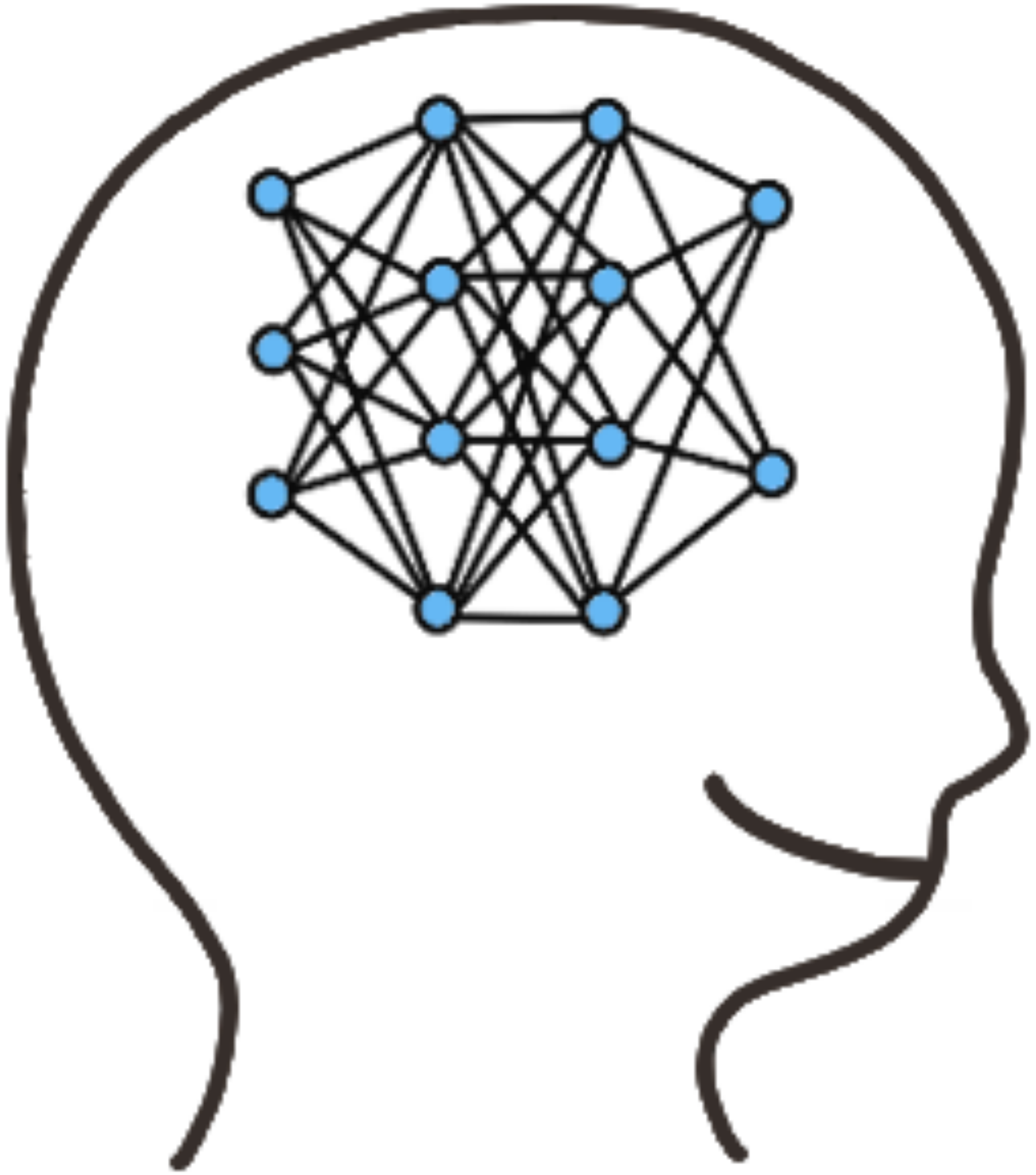
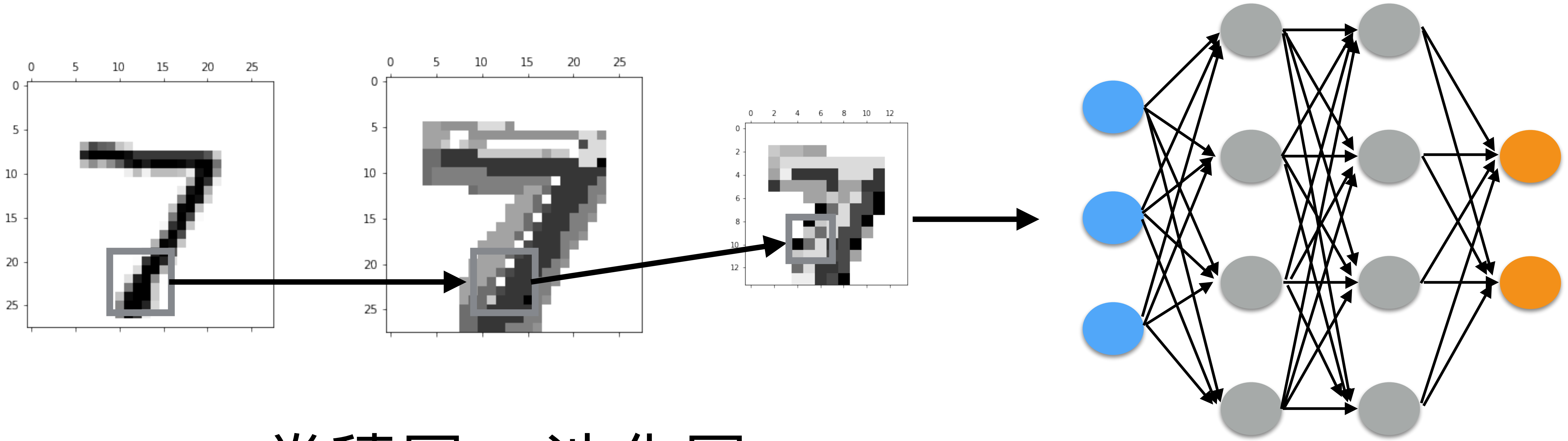


卷積神經網路

Convolutional Neural Network

卷積神經網路 (CNN)



卷積層 + 池化層
Convolution + Pooling layers

MLP

Hint

- MNIST 問題用CNN解可達99%準確率。

卷積層 (Convolution)

$$
\underset{\text{pixels}}{\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 6 & 0 \\ 3 & 3 & 2 & 0 \\ 1 & 0 & 2 & 1 \end{bmatrix}} \otimes \underset{\text{濾鏡}}{\begin{bmatrix} 2 & 2 & 1 \\ 1 & 0 & 3 \\ 3 & 0 & 2 \end{bmatrix}} = \begin{bmatrix} 19 & a_{12} \\ a_{21} & a_{22} \end{bmatrix}
$$

$$
\begin{aligned}
a_{11} &= (0\times2+0\times1+1\times3)+(0\times2+1\times0+6\times0)+(3\times1+3\times3+2\times2) \\
&= 3+0+16 \\
&= 19
\end{aligned}
$$

卷積層 (Convolution)

pixels

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

$\otimes$

濾鏡

2	2	1
1	0	3
3	0	2

=

19	13
a_{21}	a_{22}

步幅(stride)=1

卷積層 (Convolution)

pixels

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

$\otimes$

濾鏡

2	2	1
1	0	3
3	0	2

=

19	13
24	a_{22}

卷積層 (Convolution)

pixels

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

⊗

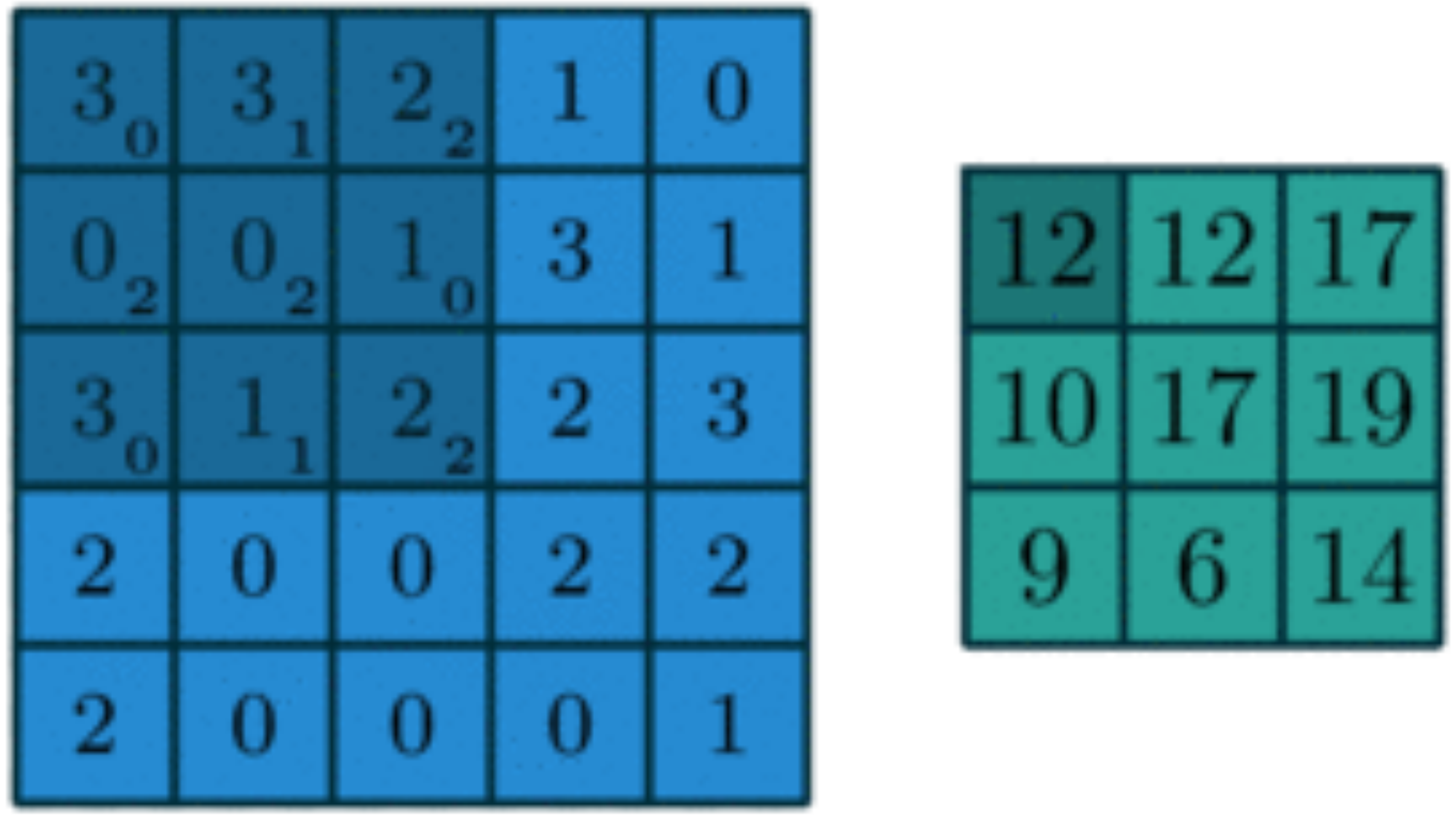
濾鏡

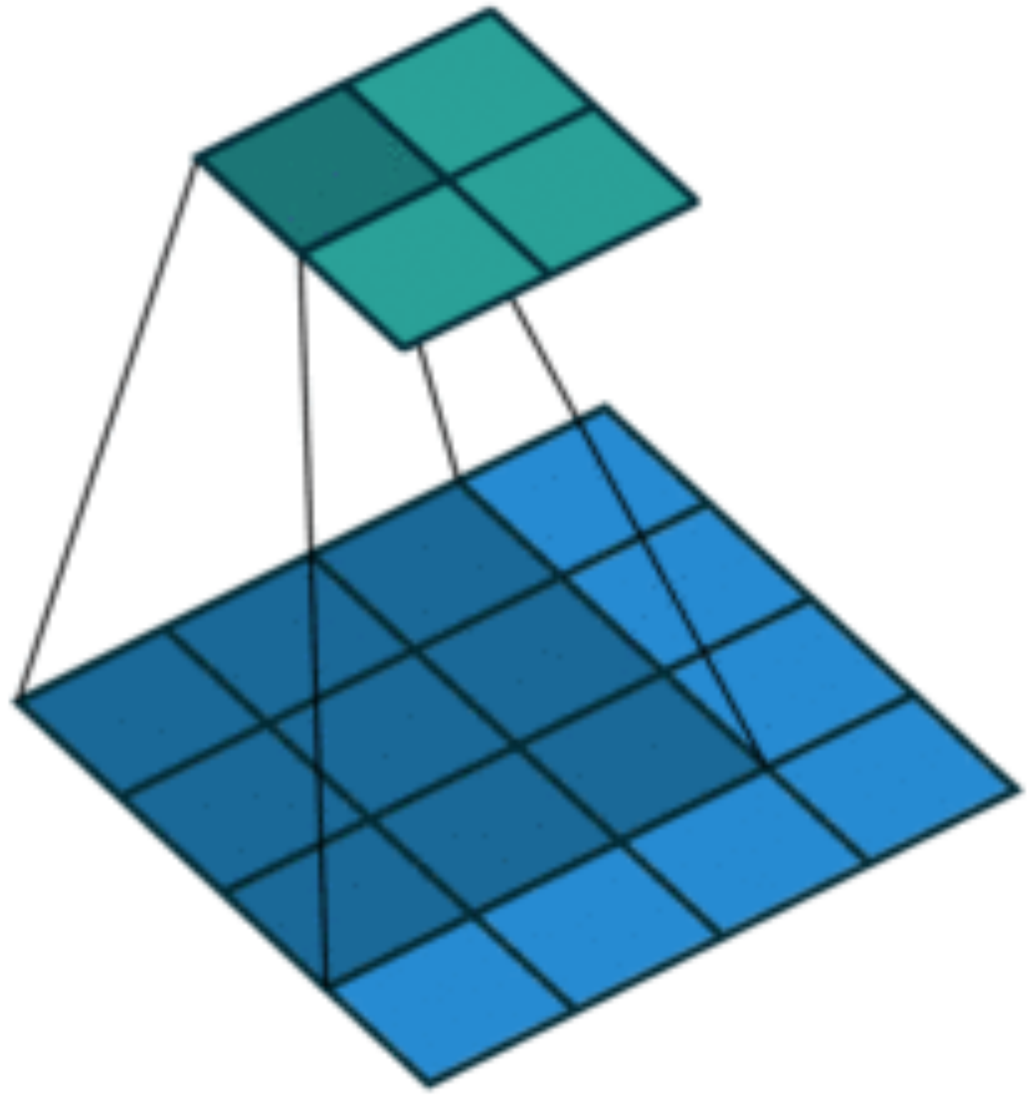
2	2	1
1	0	3
3	0	2

=

19	13
24	19

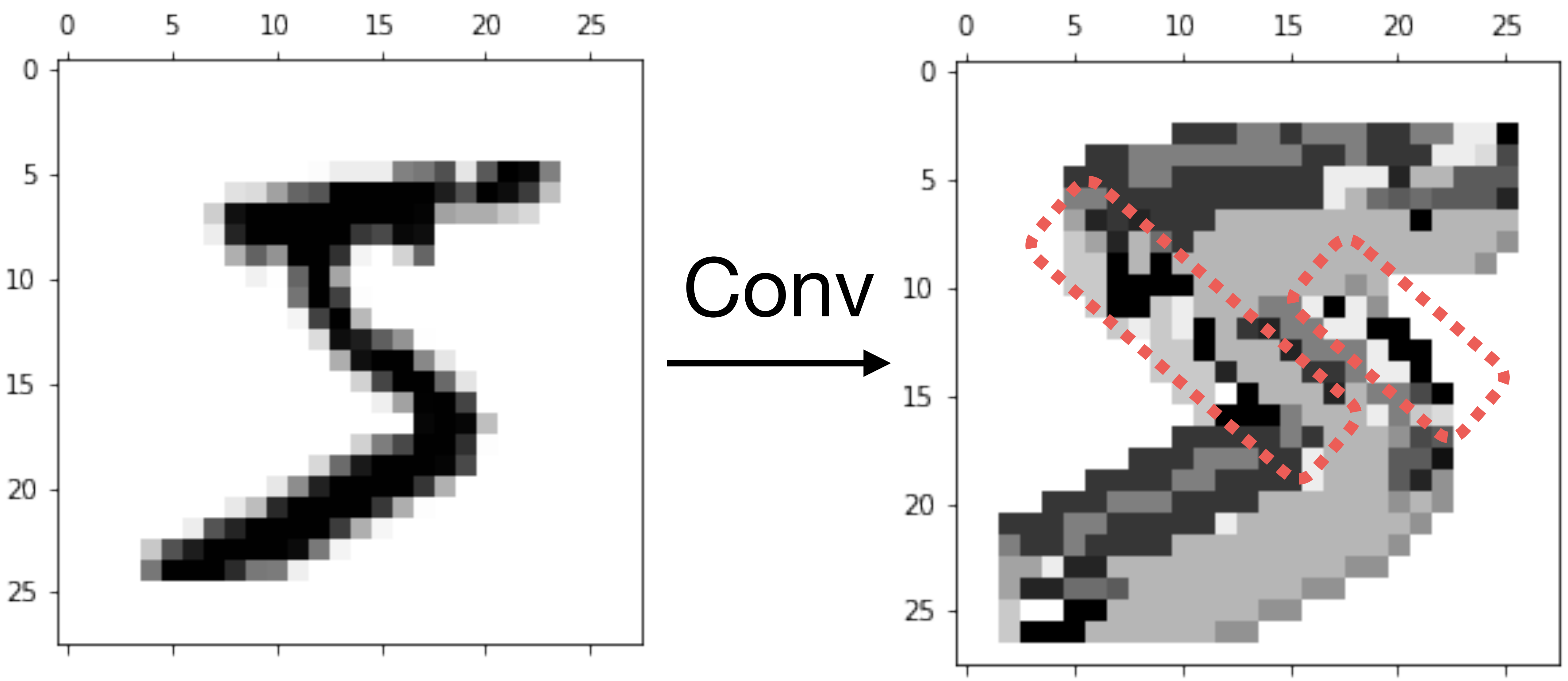
卷積層 (Convolution)





(images from Theano)

卷積層 (Convolution)



(28 x 28) (28 x 28)

填補 (Padding)

pixels

0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	6	0	0
0	3	3	2	0	0
0	1	0	2	1	0
0	0	0	0	0	0

$\otimes$

濾鏡

2	2	1
1	0	3
3	0	2

=

2	15	19	8
15	19	15	17
6	30	22	21
12	17	8	8

- 使經過濾鏡後的矩陣不會越來越小

池化層 (Pooling)

- Max-Pooling (2 x 2)

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

→

1	a_{12}
a_{21}	a_{22}

- 吸收資料中的偏差值
- 降低計算量/控制overfitting

池化層 (Pooling)

- Max-Pooling (2 x 2)

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

$\longrightarrow$

1	6
a_{21}	a_{22}

池化層 (Pooling)

- Max-Pooling (2 x 2)

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

→

1	6
3	a_{22}

池化層 (Pooling)

- Max-Pooling (2 x 2)

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

→

1	6
3	2

池化層 (Pooling)

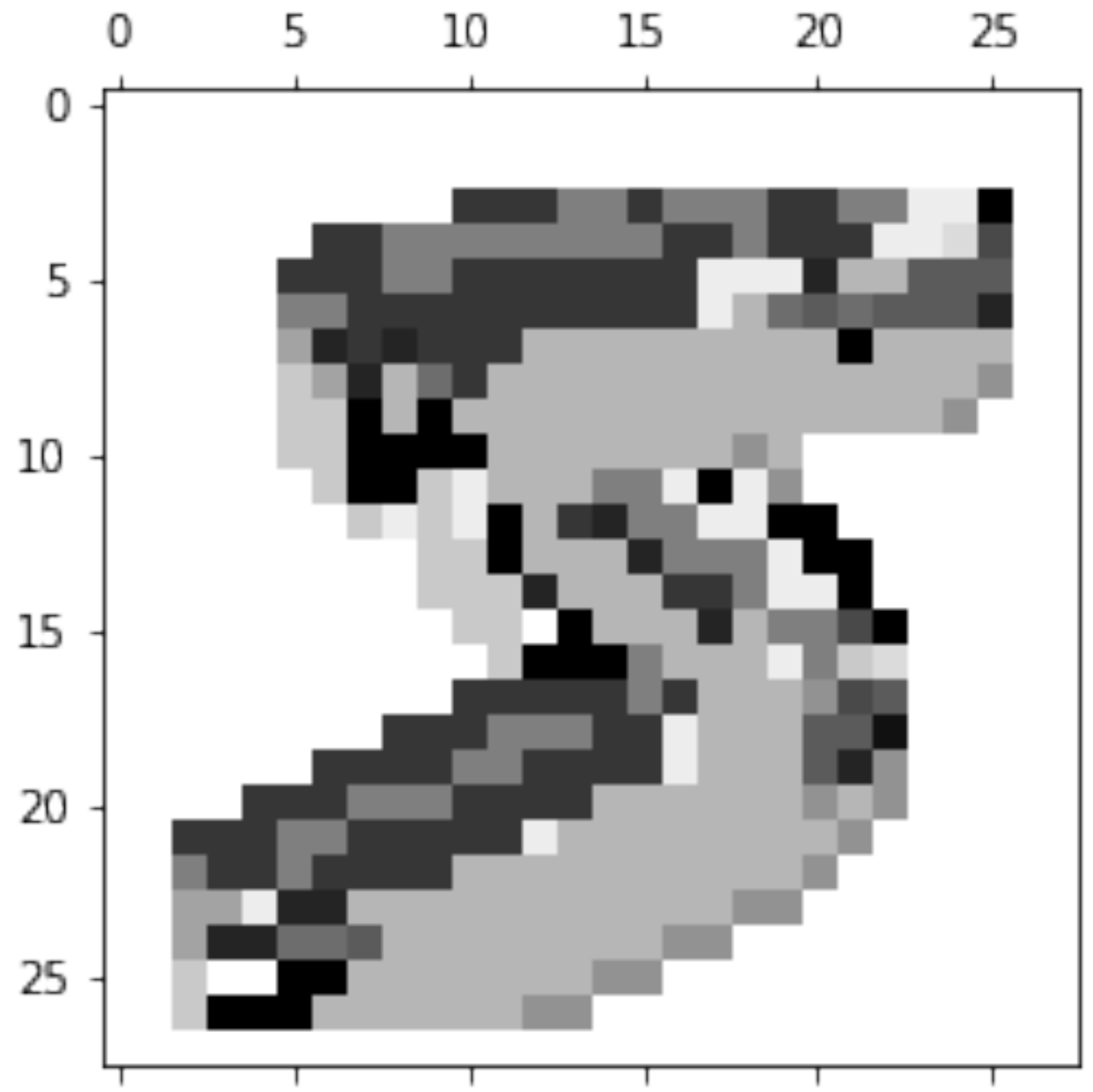
- Average-Pooling (2 x 2)

0	0	1	1
0	1	6	0
3	3	2	0
1	0	2	1

→

0.25	2
1.75	1.25

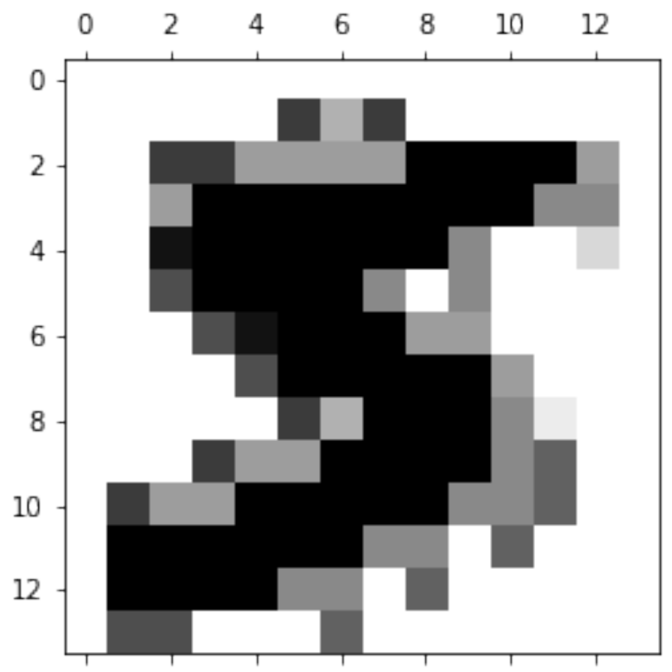
池化層 (Pooling)



(28 x 28)

Max-pooling

(2 x 2)



(14 x 14)

Conv2D

```
keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid', data_format=None, dilation_rate=(1, 1), activation=None,
use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None)
```

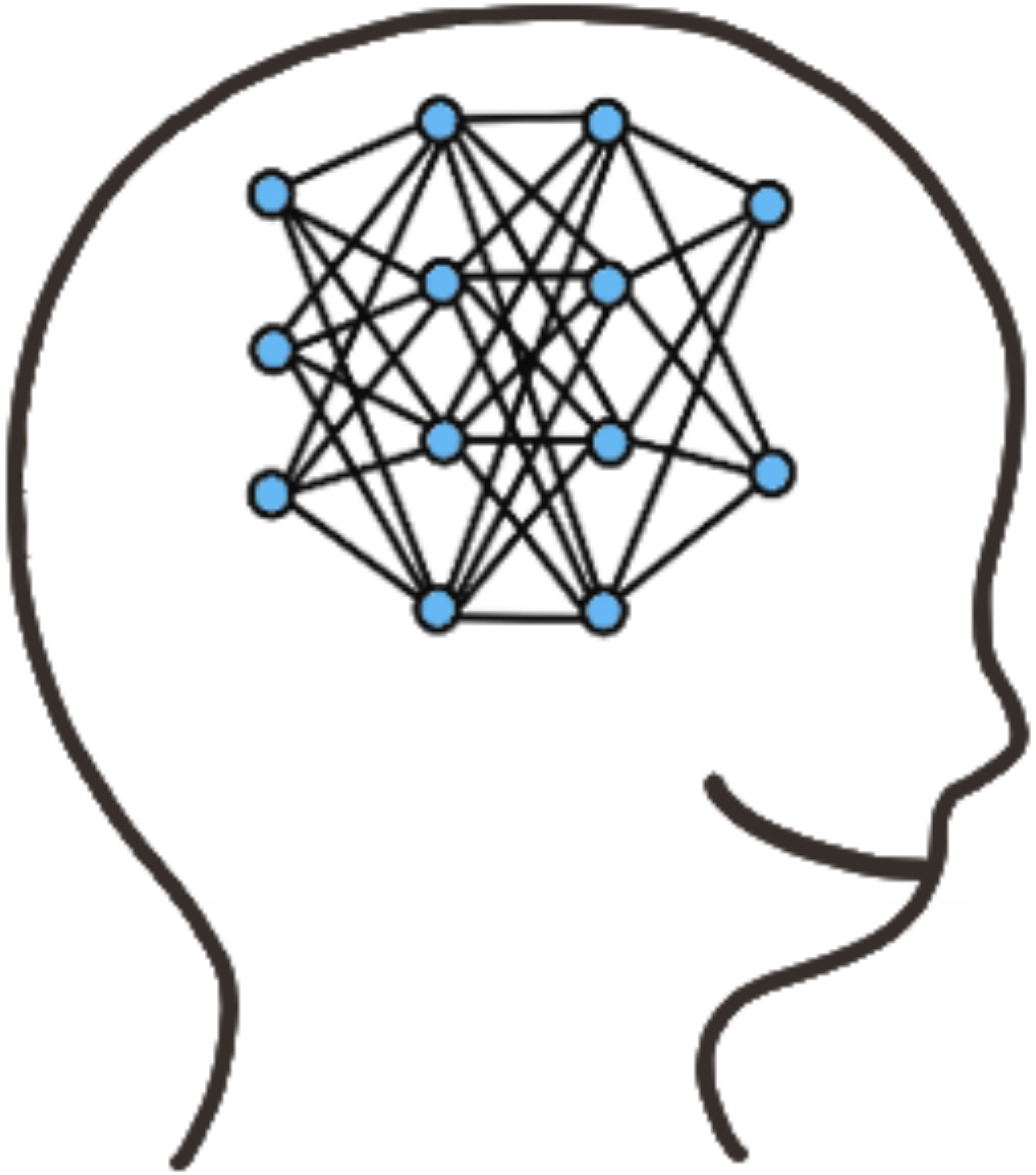
- **filters**: 濾鏡數量
- kernel_size: 濾鏡大小
- strides: 步幅(width, height)或單一數值
- padding: ‘same’-與input shape相同, ‘valid’-不使用padding
- kernel_initializer: 初始weights方法
- activation: ‘relu’ etc.
- **input_shape: 當Conv2D是Sequential第一層時要加上的參數**

MaxPooling2D

- `keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)`
  - pool_size: pool大小
  - strides: 步幅(width, height)或單一數值, None=pool_size
  - padding: ‘same’-與input shape相同, ‘valid’-不使用padding

基於CNN的架構 (補)

- LeNet
- AlexNet
- VGG
- GoogleLeNet
- ResNet

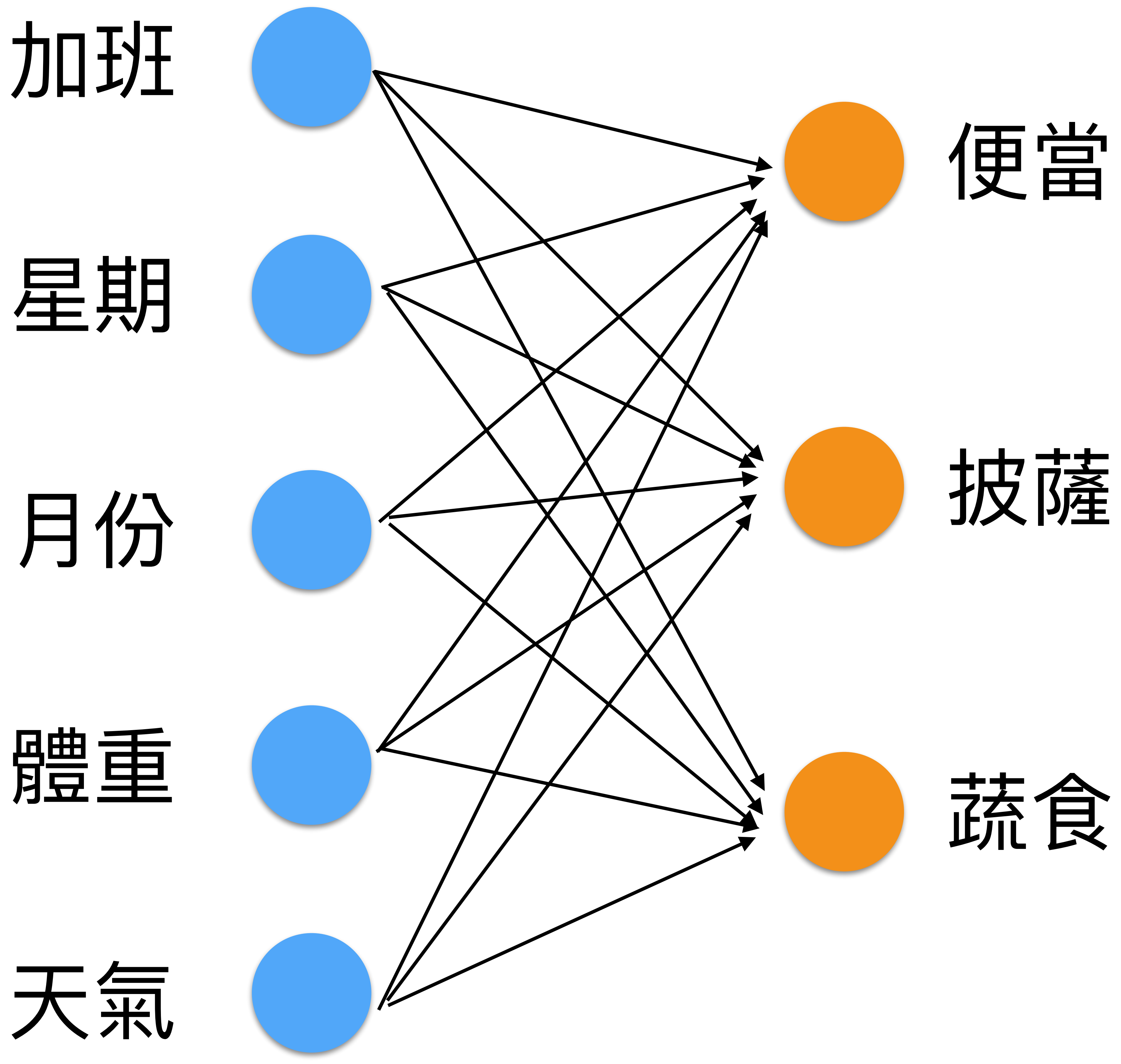


遞迴神經網路

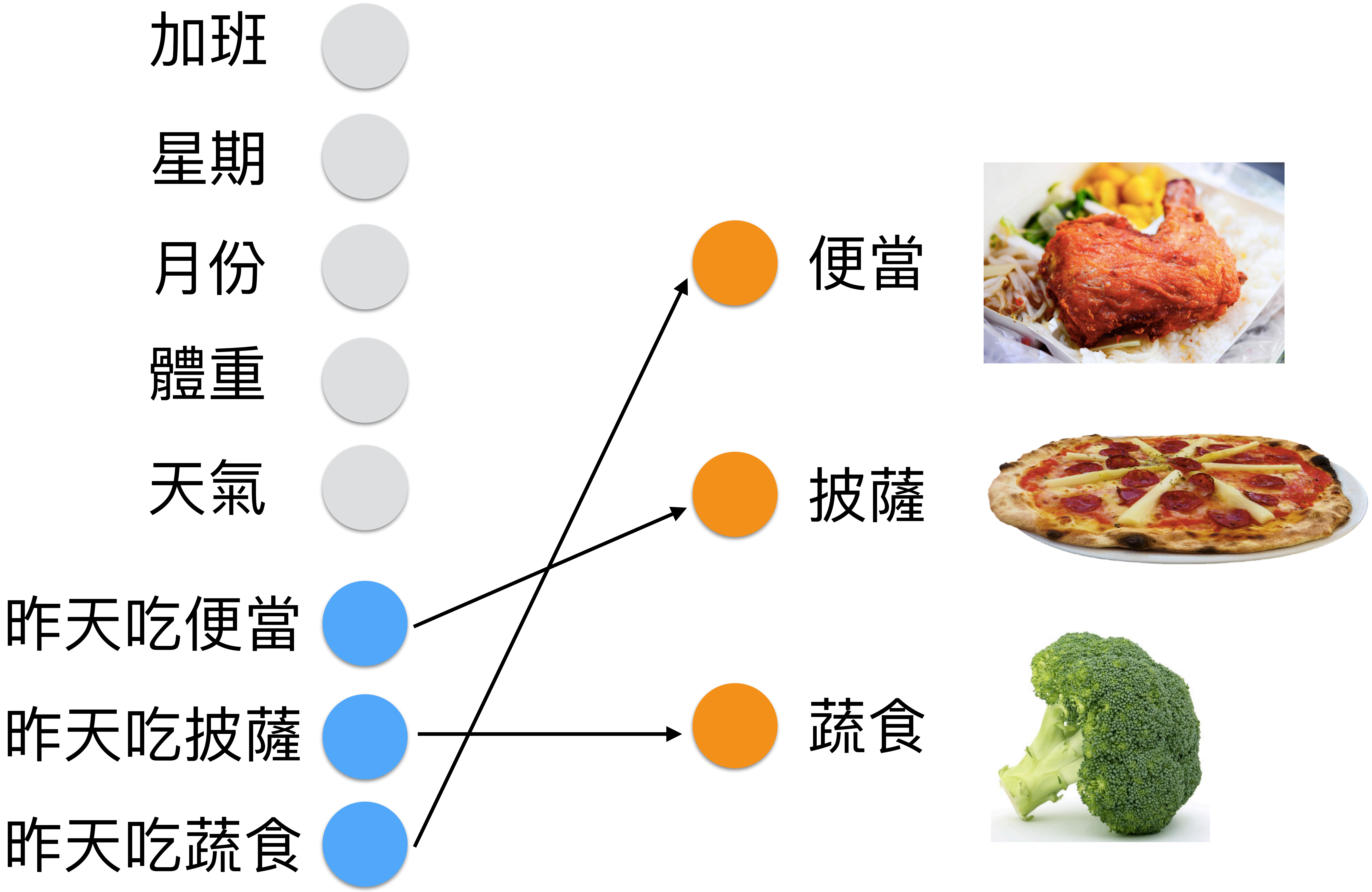
Recurrent Neural Network, RNN

Why RNN

- 晚餐吃什麼？



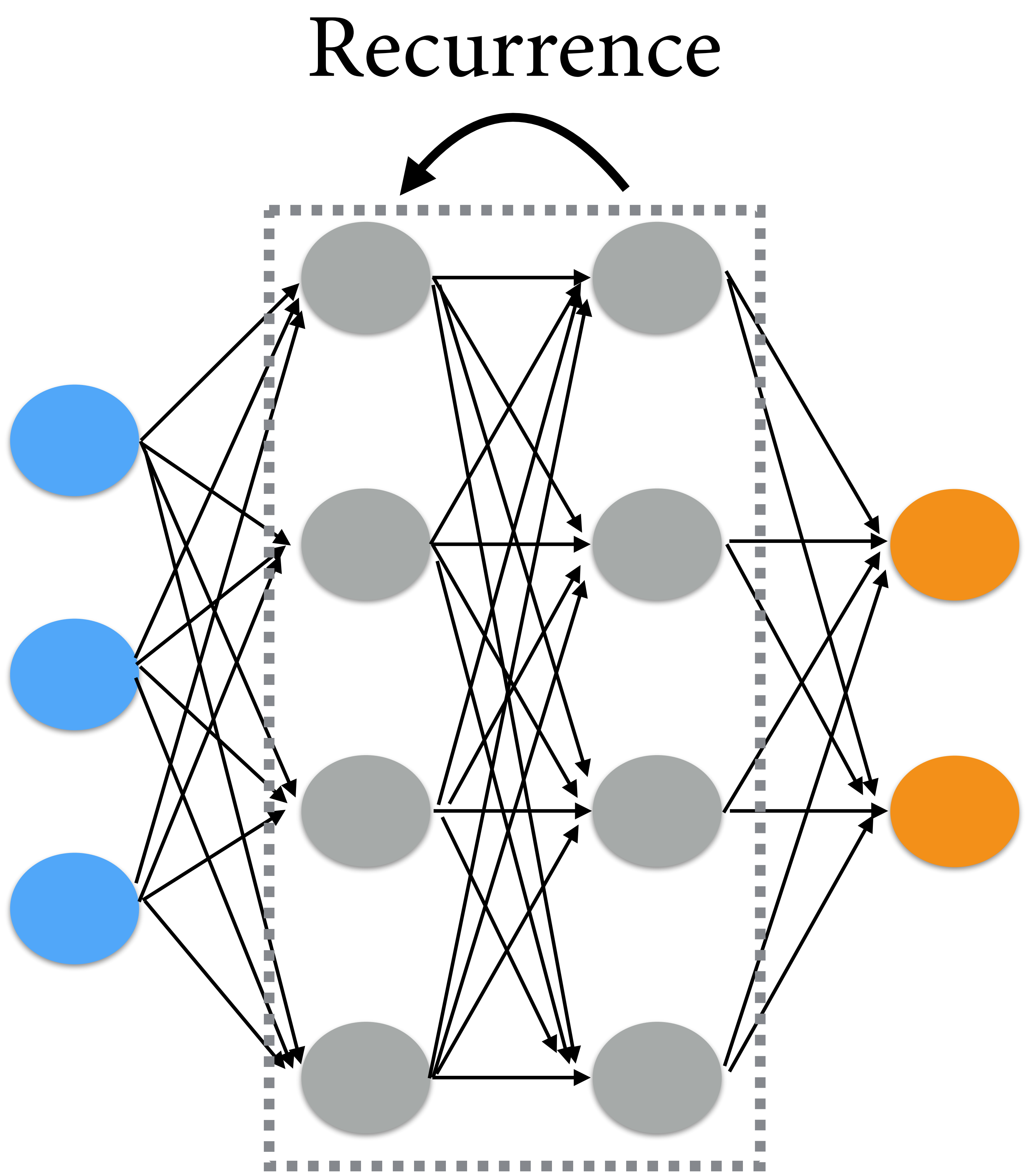
Why RNN



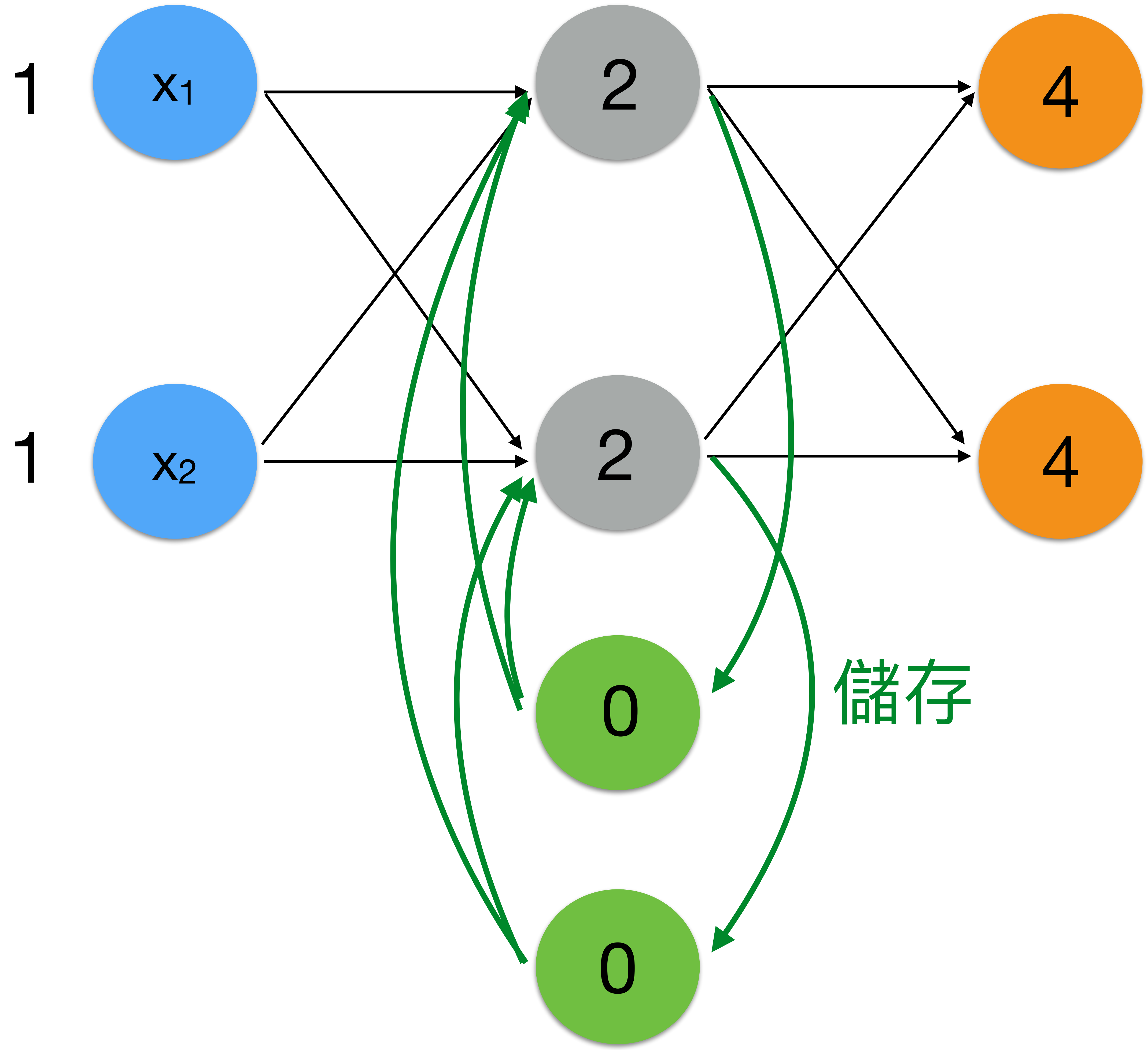
CNN vs. RNN

- 深度學習重要的兩個應用領域：
  - 圖像辨識：
    - 基於CNN的架構
    - 無時間性
  - 序列到序列 (Sequence to Sequence, Seq2Seq)：
    - 基於RNN的架構
    - 有時間性: 預測股市
    - 語音辨識、翻譯、對話生成

RNN



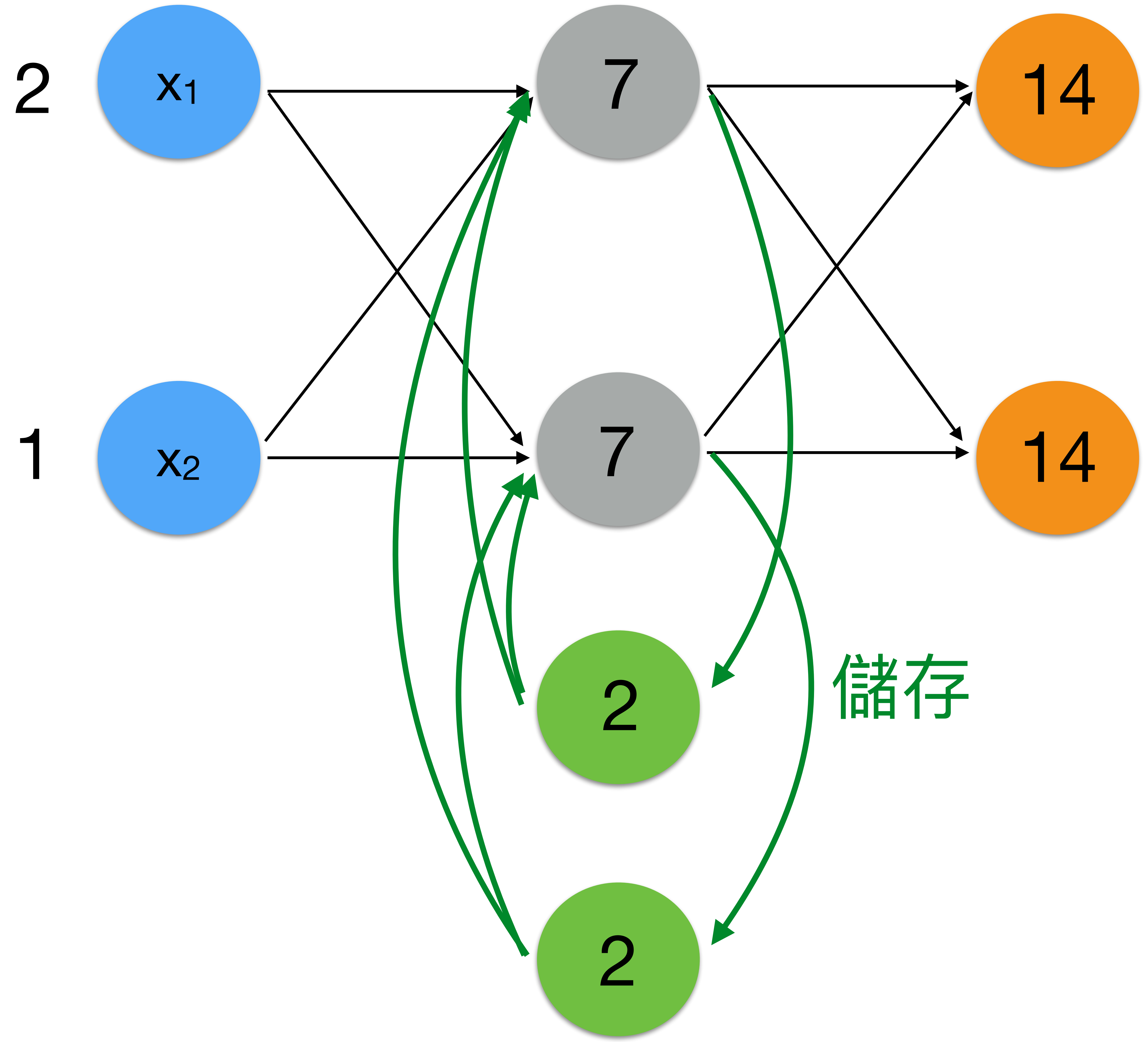
RNN example



- 假設權重皆為1，沒有bias項

$$\text{Input}=\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}2\\1\end{pmatrix}\begin{pmatrix}1\\3\end{pmatrix}\cdots\ m$$

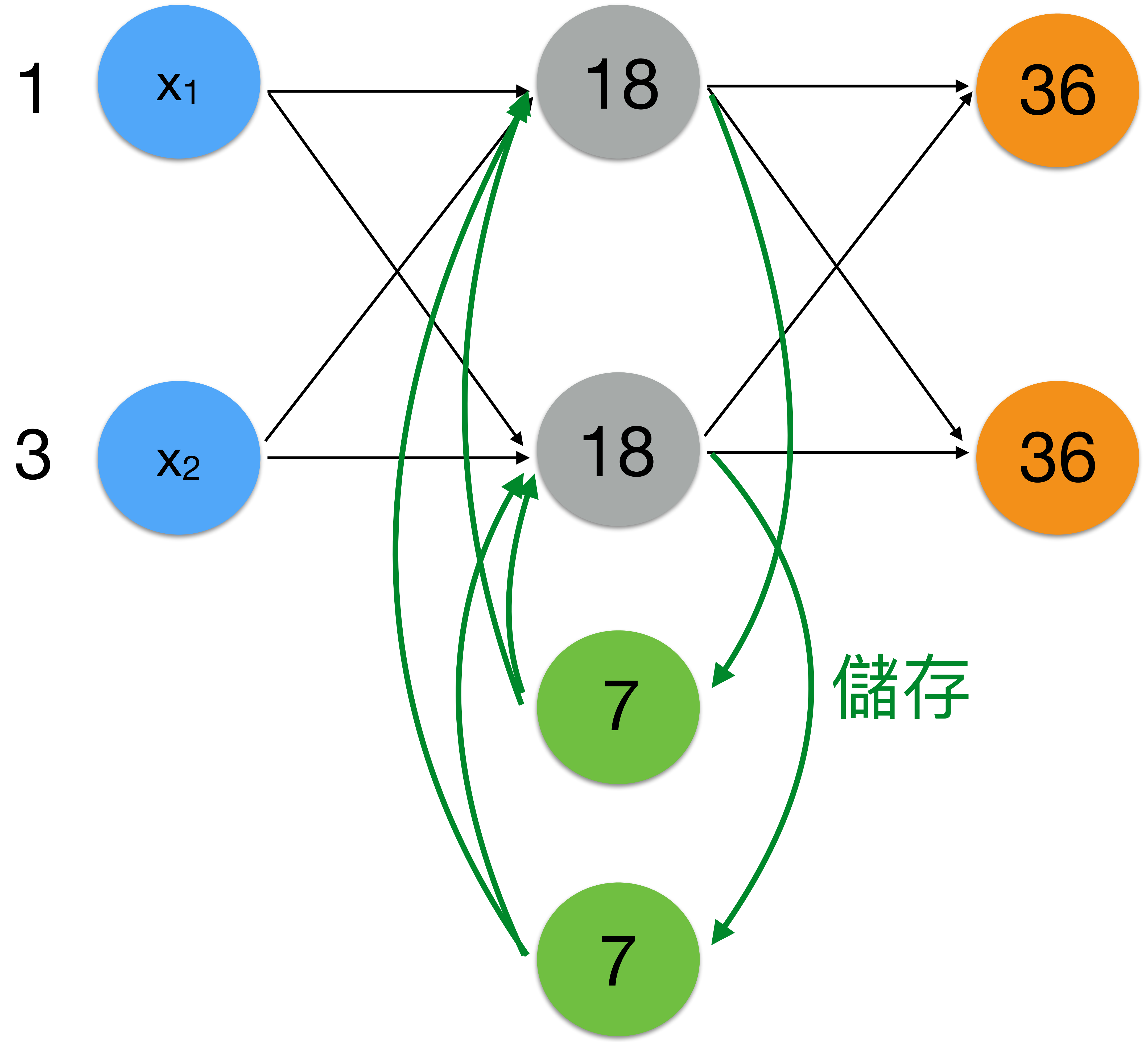
RNN example



- 假設權重皆為1，沒有bias項

$$\text{Input}=\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}2\\1\end{pmatrix}\begin{pmatrix}1\\3\end{pmatrix}\cdots\ m$$

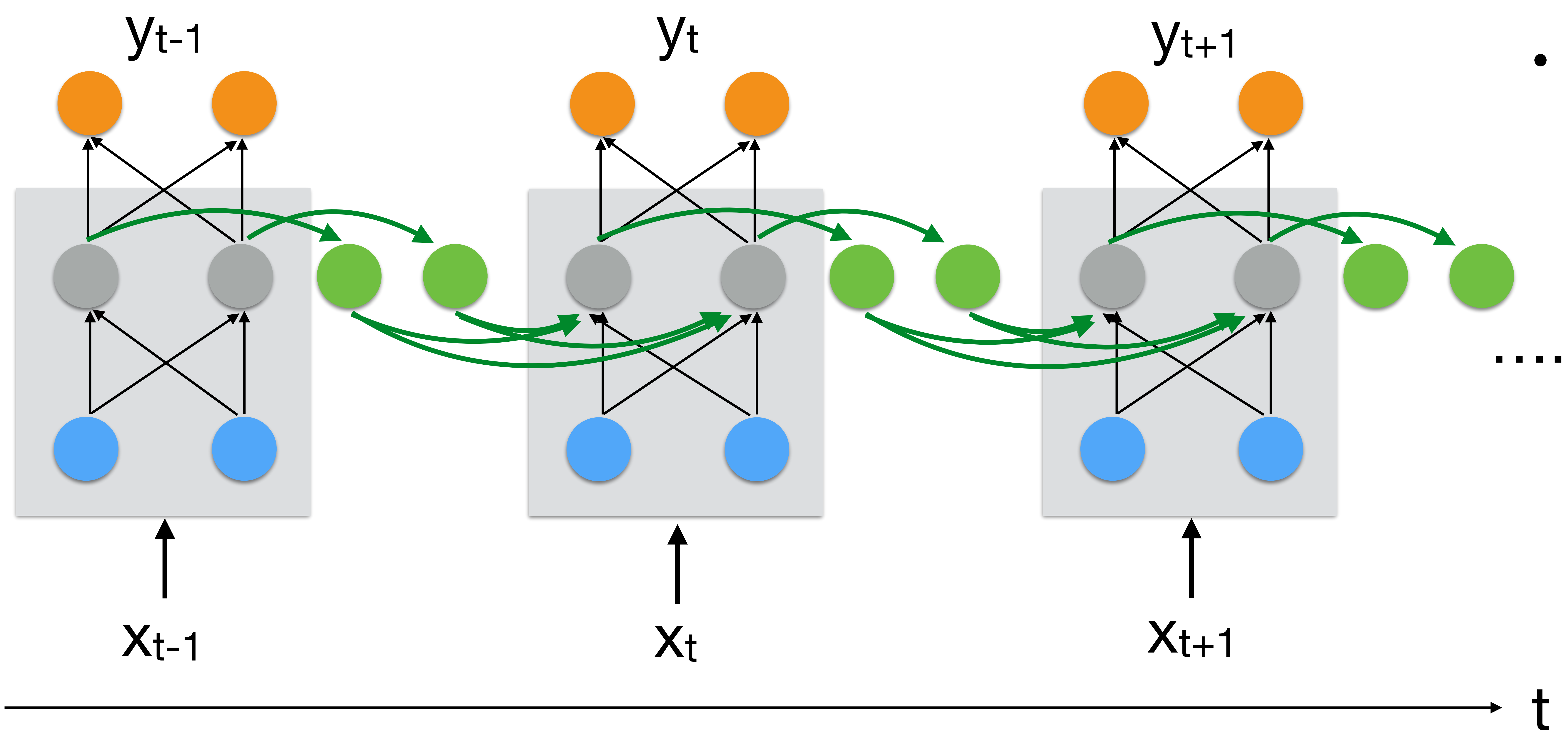
RNN example



- 假設權重皆為1，沒有bias項

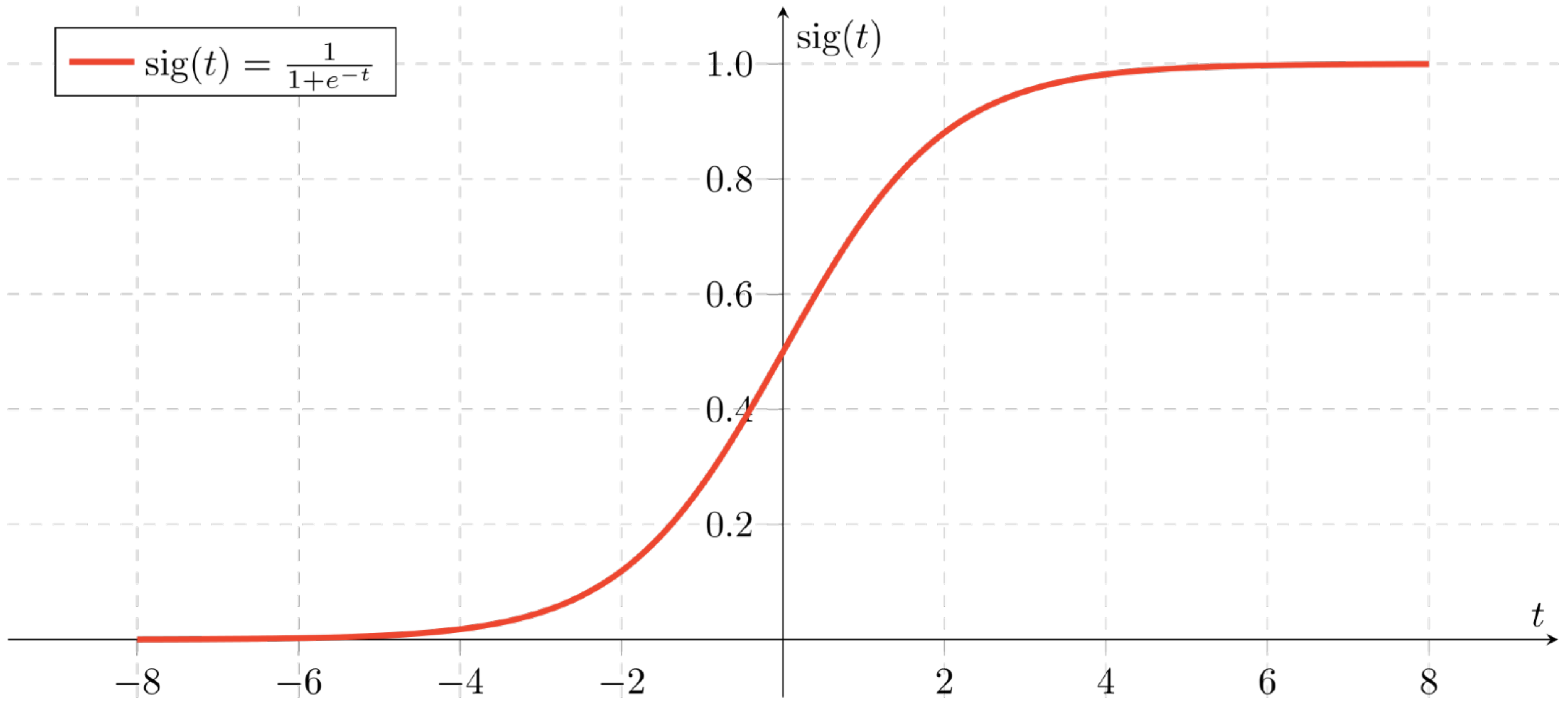
$$\text{Input}=\begin{pmatrix}1\\1\end{pmatrix}\begin{pmatrix}2\\1\end{pmatrix}\begin{pmatrix}1\\3\end{pmatrix}\cdots\ m$$

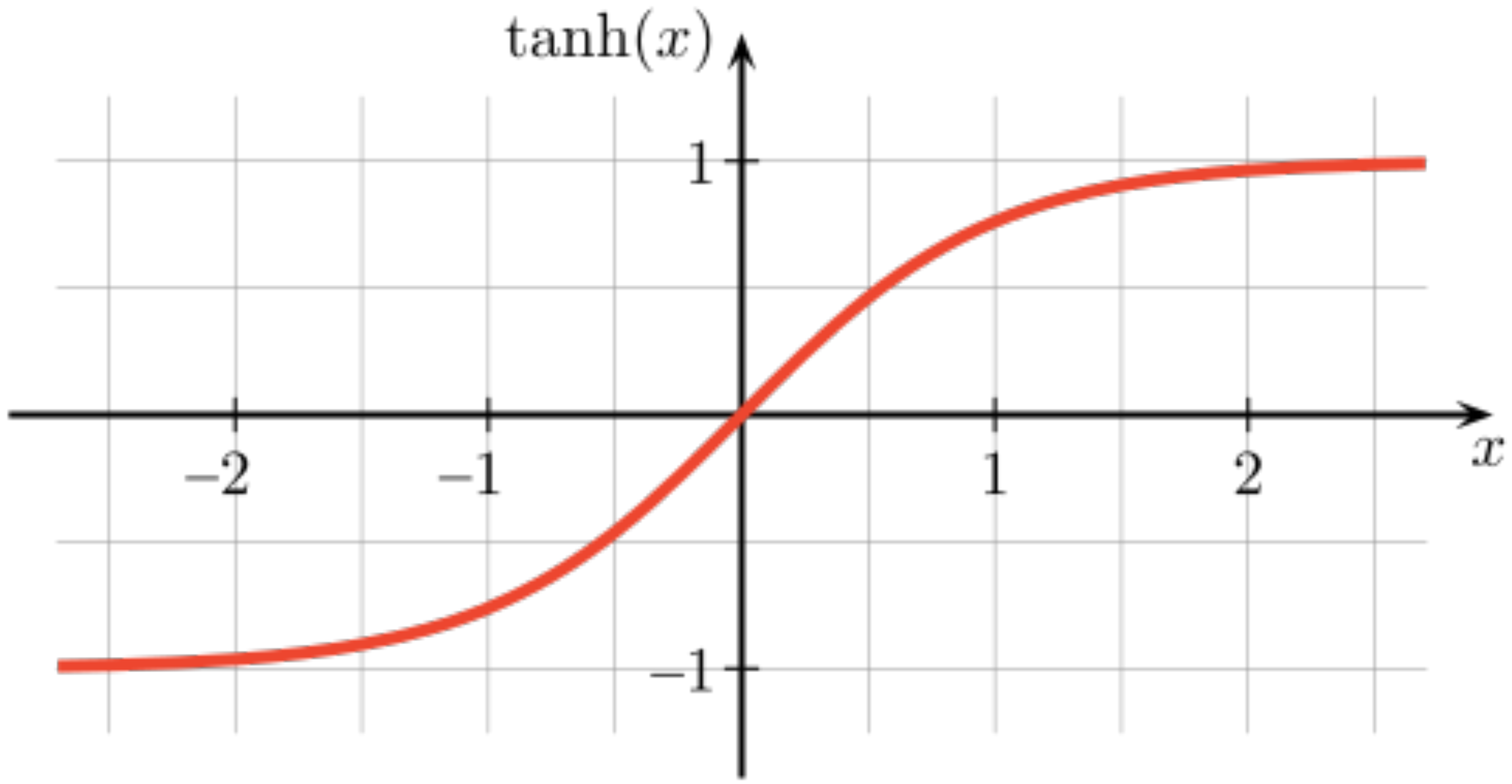
RNN



- 不同時間點間的模型權重傳遞

Activation Function



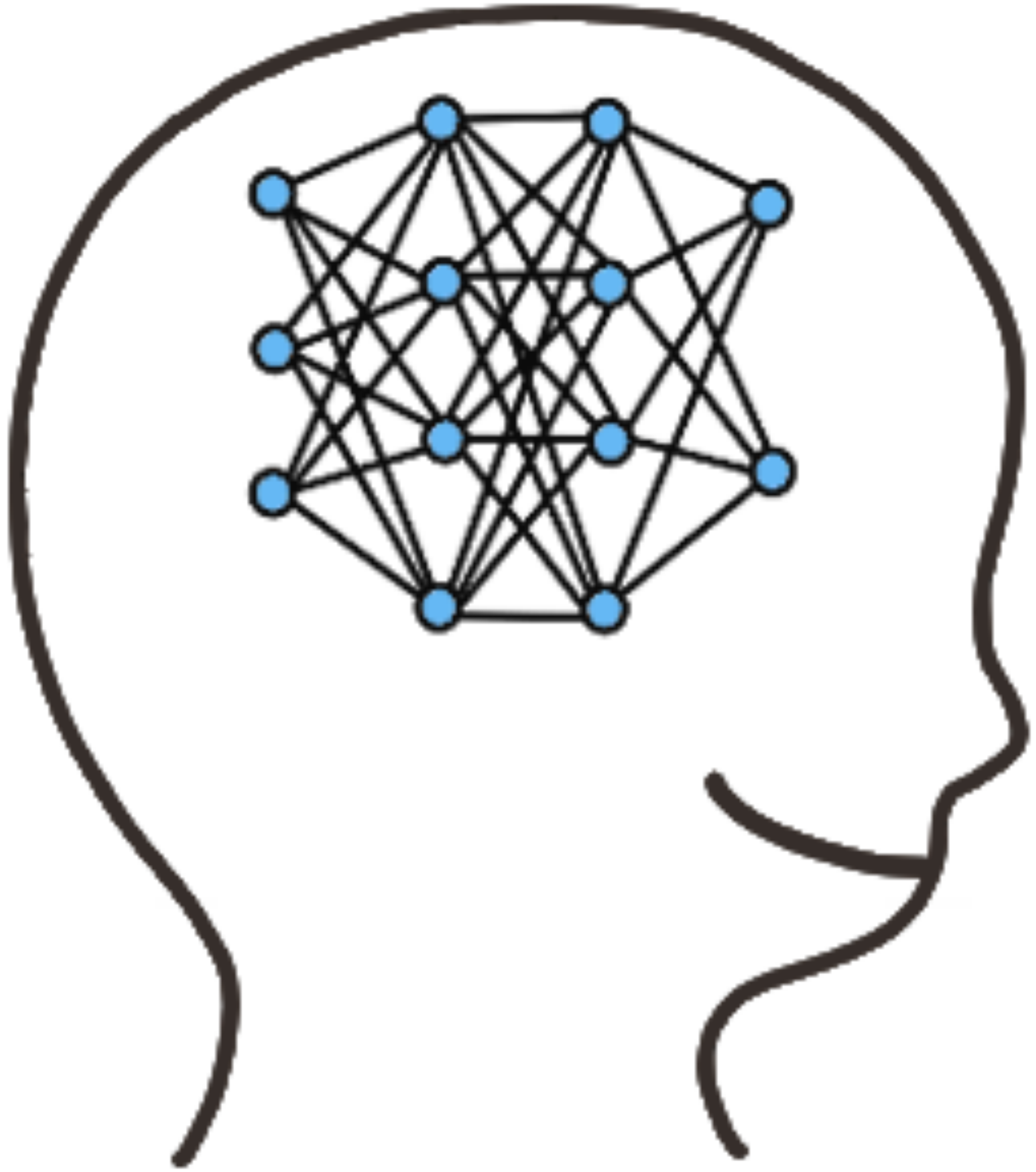


Note:

- IRNN (Hinton, 2015) 使用ReLU

SimpleRNN

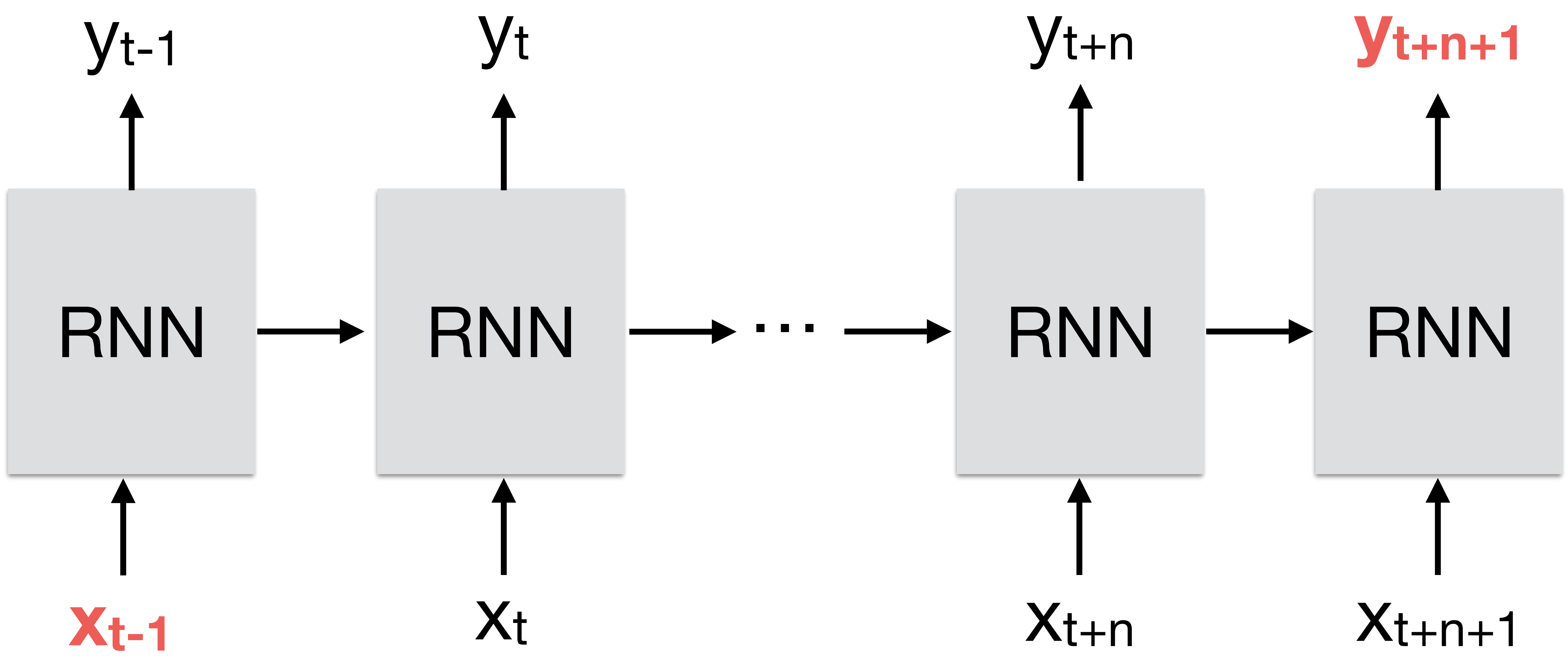
- `keras.layers.recurrent.SimpleRNN(units, activation='tanh', use_bias=True, kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initializer='zeros', kernel_regularizer=None, recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)`
  - **units**: Output 的維度
  - **dropout**: input 時的 dropout 比例
  - **recurrent_dropout**: recurrent 時的 dropout 比例
  - **activation**: Activation function to use (see activations). If you pass None, no activation is applied (ie. "linear" activation: `a(x) = x`).
  - **use_bias**: Boolean, whether the layer uses a bias vector.
  - **kernel_initializer**: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see initializers).
  - **recurrent_initializer**: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see initializers).
  - **bias_initializer**: Initializer for the bias vector (see initializers).
  - **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see regularizer).
  - **recurrent_regularizer**: Regularizer function applied to the `recurrent_kernel` weights matrix (see regularizer).
  - **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
  - **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).



長短期記憶

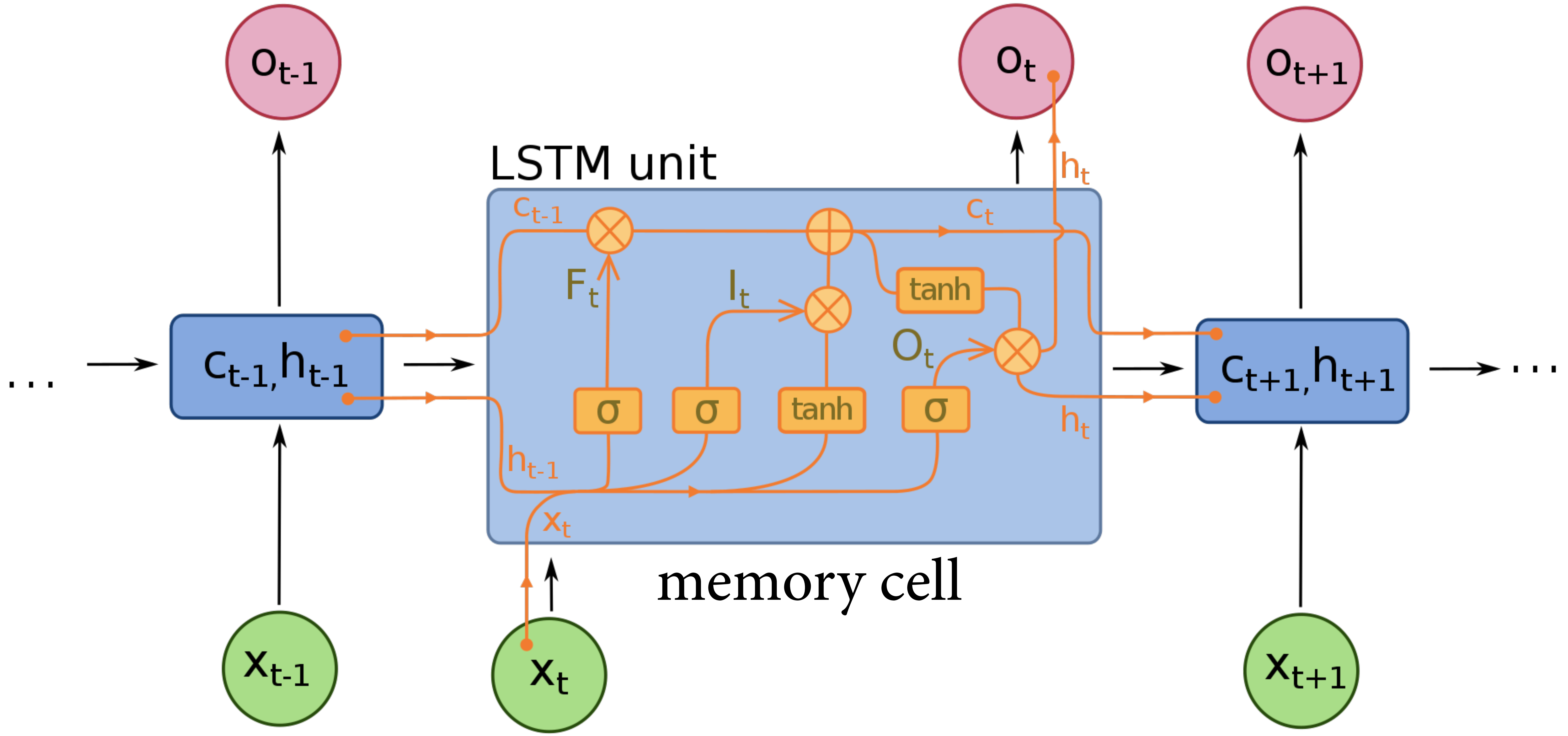
Long Short-Term Memory, LSTM

The Problem of Long-Term Dependencies



- 記憶力差，無法回溯到較久遠的input

LSTM



(from wikimedia)

- I_t: Input Gate
  - 決定是否輸入
- O_t: Output Gate
  - 決定是否輸出
- F_t: Forget Gate
  - 決定是否遺忘

LSTM

- `keras.layers.recurrent.LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid', use_bias=True, kernel_initializer='glorot_uniform', recurrent_initializer='orthogonal', bias_initializer='zeros', unit_forget_bias=True, kernel_regularizer=None, recurrent_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, recurrent_constraint=None, bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)`
  - **units**: Output 的維度
  - **dropout**: input 時的 dropout 比例
  - **recurrent_dropout**: recurrent 時的 dropout 比例
  - **activation**: Activation function to use (see activations). If you pass None, no activation is applied (ie. "linear" activation: `a(x) = x`).
  - **recurrent_activation**: Activation function to use for the recurrent step (see activations).
  - **use_bias**: Boolean, whether the layer uses a bias vector.
  - **kernel_initializer**: Initializer for the `kernel` weights matrix, used for the linear transformation of the inputs. (see initializers).
  - **recurrent_initializer**: Initializer for the `recurrent_kernel` weights matrix, used for the linear transformation of the recurrent state. (see initializers).
  - **bias_initializer**: Initializer for the bias vector (see initializers).
  - **unit_forget_bias**: Boolean. If True, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"`. This is recommended in Jozefowicz et al.
  - **kernel_regularizer**: Regularizer function applied to the `kernel` weights matrix (see regularizer).
  - **recurrent_regularizer**: Regularizer function applied to the `recurrent_kernel` weights matrix (see regularizer).
  - **bias_regularizer**: Regularizer function applied to the bias vector (see regularizer).
  - **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation"). (see regularizer).
  - **kernel_constraint**: Constraint function applied to the `kernel` weights matrix (see constraints).
  - **recurrent_constraint**: Constraint function applied to the `recurrent_kernel` weights matrix (see constraints).
  - **bias_constraint**: Constraint function applied to the bias vector (see constraints).

References

- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.
    - http://www.bioinf.jku.at/publications/older/2604.pdf