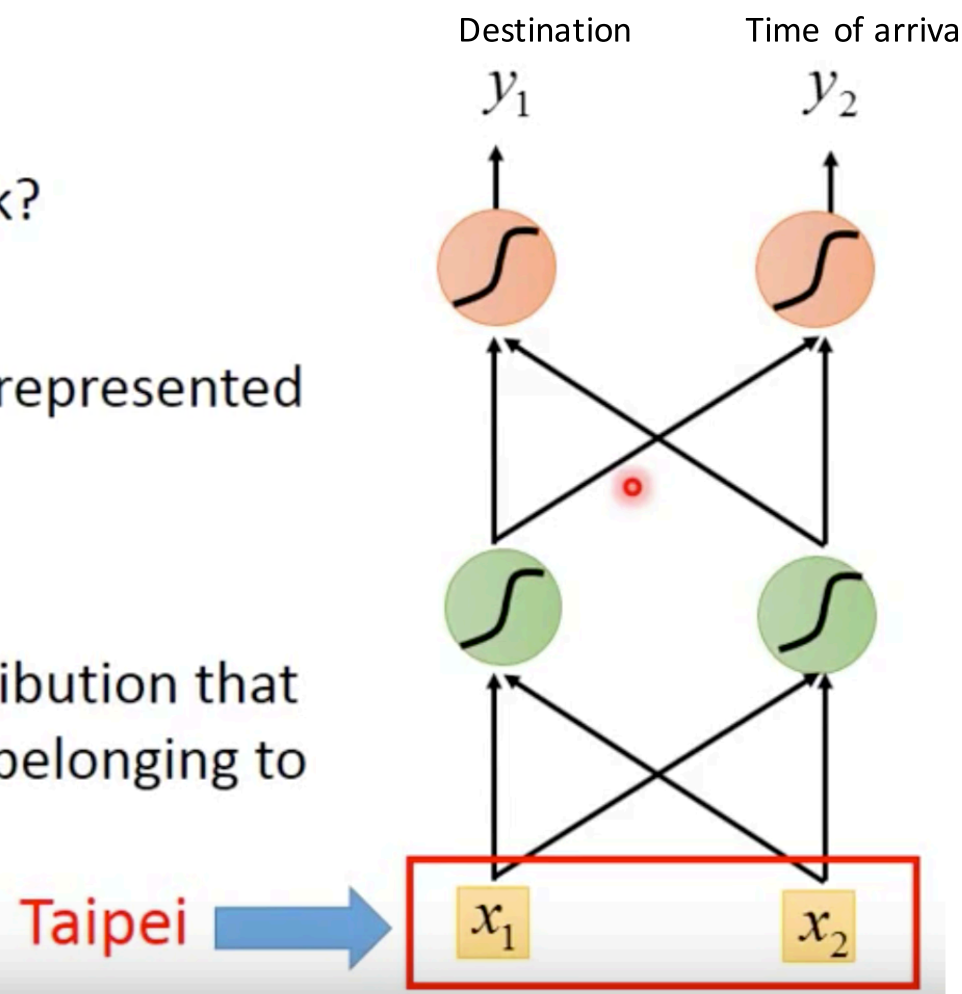
Solving slot filling by Feedforward network?

Input: a word

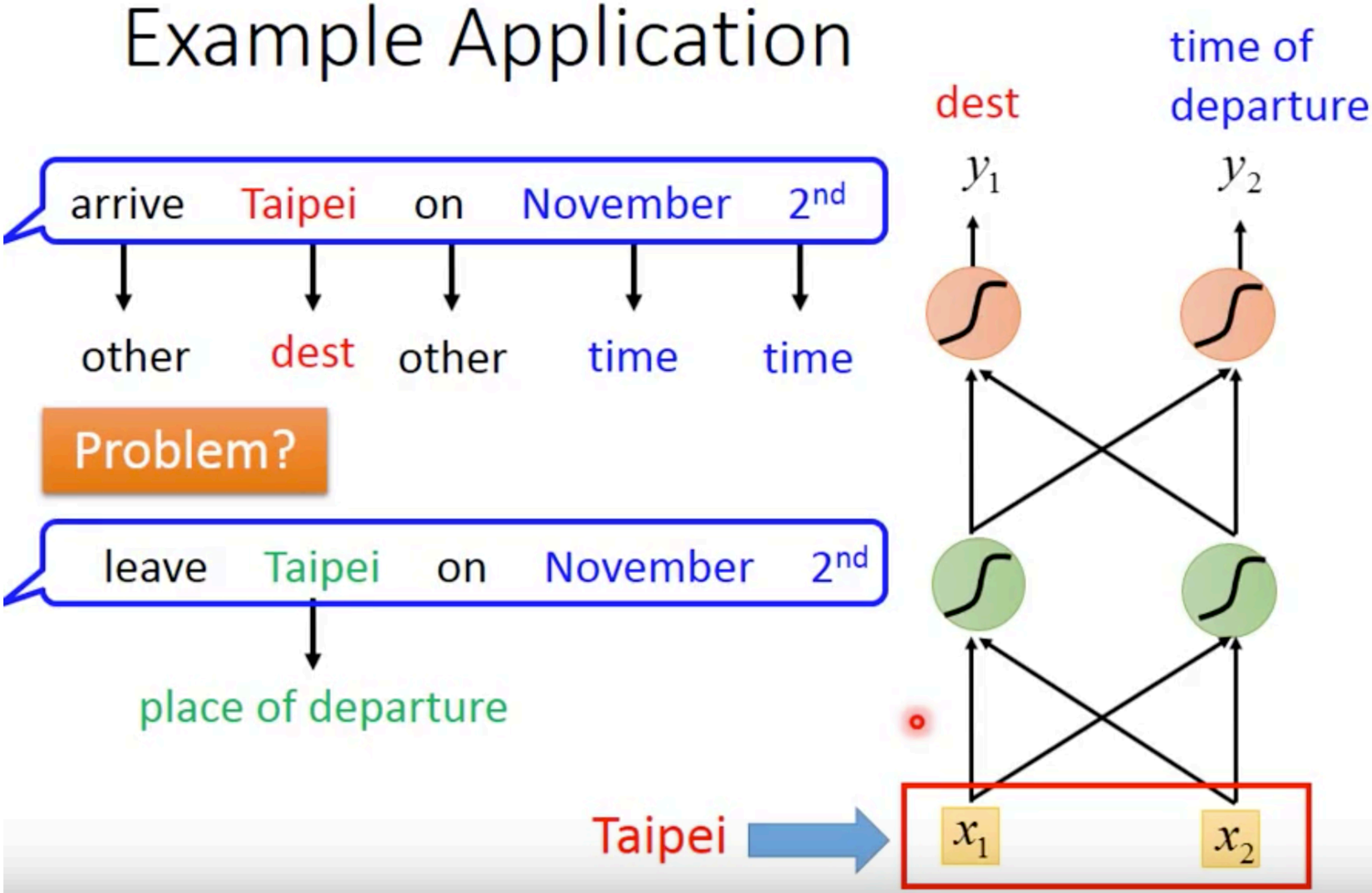
(Each word is represented as a vector)

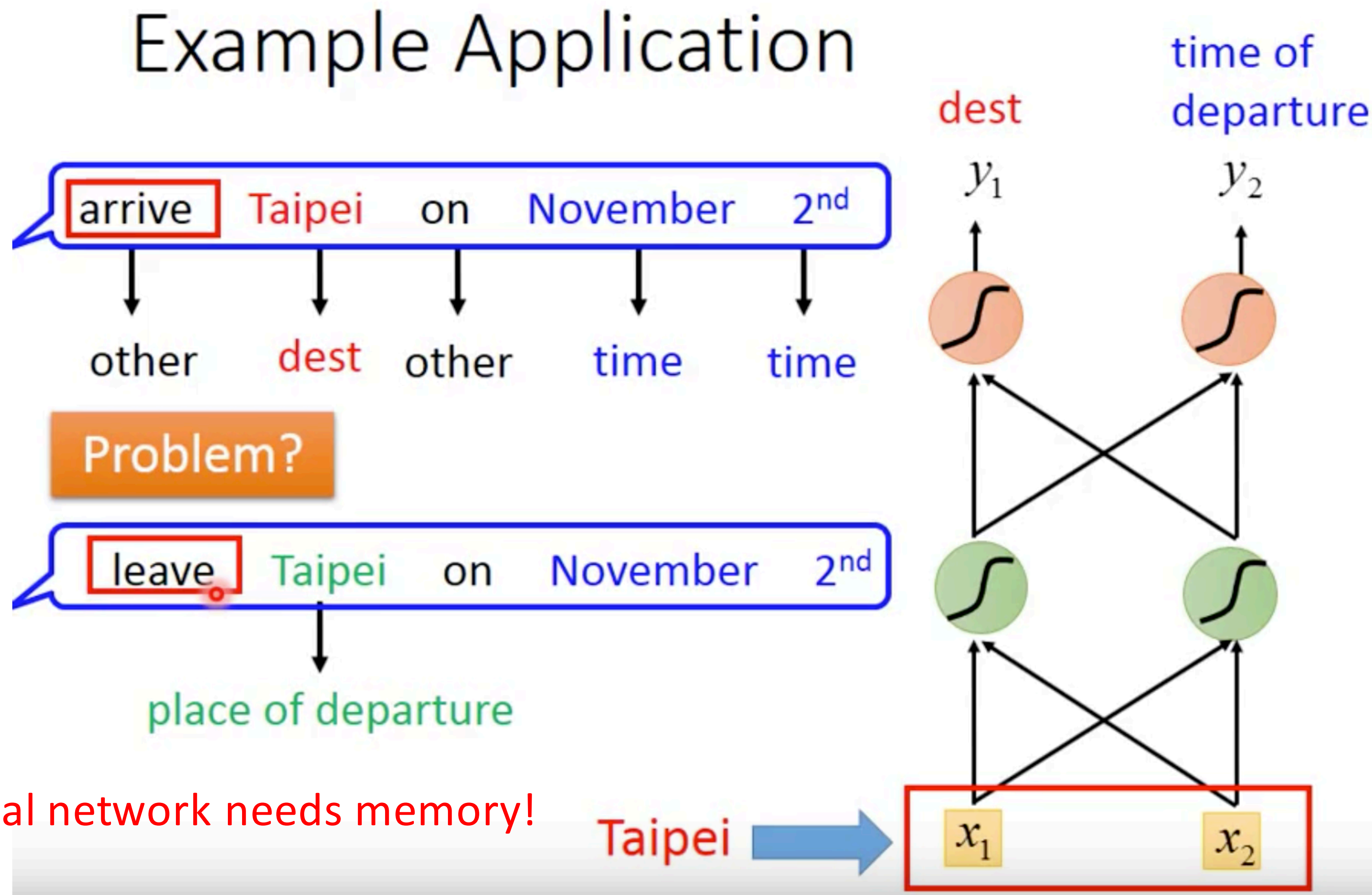
Output:

Probability distribution that the input word belonging to the slots



Example Application



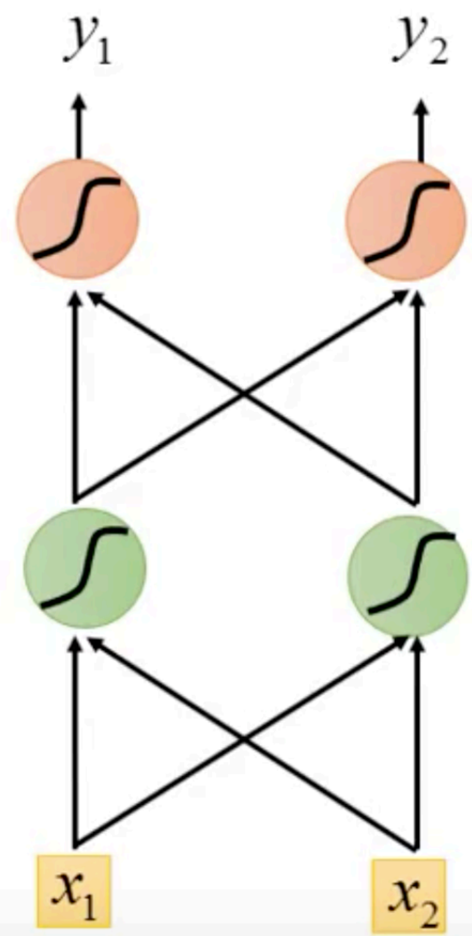


Neural network needs memory!

Recurrent Neural Network (RNN)

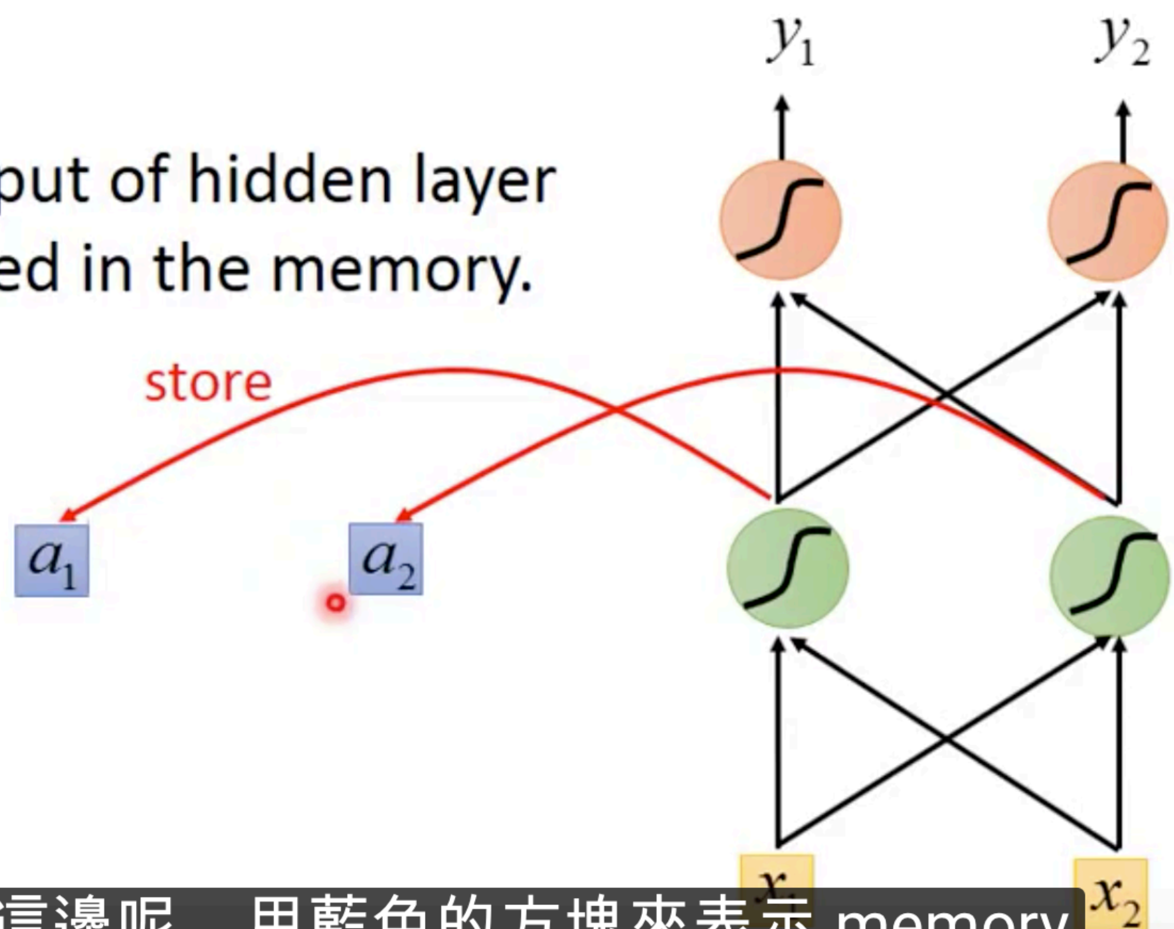
The output of hidden layer are stored in the memory.

a_1 a_2



Recurrent Neural Network (RNN)

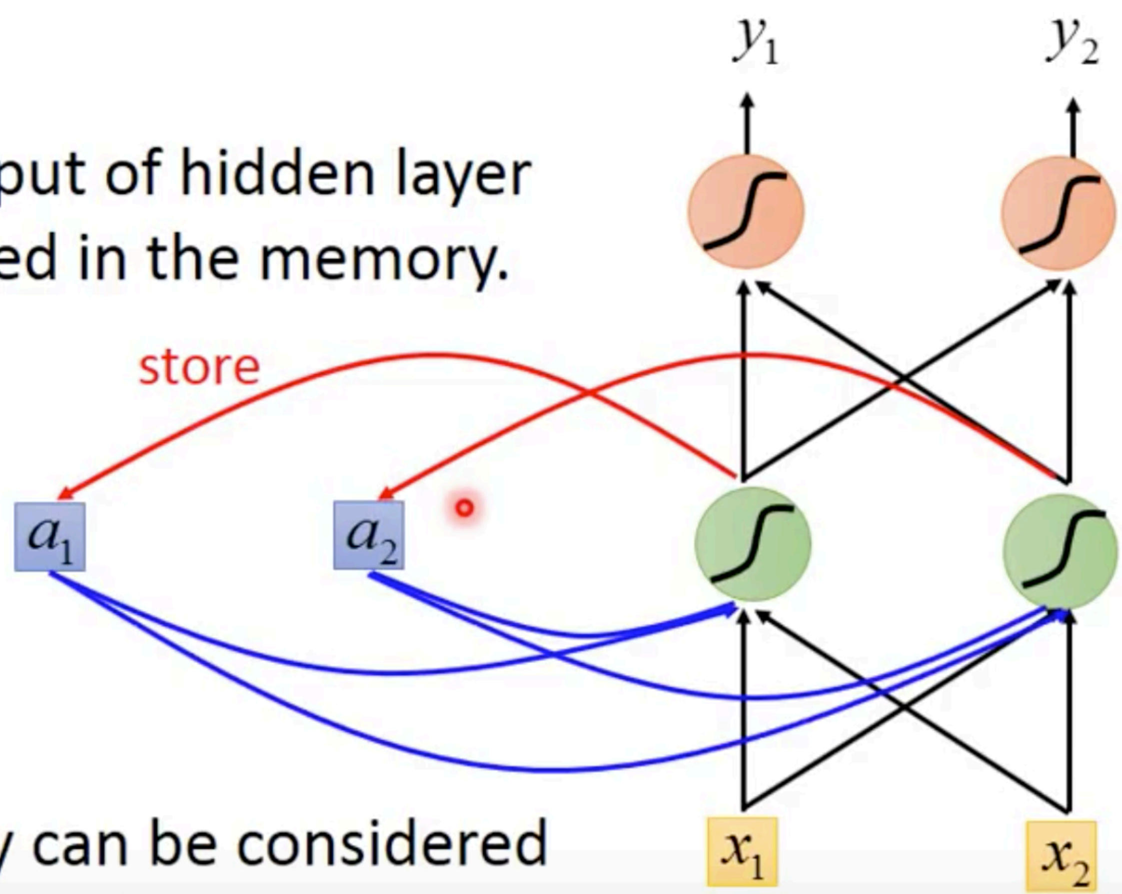
The output of hidden layer are stored in the memory.



這邊呢，用藍色的方塊來表示 memory

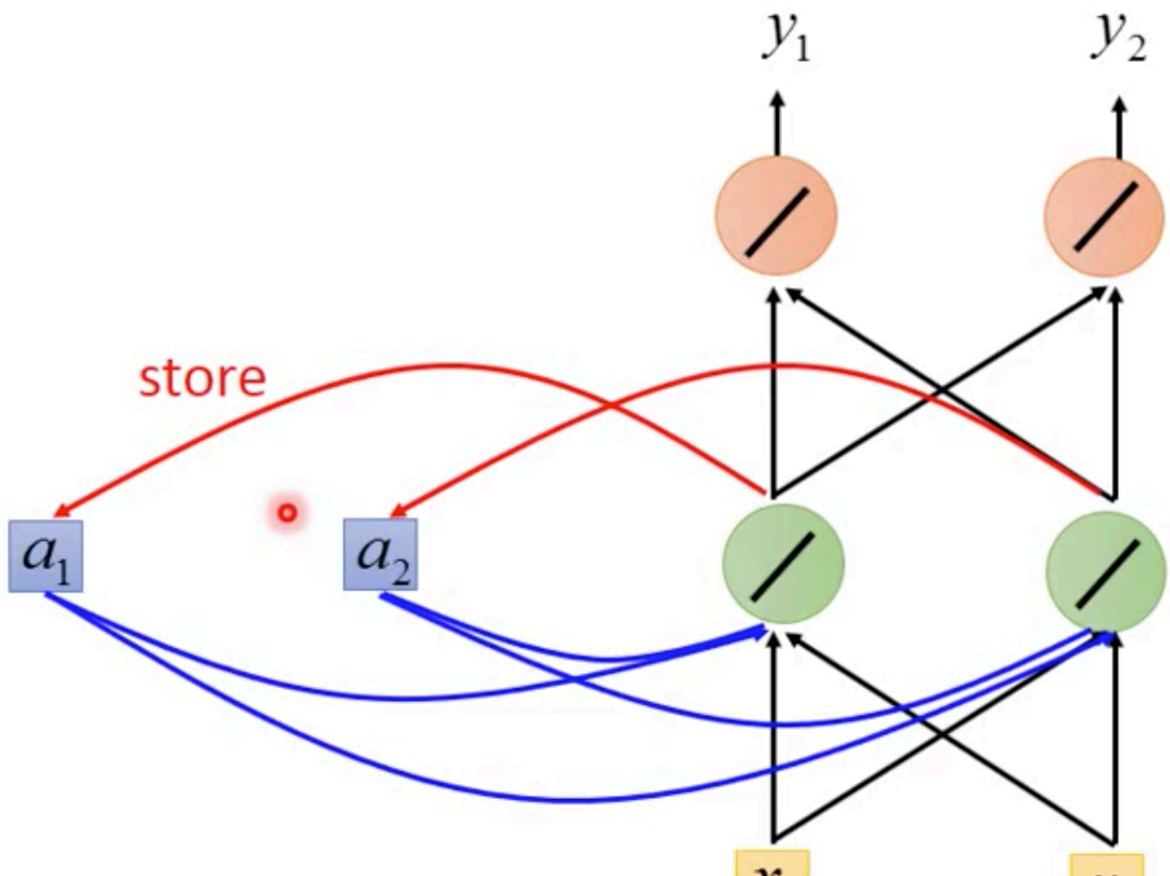
Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.



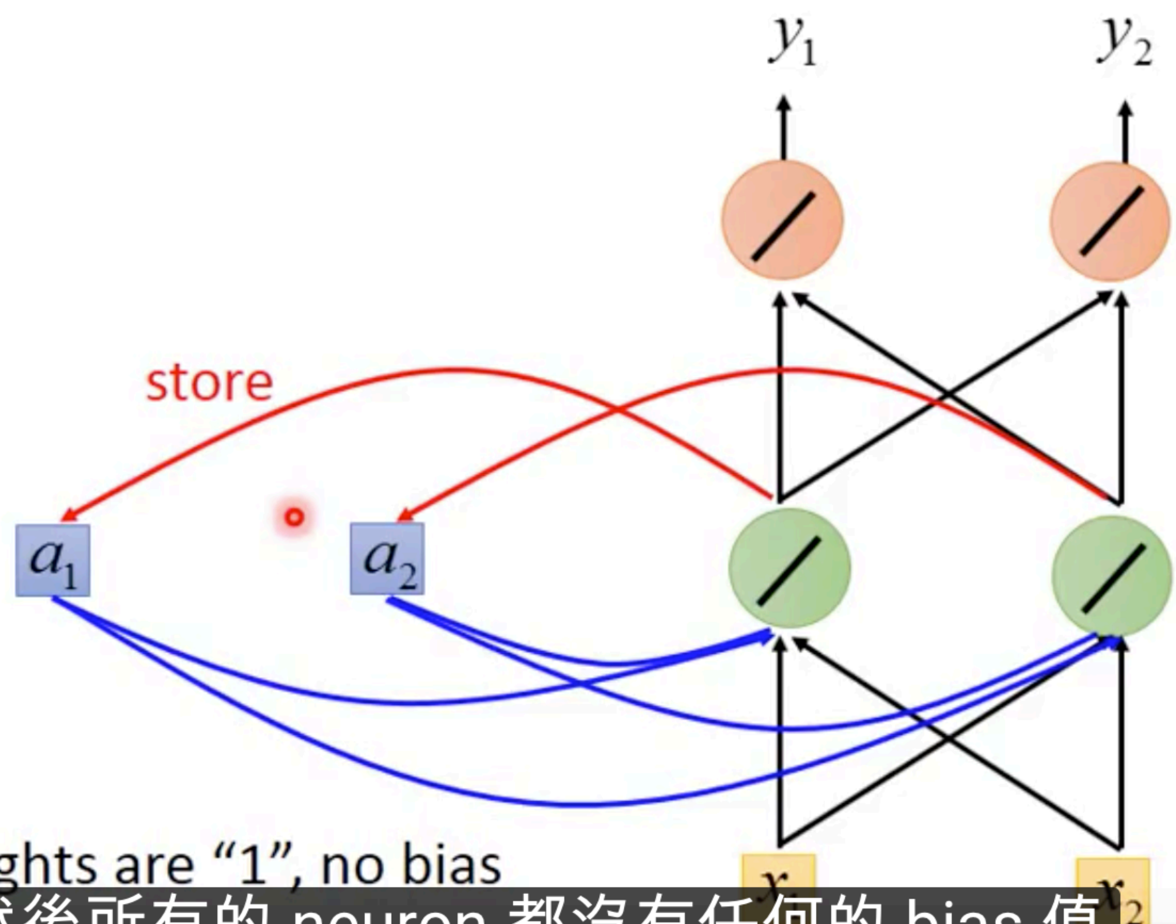
Memory can be considered as another input.

Example



那我想直接舉個例子，大家可能會比較清楚

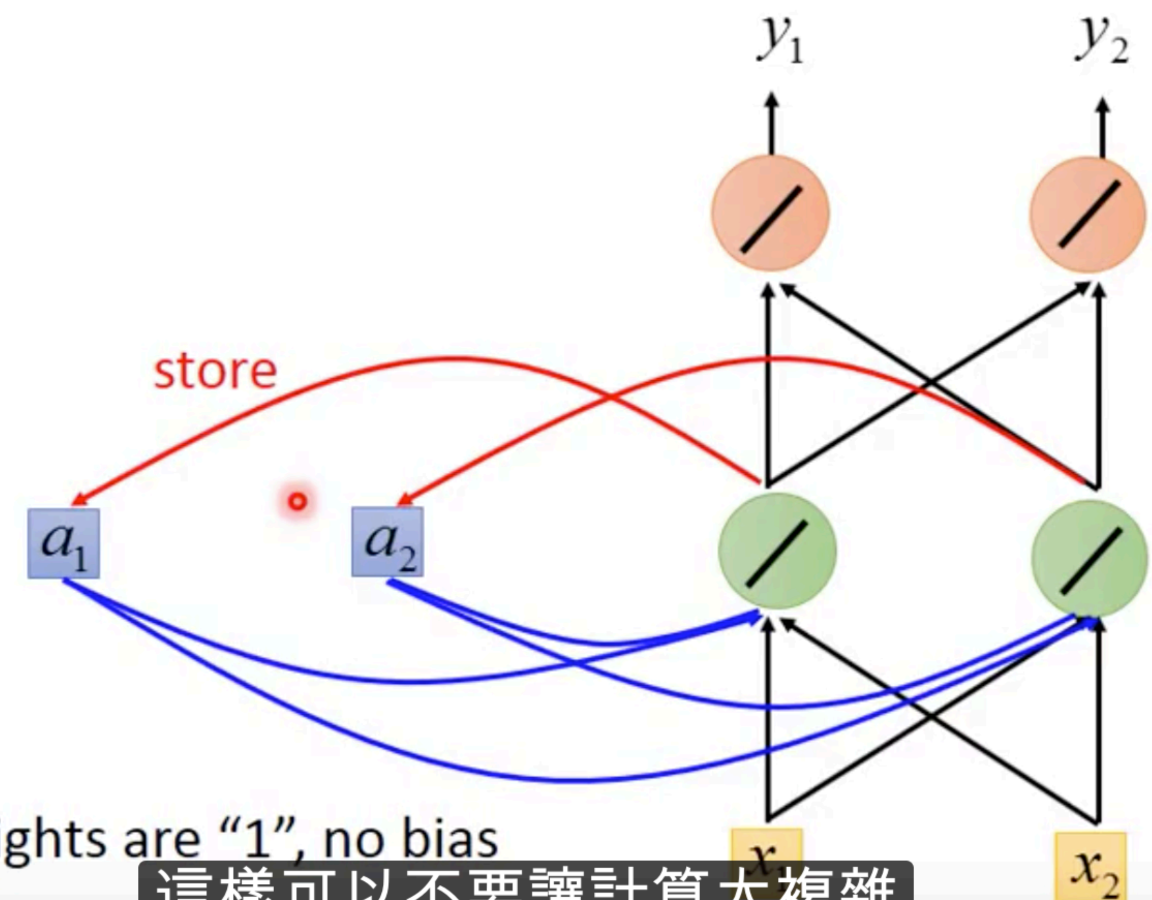
Example



All the weights are "1", no bias

然後所有的 neuron 都沒有任何的 bias 值

Example



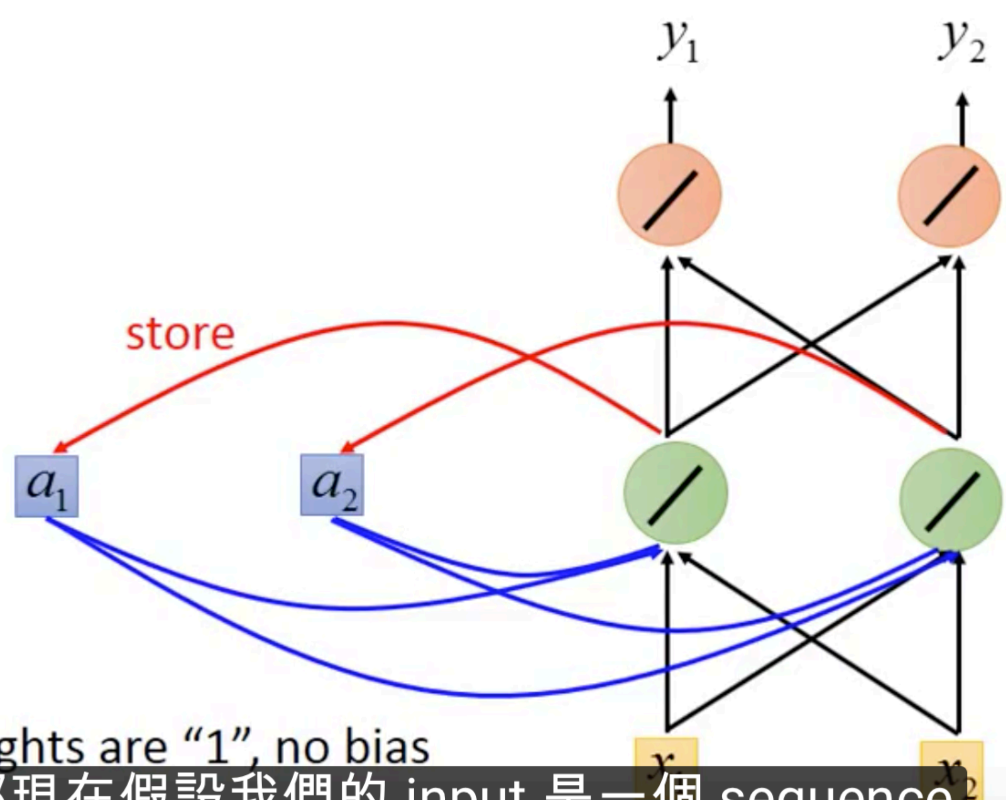
All the weights are "1", no bias

All activation functions are linear

這樣可以不要讓計算太複雜

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

Example



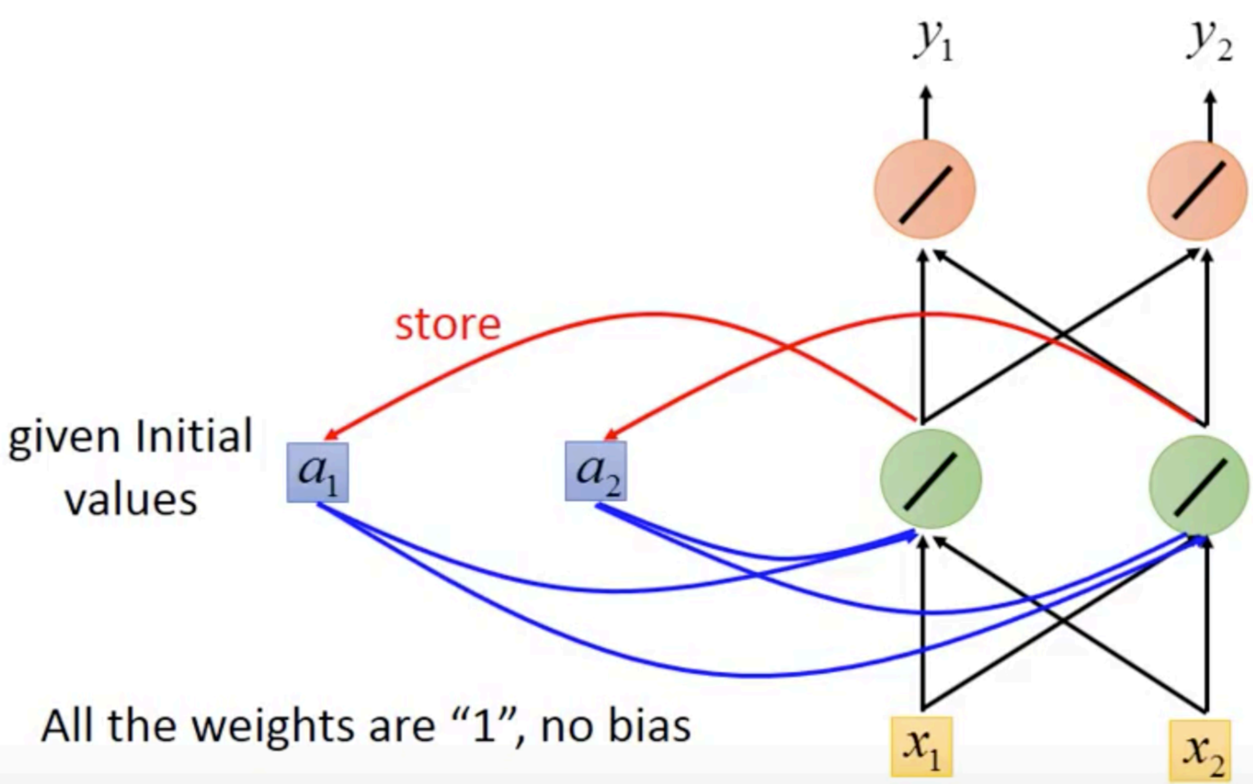
All the weights are "1", no bias

All activation functions are linear

那現在假設我們的 input 是一個 sequence

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

Example

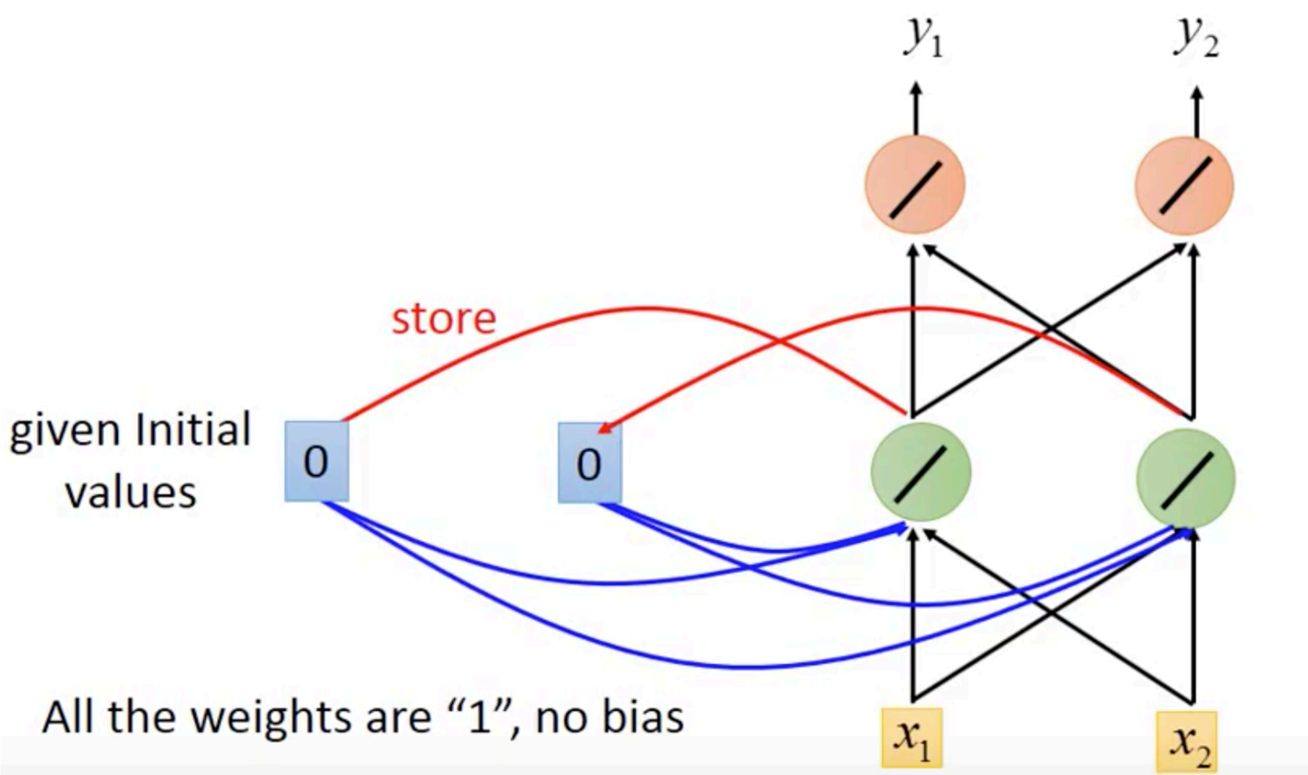


All the weights are "1", no bias

All activation functions are linear

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

Example

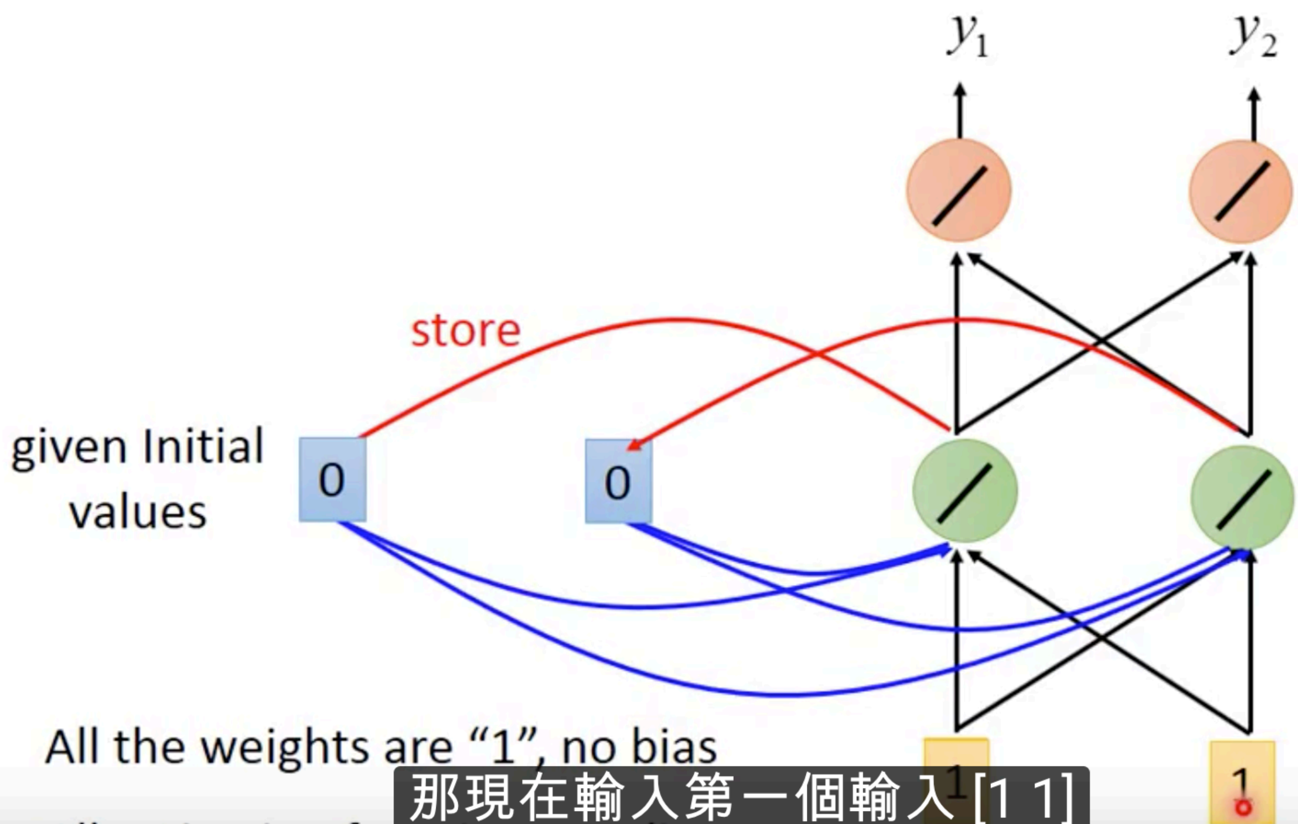


All the weights are “1”, no bias

All activation functions are linear

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

Example



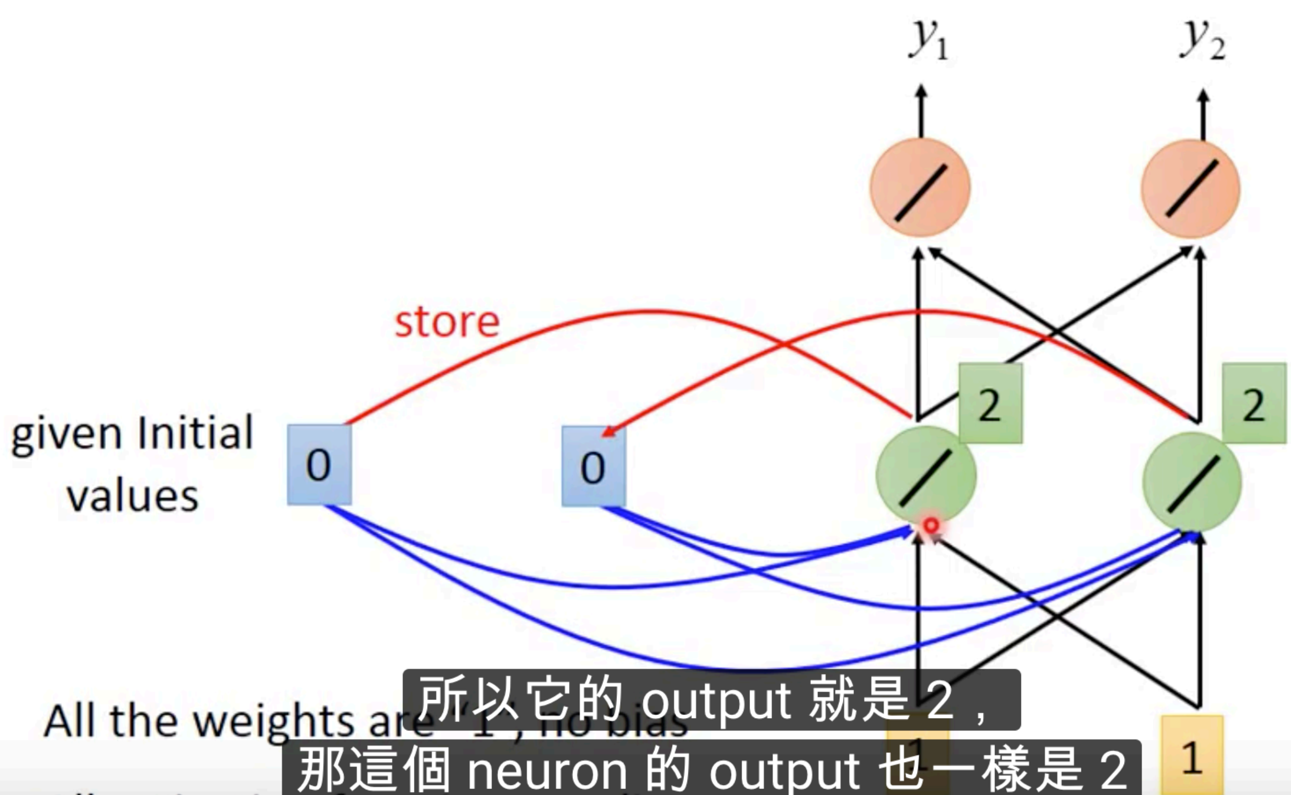
All the weights are “1”, no bias

All activation functions are linear

那現在輸入第一個輸入[1 1]

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

Example



All the weights are "1", no bias

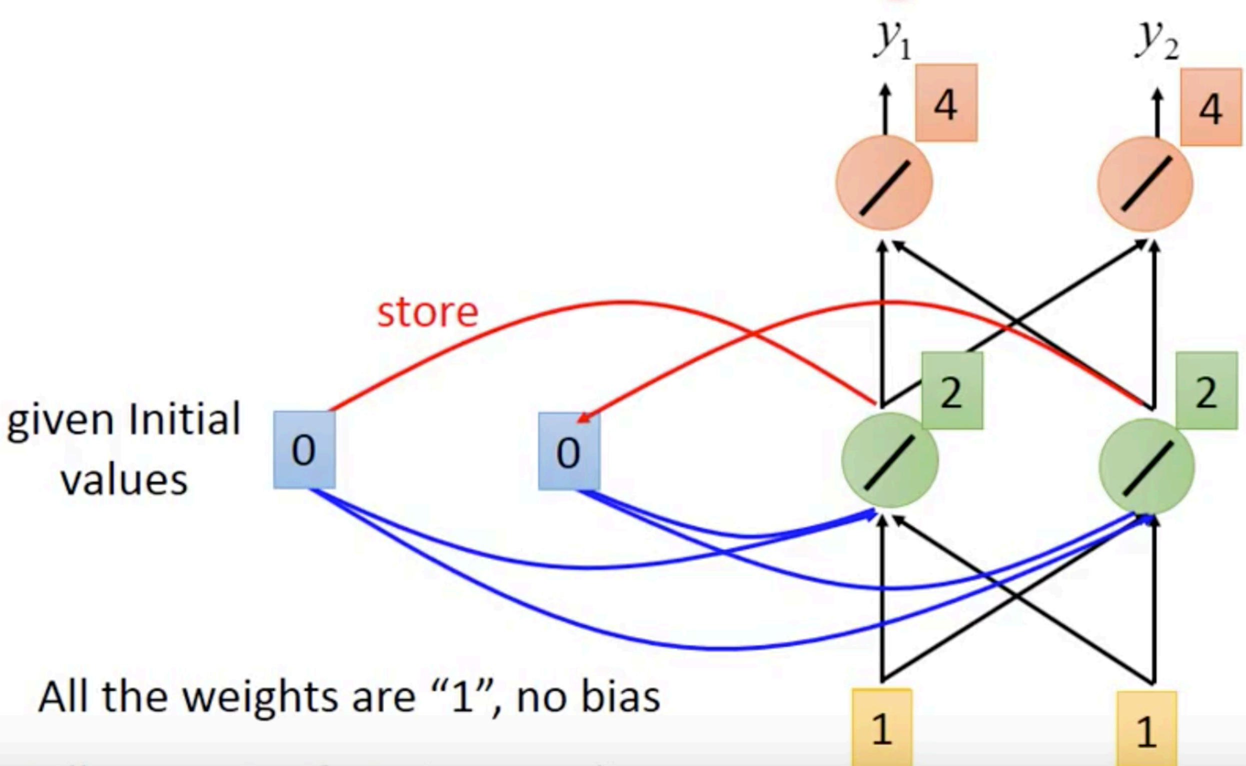
All activation functions are linear

所以它的 output 就是 2，
那這個 neuron 的 output 也一樣是 2

Example

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

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



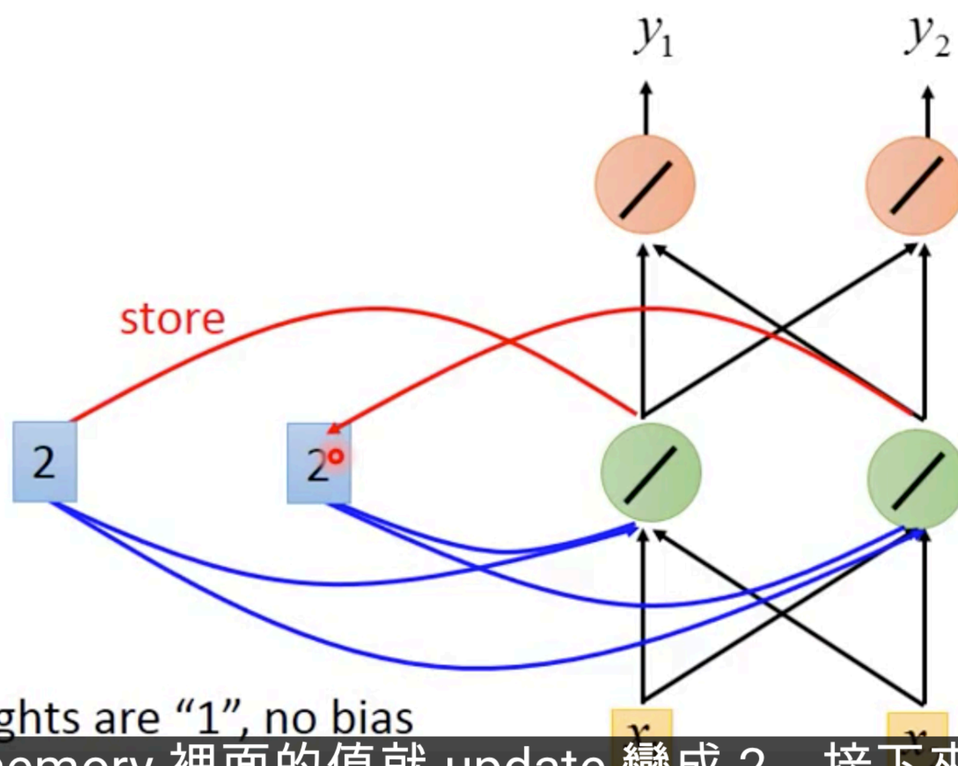
All the weights are "1", no bias

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



All the weights are “1”, no bias

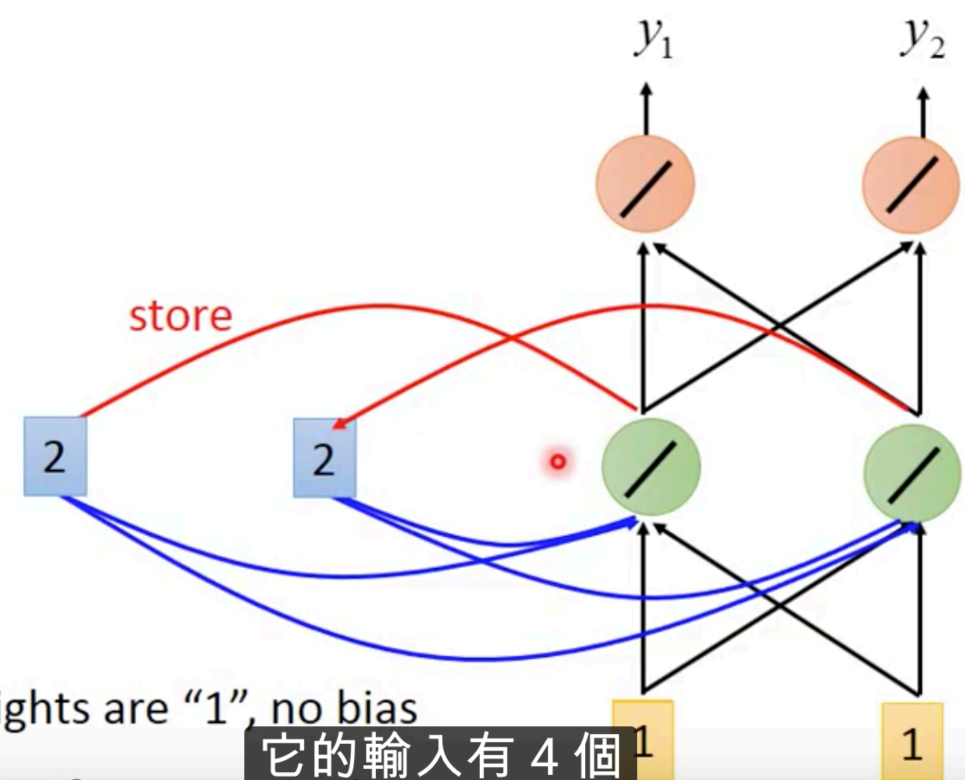
All activation functions are linear

所以 memory 裡面的值就 update 變成 2，接下來呢

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

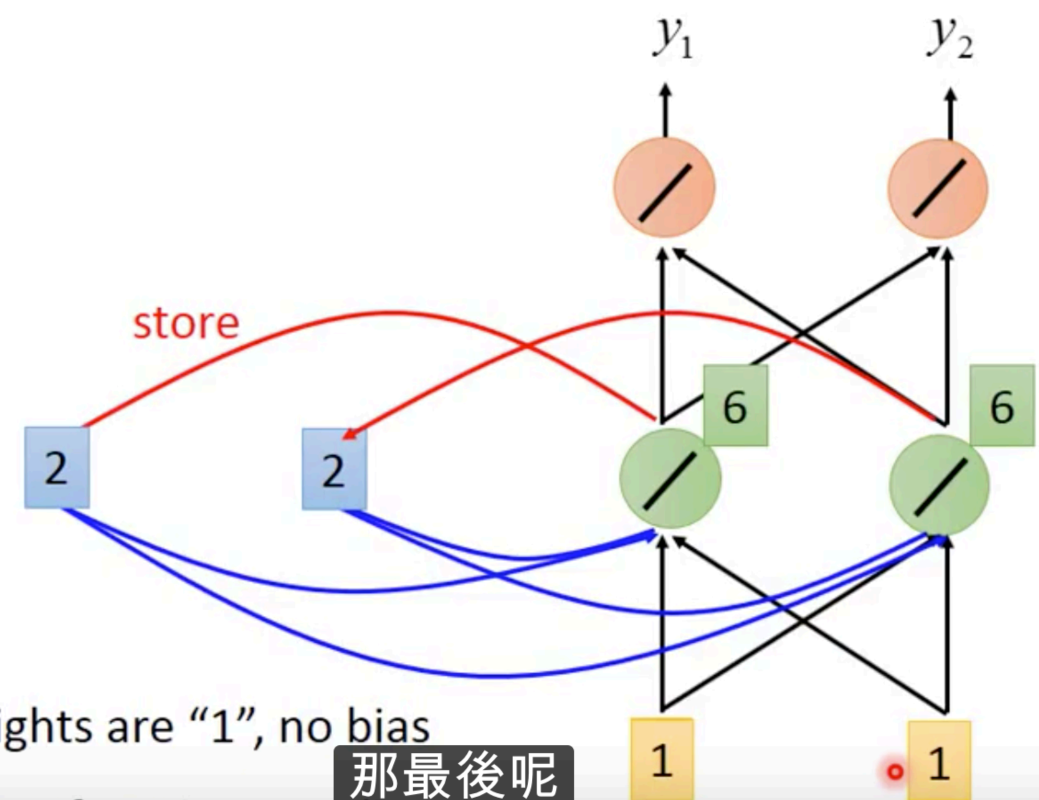
All activation functions are linear

它的輸入有 4 個

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

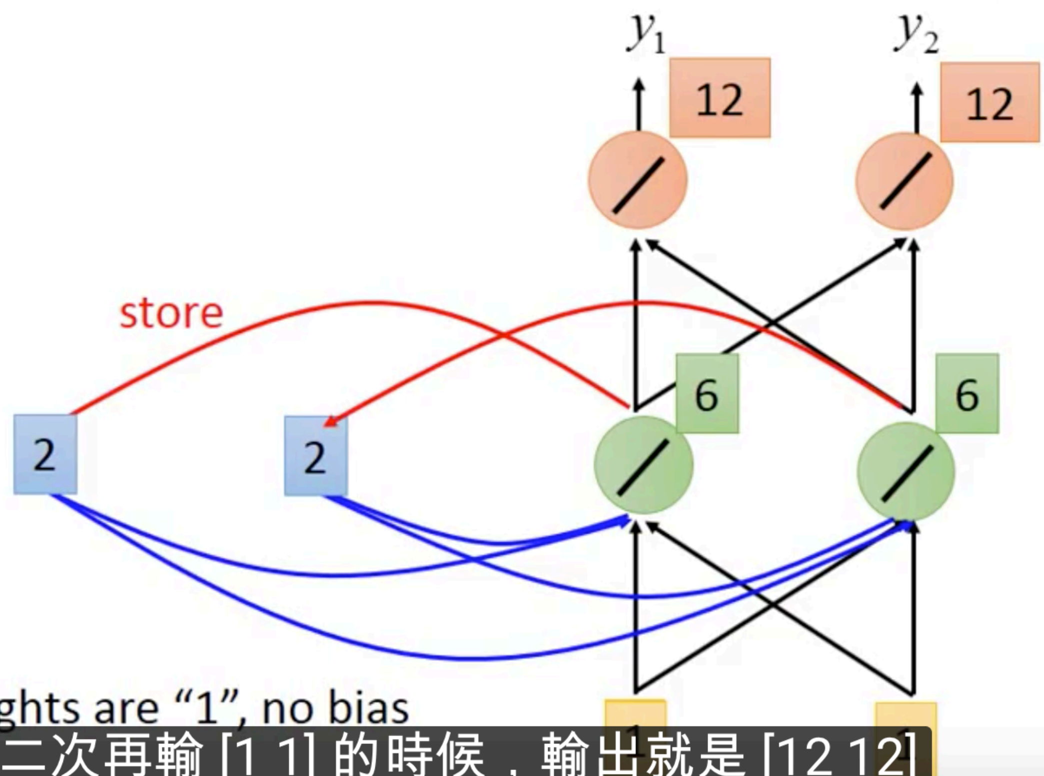
All activation functions are linear

那最後呢

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

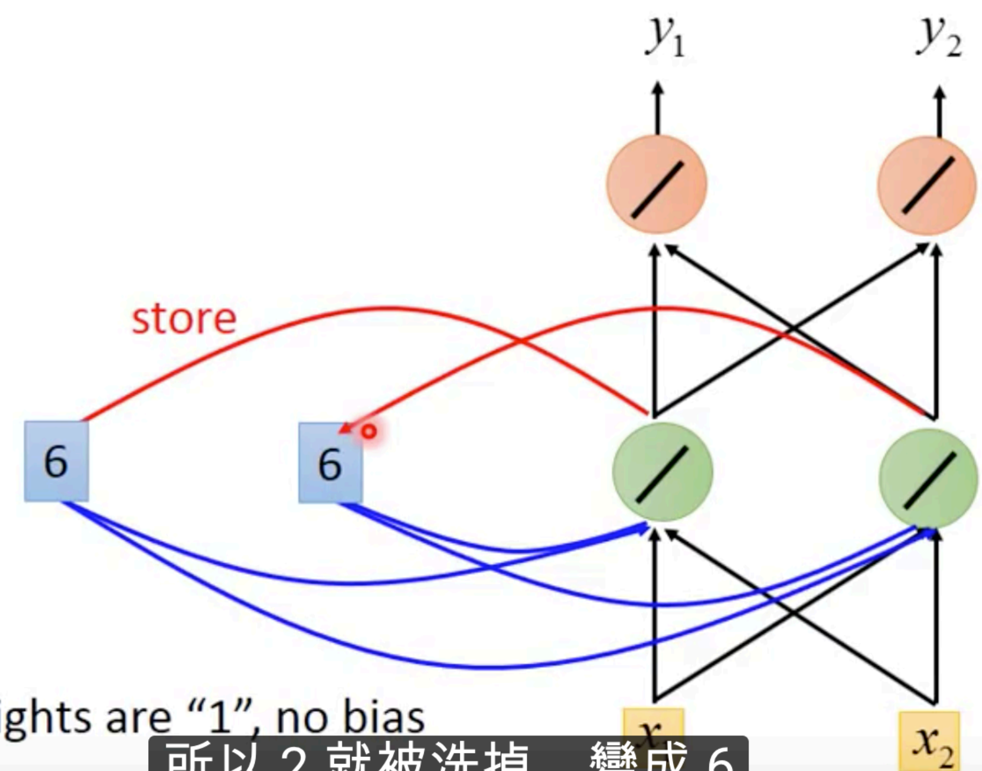
All activation functions are linear

第二次再輸 [1 1] 的時候，輸出就是 [12 12]

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

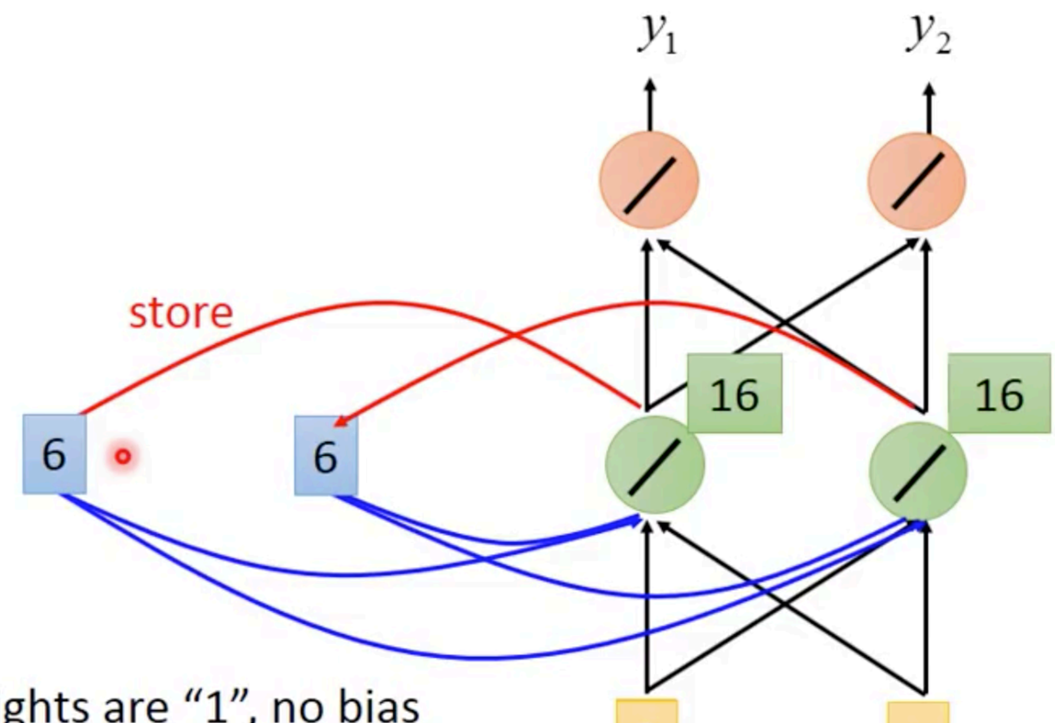
All activation functions are linear

所以 2 就被洗掉，變成 6

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

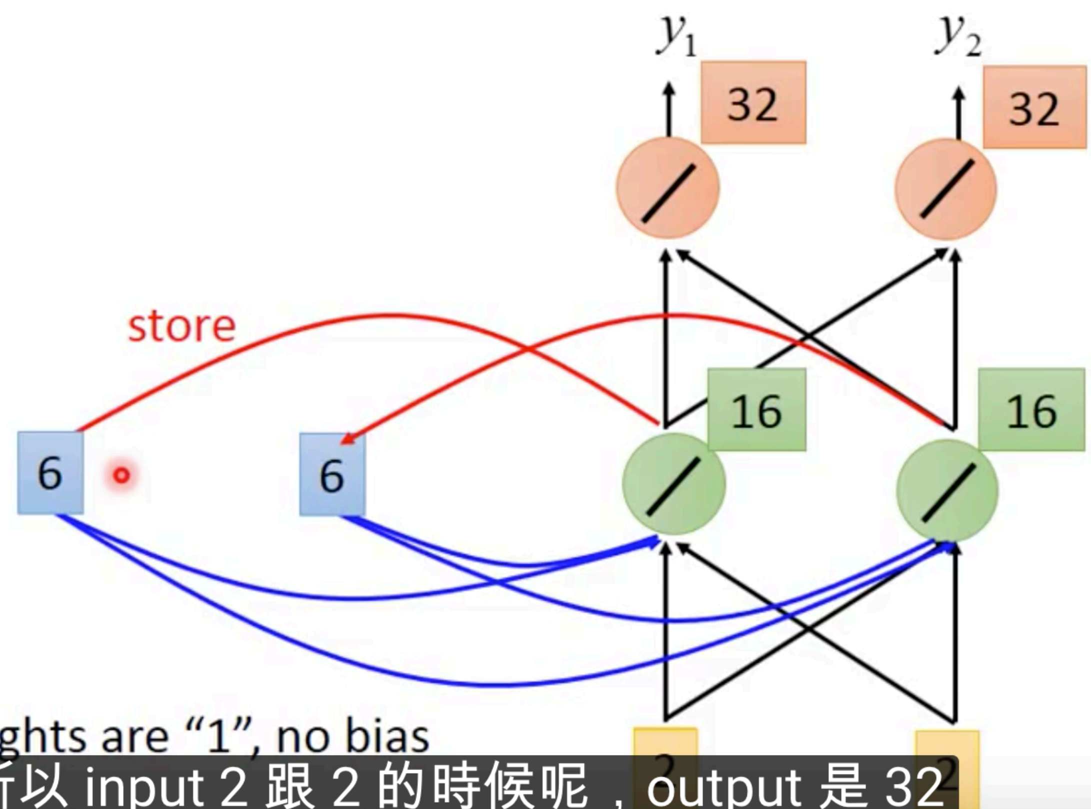
All activation functions are linear

那紅色 neuron 的 output 就是 32

Example

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

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



All the weights are "1", no bias

All activation functions are linear

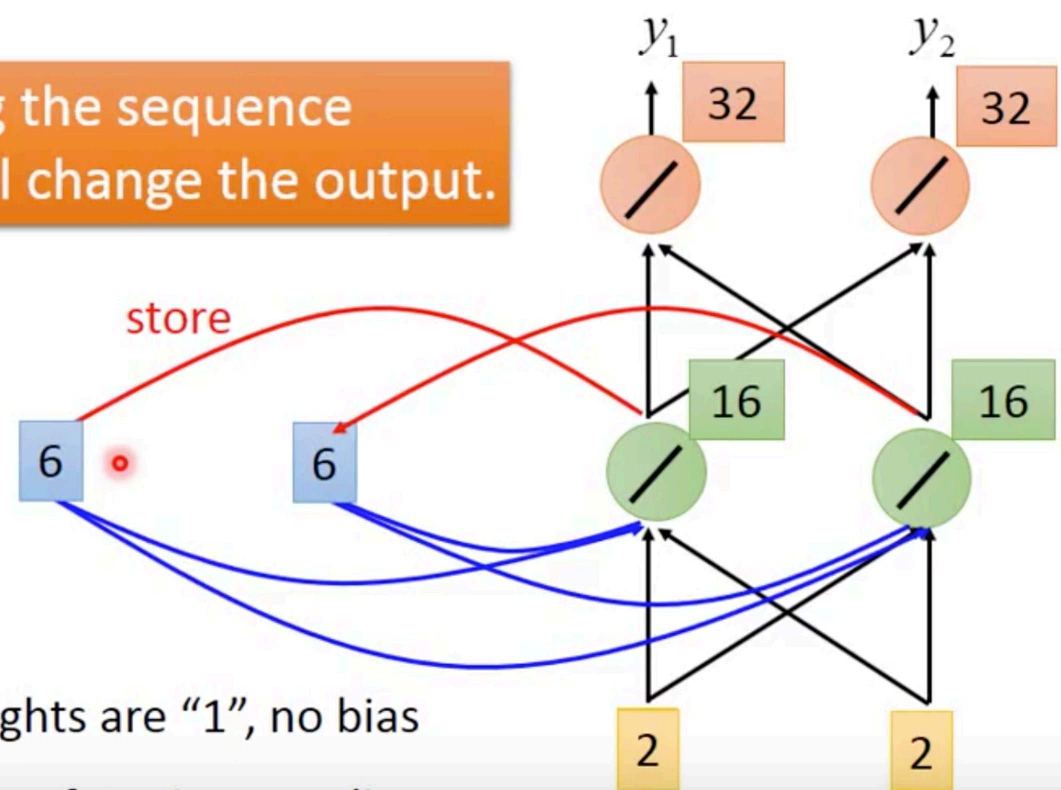
所以 input 2 跟 2 的時候呢，output 是 32

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.



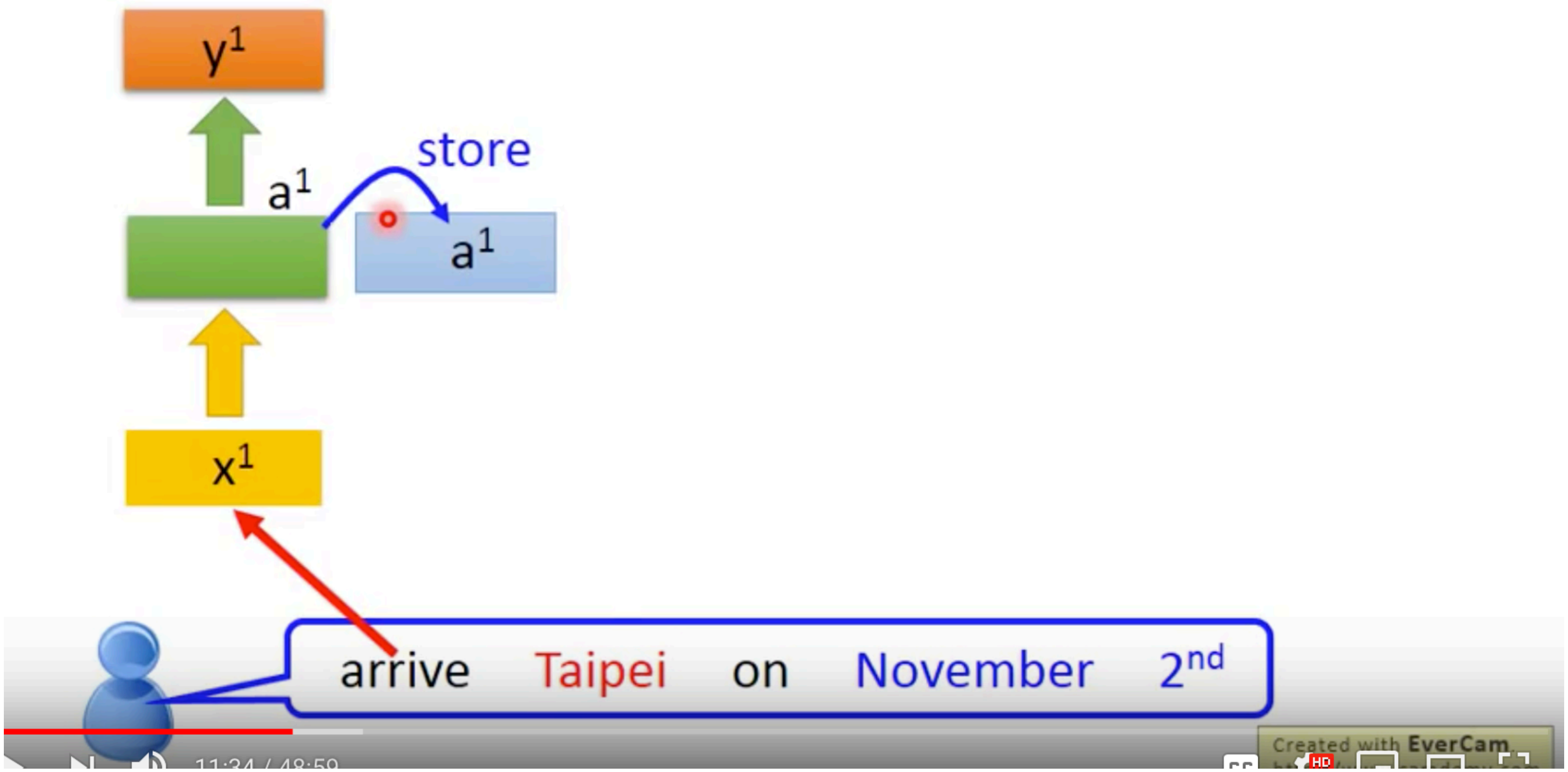
All the weights are "1", no bias

All activation functions are linear

RNN

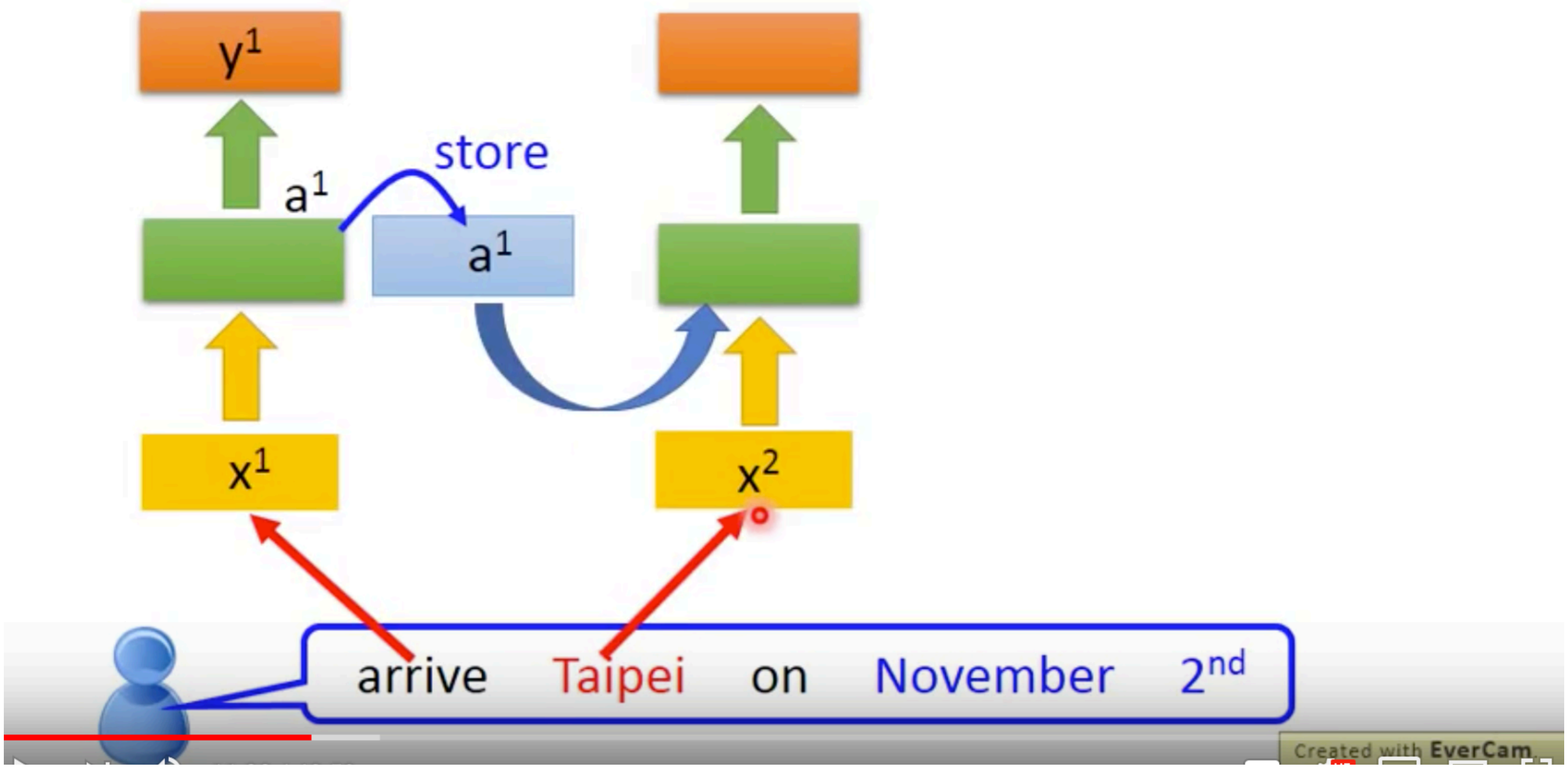
RNN

Probability of
"arrive" in each slot



RNN

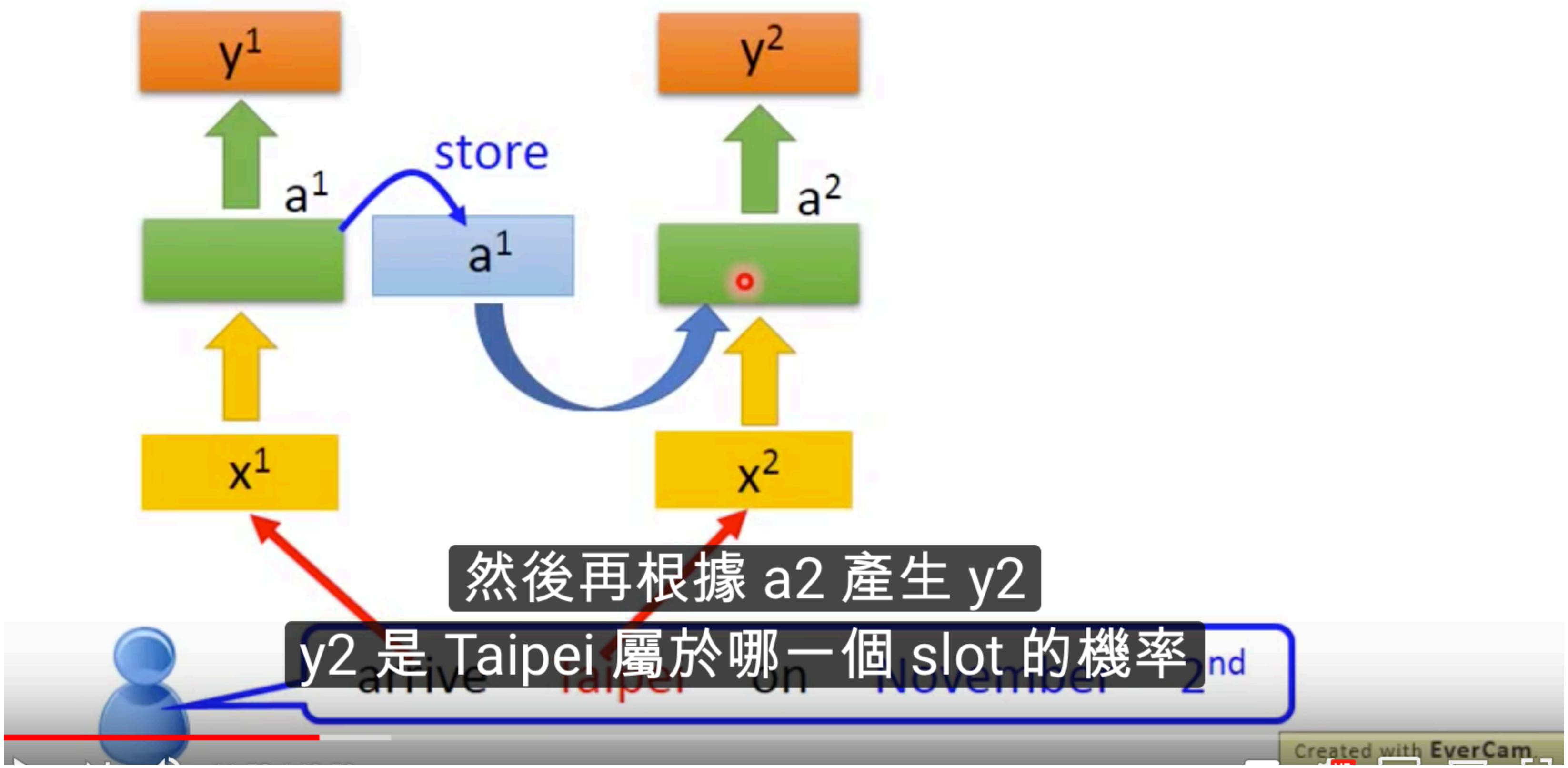
Probability of
"arrive" in each slot



RNN

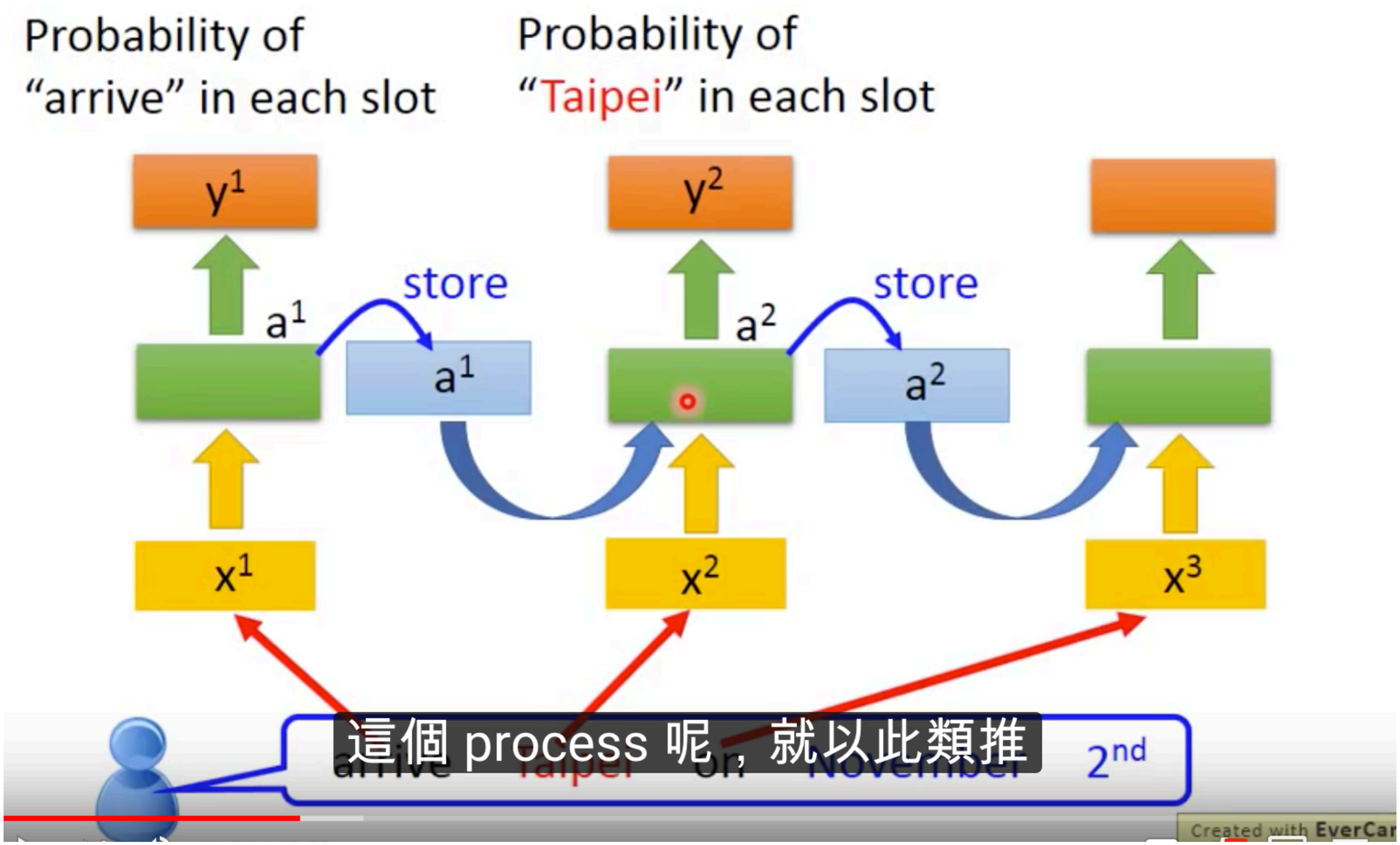
Probability of "arrive" in each slot

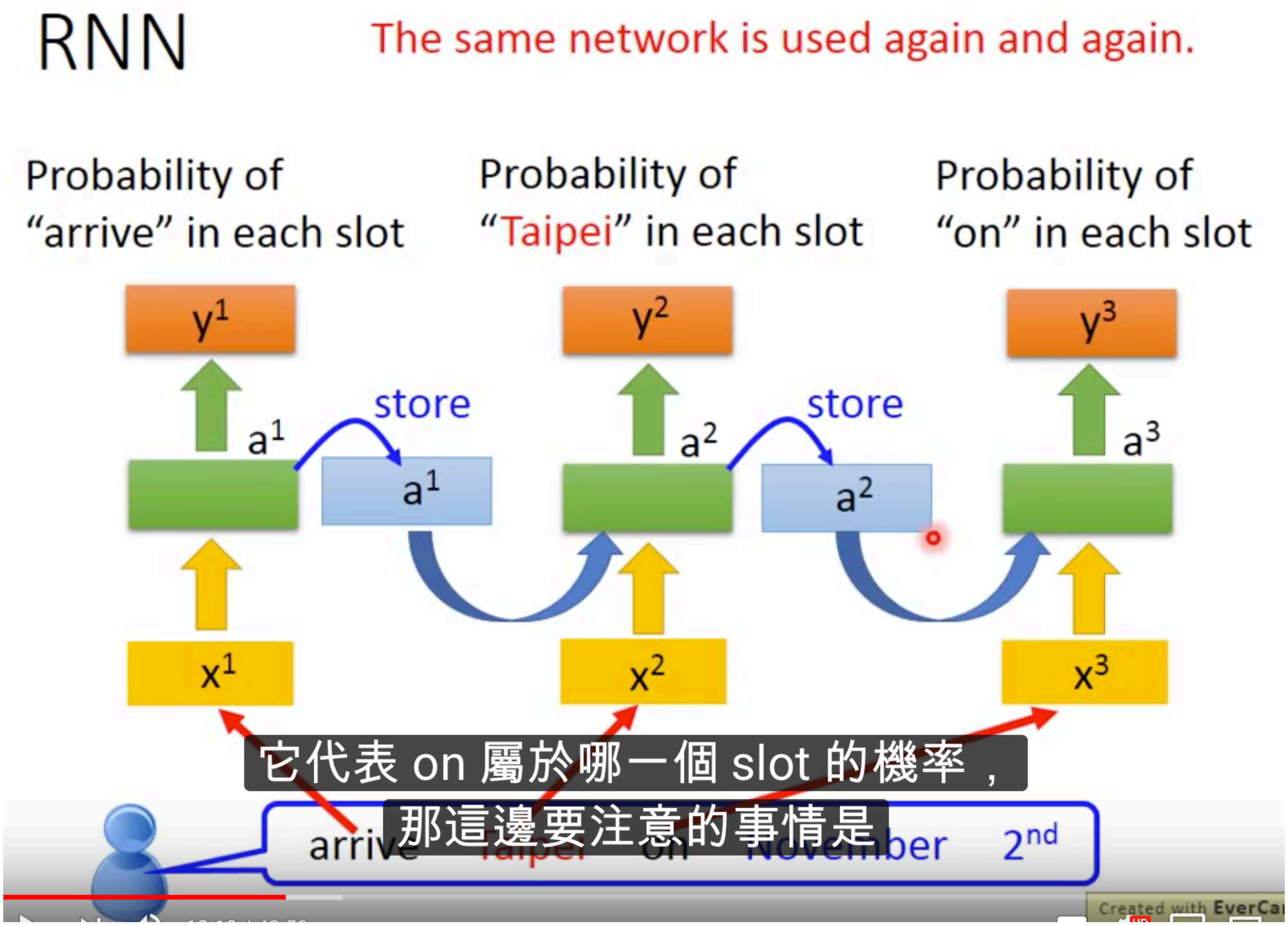
Probability of "Taipei" in each slot

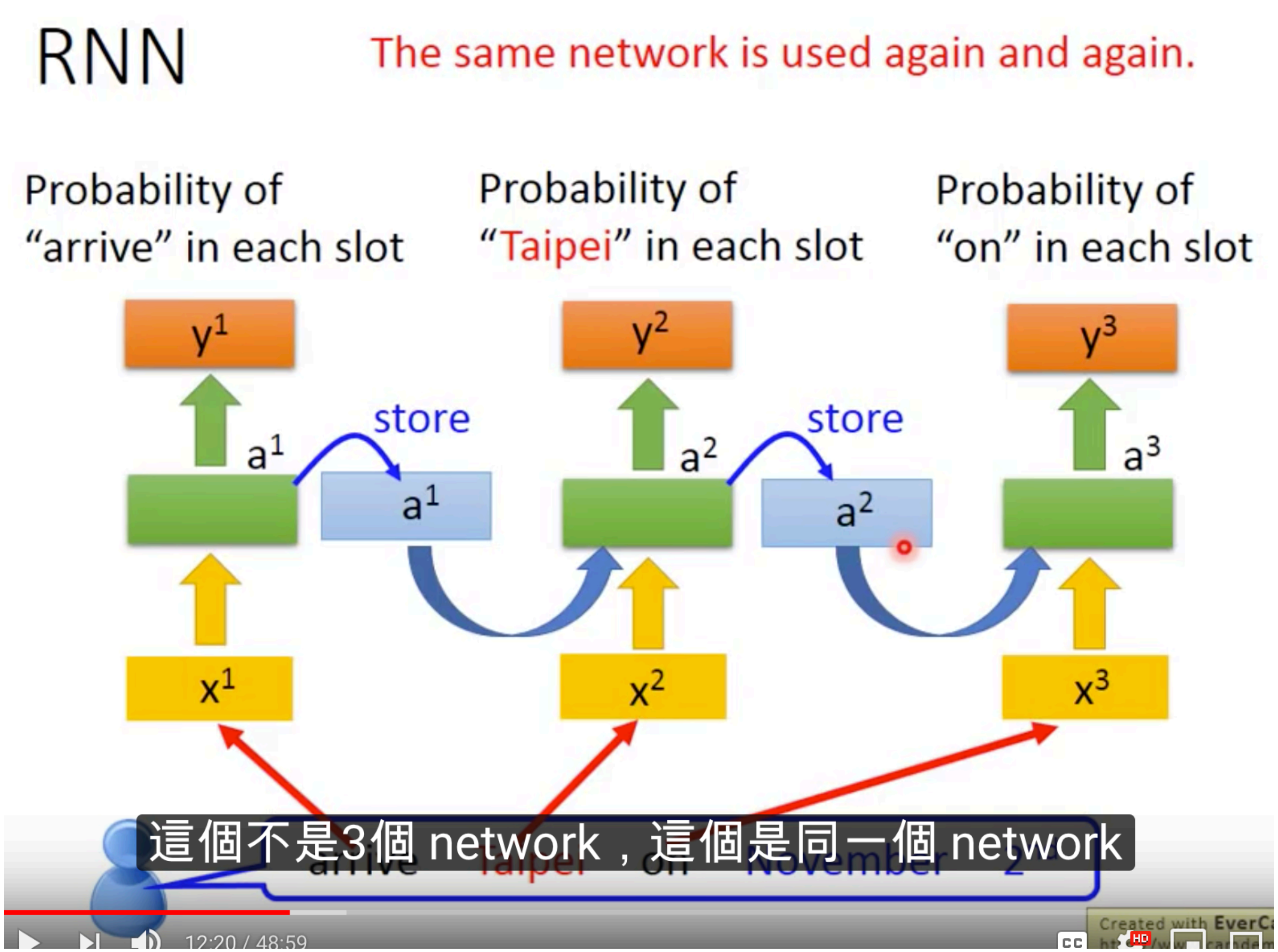


RNN

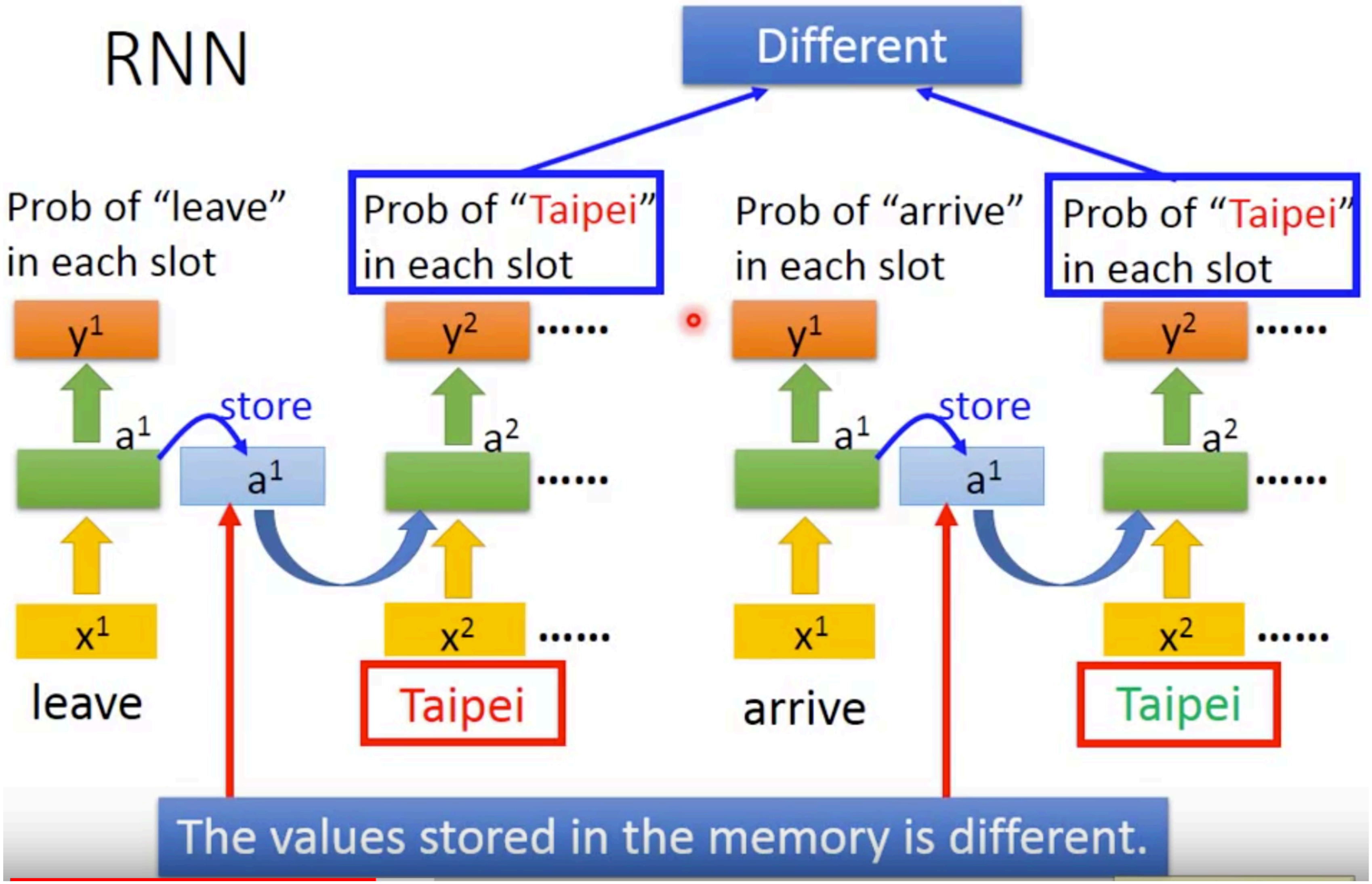
The same network is used again and again.




RNN
The same network is used again and again.
Probability of "arrive" in each slot
Probability of "Taipei" in each slot
Probability of "on" in each slot
y1
y2
y3
a1
a2
a3
store
store
a1
a2
x1
x2
x3
arrive
2nd
它代表 on 屬於哪一個 slot 的機率，
那這邊要注意的事情是

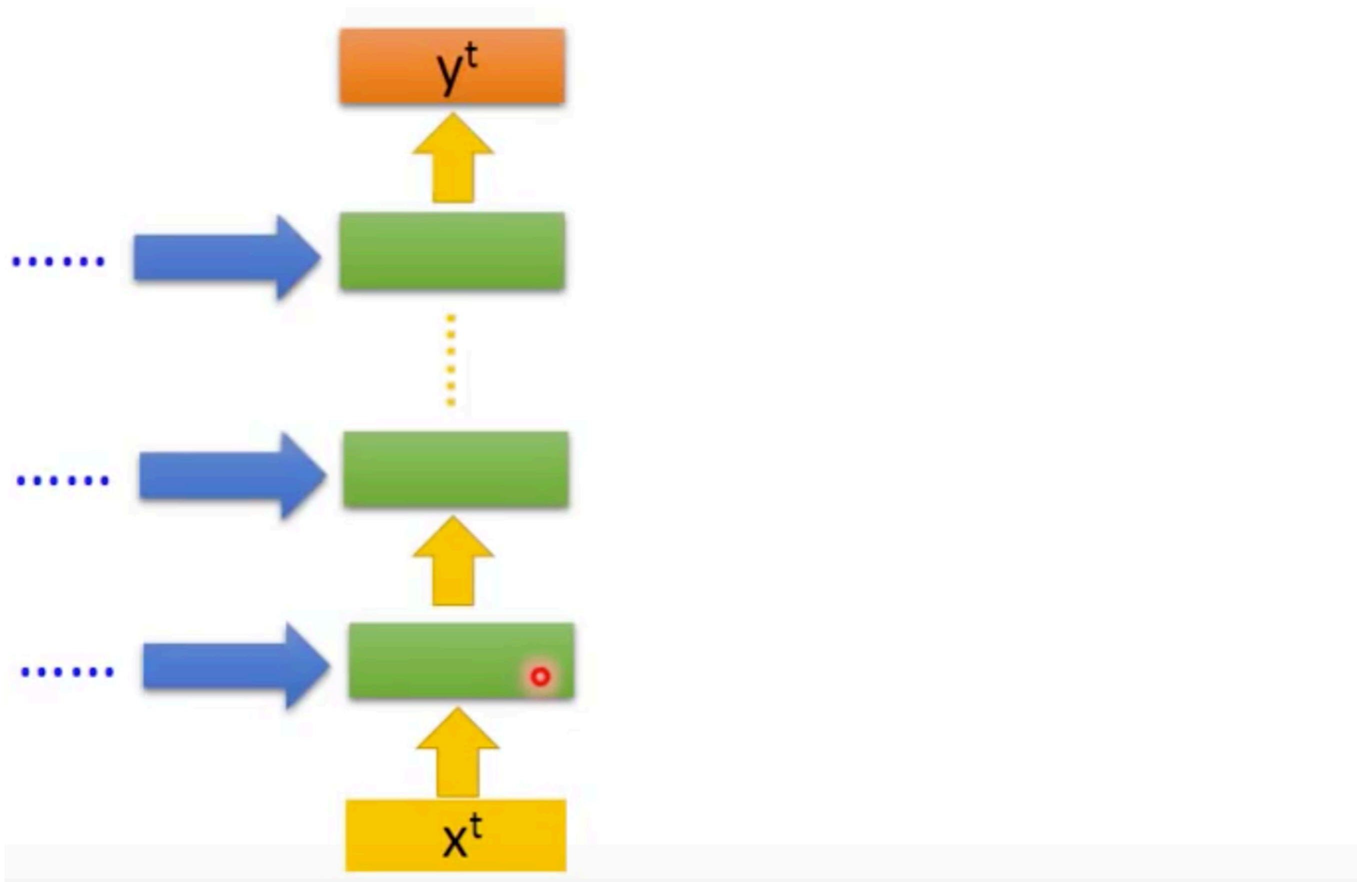


Note: they are not three networks. They are the same network used three times.

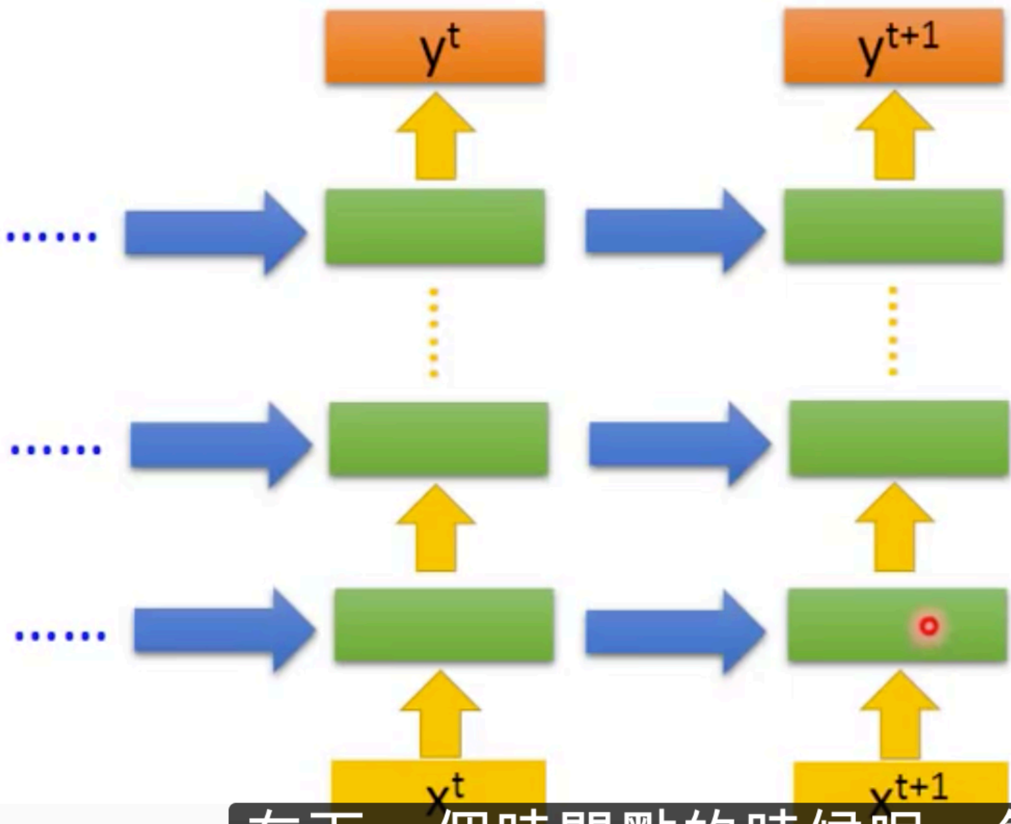

RNN
Different
Prob of "leave" in each slot
Prob of "Taipei" in each slot
Prob of "arrive" in each slot
Prob of "Taipei" in each slot
y1
y2
a1
a2
store
a1
x1
x2
leave
Taipei
arrive
Taipei
The values stored in the memory is different.

Of course it can be deep …

Of course it can be deep …

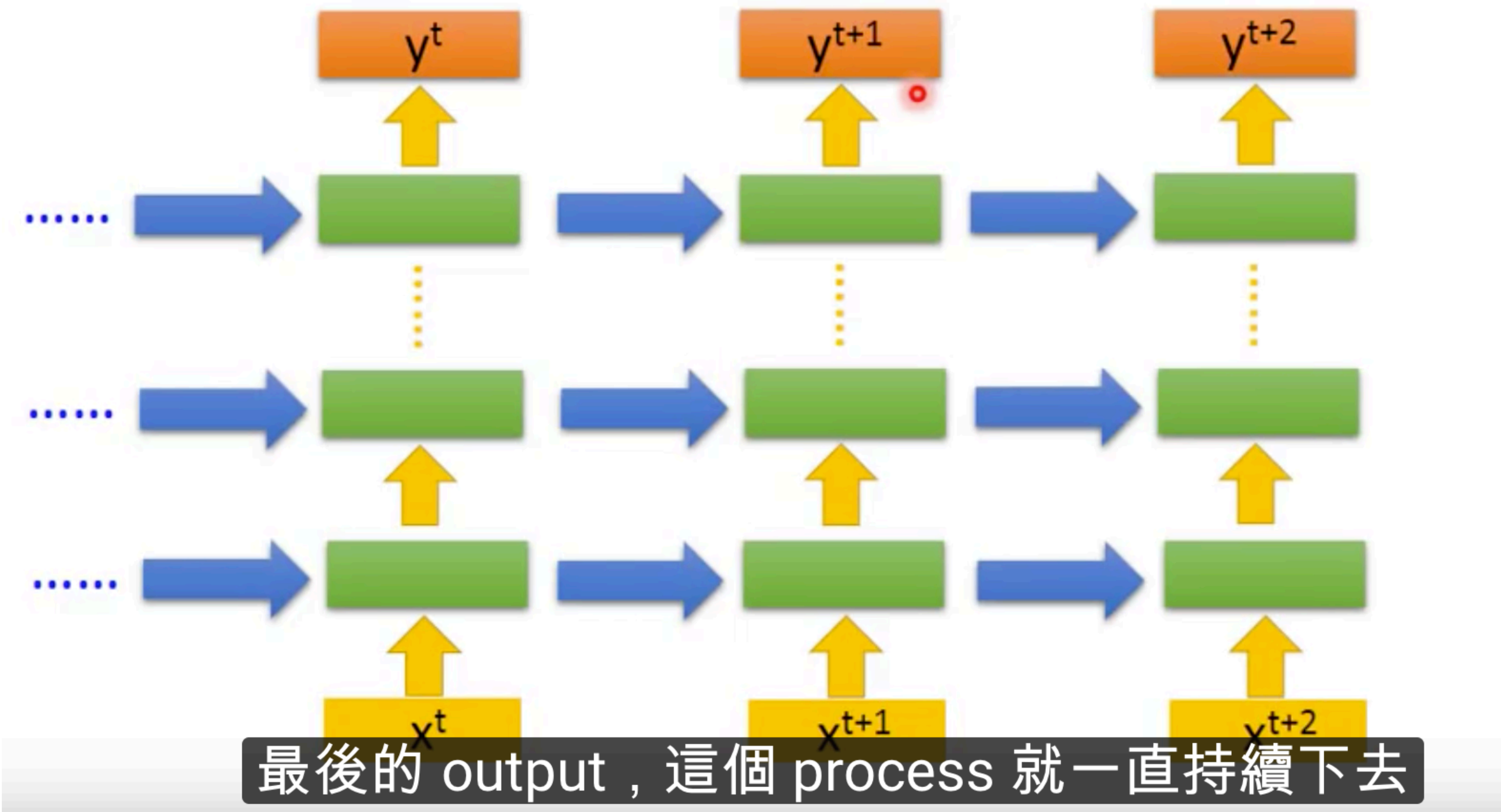


Of course it can be deep …

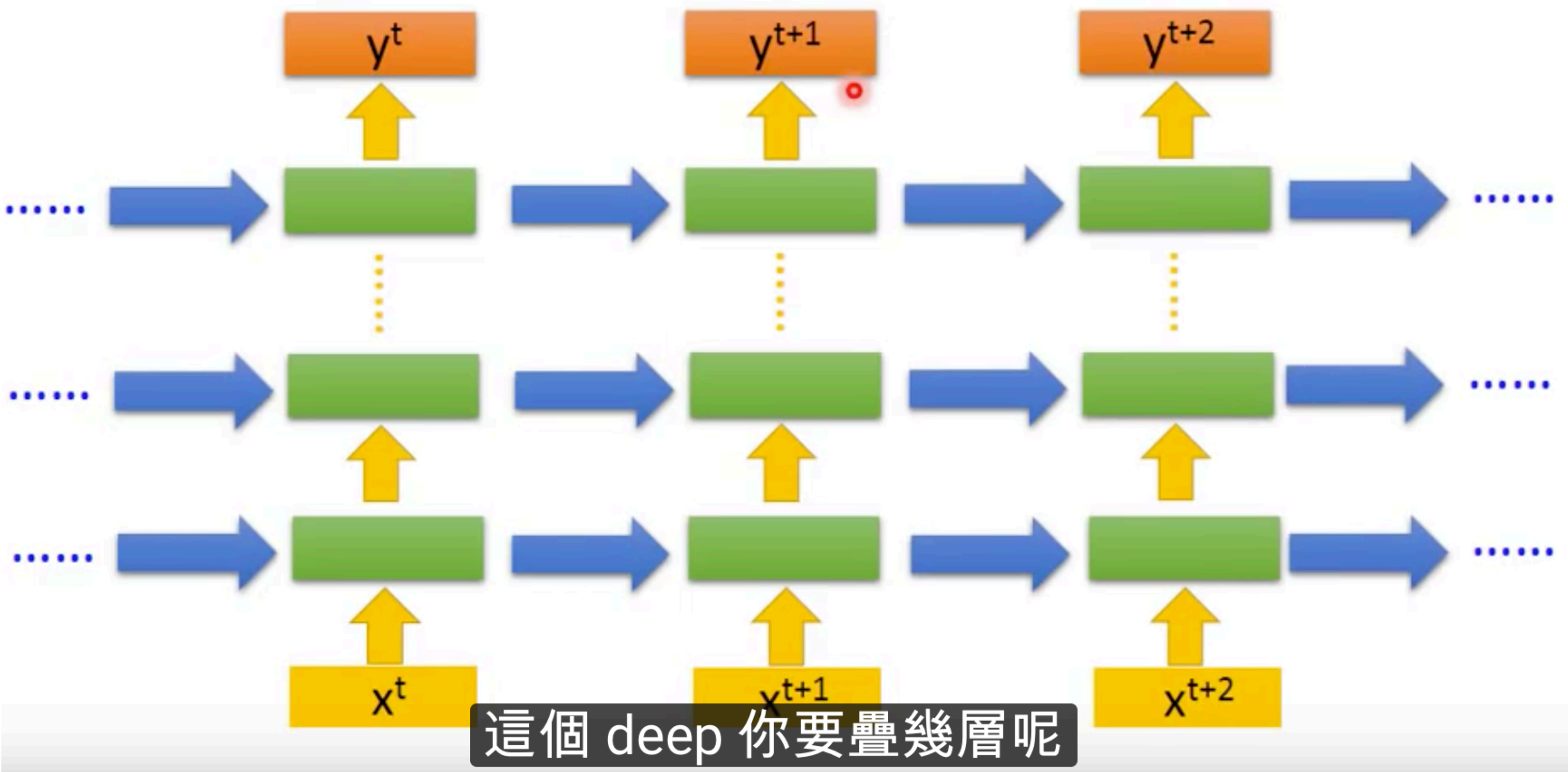


在下一個時間點的時候呢，每一個 hidden layer

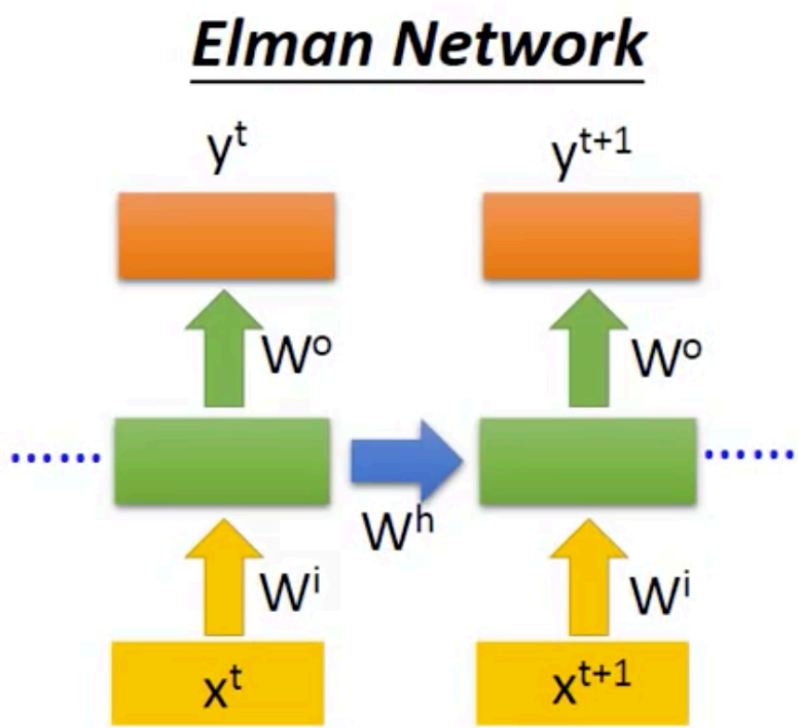
Of course it can be deep …



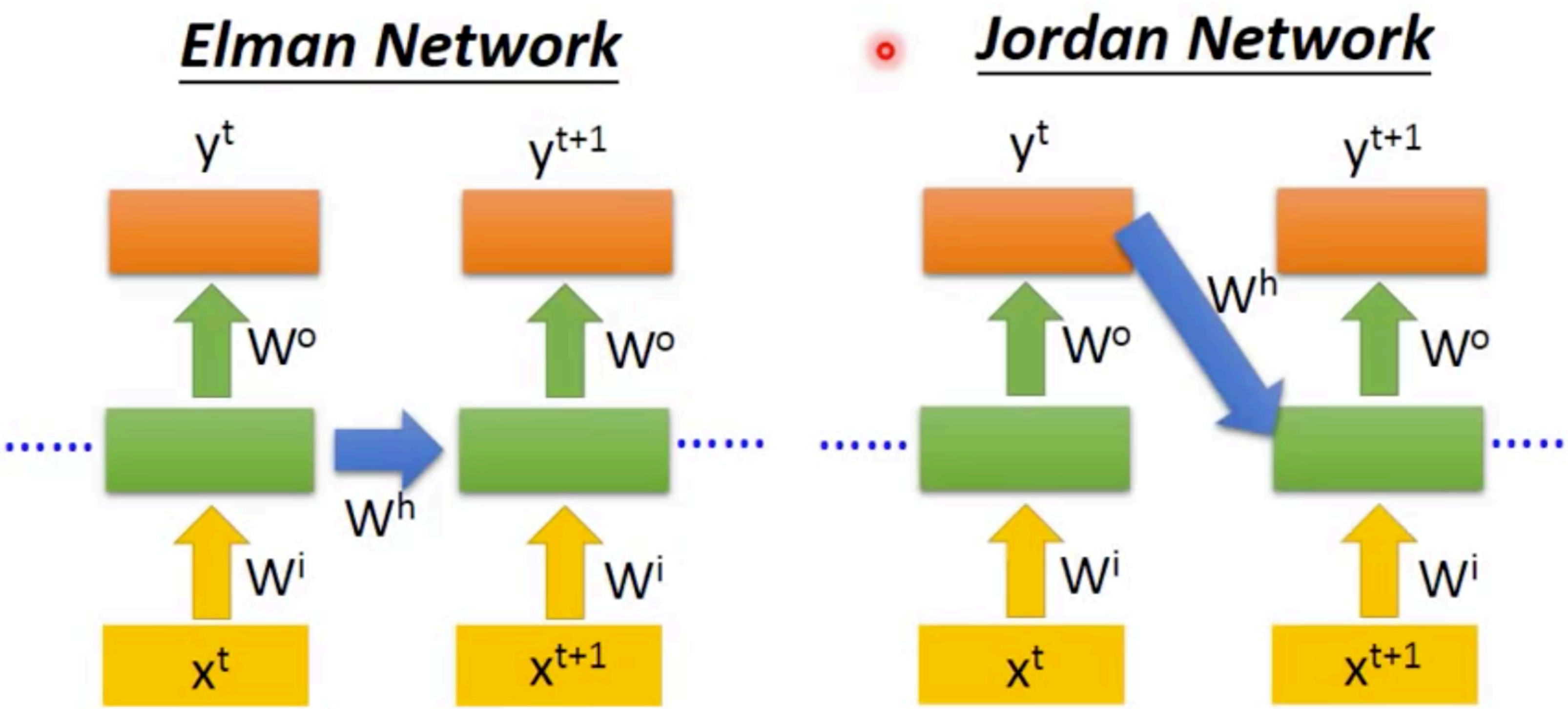
Of course it can be deep …



Elman Network & Jordan Network

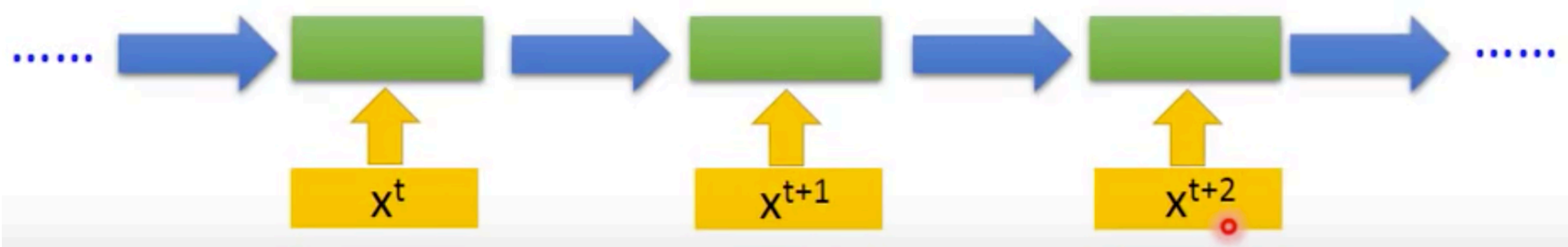


Elman Network & Jordan Network

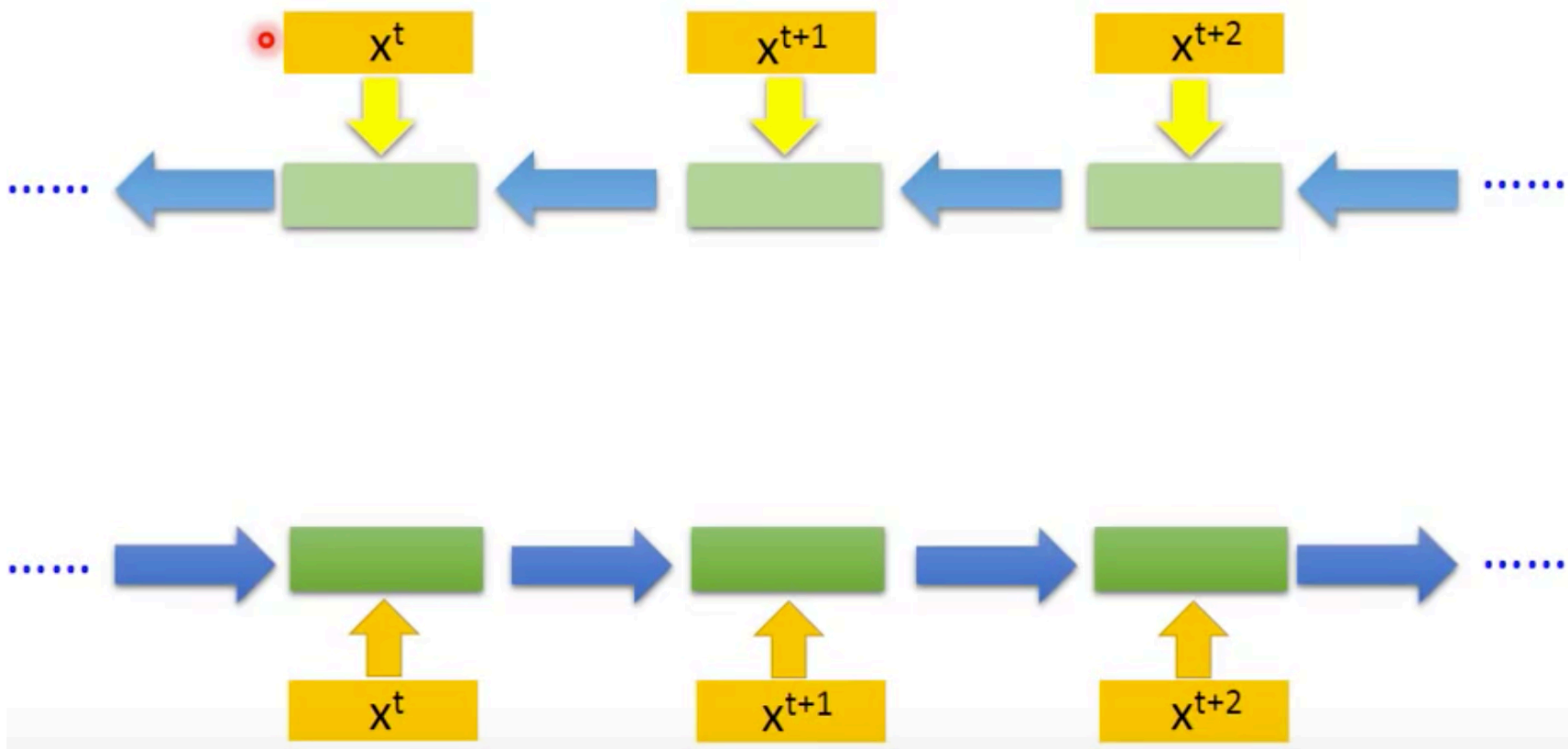


Bidirectional RNN

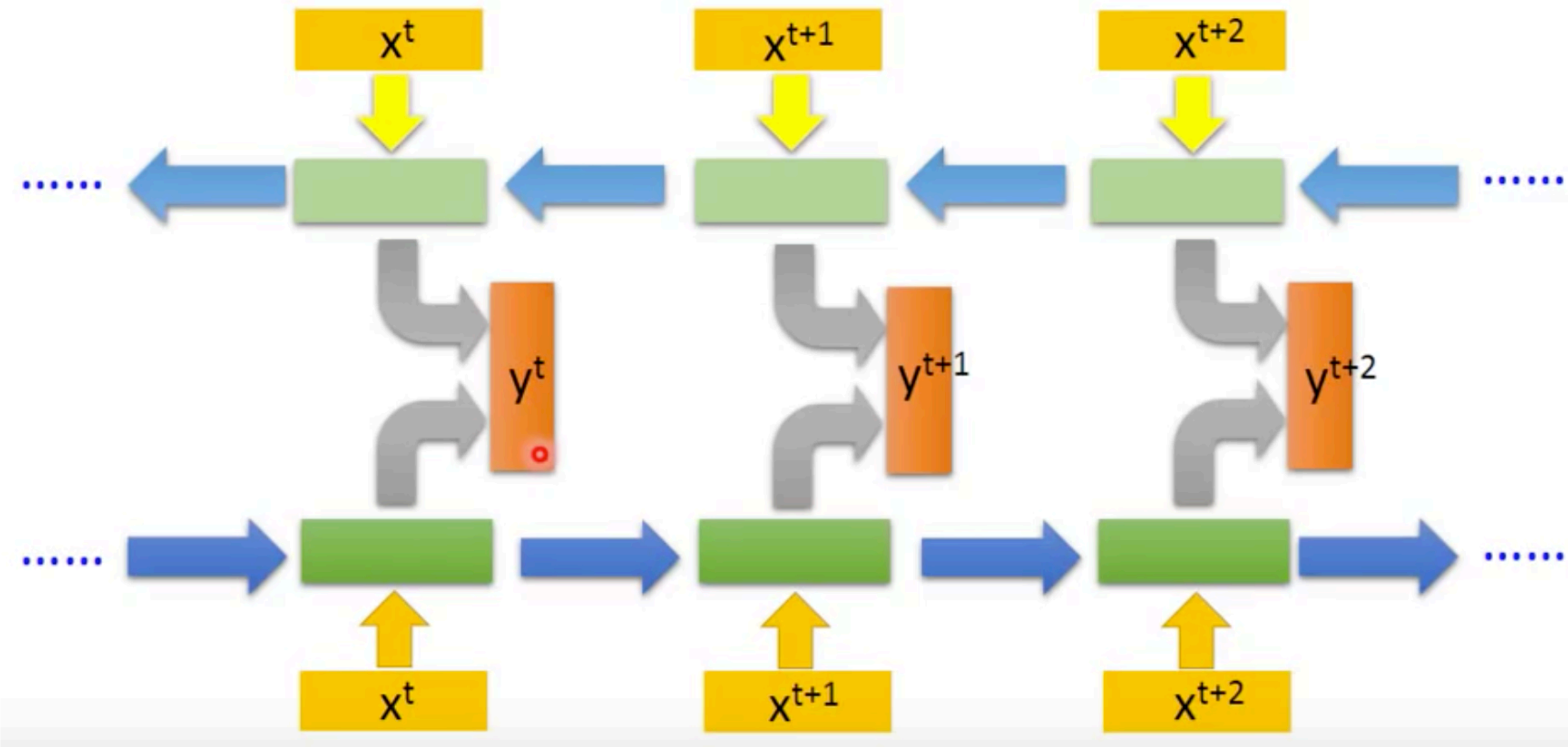
Bidirectional RNN



Bidirectional RNN

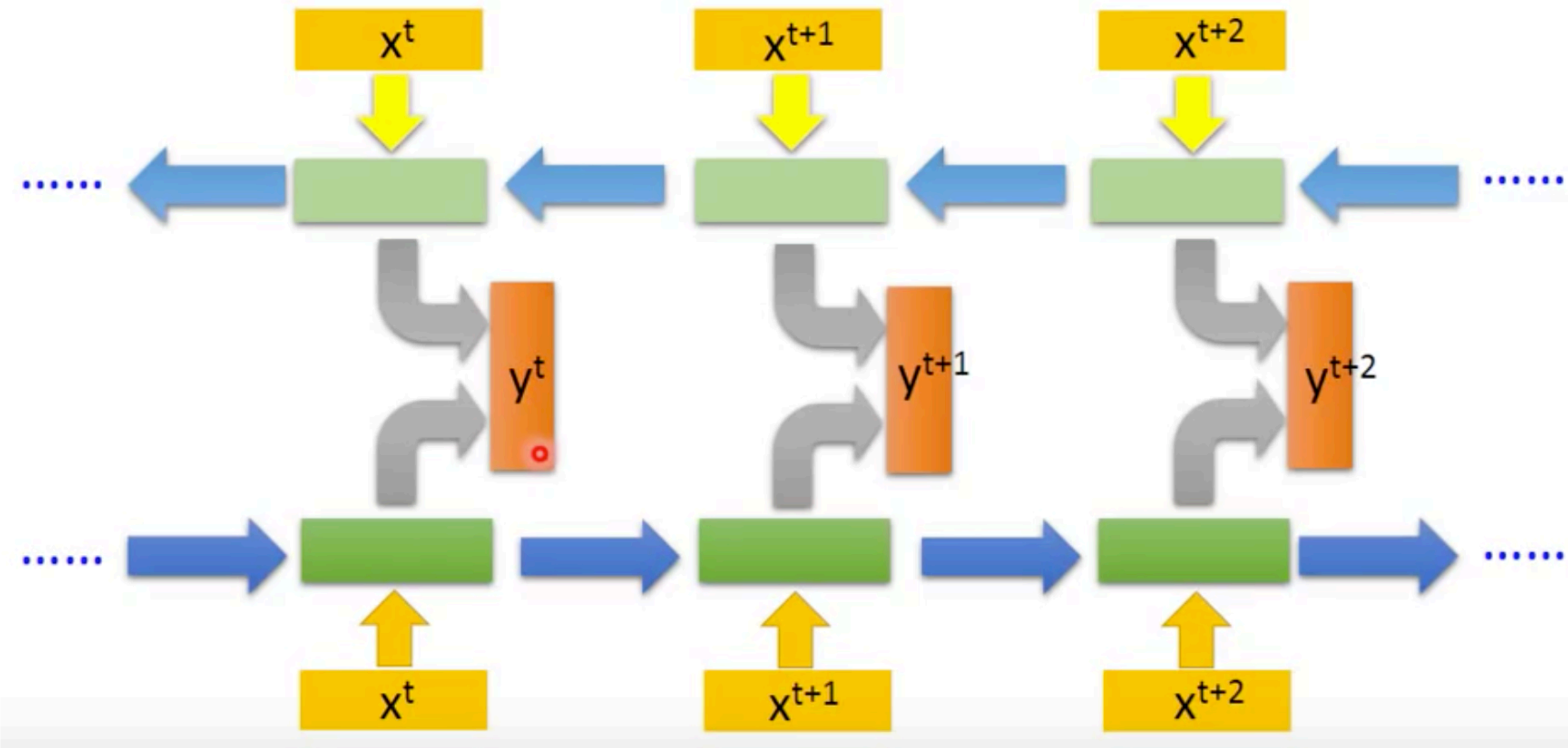


Bidirectional RNN



Bidirectional RNN

Benefit: every part of output considers the whole input sequence



The above is actually just a simple version of RNN (called SimpleRNN).

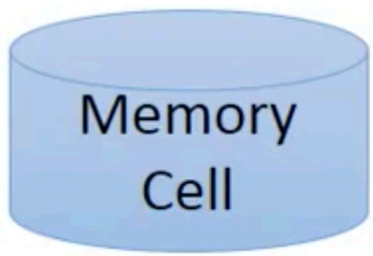
Issues with the SimpleRNN: training is difficult, due to issues including
"exploding gradient" or "vanishing gradient" in the gradient descent method.

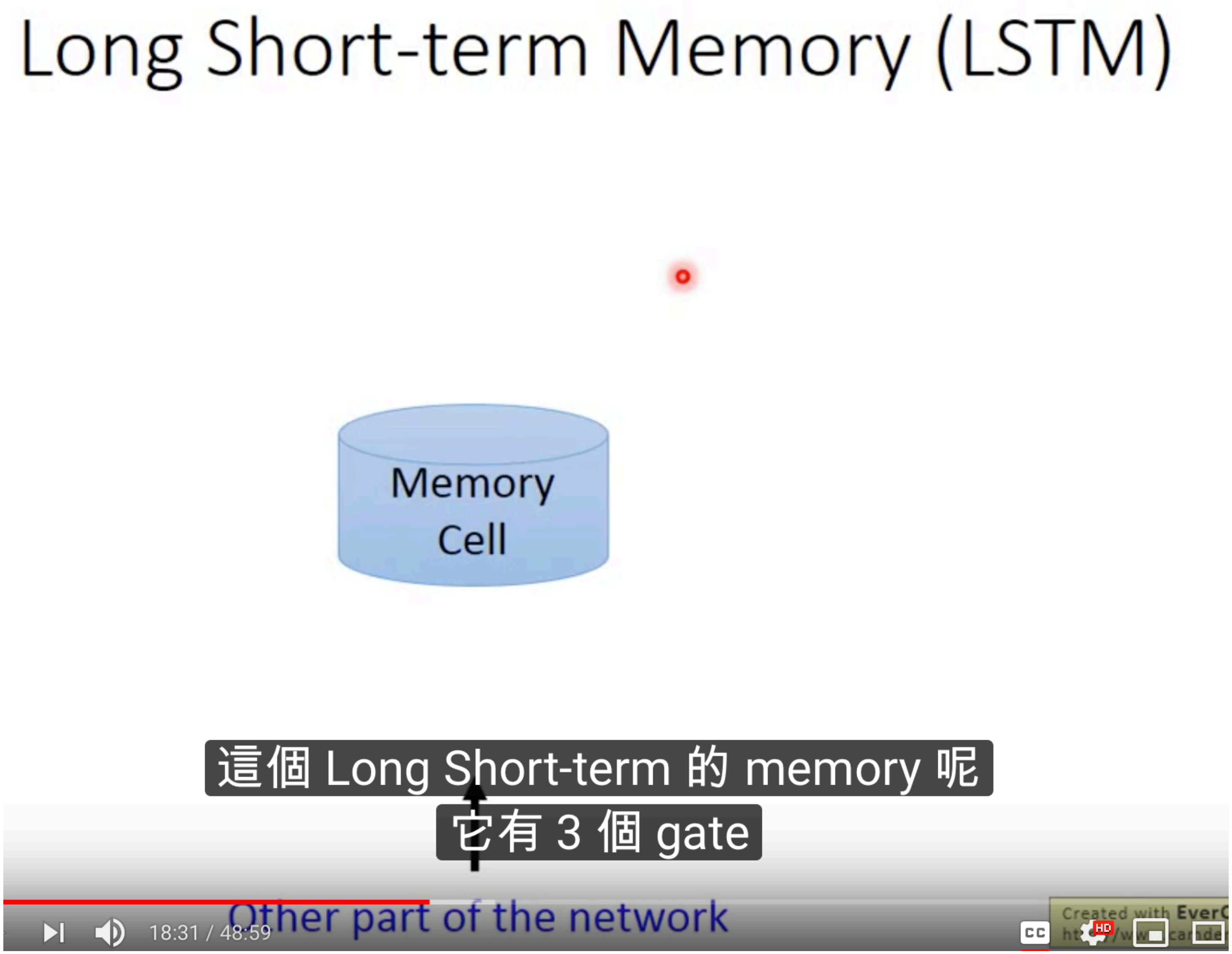
More advanced types of RNN:
LSTM, and GRU (a simpler version than LSTM).

When people use RNN, they mostly use LSTM or GRU.

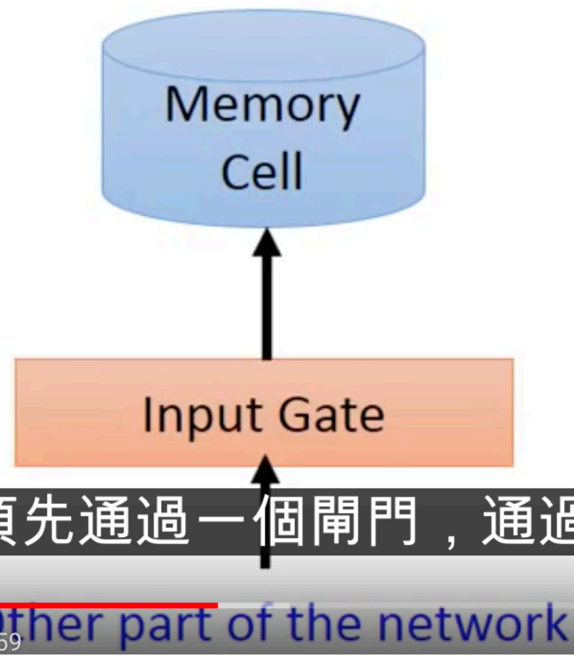
Keras let you create SimpleRNN, LSTM or GRU using just one line of code.

Long Short-term Memory (LSTM)



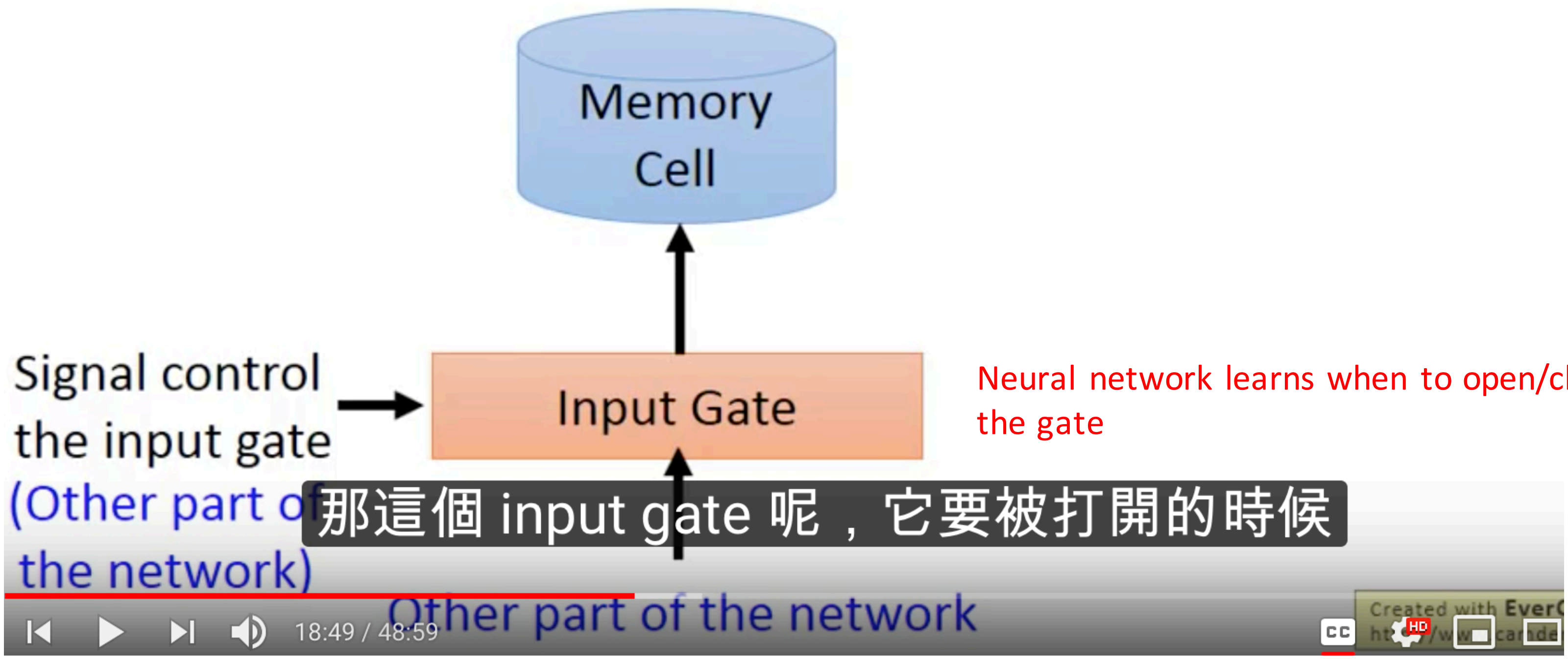

Long Short-term Memory (LSTM)
Memory
Cell
Other part of the network
這個 Long Short-term 的 memory 呢
它有 3 個 gate
18:31 / 48:59

Long Short-term Memory (LSTM)



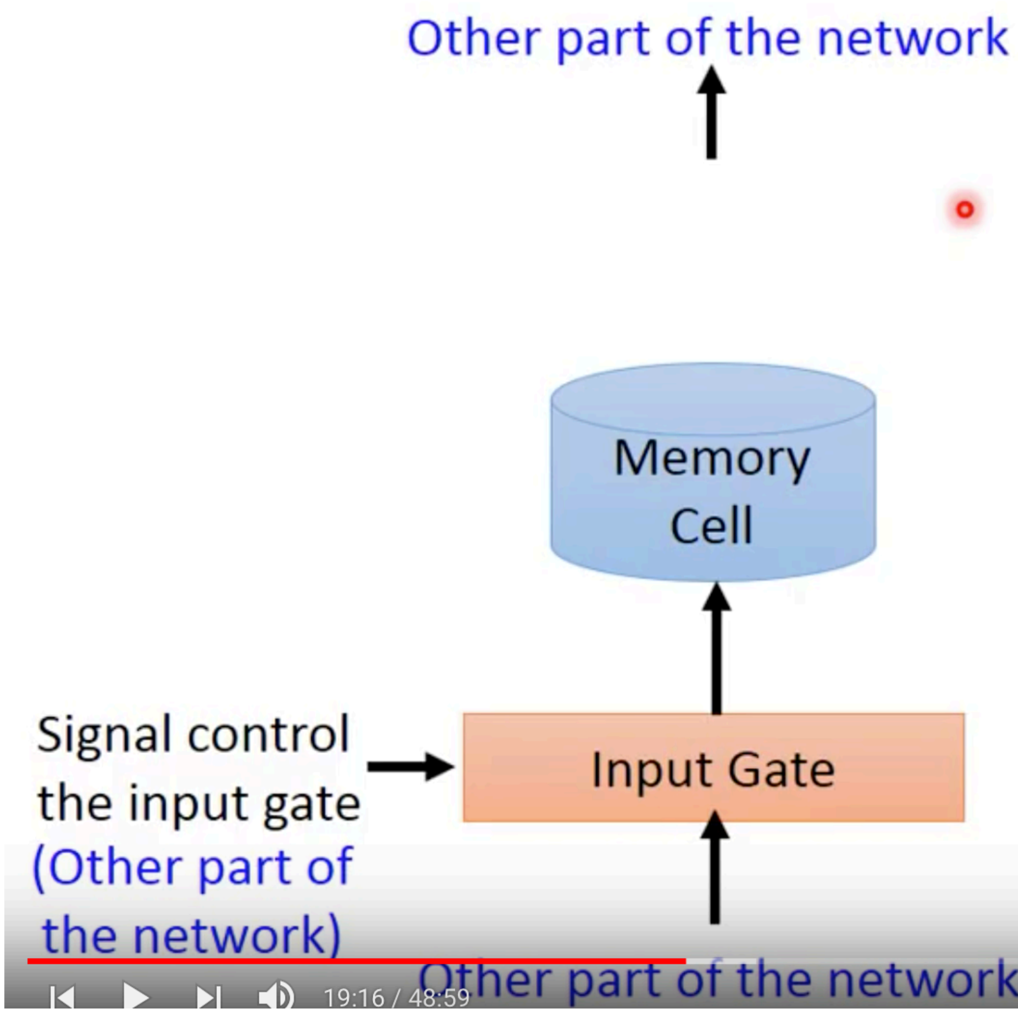
它必須先通過一個閘門，通過一個 input gate

Long Short-term Memory (LSTM)

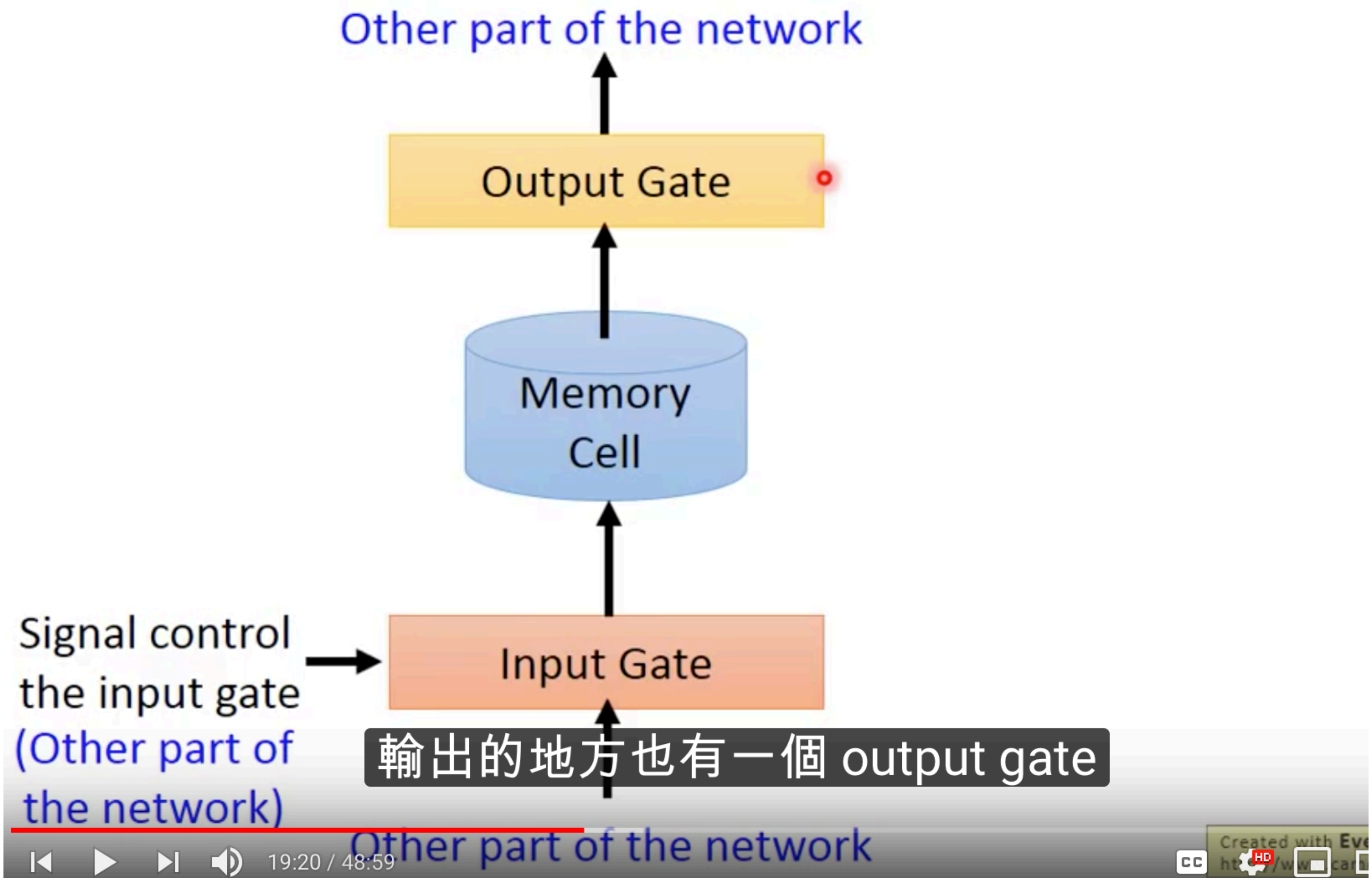


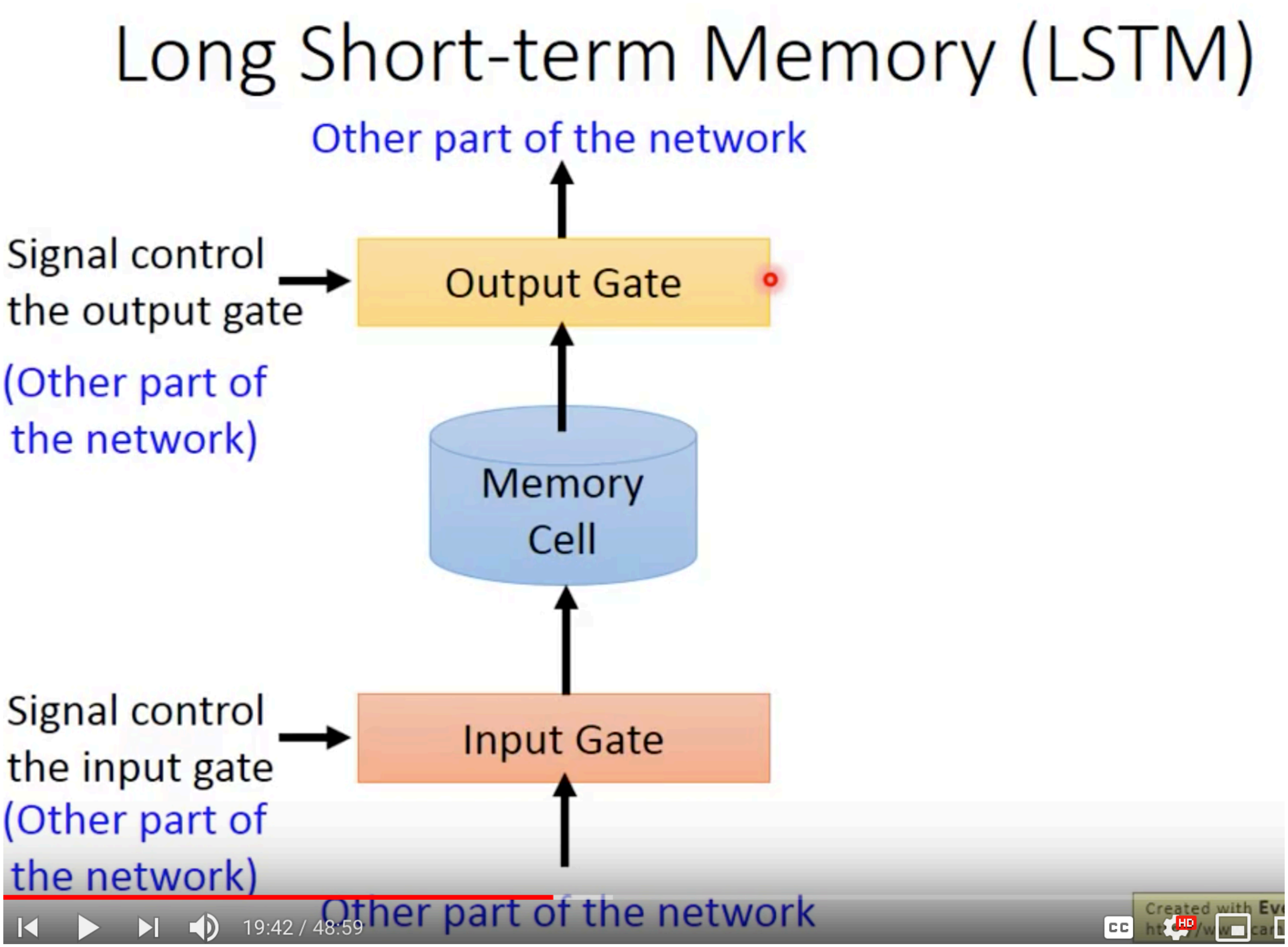
Neural network learns when to open/close the gate

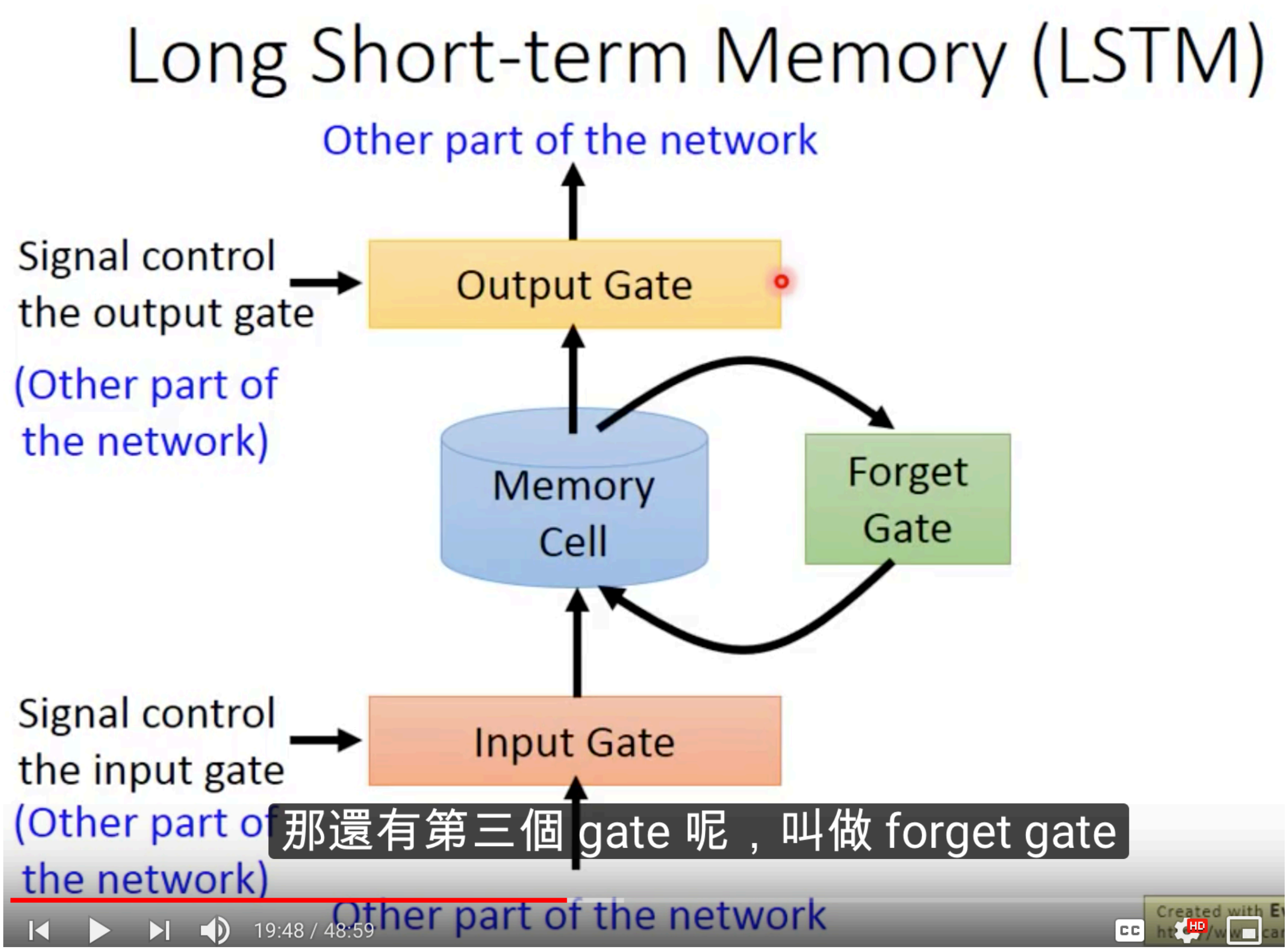
Long Short-term Memory (LSTM)



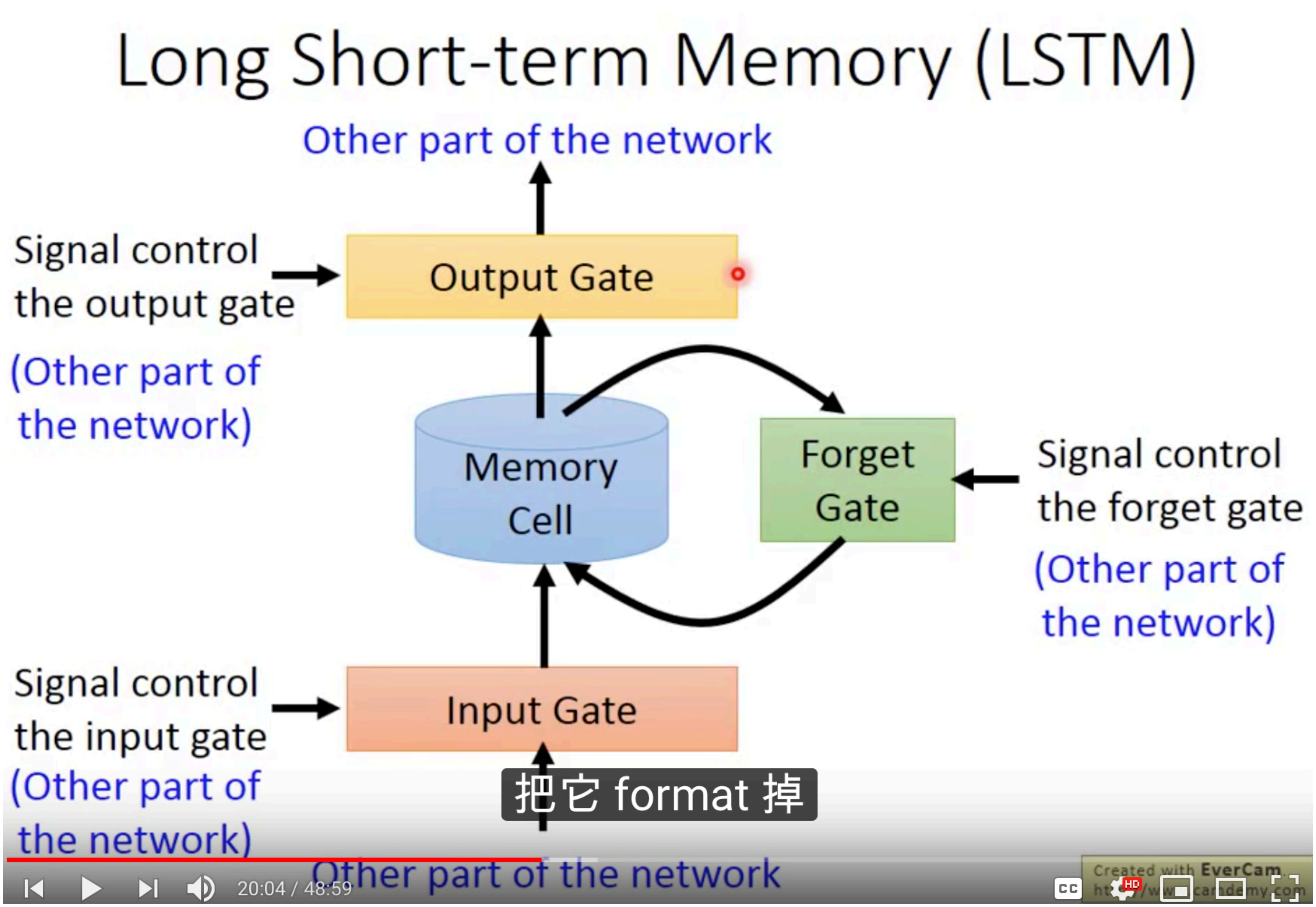
Long Short-term Memory (LSTM)

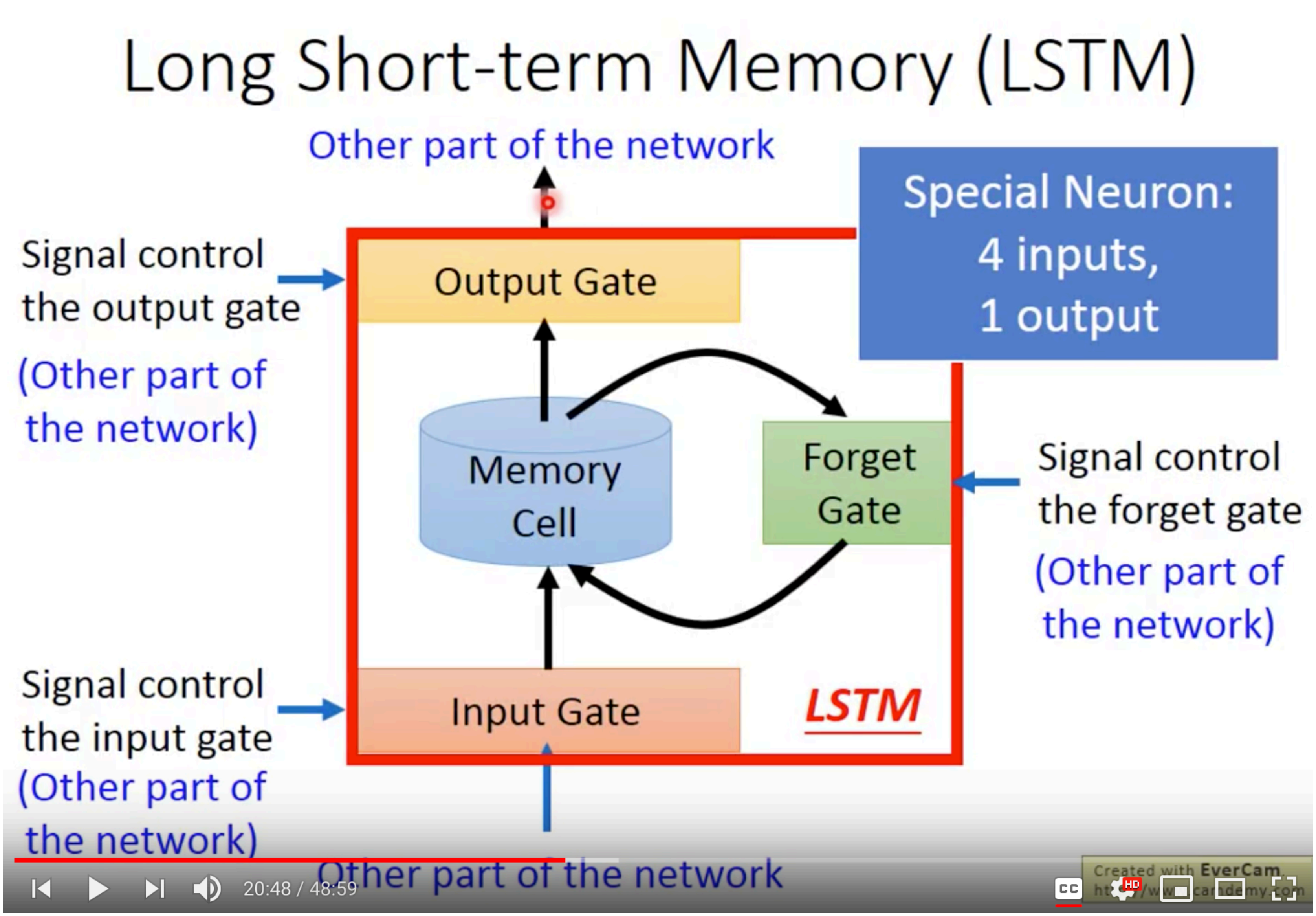



Long Short-term Memory (LSTM)
Other part of the network
Signal control the output gate
Output Gate
(Other part of the network)
Memory Cell
Signal control the input gate
Input Gate
(Other part of the network)
Other part of the network
19:42 / 48:59



Long Short-term Memory (LSTM)
Other part of the network
Signal control the output gate
(Other part of the network)
Output Gate
Memory Cell
Forget Gate
Signal control the input gate
(Other part of the network)
Input Gate
Other part of the network
那還有第三個 gate 呢，叫做 forget gate
19:48 / 48:59

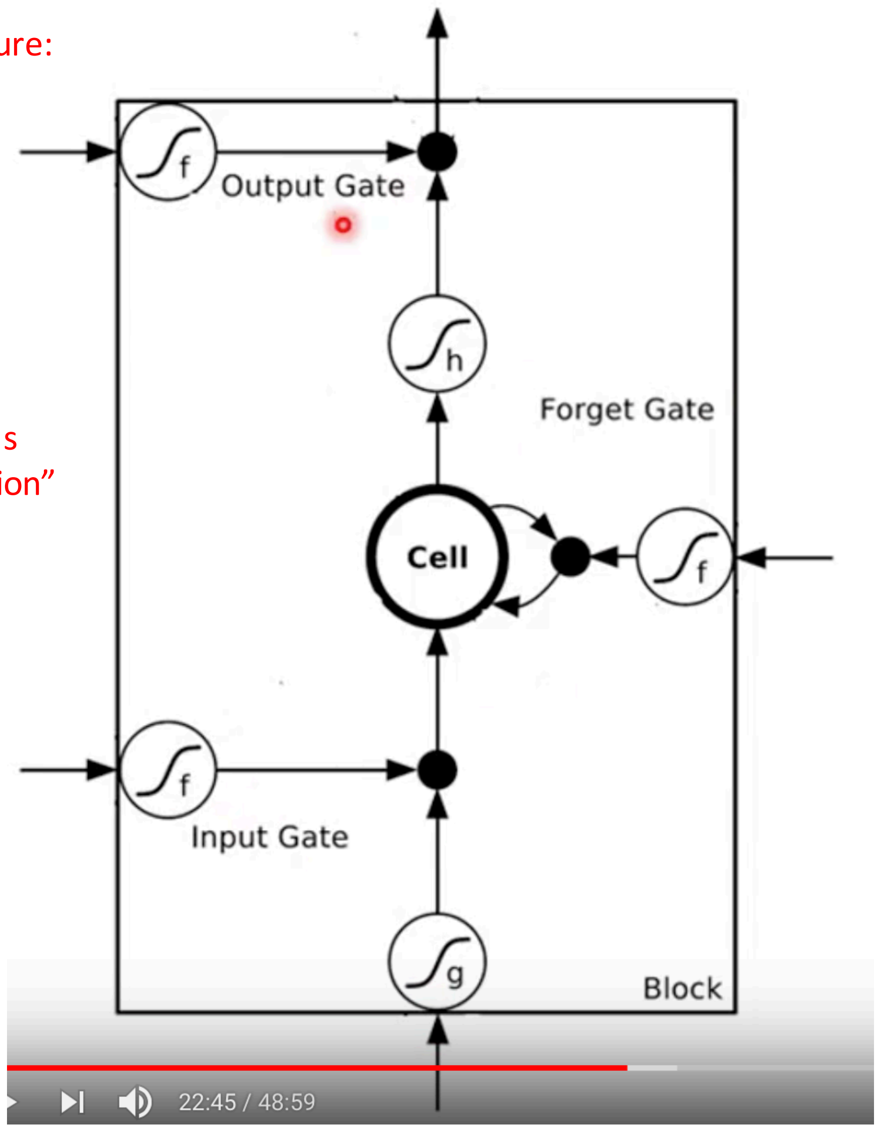

Long Short-term Memory (LSTM)
Other part of the network
Signal control the output gate
(Other part of the network)
Output Gate
Memory Cell
Forget Gate
Signal control the forget gate
(Other part of the network)
Signal control the input gate
(Other part of the network)
Input Gate
Other part of the network
把它 format 掉
20:04 / 48:59
Created with EverCam

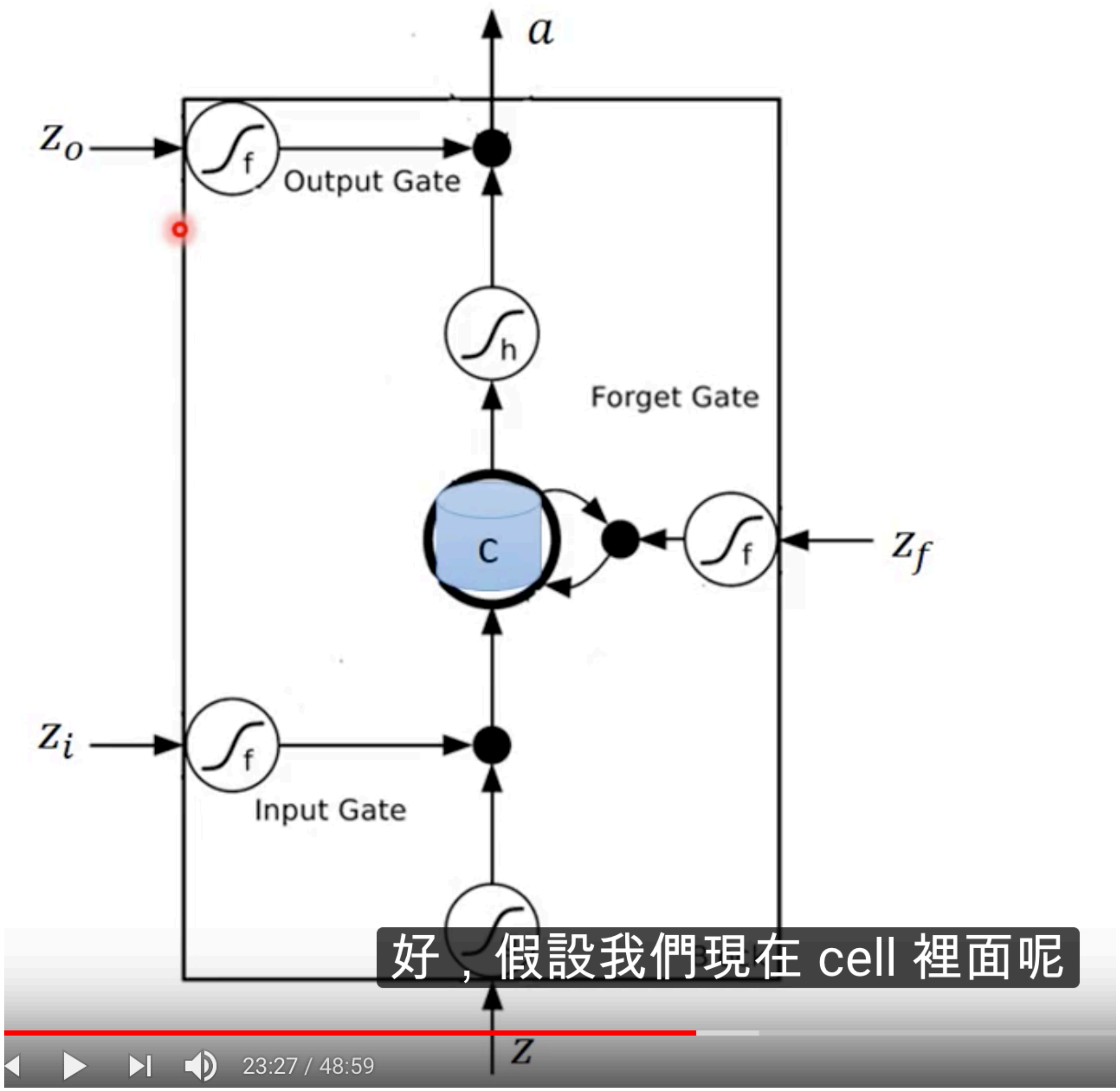

Long Short-term Memory (LSTM)
Other part of the network
Special Neuron:
4 inputs,
1 output
Signal control the output gate
(Other part of the network)
Output Gate
Memory Cell
Forget Gate
Signal control the forget gate
(Other part of the network)
Signal control the input gate
(Other part of the network)
Input Gate
LSTM
Other part of the network
20:48 / 48:59
Created with EverCam

A more detailed look at its structure:

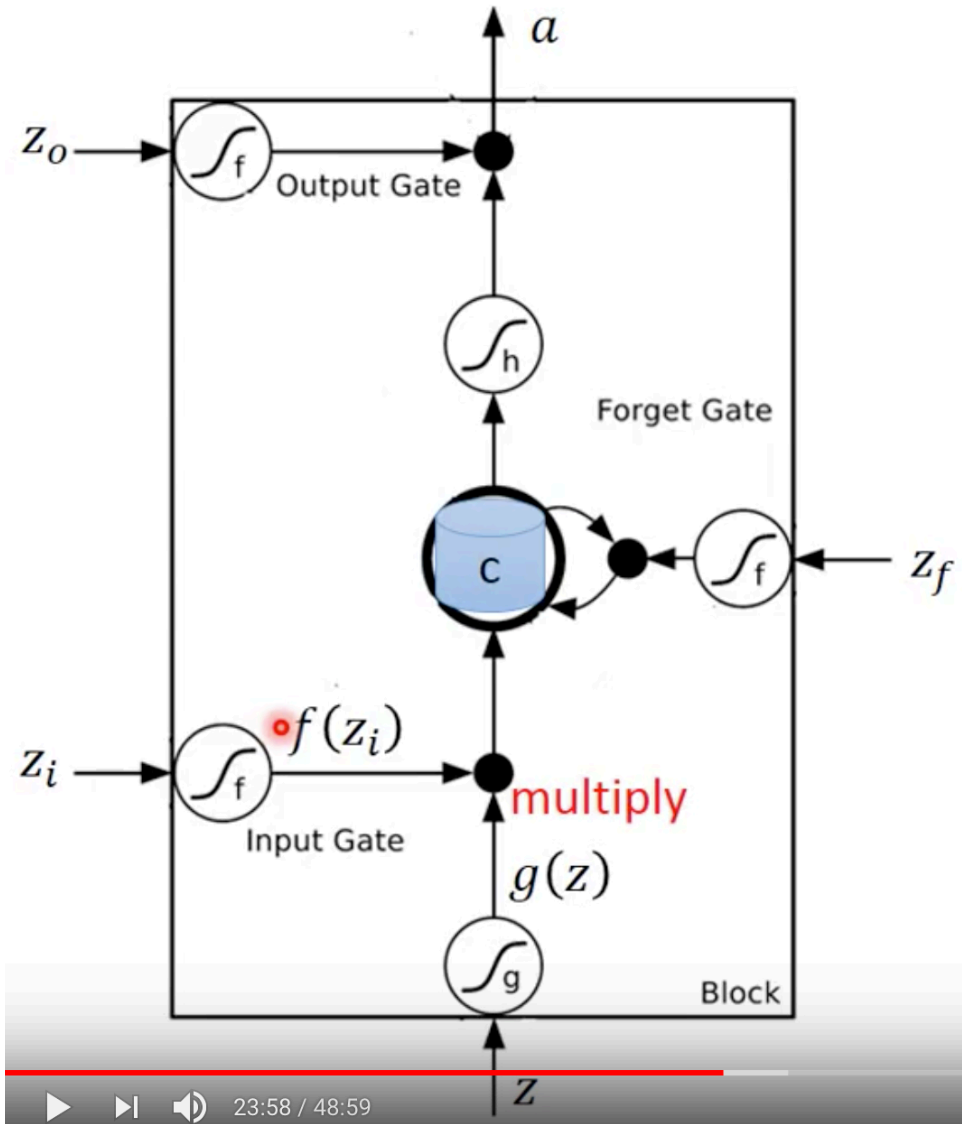
Every input and output here is a real number.

The whole thing can be seen as replacing the "activation function" of an ordinary neuron.

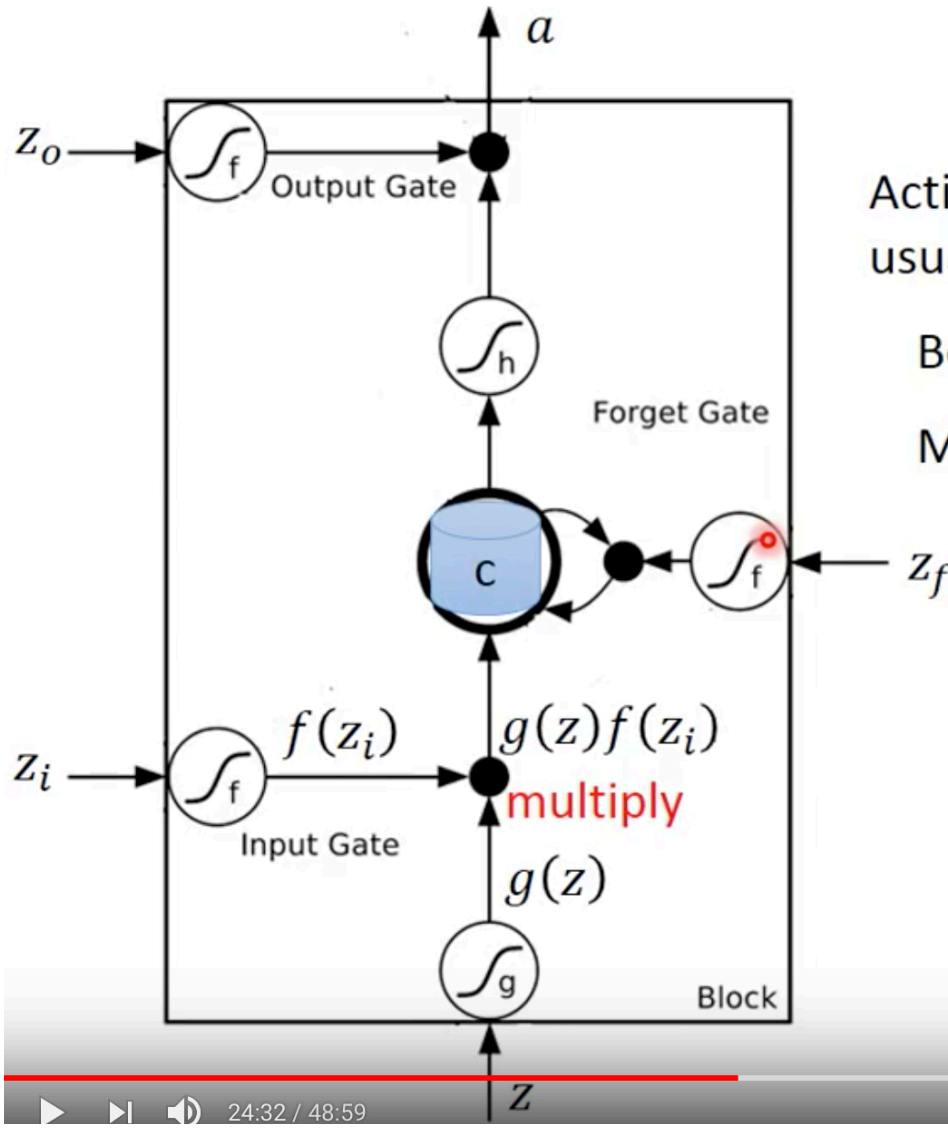




a
z_o
f
Output Gate
h
Forget Gate
c
f
z_f
z_i
f
Input Gate
z
好，假設我們現在 cell 裡面呢
23:27 / 48:59



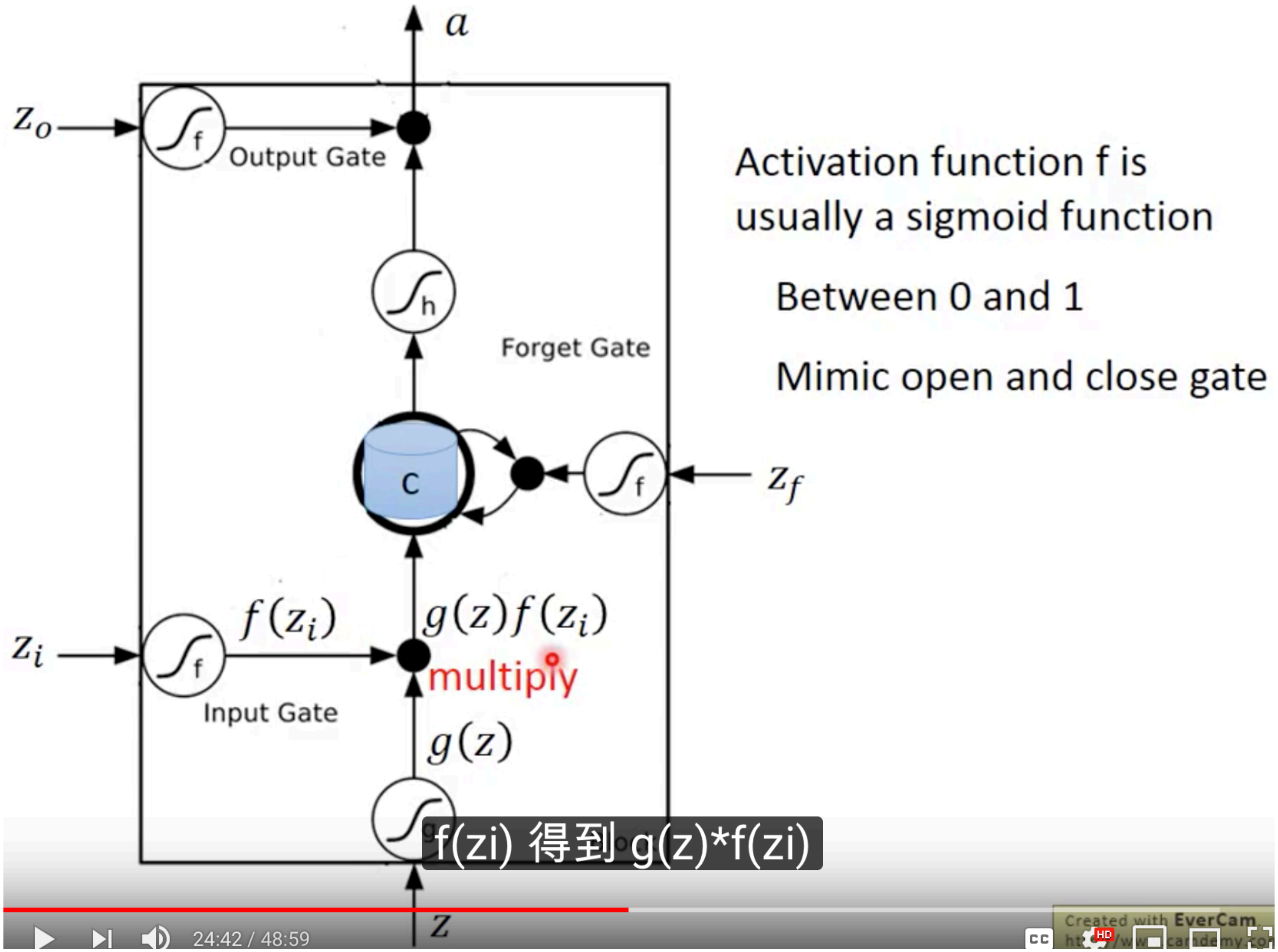
a
z_o
Output Gate
Forget Gate
c
z_f
f(z_i)
z_i
multiply
Input Gate
g(z)
Block
z
23:58 / 48:59

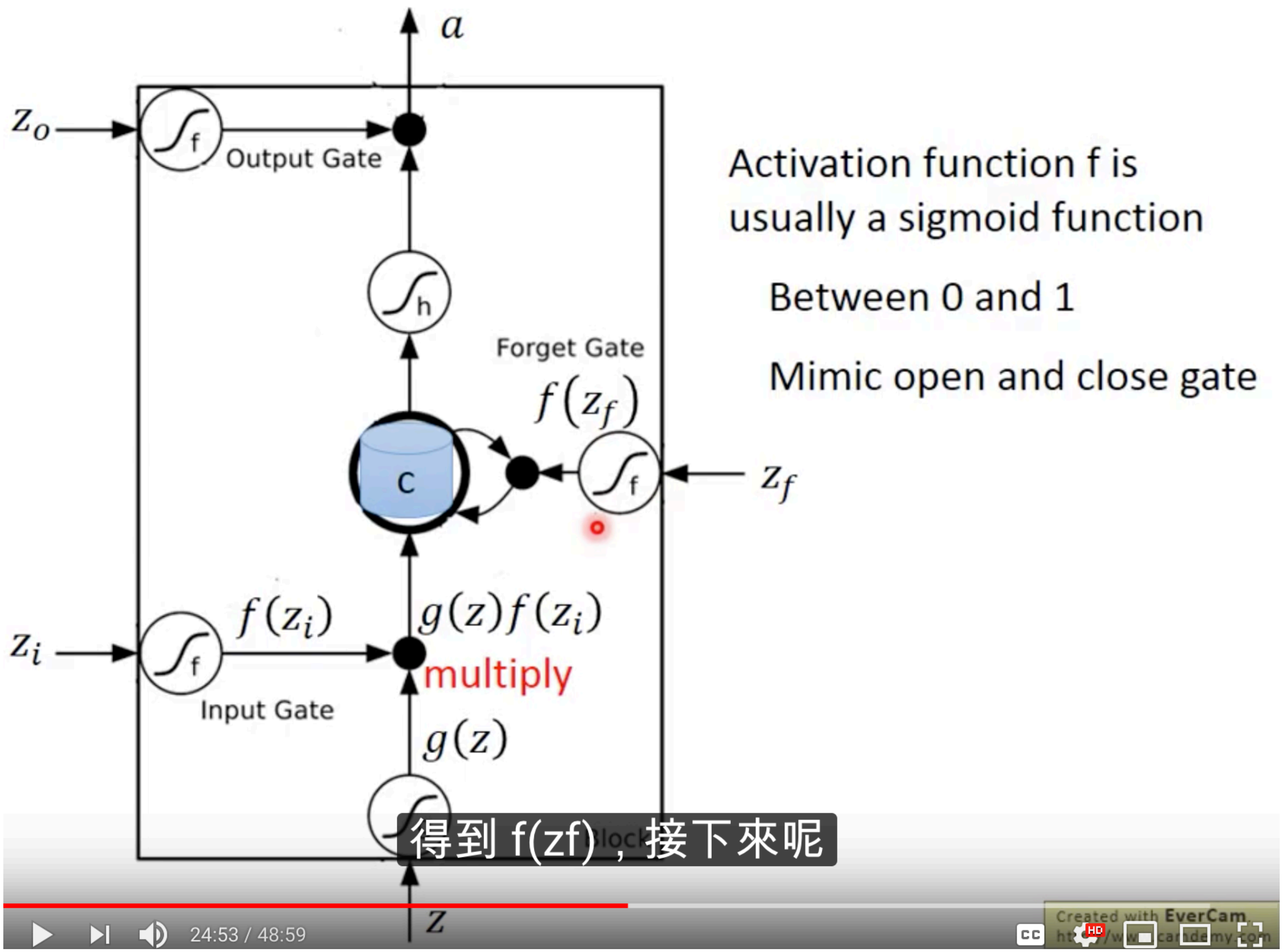


Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

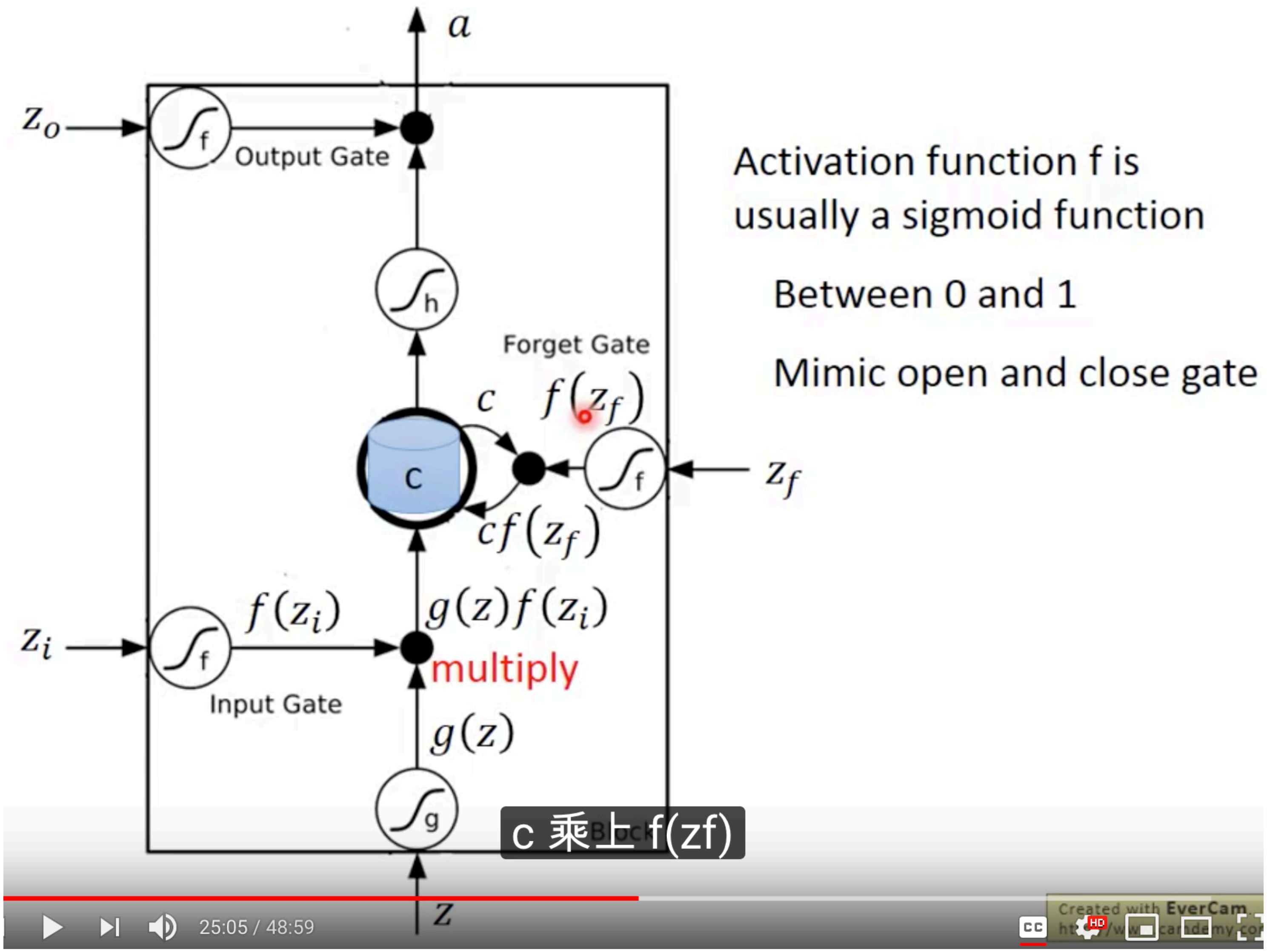

a
z_o
Output Gate
Activation function f is
usually a sigmoid function
Between 0 and 1
Mimic open and close gate
h
Forget Gate
c
z_f
$f(z_i)$
$g(z)f(z_i)$
z_i
multiply
Input Gate
$g(z)$
f(zi) 得到 g(z)*f(zi)
z
24:42 / 48:59
Created with EverCam



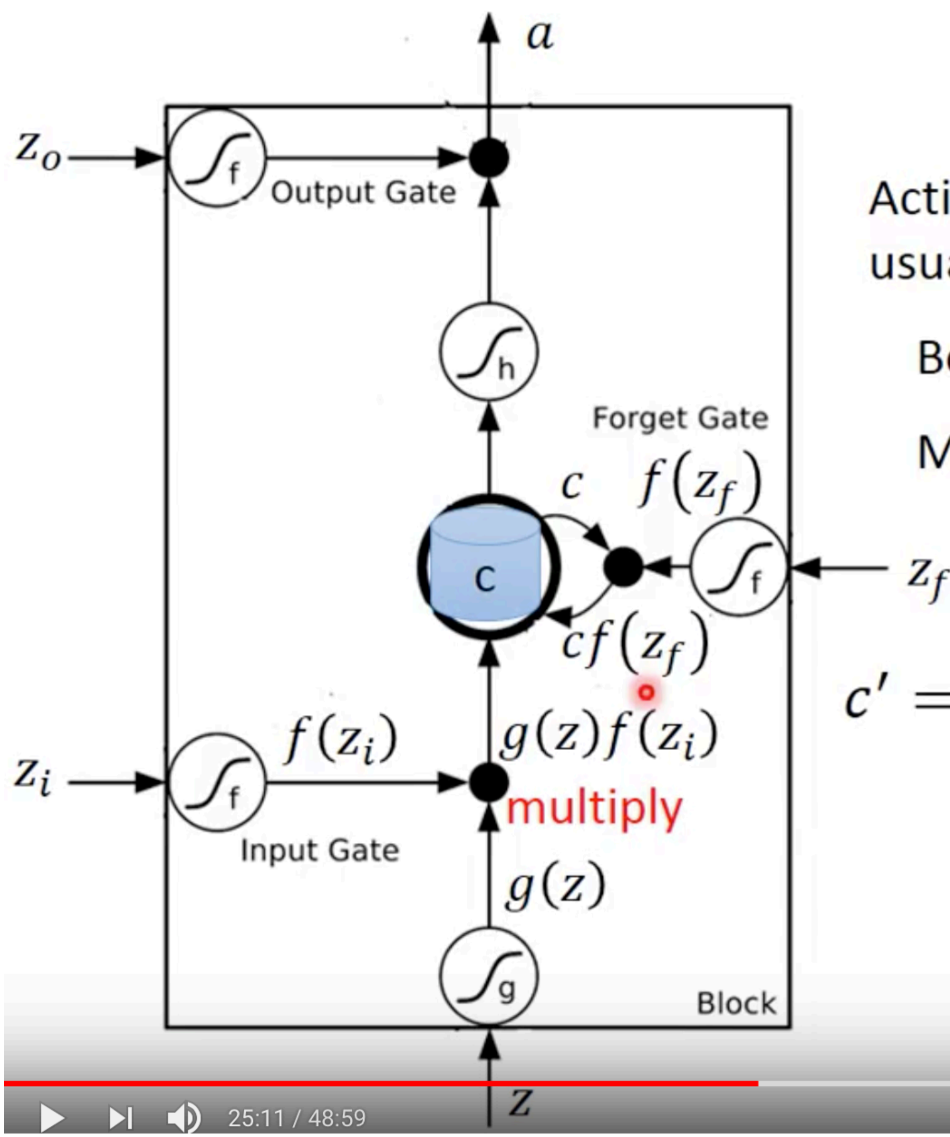
Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate



a
z_o
Output Gate
Forget Gate
c
$f(z_f)$
c
z_f
$cf(z_f)$
z_i
$f(z_i)$
$g(z)f(z_i)$
multiply
Input Gate
$g(z)$
z
Activation function f is usually a sigmoid function
Between 0 and 1
Mimic open and close gate
c 乘上 f(zf)
25:05 / 48:59
Created with EverCam

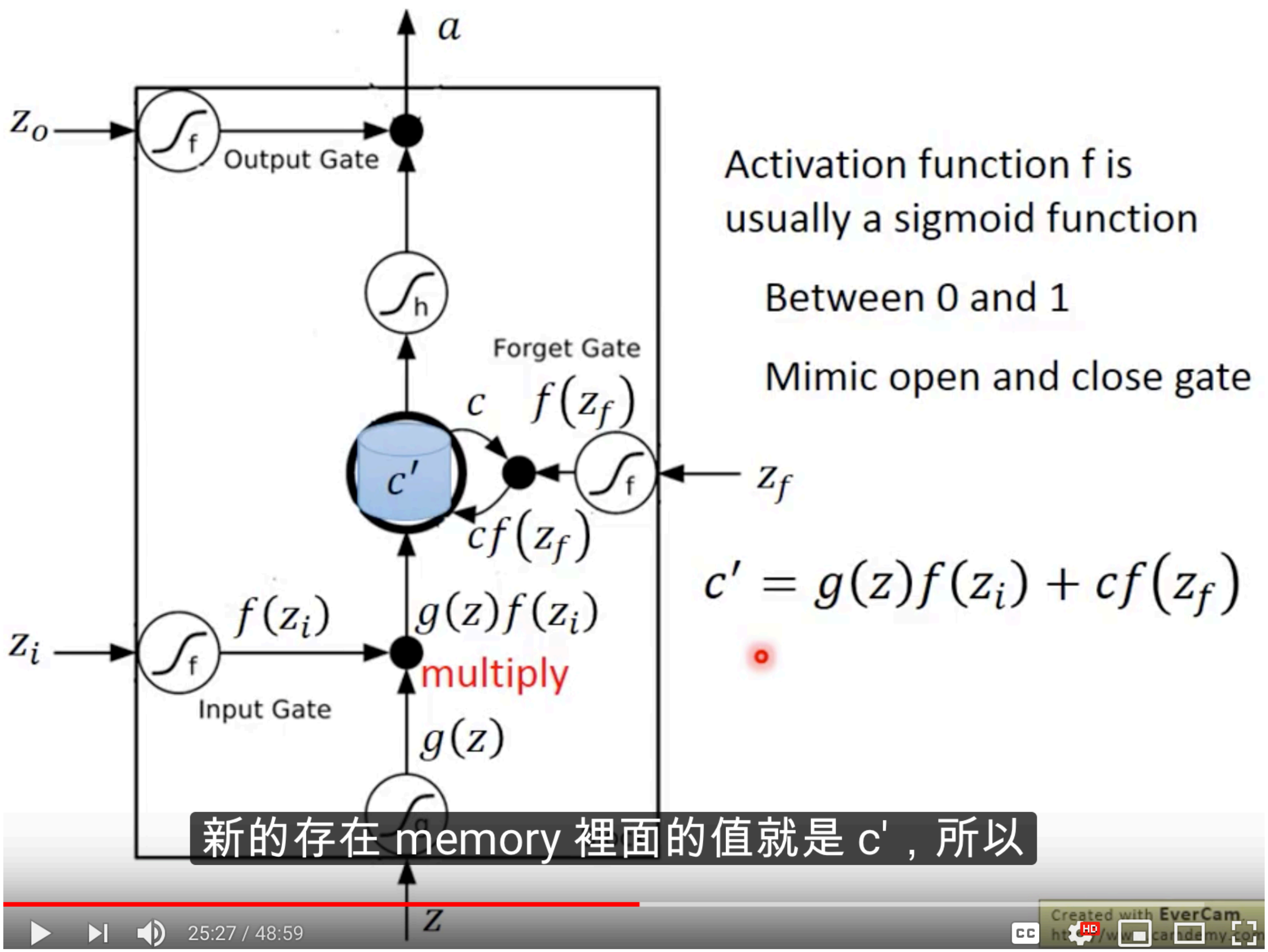


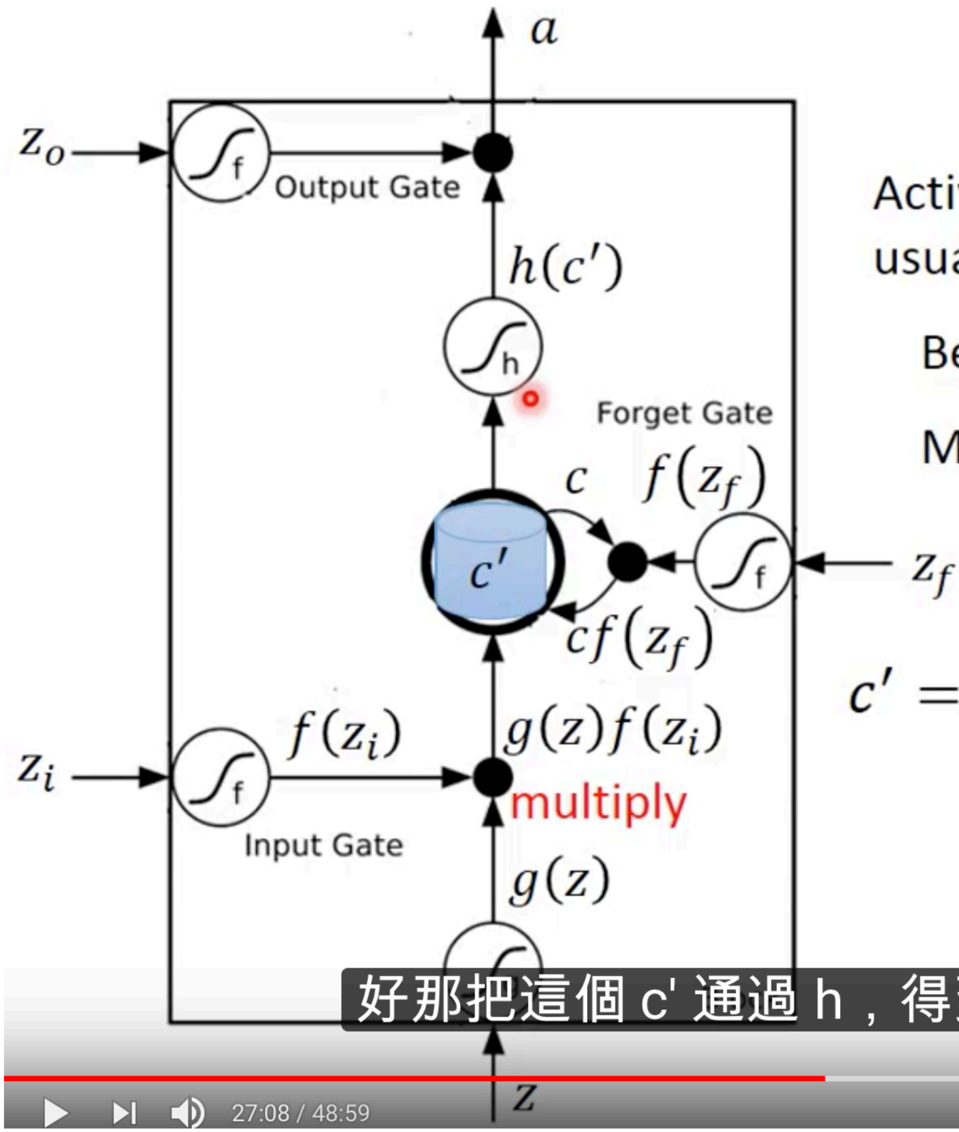
Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

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


a
z_o
Output Gate
Activation function f is
usually a sigmoid function
Between 0 and 1
Mimic open and close gate
Forget Gate
c
$f(z_f)$
c'
z_f
$cf(z_f)$
$c' = g(z)f(z_i) + cf(z_f)$
z_i
$f(z_i)$
$g(z)f(z_i)$
multiply
Input Gate
$g(z)$
z
新的存在 memory 裡面的值就是 c'，所以
25:27 / 48:59
Created with EverCam



Activation function f is usually a sigmoid function

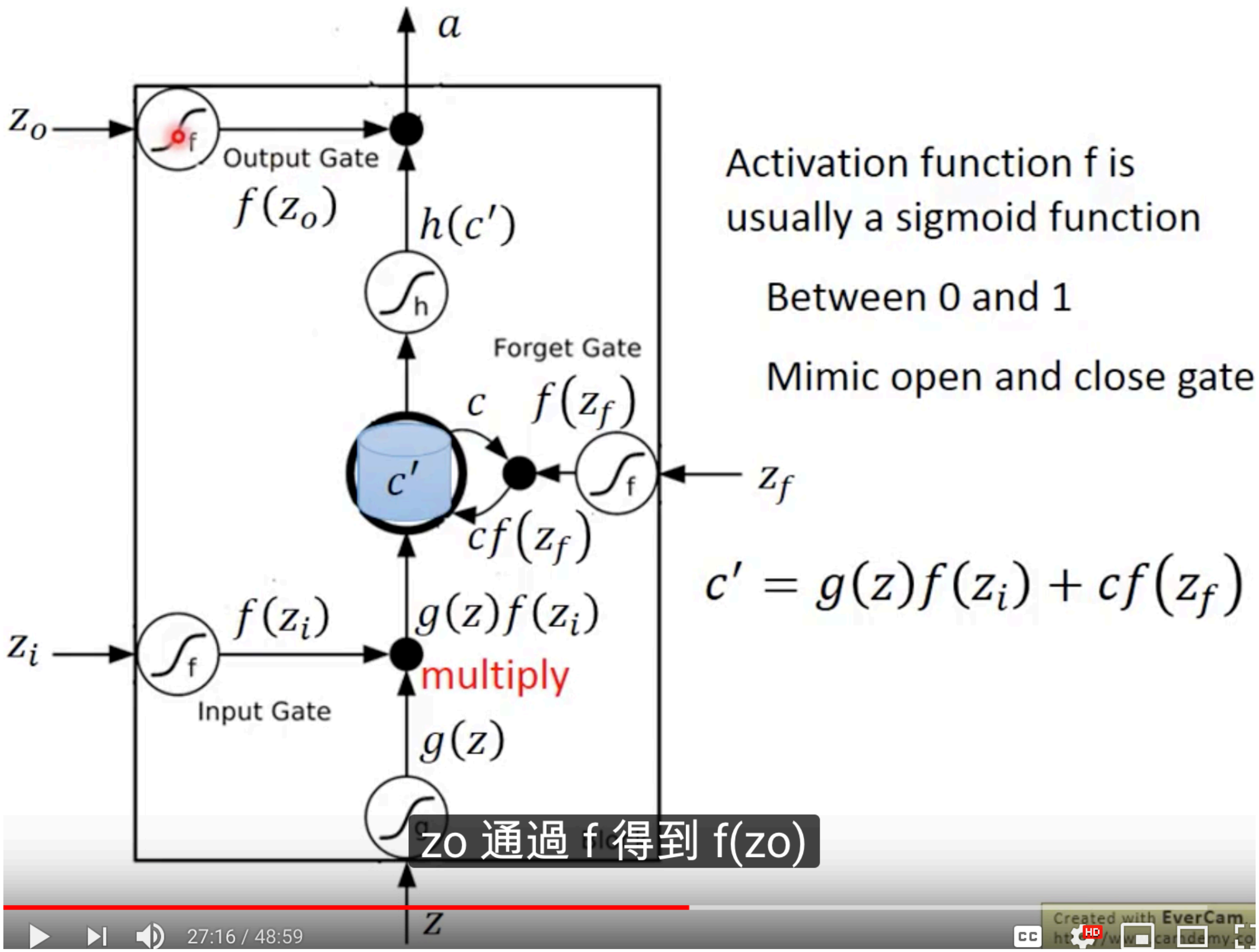
Between 0 and 1

Mimic open and close gate

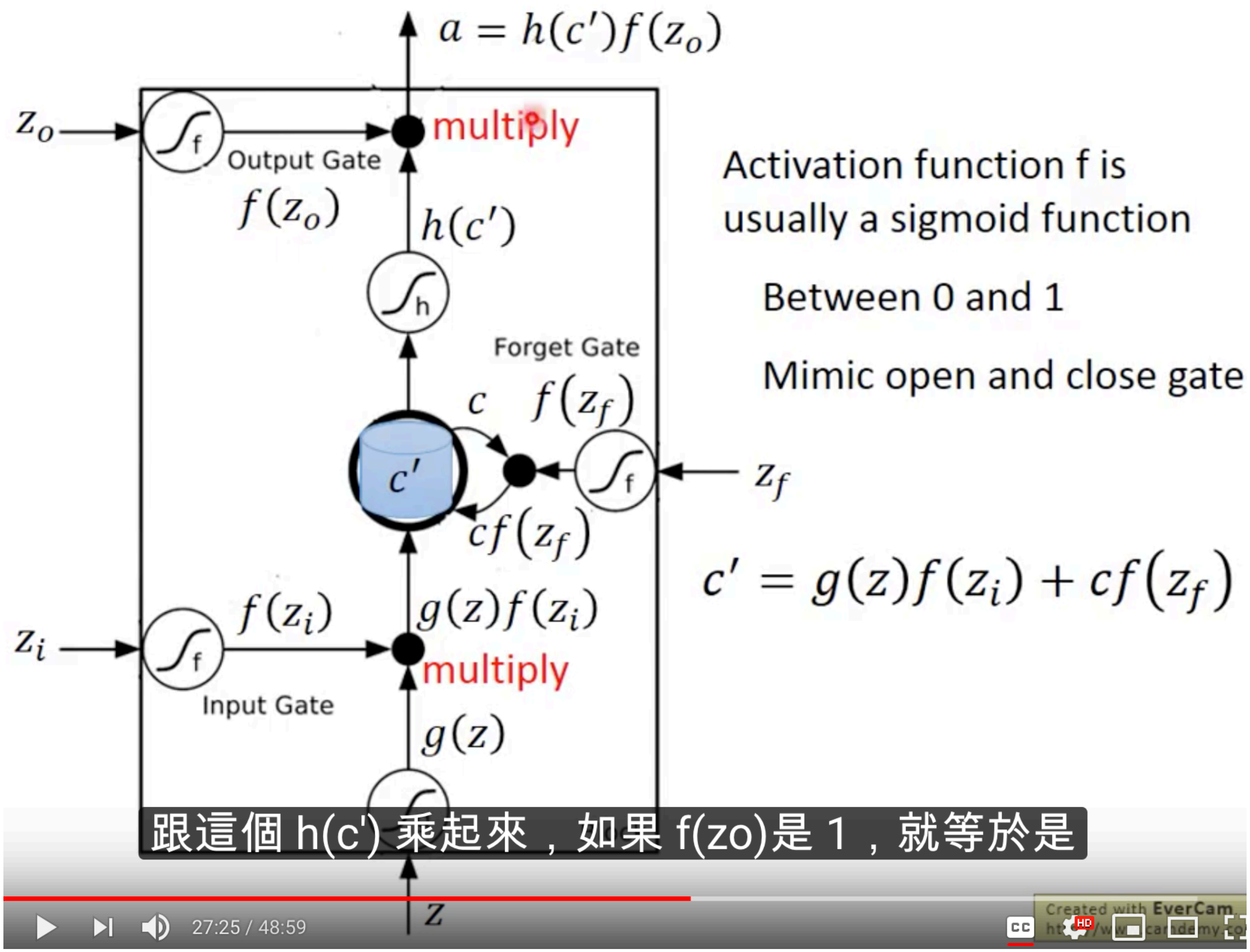
$$c' = g(z)f(z_i) + cf(z_f)$$

好那把這個 c' 通過 h，得到 h(c')

27:08 / 48:59



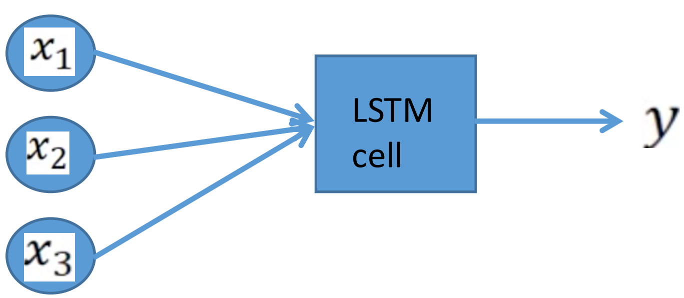
a
z_o
Output Gate
$f(z_o)$
$h(c')$
Forget Gate
c
$f(z_f)$
c'
z_f
$cf(z_f)$
z_i
$f(z_i)$
$g(z)f(z_i)$
multiply
Input Gate
$g(z)$
z
Activation function f is usually a sigmoid function
Between 0 and 1
Mimic open and close gate
$c' = g(z)f(z_i) + cf(z_f)$
zo 通過 f 得到 f(zo)
27:16 / 48:59
Created with EverCam


$a = h(c')f(z_o)$
z_o
Output Gate
$f(z_o)$
multiply
$h(c')$
Forget Gate
c
$f(z_f)$
c'
z_f
$cf(z_f)$
z_i
$f(z_i)$
$g(z)f(z_i)$
multiply
Input Gate
$g(z)$
z
Activation function f is usually a sigmoid function
Between 0 and 1
Mimic open and close gate
$c' = g(z)f(z_i) + cf(z_f)$
跟這個 h(c') 乘起來，如果 f(zo)是 1，就等於是
27:25 / 48:59
Created with EverCam

LSTM - Example

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



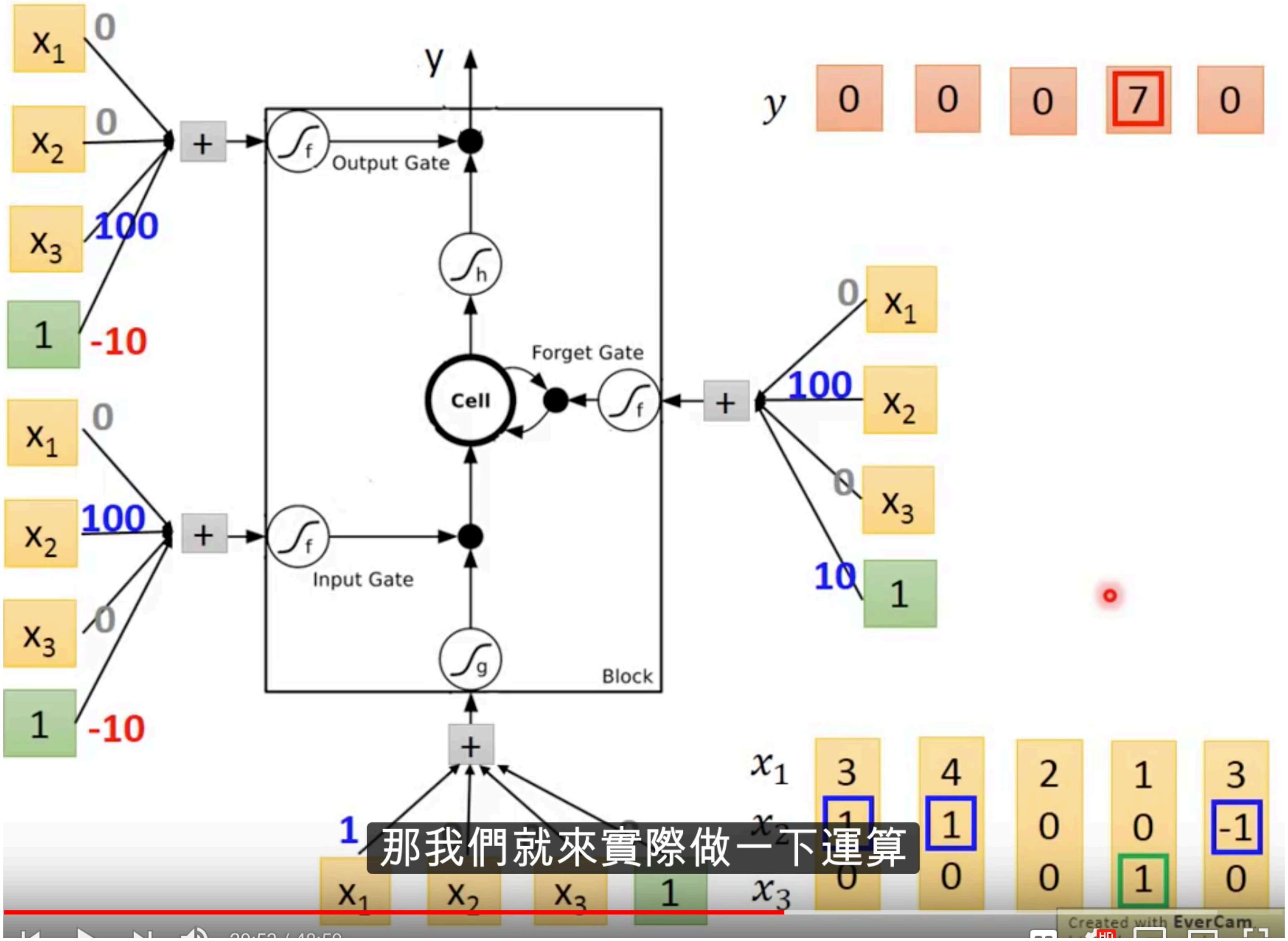
LSTM - Example

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	7	0	0	6

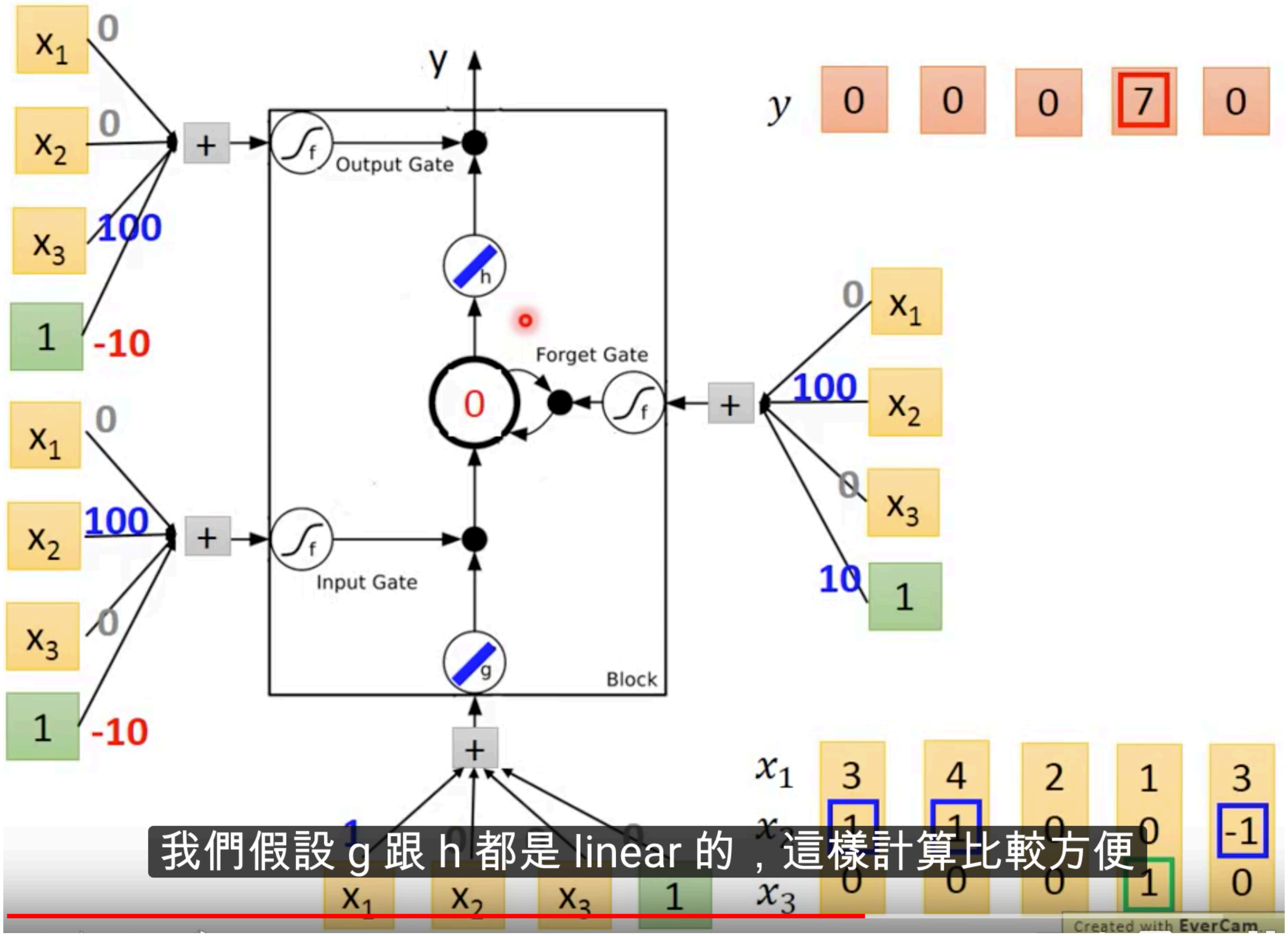
When $x_2 = 1$, add the numbers of x_1 into the memory

When $x_2 = -1$, reset the memory

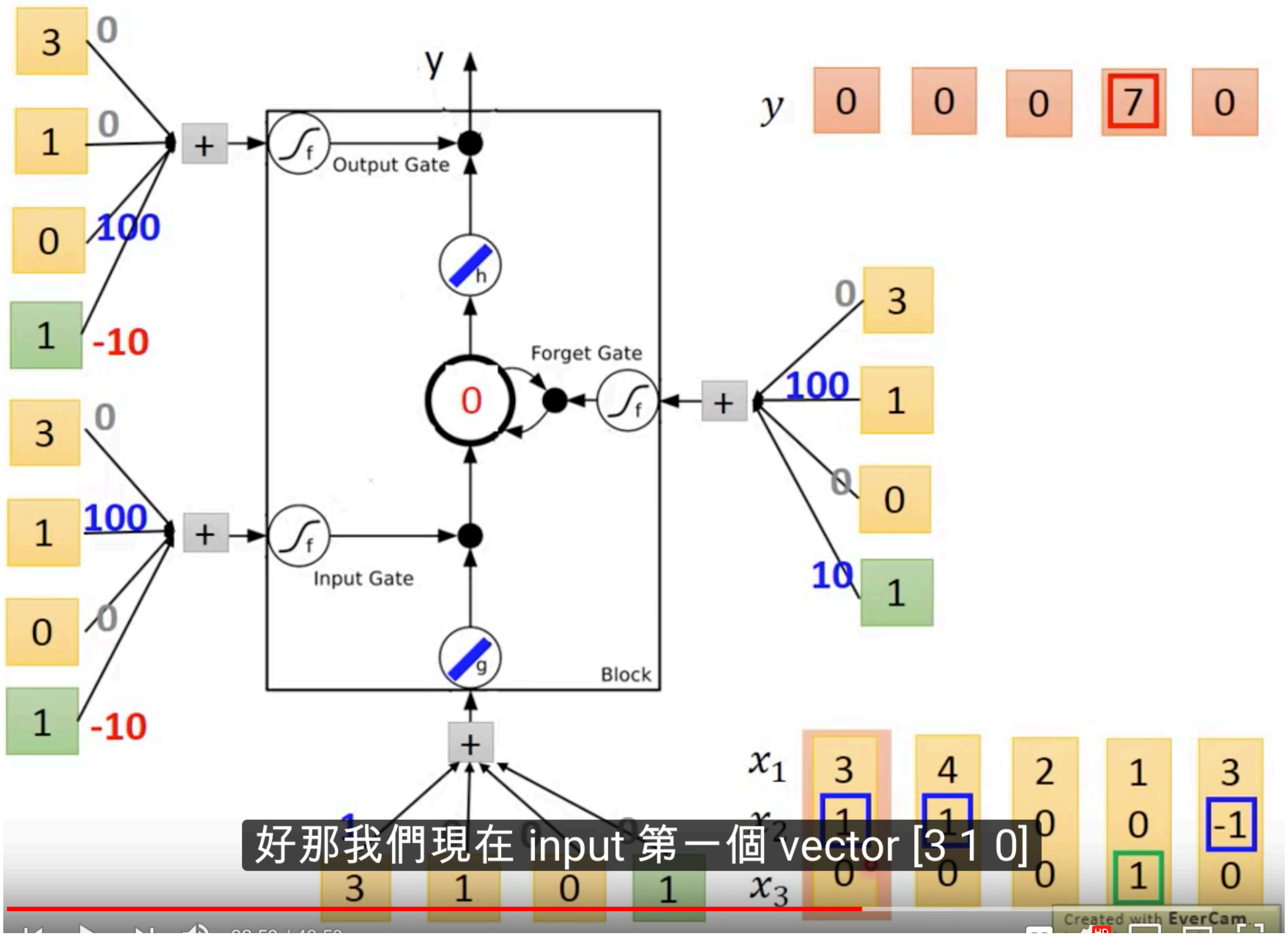
When $x_3 = 1$, output the number in the memory.



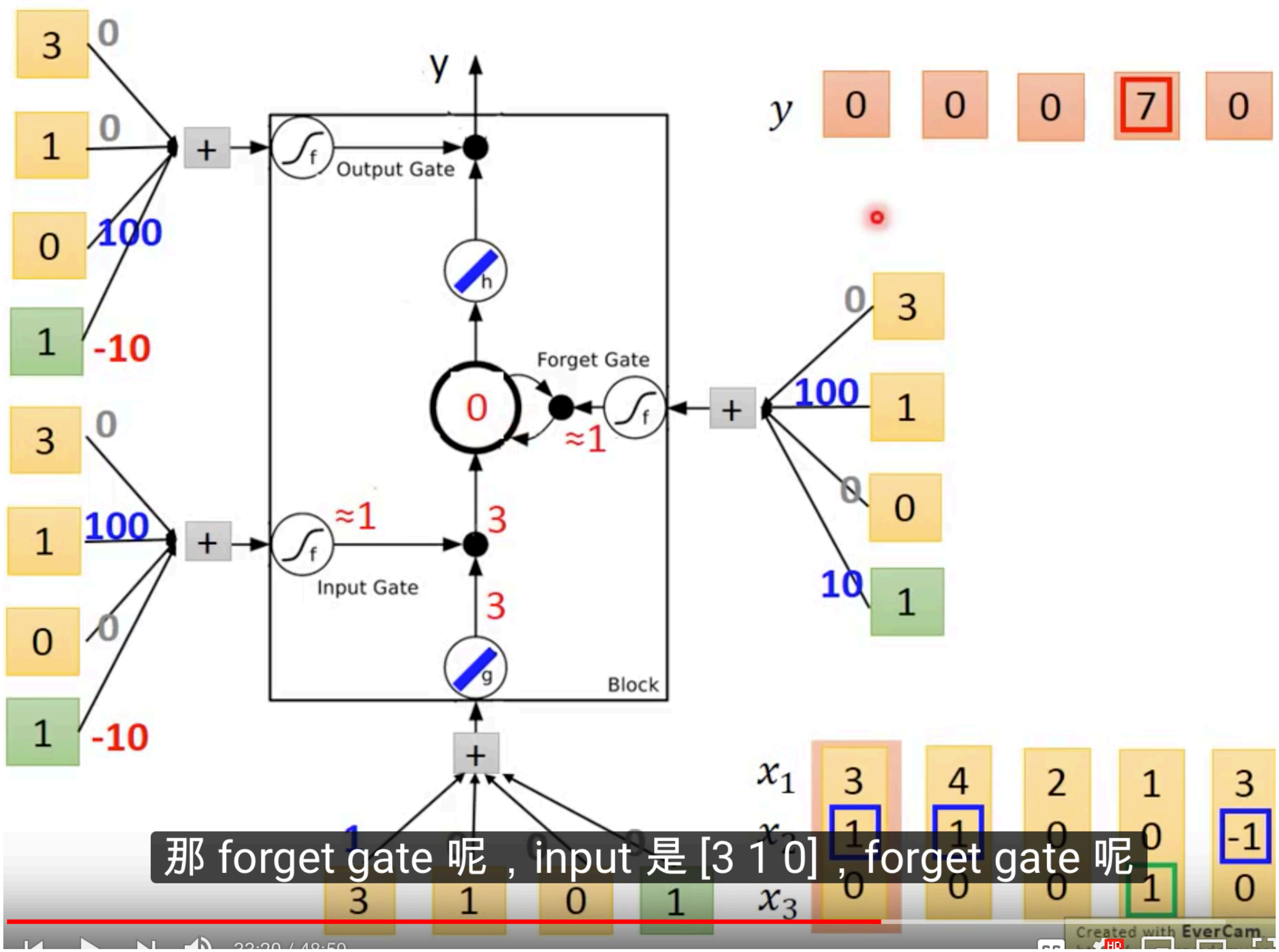
y
Output Gate
Forget Gate
Cell
Input Gate
Block
x1 x2 x3 1
0 0 100 -10
0 100 0 -10
0 100 0 10
1
y 0 0 0 7 0
x1 3 4 2 1 3
x2 1 1 0 0 -1
x3 0 0 0 1 0
那我們就來實際做一下運算
Created with EverCam



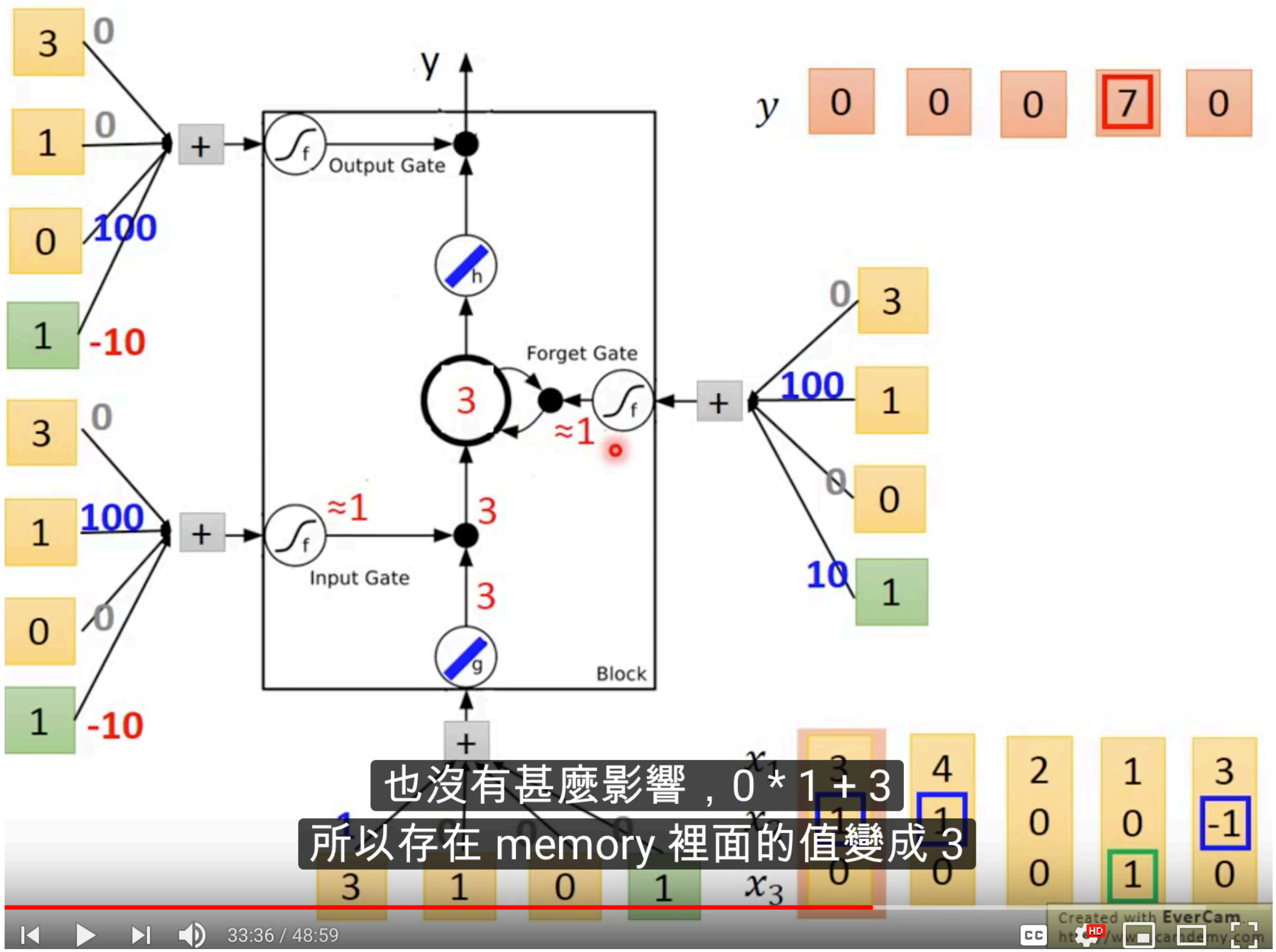
y
Output Gate
Forget Gate
Input Gate
Block
x1 0
x2 0
x3 100
1 -10
x1 0
x2 100
x3 0
1 -10
0 x1
100 x2
0 x3
10 1
x1 x2 x3 1
y 0 0 0 7 0
x1 3 4 2 1 3
x3 0 0 0 1 0
我們假設 g 跟 h 都是 linear 的，這樣計算比較方便
Created with EverCam

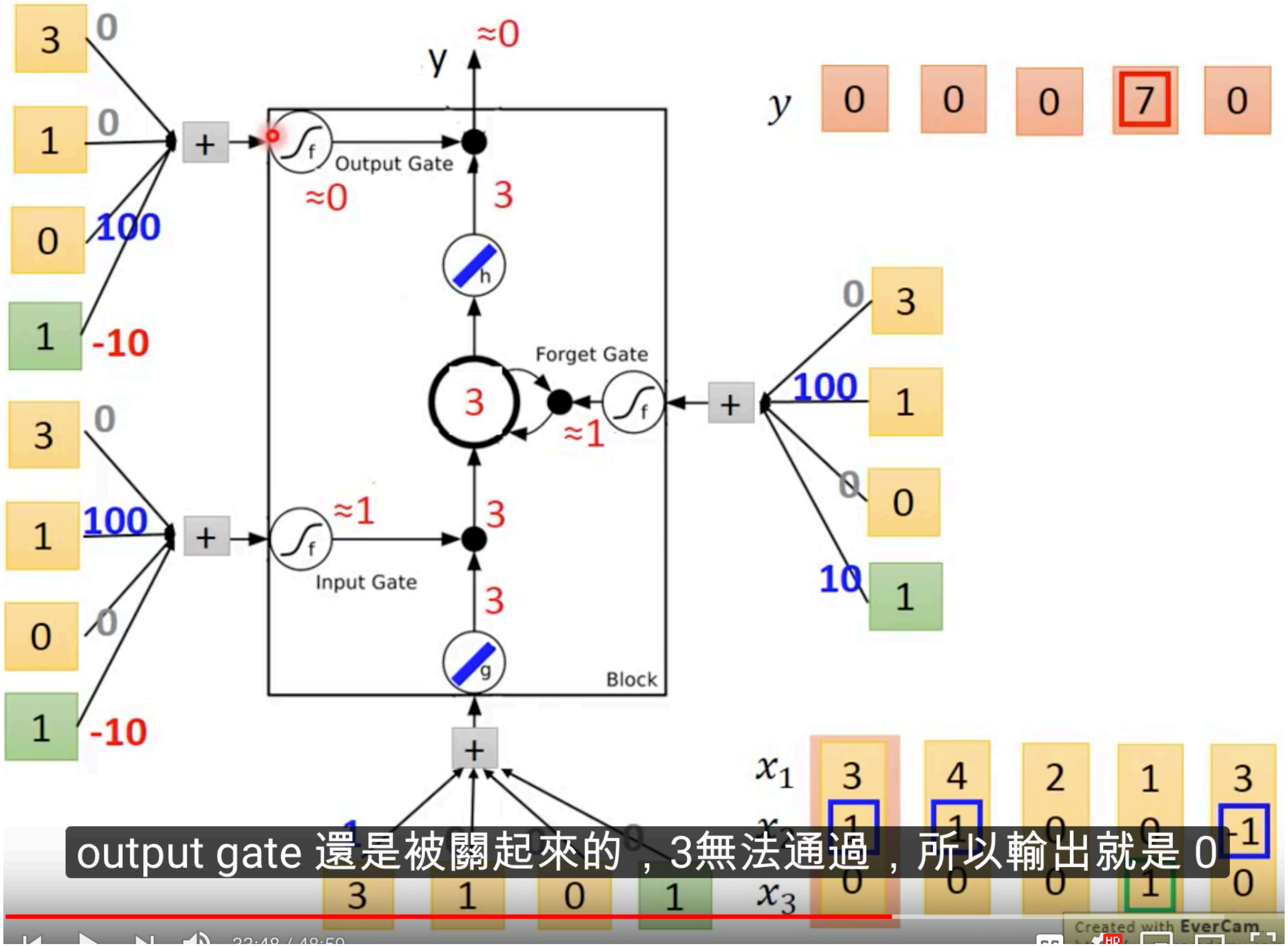


y
Output Gate
Forget Gate
Input Gate
Block
0
100
-10
10
x1
x2
x3
3 4 2 1 3
1 1 0 0 -1
0 0 0 1 0
y
0 0 0 7 0
好那我們現在 input 第一個 vector [3 1 0]
Created with EverCam

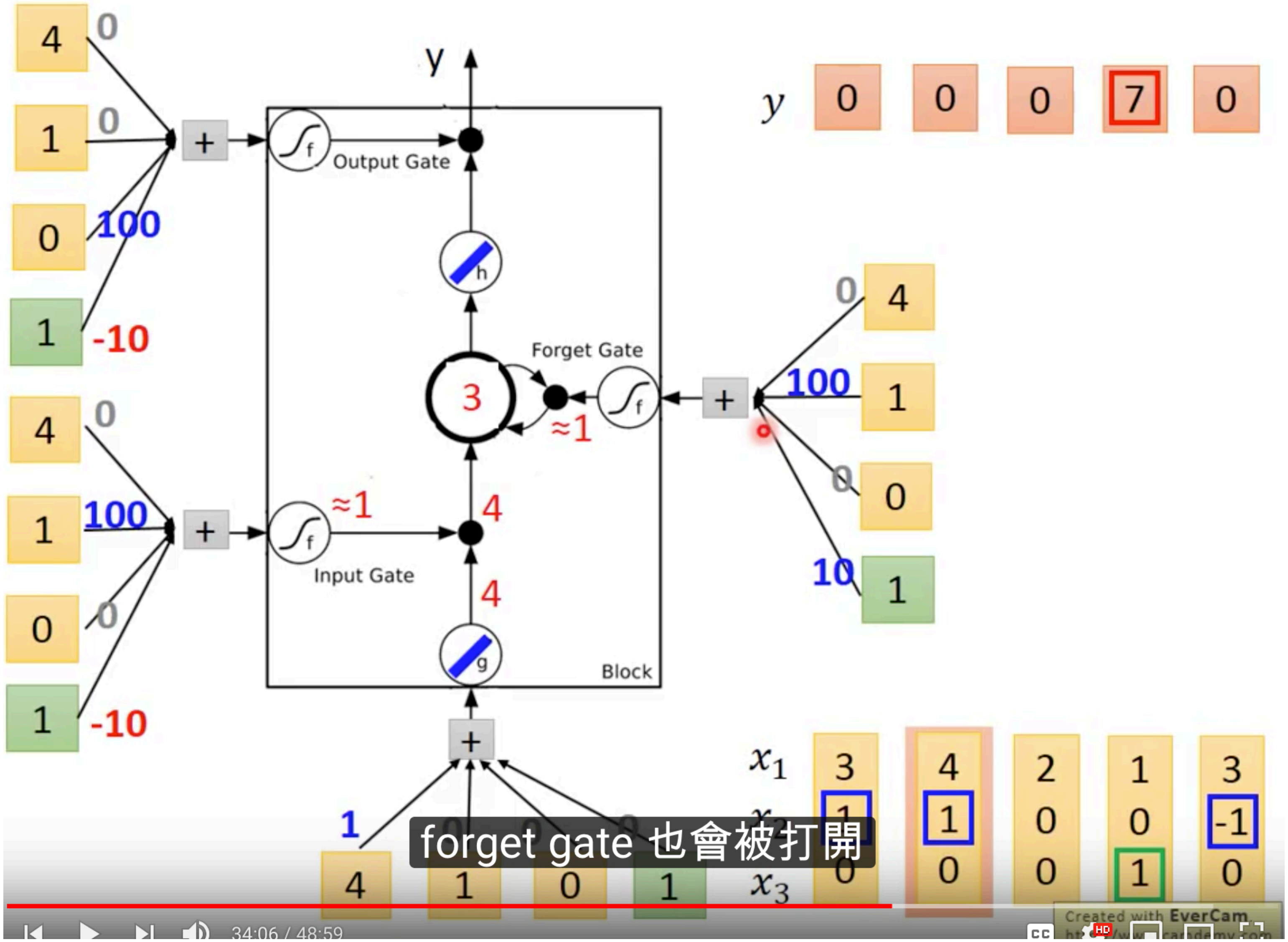


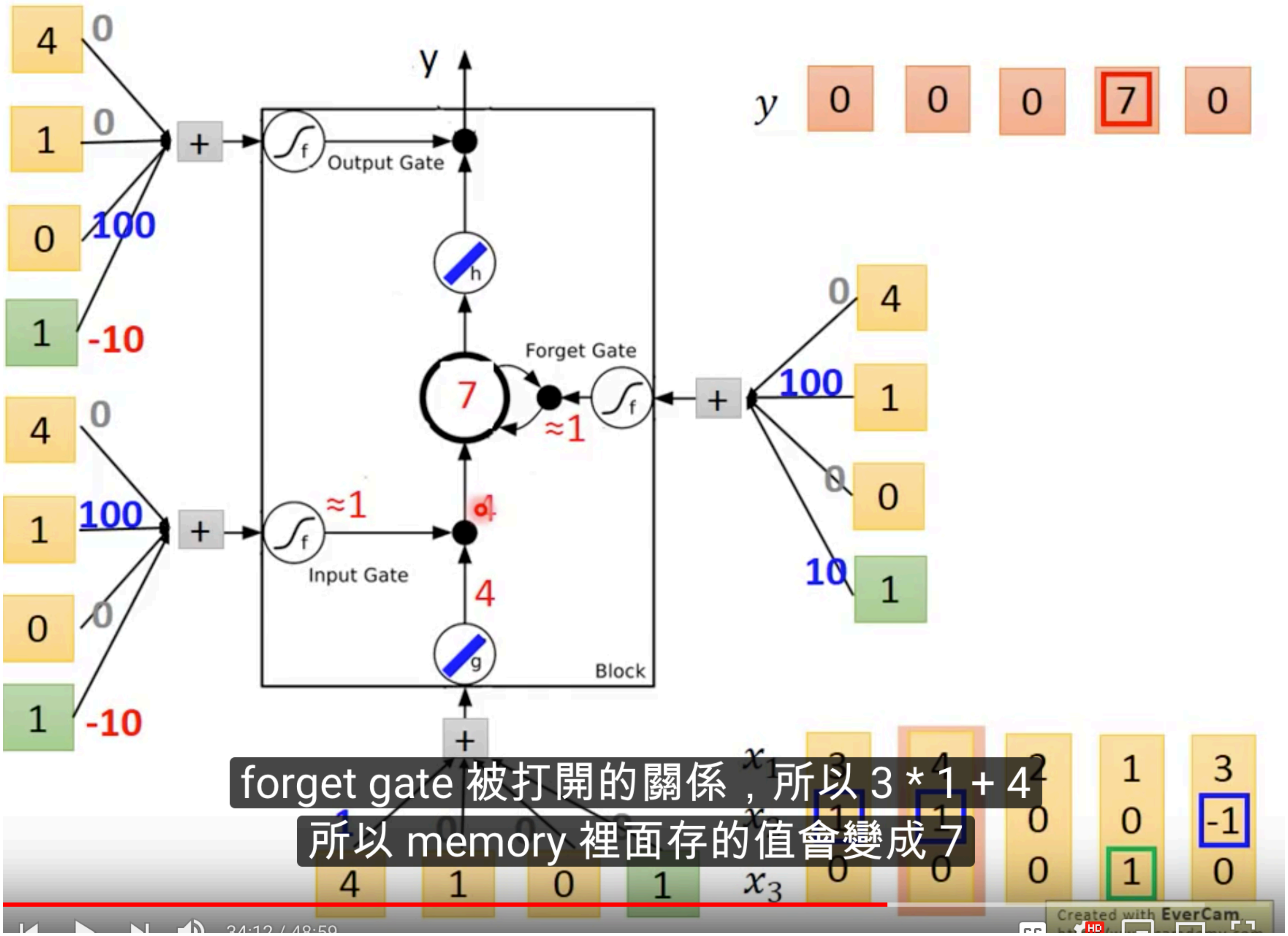
y
Output Gate
Forget Gate
Input Gate
Block
≈1
≈1
100
-10
10
y
0
0
0
7
0
x1
x3
3
4
2
1
3
-1
那 forget gate 呢，input 是 [3 1 0]，forget gate 呢
Created with EverCam

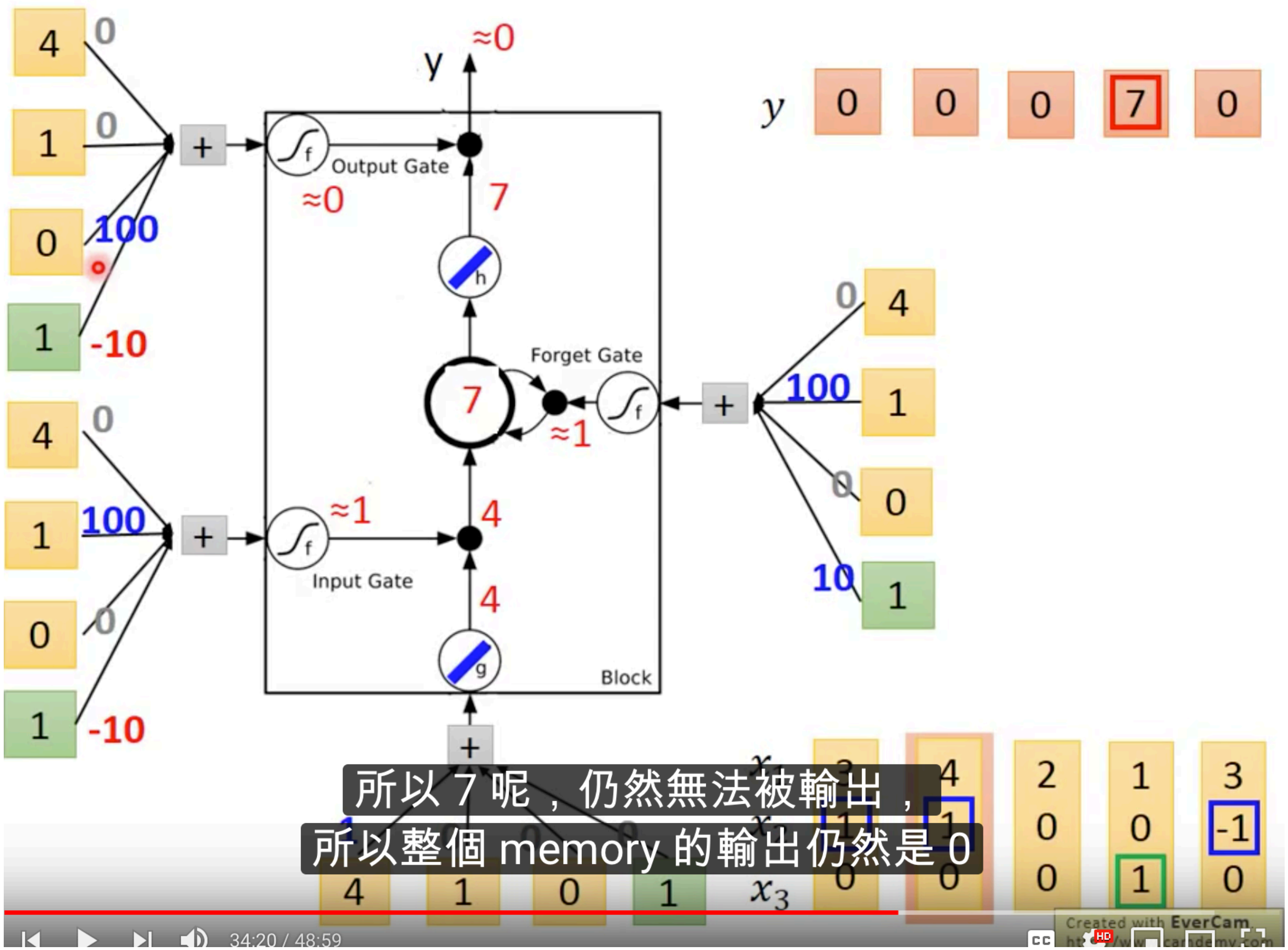

y
Output Gate
Forget Gate
Input Gate
Block
也沒有甚麼影響，0 * 1 + 3
所以存在 memory 裡面的值變成 3
33:36 / 48:59
Created with EverCam



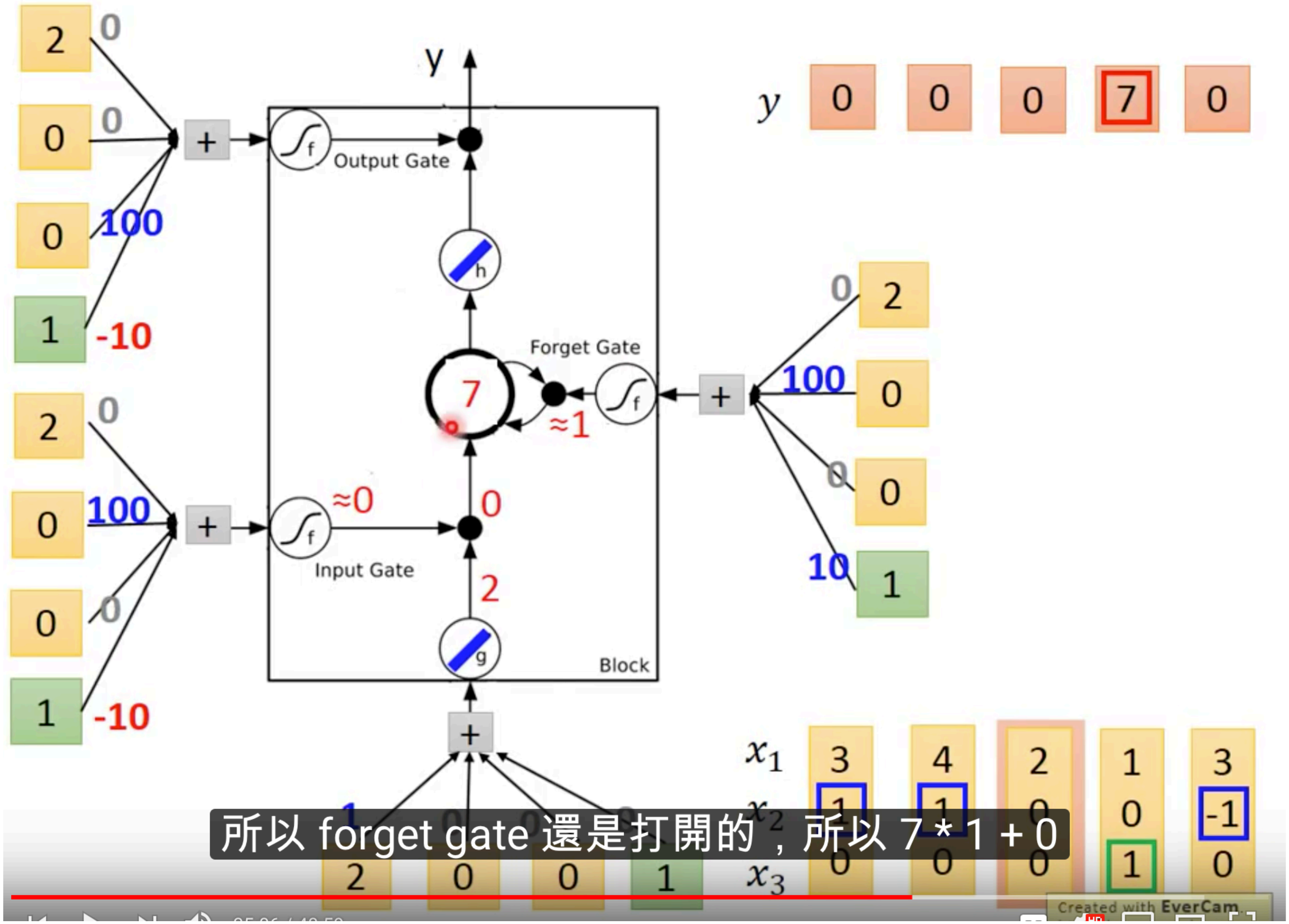
y
≈0
Output Gate
≈0
3
Forget Gate
3
≈1
≈1
Input Gate
3
3
Block
100
-10
100
-10
100
10
y
0
0
0
7
0
x_1
3
4
2
1
3
x_3
output gate 還是被關起來的，3無法通過，所以輸出就是 0
Created with EverCam

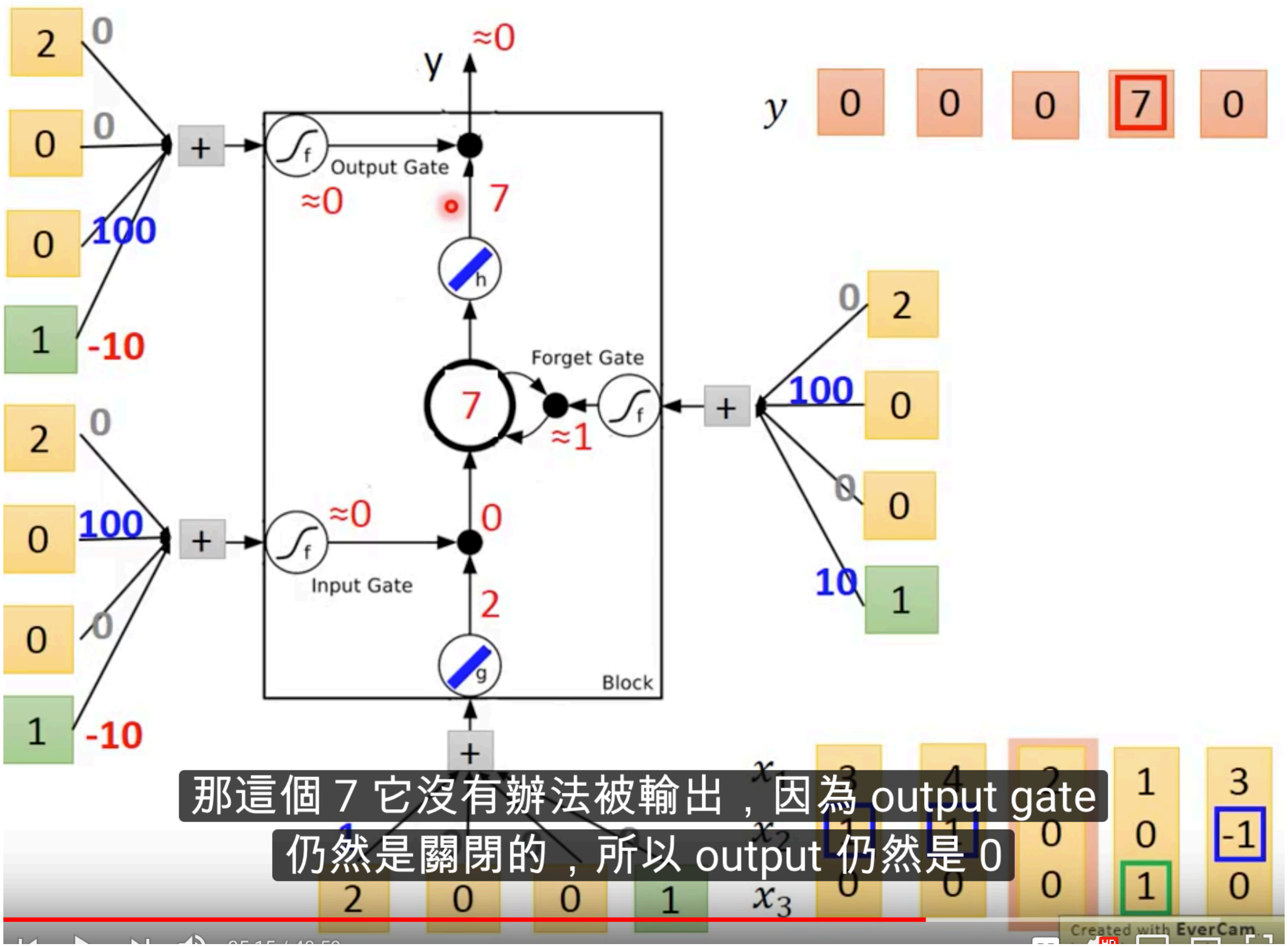

y
Output Gate
Forget Gate
Input Gate
Block
≈1
≈1
3
4
4
100
-10
10
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2
x_3
1 0 1 -1
0 0 0 1 0
forget gate 也會被打開
34:06 / 48:59
Created with EverCam

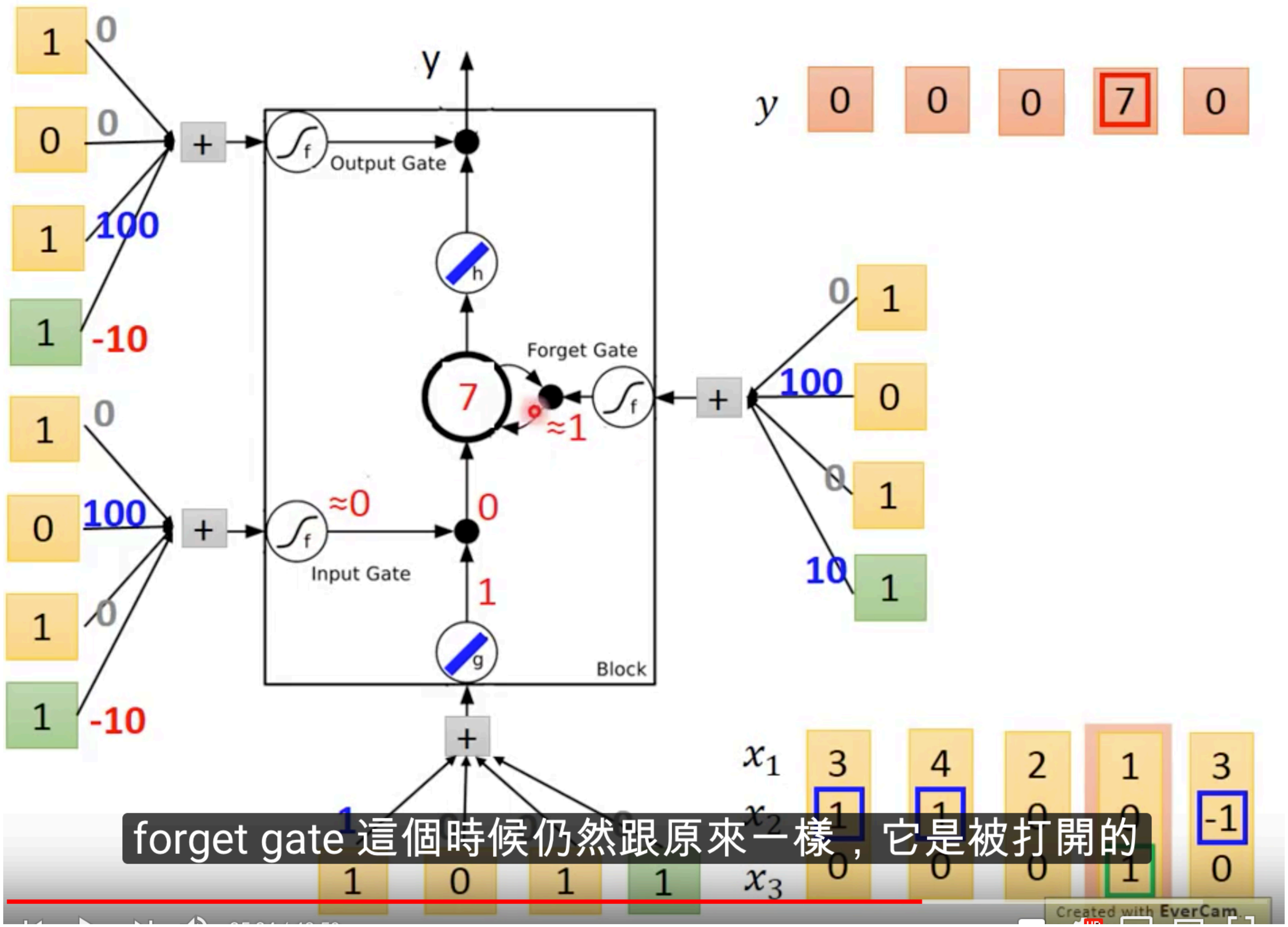

y
Output Gate
Forget Gate
Input Gate
Block
≈1
≈1
7
4
100
-10
100
-10
100
10
y 0 0 0 7 0
x_3
forget gate 被打開的關係，所以 3 * 1 + 4
所以 memory 裡面存的值會變成 7
Created with EverCam



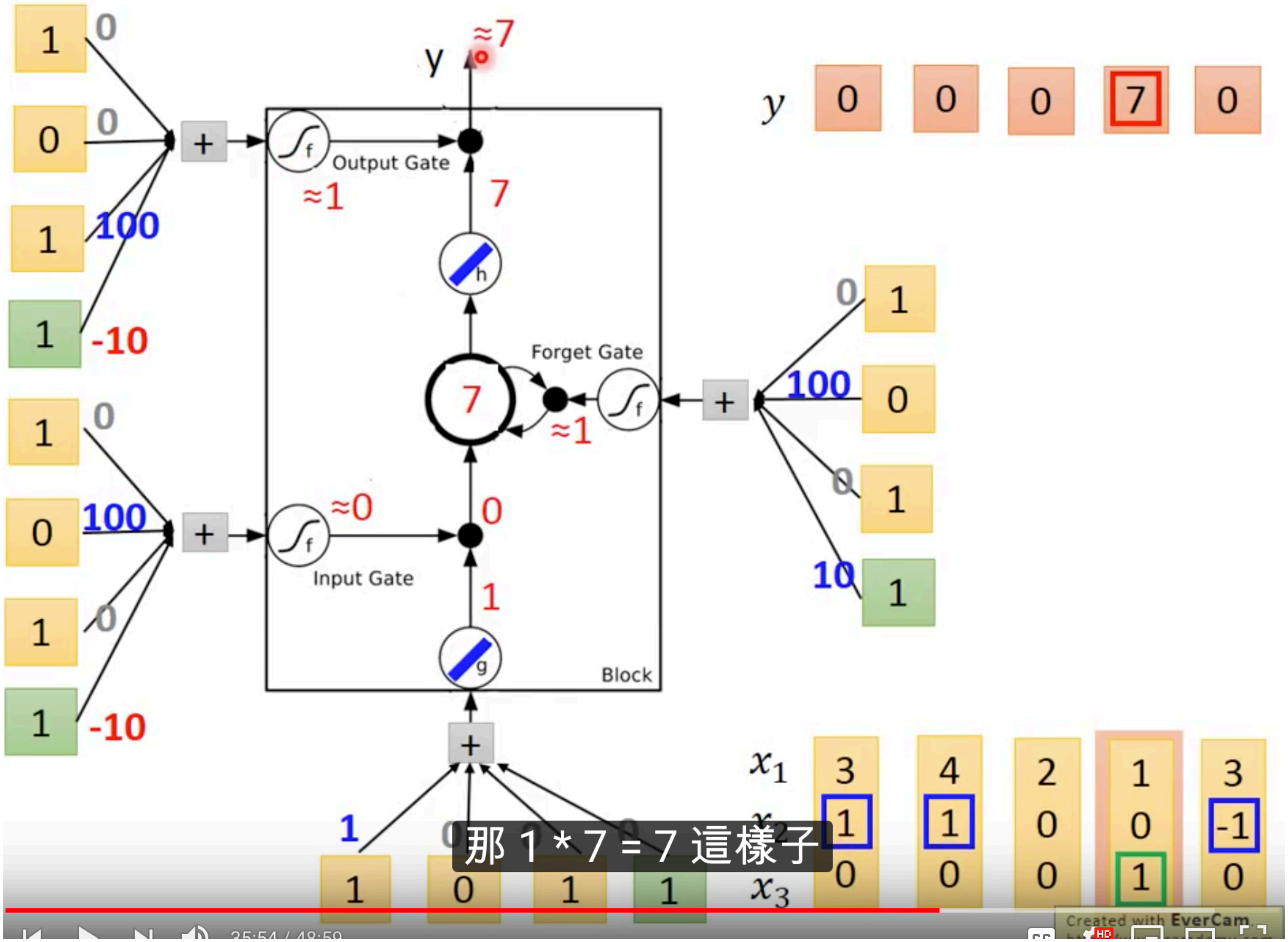
所以 7 呢，仍然無法被輸出，
所以整個 memory 的輸出仍然是 0
Output Gate
Input Gate
Forget Gate
Block
y
≈0
≈1
100
-10
10
34:20 / 48:59
Created with EverCam


y
Output Gate
Forget Gate
Input Gate
Block
7
≈1
≈0
0
2
100
-10
10
+
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2
x_3
所以 forget gate 還是打開的，所以 7 * 1 + 0
Created with EverCam

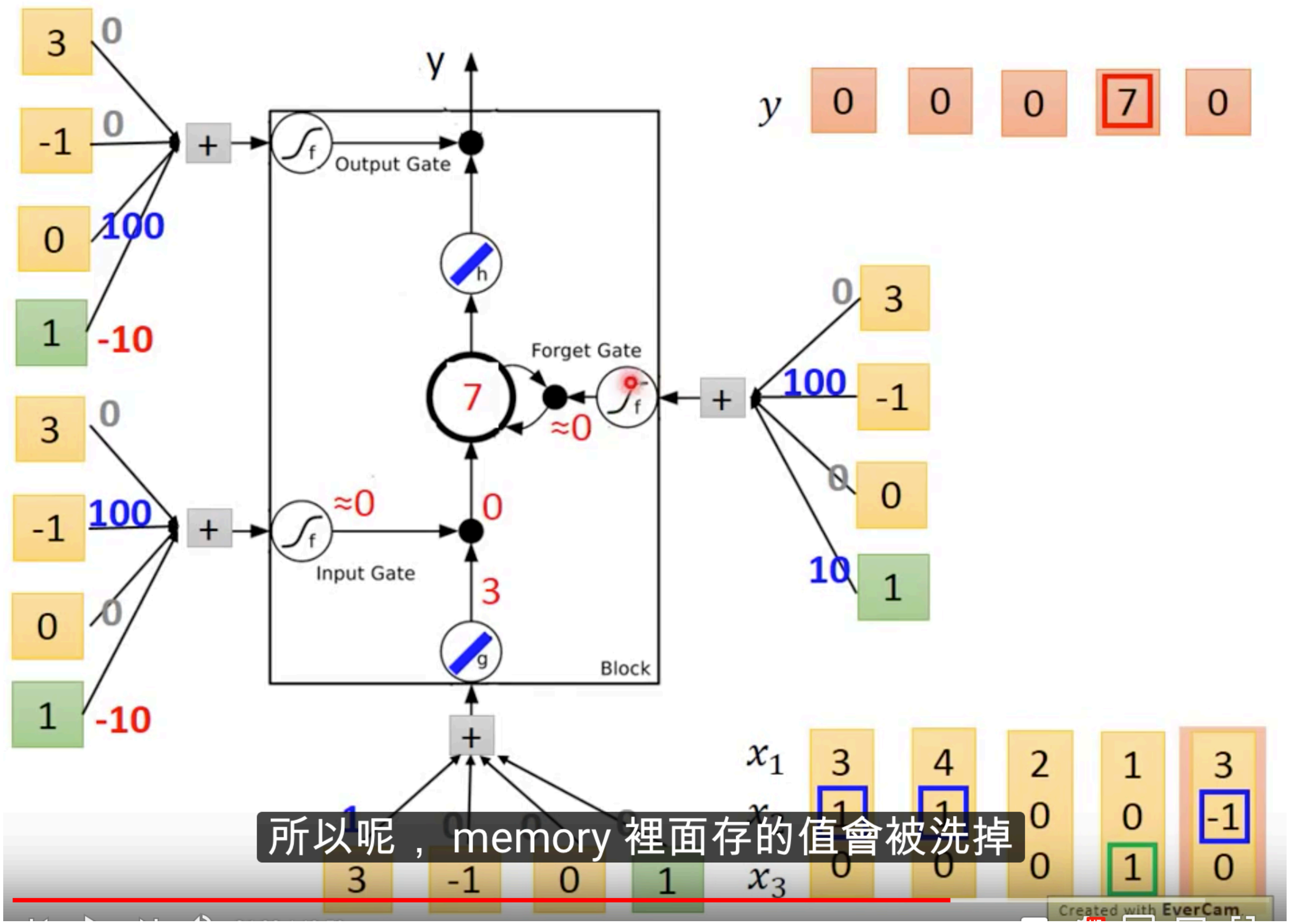

y
≈0
Output Gate
7
Forget Gate
≈1
Input Gate
Block
100
-10
10
那這個 7 它沒有辦法被輸出，因為 output gate
仍然是關閉的，所以 output 仍然是 0
Created with EverCam

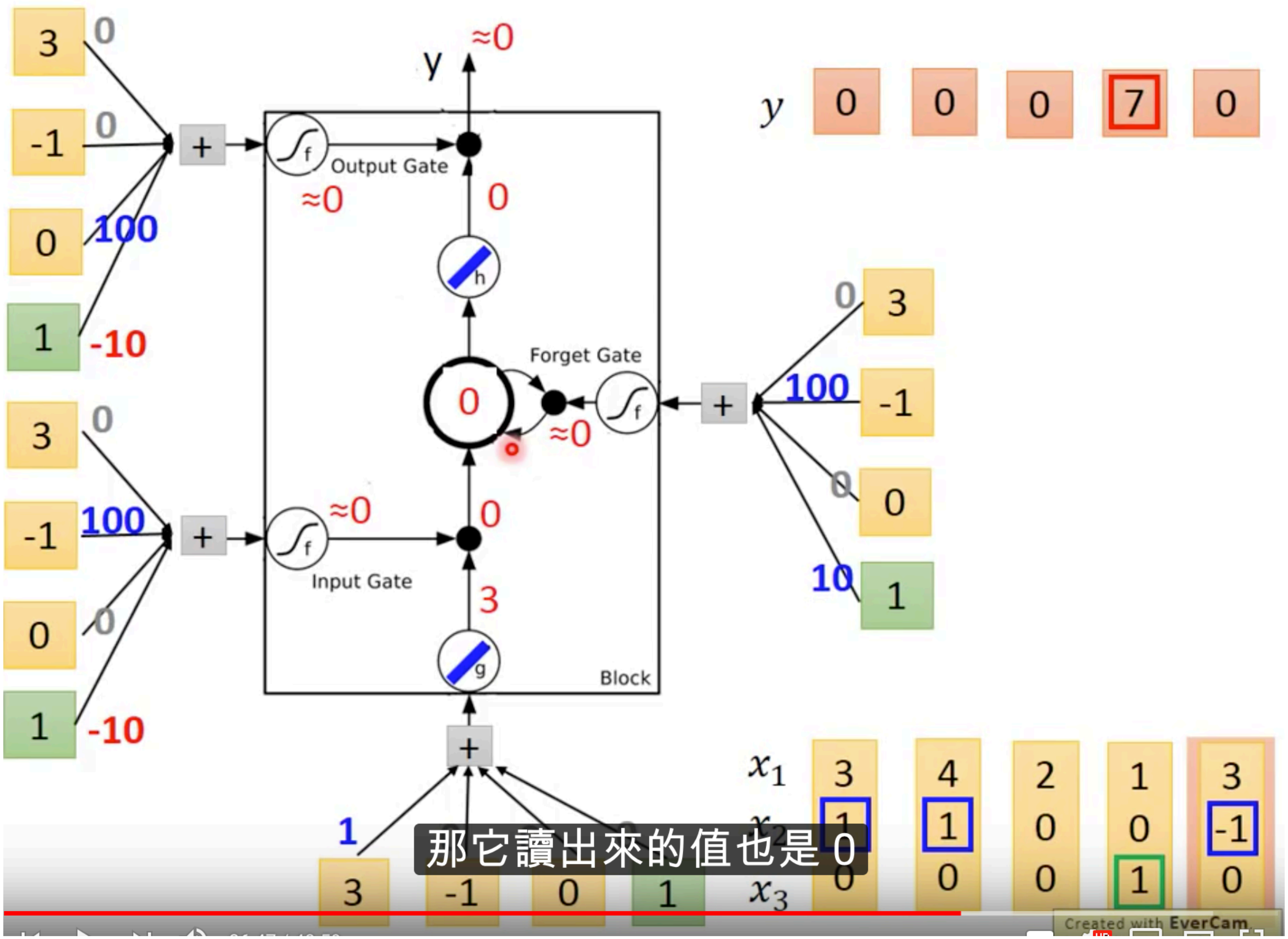


y
Output Gate
Forget Gate
Input Gate
Block
≈1
≈0
7
forget gate 這個時候仍然跟原來一樣，它是被打開的
Created with EverCam



y
≈7
Output Gate
≈1
7
Forget Gate
≈1
Input Gate
≈0
0
1
Block
100
-10
10
x_1
x_3
那 1 * 7 = 7 這樣子
Created with EverCam

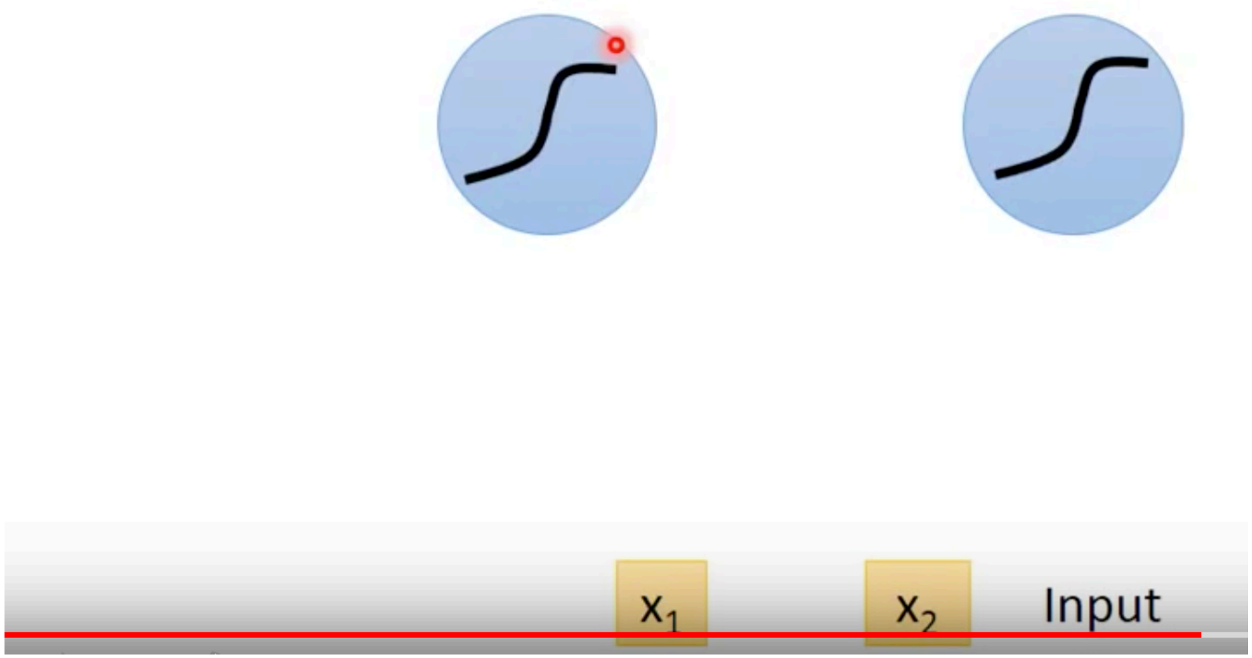

y
Output Gate
Forget Gate
Input Gate
Block
7
≈0
0
3
100
-10
10
y 0 0 0 7 0
x1 3 4 2 1 3
x3
所以呢，memory 裡面存的值會被洗掉
Created with EverCam



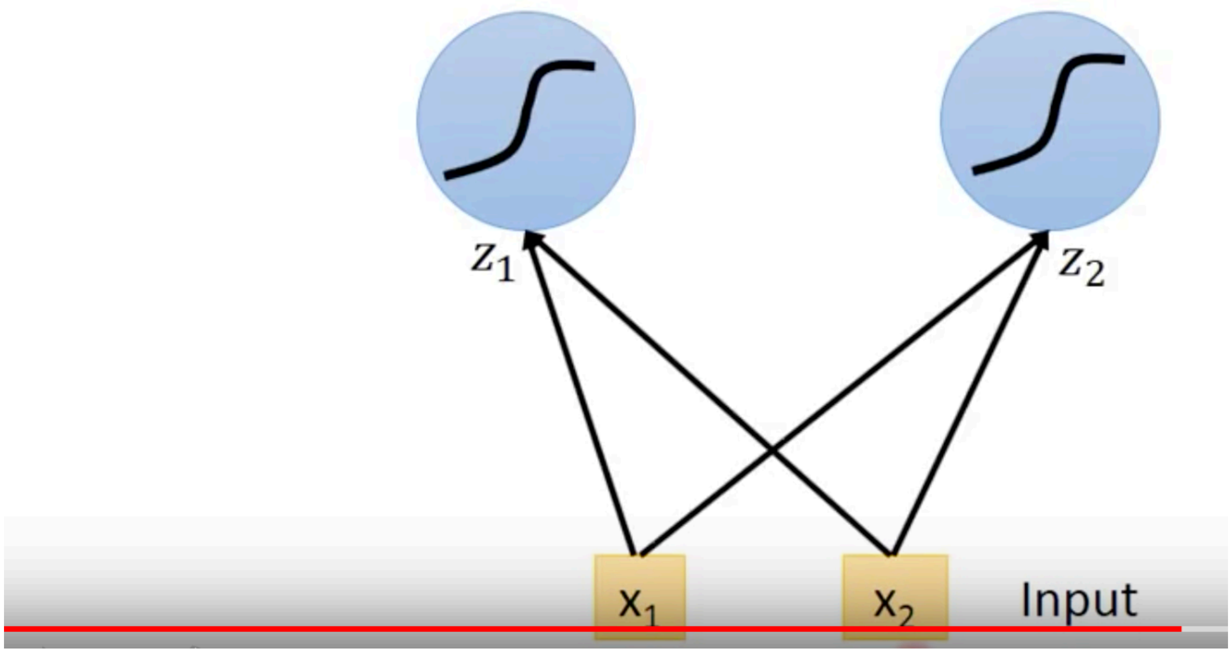
y
≈0
Output Gate
Forget Gate
Input Gate
Block
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0
那它讀出來的值也是 0
Created with EverCam

Original Network:

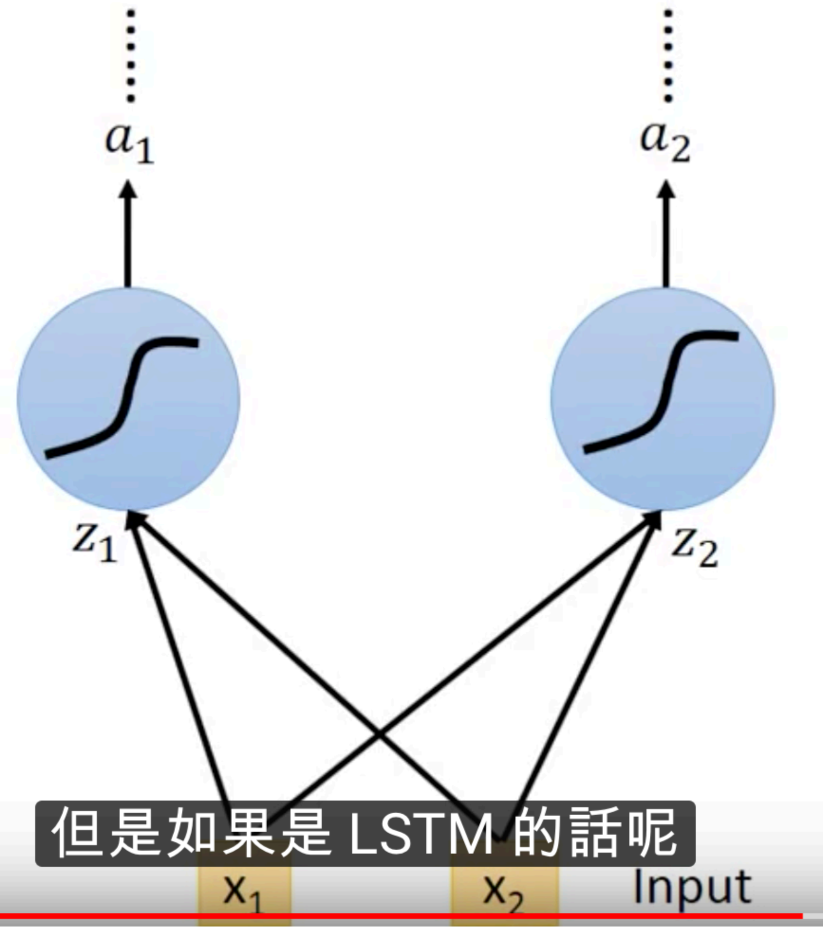
How to understand LSTM as the original network:



Original Network:



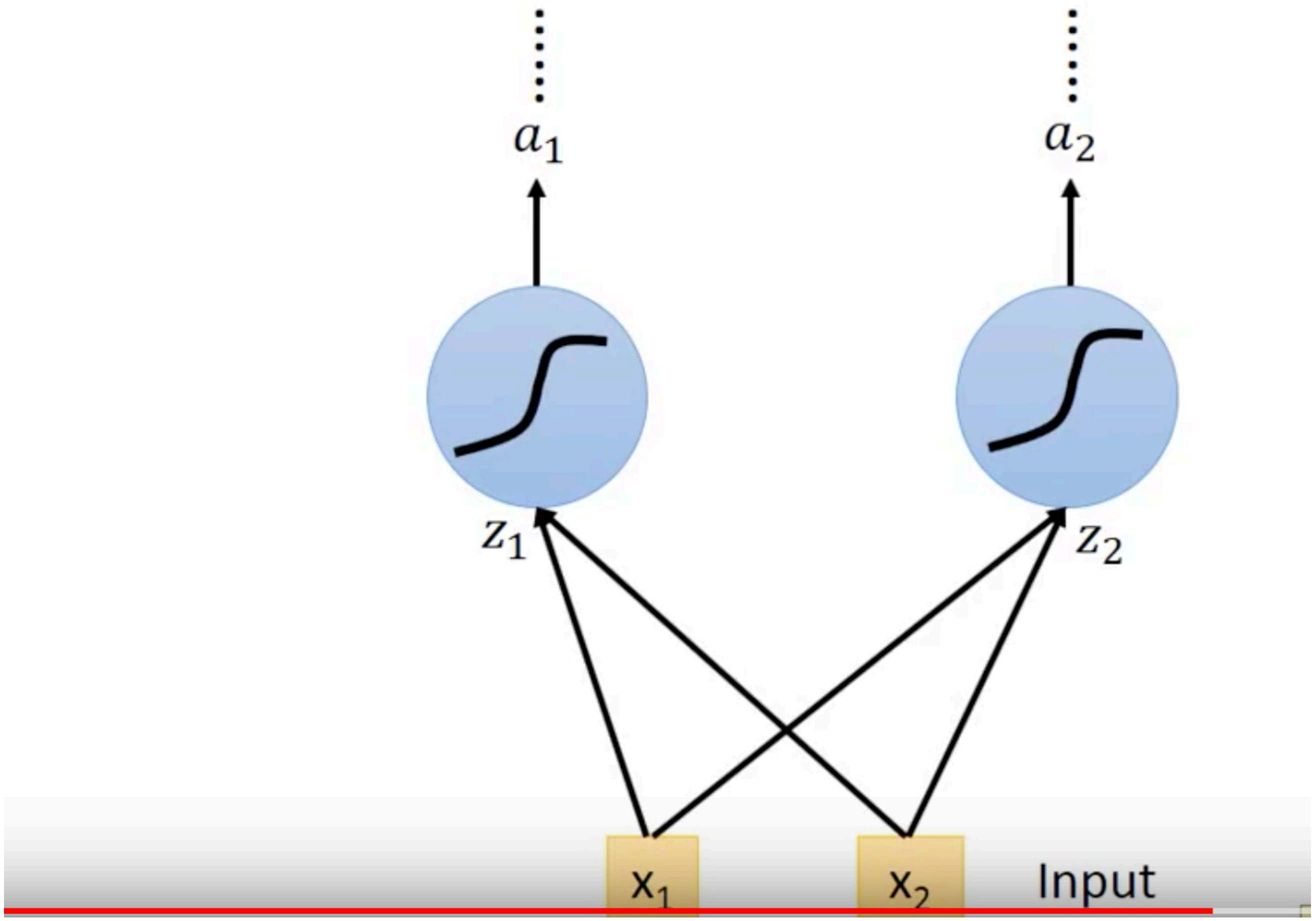
Original Network:

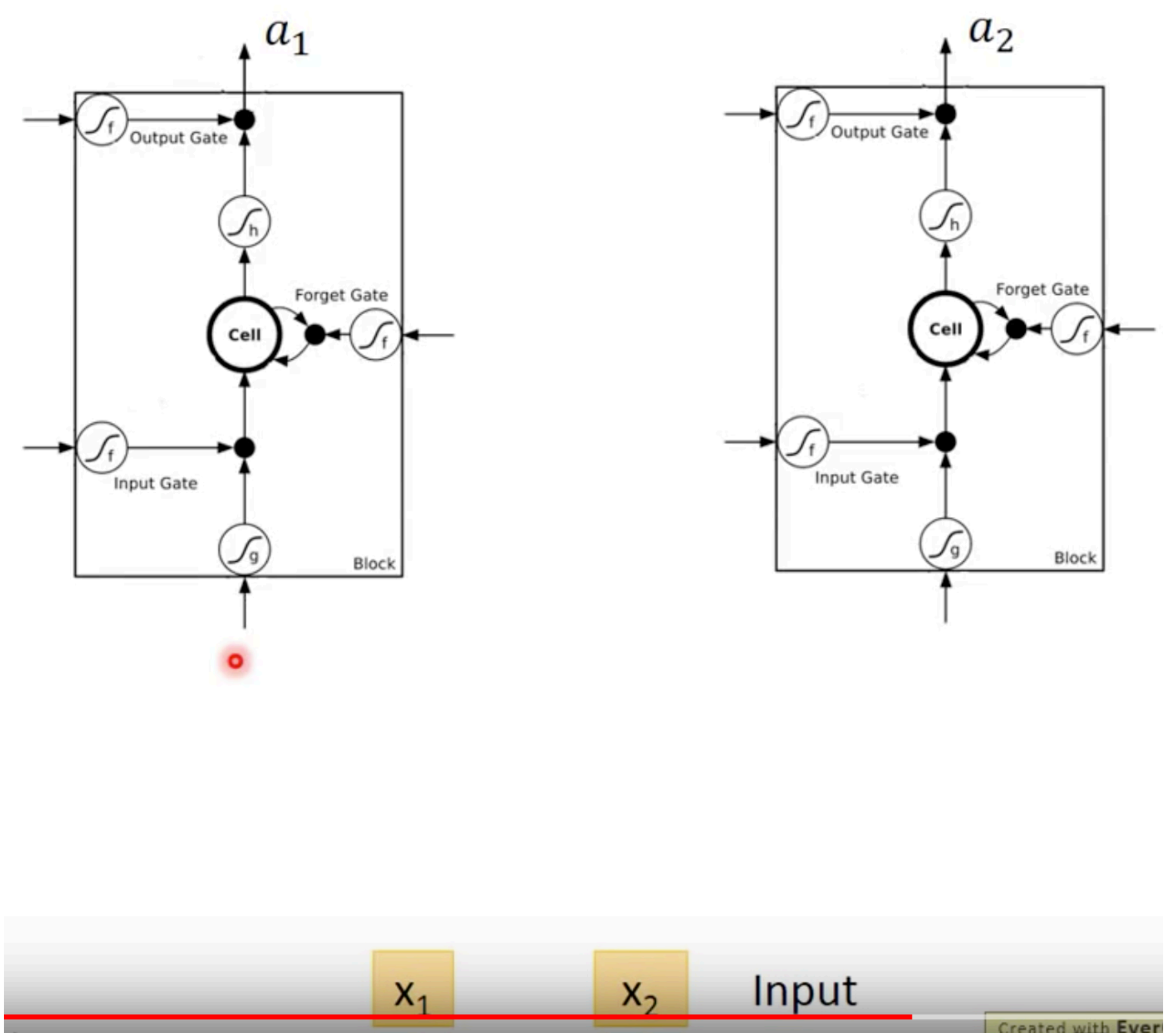


但是如果是 LSTM 的話呢

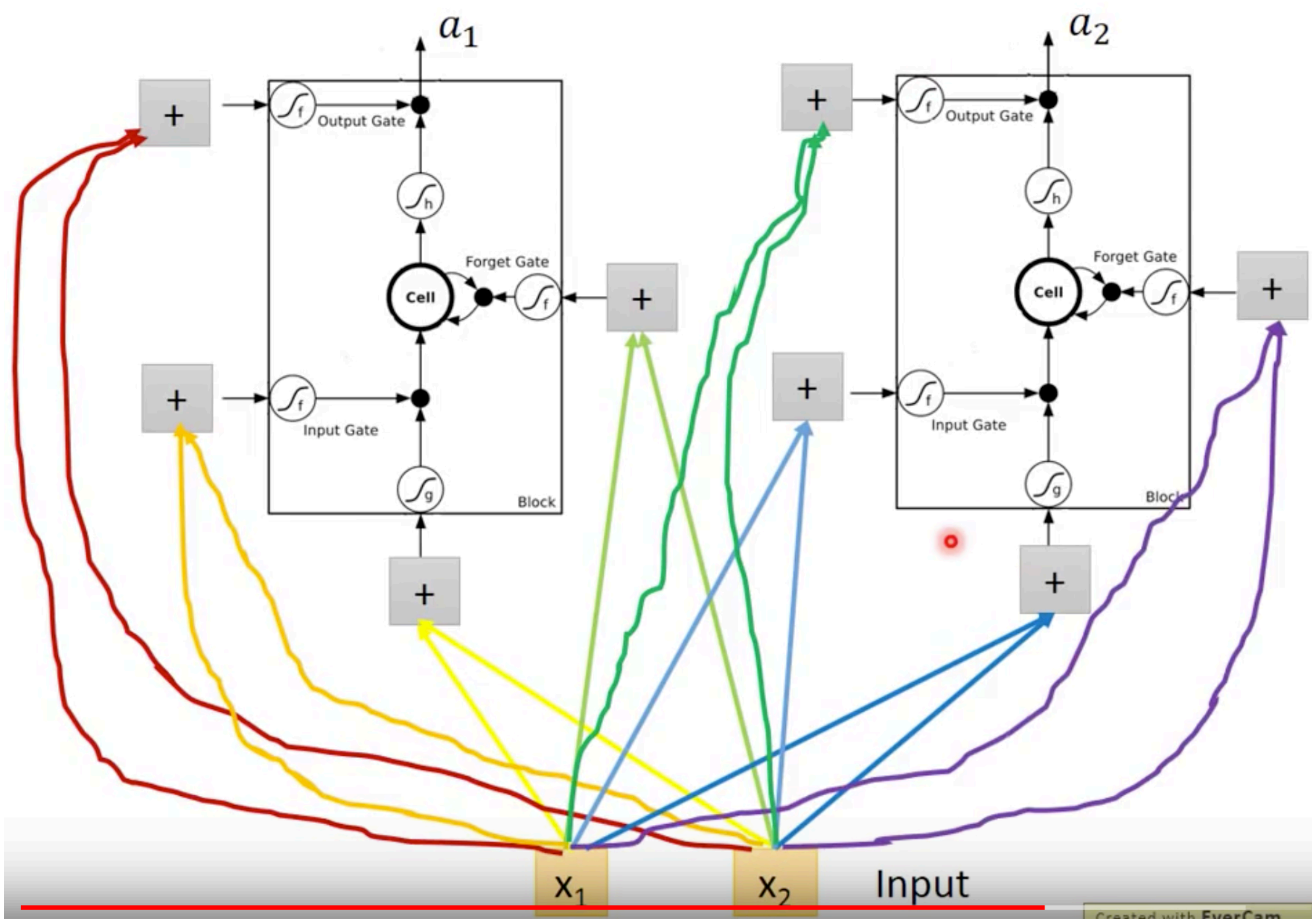
Original Network:

➢Simply replace the neurons with LSTM

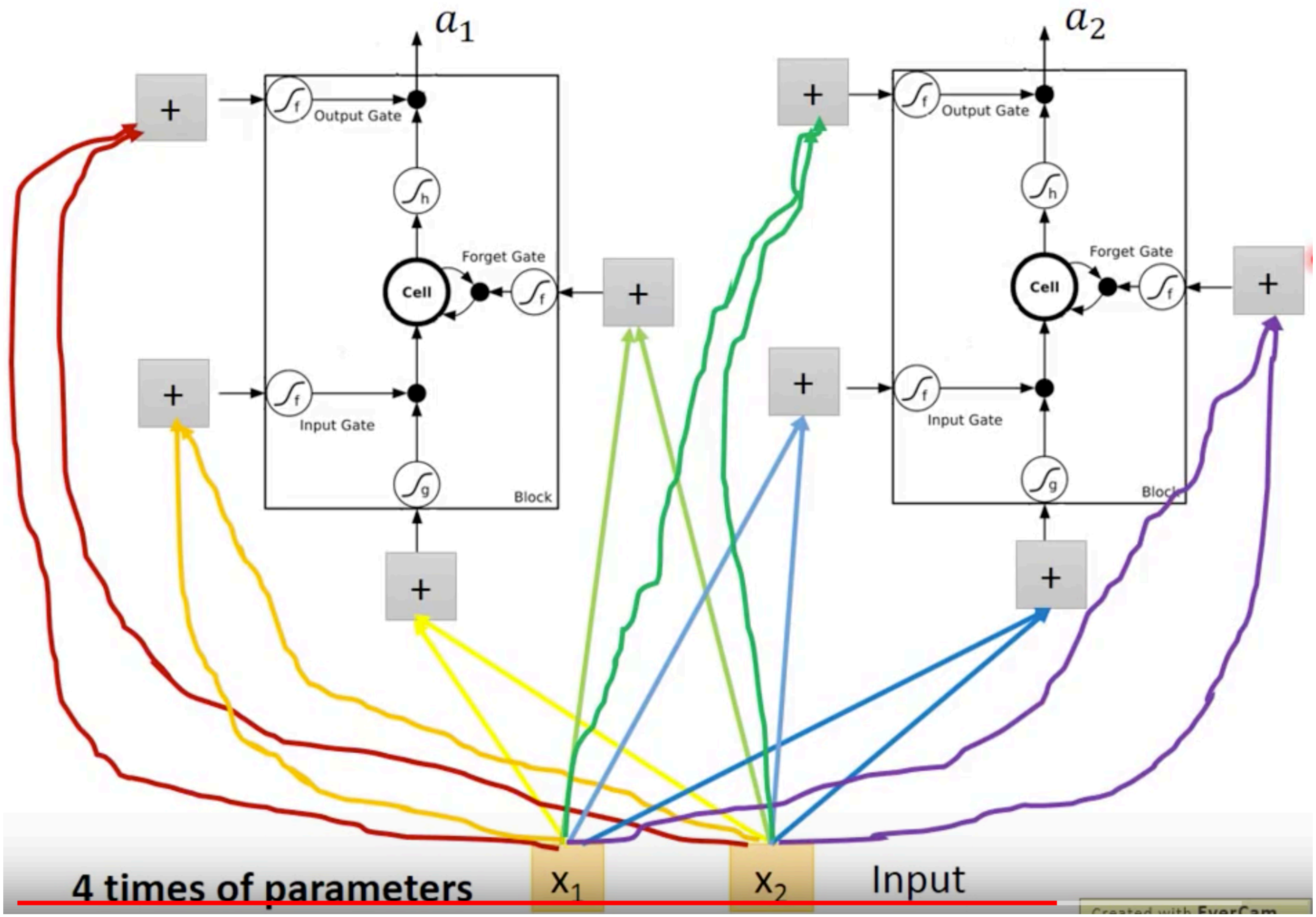




a1
Output Gate
Forget Gate
Cell
Input Gate
Block
a2
Output Gate
Forget Gate
Cell
Input Gate
Block
x1
x2
Input



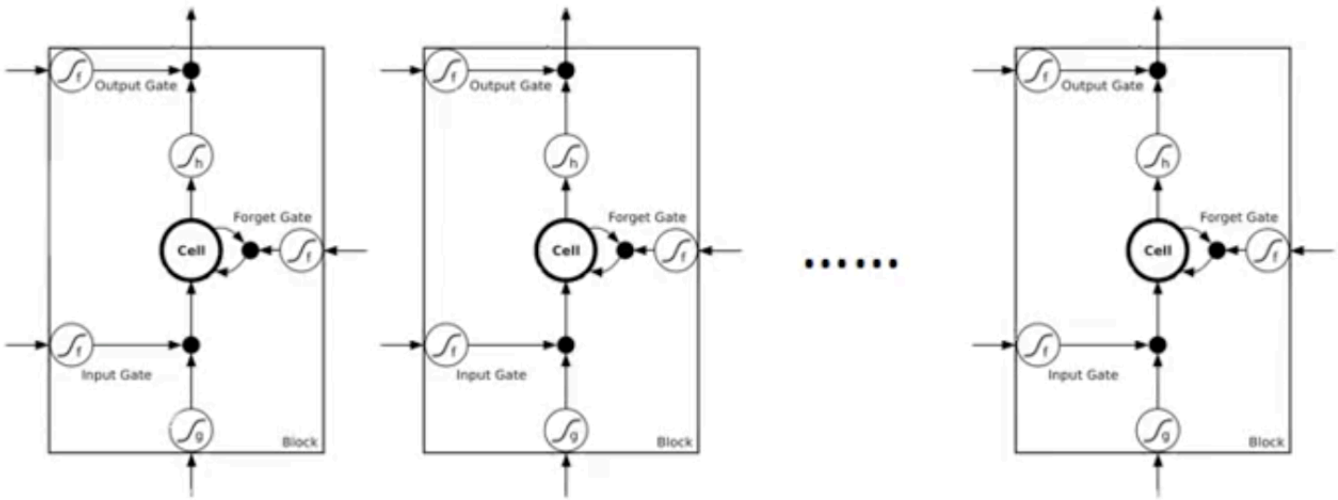
Each edge has a different weight



a1
a2
Output Gate
Forget Gate
Cell
Input Gate
Block
+
x1
x2
Input
4 times of parameters
Created with EverCam

How to understand LSTM as RNN

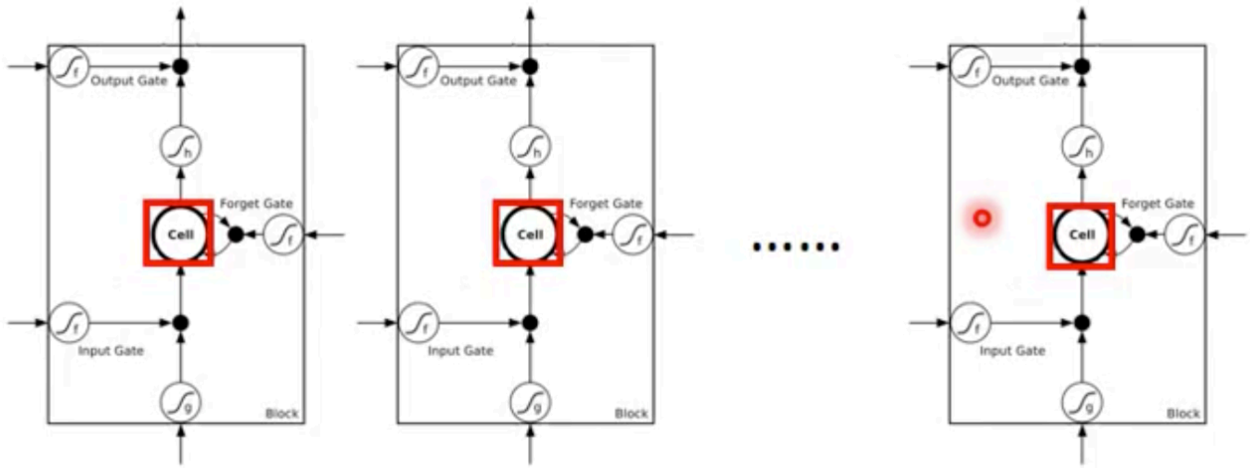
LSTM



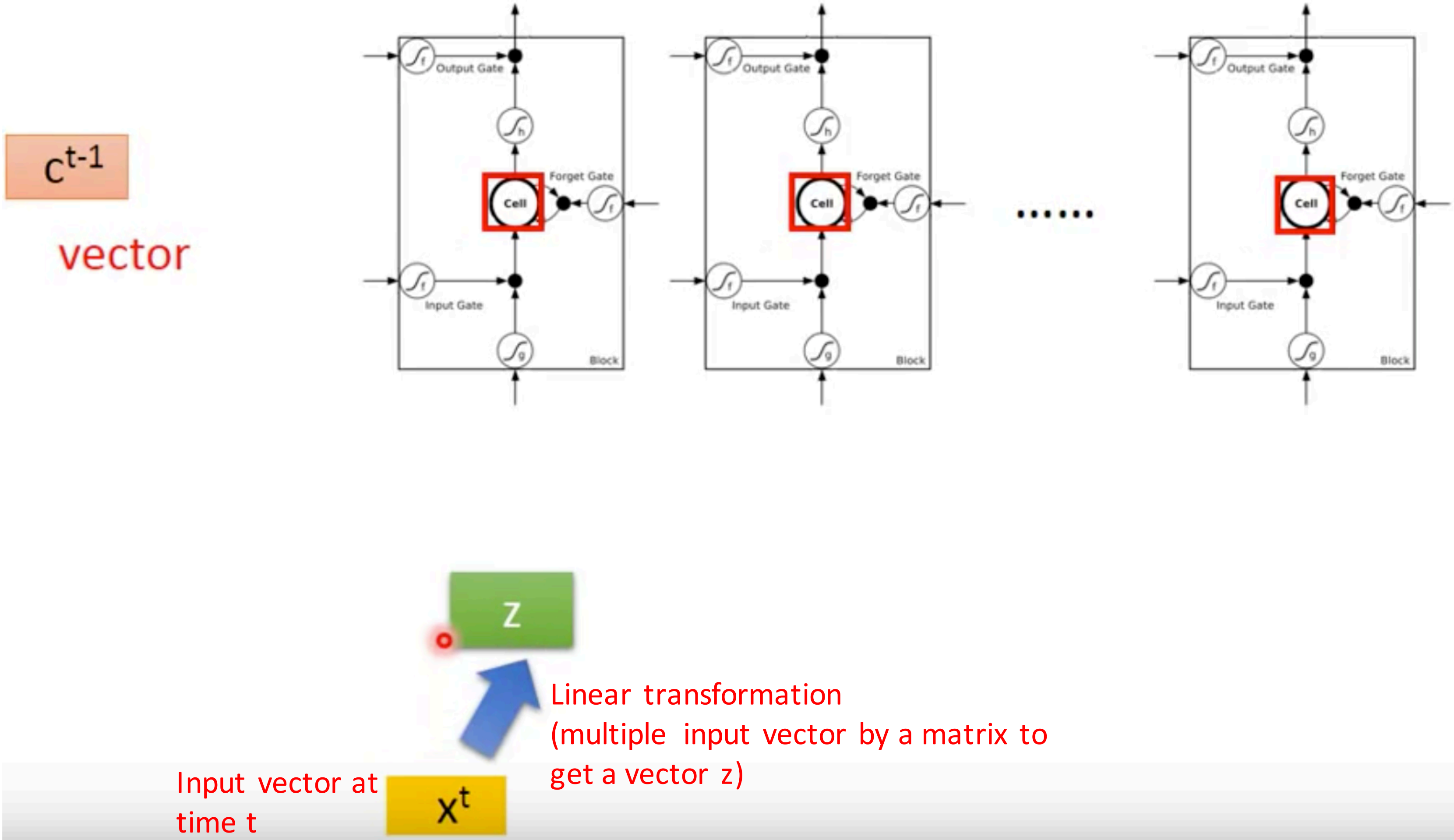
LSTM

c^{t-1}

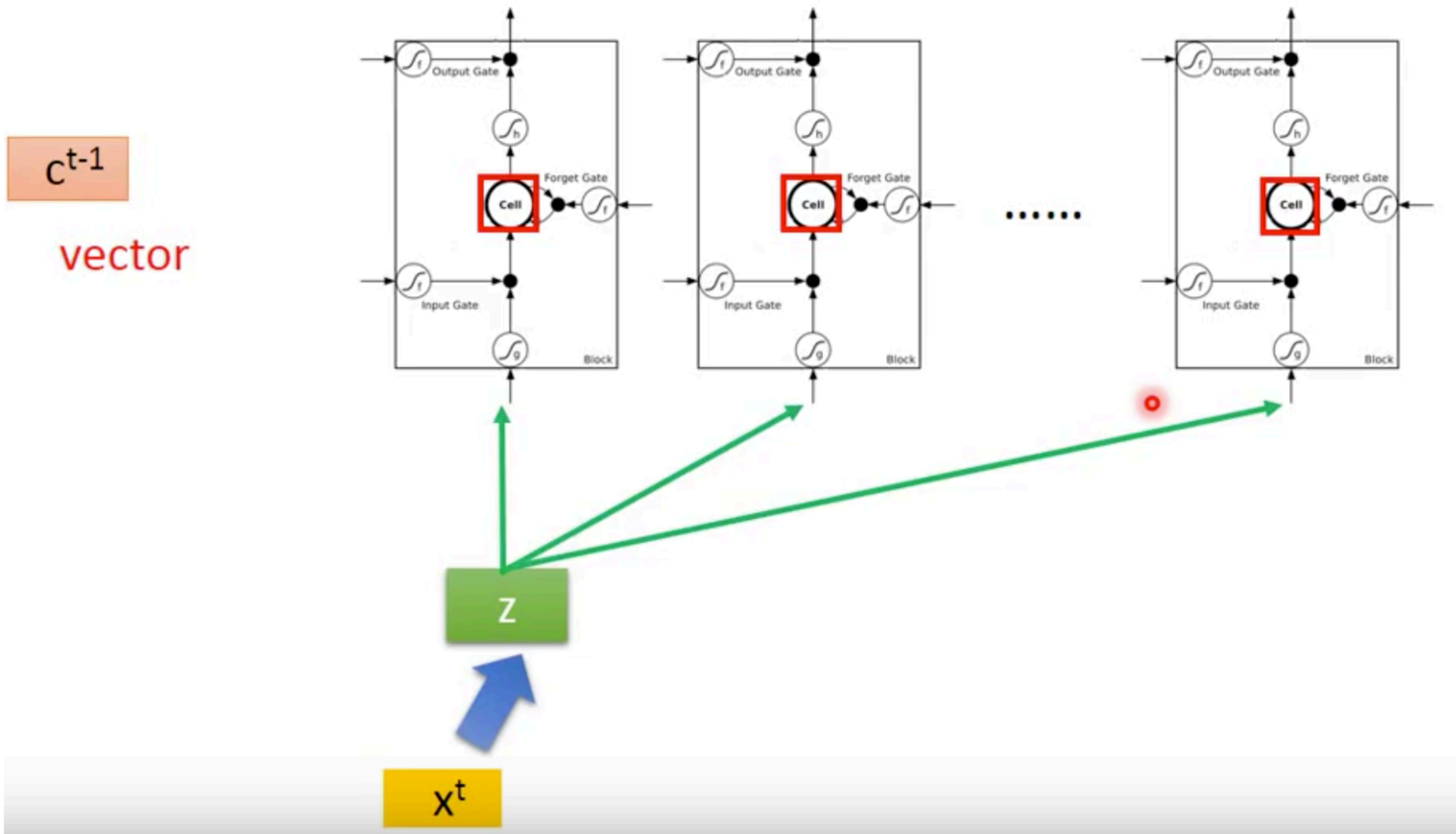
The vector of memory values at time t-1 (where each memory's value is a real number)



LSTM


Output Gate
Forget Gate
Cell
Input Gate
Block
......
c^{t-1}
vector
z
Linear transformation
(multiple input vector by a matrix to
get a vector z)
Input vector at
time t
x^t

LSTM

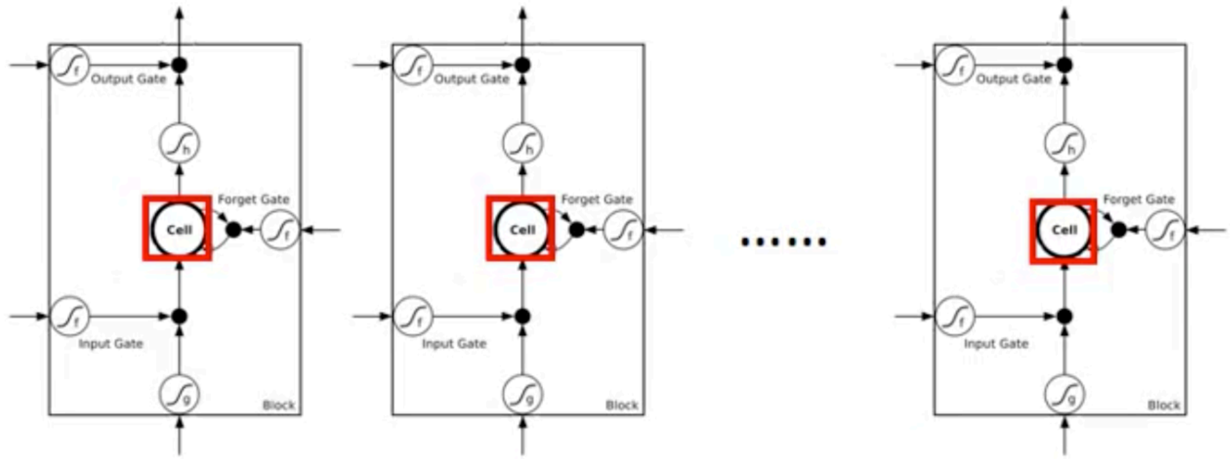


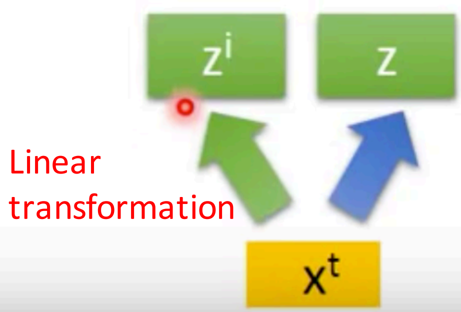
c^{t-1}
vector
Output Gate
Forget Gate
Cell
Input Gate
Block
......
z
x^t

LSTM

c^{t-1}

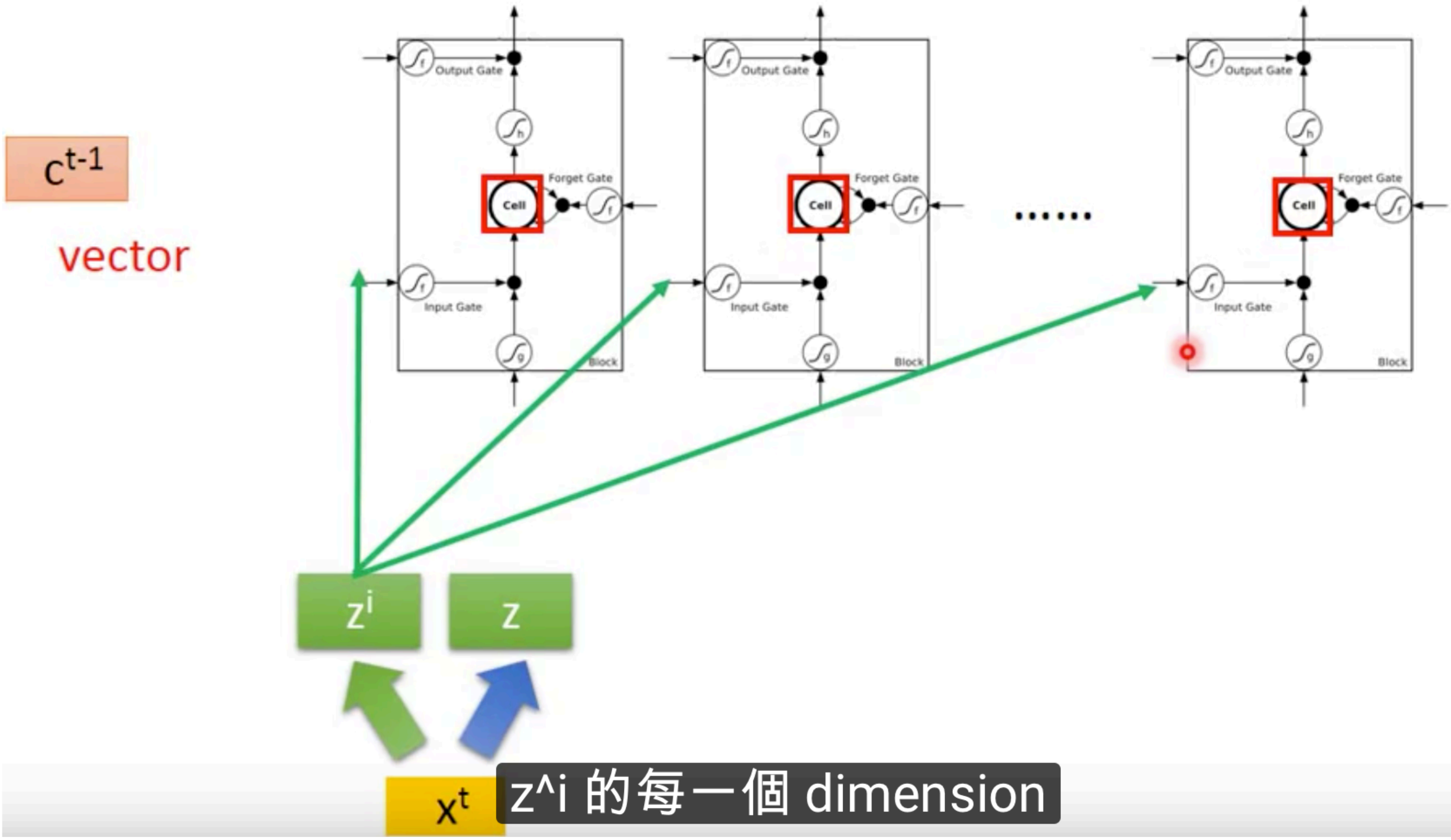
vector





得到 z^i

LSTM

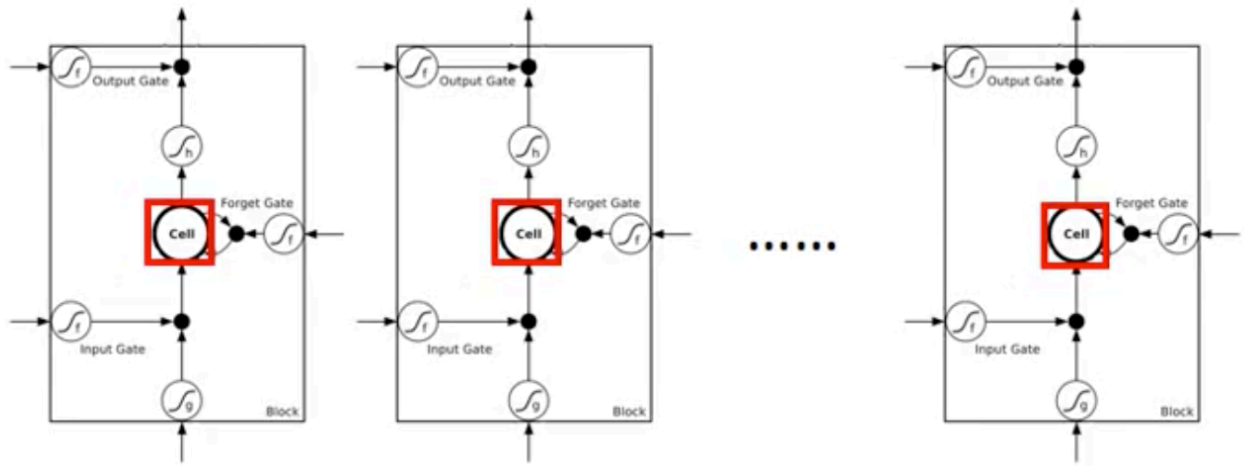


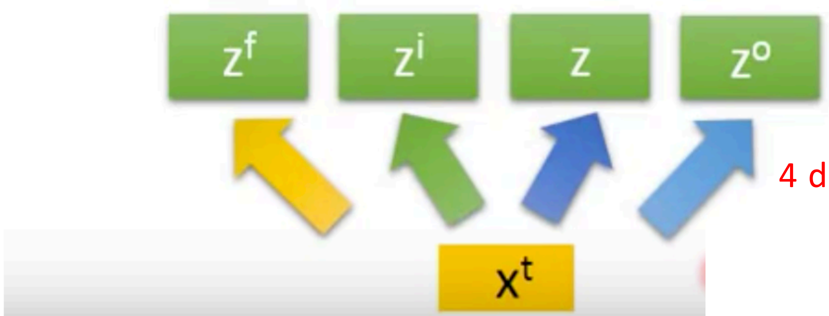
z^i 的每一個 dimension

LSTM

c^{t-1}

vector

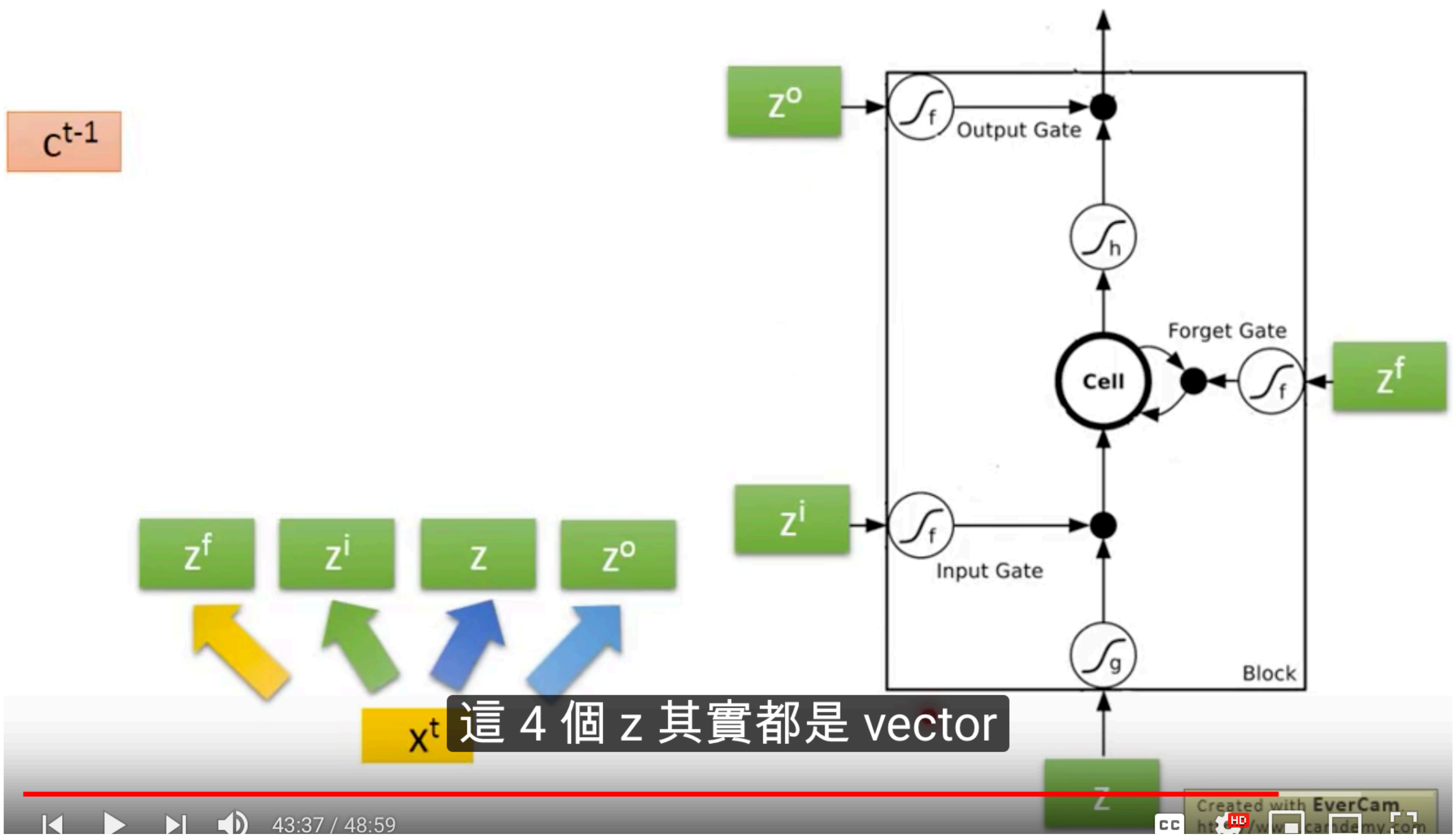




4 vectors

4 different linear transformations

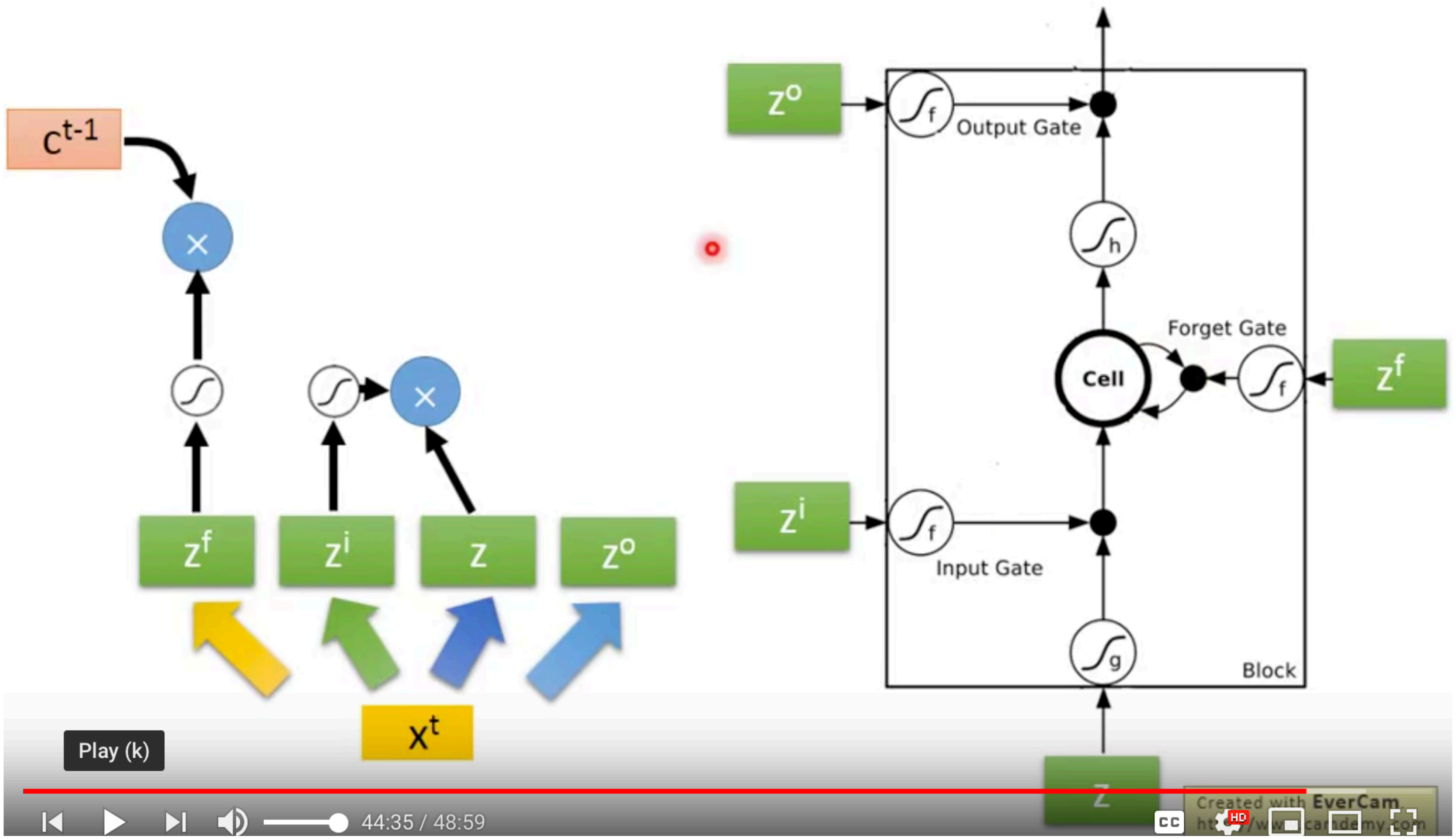
LSTM

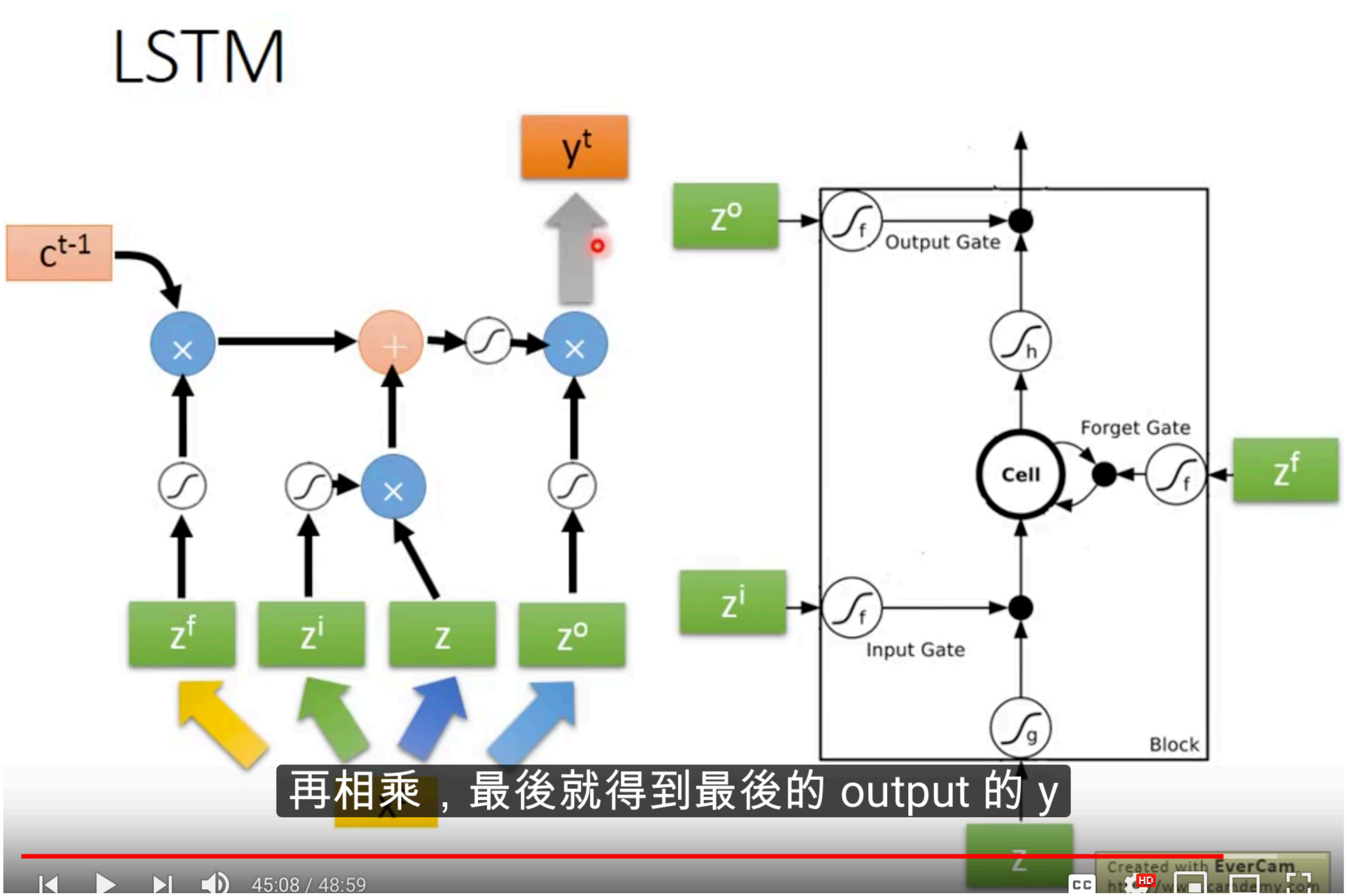


c^{t-1}
z^f
z^i
z
z^o
x^t
這 4 個 z 其實都是 vector
z^o
Output Gate
z^i
Input Gate
Forget Gate
Cell
z^f
Block
z
43:37 / 48:59

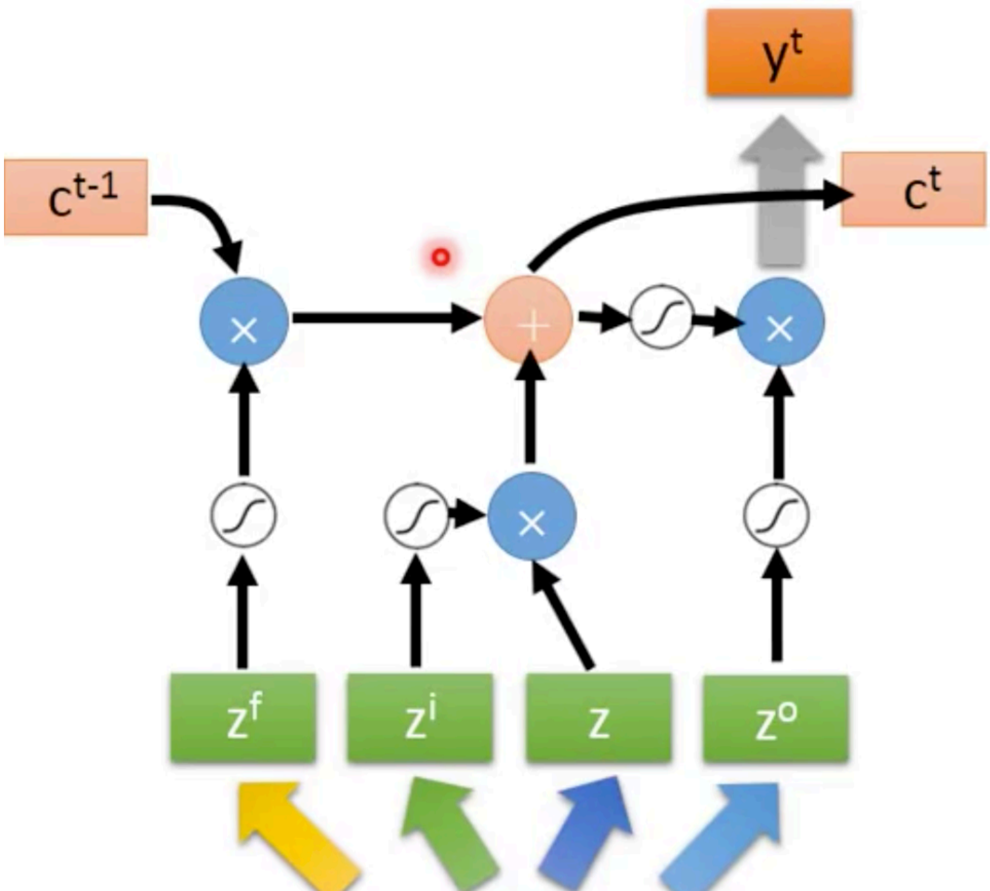
LSTM

Element-wise vector operation for the layer



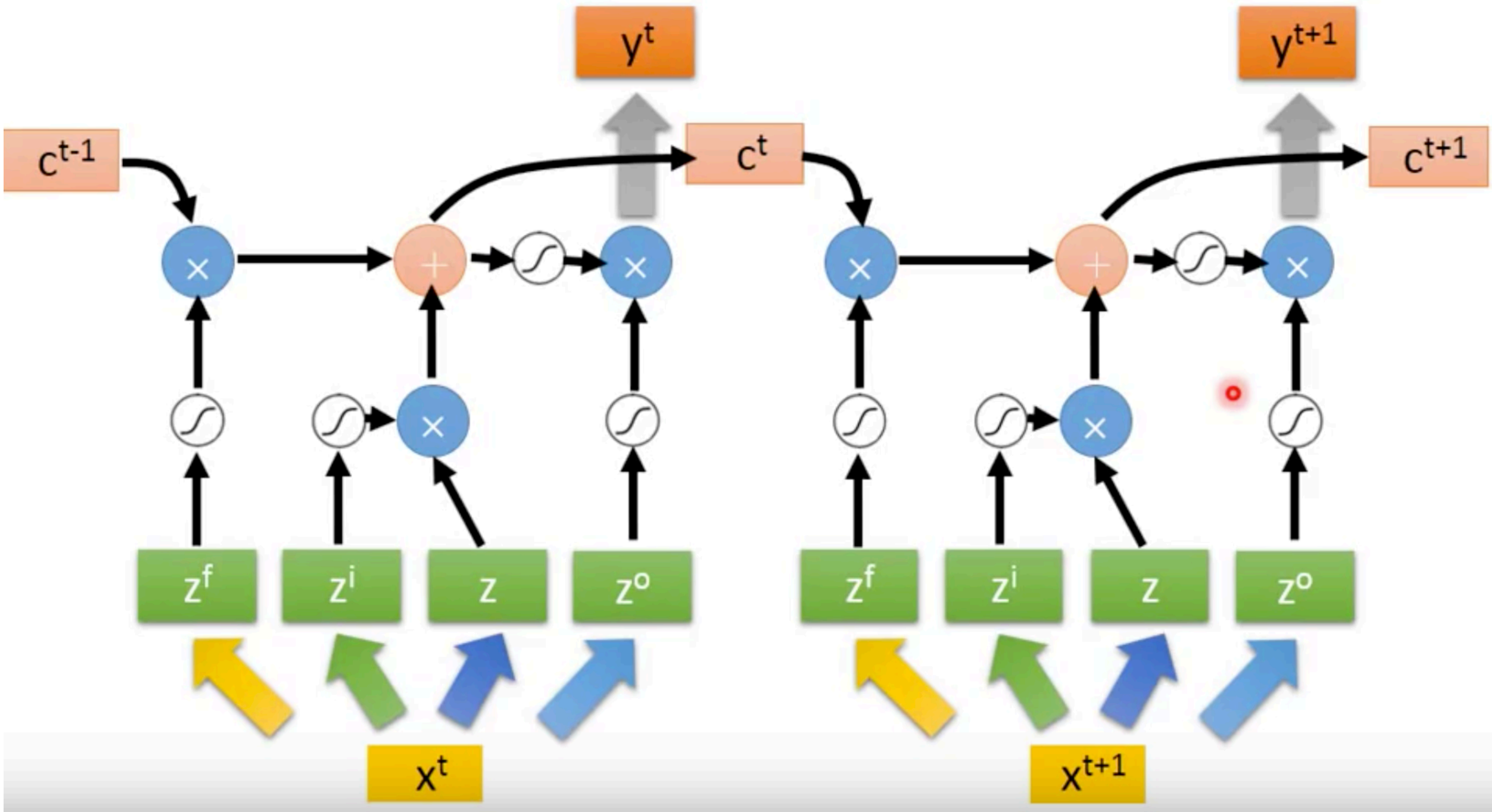

LSTM
c^{t-1}
y^t
z^f
z^i
z
z^o
Output Gate
Forget Gate
Cell
Input Gate
Block
再相乘，最後就得到最後的 output 的 y
45:08 / 48:59
Created with EverCam

LSTM



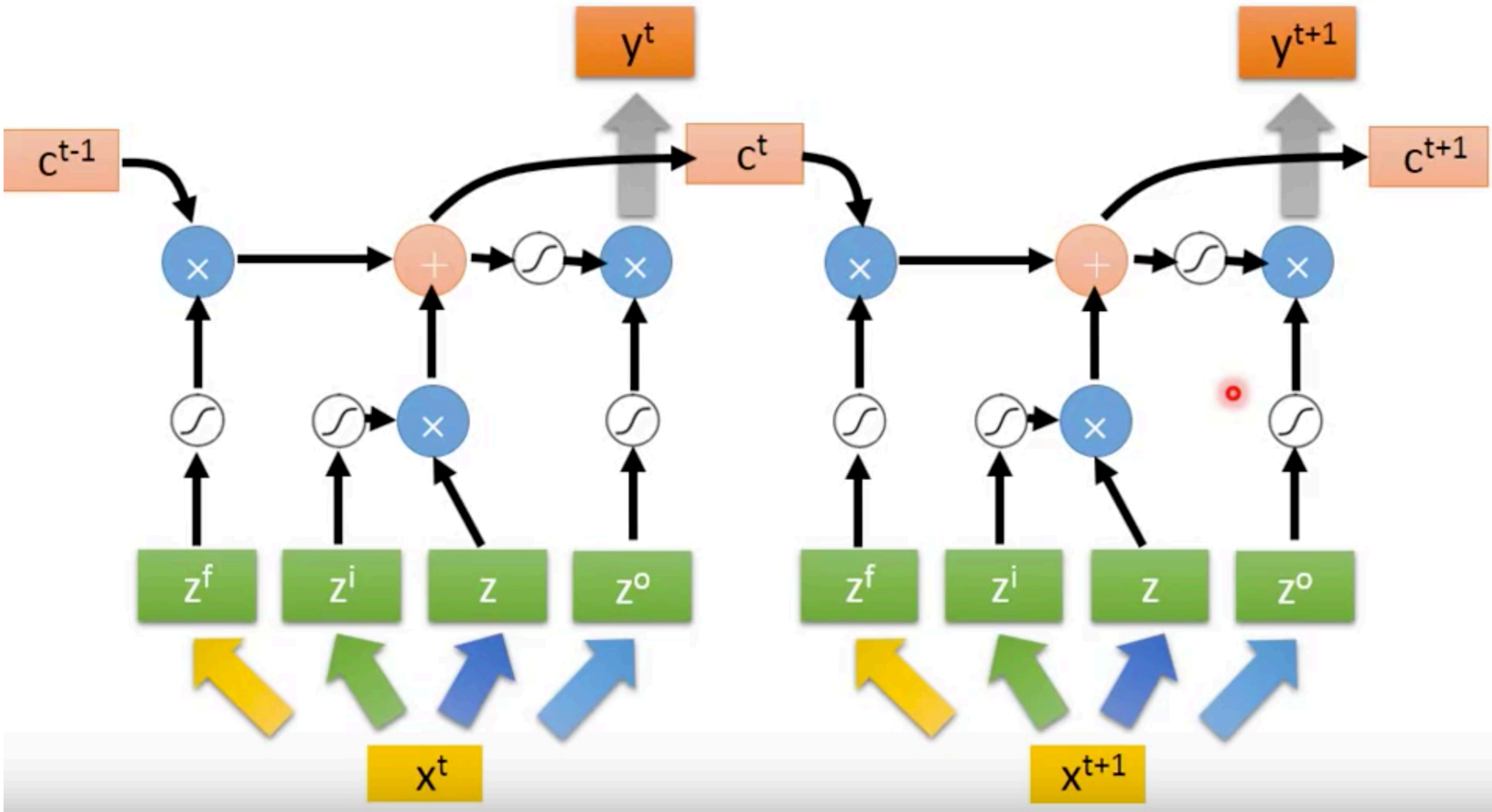
這個時候相加以後的結果，就是 memory 裡面存的值

LSTM



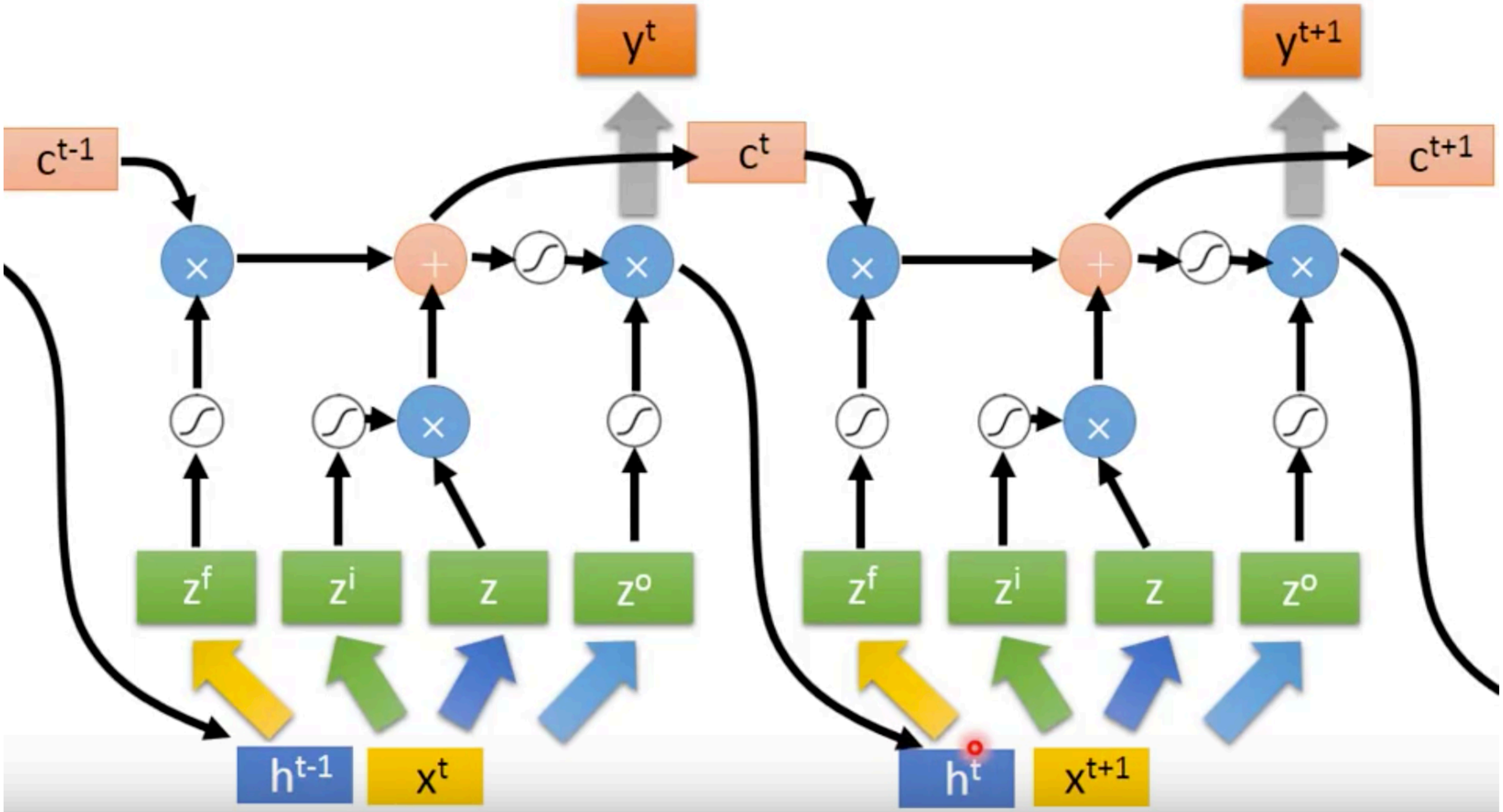
This is still just a simple version of LSTM

LSTM



LSTM

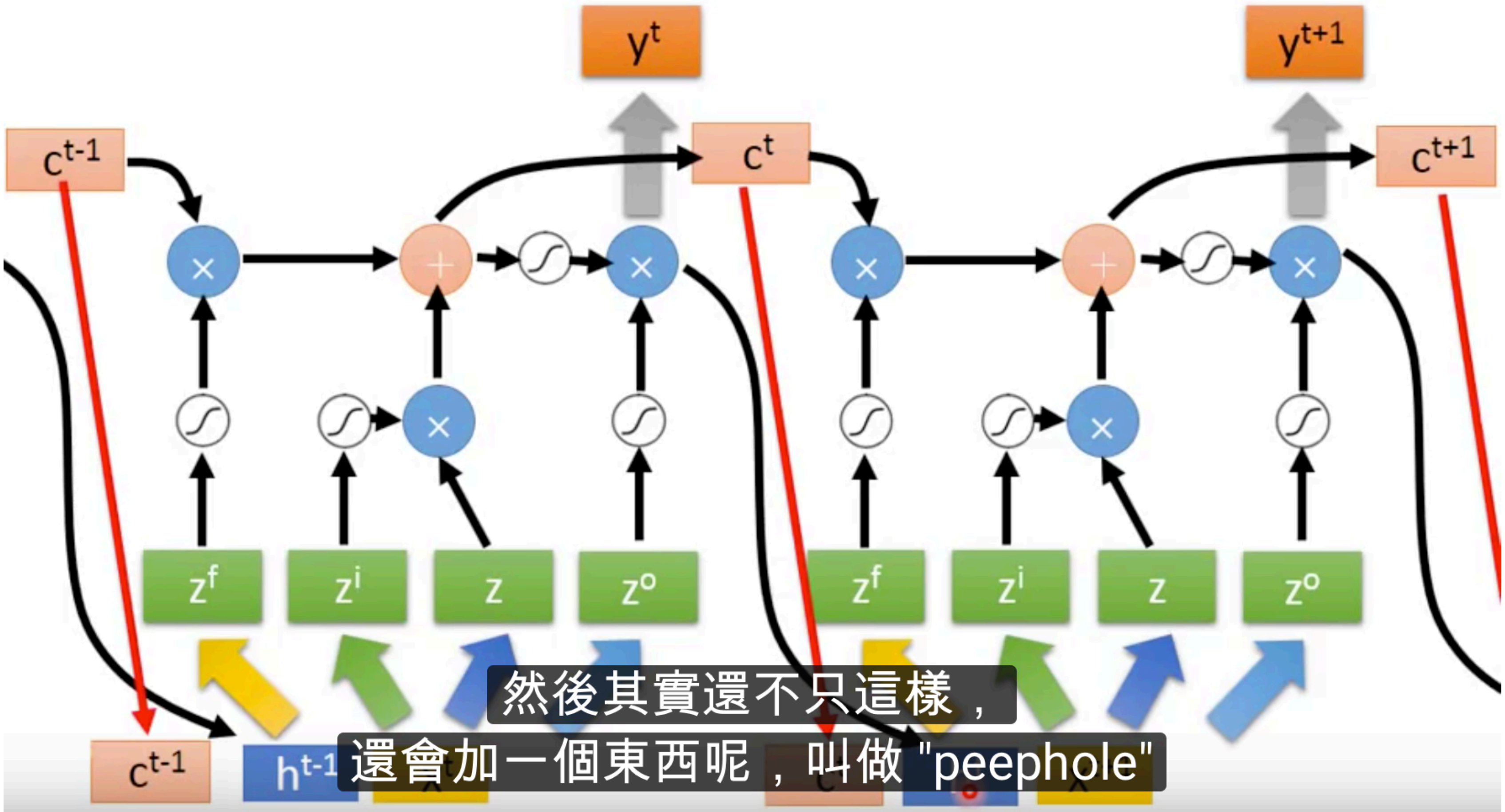
Real LSTM



LSTM

Real LSTM

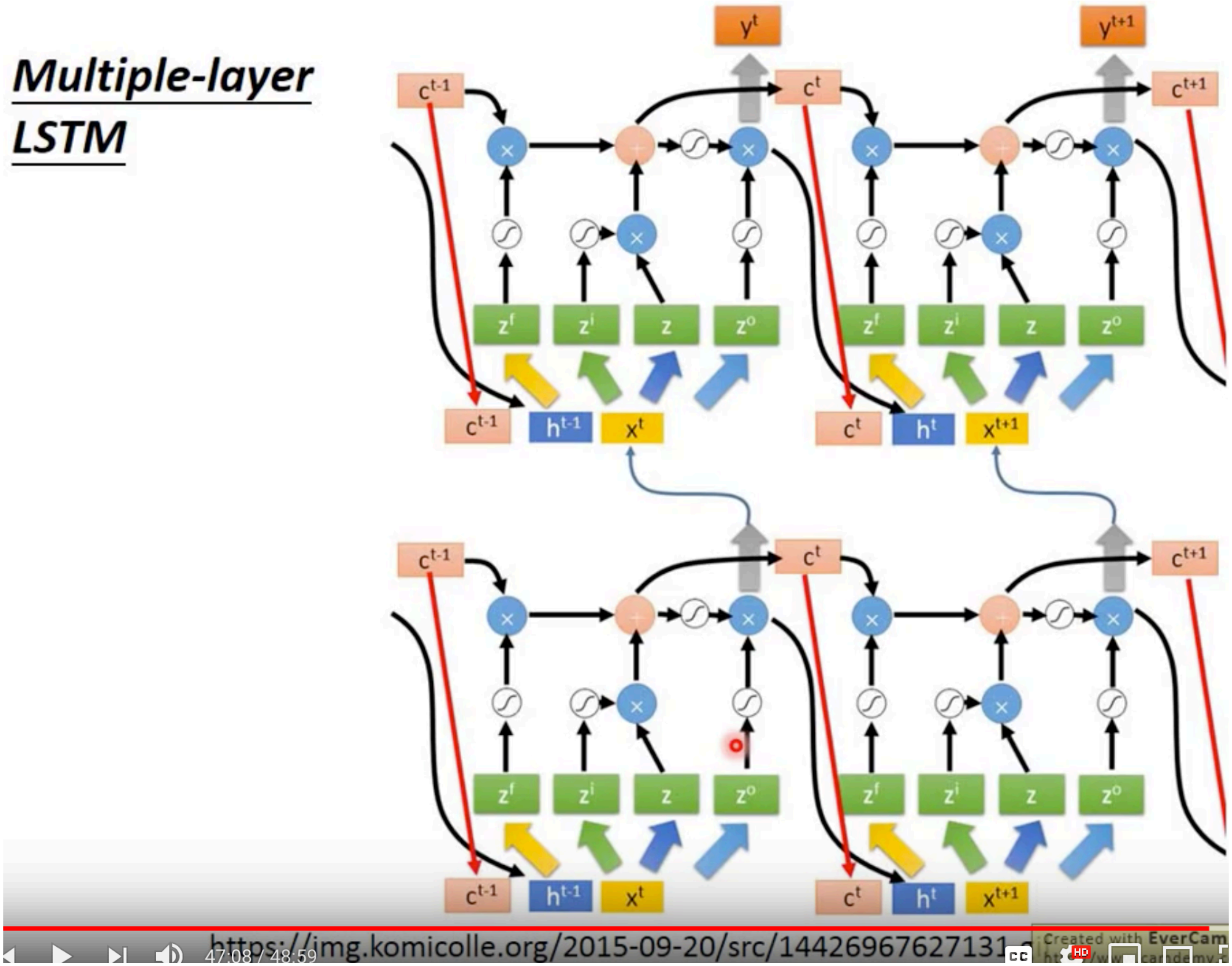
Extension: “peephole”



然後其實還不只這樣，

還會加一個東西呢，叫做 "peephole"

Multiple-layer LSTM

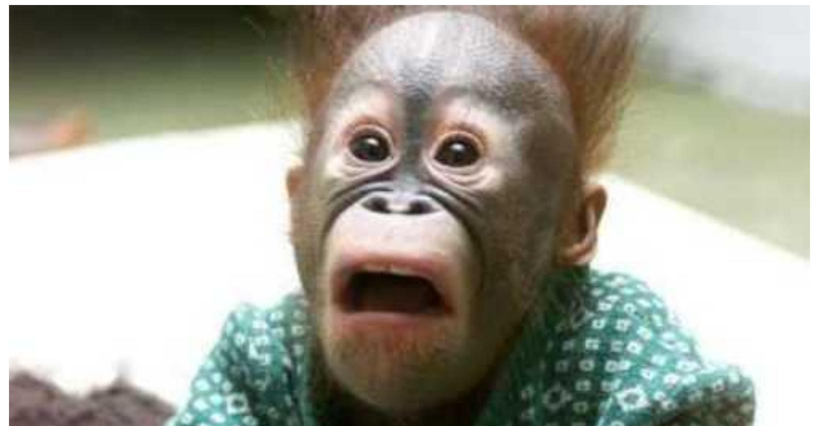


The first time a person sees

LSTM

The first time a person sees

LSTM



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

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