MAE 688: Machine Learning for Mechanical Engineers

Lecture 5-Recurrent Neural Network/LSTM

Dr. Bing Dong

Director, Built Environment Science and Technology (BEST) Lab
Professor
Mechanical and Aerospace Engineering

Syracuse University

Motivation

"Humans don't start their thinking from scratch every second. As you read this sentences, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence."

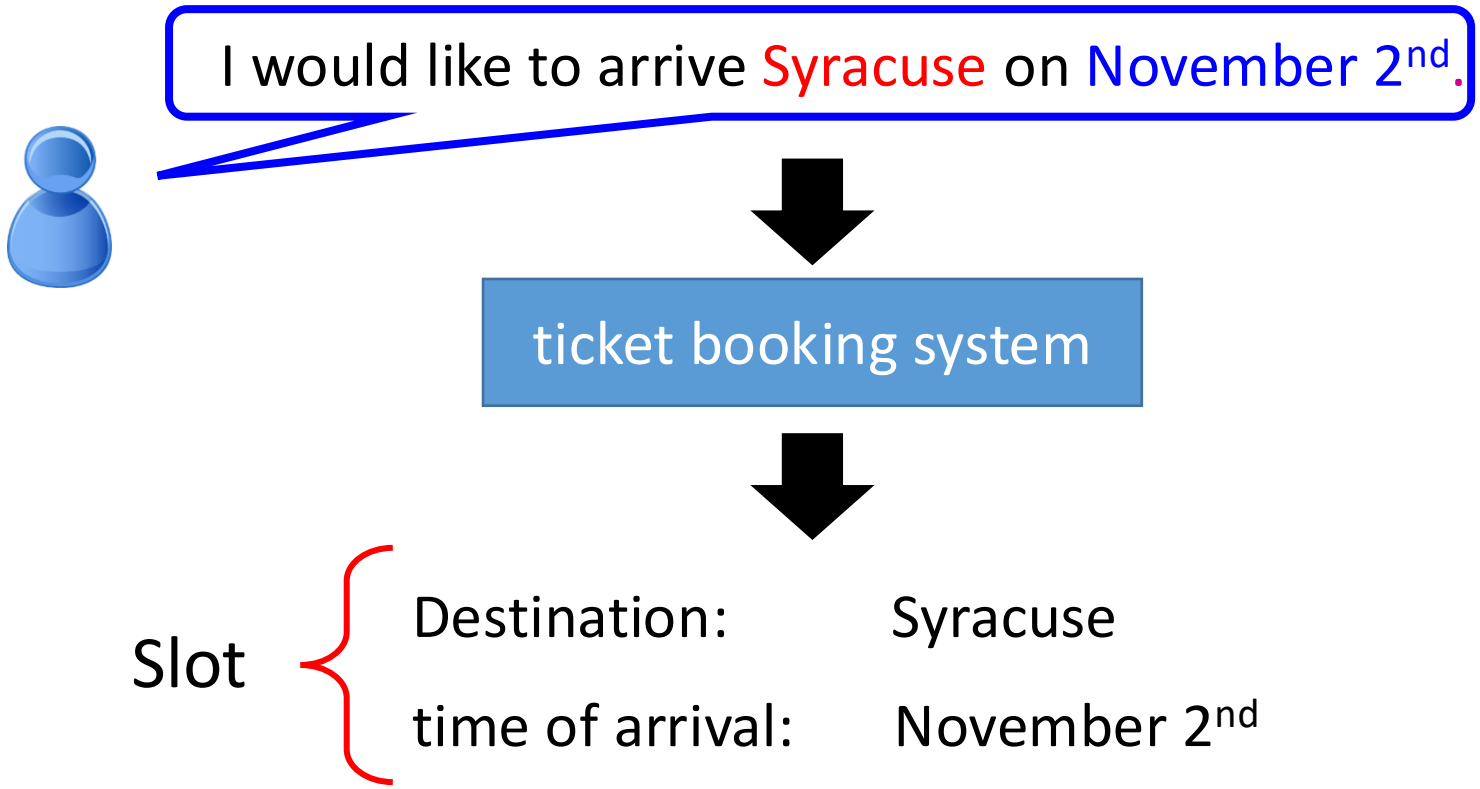
Motivation

"Humans don't start their thinking from scratch every second. As you read this sentences, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence."

"Traditional neural networks can't do this"

Example Application—Speech Recognition

- Slot Filling

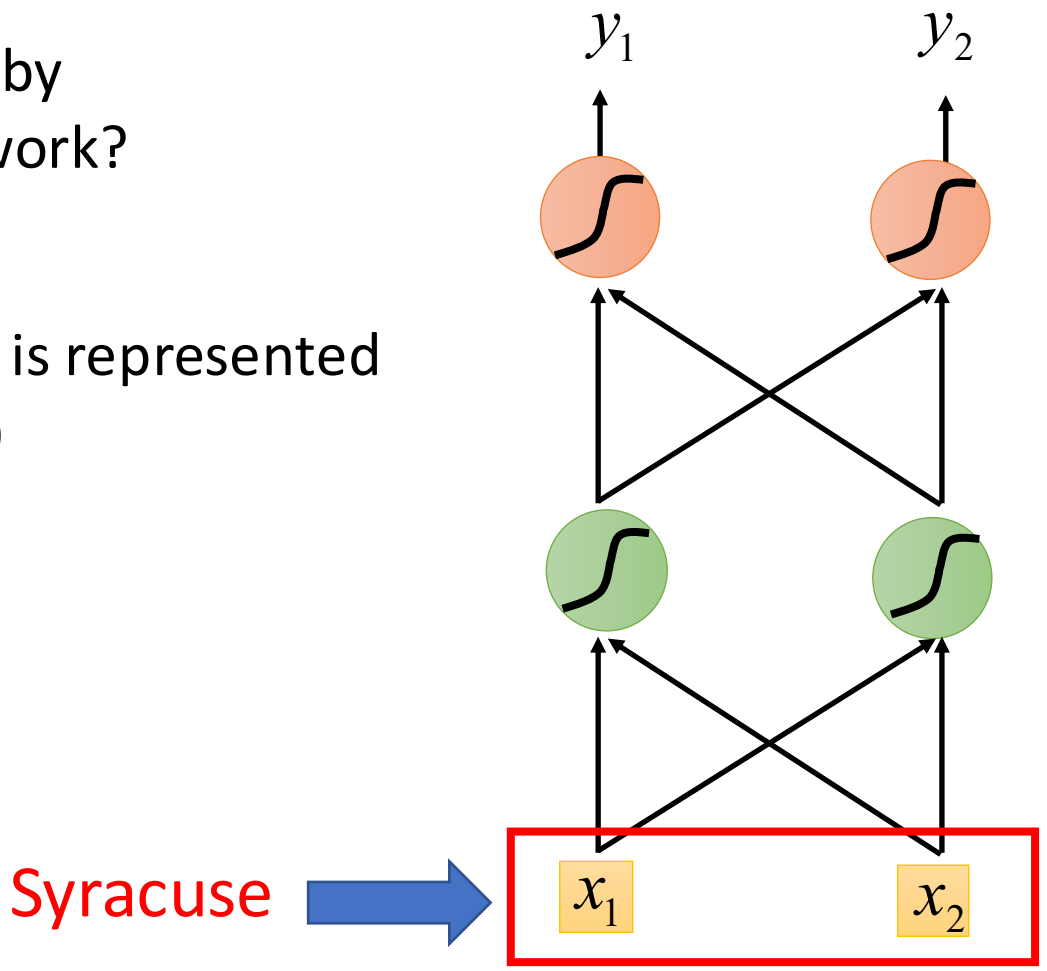


Example Application—Speech Recognition

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.

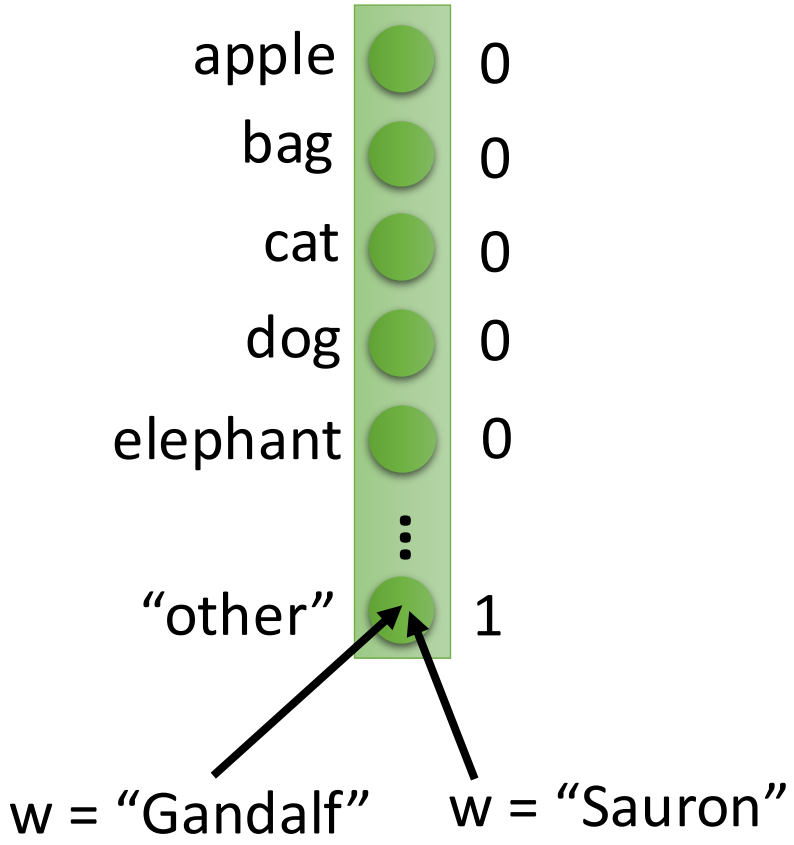
Each dimension corresponds to a word in the lexicon

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

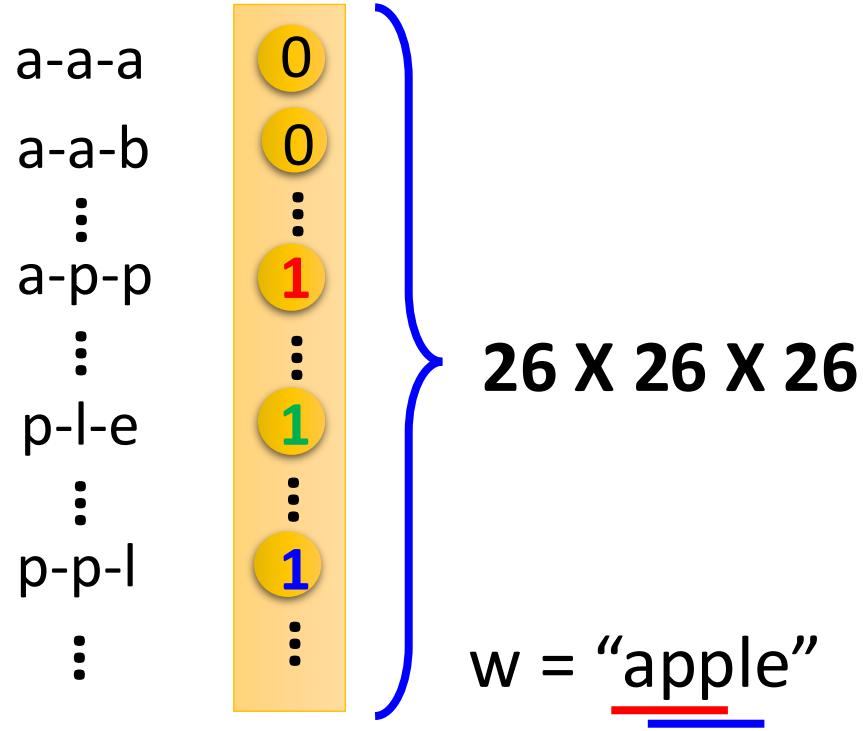
apple = [1 0 0 0 0]
bag = [0 1 0 0 0]
cat = [0 0 1 0 0]
dog = [0 0 0 1 0]
elephant = [0 0 0 0 1]

Beyond 1-of-N encoding

Dimension for "Other"



Word hashing



Example Application

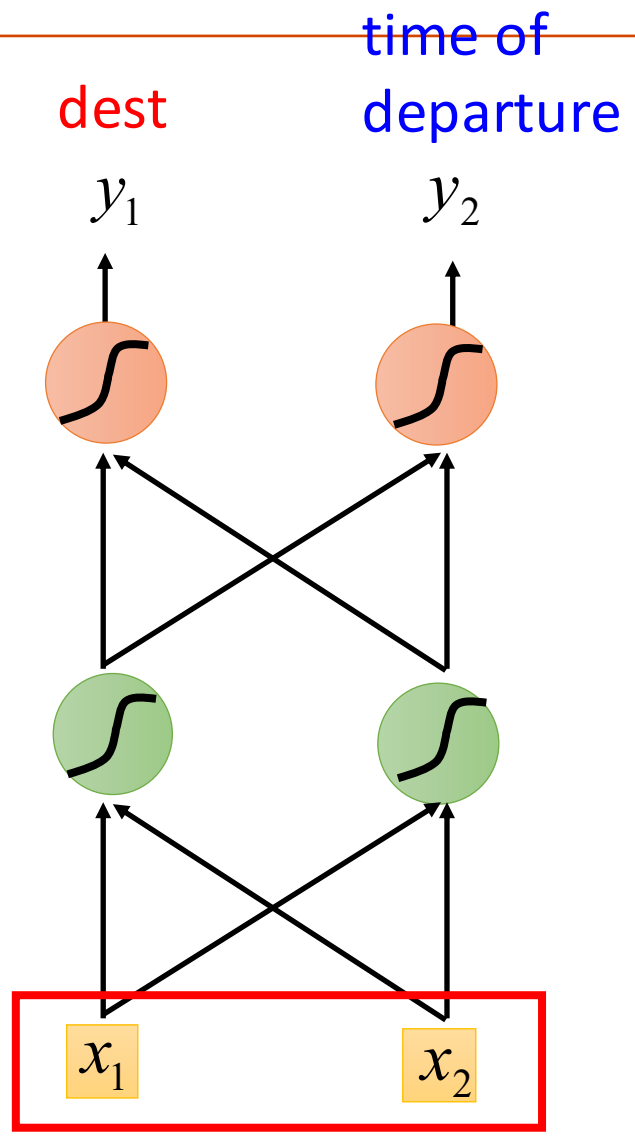
Solving slot filling by Feedforward network?

Input: a word

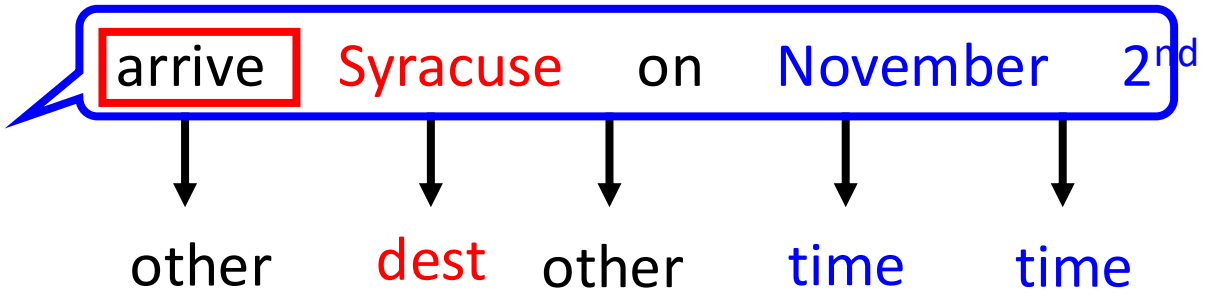
(Each word is represented as a vector)

Output:

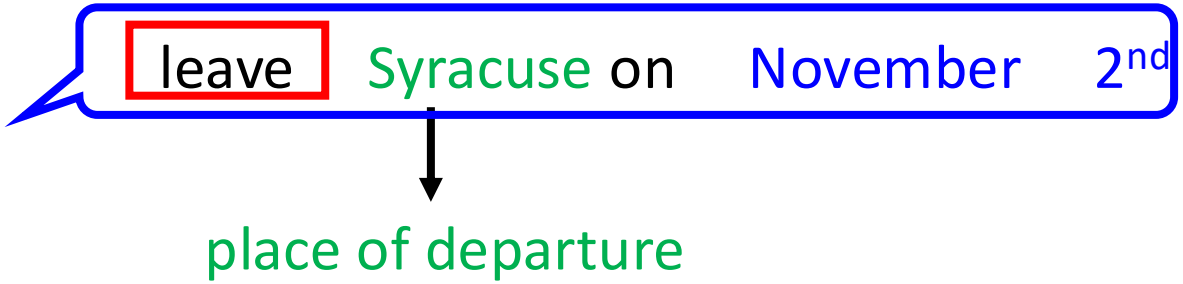
Probability distribution that the input word belonging to the slots



Example Application

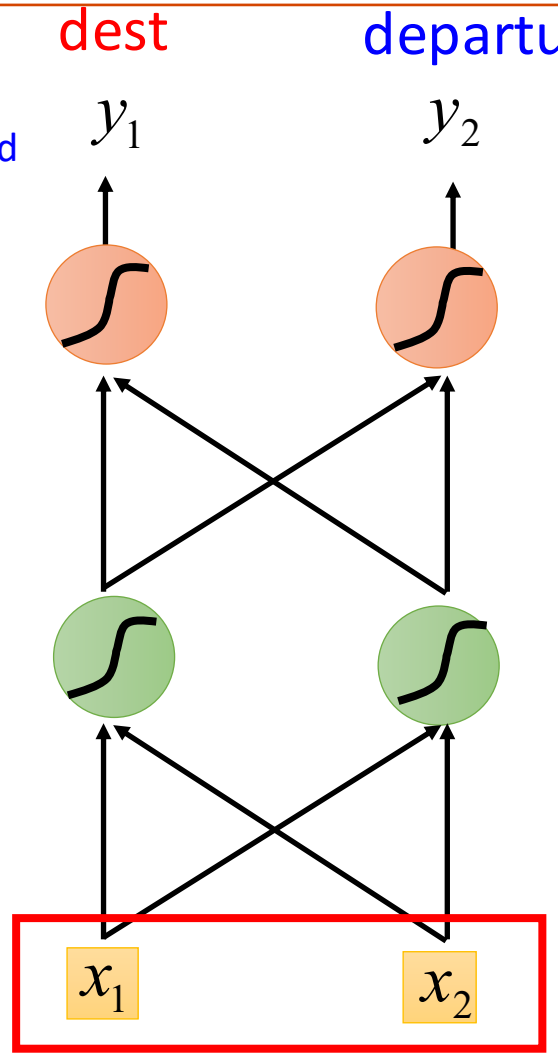


Problem?



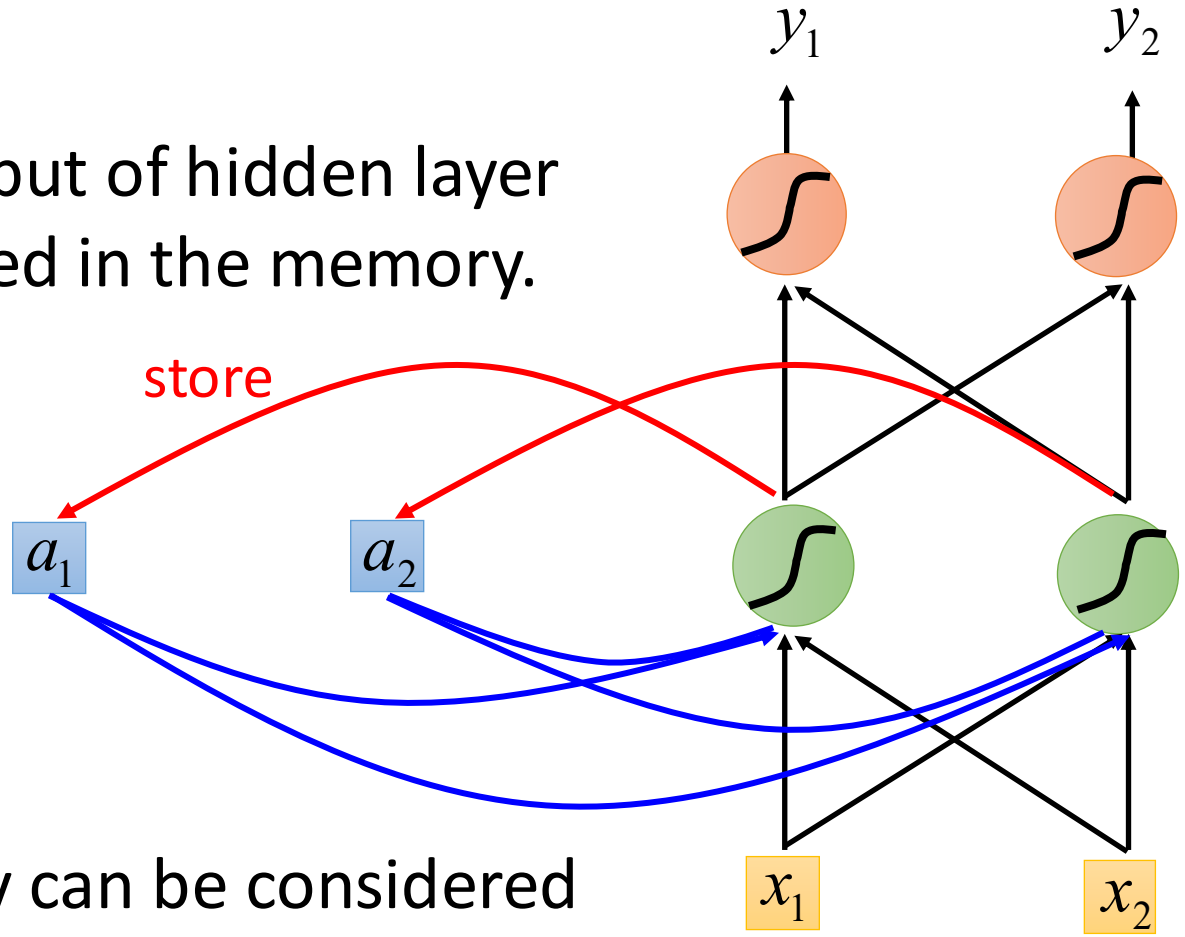
Neural network needs memory!

Syracuse



Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.

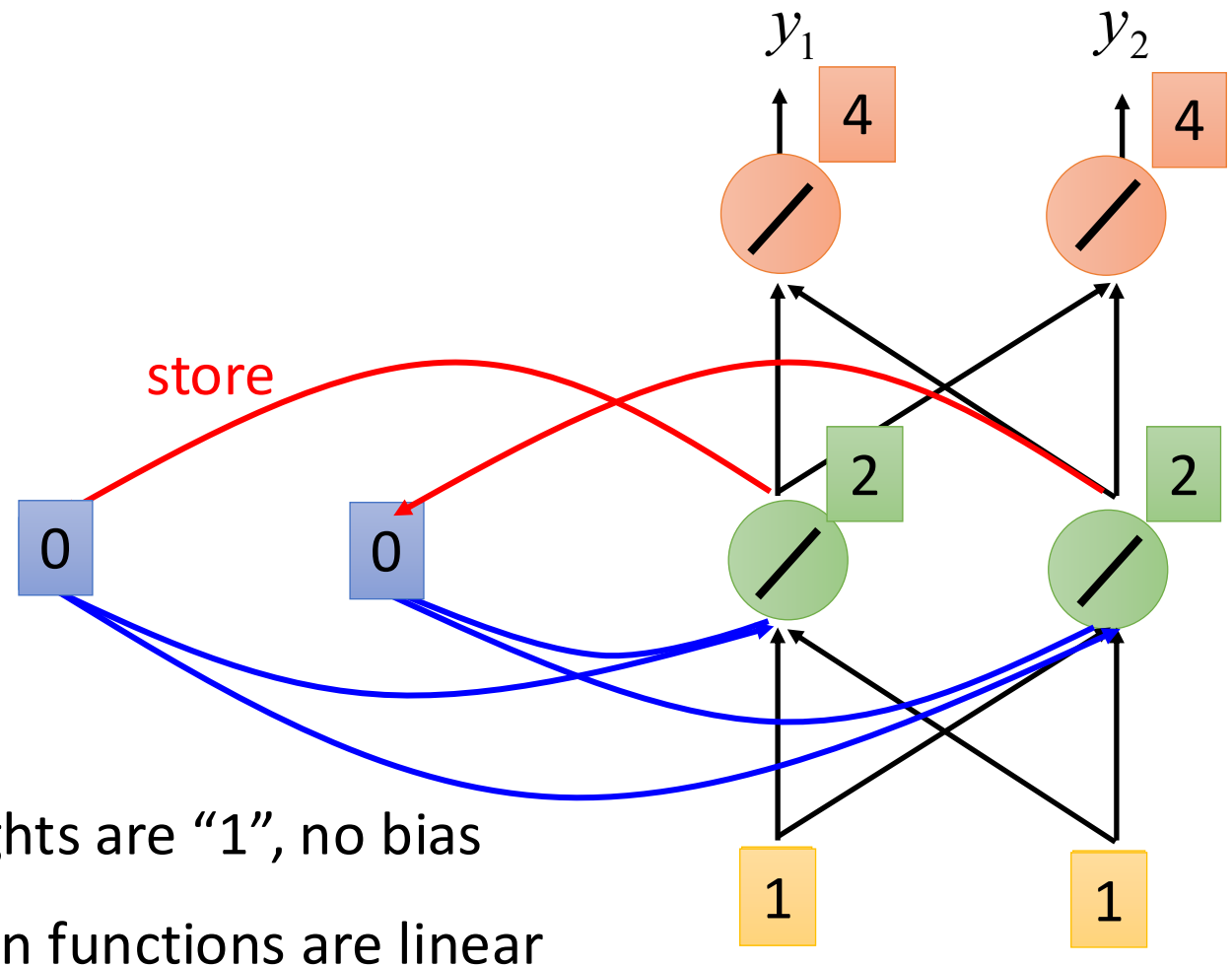


Memory can be considered as another input.

Example

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

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



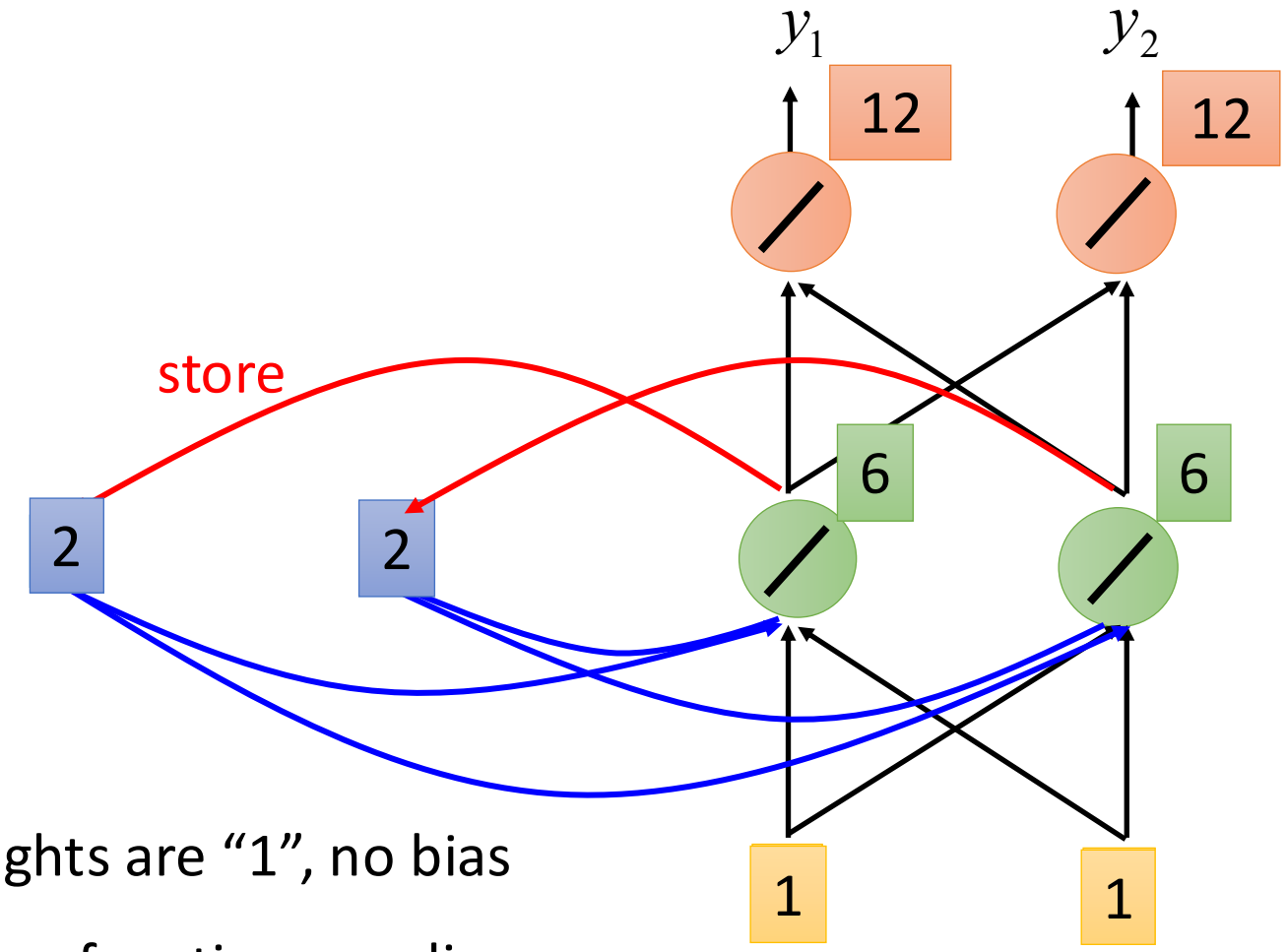
All the weights are “1”, no bias

All activation functions are linear

Example

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

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



All the weights are “1”, no bias

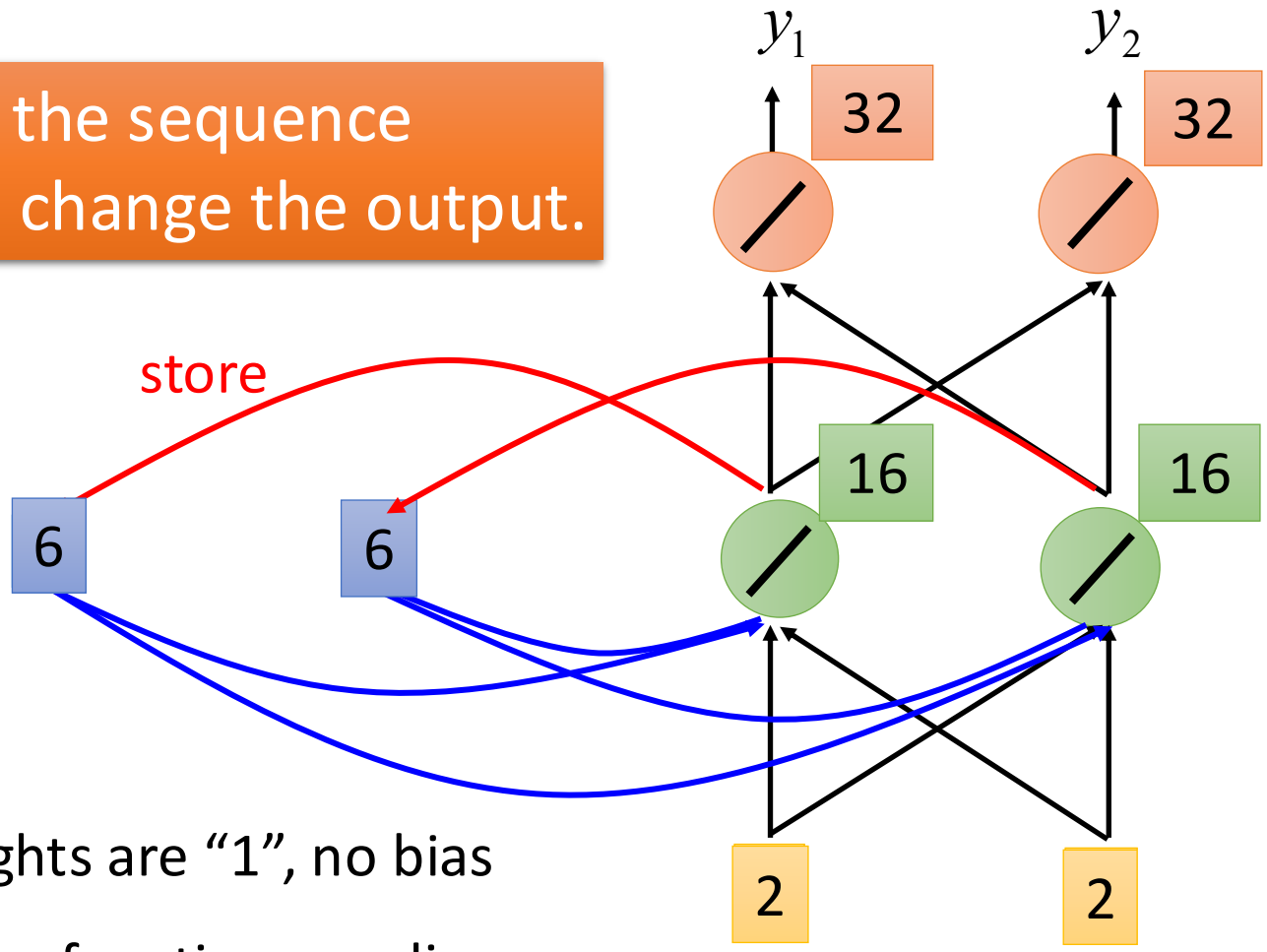
All activation functions are linear

Example

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

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

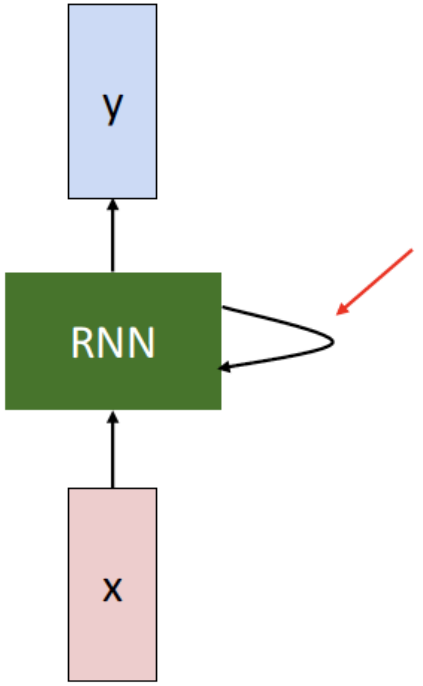
Changing the sequence order will change the output.



All the weights are “1”, no bias

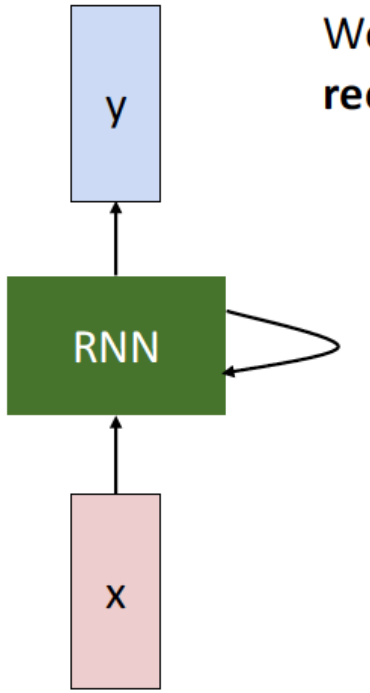
All activation functions are linear

RNN Architecture



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

RNN Architecture



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state

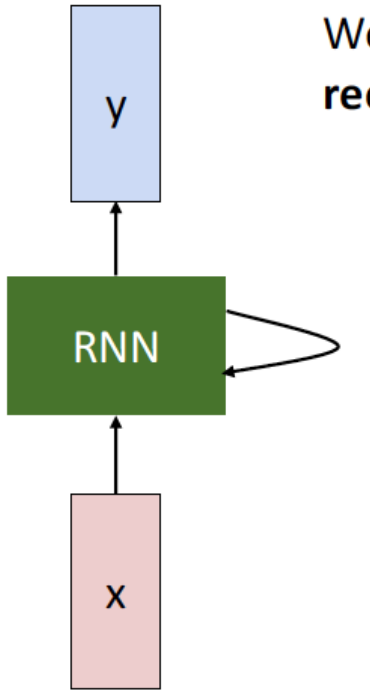
old state

input vector at some time step

some function with parameters W

RNN Architecture

Notice: the same function and the same set of parameters are used at every time step.



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

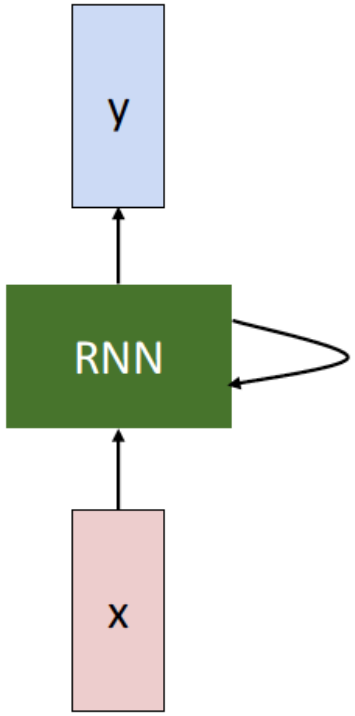
new state

some function with parameters W

old state

input vector at some time step

RNN Architecture



The state consists of a single *"hidden"* vector **h**:

$$h_t = f_W(h_{t-1}, x_t)$$

(also bias term)

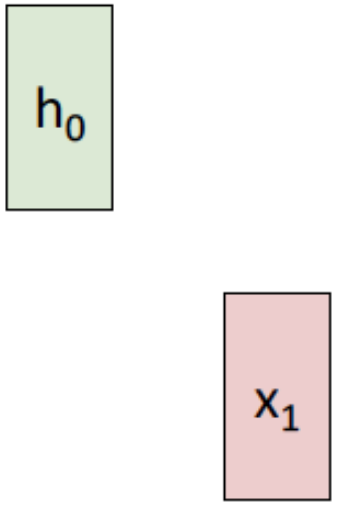
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

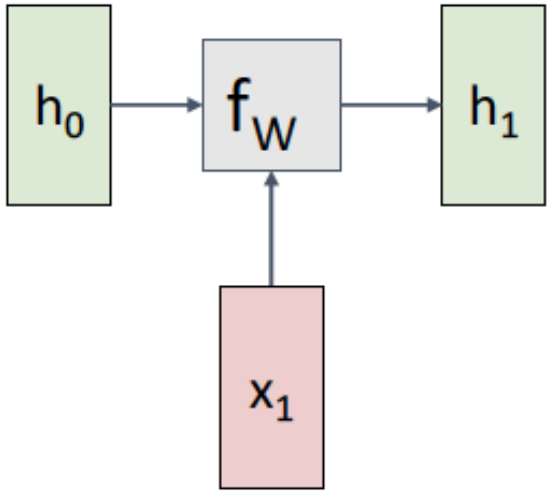
Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

RNN Computational Graph

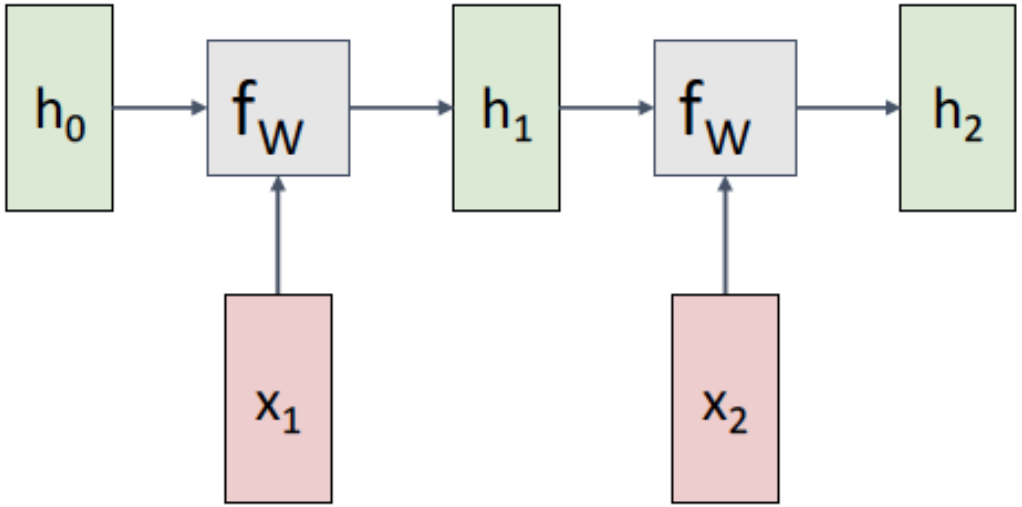
Initial hidden state
Either set to all 0,
Or learn it



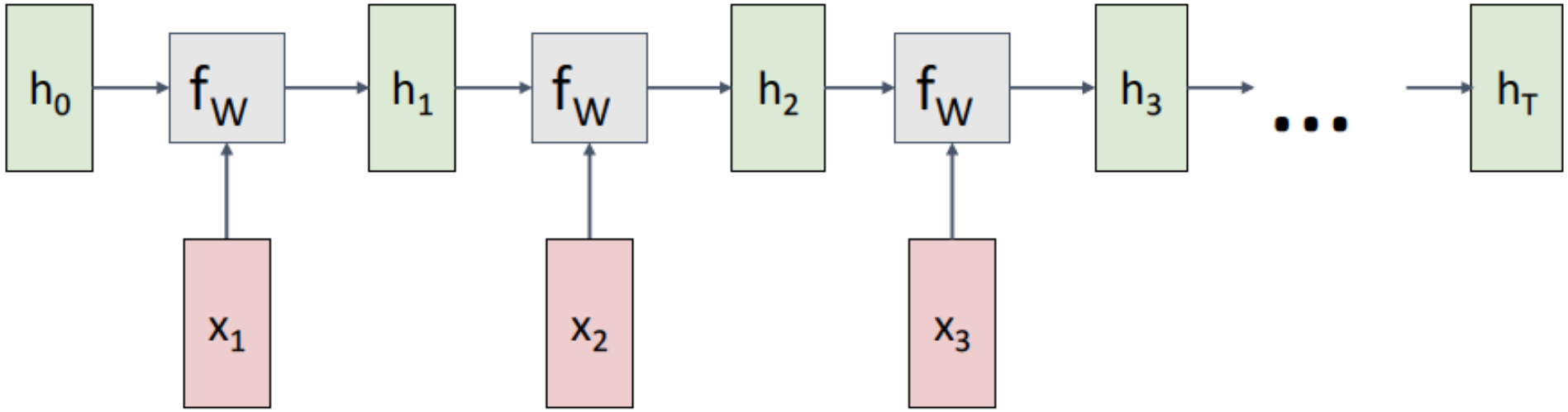
RNN Computational Graph



RNN Computational Graph

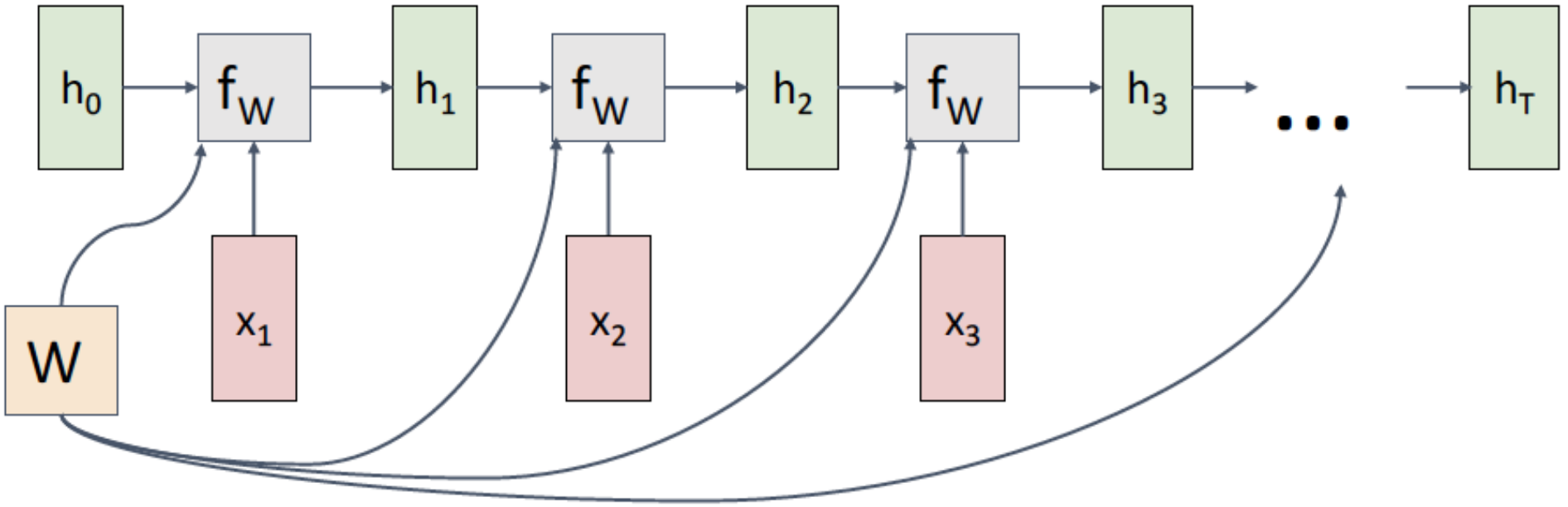


RNN Computational Graph

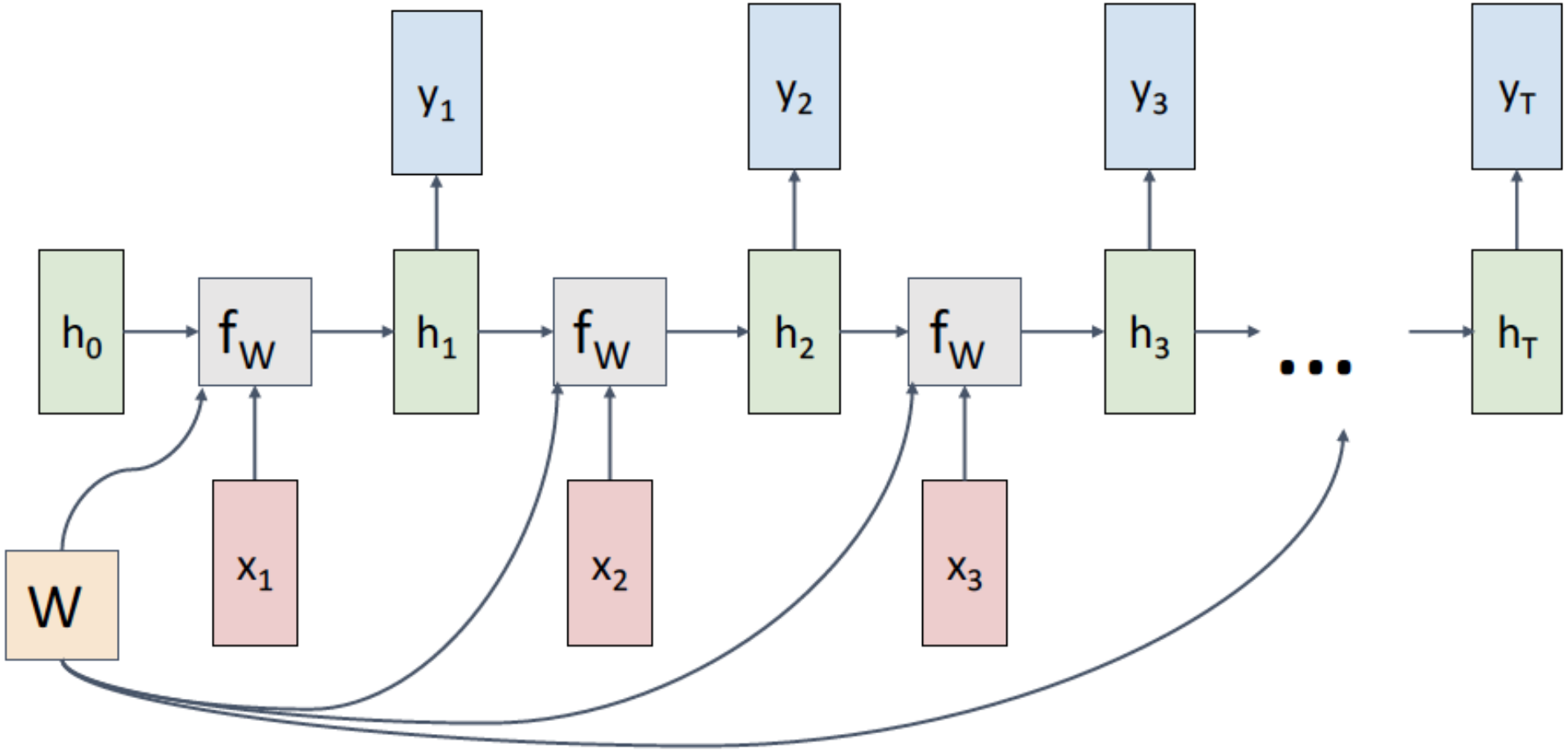


RNN Computational Graph

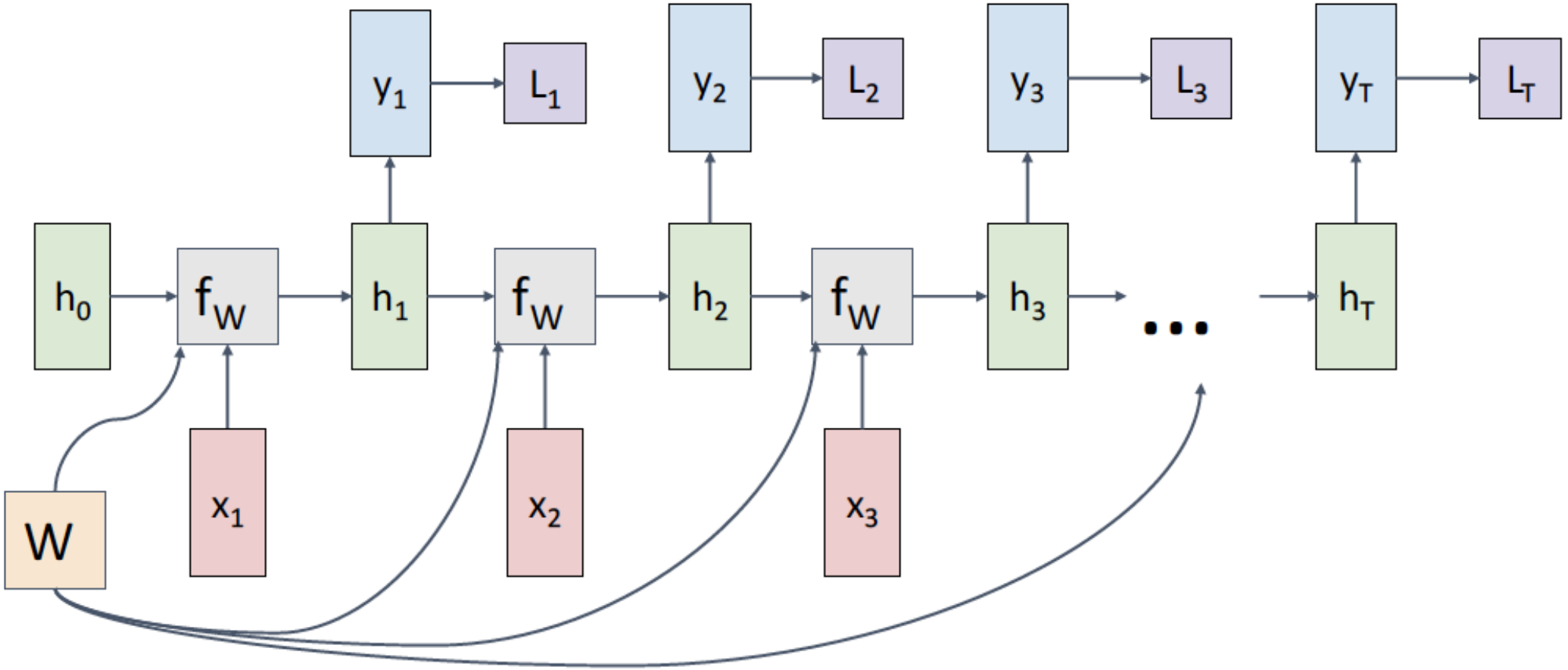
Re-use the same weight matrix at every time-step



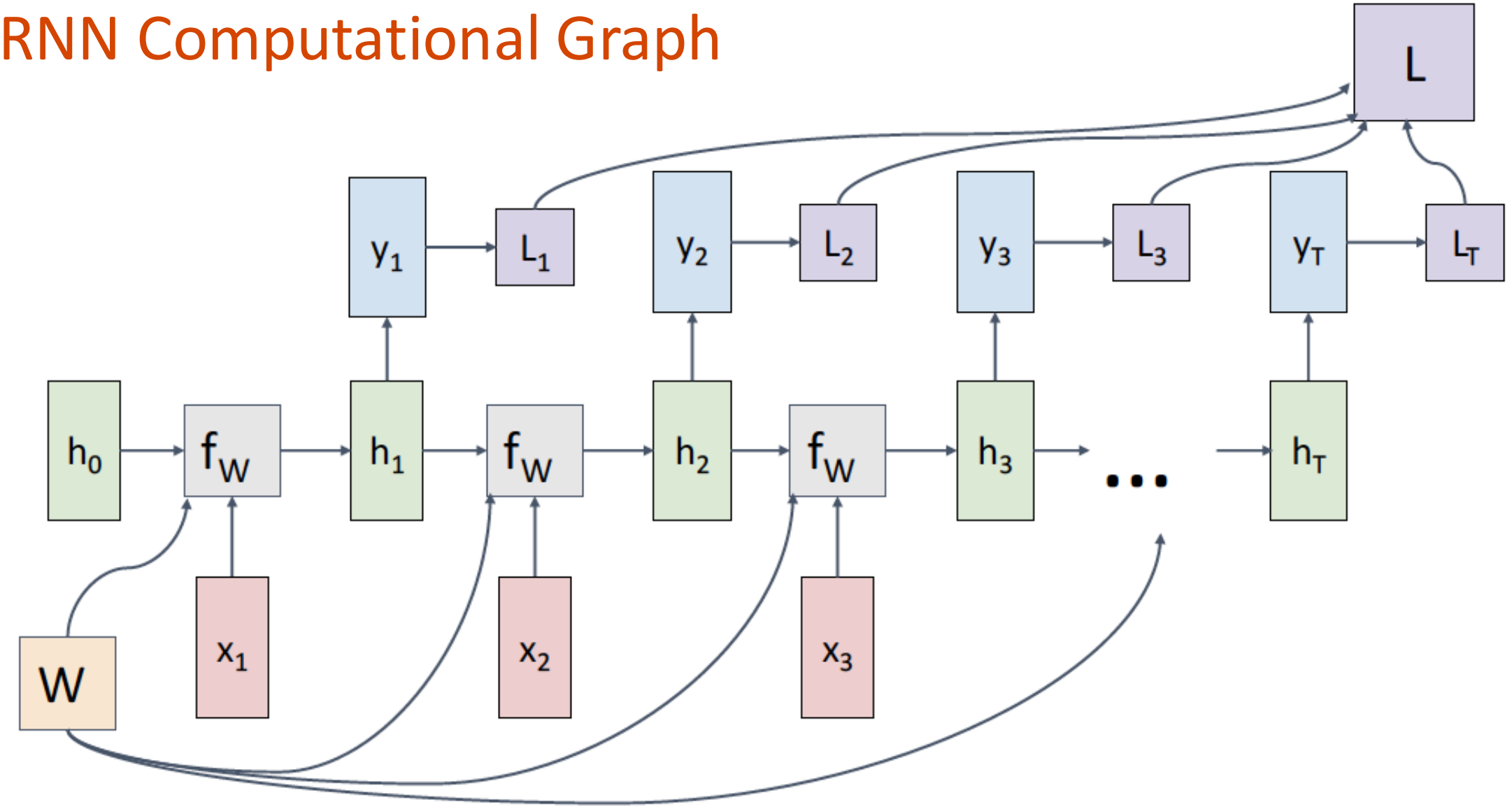
RNN Computational Graph (many to many)



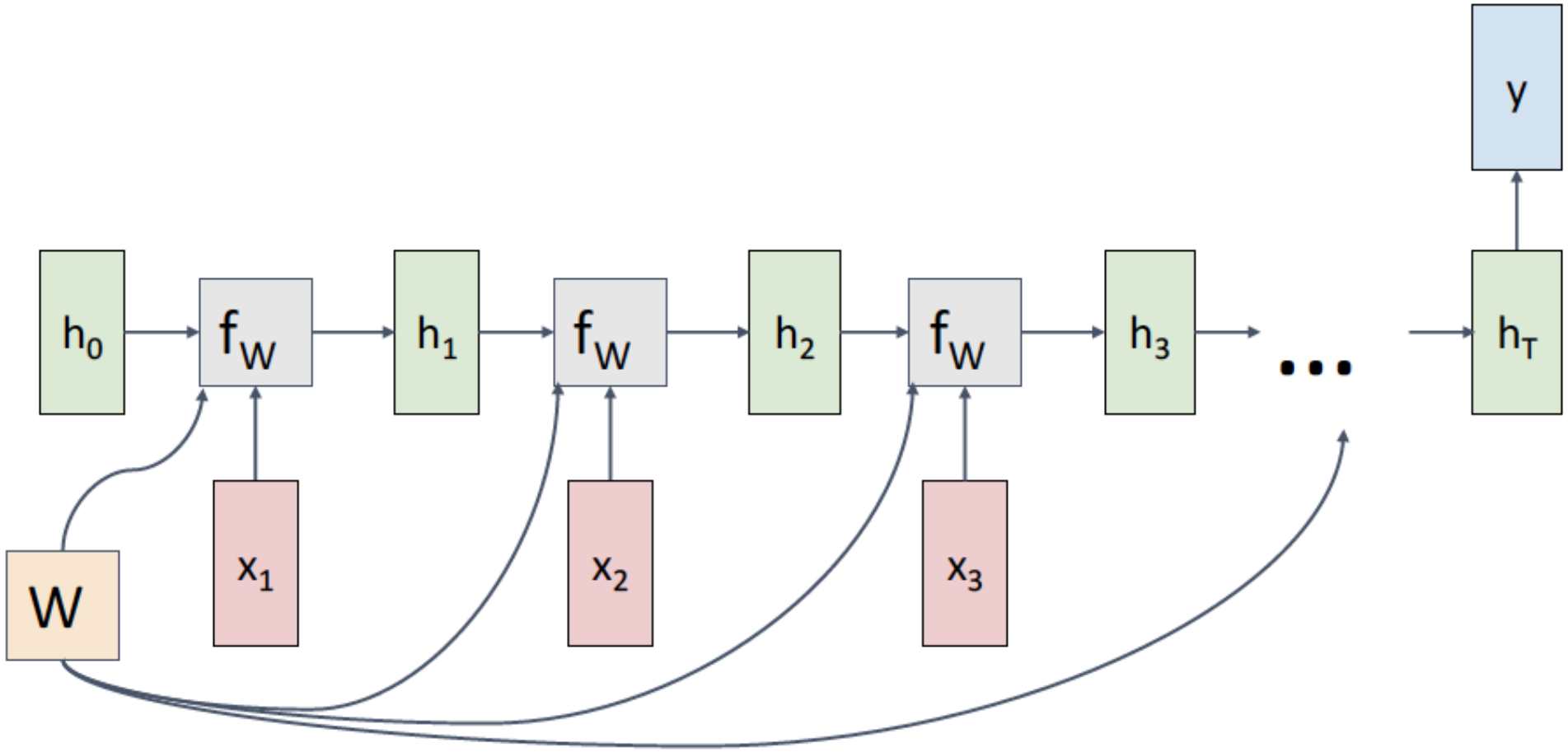
RNN Computational Graph (many to many)



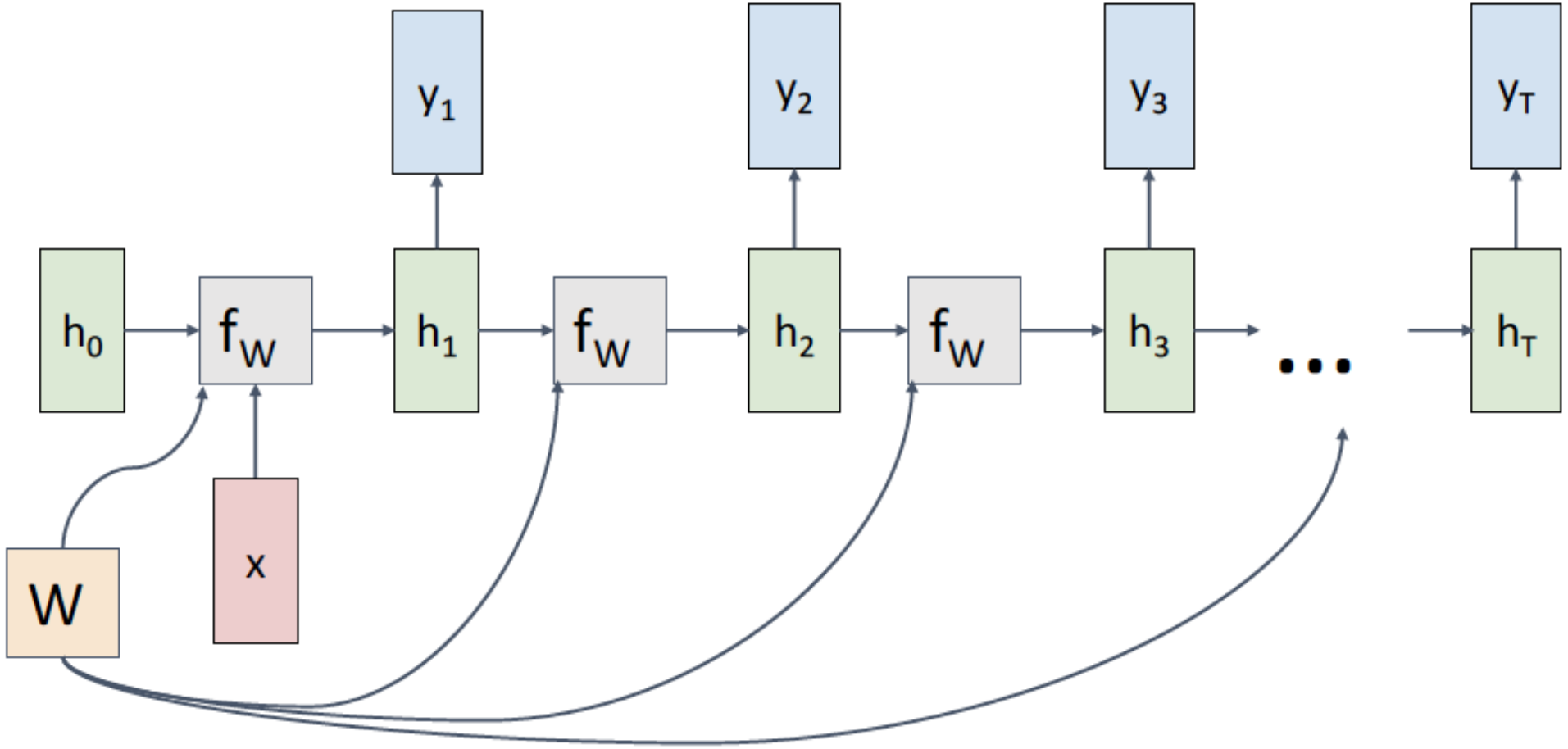
RNN Computational Graph



RNN Computational Graph (many to one)



RNN Computational Graph (one to Many)



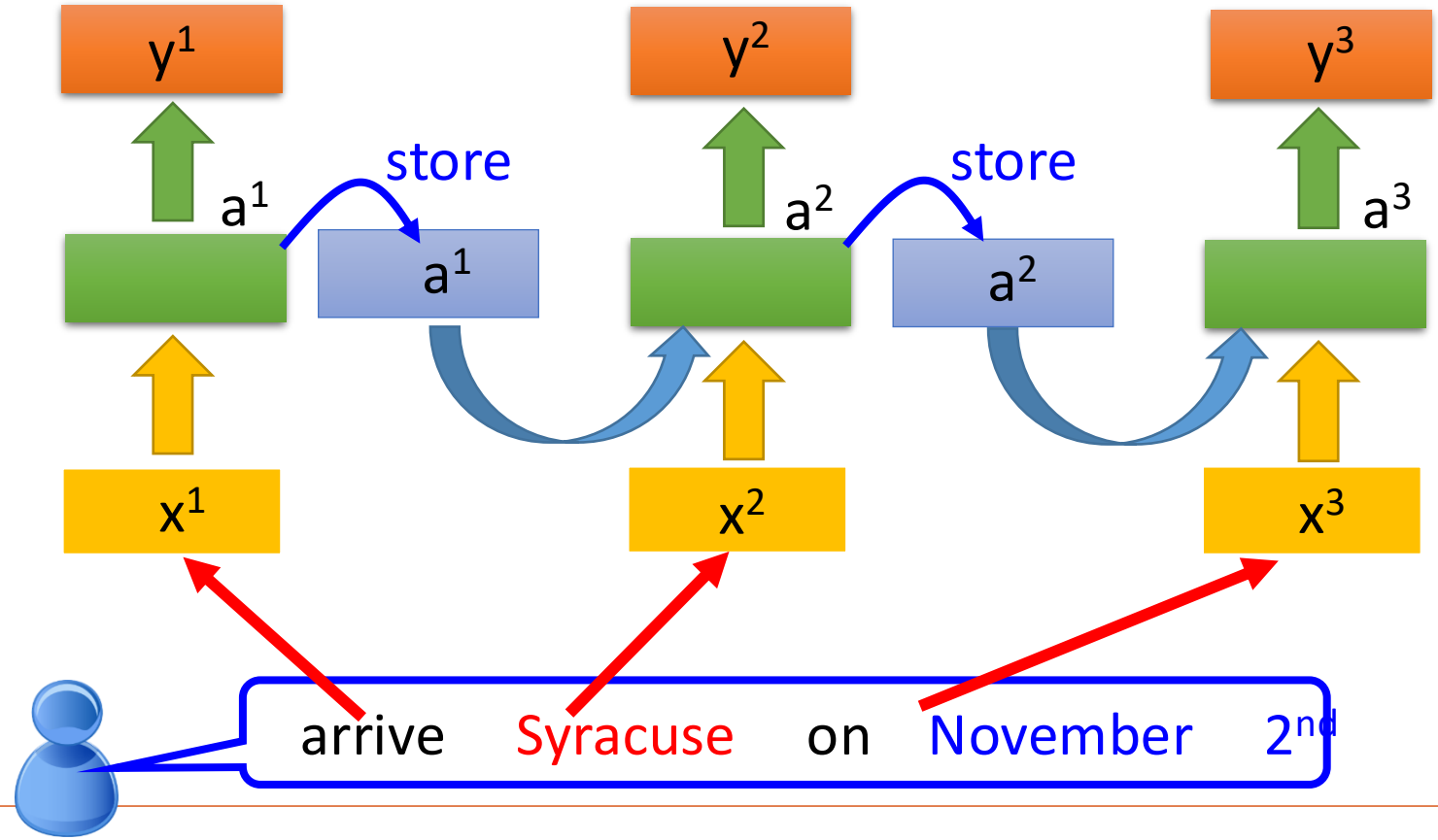
RNN

The same network is used again and again.

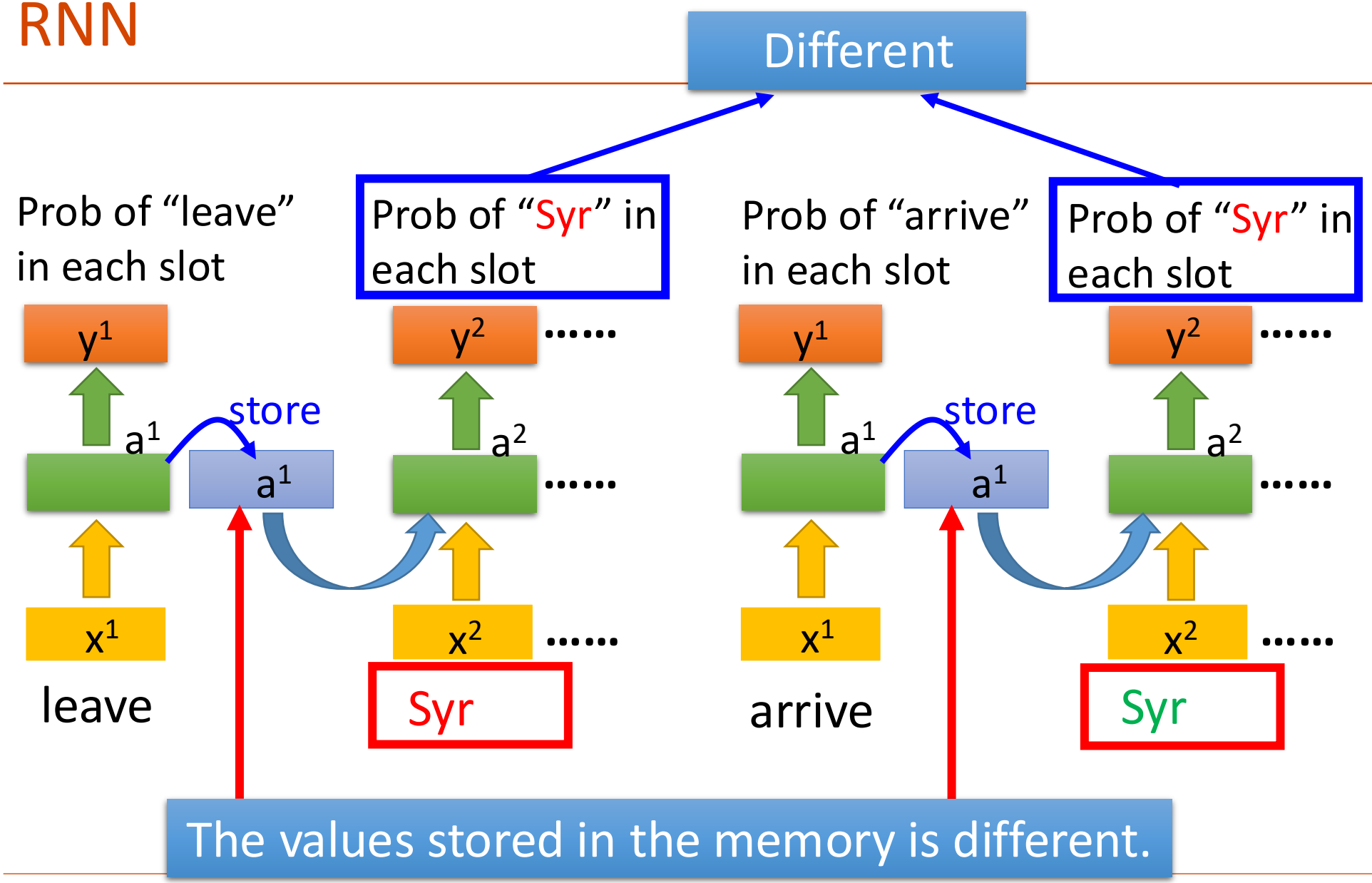
Probability of "arrive" in each slot

Probability of "Syr" in each slot

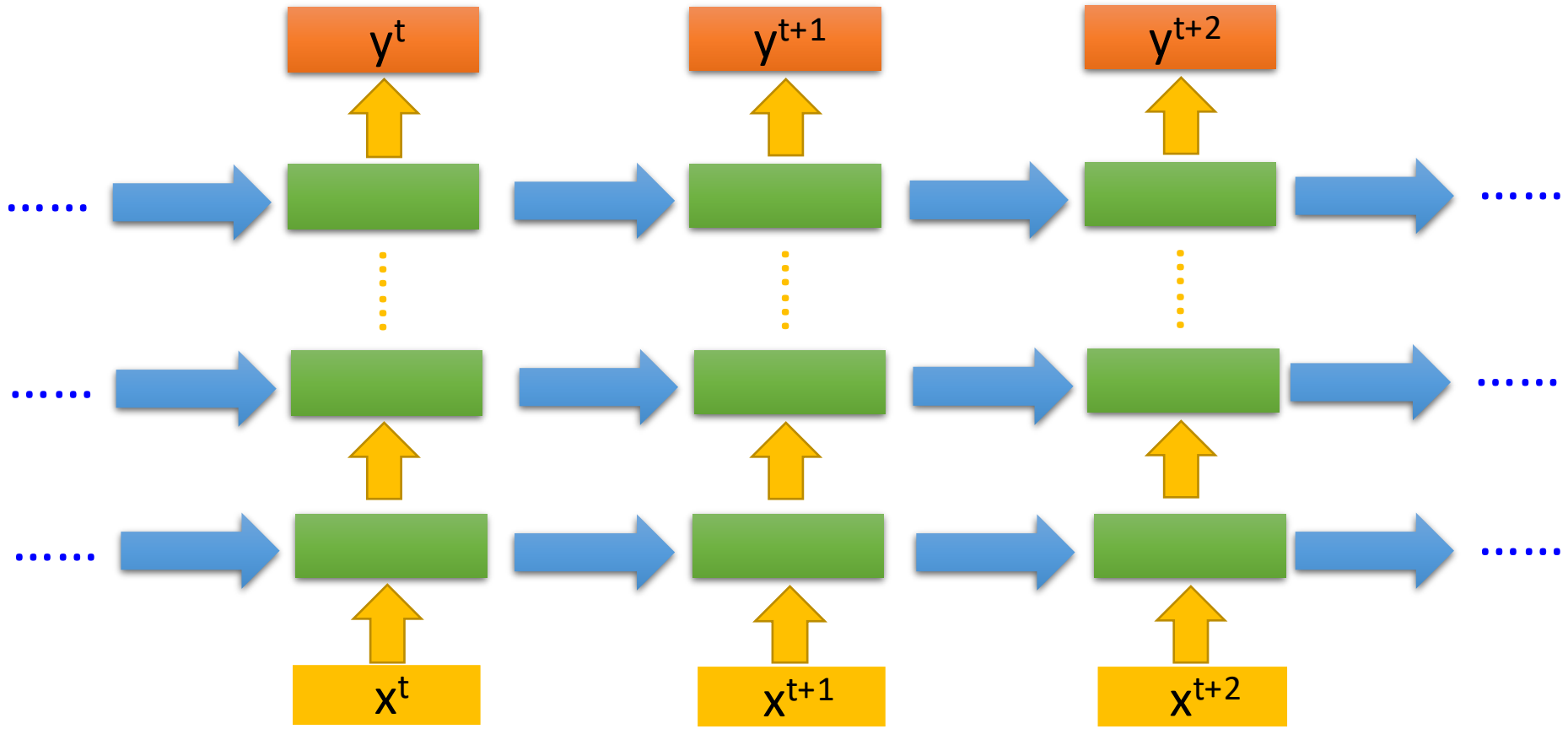
Probability of "on" in each slot



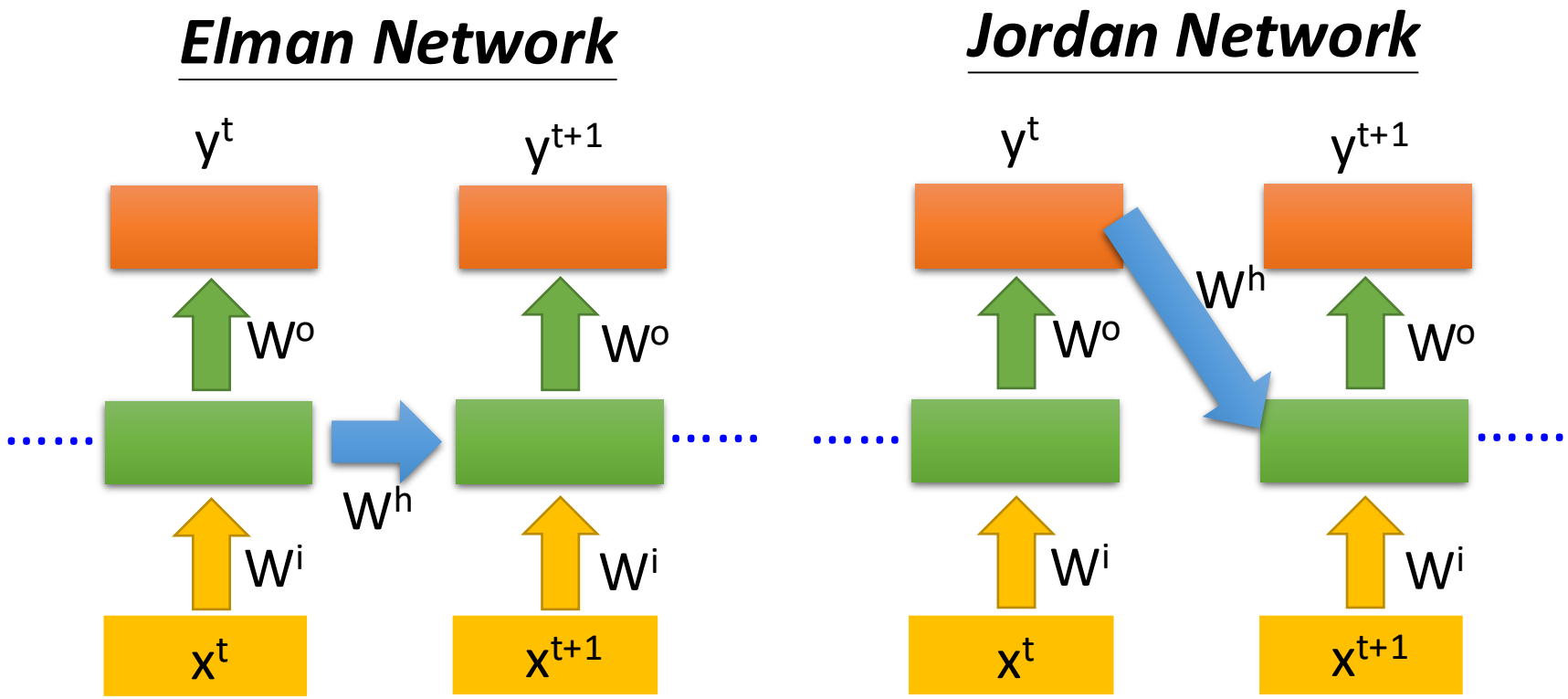
RNN


Different
Prob of "leave" in each slot
Prob of "Syr" in each slot
Prob of "arrive" in each slot
Prob of "Syr" in each slot
store
store
leave
Syr
arrive
Syr
The values stored in the memory is different.

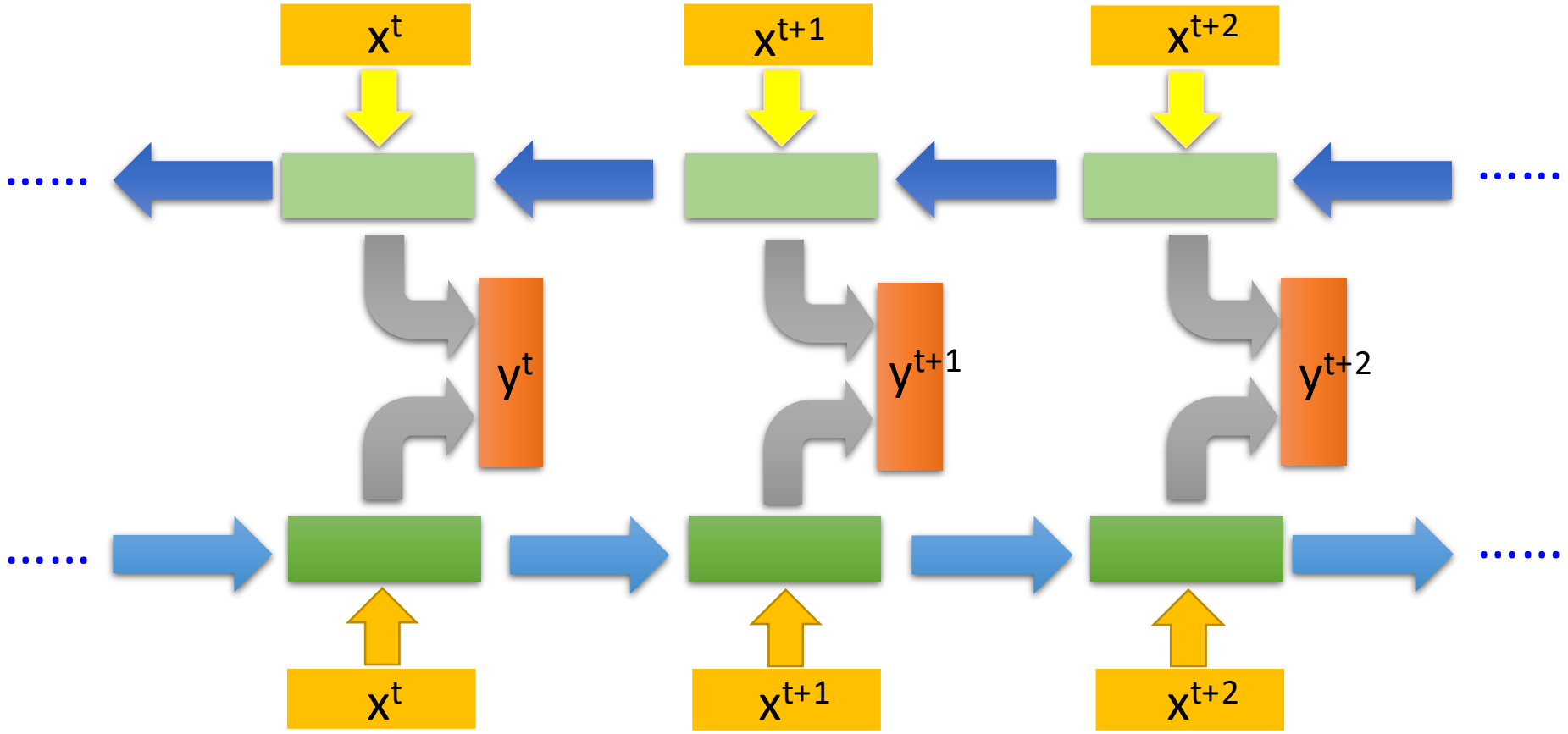
Of course it can be deep …



Elman Network & Jordan Network

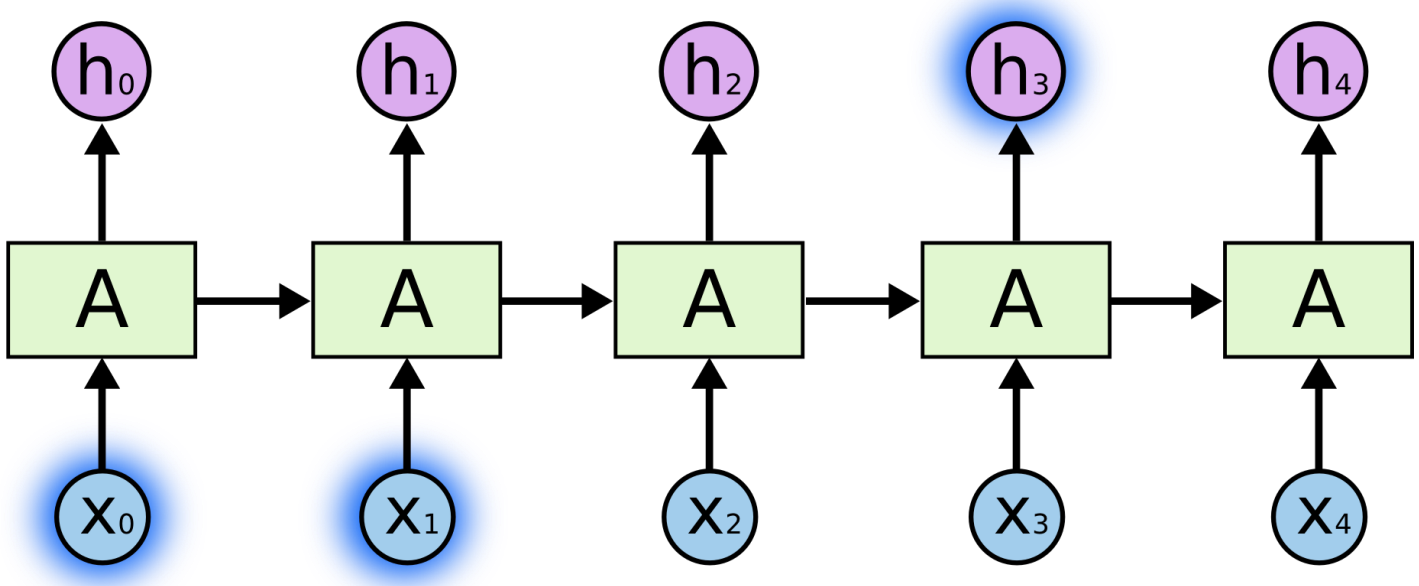


Bidirectional RNN



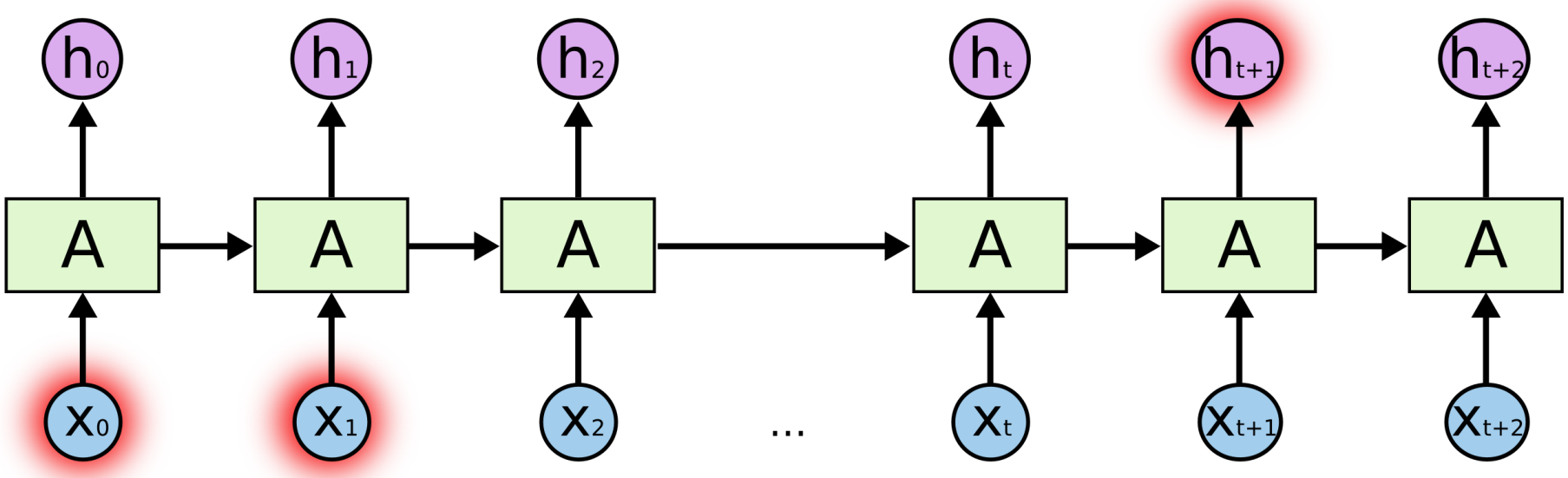
From RNN to LSTM

The clouds are in the __



From RNN to LSTM

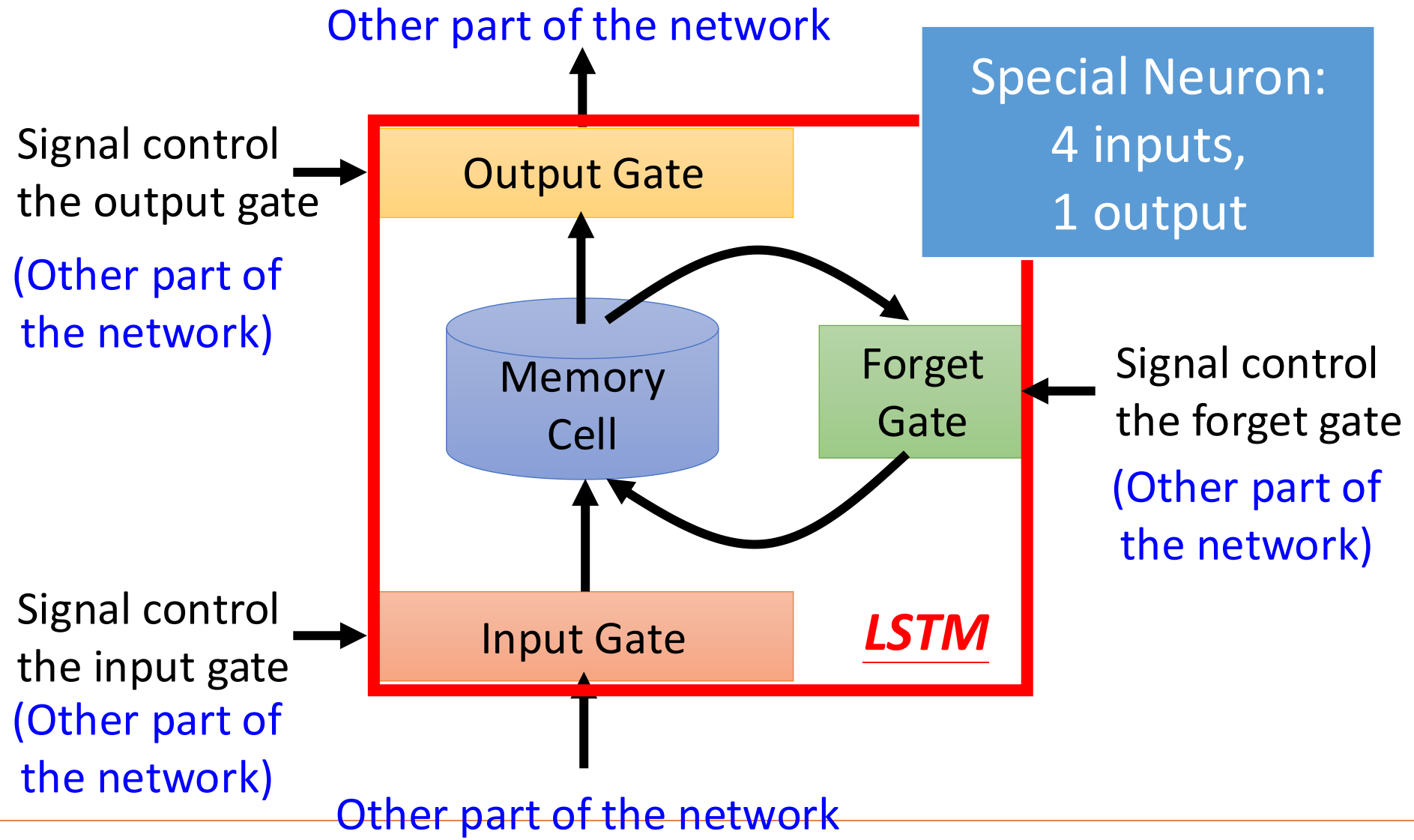
"I grew up in France… I speak fluent *French*."

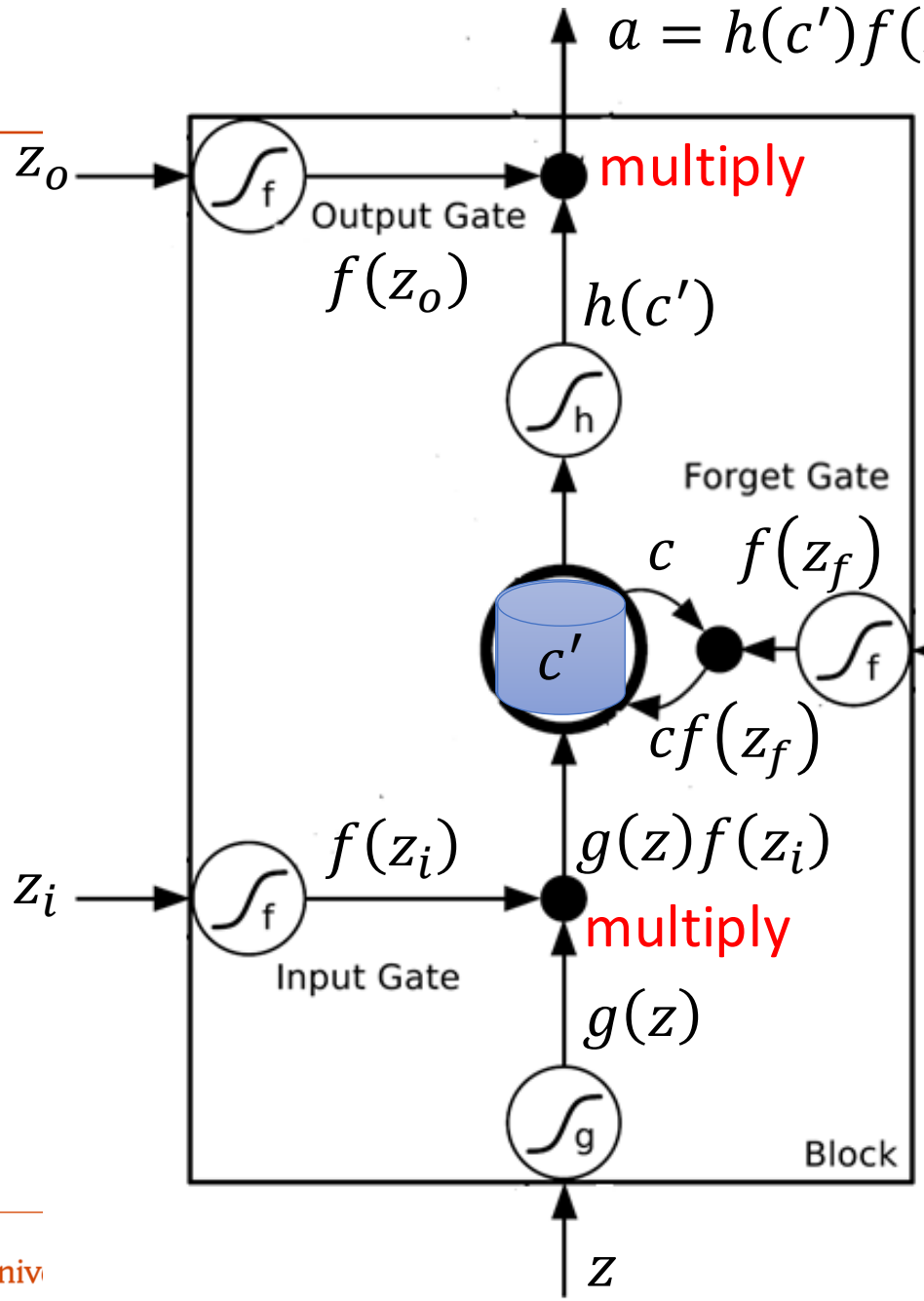


LSTM

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies.

Long Short-term Memory (LSTM)





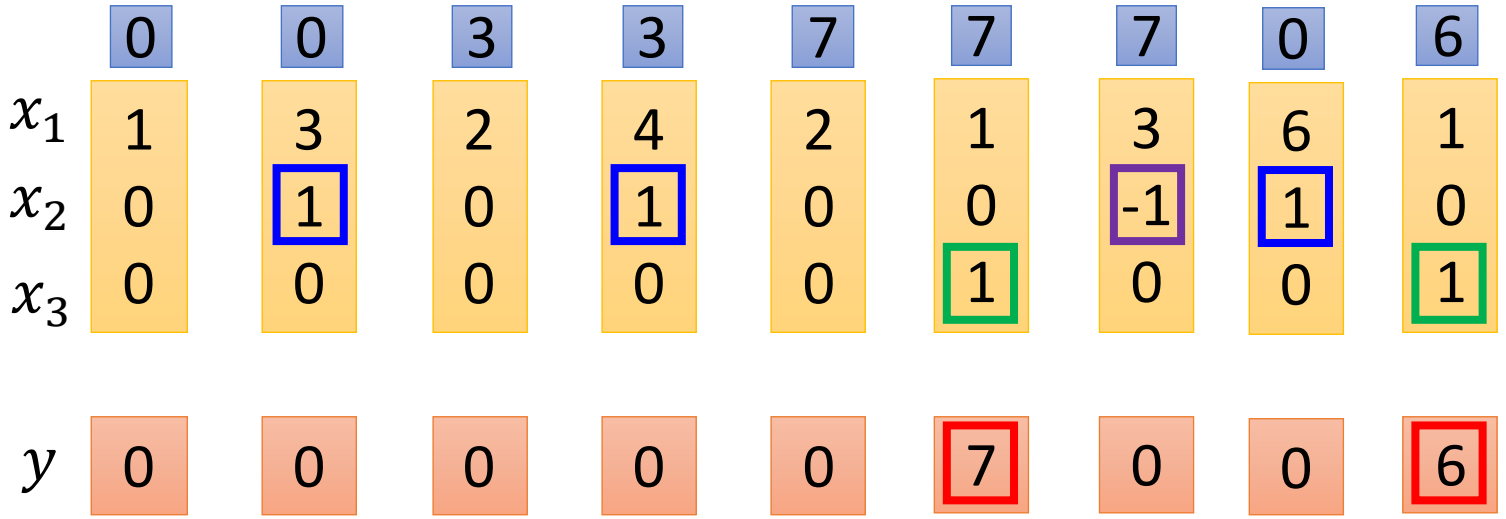
Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

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

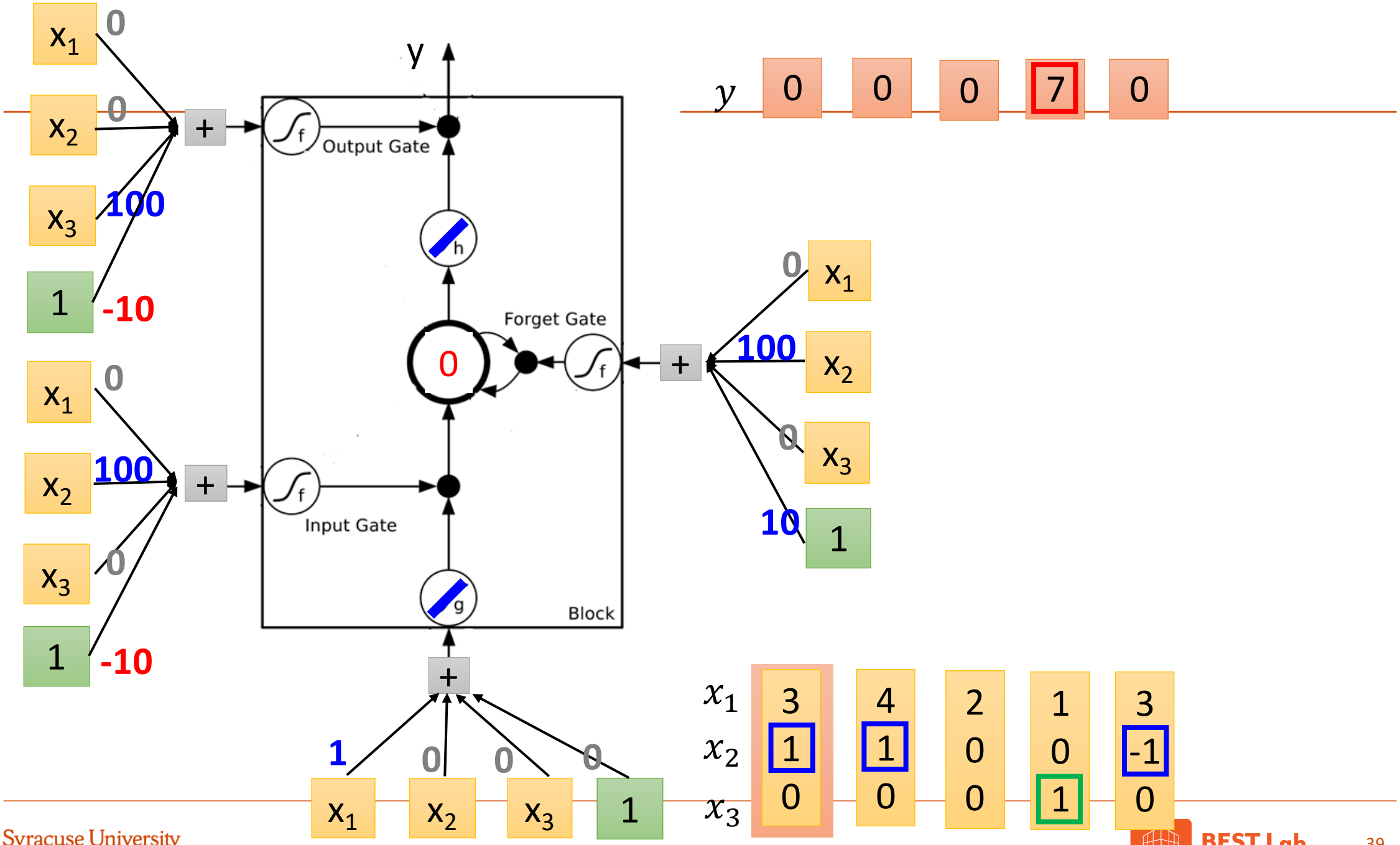
LSTM - Example

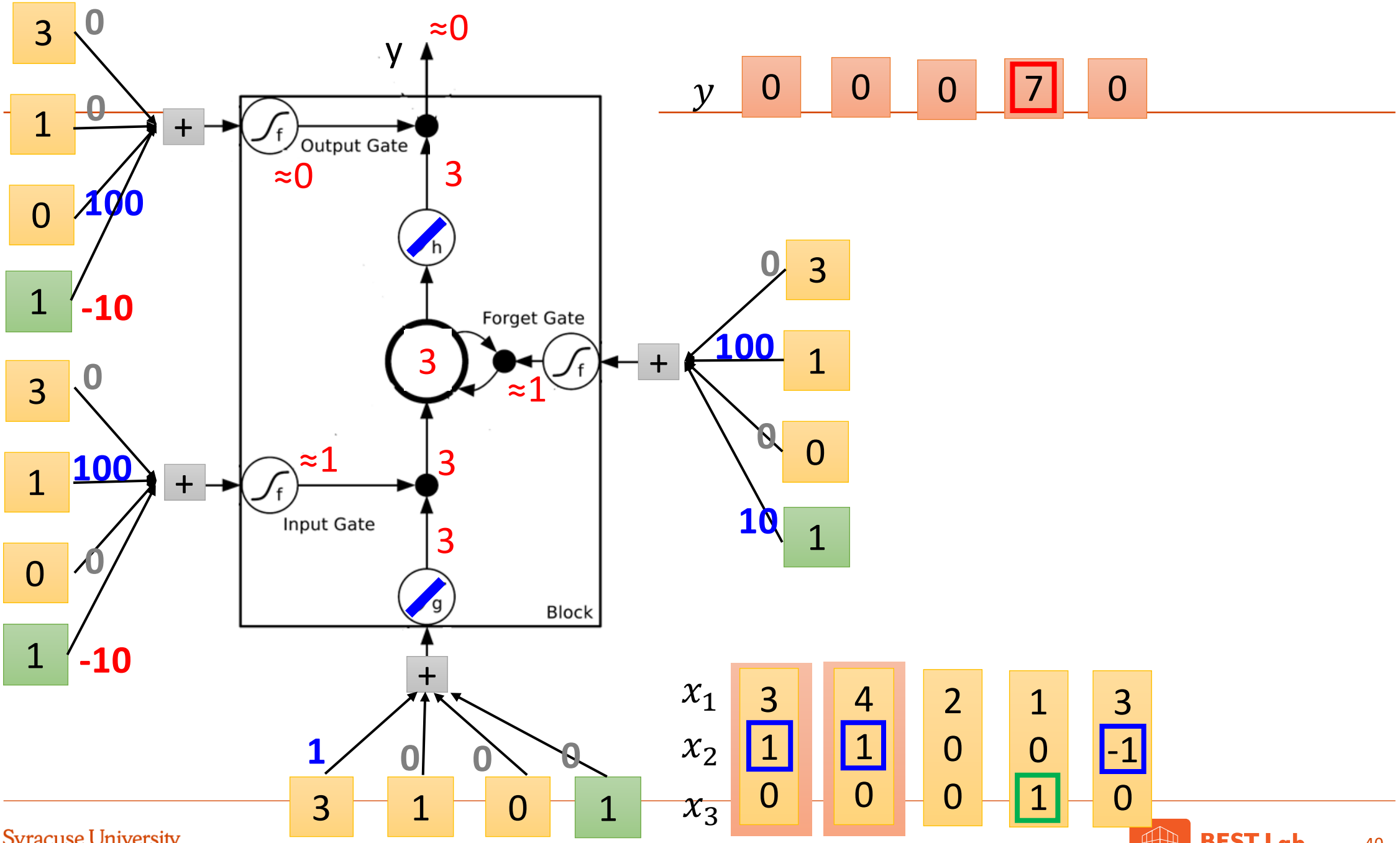


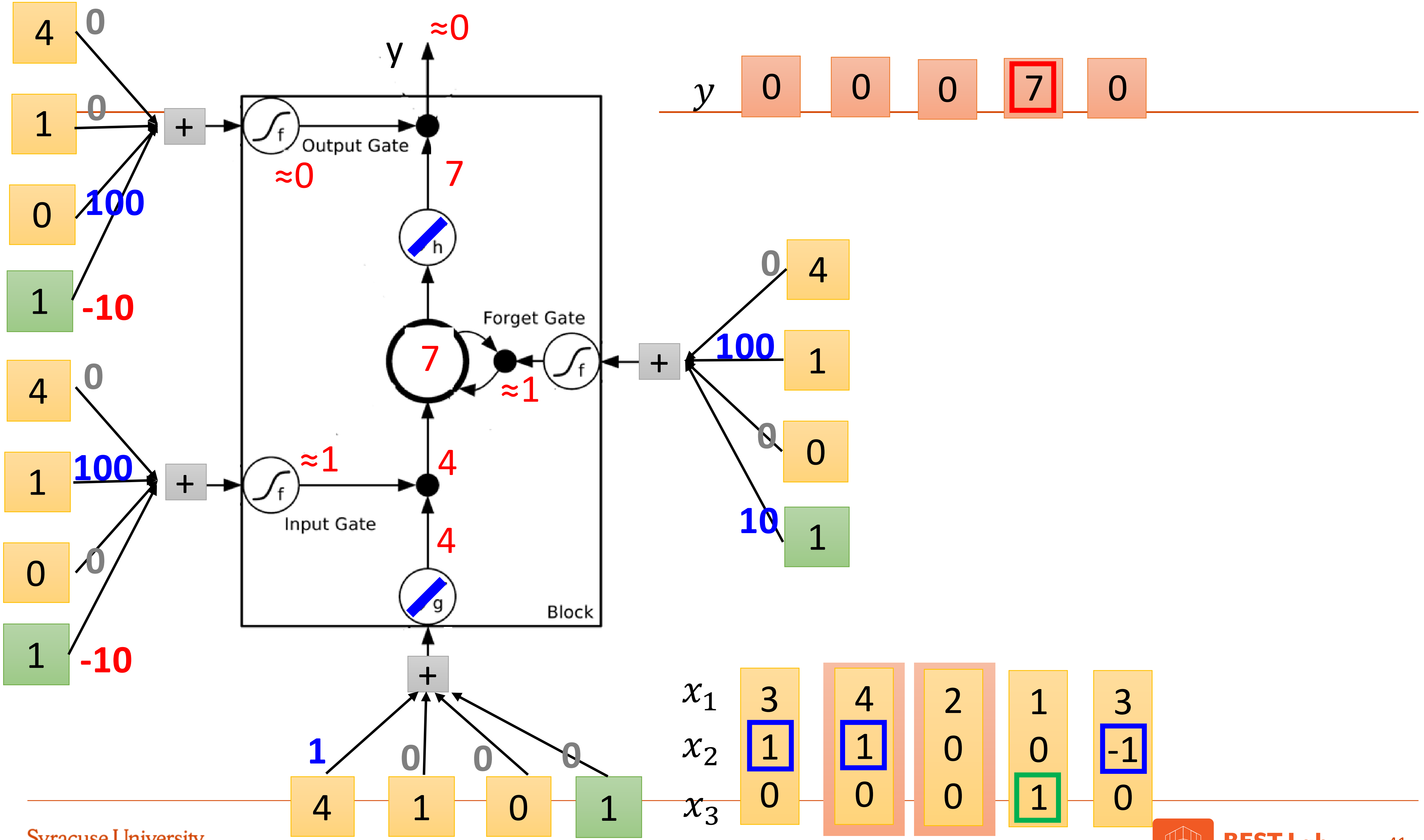
When x_2 = 1, add the numbers of x_1 into the memory

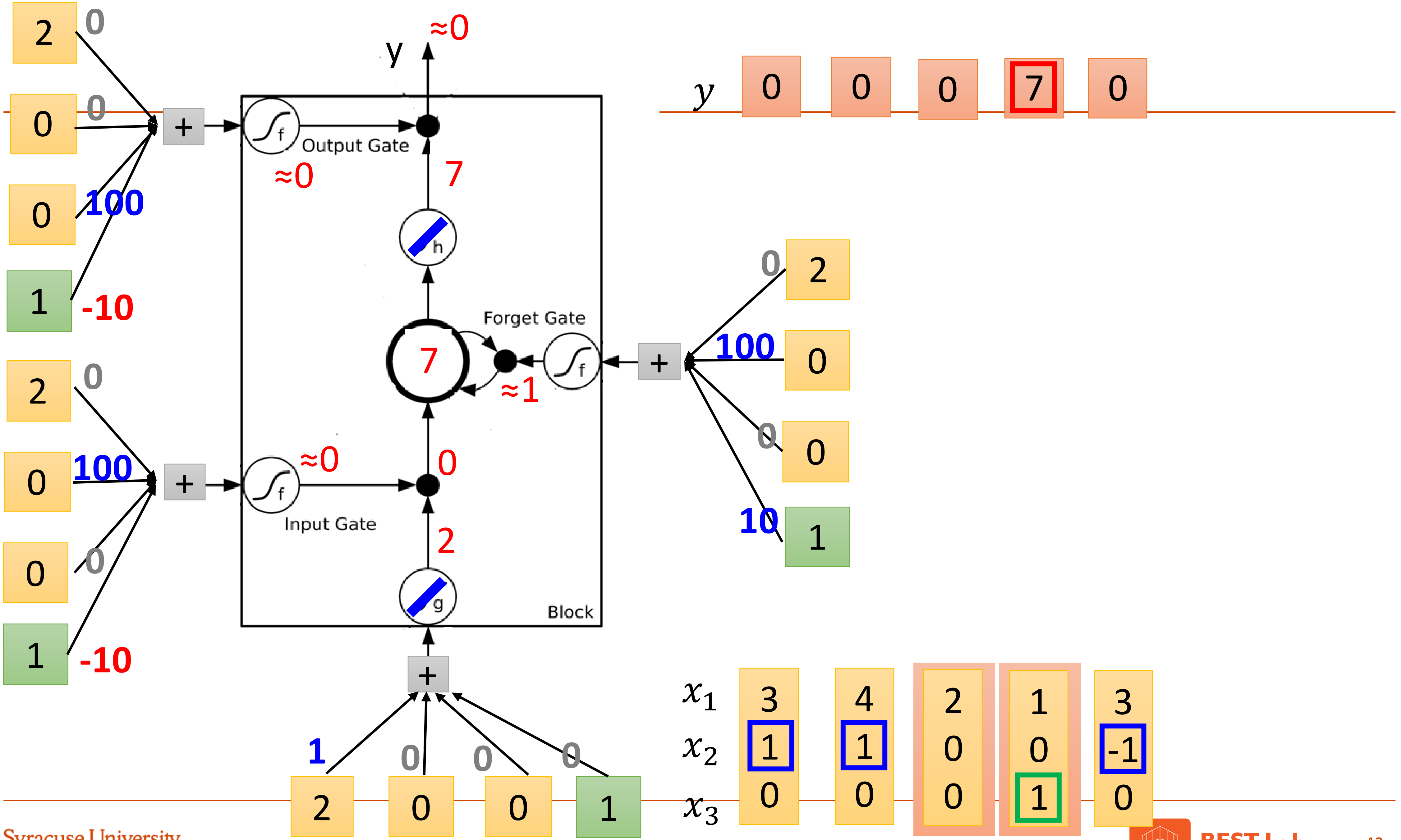
When x_2 = -1, reset the memory

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

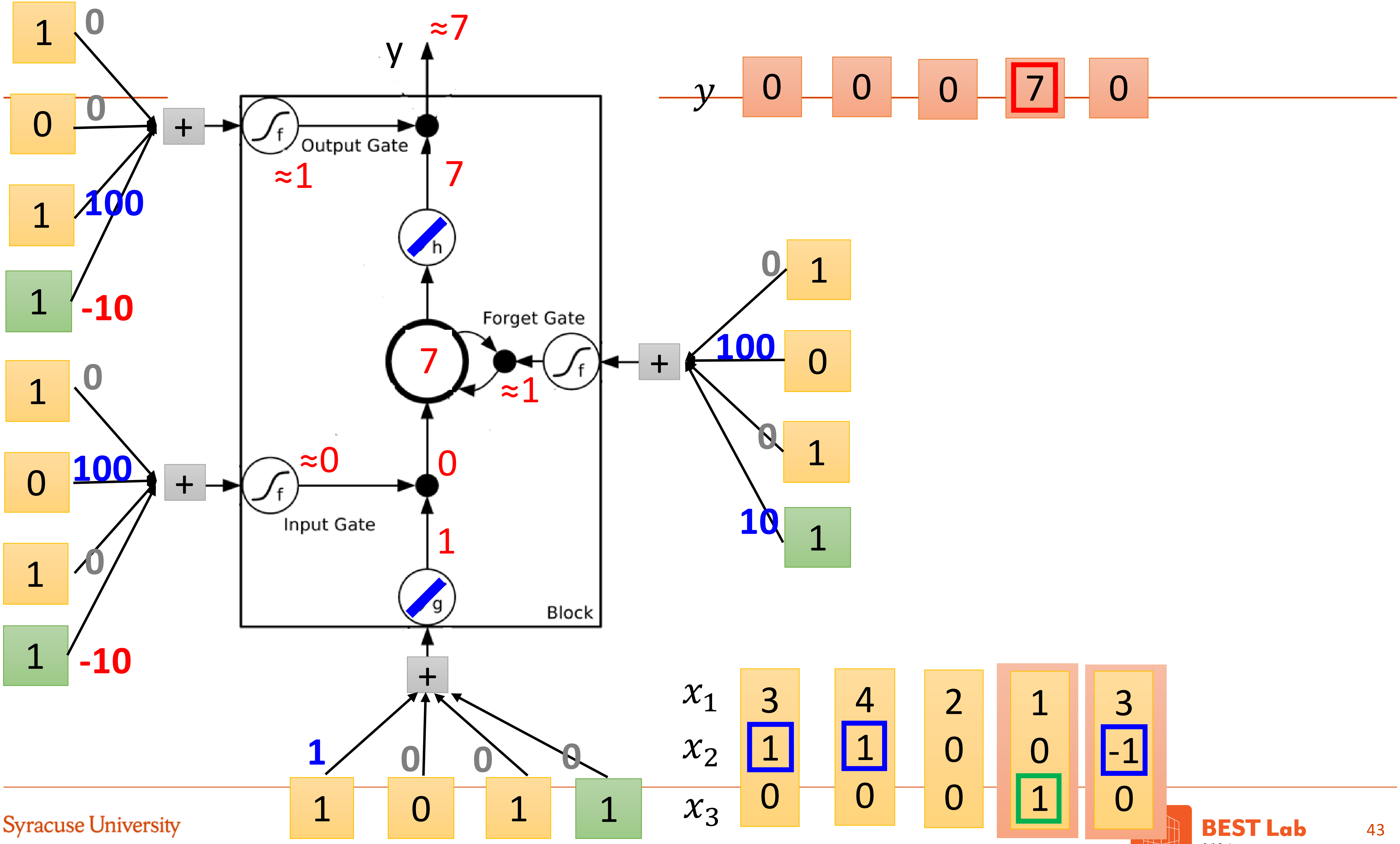

y
Output Gate
Forget Gate
Input Gate
Block
f
h
g
+
0
100
-10
1
x_1
x_2
x_3
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0


y
≈0
Output Gate
≈0
3
Forget Gate
3
≈1
Input Gate
≈1
3
3
Block
3
0
1
100
0
1
-10
3
0
1
100
0
0
1
-10
0
3
1
100
0
0
10
1
1
0
0
0
3
1
0
1
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

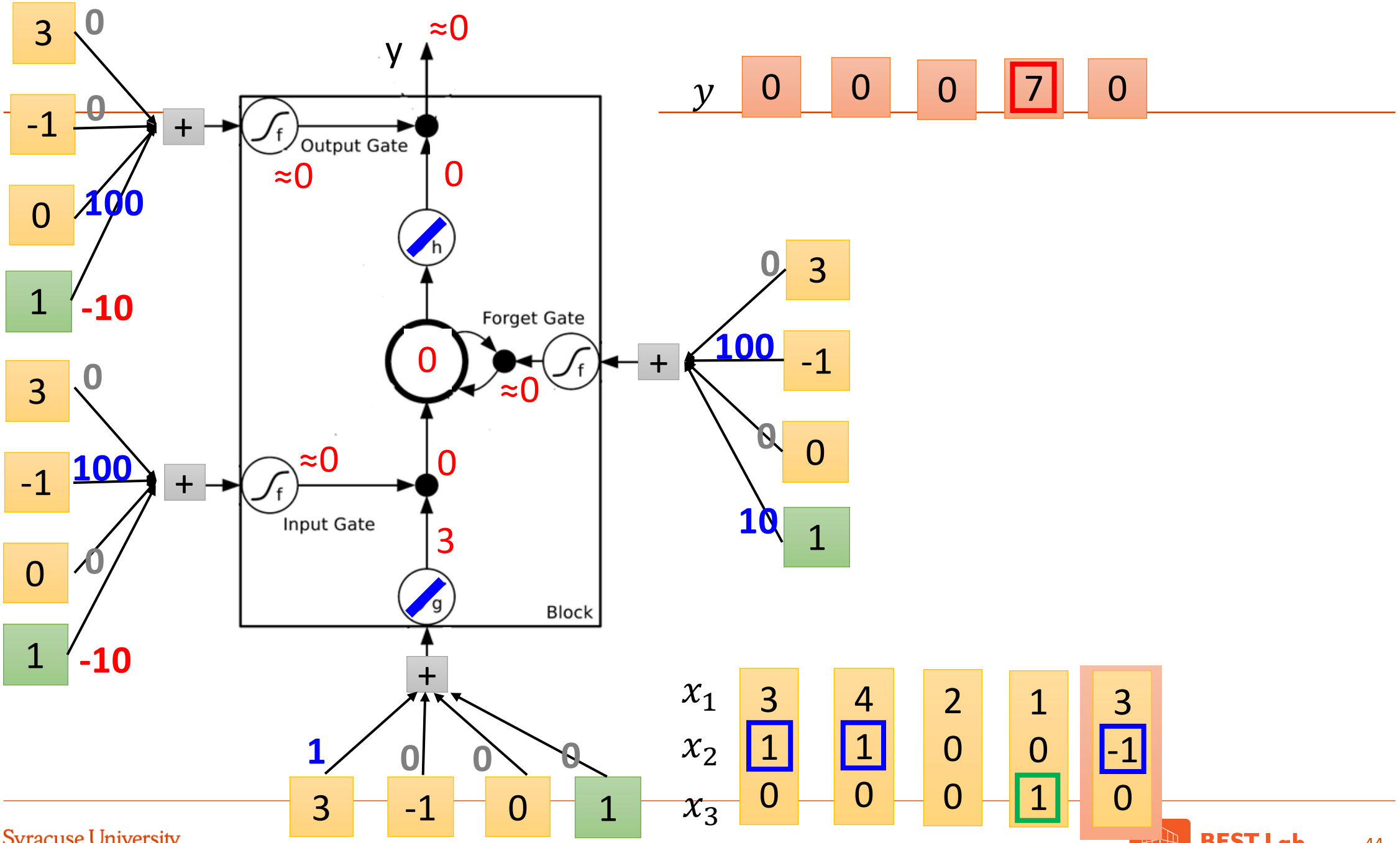

y
≈0
Output Gate
≈0
7
h
7
Forget Gate
≈1
≈1
Input Gate
4
4
g
Block
4 0
1 0
0 100
1 -10
4 0
1 100
0 0
1 -10
0 4
100 1
0 0
10 1
1 0 0 0
4 1 0 1
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0



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

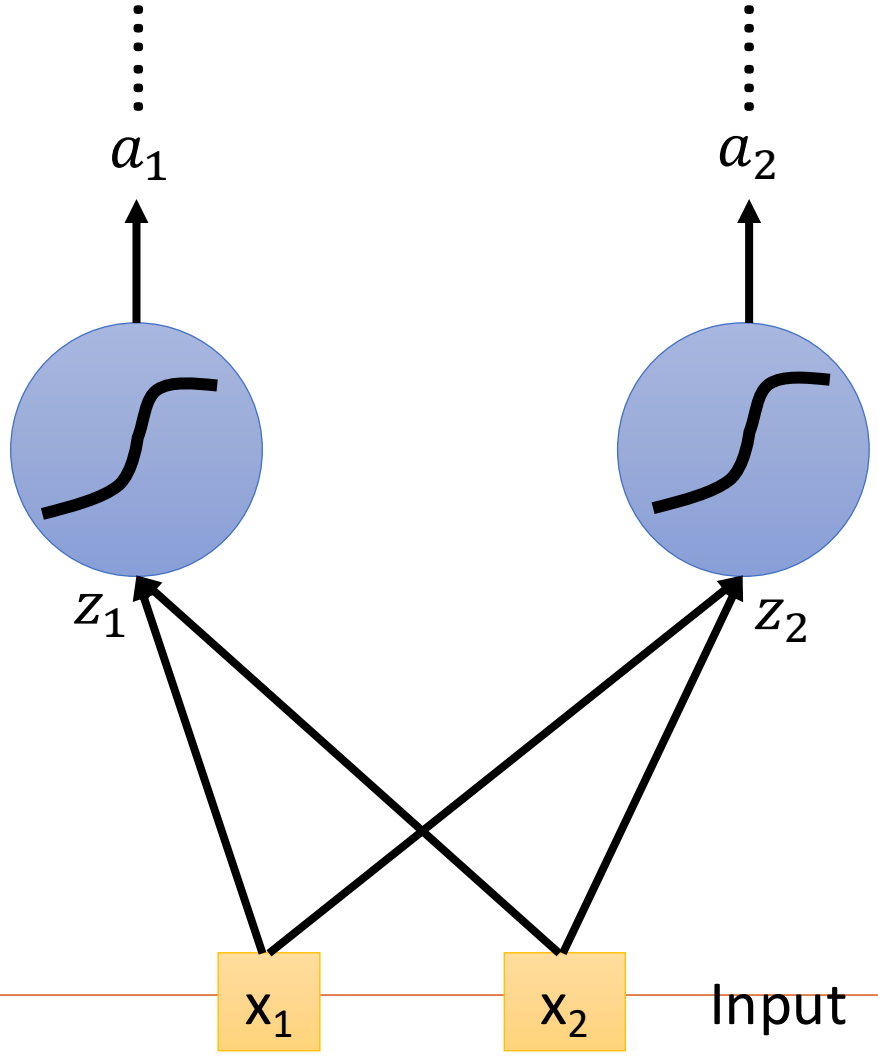


y
≈7
Output Gate
≈1
7
Forget Gate
≈1
≈0
0
1
Input Gate
Block
100
-10
10
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0

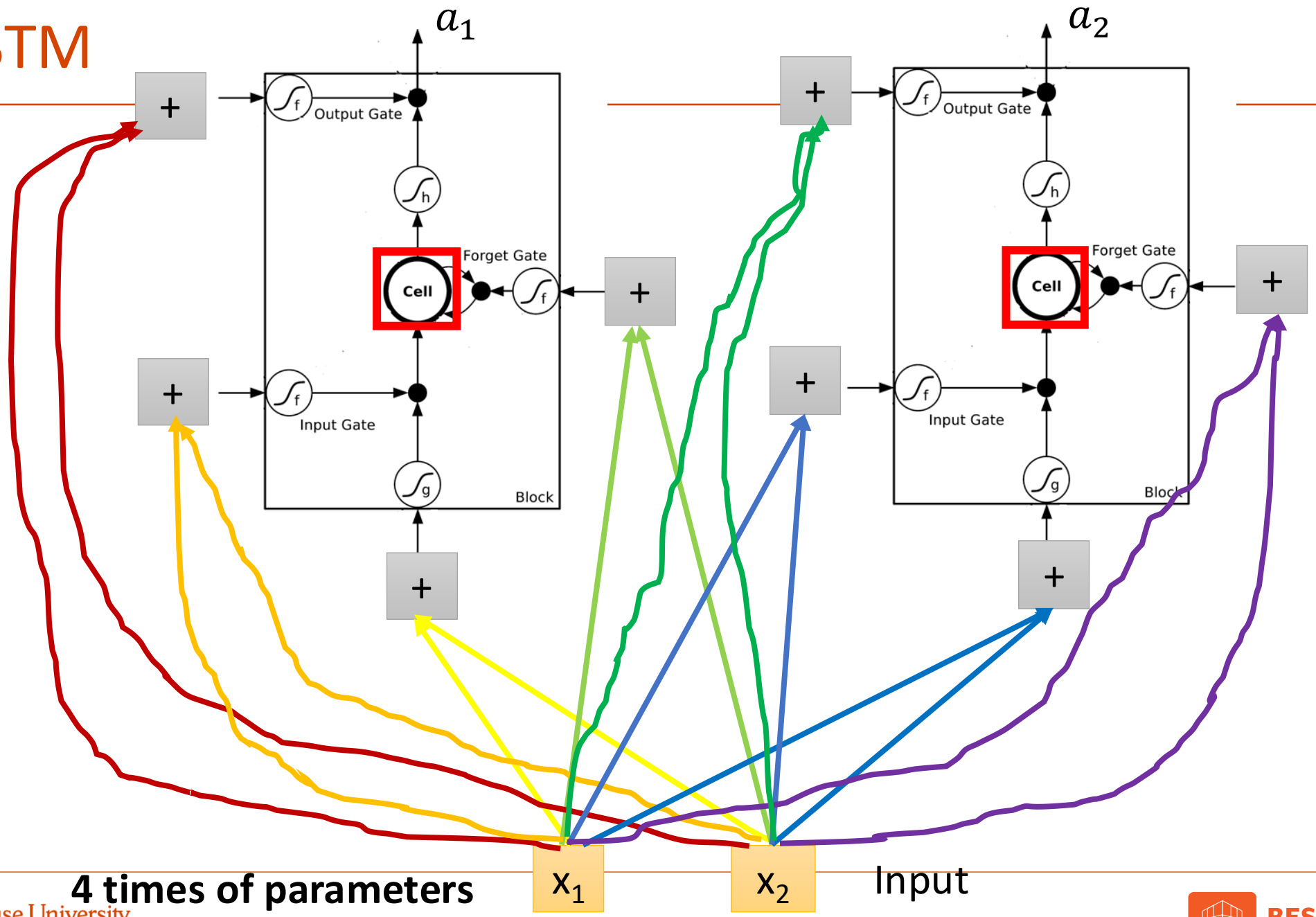

y
≈0
Output Gate
≈0
0
Forget Gate
0
≈0
Input Gate
≈0
0
3
Block
+
3
0
-1
0
100
1
-10
3
0
-1
100
0
0
1
-10
0
3
100
-1
0
0
10
1
1
0
0
0
3
-1
0
1
y 0 0 0 7 0
x_1 3 4 2 1 3
x_2 1 1 0 0 -1
x_3 0 0 0 1 0

Original Network

- Simply replace the neurons with LSTM

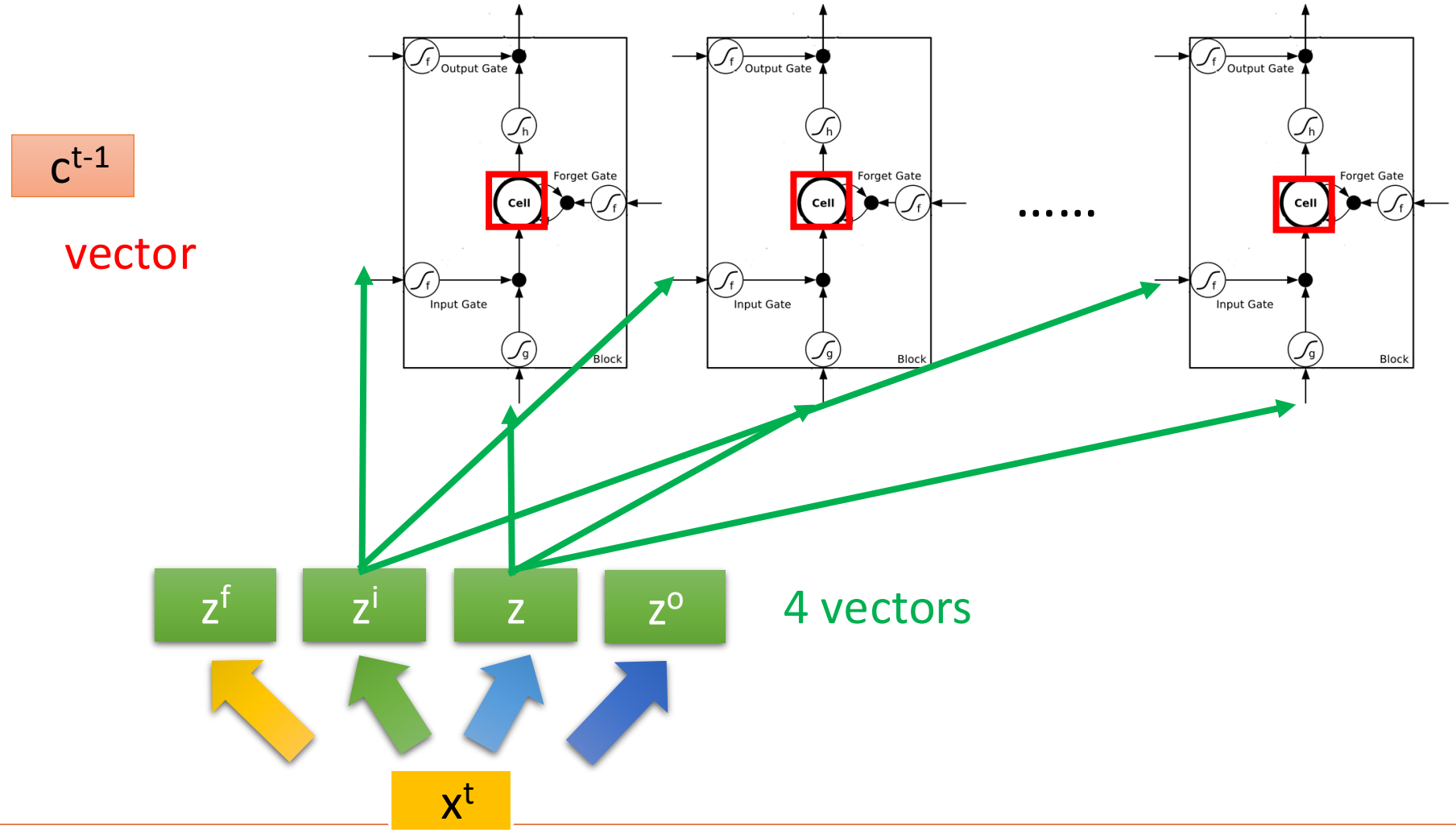


LSTM

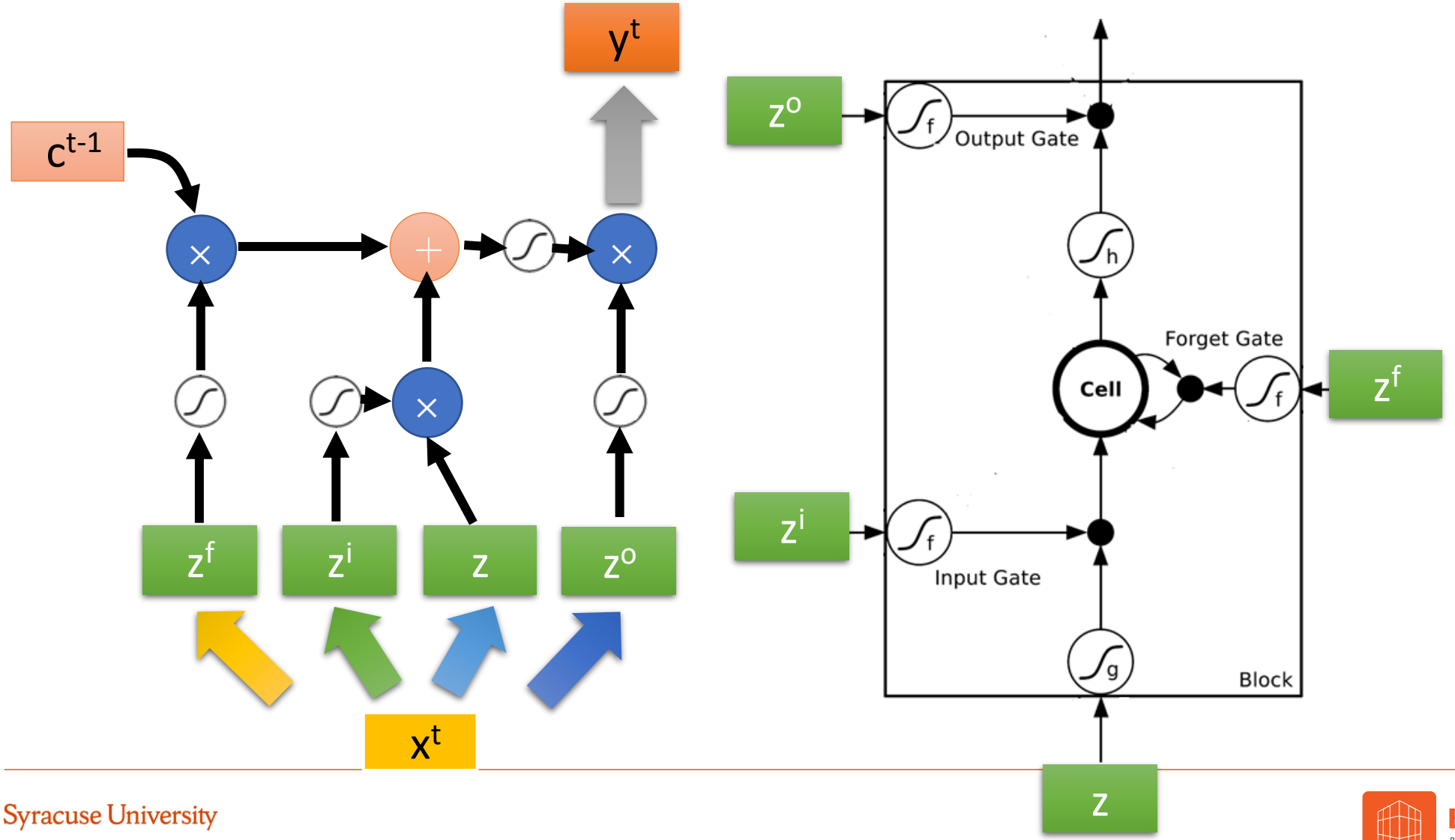


4 times of parameters

LSTM

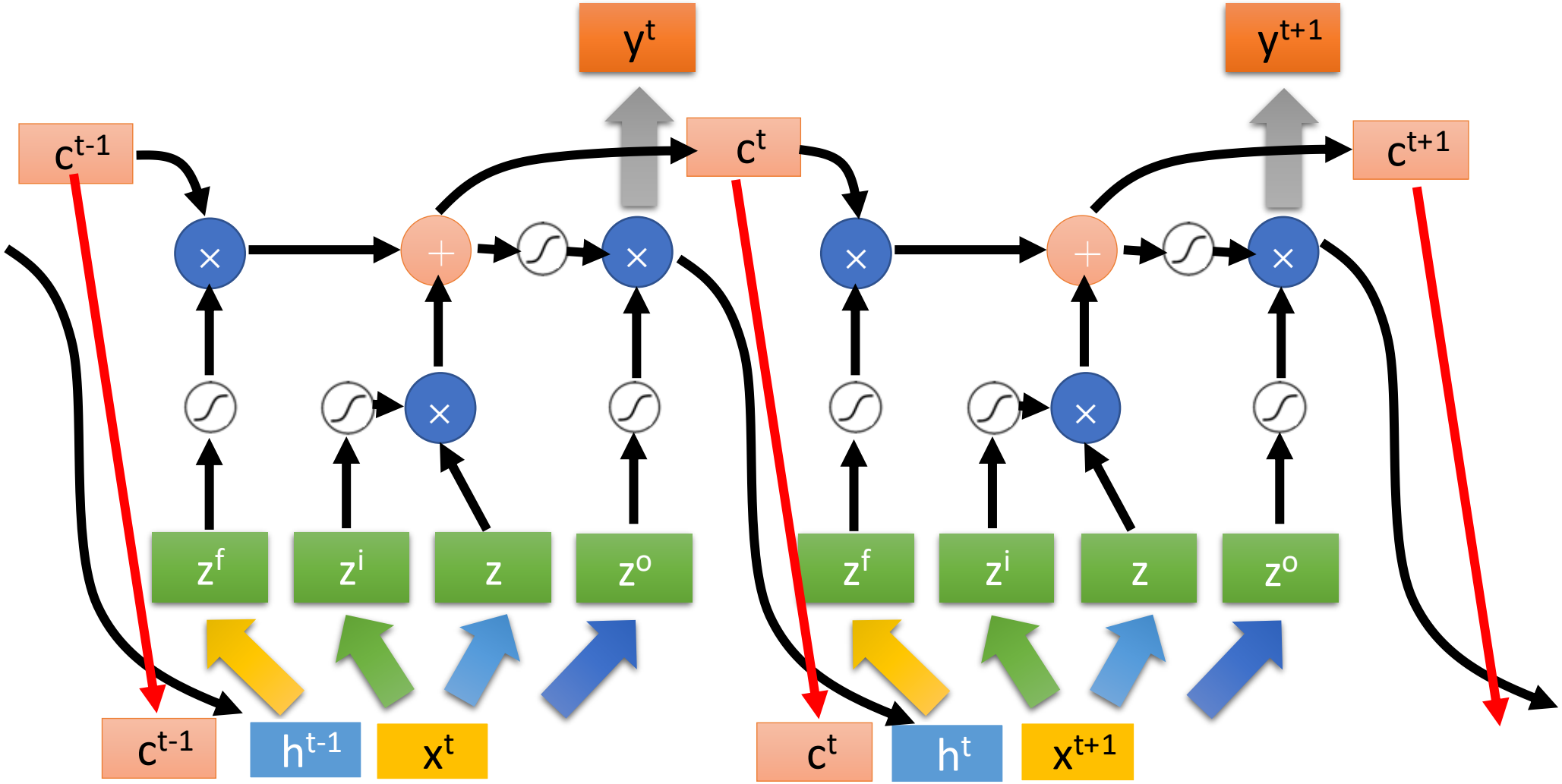

c^{t-1}
vector
Output Gate
Forget Gate
Cell
Input Gate
Block
.....
z^f
z^i
z
z^o
4 vectors
x^t

LSTM

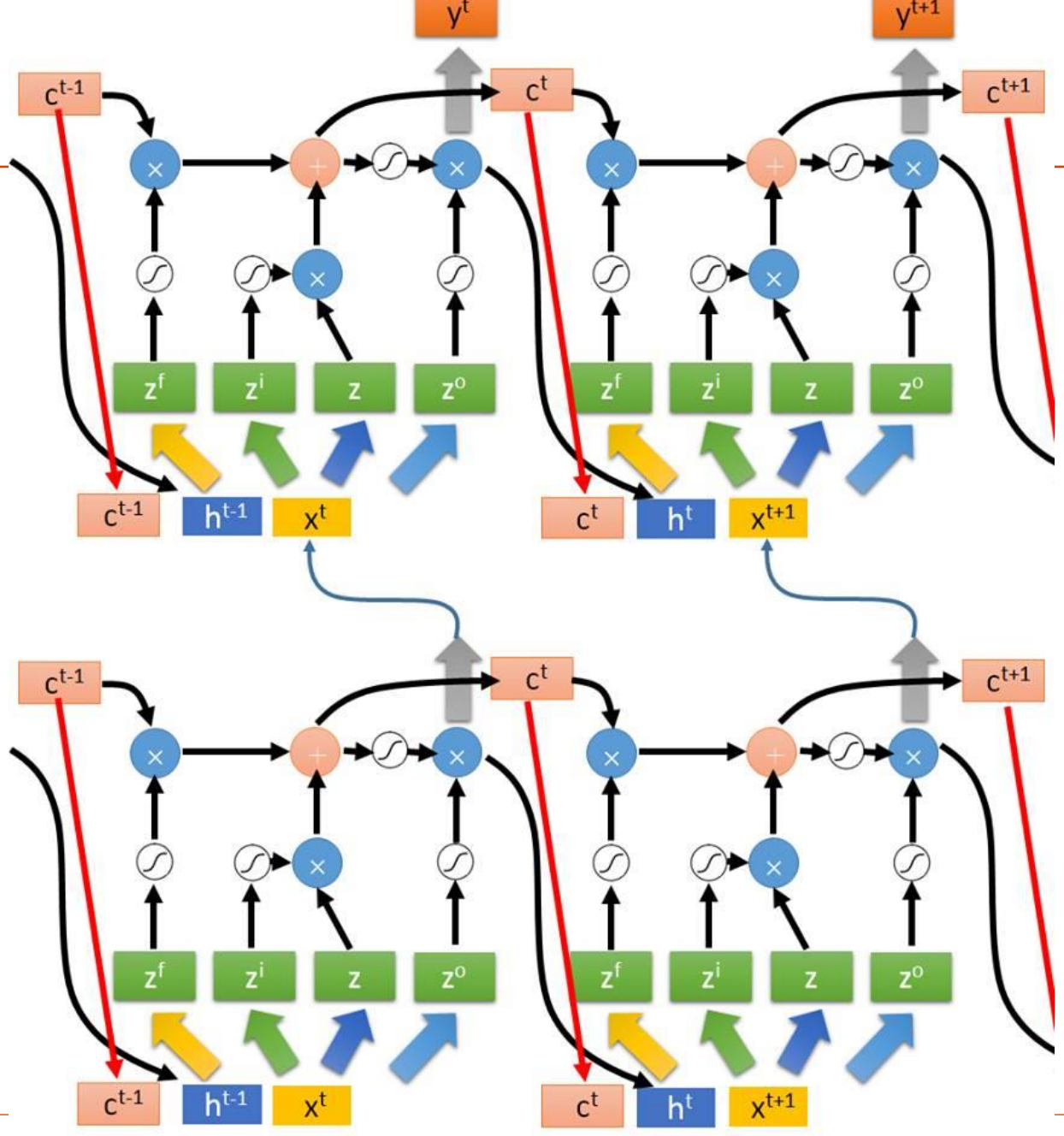

y^t
c^{t-1}
z^f
z^i
z
z^o
x^t
z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

LSTM

Extension: “peephole”

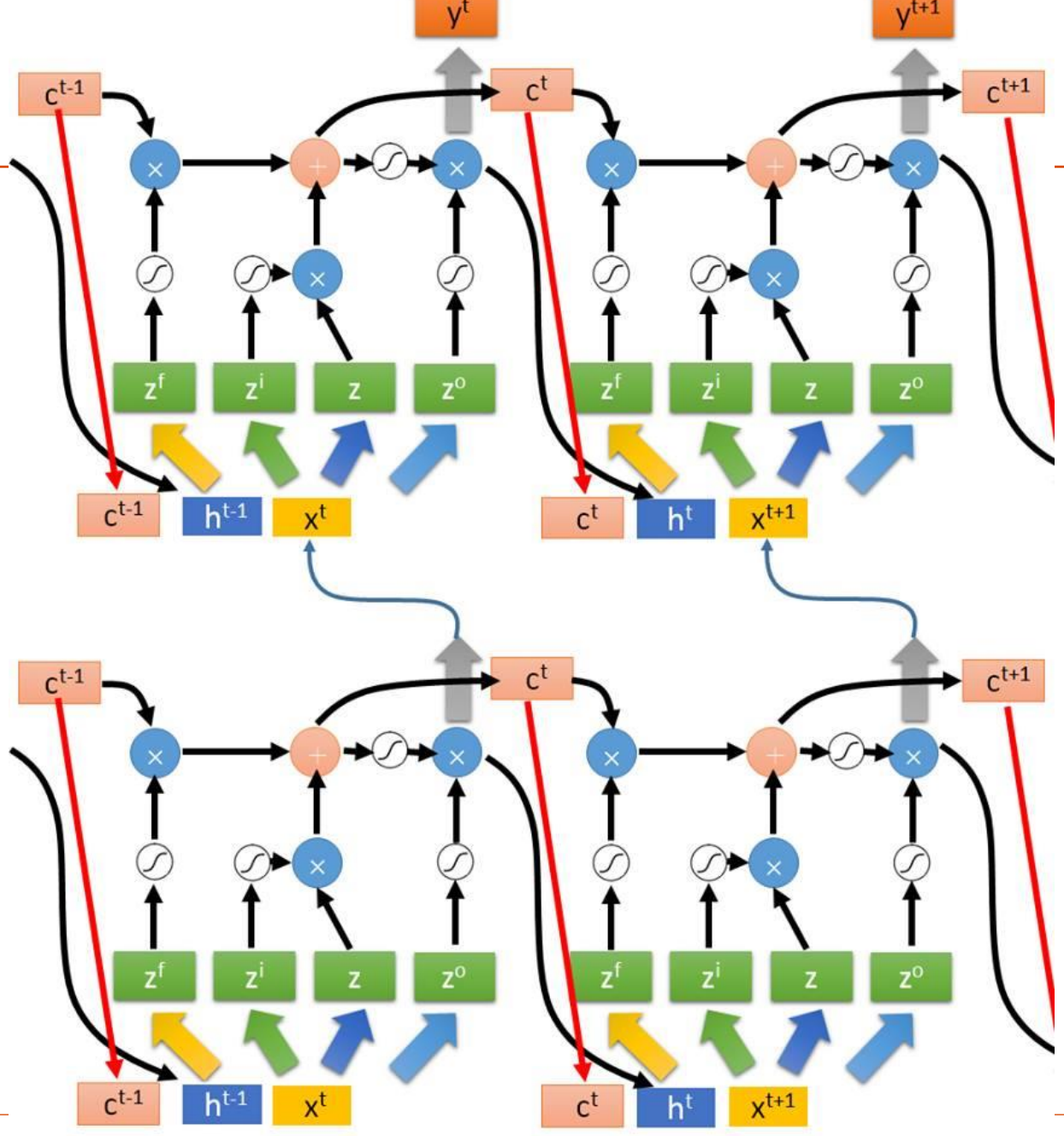


Multiple-layer LSTM



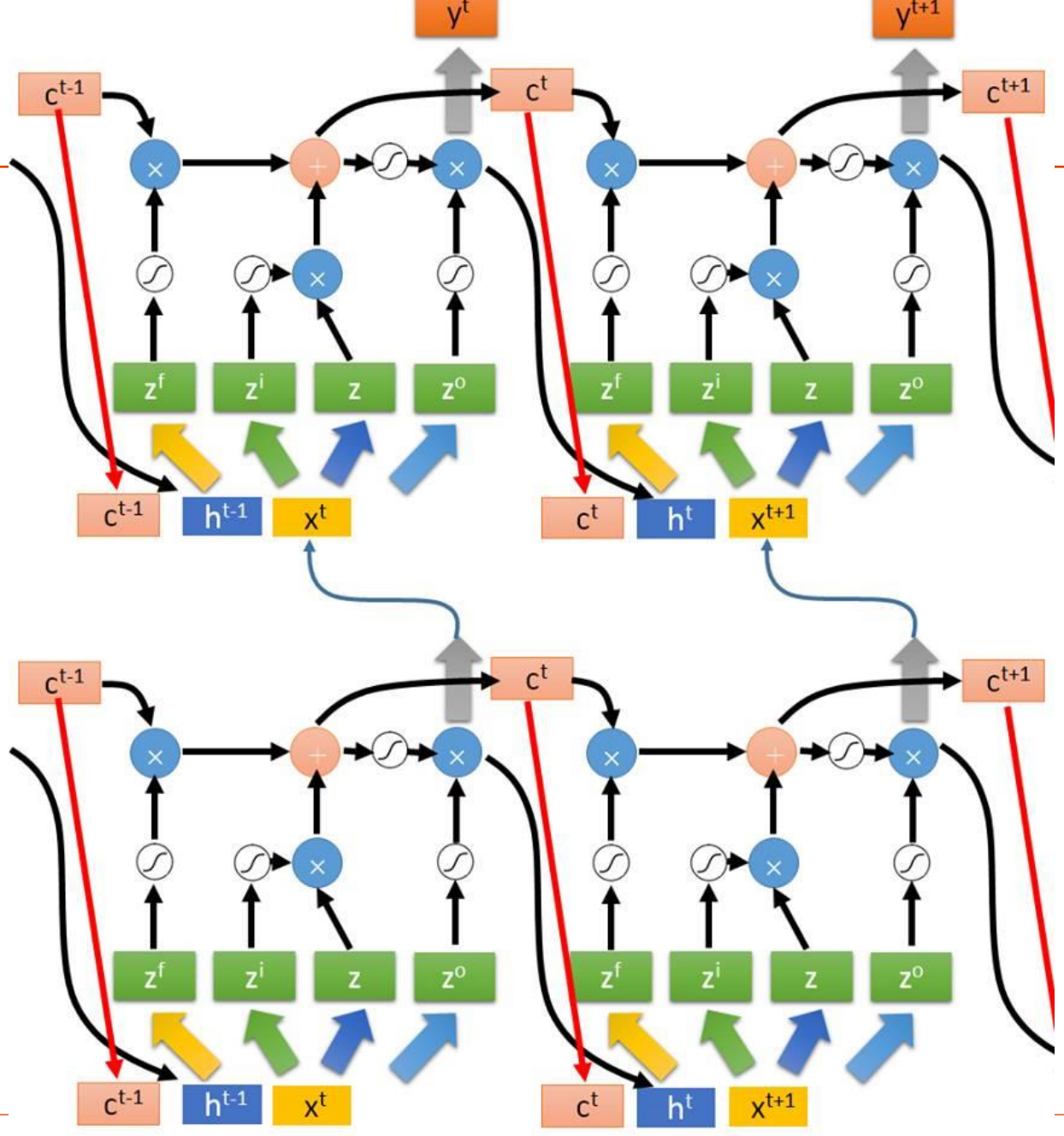
https://img.komicolle.org/2015-09-20/src/14426967627131.gif

Multiple-layer LSTM



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

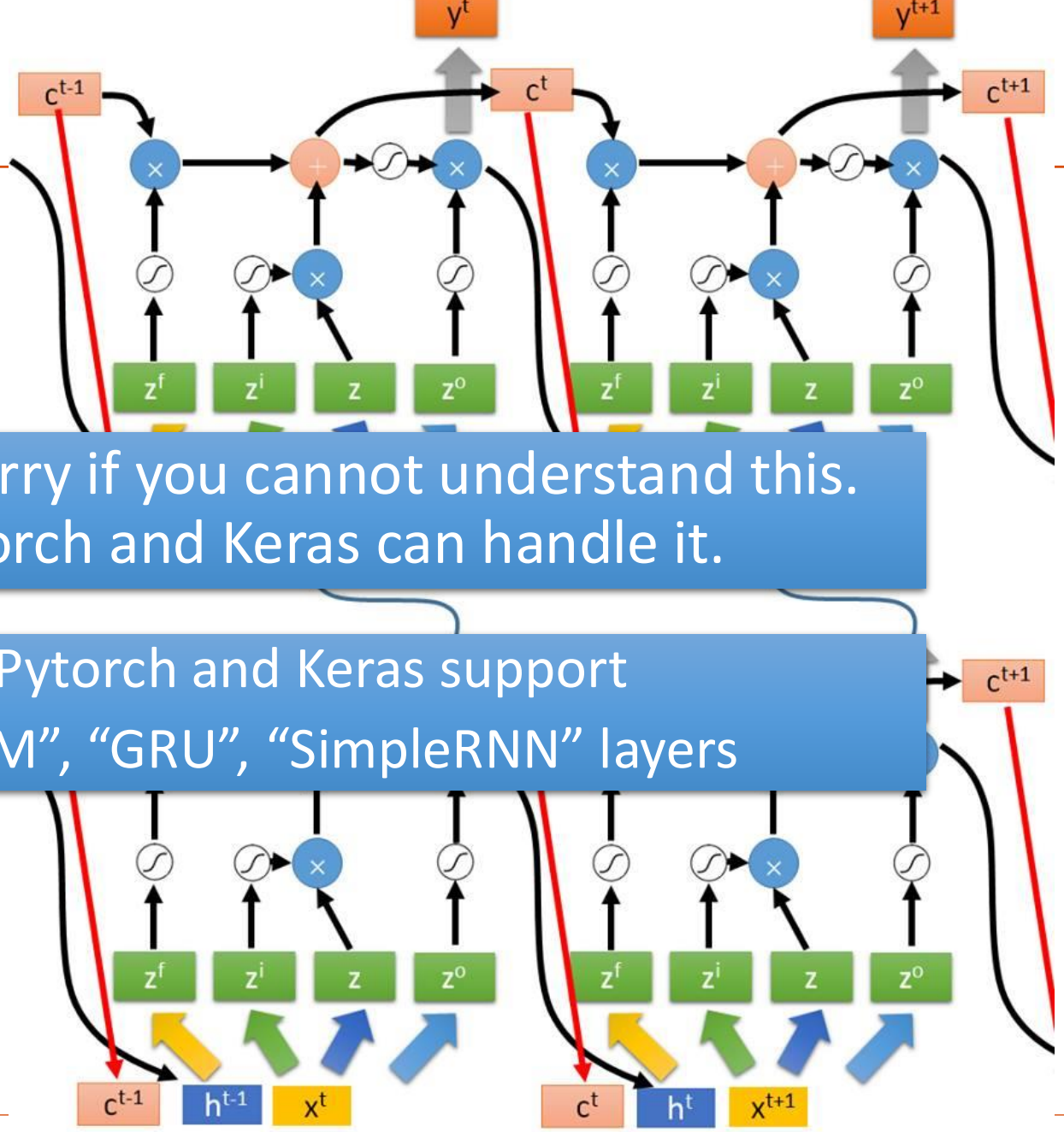
Multiple-layer LSTM



This is quite standard now.

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

Multiple-layer LSTM



Don't worry if you cannot understand this.
Pytorch and Keras can handle it.

Pytorch and Keras support
"LSTM", "GRU", "SimpleRNN" layers

This is quite standard now.

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

Any Questions?

bidong@syr.edu