Variants of Neural Networks

Slides are provided by Prof. Hung-yi Lee

Why Deep Learning?

- Previous systems required feature engineering for every new problem.
- Deep learning enables end-to-end learning for a given loss and architecture.
- Weak prior knowledge can be encoded via architecture (e.g., convolutions, recurrent)

Variants of Neural Networks

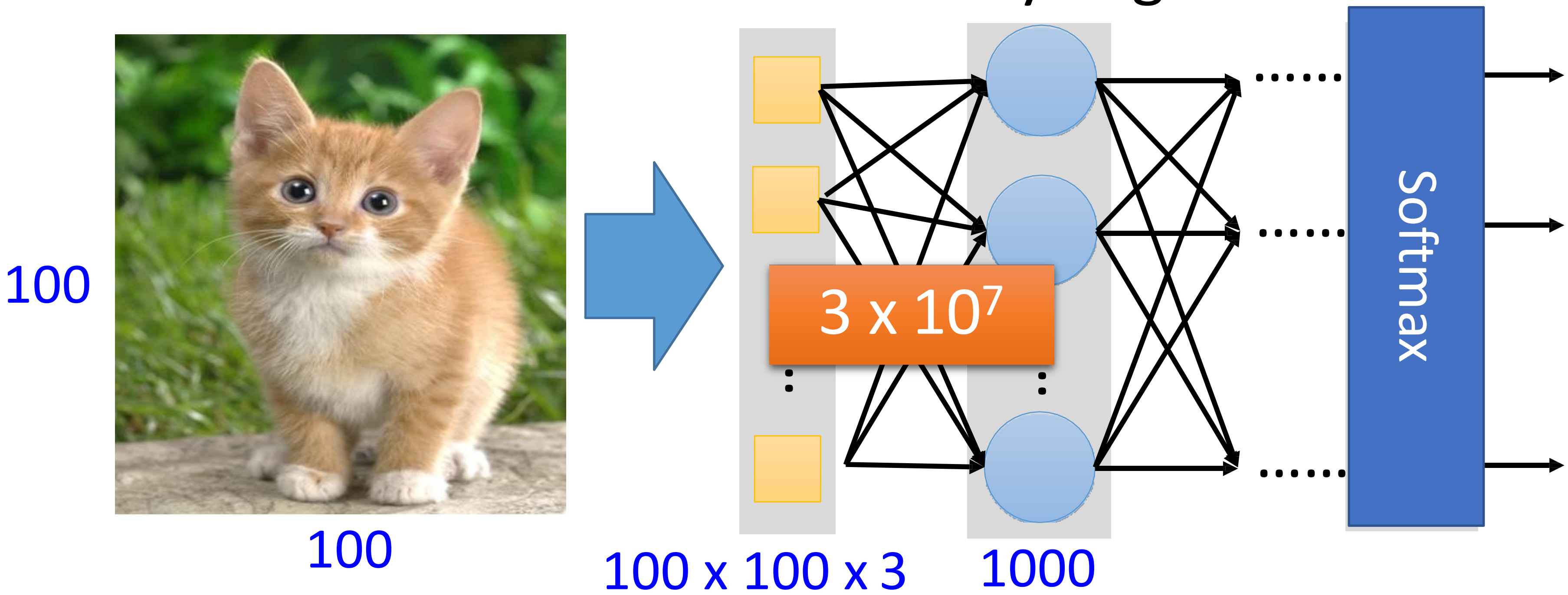
Convolutional Neural Network (CNN)

Widely used in image processing

Recurrent Neural Network (RNN)

Why CNN for Image?

- When processing image, the first layer of fully connected network would be very large



Can the fully connected network be simplified by considering the properties of image recognition?

Why CNN for Image

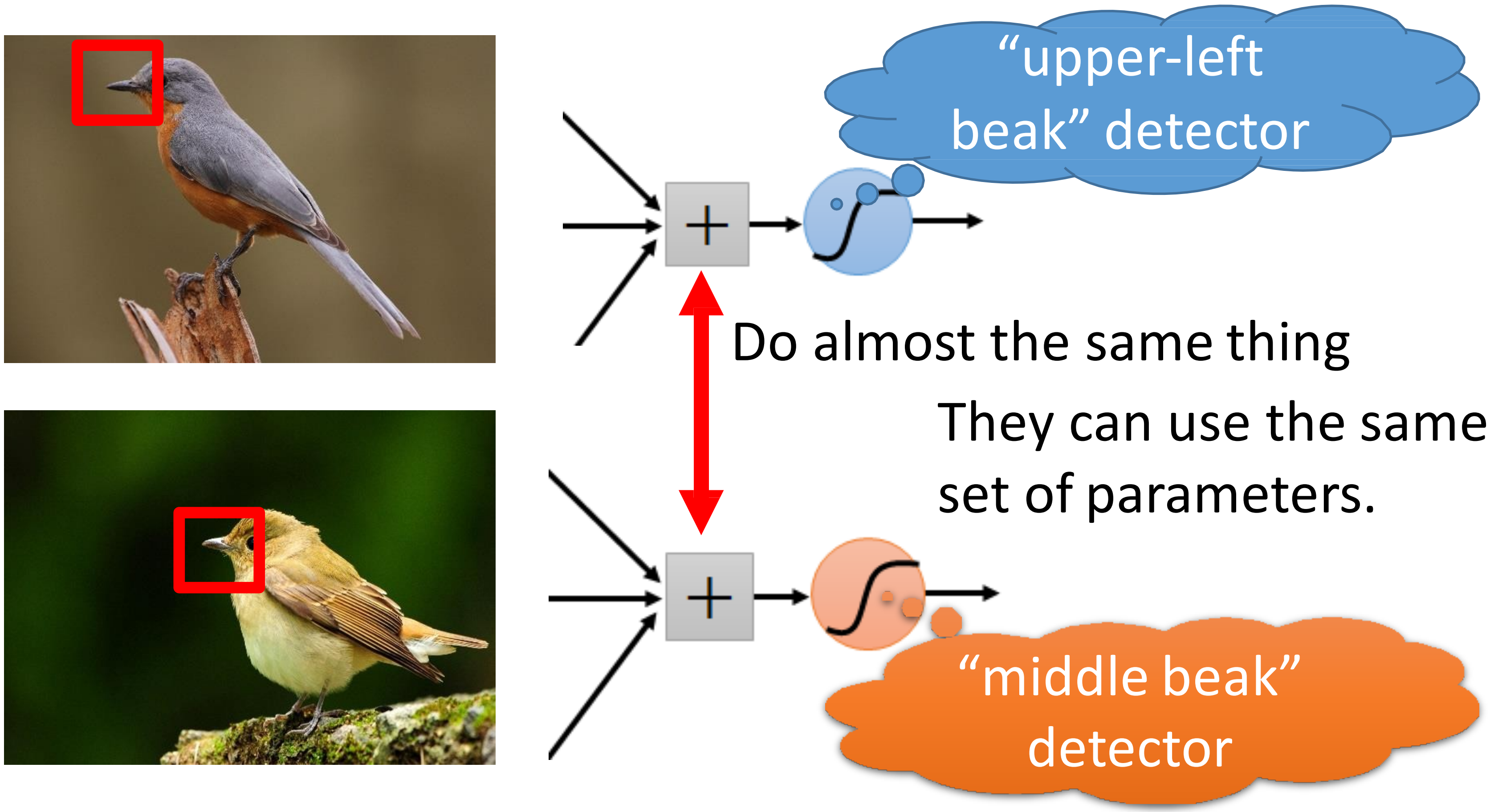
- Some patterns are much smaller than the whole image

A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters

Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

Subsampling the pixels will not change the object

bird

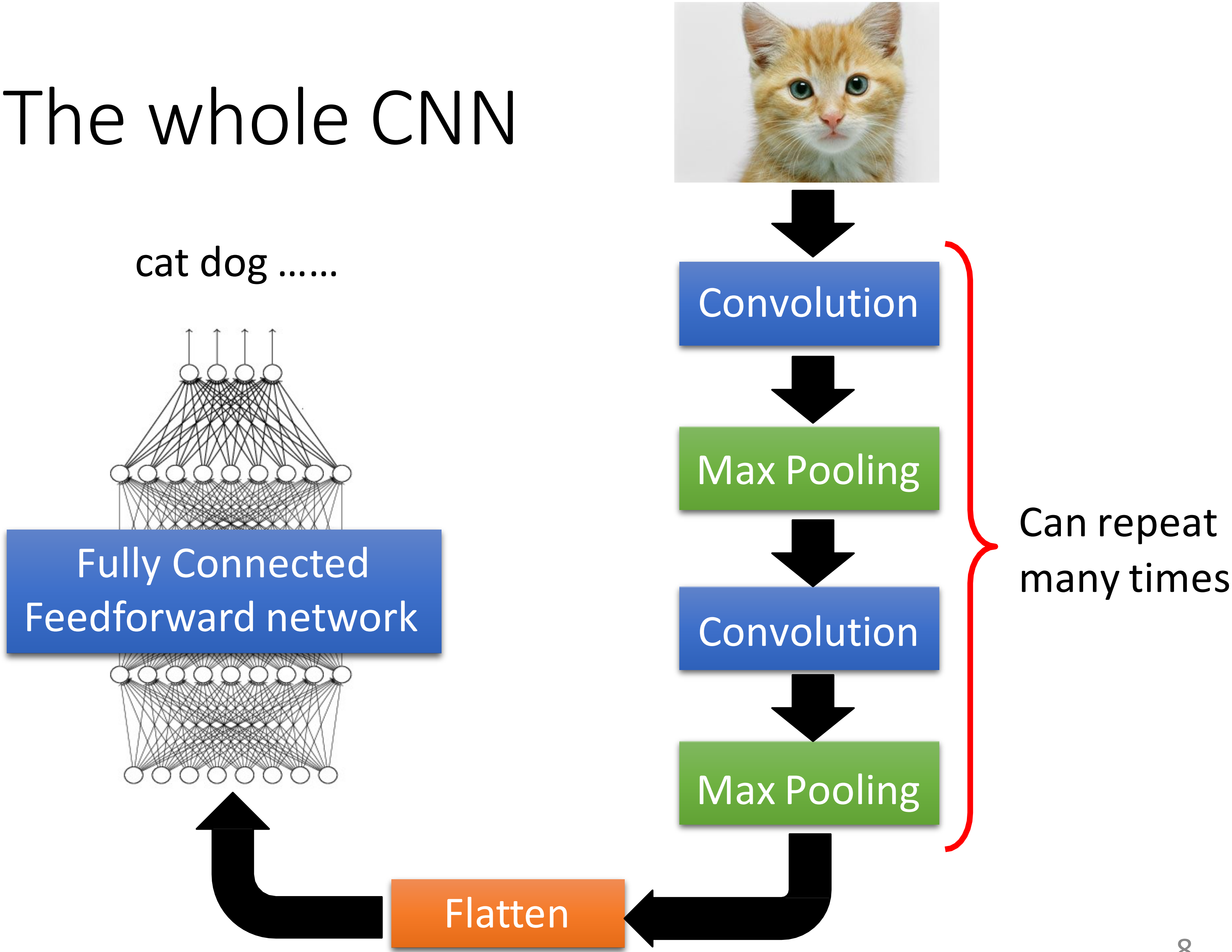
subsampling

bird

We can subsample the pixels to make image smaller

Less parameters for the network to process the image

The whole CNN



The whole CNN

Property 1

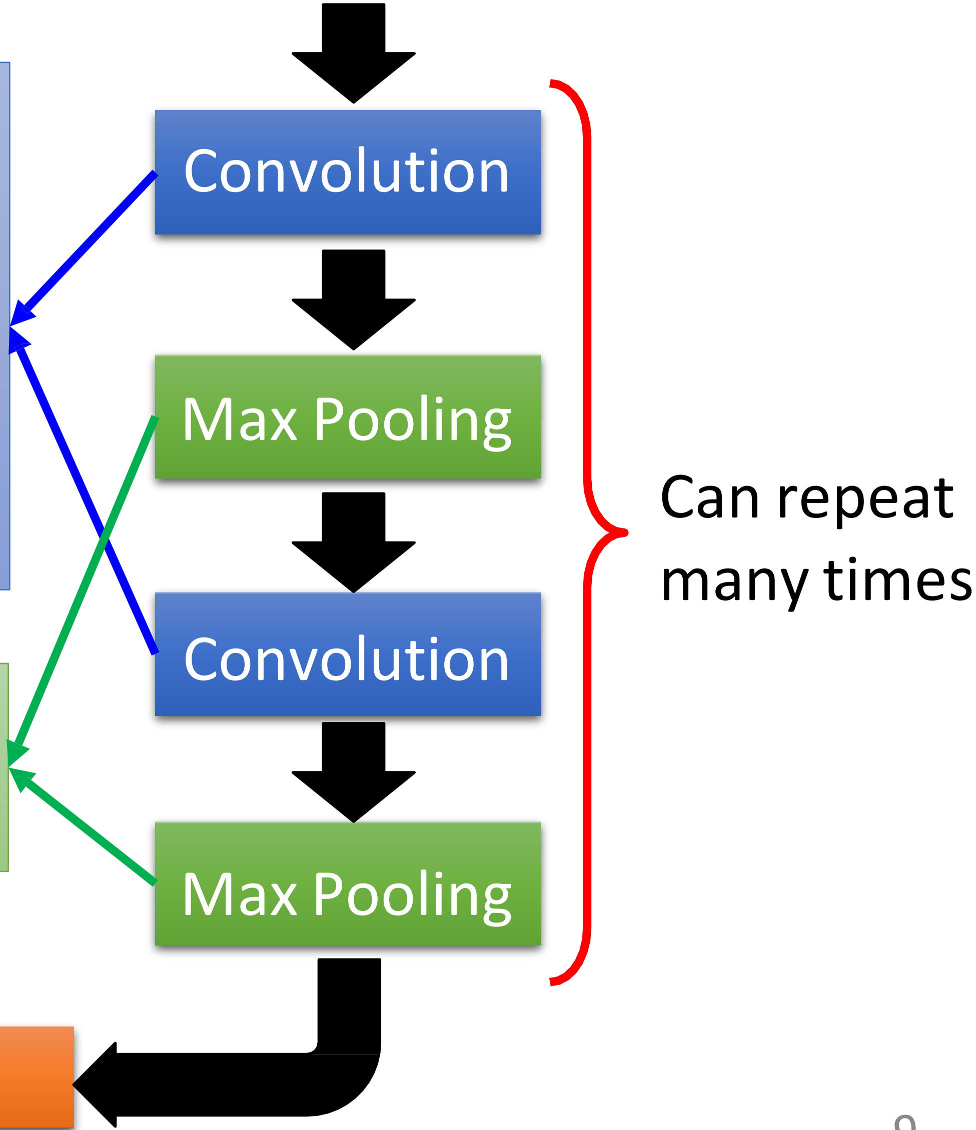
- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

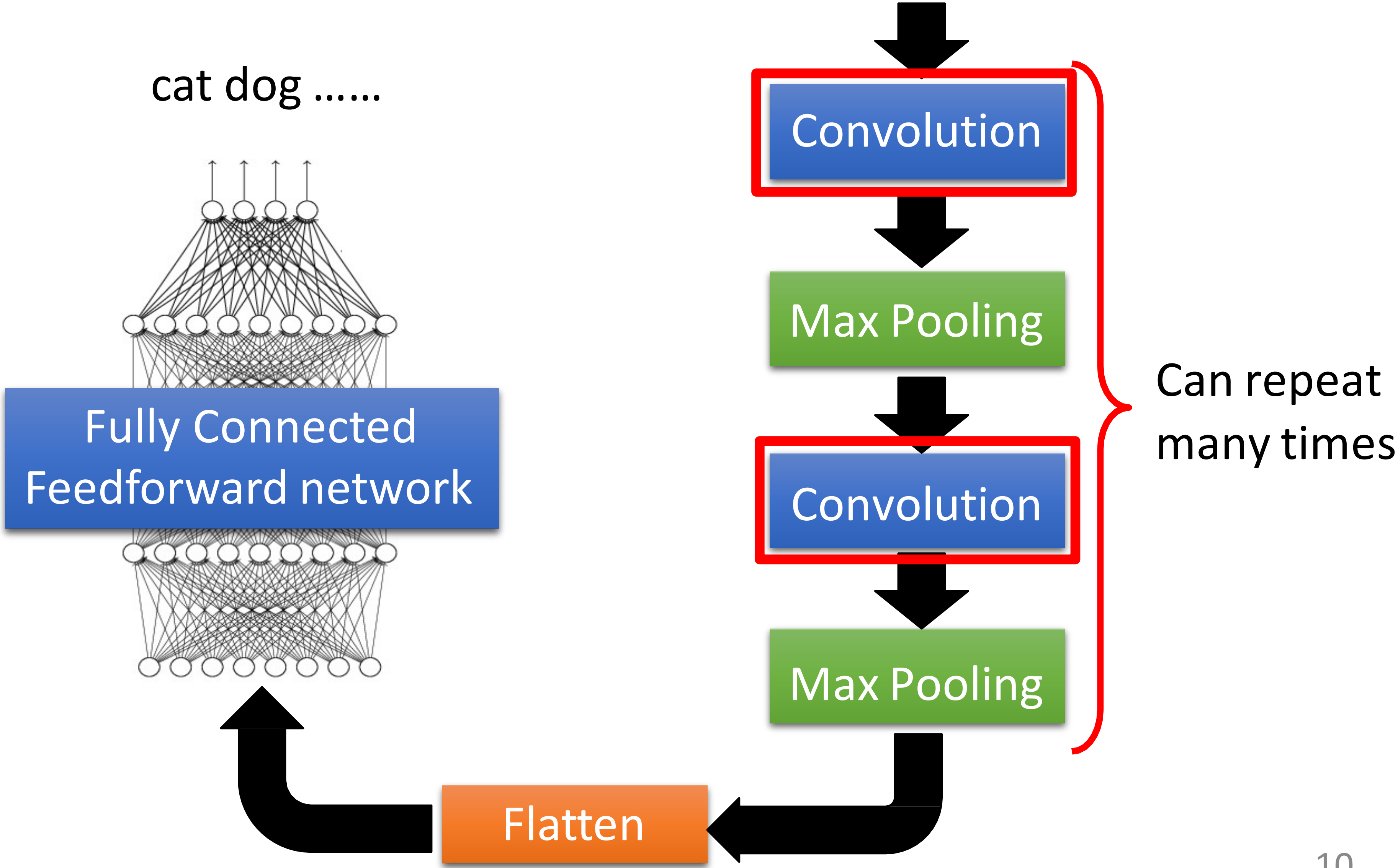
Property 3

- Subsampling the pixels will not change the object



Flatten

The whole CNN



CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -1

CNN – Convolution

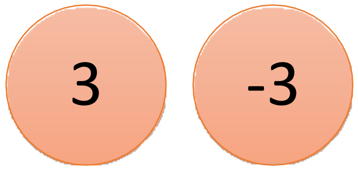
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



We set stride=1 below

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2

CNN – Convolution

-1	1	-1
-1	1	-1
-1	1	-1

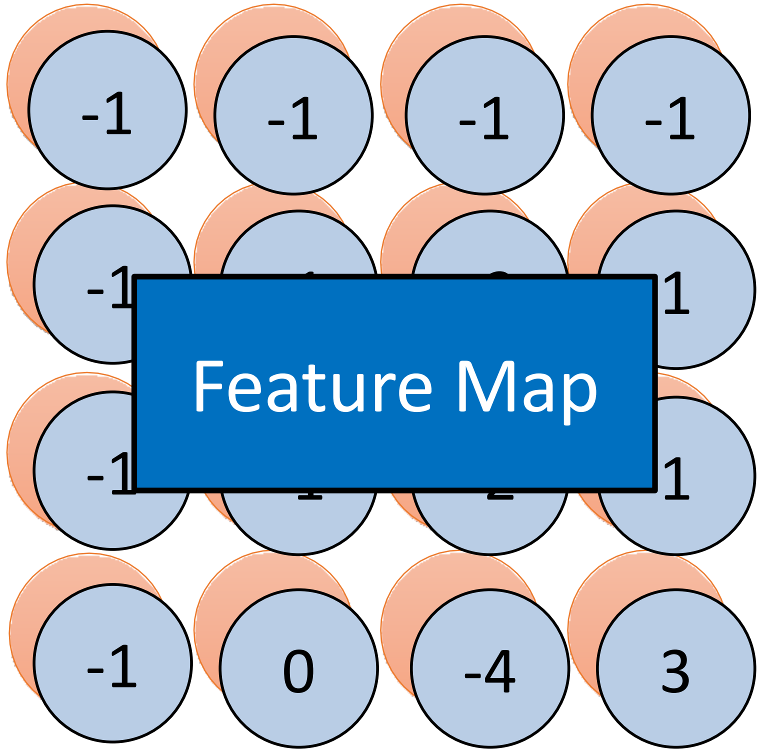
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Do the same process for every filter



4 x 4 image

CNN – Zero Padding

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

0	0	0					
0	1	0	0	0	0	1	
0	0	1	0	0	1	0	
	0	0	1	1	0	0	
	1	0	0	0	1	0	
	0	1	0	0	1	0	0
	0	0	1	0	1	0	0
					0	0	0

6 x 6 image

You will get another 6 x 6 images in this way

Zero padding

CNN – Colorful image

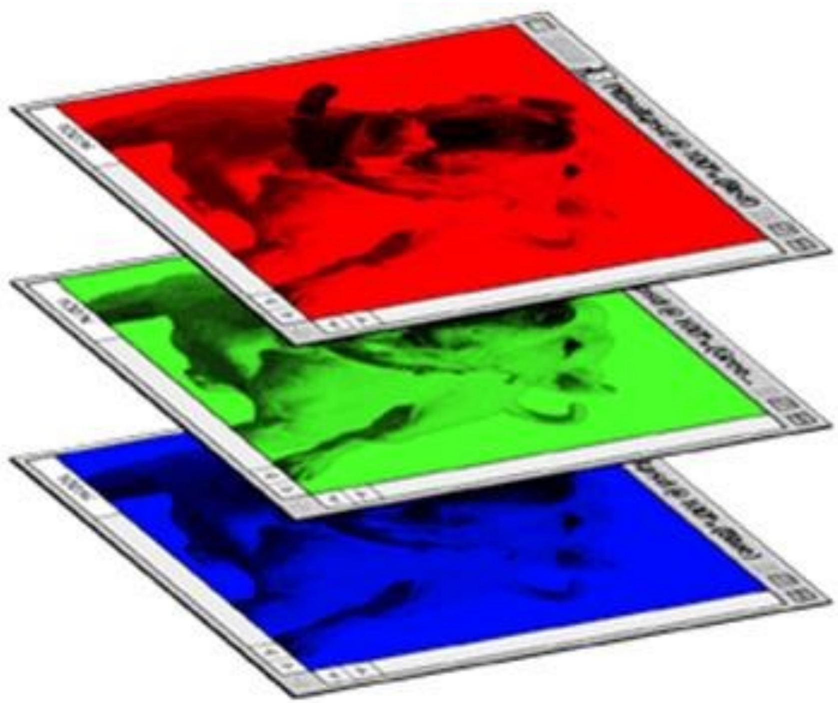
Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

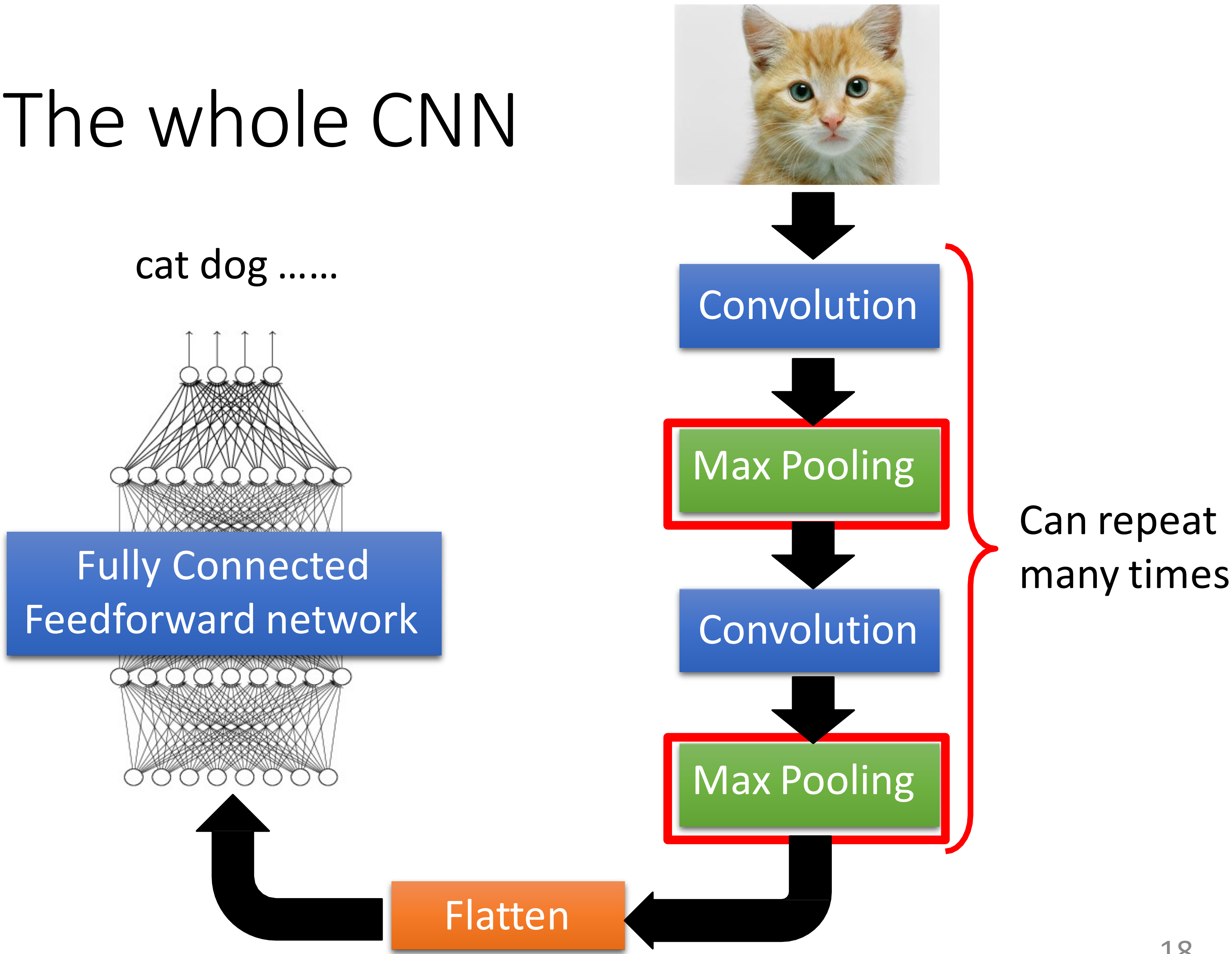
-1	1	-1
-1	1	-1
-1	1	-1

Colorful image



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

The whole CNN



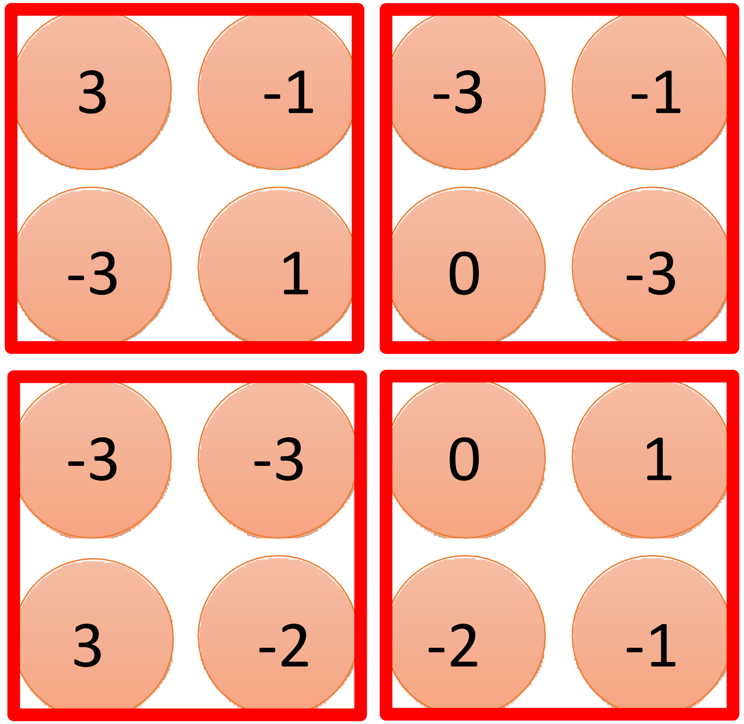
CNN – Max Pooling

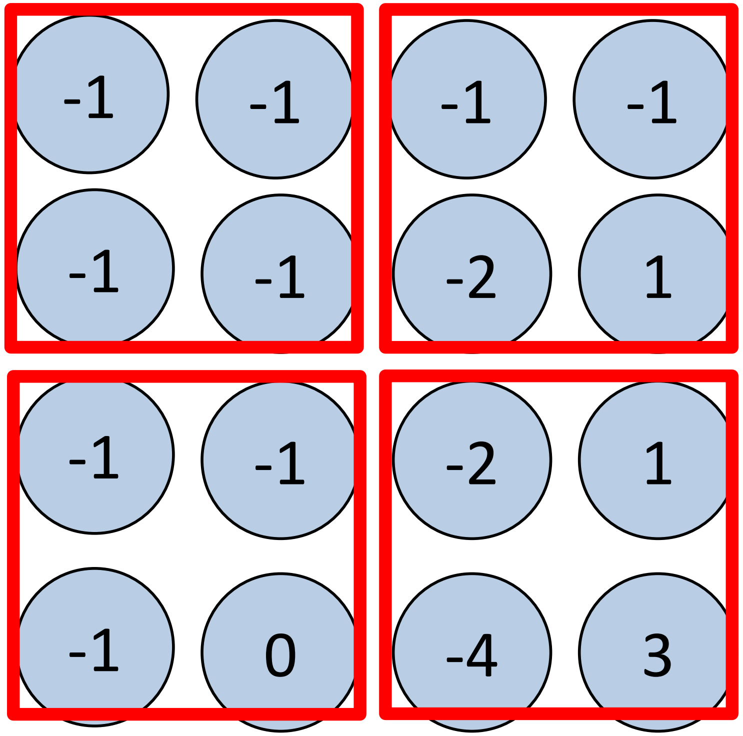
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2



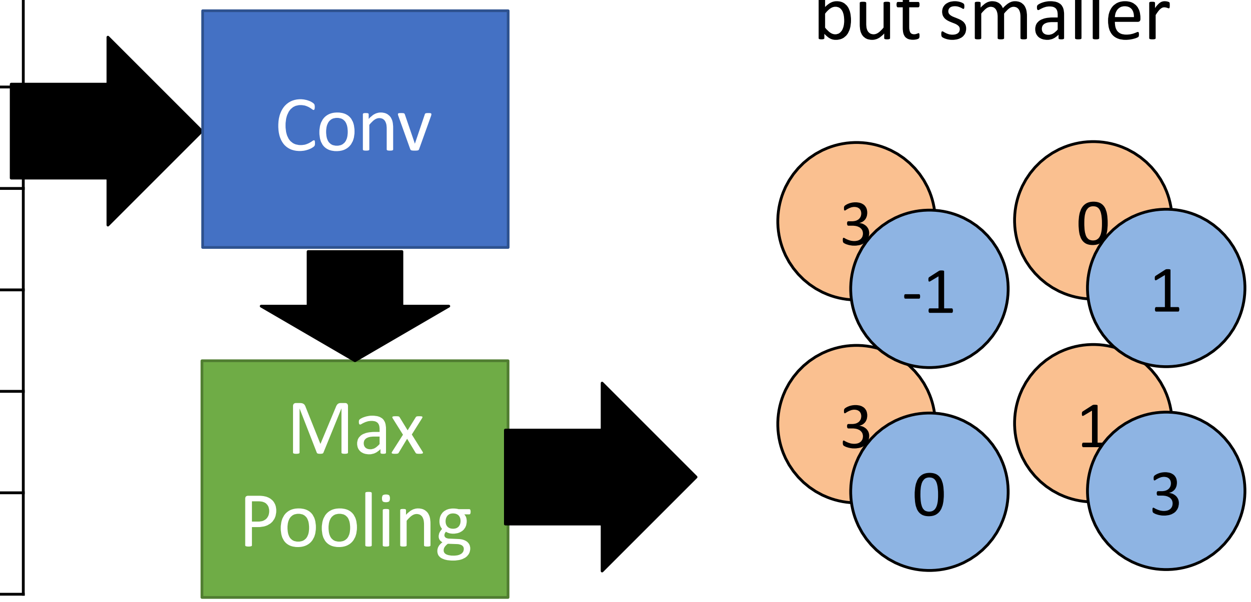


CNN – Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

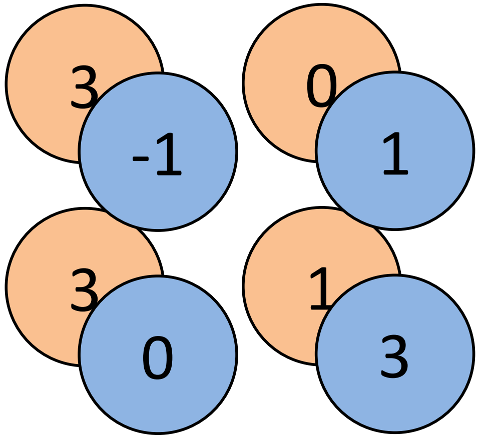
New image but smaller



2 x 2 image

Each filter is a channel

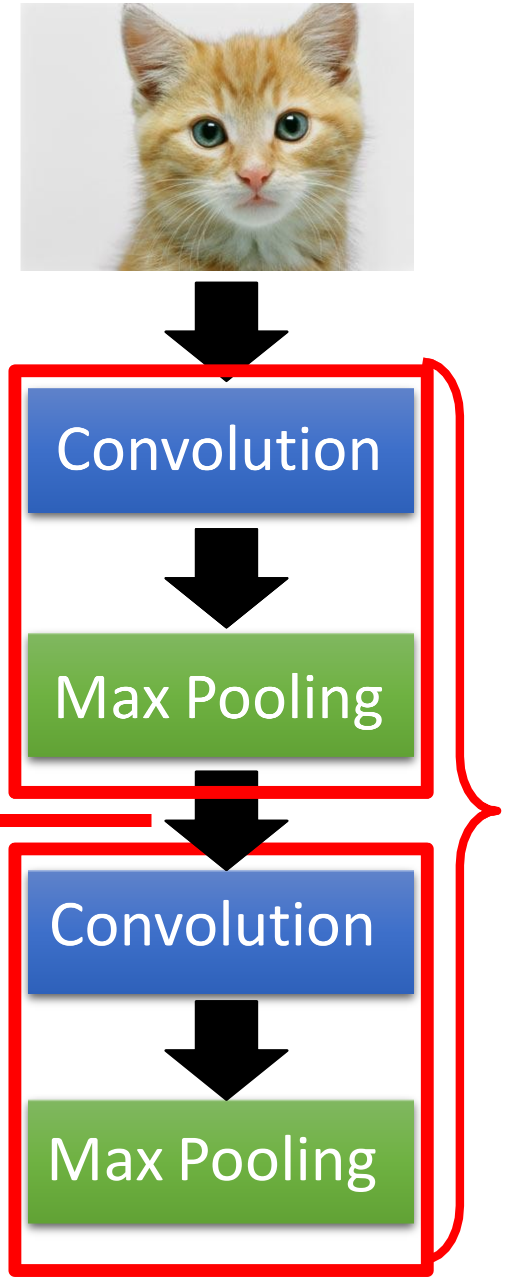
The whole CNN



A new image

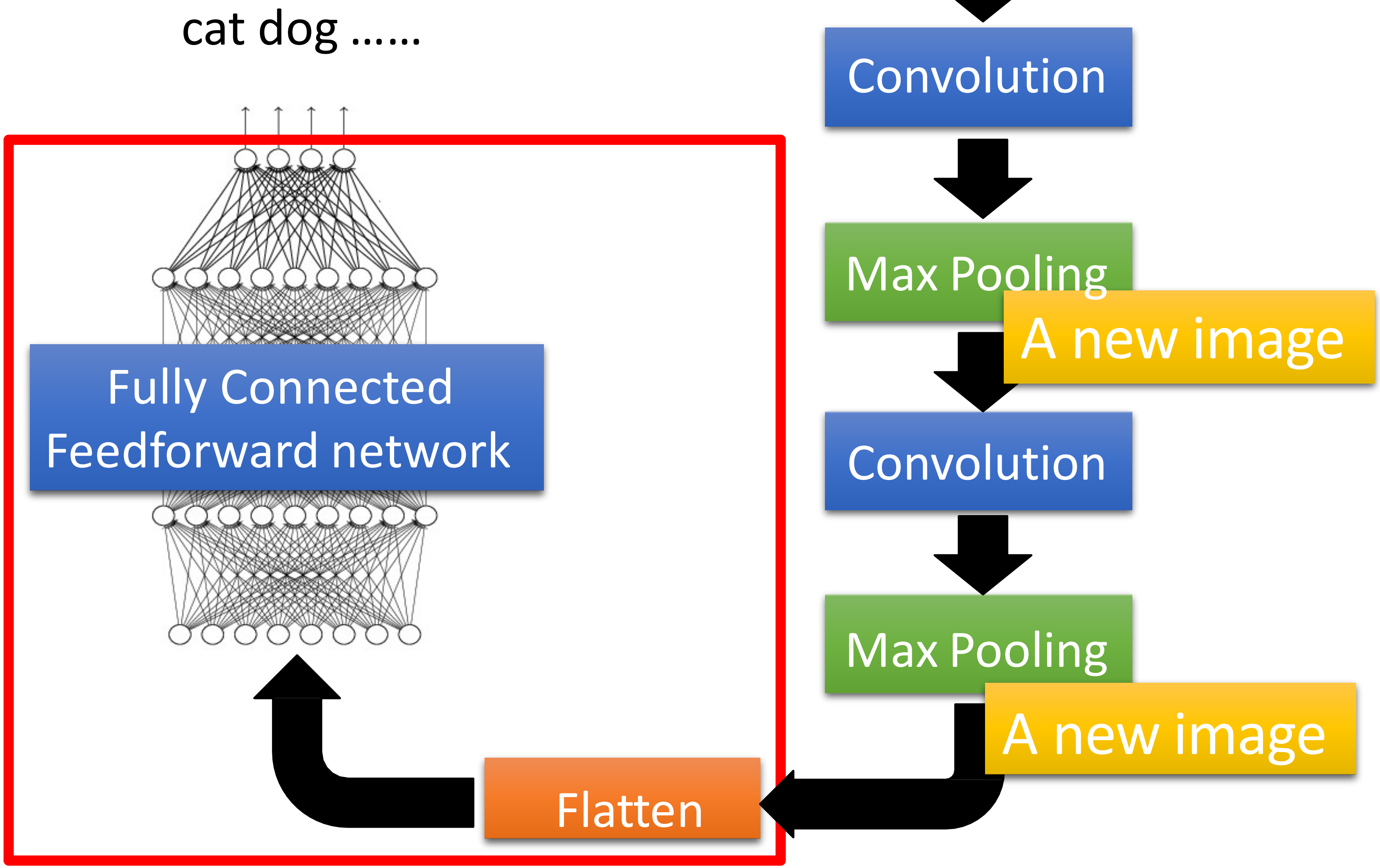
Smaller than the original image

The number of the channel is the number of filters

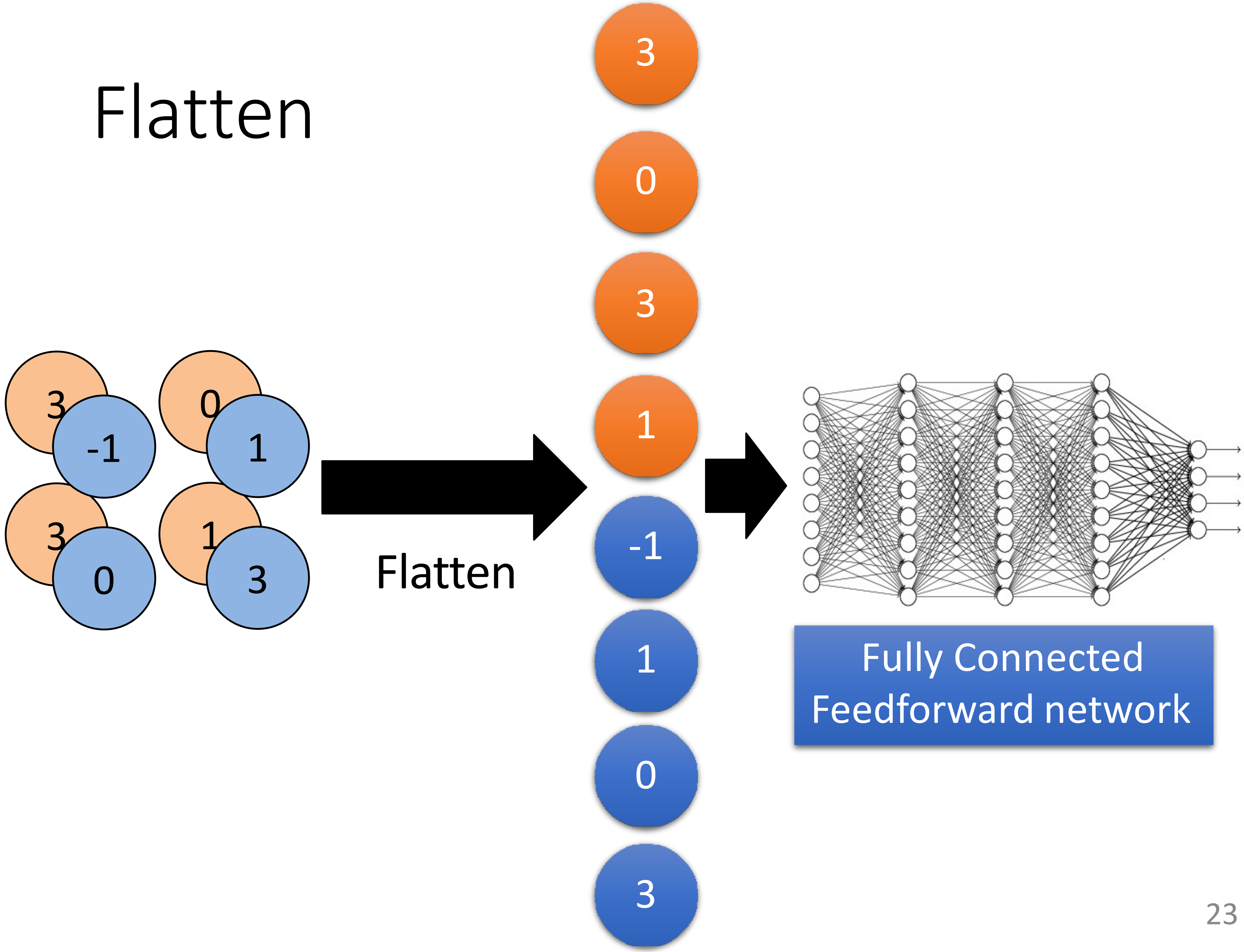


Can repeat many times

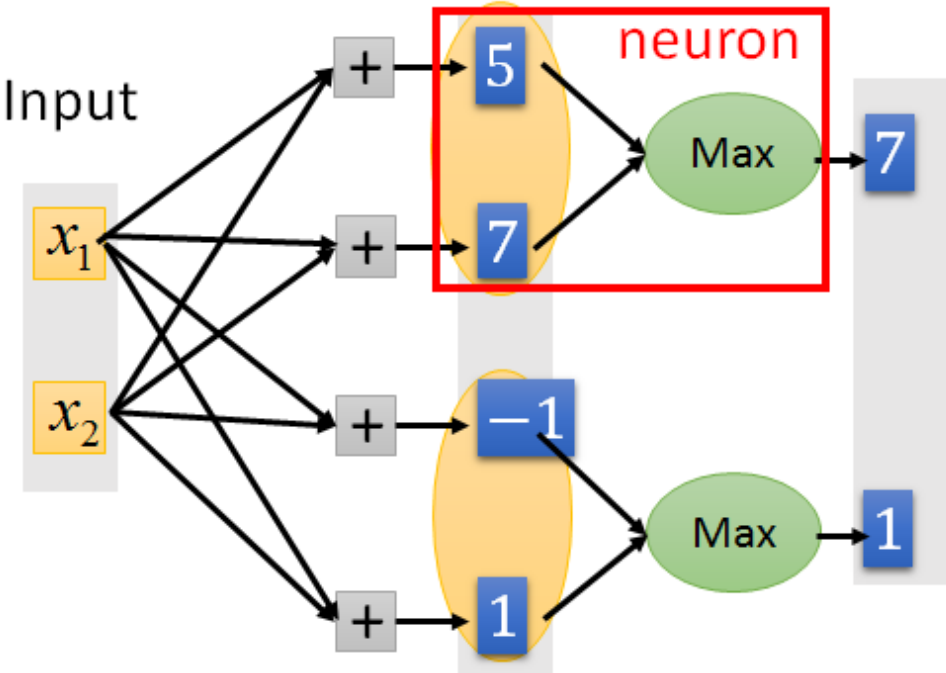
The whole CNN

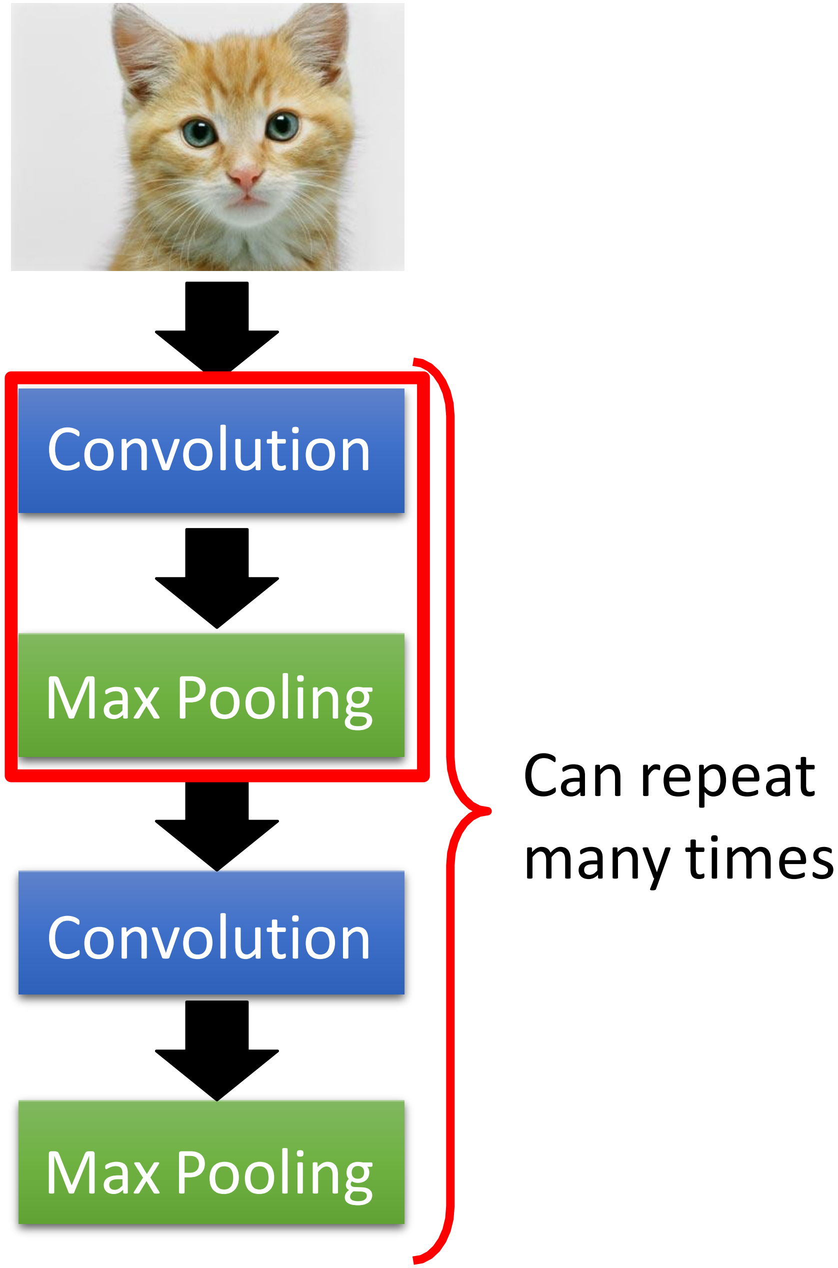


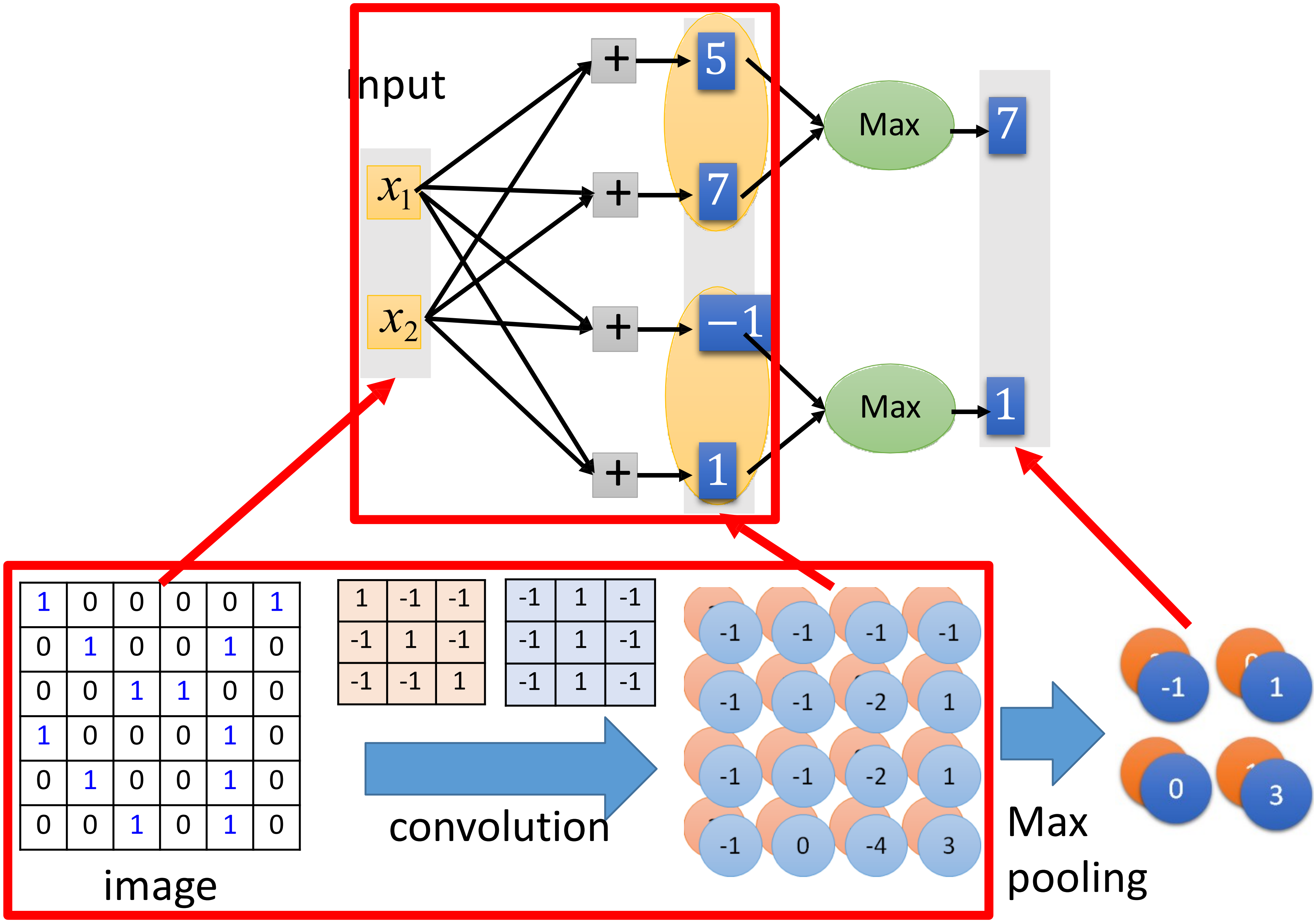
Flatten


3
-1
0
1
3
0
1
3
Flatten
3
0
3
1
-1
1
0
3
Fully Connected
Feedforward network

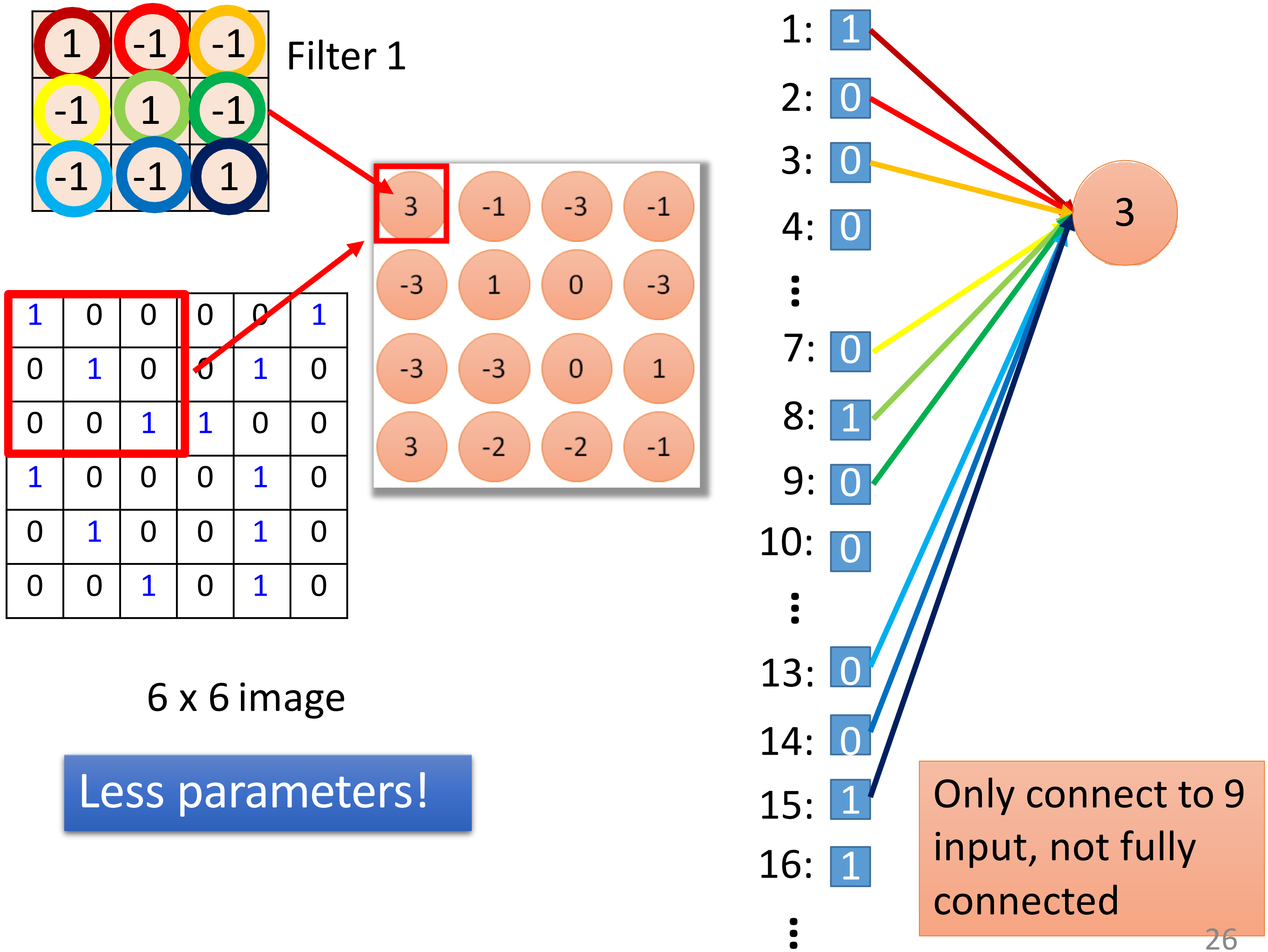
The whole CNN

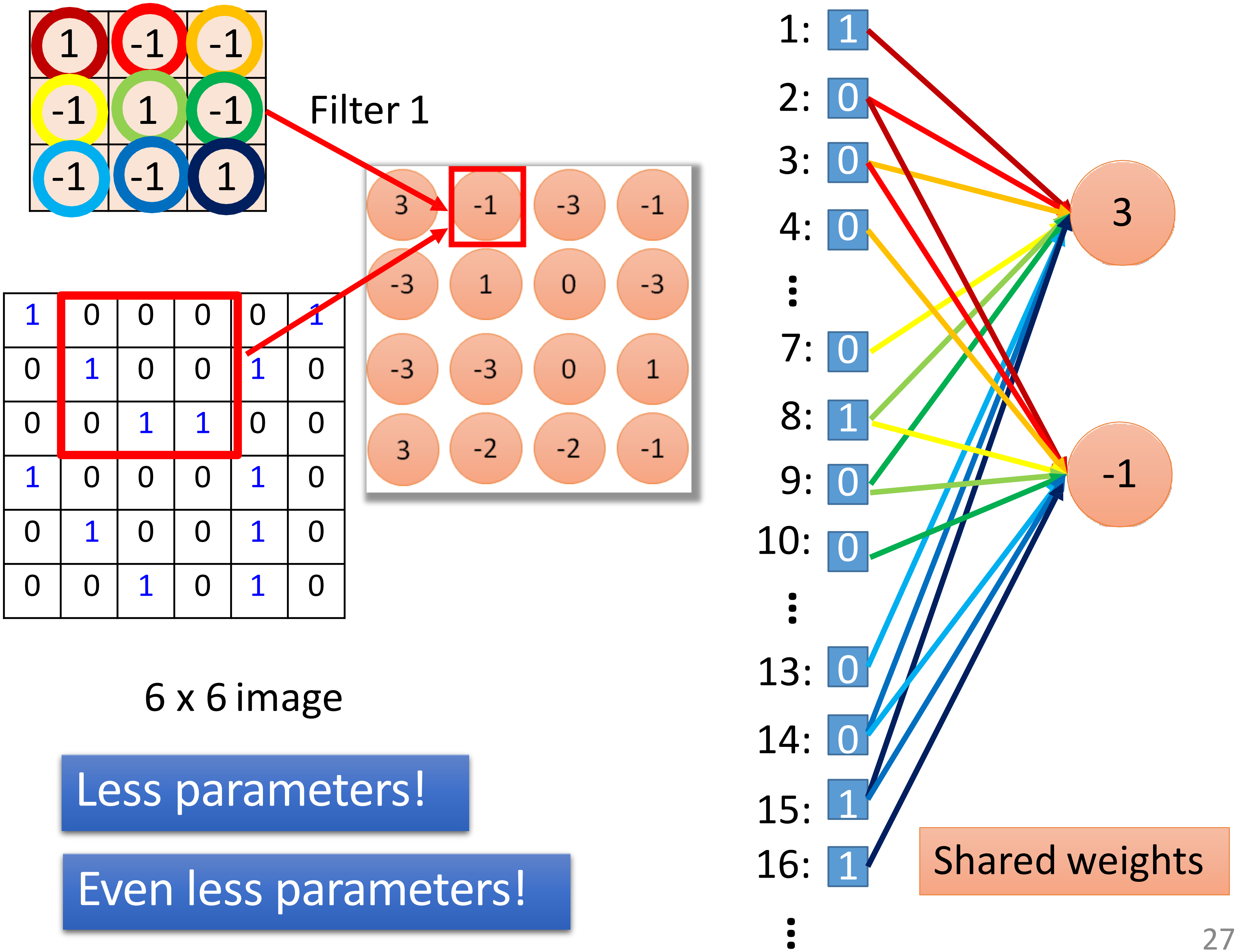




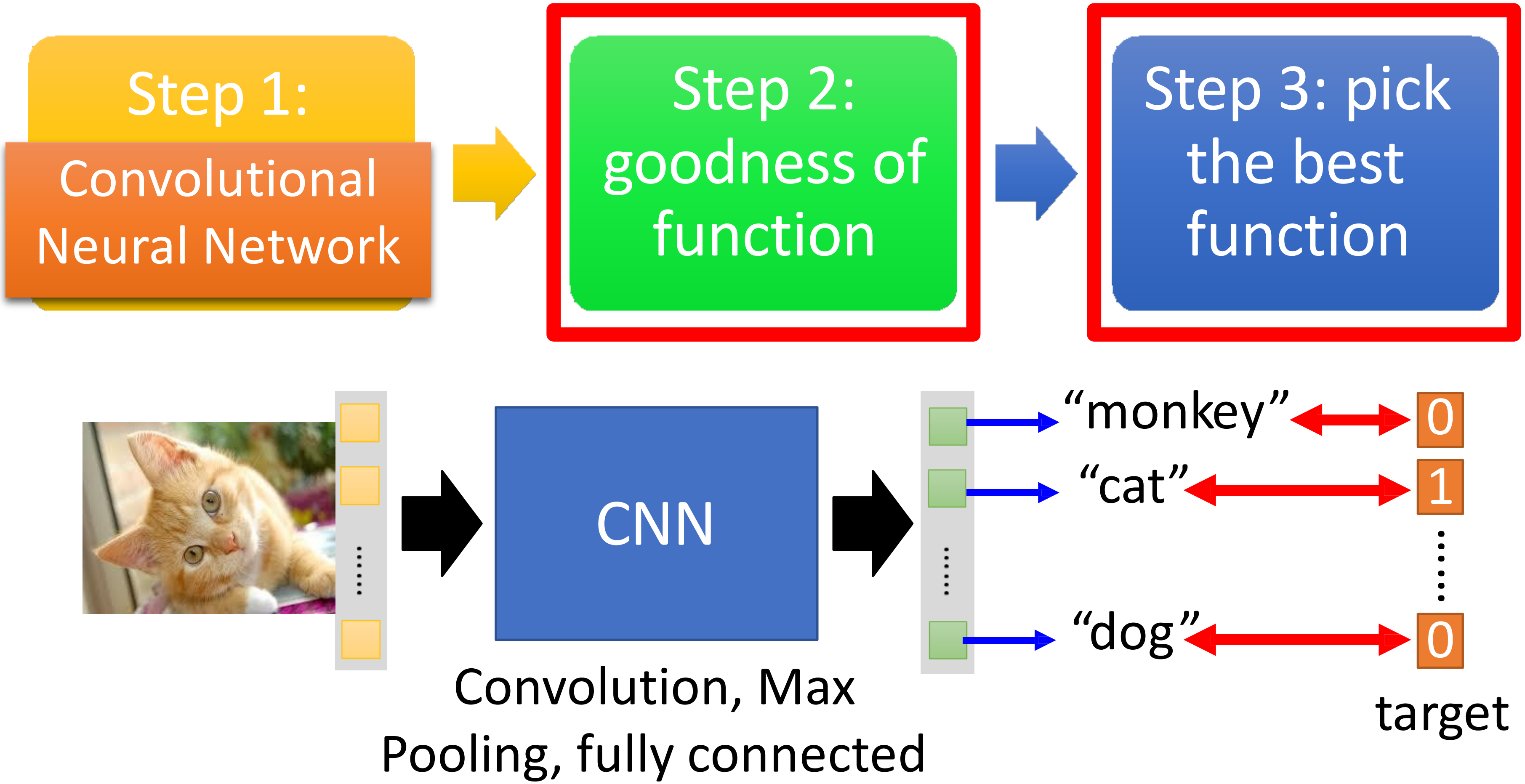


(Ignoring the non-linear activation function after the convolution.)


Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
6 x 6 image
Less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
Only connect to 9 input, not fully connected


1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Less parameters!
Even less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1
Shared weights

Convolutional Neural Network



Learning: Nothing special, just gradient descent ……

Playing Go

19 x 19 matrix (image)

Network

Next move (19 x 19 positions)

19 x 19 vector

Black: 1

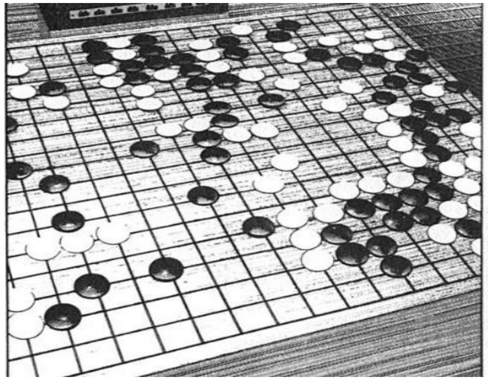
white: -1

none: 0

Fully-connected feedword network can be used

But CNN performs much better.

Playing Go



record of previous plays

進藤光 v.s. 社清春

黑: 5之五

→ 白: 天元

→ 黑: 五之5

Training:

Network

Target: “天元” = 1
else = 0

Network

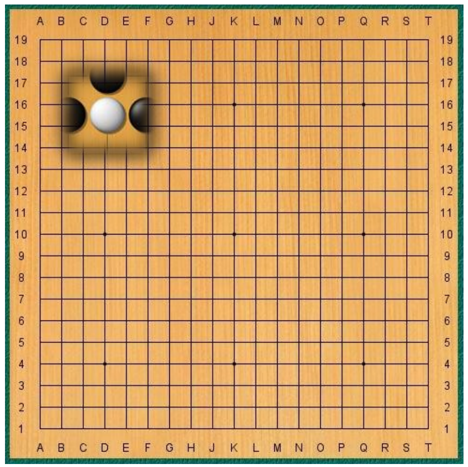
Target:
“五之 5” = 1
else = 0

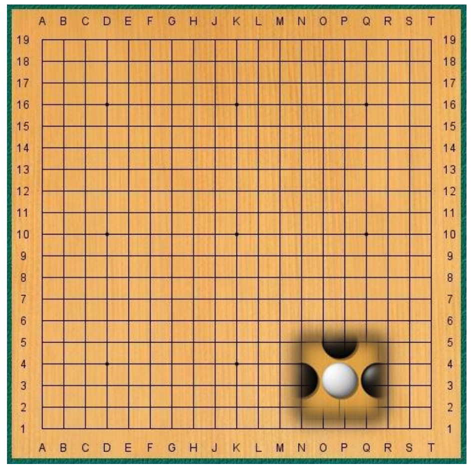
Why CNN for playing Go?

- Some patterns are much smaller than the whole image

Alpha Go uses 5 x 5 for first layer

- The same patterns appear in different regions.





Why CNN for playing Go?

Subsampling the pixels will not change the object

Max Pooling

How to explain this?

Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

Alpha Go does not use Max Pooling ……

Variants of Neural Networks

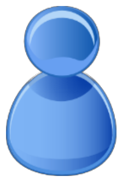
Convolutional Neural Network (CNN)

Recurrent Neural Network (RNN)

Neural Network with Memory

Example Application

- Slot Filling



I would like to arrive Taipei on November 2$^{\text{nd}}$.

ticket booking system

Slot

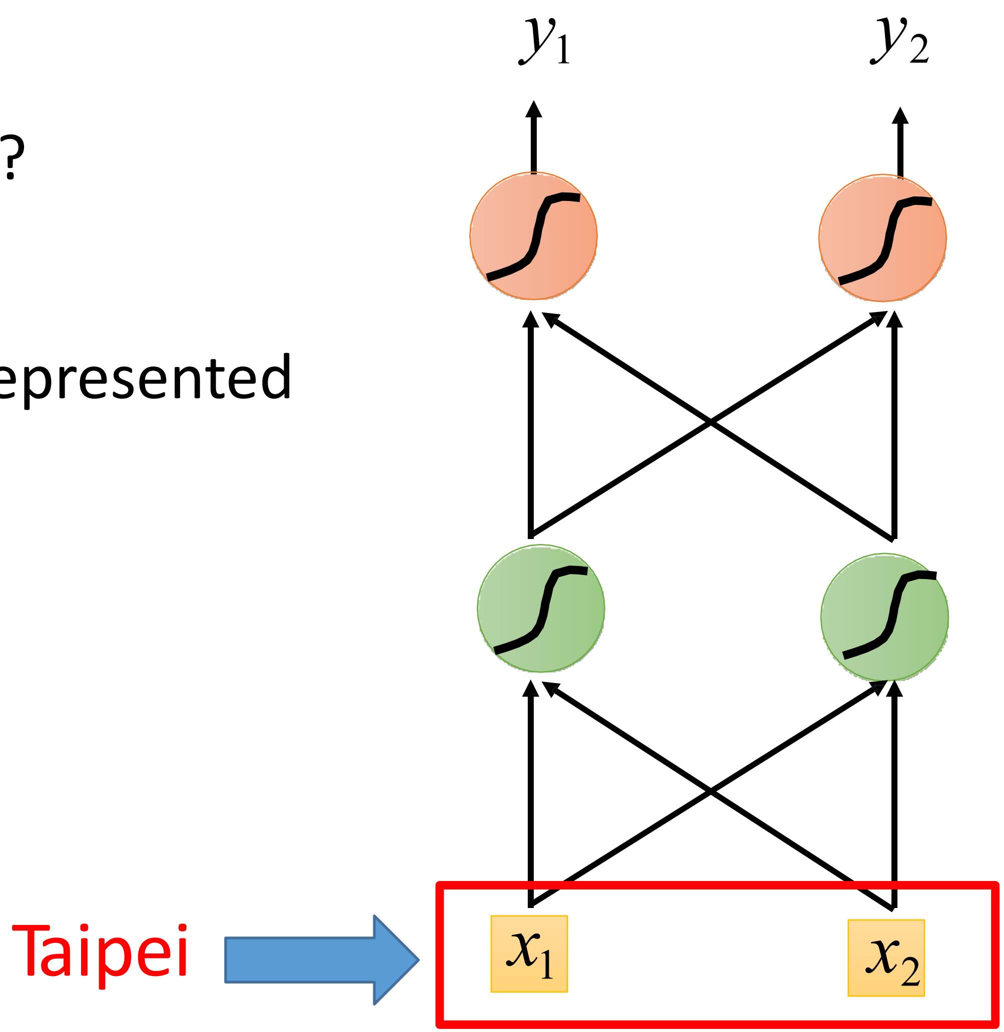
Destination:	Taipei
time of arrival:	November 2$^{\text{nd}}$

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?

<u>1-of-N Encoding</u> 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]

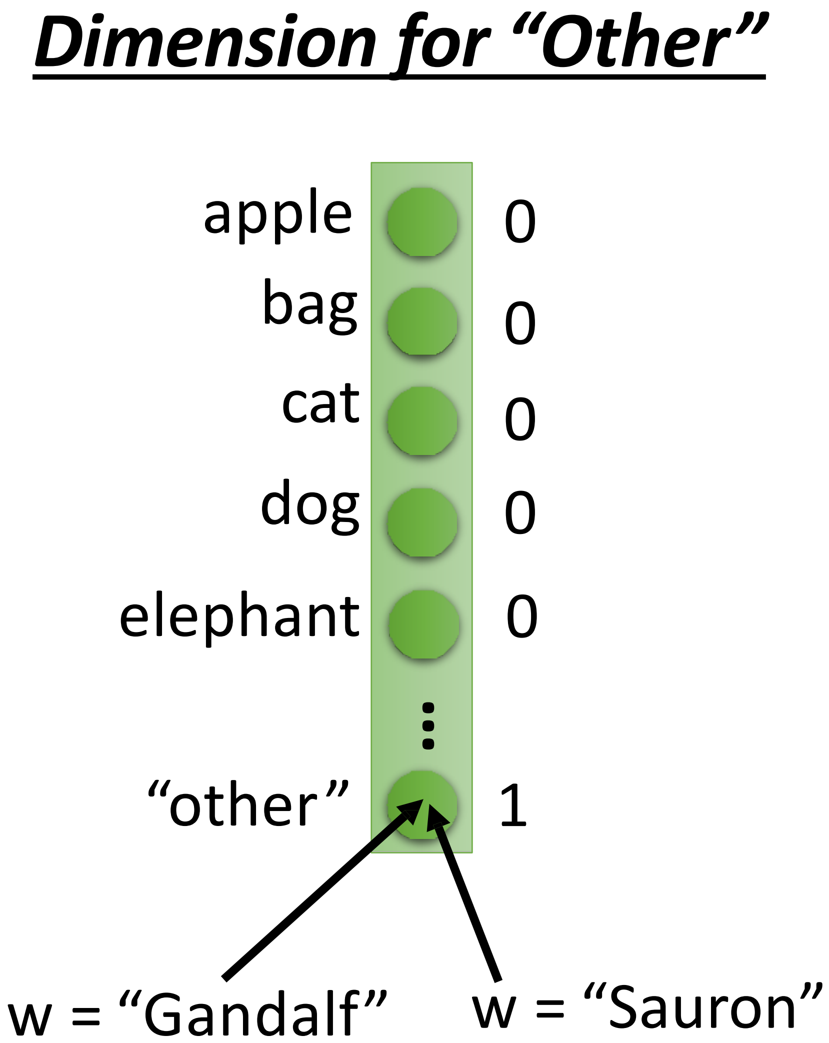
bag = [0 1 0 0 0]

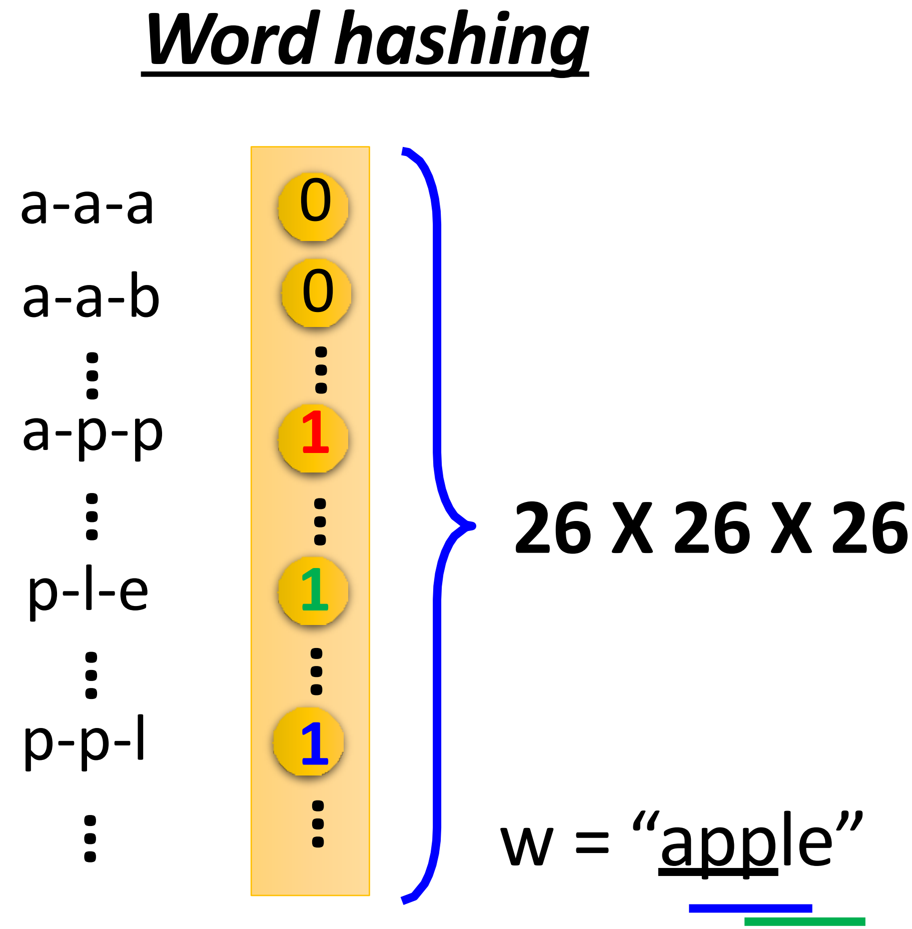
cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

elephant = [0 0 0 0 1]

Beyond 1-of-N encoding





Example Application

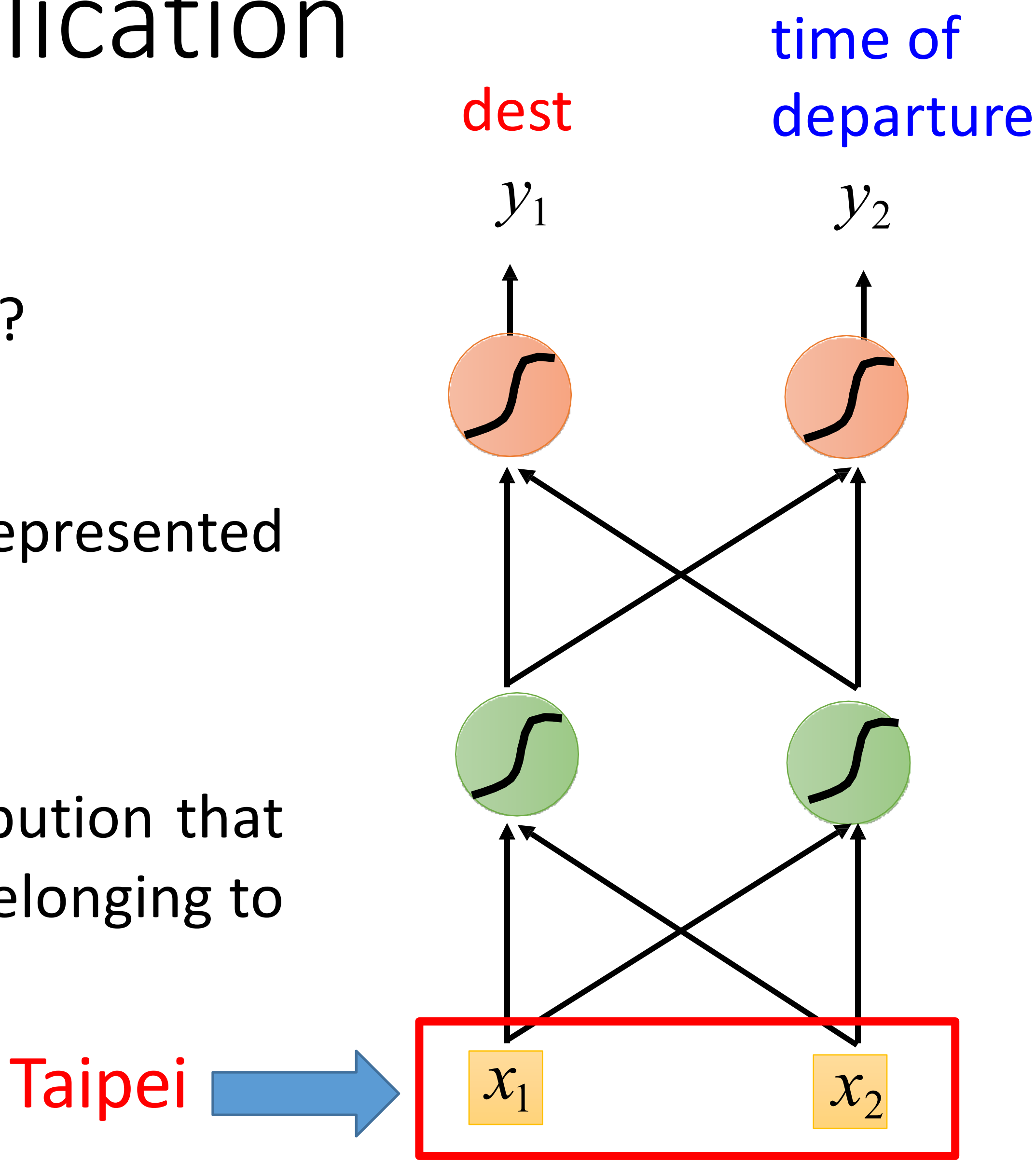
Solving slot filling by Feedforward network?

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}

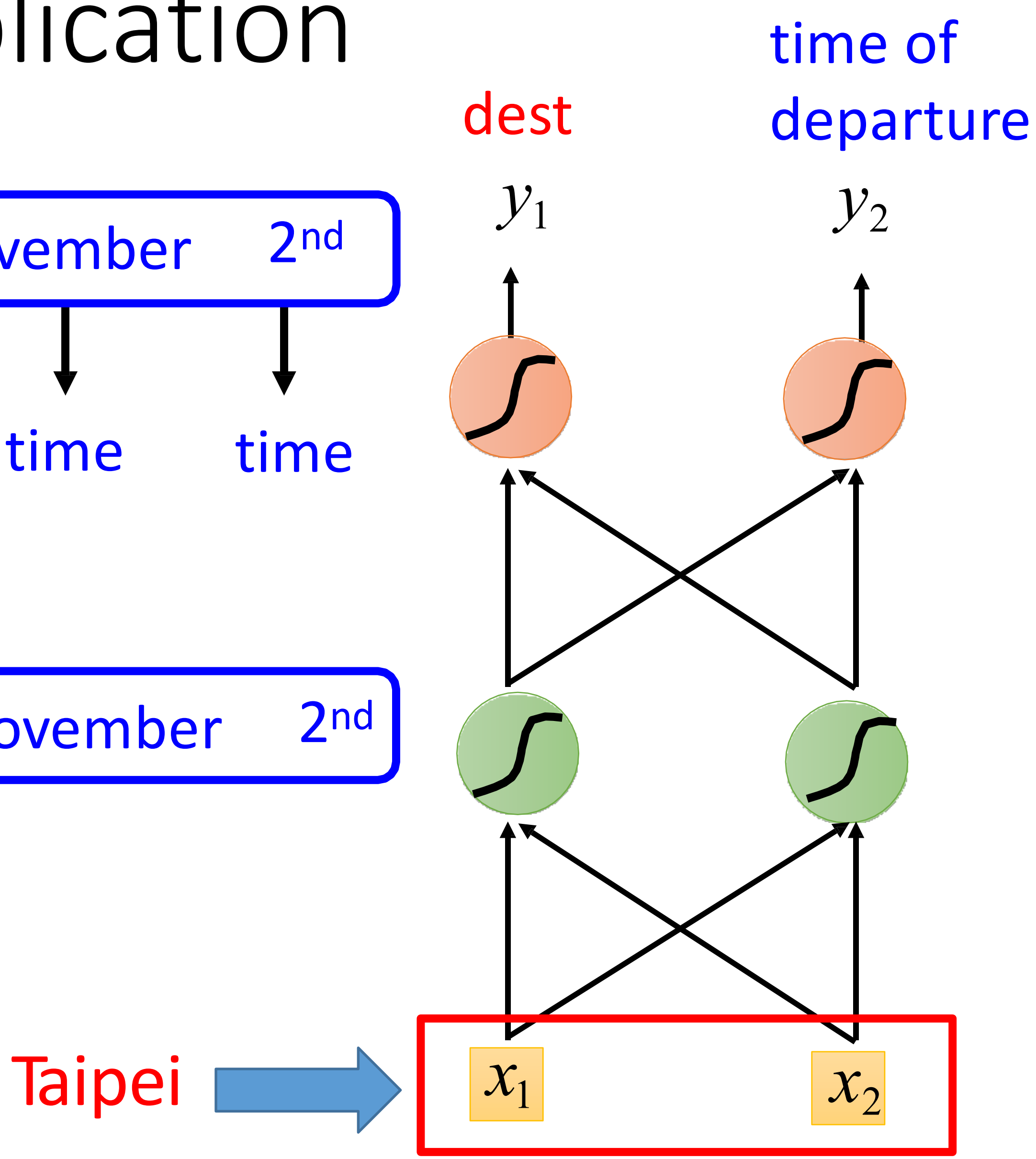
other dest other time time

Problem?

leave Taipei on November 2^{nd}

place of departure

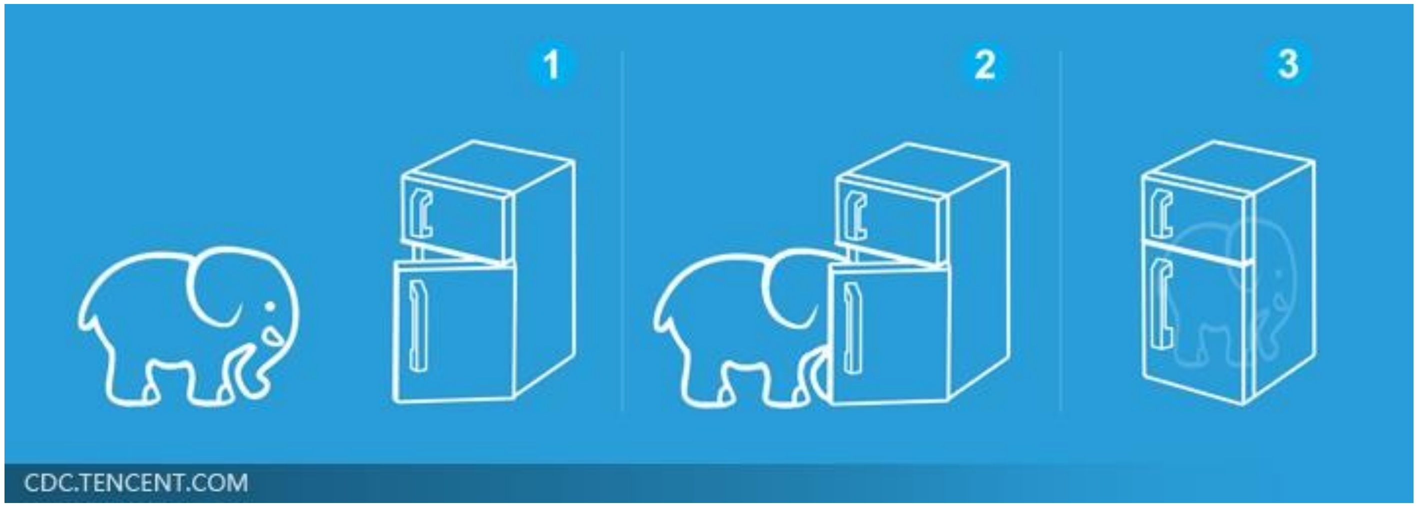
Neural network needs memory!



Three Steps for Deep Learning

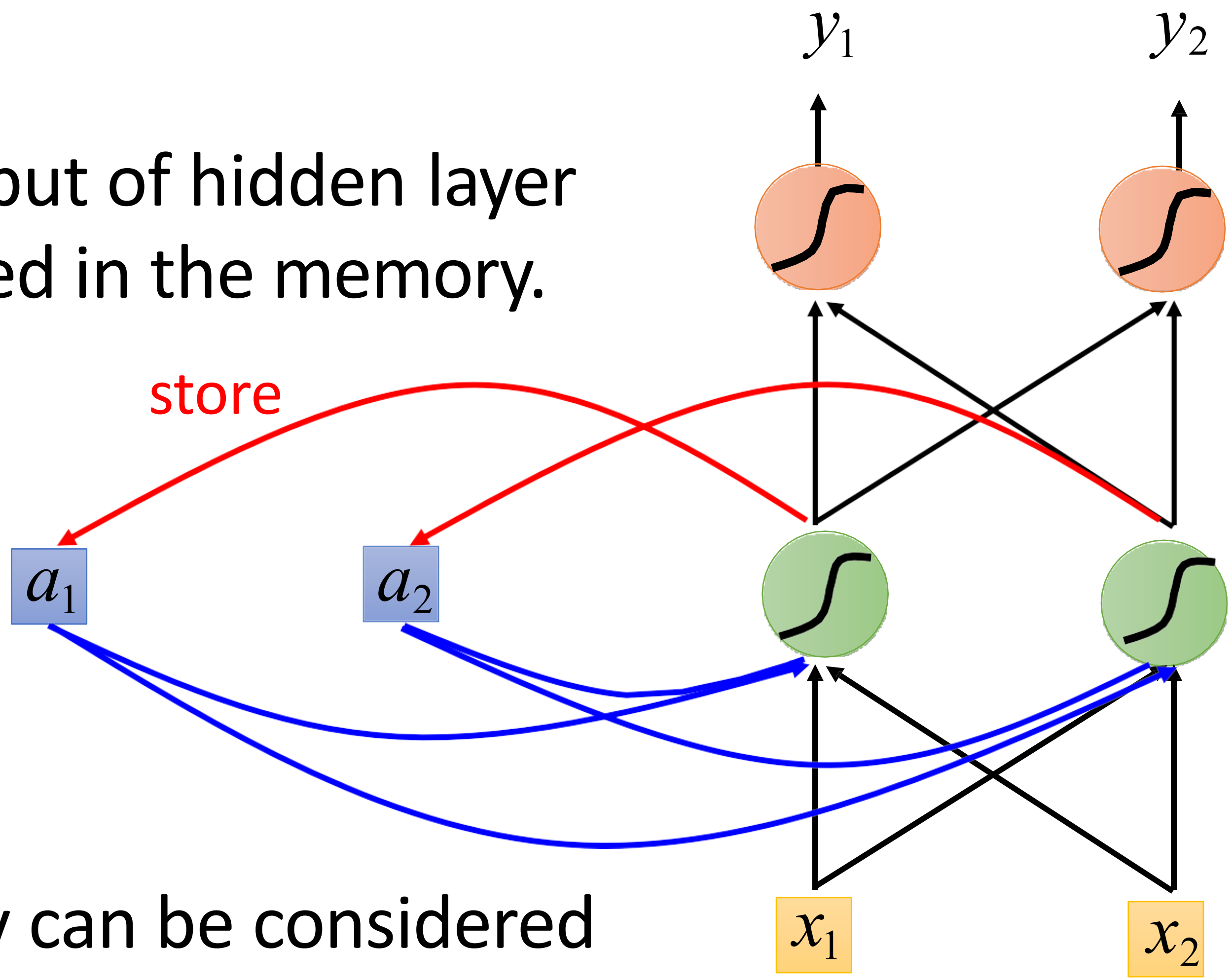
Step 1: Recurrent Neural Network → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



Recurrent Neural Network (RNN)

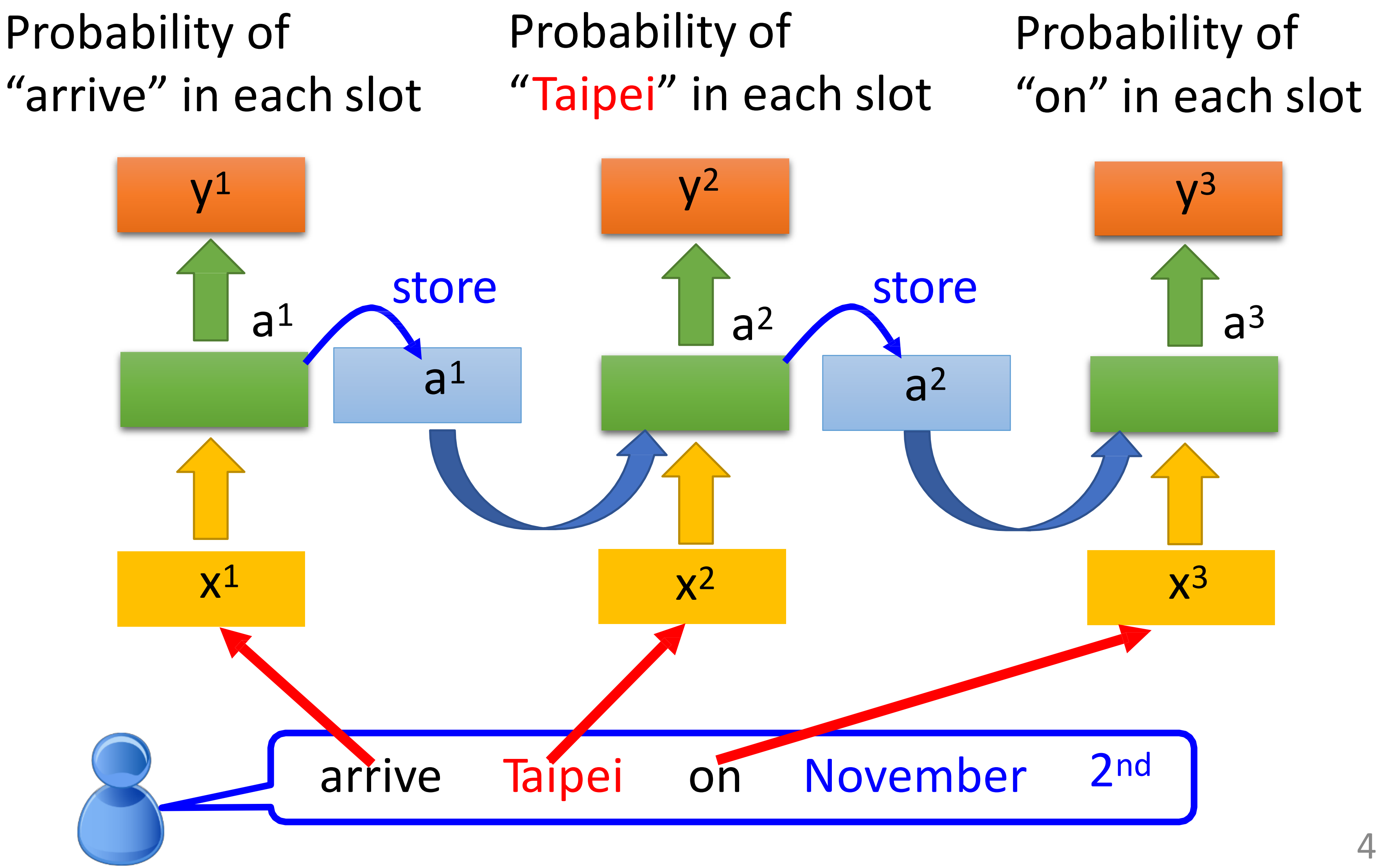
The output of hidden layer are stored in the memory.



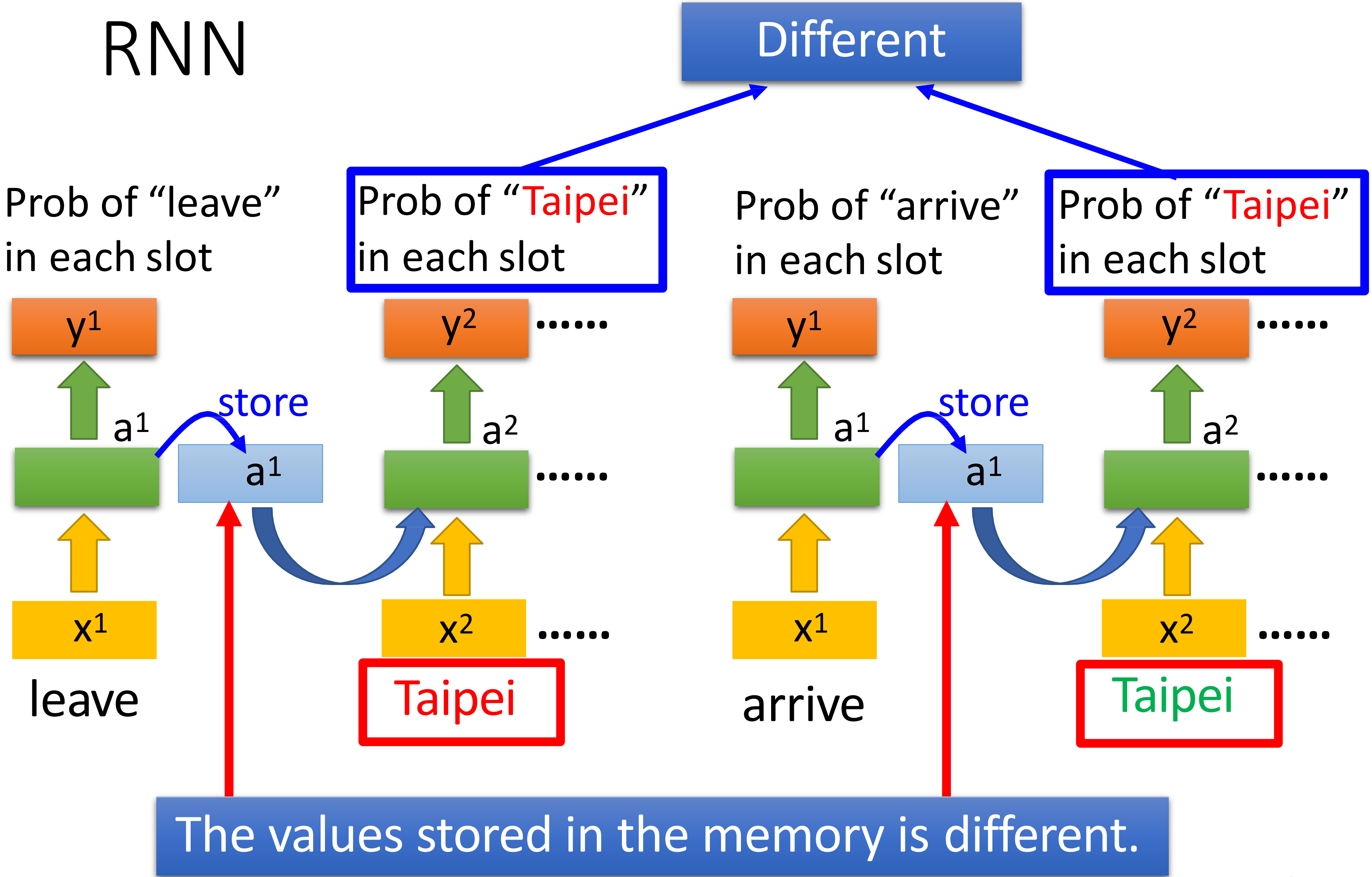
Memory can be considered as another input.

RNN

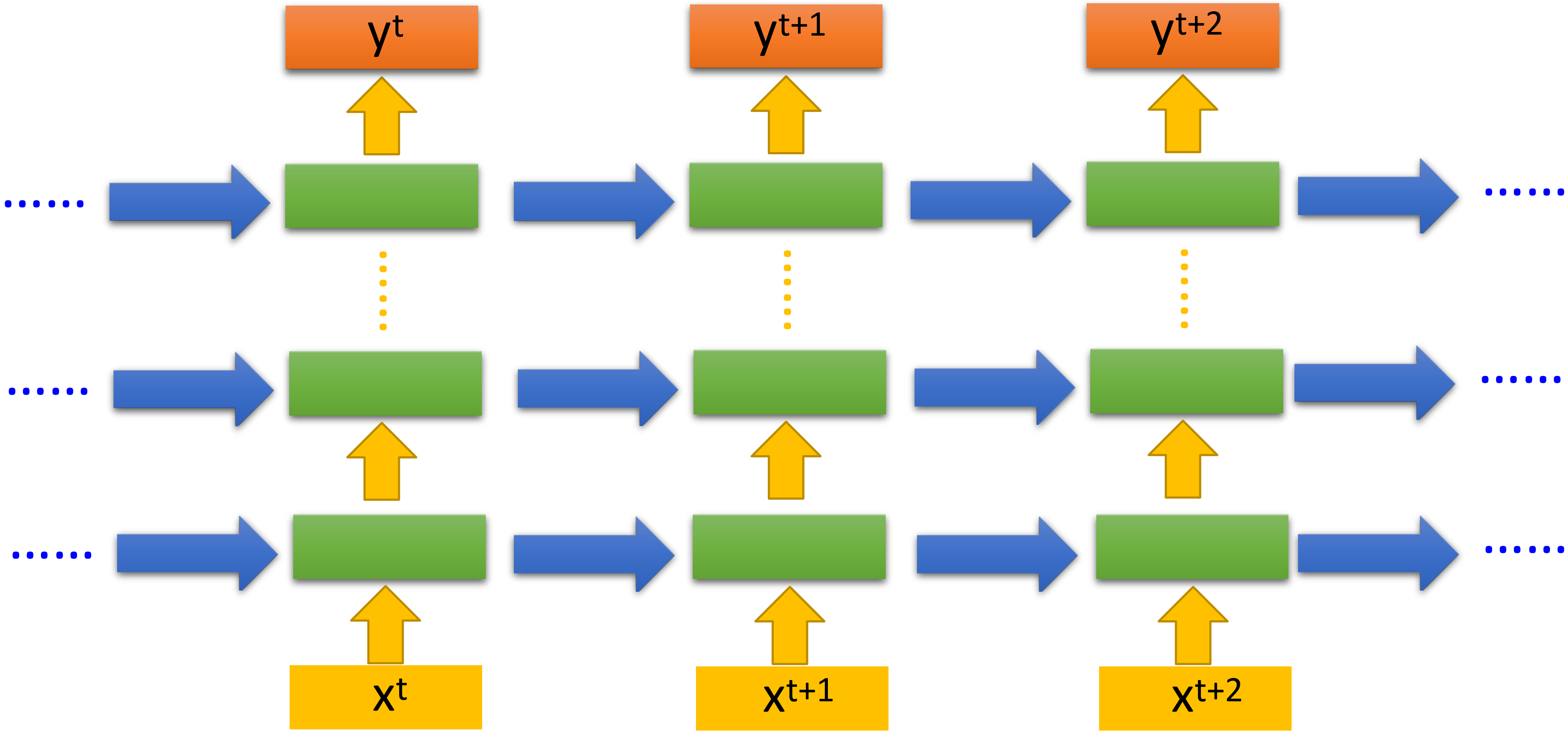
The same network is used again and again.



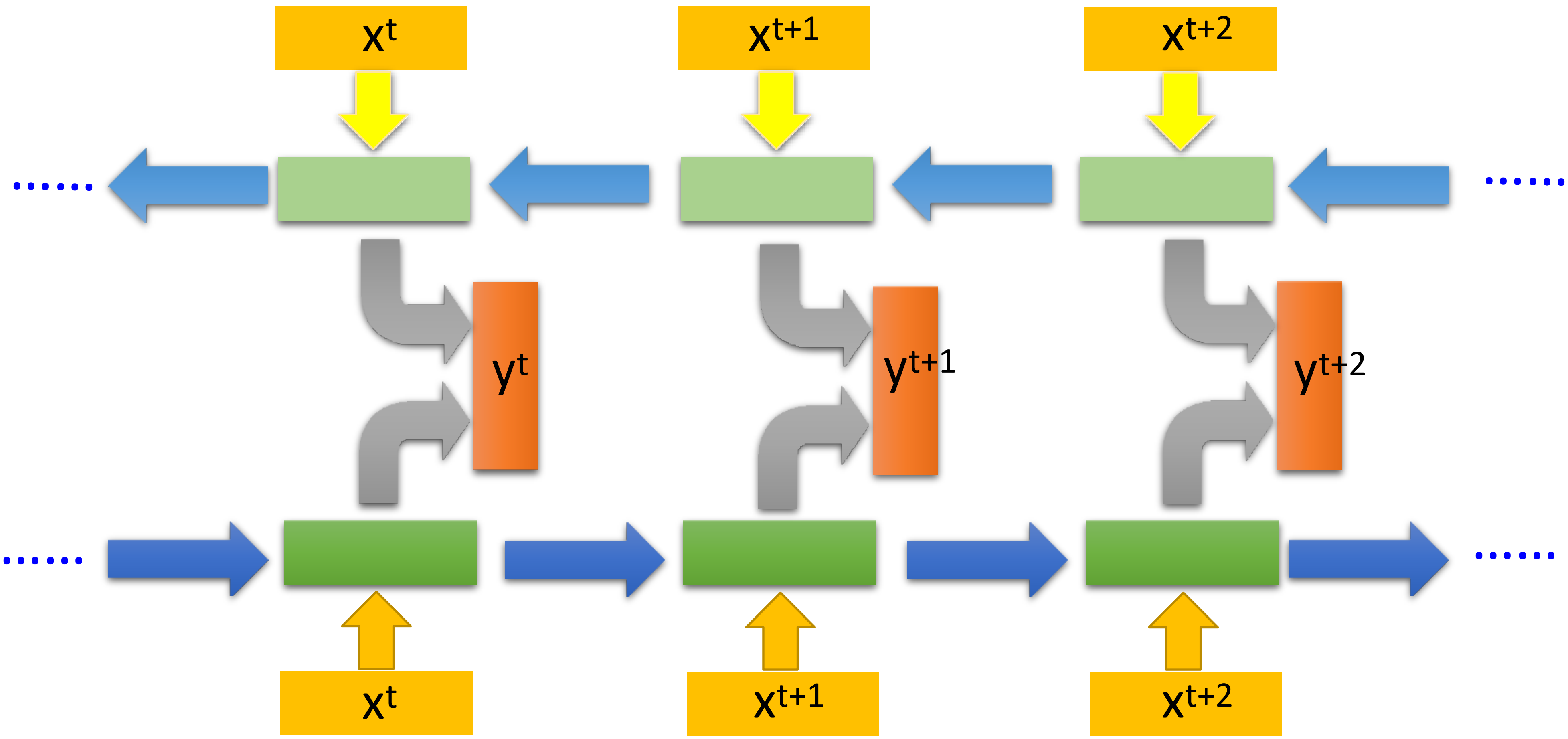
RNN


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
y^1
y^2
......
store
a^1
a^2
x^1
x^2
leave
Taipei
arrive
Taipei
The values stored in the memory is different.

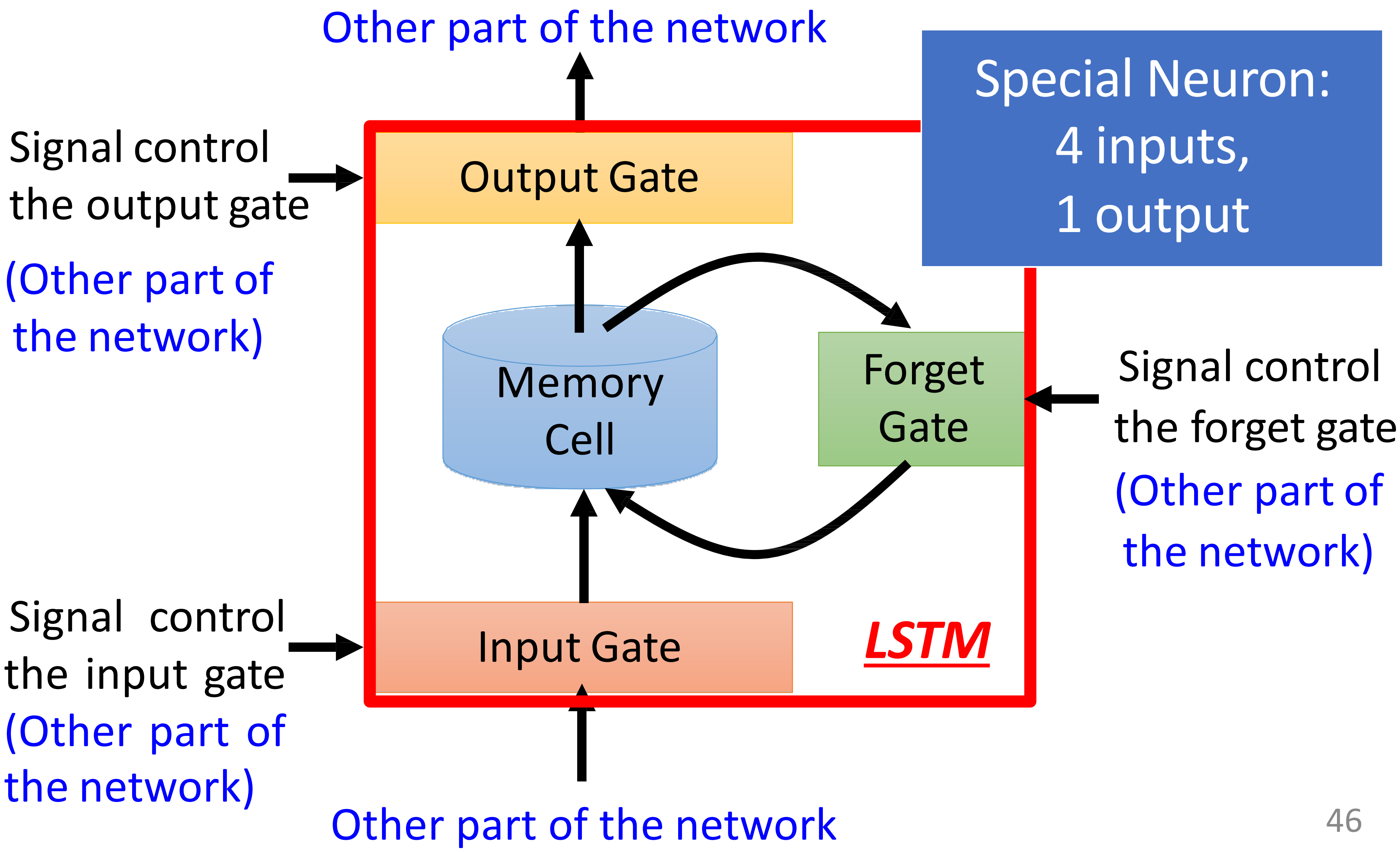
Of course it can be deep …

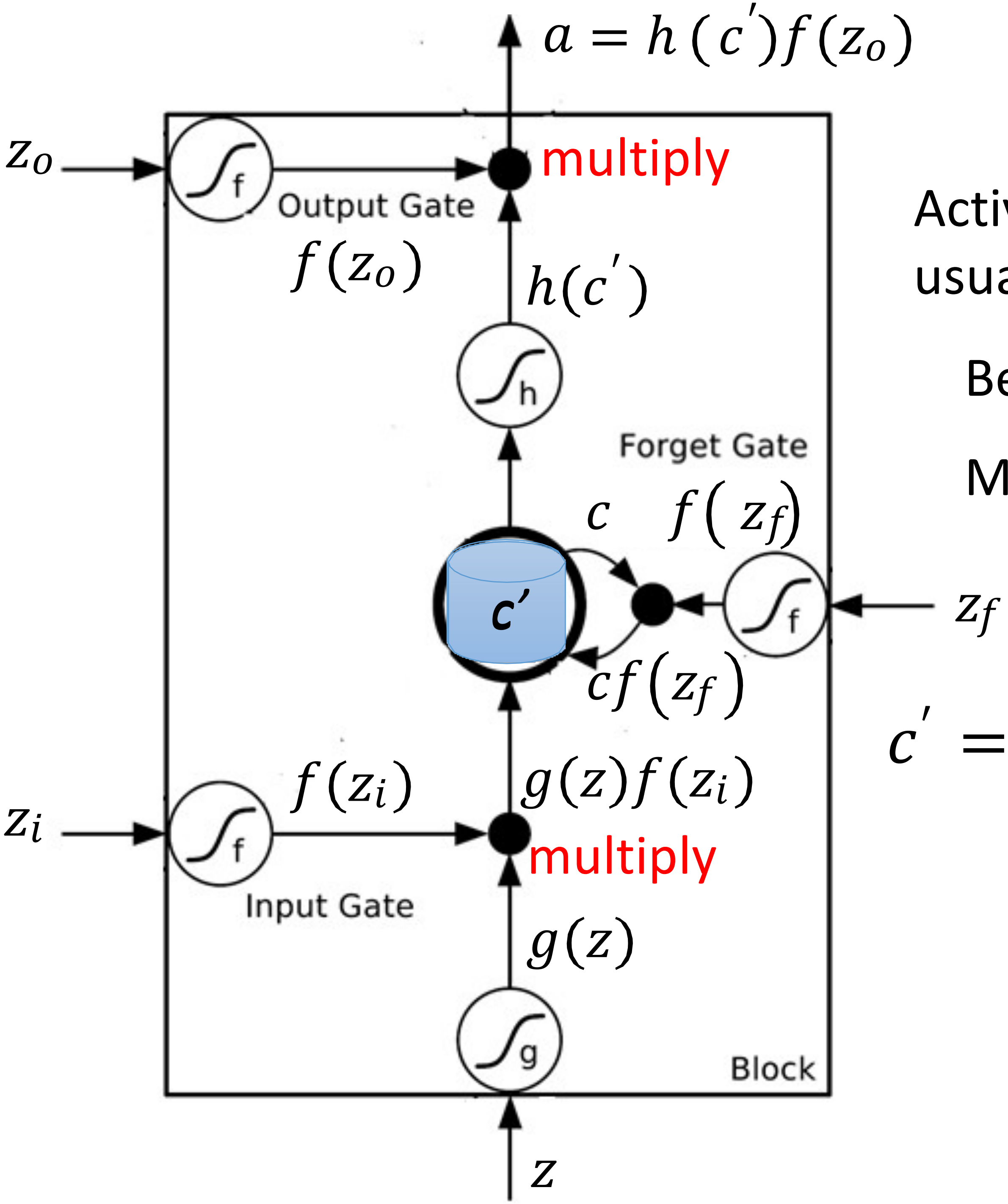


Bidirectional RNN



Long Short-term Memory (LSTM)



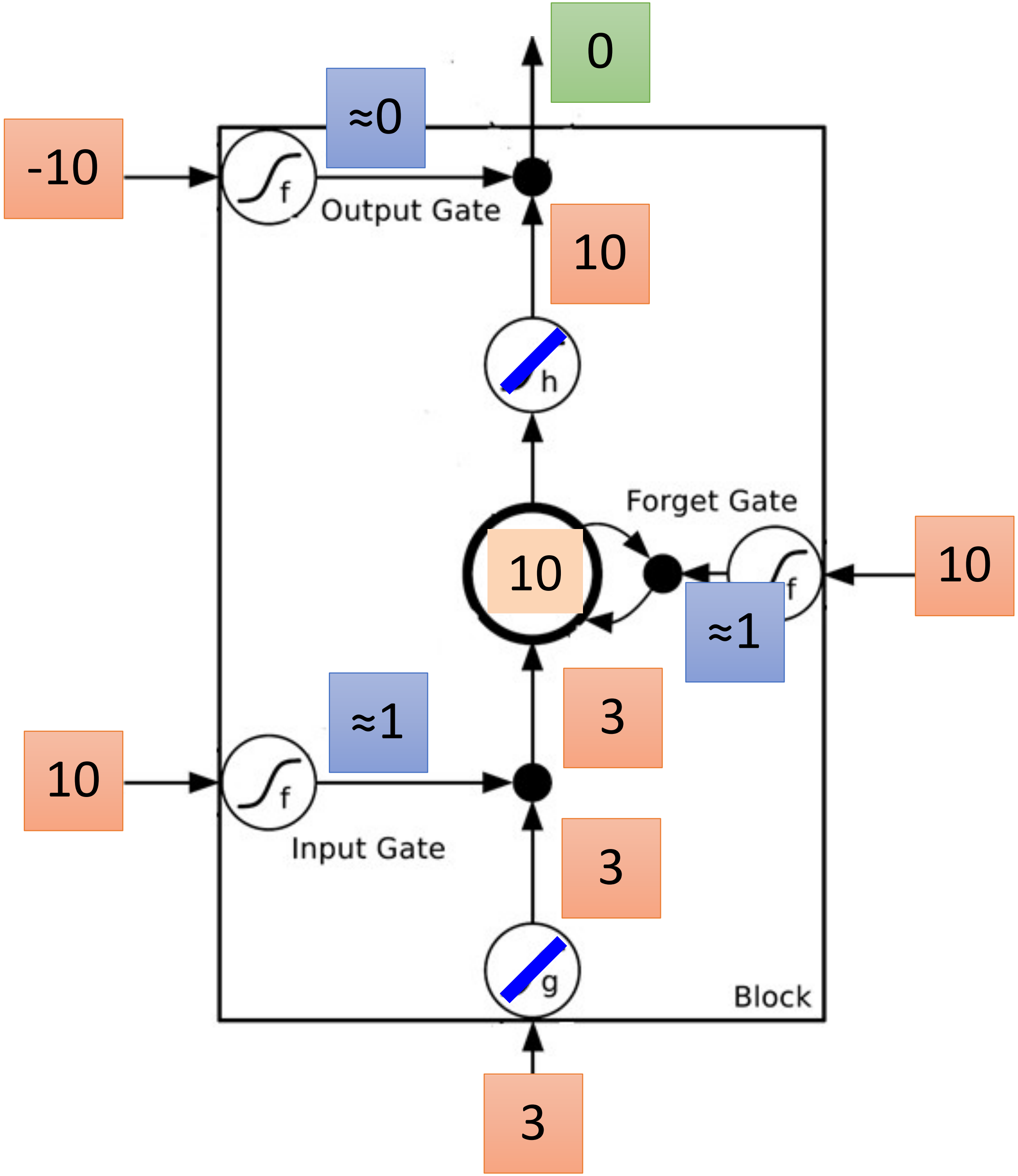


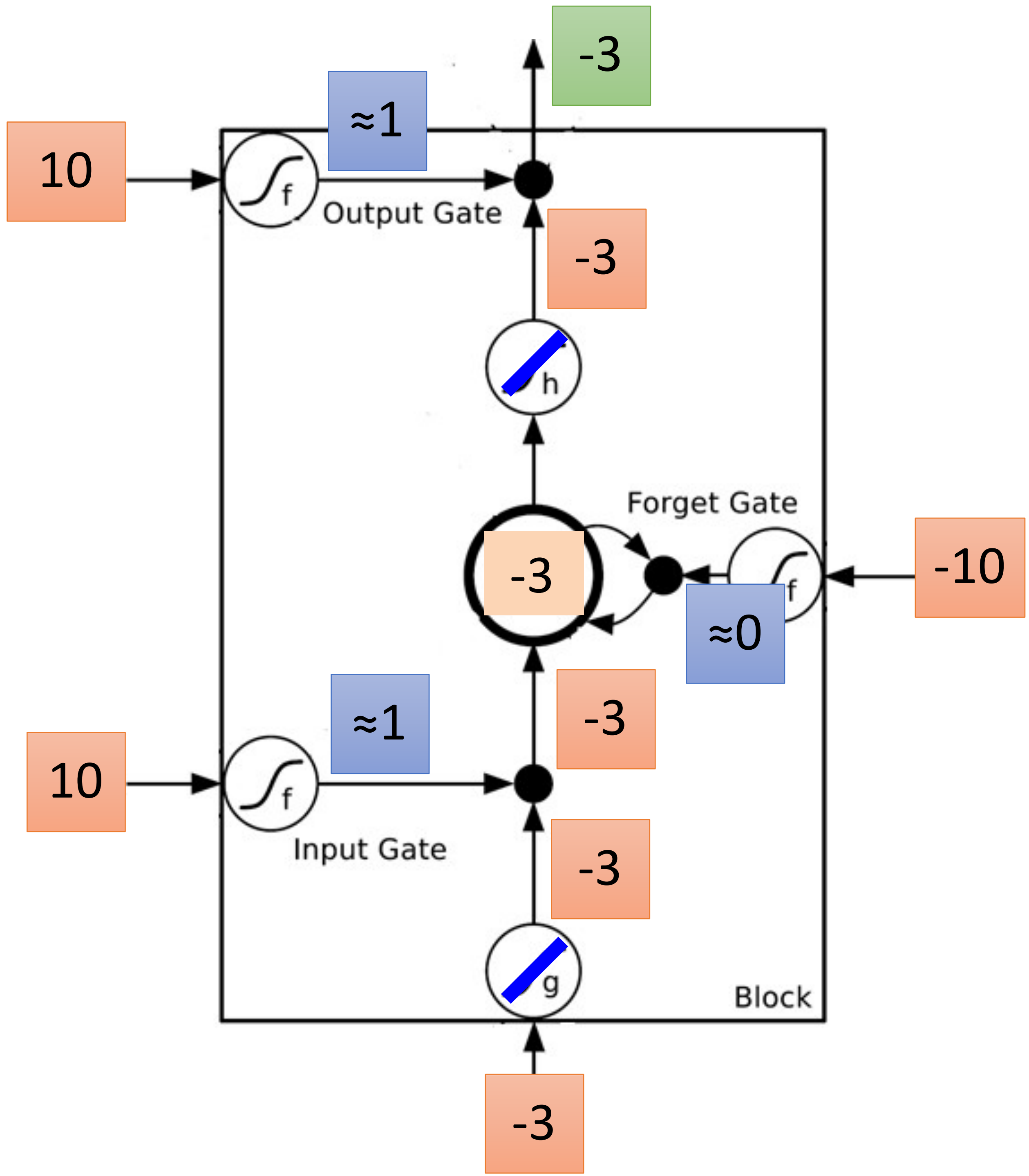
Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

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


0
≈0
-10
Output Gate
10
h
Forget Gate
10
10
≈1
≈1
10
3
3
Input Gate
g
Block
3

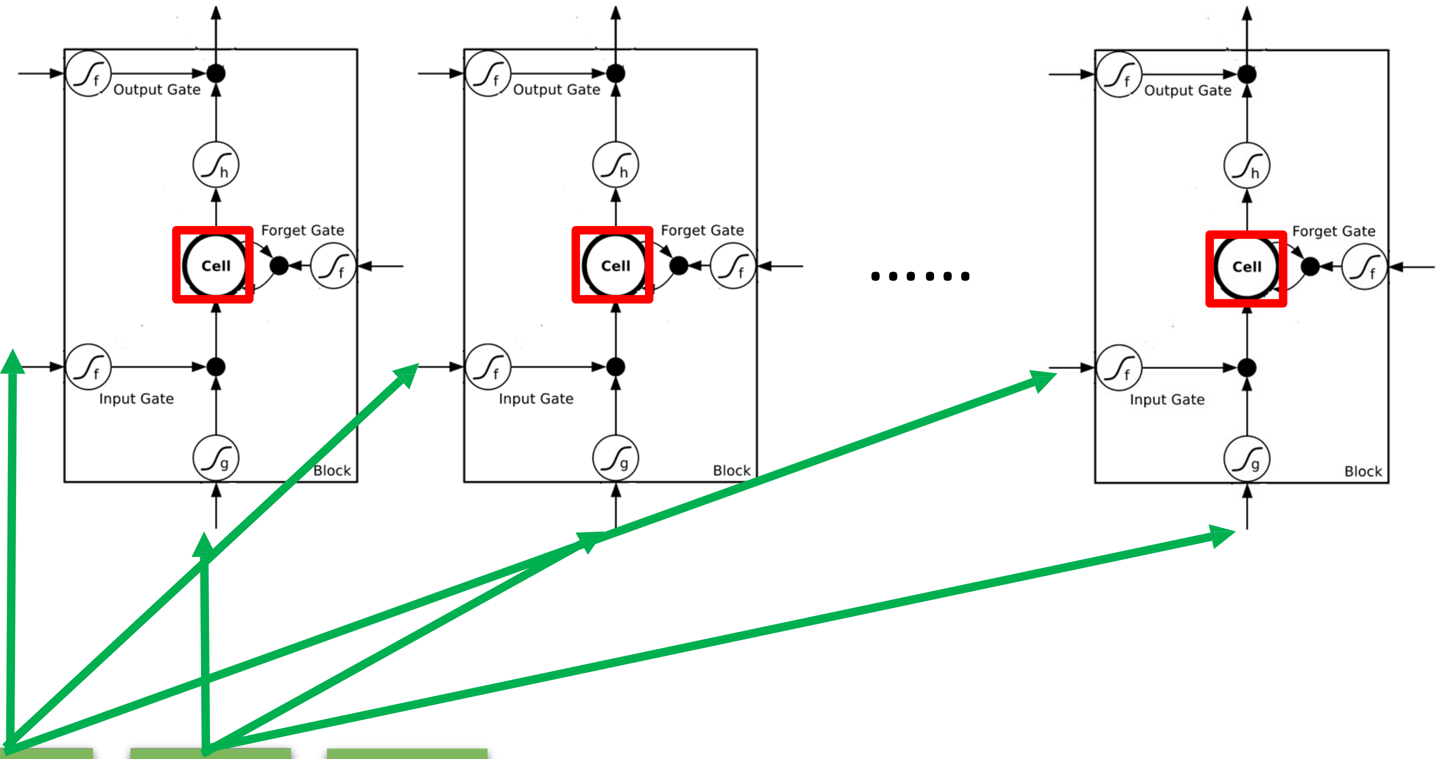


-3
≈1
10
Output Gate
-3
h
Forget Gate
-3
-10
≈0
≈1
-3
10
Input Gate
-3
g
Block
-3

LSTM

c^{t-1}

vector

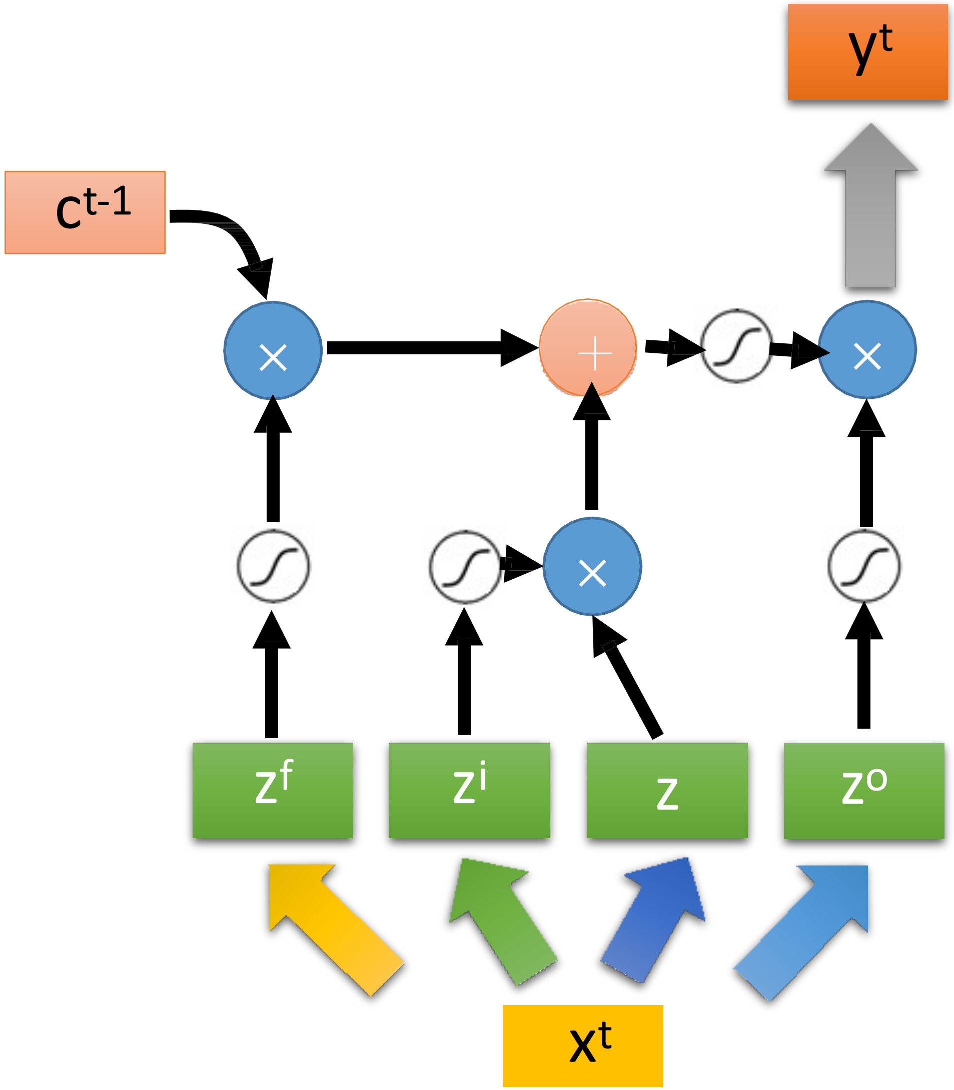


z^f z^i z z^o

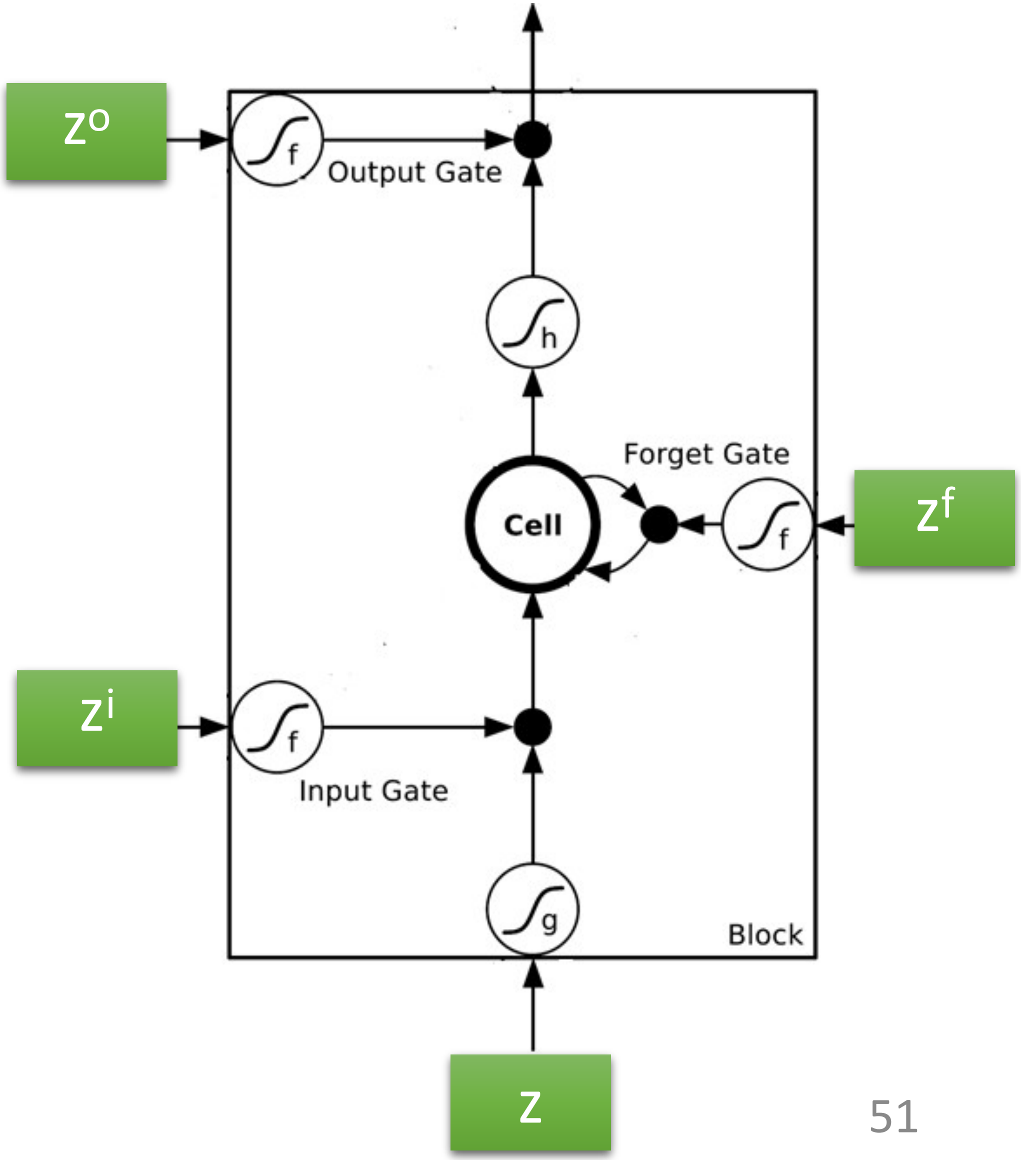
4 vectors

x^t

LSTM



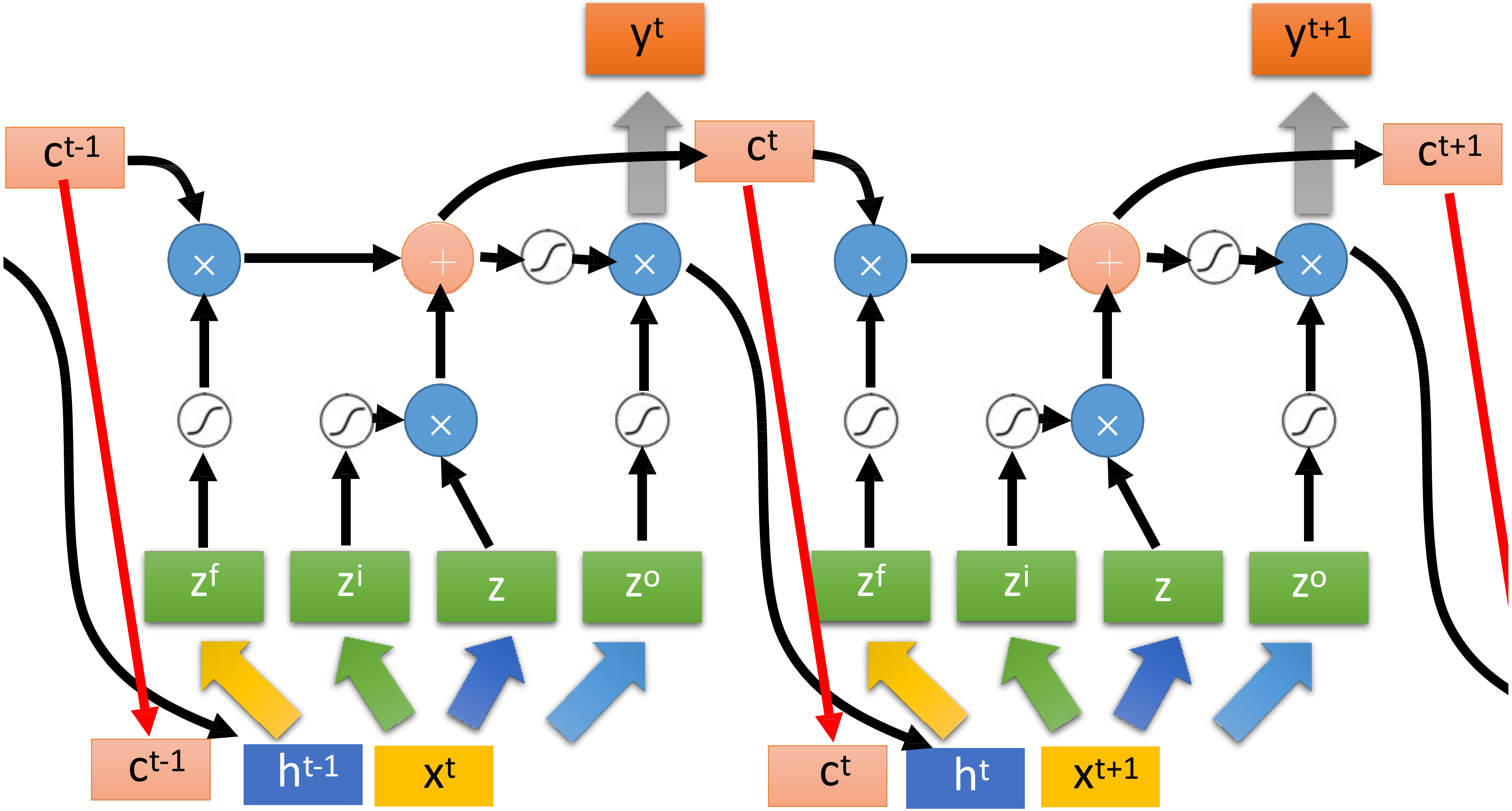
y^t
c^{t-1}
z^f
z^i
z
z^o
x^t



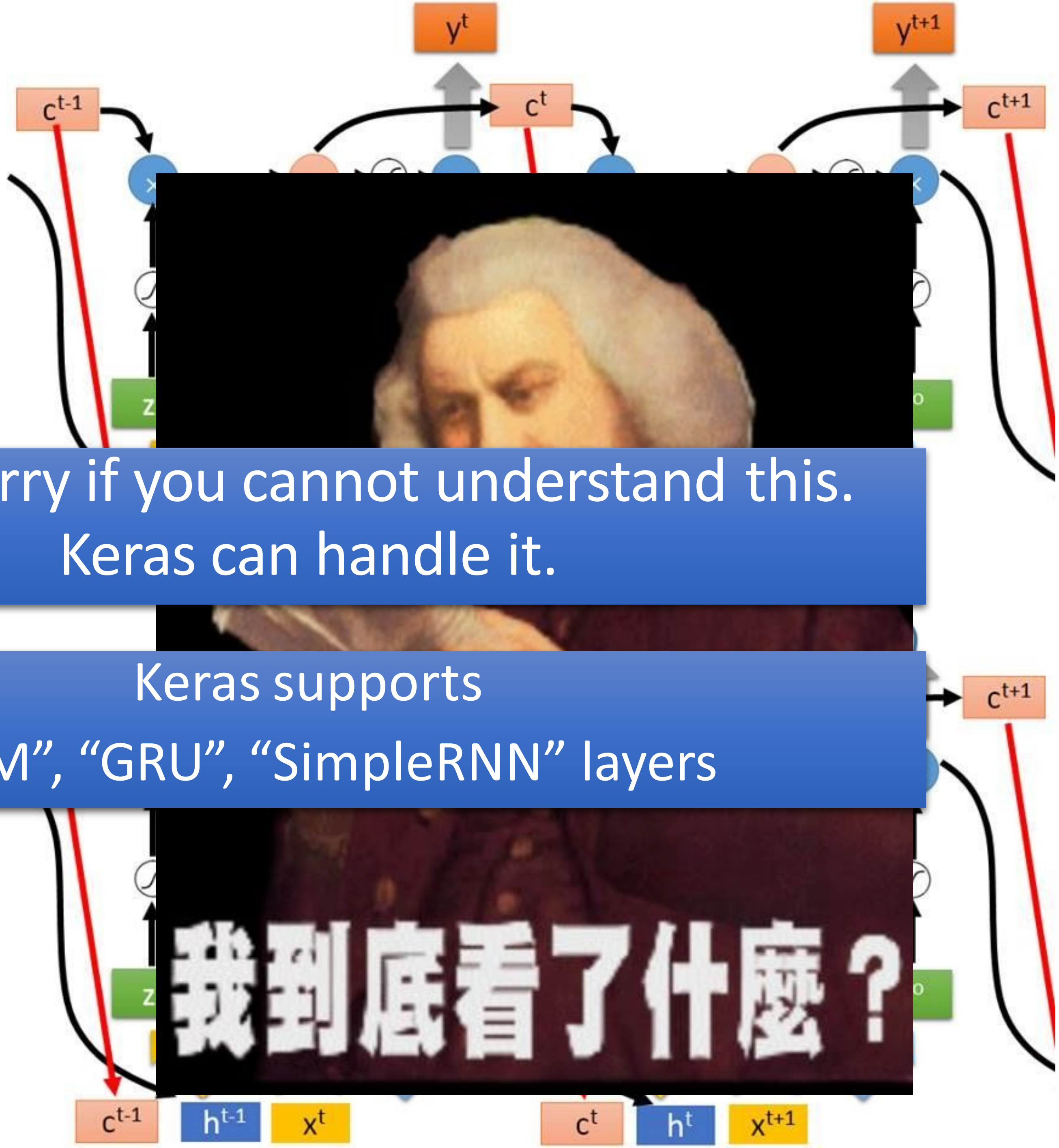
z^o
Output Gate
f
h
Forget Gate
Cell
f
z^f
z^i
f
Input Gate
g
Block
z

LSTM

Extension: "peephole"



Multiple-layer LSTM



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

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

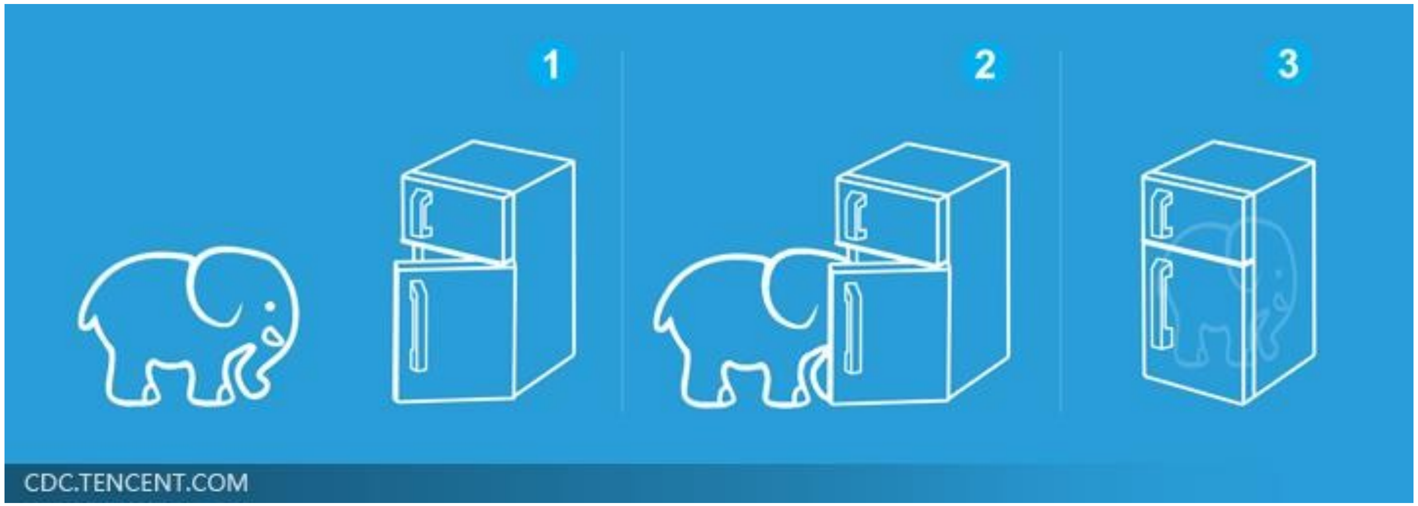
This is quite standard now.

https://img.komicolle.org/2015-09-20/src/1442696762731.gif

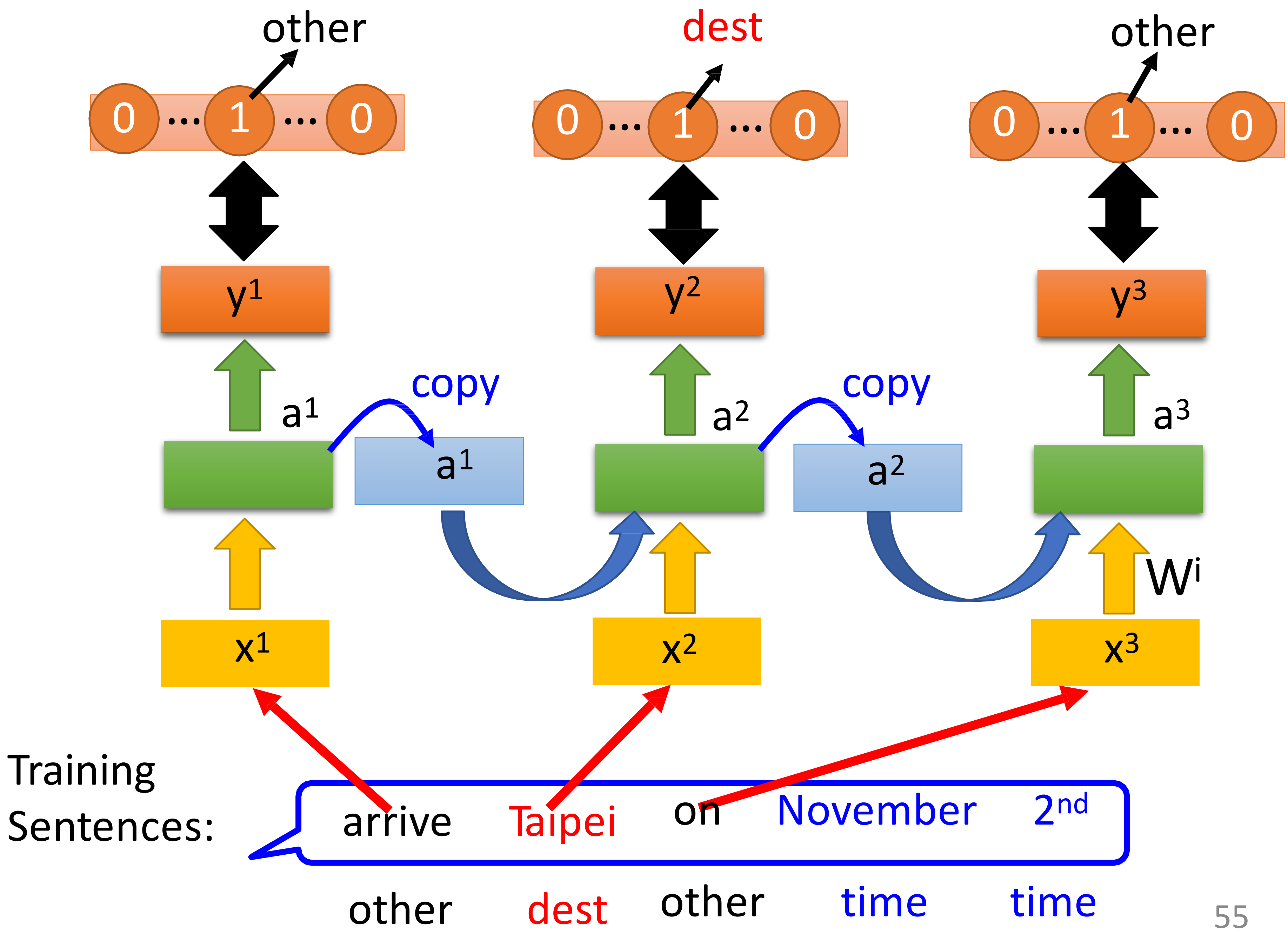
Three Steps for Deep Learning

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



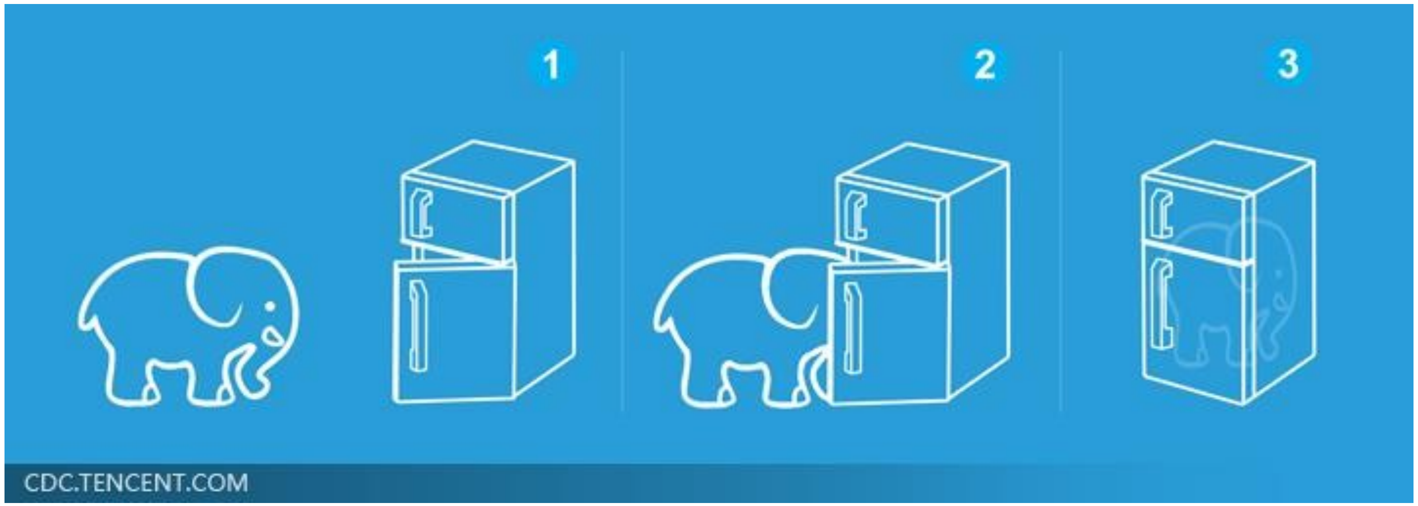
Learning Target



Three Steps for Deep Learning

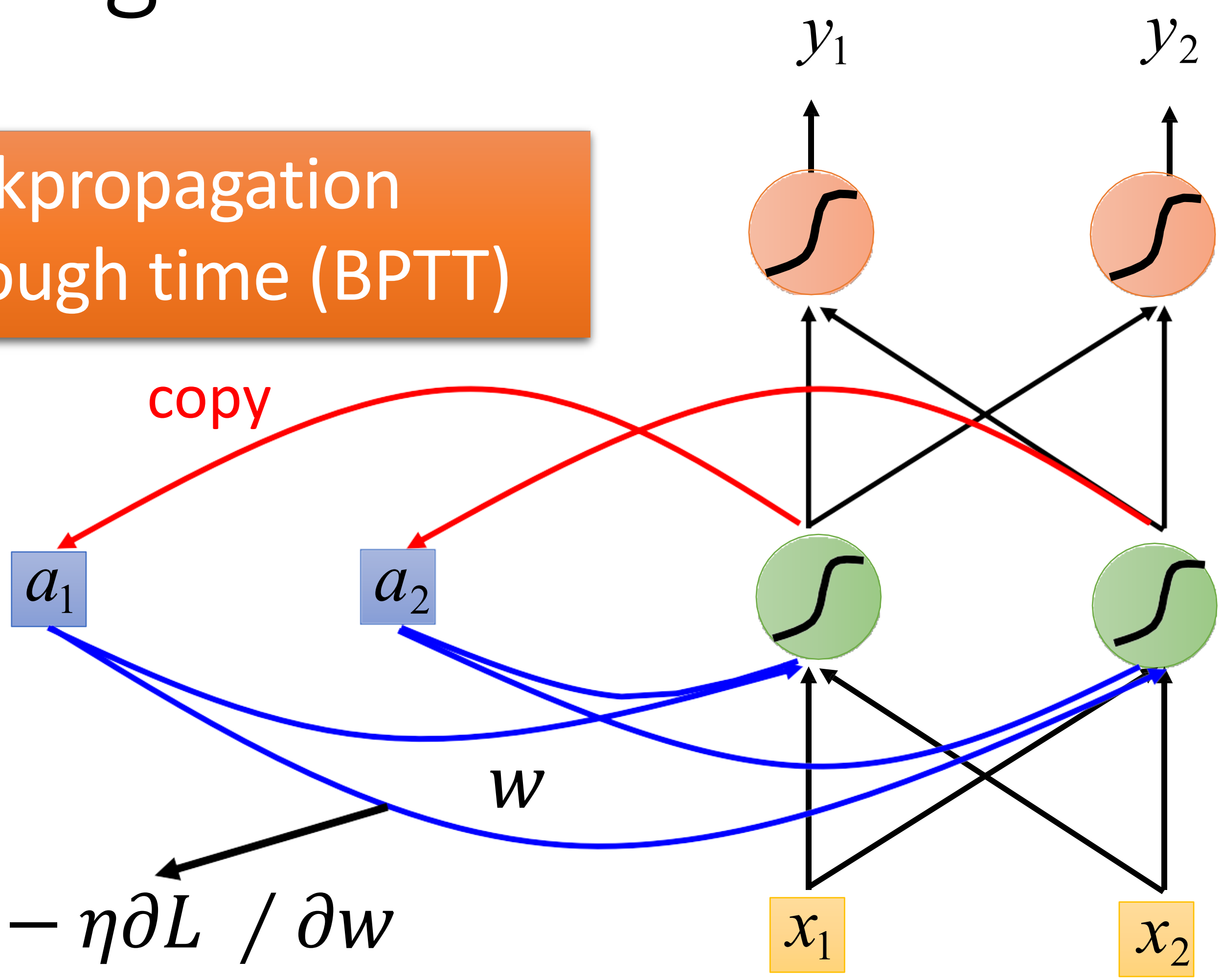
Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

Deep Learning is so simple ……



Learning

Backpropagation through time (BPTT)



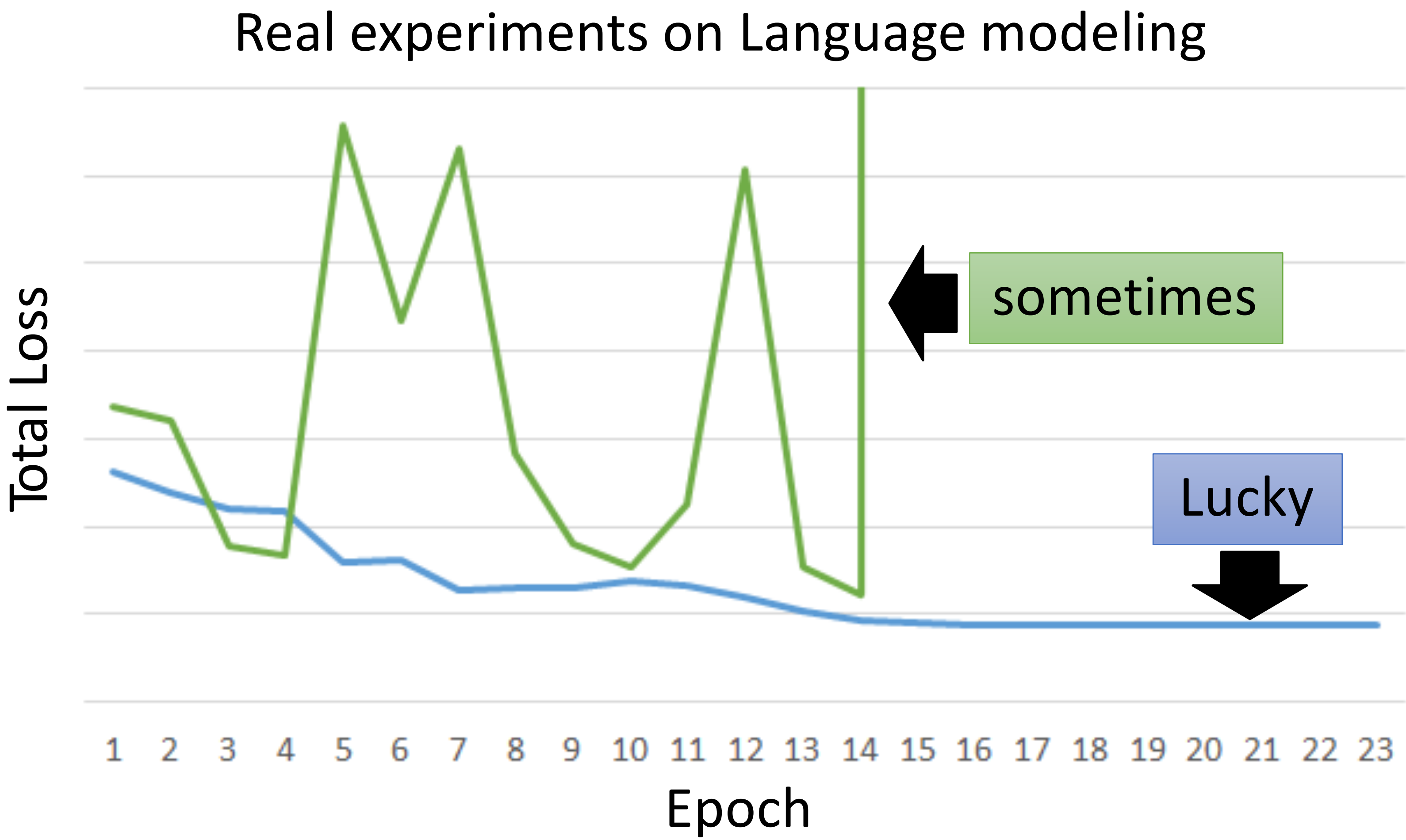
$$w \leftarrow w - \eta \partial L \ / \ \partial w$$

RNN Learning is very difficult in practice.

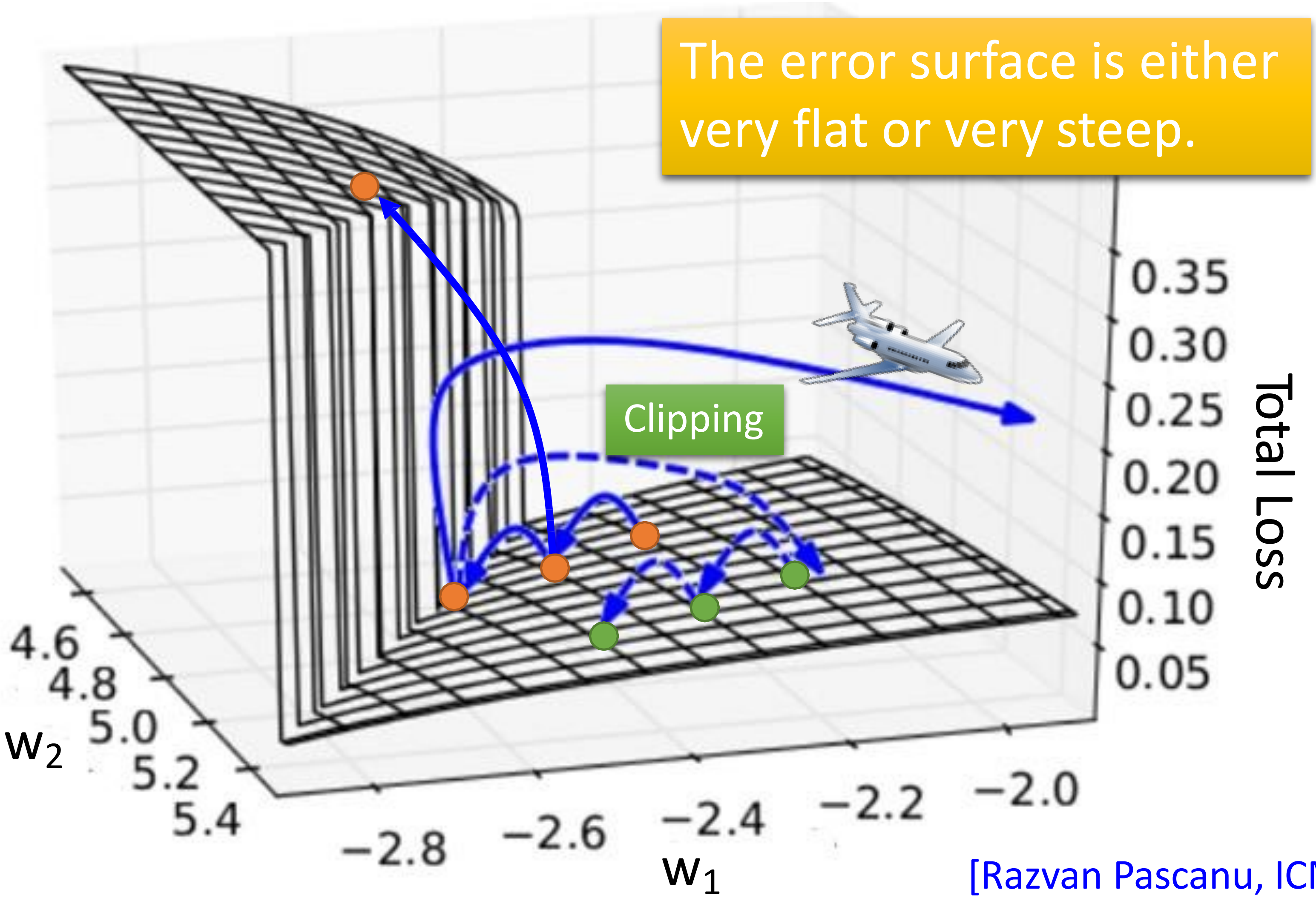
感謝 曾柏翔同學
提供實驗結果

Unfortunately ……

- RNN-based network is not always easy to learn



The error surface is rough.



[Razvan Pascanu, ICML'13]

Why?

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

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

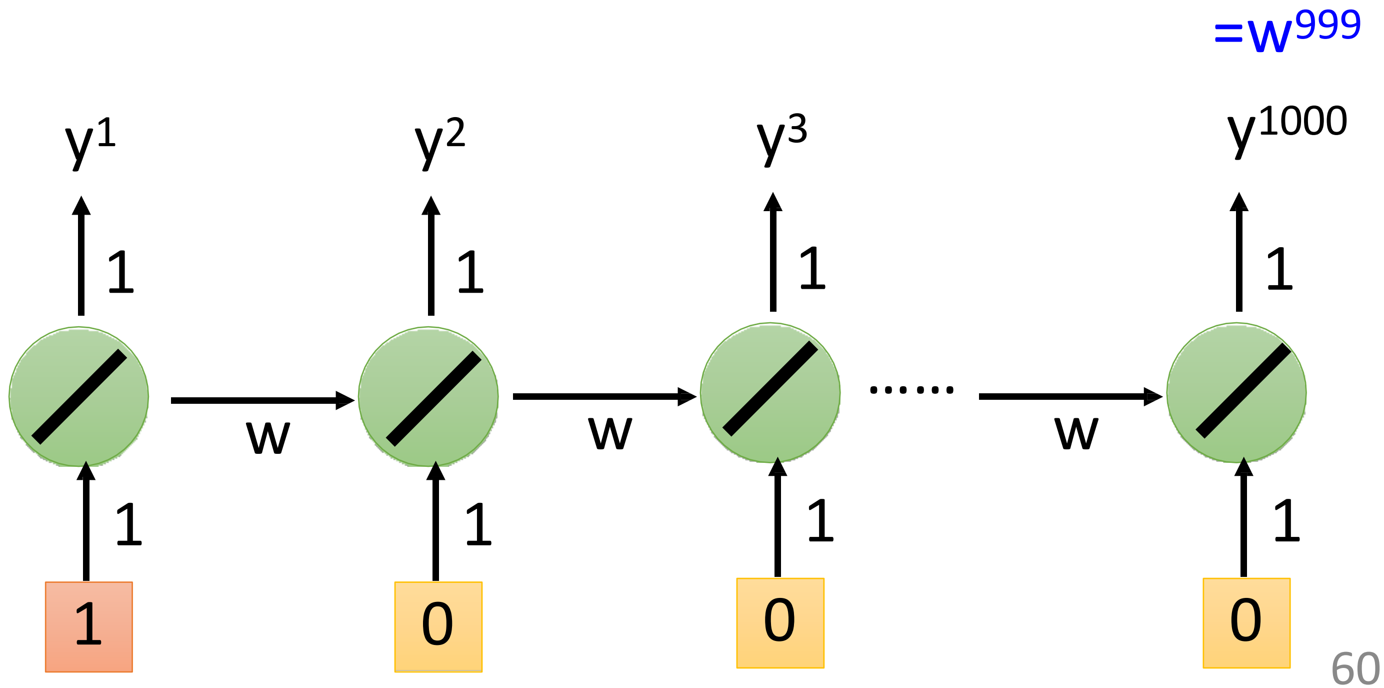
Large $\partial L / \partial w$ ⟹ Small Learning rate?

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

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

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

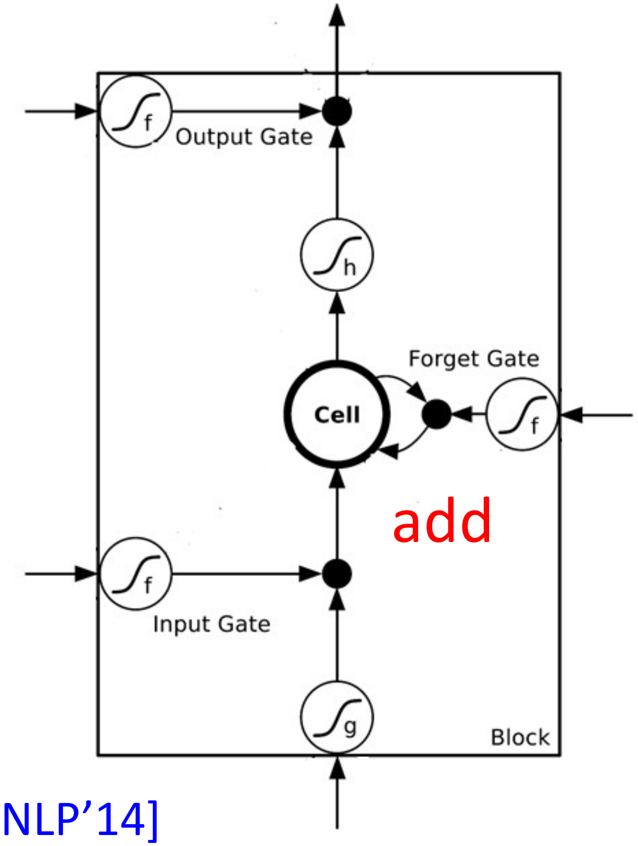
Toy Example



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.)

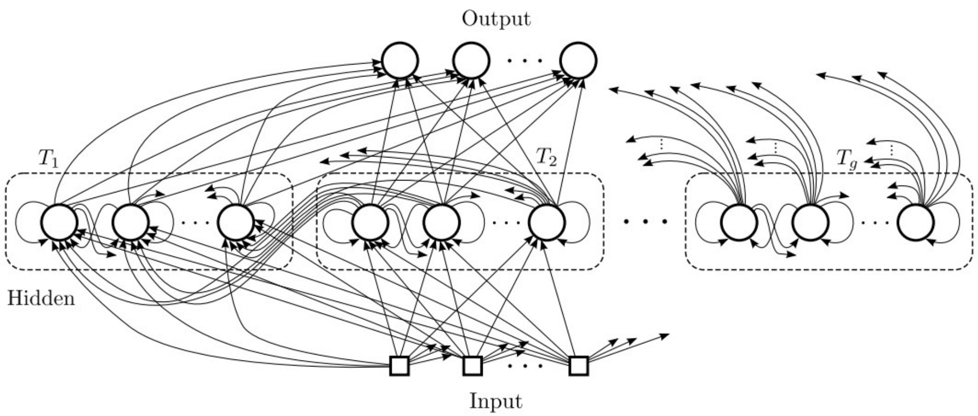


Gated Recurrent Unit (GRU): simpler than LSTM

[Cho, EMNLP'14]

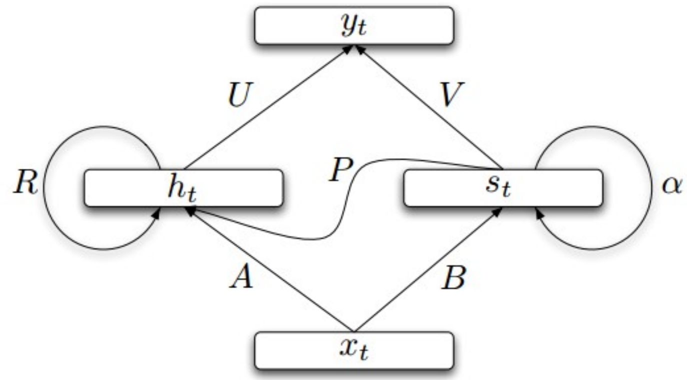
Helpful Techniques

Clockwise RNN



[Jan Koutnik, JMLR'14]

Structurally Constrained Recurrent Network (SCRN)



[Tomas Mikolov, ICLR'15]

Vanilla RNN Initialized with Identity matrix + ReLU activation function [Quoc V. Le, arXiv'15]

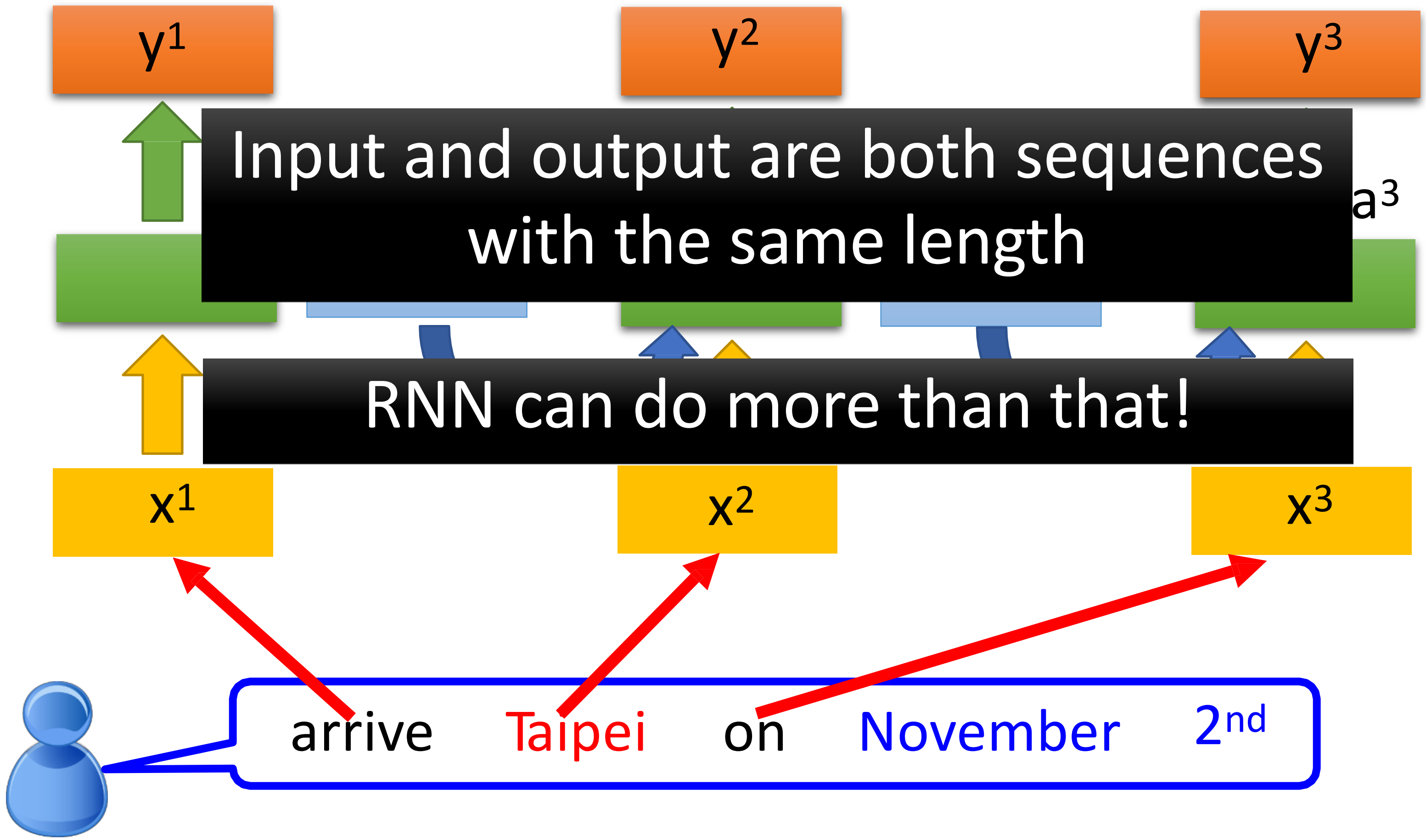
- ➢ Outperform or be comparable with LSTM in 4 different tasks

More Applications ……

Probability of "arrive" in each slot

Probability of "Taipei" in each slot

Probability of "on" in each slot

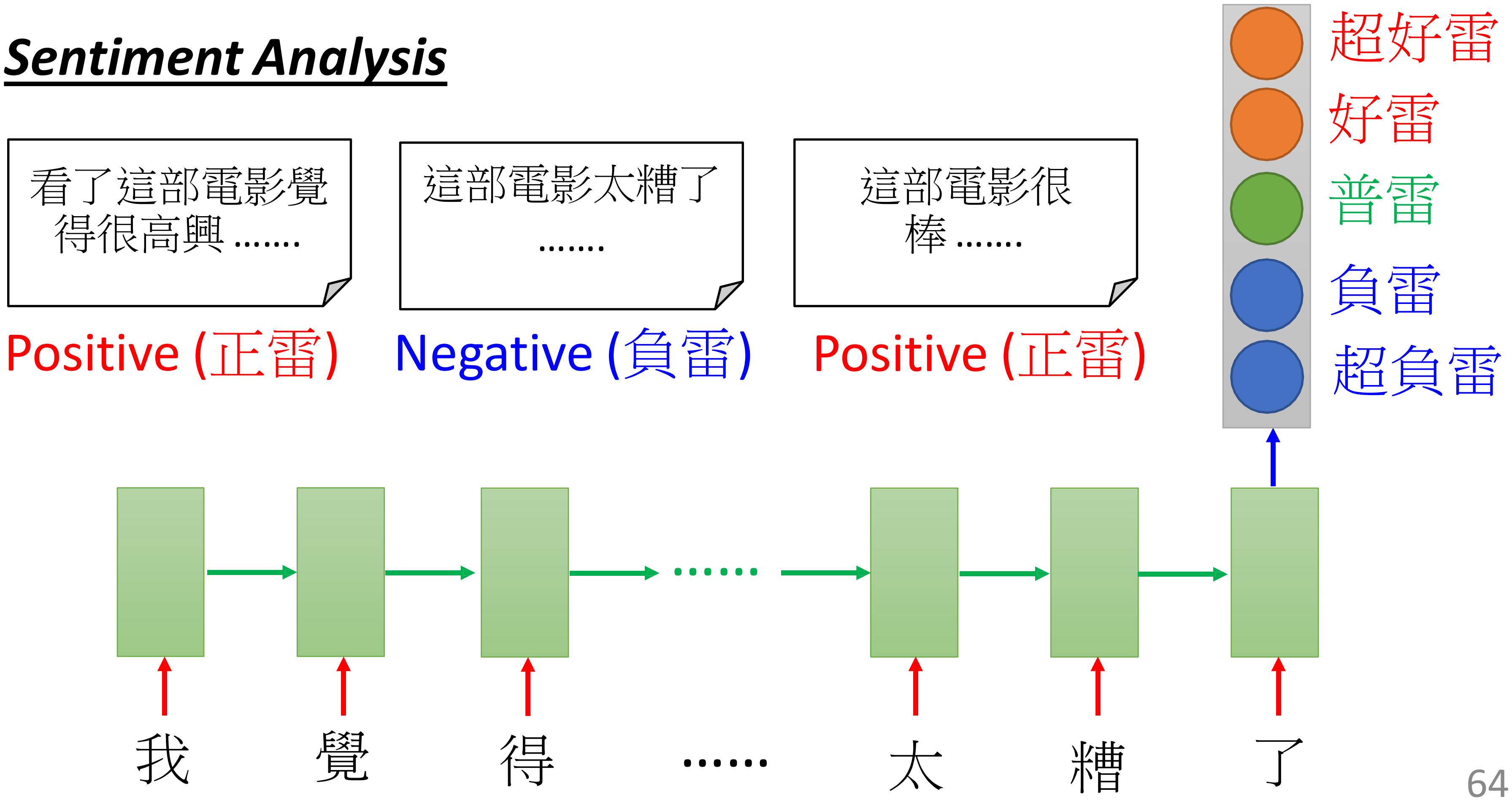


Many to one

Keras Example: https://github.com/fchollet/keras/blob/master/examples/imdb_lstm.py

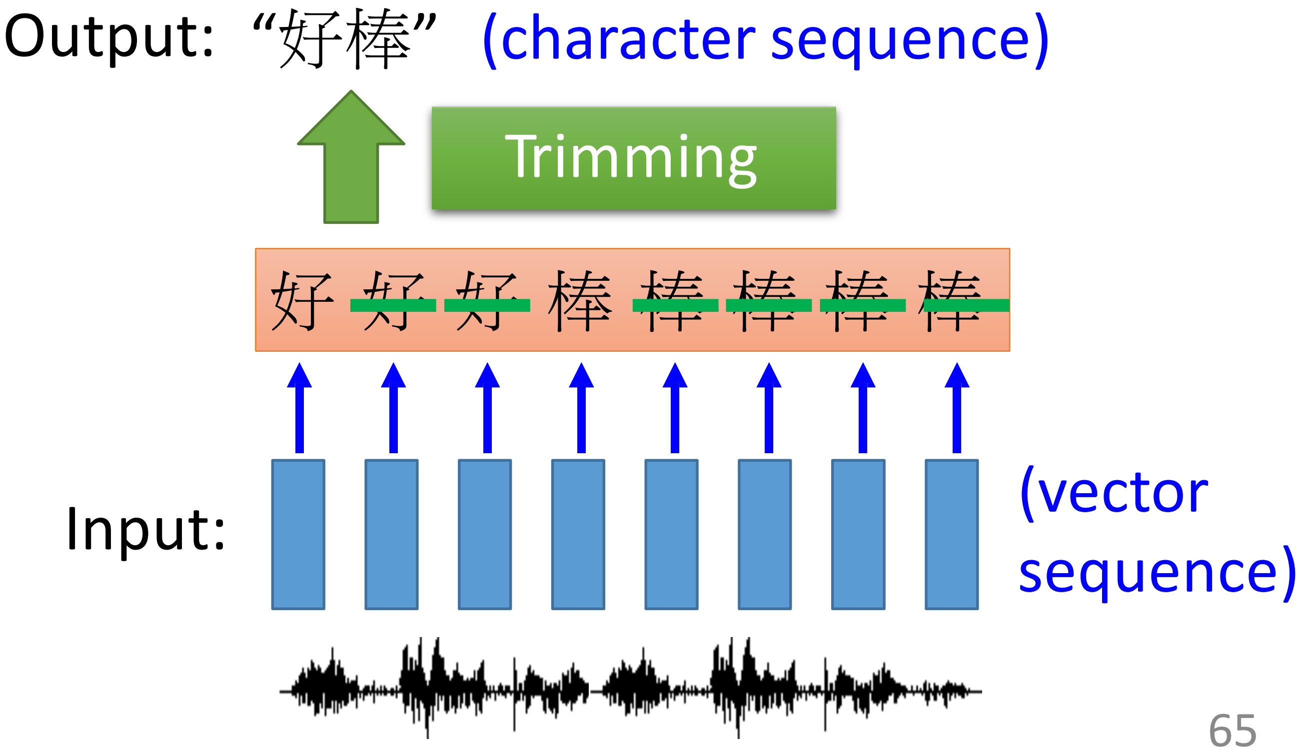
- Input is a vector sequence, but output is only one vector

Sentiment Analysis



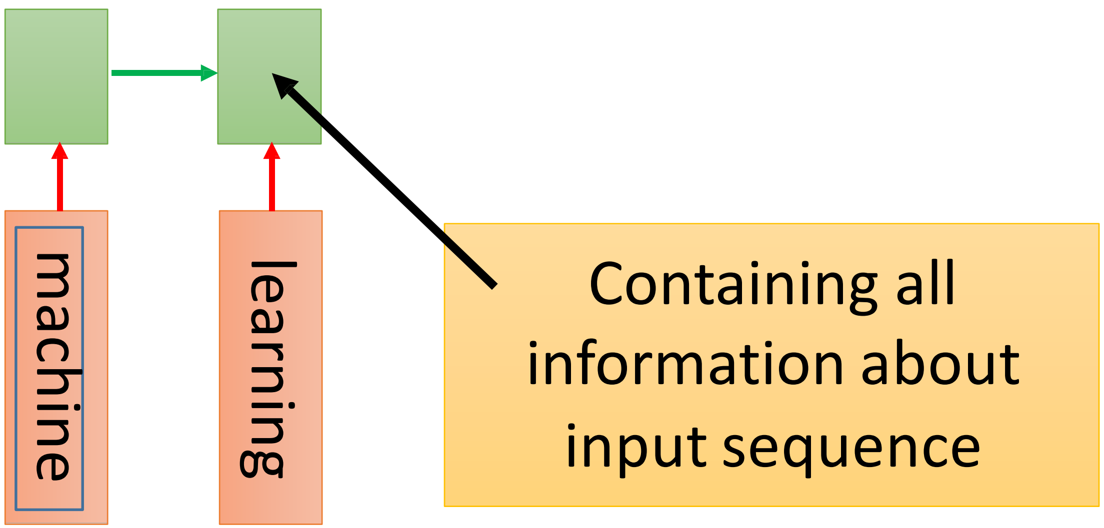
Many to Many (Output is shorter)

- Both input and output are both sequences, ***but the output is shorter.***
  - E.g. ***Speech Recognition***



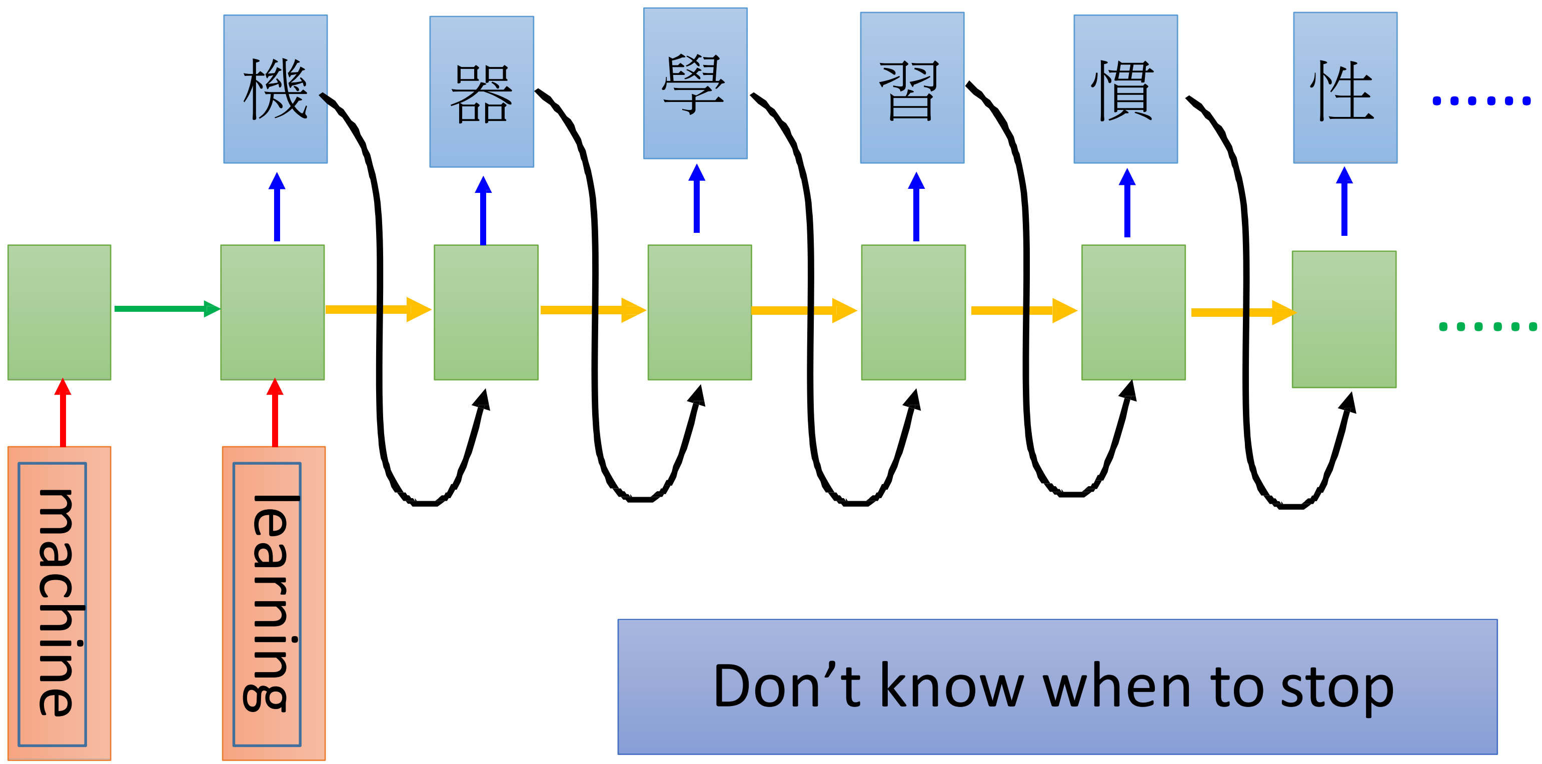
Many to Many (No Limitation)

- Both input and output are both sequences ***with different lengths***. → ***Sequence to sequence learning***
  - E.g. ***Machine Translation*** (machine learning→機器學習)



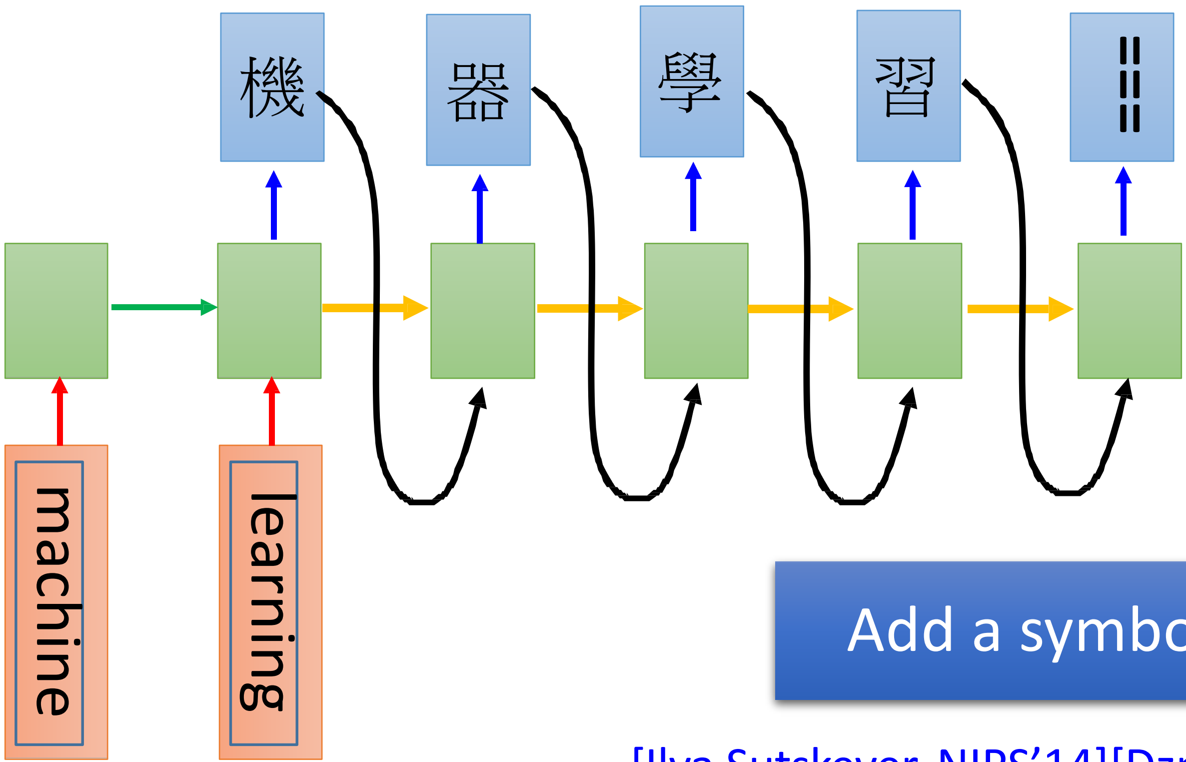
Many to Many (No Limitation)

- Both input and output are both sequences ***with different lengths***. → ***Sequence to sequence learning***
  - E.g. ***Machine Translation*** (machine learning→機器學習)



Many to Many (No Limitation)

- Both input and output are both sequences ***with different lengths***. → ***Sequence to sequence learning***
  - E.g. ***Machine Translation*** (machine learning→機器學習)



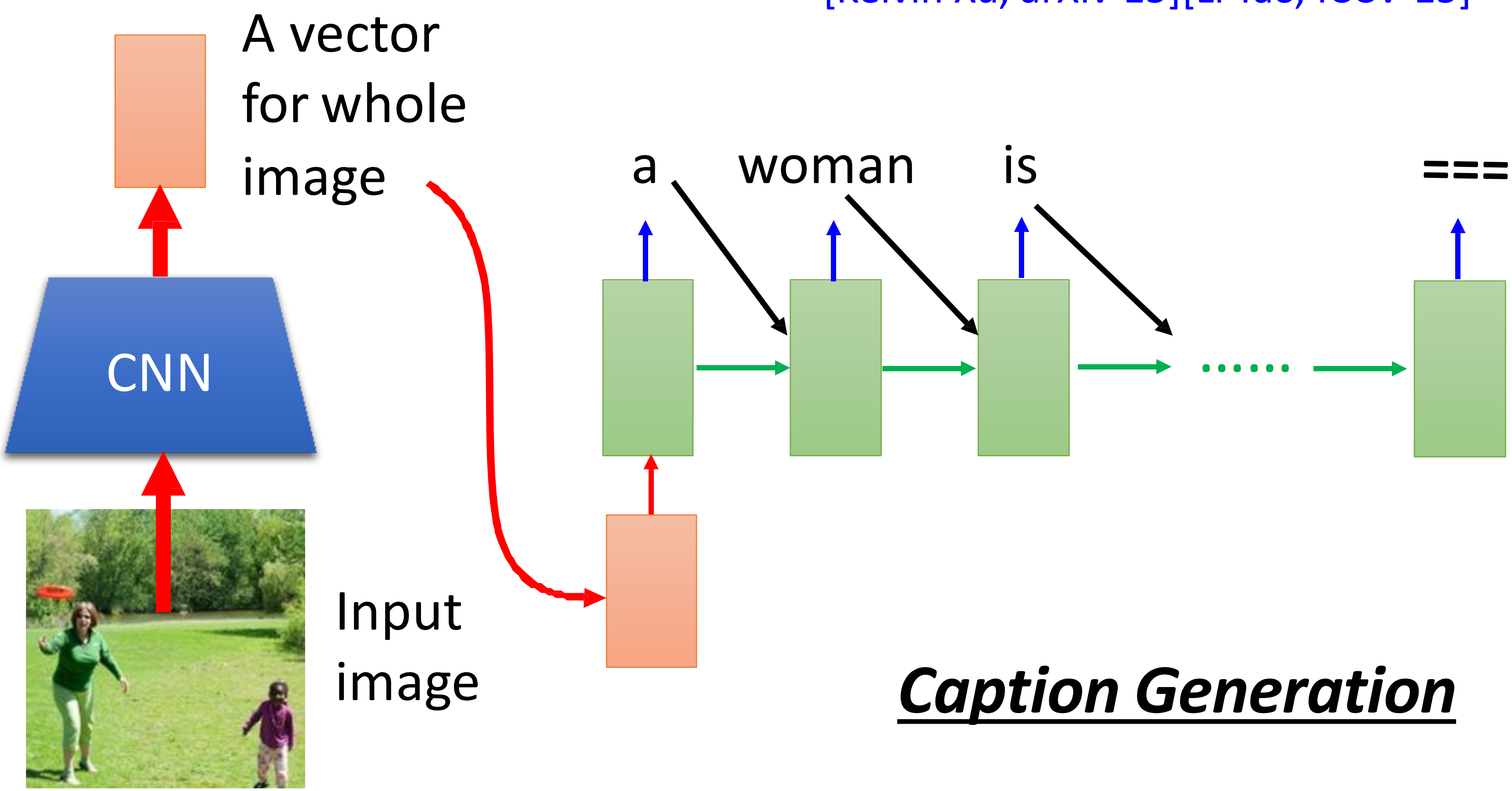
Add a symbol "===" (斷)

[Ilya Sutskever, NIPS'14][Dzmitry Bahdanau, arXiv'15]

One to Many

- Input an image, but output a sequence of words

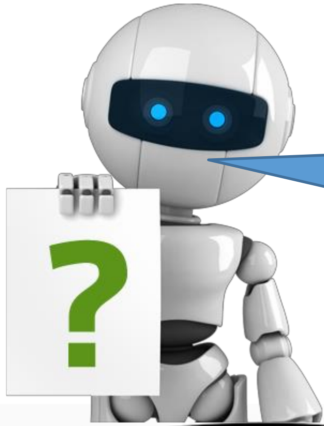
[Kelvin Xu, arXiv'15][Li Yao, ICCV'15]



<u>Caption Generation</u>

Application: Video Caption Generation

Video



A girl is running.

A group of people is knocked by a tree.

A group of people is walking in the forest.