

Recurrent Neural Network

臺灣大學人工智慧中心 科技部人工智慧技術暨全幅健康照護聯合研究中心 <http://ai.ntu.edu.tw/>

前導知識：字轉成vector的方式

- 1-of-N encoding

1-of-N encoding

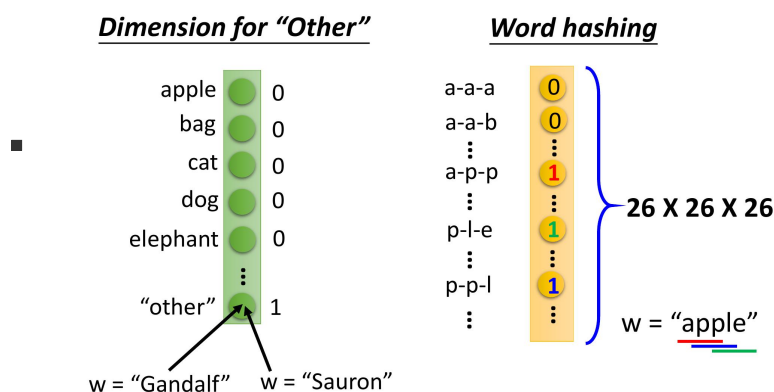
How to represent each word as a vector?

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
 - 1-of-N encoding 的一些問題
 - 起因：很多詞彙可能從來沒有見過
 - 解法：多加一個 dimension，這個 dimension 代表 other
 - Beyond
 - 用某一個詞彙的字母來表示，像是 n-gram
 - 舉例 apple (3-gram)：有 "app"、"ppl"、"ple"

Beyond 1-of-N encoding

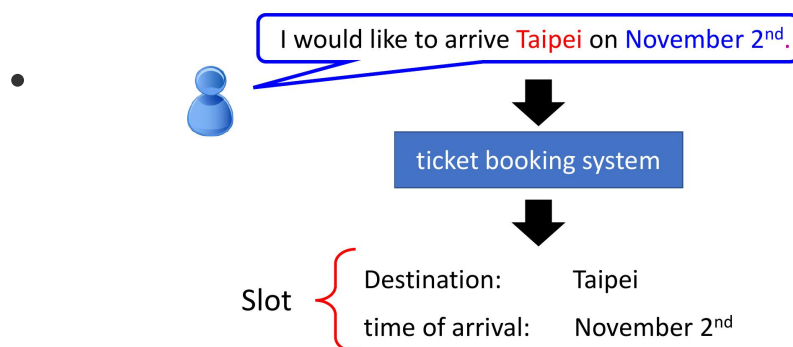


Task example: Slot Filling

- 實際例子：智慧客服、訂票系統
- 分析一段句子有哪些slot，如：Destination、Time of Arrival

Example Application

- Slot Filling



解法 (Naive Model)

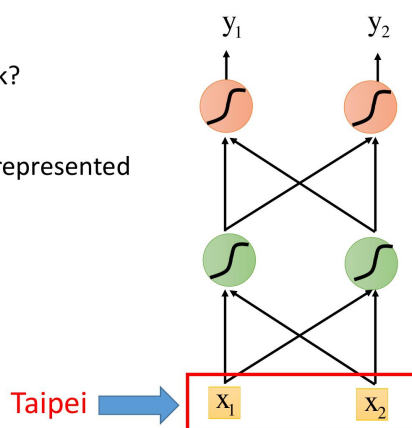
- 可以用一個 Feedforward 的 Neural Network 來解
-

Example Application

Solving slot filling by
Feedforward network?

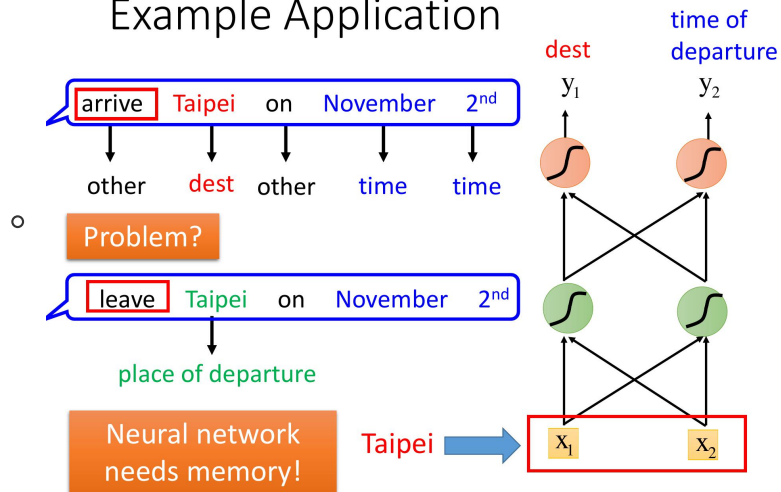
Input: a word

(Each word is represented
as a vector)



- input 是一個詞彙（像 Taipei 可以轉成一個 vector）
- output 是一個 probability distribution，代表說我們現在 input 的這個詞彙屬於哪一個 slot 的機率（像 Taipei 屬於 destination 的機率，還是屬於 time of departure 的機率）
- Problem: example

Example Application

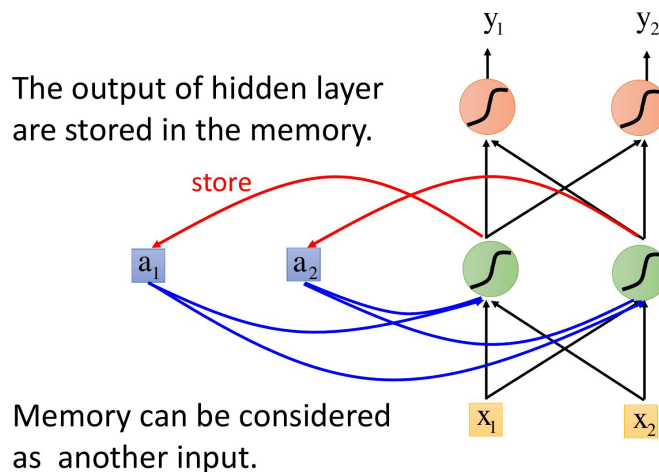


- Taipei 在下面的例子並不是destination，但對於model來說，上下input一樣，會得到相同結果。
- 希望 model 可以有 **記憶力**（先看過arrive或是leave）

解法 (RNN)

Recurrent Neural Network, 有記憶力的 neural network

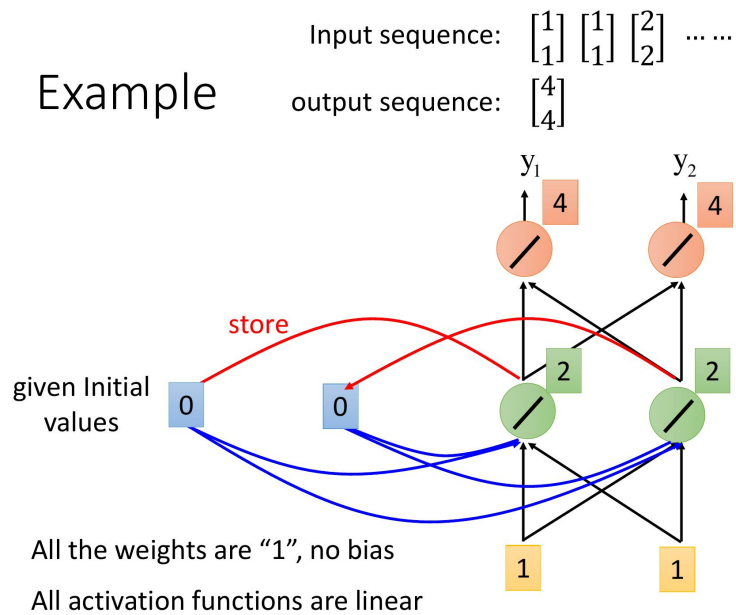
Recurrent Neural Network (RNN)



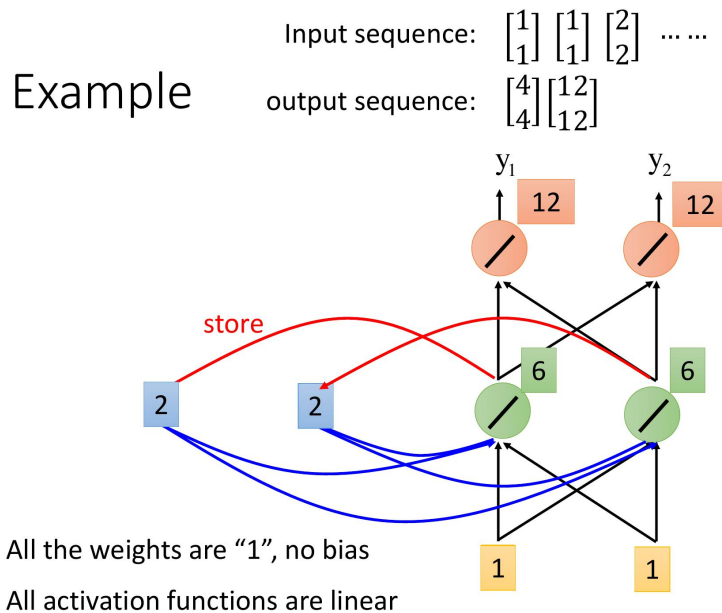
- 每一次我們的 hidden layer 裡面的 neuron 產生 output 的時候，這個 output 都會被存到 memory（藍色的方塊）裡面去
- 下一次當有 input 的時候，這些 neuron 不是只會考慮input 的這個 x_1 跟 x_2 ，它還會考慮存在這些 memory 裡面的值（hidden layer 的值）
- 換句話說：除了 x_1 跟 x_2 以外，存在 memory 裡面的值（ a_1 、 a_2 ）也會影響它的 output

實際例子

- 假設條件：下圖的這個 network
 - 所有的 weight 都是 1
 - 所有的 neuron 沒有 bias 值
 - 所有的 activation function 都是 linear
 - memory 起始值 0
 - input 是 $\begin{bmatrix} 1 & 1 \end{bmatrix}$ 、 $\begin{bmatrix} 1 & 1 \end{bmatrix}$ 、 $\begin{bmatrix} 2 & 2 \end{bmatrix}$
-



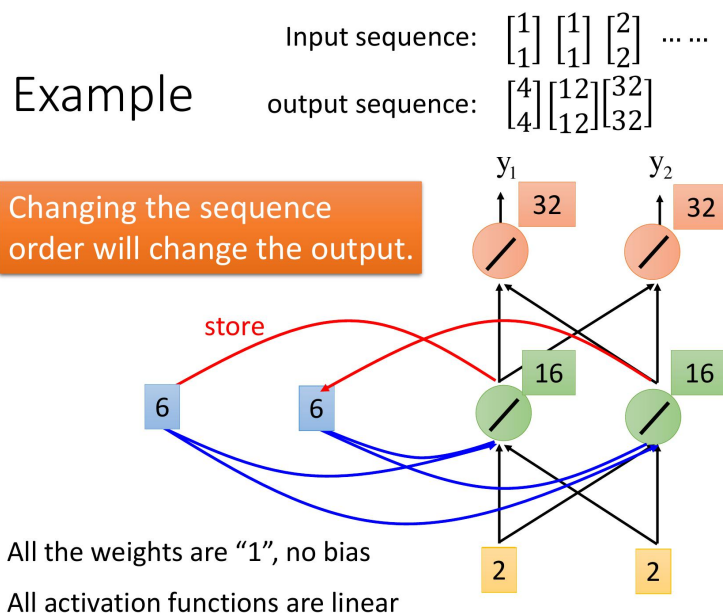
- RNN吃到第一個 $\begin{bmatrix} 1 & 1 \end{bmatrix}$
 - 第一層（綠色）Neuron 吃到的
 - input（黃色）： $\begin{bmatrix} 1 & 1 \end{bmatrix}$
 - memory（藍色）： $\begin{bmatrix} 0 & 0 \end{bmatrix}$
 - output: 2（得到的 output 存回 memory）
 - 第二層（紅色）Neuron 吃到的
 - input（綠色）： $\begin{bmatrix} 2 & 2 \end{bmatrix}$
 - output: 4
-



- RNN吃到第二個 $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
 - 第一層（綠色） Neuron 吃到的
 - input（黃色）： $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$
 - memory（藍色）： $\begin{bmatrix} 2 \\ 2 \end{bmatrix}$
 - output: $2 + 2 + 1 + 1 = 6$ （得到的 output 存回 memory）
 - 第二層（紅色） Neuron 吃到的
 - input（綠色）： $\begin{bmatrix} 6 \\ 6 \end{bmatrix}$
 - output: 12

小結論：

- 對 Recurrent Neural Network 來說，就算是輸入一樣的東西，它的 output 是有可能會不一樣的
- 因為存在 memory，裡面的值會改變結果

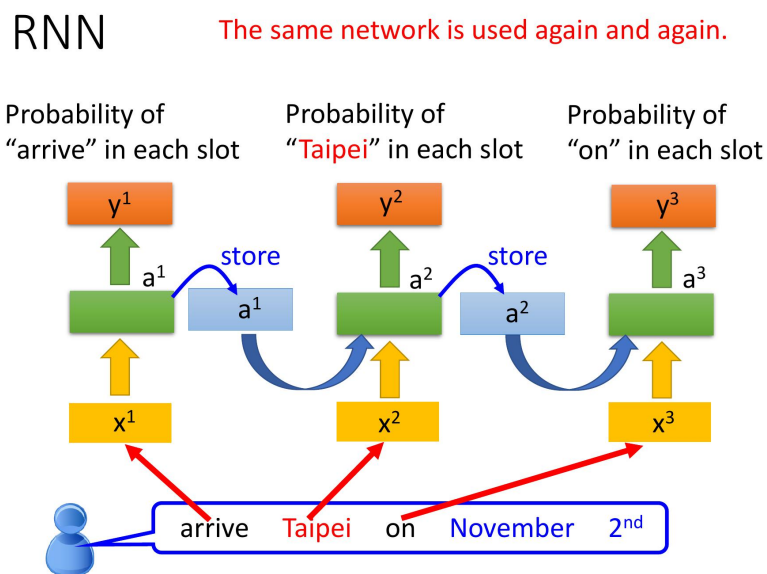


- RNN吃到第三個 [2 2]
 - memory 起始值 0
 - 第一層（綠色） Neuron 吃到的
 - input（黃色）：[1 1]
 - memory（藍色）：[6 6]
 - output: $6 + 6 + 2 + 2 = 16$ （得到的 output 存回 memory）
 - 第二層（紅色） Neuron 吃到的
 - input（綠色）：[16 16]
 - output: 32

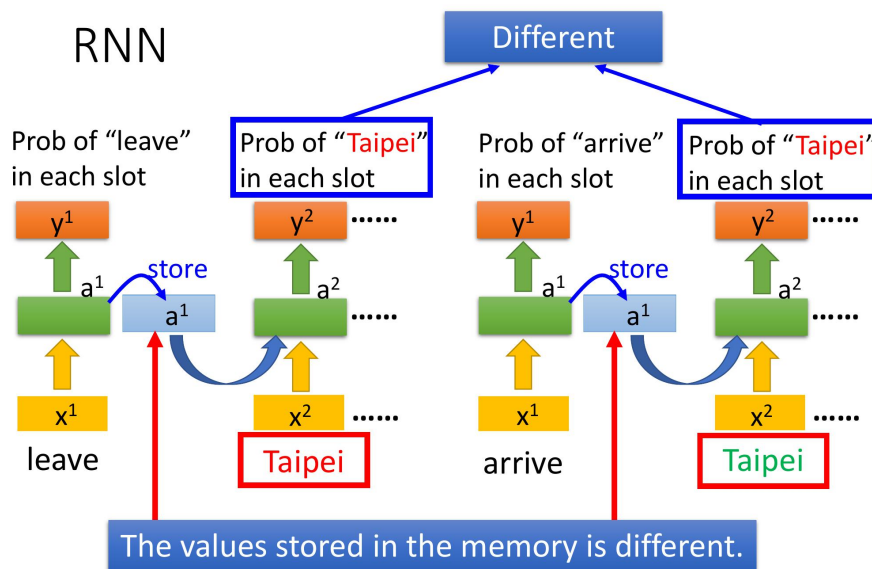
使用 RNN 要注意的事

- input 的 sequence 並不是 independent, sequence 的 order 很重要
- 任意調換 input sequence 的順序, 那 output 會完全不一樣

回歸 slot filling



- 有一個使用者說 **arrive Taipei on November 2nd**
- 當RNN 吃到 arrive
 - 那 arrive 就變成一個 vector, 丟到 neural network 裡面
 - output 為 a^1 , 是一個 vector
 - 根據 a^1 , 產生 y^1 (arrive 屬於哪一個 slot 的機率)
 - a^1 會被存到 memory 裡面
- 當RNN 吃到 Taipei
 - hidden layer 會同時考慮 Taipei 這個 input 跟存在 memory 裡面的 a^1 , 得到 a^2
 - 根據 a^2 , 產生 y^2 (Taipei 屬於哪一個 slot 的機率)
- 依此類推 ...
- 並非3個 network, 而是同一個 network 在 **三個不同的時間點**, 被 **使用了3次**



所以有了 memory 以後，剛才講的輸入同一個詞彙，希望它 output 不同的這個問題，就有可能被解決。

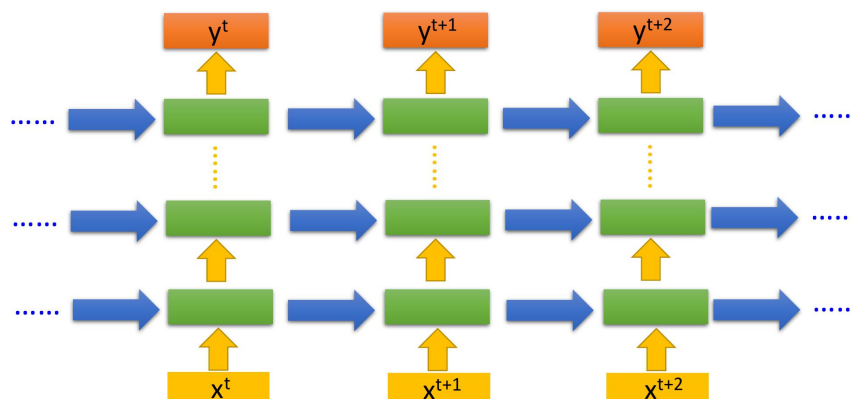
比如說，同樣是輸入 Taipei 這個詞彙，但是因為紅色 Taipei 前面接的是 leave，綠色 Taipei 前面接的是 arrive。

因為 leave 跟 arrive 它們的 vector 不一樣，所以 hidden layer 的 output 也會不同，所以存在 memory 裡面的值呢，也會不同。

即使 x2 是一模一樣的，但是因為存在 memory 裡面的值不同，所以 hidden layer 的 output 也會不一樣，所以最後的 output 也就會不一樣。

其他 Recurrent Neural Network 的架構設計

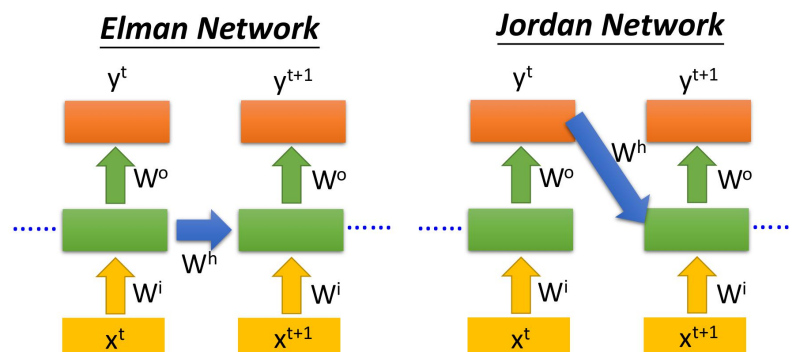
Of course it can be deep ...



- 可以是 deep 的 Recurrent Neural Network
 - 比如說，我們把 x_1 丟進去以後，它可以通過很多個 hidden layer，才得到最後的 output
 - 每一個 hidden layer 的 output 都會被存在 memory 裡面，在下一個時間點的時候呢再讀出來
 - 要多 deep，要幾層，都可以
-

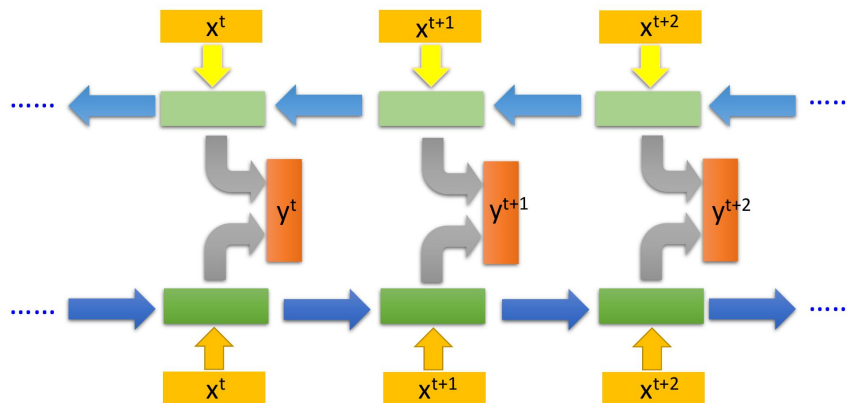
一些有名字的 Recurrent Neural Network

Elman Network & Jordan Network



- Elman Network
 - 把 hidden layer 的值存起來，在下一個時間點再讀出來
 - Jordan Network
 - 存的是整個 network 的 output 的值，把 output 的值存在 memory 裡面
 - 傳說這個可以得到比較好的 performance
 - 因為這邊的 hidden layer，它是沒有 target 的。
 - 如果有 target，可以比較好控制它學到什麼樣的 hidden 的 information 放到 memory 裡面。
-

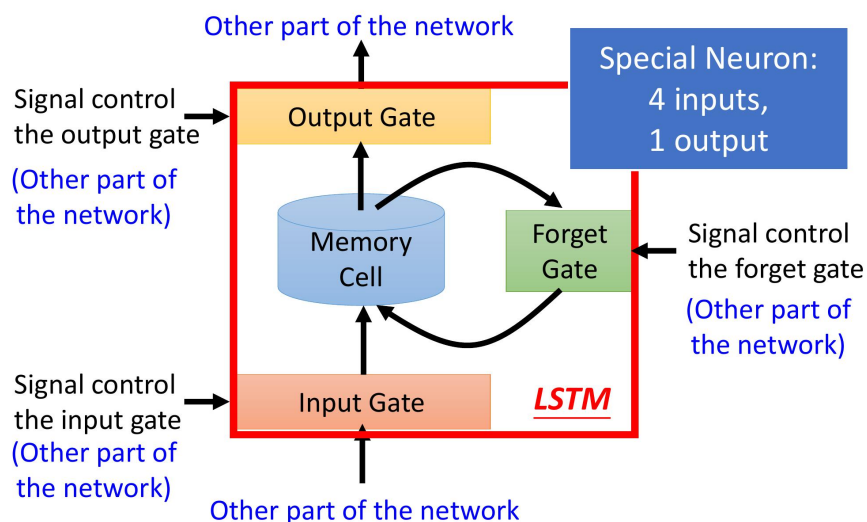
Bidirectional RNN



- 假設句子裡面的每一個詞彙我們都用 x^t 來表示它的話，它就是先讀 x^t ，再讀 x^{t+1} ，再讀 x^{t+2} 。但是，其實它的讀取方向也可以是反過來，它可以先讀 x^{t+2} ，再讀 x^{t+1} ，再讀 x^t
- 同時 train 一個正向的 Recurrent Neural Network，又同時 train 一個逆向的 Recurrent Neural Network。把這兩個 Recurrent Neural Network 的 hidden layer 拿出來，都接給一個 output layer，得到最後的 y
- 舉例：在 input x^t 的時候。正向的 network 的 output 跟逆向的 network 的 output，都丟給 output layer。產生 y^t 。
- 好處：network 看的範圍比較廣，看頭又看尾

Long Short-Term Memory (LSTM)

Long Short-term Memory (LSTM)




LSTM 有 3 個 gate，由model自己學要打開還是關起來

- input gate
 - 當外界，當 neural network 的其他部分，某個 neuron 的 output 想要被寫到 memory cell 裡面的時候，必須要通過
 - 要被打開的時候，你才能夠把值寫到 memory cell 裡面去
- output gate
 - output gate 會決定說，外界、其他的 neuron 可不可以從 memory 裡面把值讀出來
- forget gate
 - 甚麼時候 memory 要把過去記得的東西忘掉，或是它甚麼時候要把過去記得的東西 format 掉

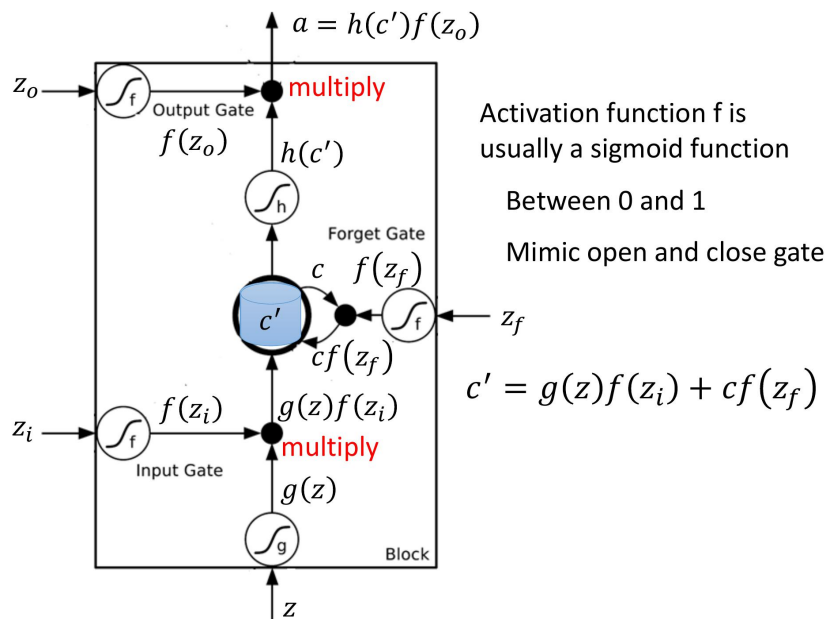
整個 LSTM 呢，可以看成它有4個 input，1 個 output

- 4 個input
 - 想要被存到 memory cell 裡面的值
 - 然後它不一定存得進去，這 depend on input gate 要不要讓這個information 過去
 - 操控 input gate 的這個訊號
 - 操控 output gate 的這個訊號
 - 操控 forget gate 的訊號

Long Short-Term Memory (LSTM) 小小的冷知識

- 就是這個 dash，，應該放在 short 跟 term 之間
 - 只是比較長的 short-term memory
 - 之前我們看那個 Recurrent neural network 阿，它的 memory 在每一個時間點都會被洗掉。所以這個 short-term 是非常 short
 - 但是如果是 long short-term 的話，它可以記得比較長一點，只要 forget gate不要決定要 format 的話，它的值就會被存起來。
-

Formulation of LSTM



● 名詞解釋

- 要被存到 cell 裡面的 input 叫做 z
- 操控 input gate 的 signal 叫做 z_i
- 操控 forget gate 的 signal 叫做 z_f
- 操控 output gate 的 signal 叫做 z_o
- 綜合這些東西以後，最後會得到一個 output，這邊寫作 a

假設裡面一開始已經存了值 c ，現在呢，假設要輸入 z

● 得到真正的 input

- z 通過一個 activation function 得到 $g(z)$
- 把 z_i 通過另外一個 activation function，得到 $f(z_i)$
 - 假設 $f(z_i) = 0$ ，input gate 被關閉的時候，那 $g(z) * f(z_i)$ 就等於 0，就好像是沒有輸入一樣

● 利用前一步的 Input，來 update memory

- 把 $g(z)$ 乘上這個 input gate 的值 $f(z_i)$ 得到 $g(z)*f(z_i)$
- z_f 這個 signal 也通過這個 activation function，得到 $f(z_f)$
- 把存在 memory 裡面的值 c 乘上 $f(z_f)$ ，得到 $c*f(z_f)$
- 把 $c*f(z_f)$ 加上 $g(z)*f(z_i)$ 得到 c' ，存回去 memory
 - 假設 $f(z_f)$ 是 0，forget gate 被關閉的時候，0 乘上 c 還是 0，也就是過去存在 memory 裡面的值呢，就會被遺忘

● 得到真正的 output a

- 把這個 c' 通過 h ，得到 $h(c')$
- z_o 通過 f 得到 $f(z_o)$ 跟這個 $h(c')$ 乘起來，得到 $h(c') * f(z_o)$ ，也就是最後的 a

- 這3個 z_i , z_f , z_o 他們通過的這3個 activation function f ，通常我們會選擇 **sigmoid** function

- sigmoid 的值是介在 0~1 之間的，而這個 0~1 之間的，代表了這個 gate 被打開的程度
- 如果這個 f 的 output 這個 activation function 的 output 是 1，代表這個 gate 是處於被打開的狀態，反之呢，代表這個 gate 是被關起來的。

經典的手爆LSTM舉例

LSTM - Example

	0	0	3	3	7	7	7	0	6
x_1	1	3	2	4	2	1	3	6	1
x_2	0	1	0	1	0	0	-1	1	0
x_3	0	0	0	0	0	1	0	0	1
y	0	0	0	0	0	7	0	0	6

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

When $x_2 = -1$, reset the memory

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

介紹

- 在 network 裡面，只有一個 LSTM 的 cell
- 那我們的 input 都是三維的 vector
- output 都是一維的 vector

Memory Gate模擬

- 第二個 dimension x_2 的值是 1 的時候
 - x_1 的值就會被寫到 memory 裡面去
- 假設 x_2 的值是 -1 的時候
 - memory 就會被 reset

Output Gate模擬

- 假設 x_3 等於 1 的時候
 - 你才會把 output gate 打開，才能夠看到輸出

手爆的完整過程查看影片比較清楚

跟原本RNN的關係

- 原來的 neural network 裡面會有很多的 neuron
 - 我們會把 input 乘上很多不同的 weight, 然後當作是不同 neuron 的輸入
 - 然後每一個 neuron 它都是一個 function。它輸入一個 scalar, output 另外一個 scalar
- LSTM
 - 把那個 LSTM 的那個 memory cell 想成是一個 neuron 就好
 - 做的事情只是把原來一個簡單的 neuron, 換成一個 LSTM 的 cell
 - 而現在的 input 它會乘上不同的 weight, 當作 LSTM 的不同的輸入

- 怎麼得到LSTM的四個input? 現在假設只有兩個neuron:
 - 那 x_1, x_2 乘上某一組 weight, 會去操控第一個 LSTM 的 output gate
 - 乘上另外一組 weight, 操控第一個 LSTM 的 input gate
 - 乘上一組 weight, 當作第一個 LSTM 的 input
 - 乘上另外一組 weight, 當作另外一個 LSTM 的 forget gate 的 input
- 在原來的 neural network 裡面, 一個 neuron 就是一個 input, 一個 output,
- 在 LSTM 裡面它需要 4 個 input, 它才能夠產生一個 output



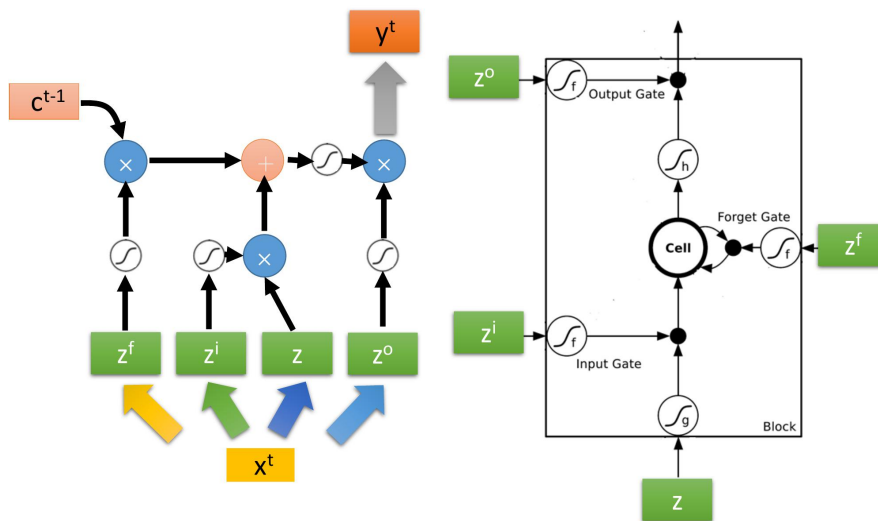
- 一般的 neural network 只需要部分的參數
- LSTM 還要操控另外三個 gate, 所以他需要 4 倍的參數

再更細一點的解釋

Diagram illustrating the internal structure of an unrolled LSTM network. The network consists of three blocks, each containing an Input Gate, a Forget Gate, a Cell, and an Output Gate. The input vectors z^f , z^i , z , and z^o are shown as green boxes, and the hidden state x^t is shown as a yellow box. Green arrows indicate the flow of information from the input vectors to the corresponding gates in the first block. The text "4 vectors" is written next to the input vectors.

- 假設我們現在有一整排的 neuron，假設有一整排的 LSTM
- 那這一整排的 LSTM 裡面，每一個 LSTM 的 cell，它裡面都存了一個 scalar。把所有的 scalar 接起來，就變成一個 vector，這邊寫成 $c^{(t-1)}$ （橘色）
- 這邊每一個 memory 它裡面存的 scalar，就是代表這個 vector 裡面的一個 dimension
- 先得到一整排 input z
 - 在時間點 t ，input 一個 vector, x^t ，這個 vector，它會先乘上一個 linear 的 transform，變成另外一個 vector z
 - z 這個 vector 的每一個 dimension 呢，就代表了操控每一個 LSTM 的 input。所以 z 它的 dimension 就正好是 LSTM 的 memory cell 的數目。
- 再得到一整排 input gate z^i
 - 這個 x^t 會再乘上另外一個 transform，得到 z^i
 - 這個 z^i 呢，它的 dimension 也跟 cell 的數目一樣，每一個 dimension 都會去操控一個 memory
- 再得到一整排 forget gate z^f ，一整排 output gate z^o
- 這 4 個 vector 的 dimension 都跟 cell 的數目是一樣的，那這 4 個 vector 合起來就會去操控這些 memory cell 的運作

LSTM



現在 input 分別是 z , z^i , z^f , z^o ，注意一下就是這 4 個 z 其實都是 vector

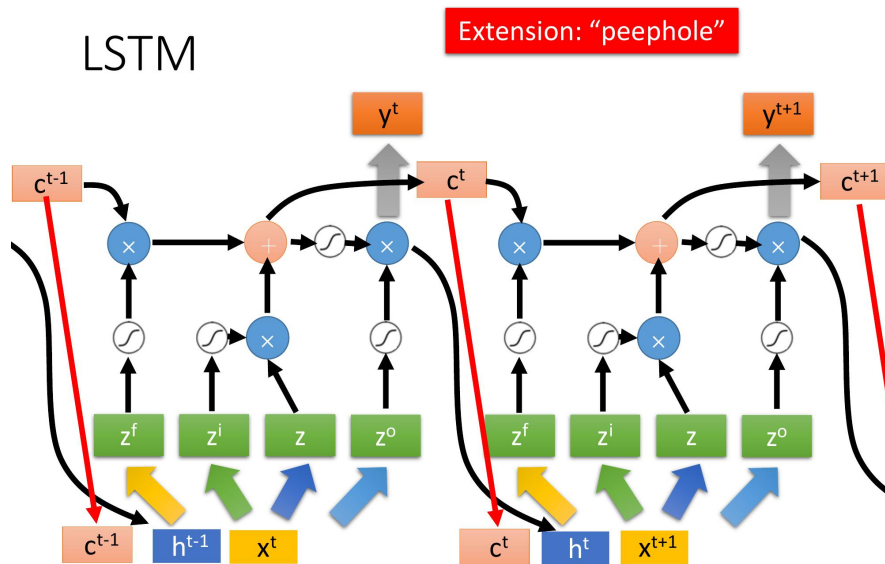
所有的 cell 是可以共同一起被運算的，怎麼一起共同被運算呢

- 把 z^i 先通過 activation function，然後把它跟 z 相乘
 - 這個乘呢，是這個 element-wise 的 product 的意思
- 這個 z^f 也要通過 forget gate 的 activation function 的值，跟之前已經存在 cell 裡面的值，兩者相乘，存回 memory
- 把上述這兩個值加起來（ z^i 跟 z 相乘的值加上 z^f 跟 $c^{(t-1)}$ 相乘的值）
- 把 z^o 通過 activation function 的值，跟相加以後的結果再相乘，就得到最後的 output 的 y

（上述的過程跟單一 cell 都差不多，只是這次是一整排一起做，直接矩陣相乘）

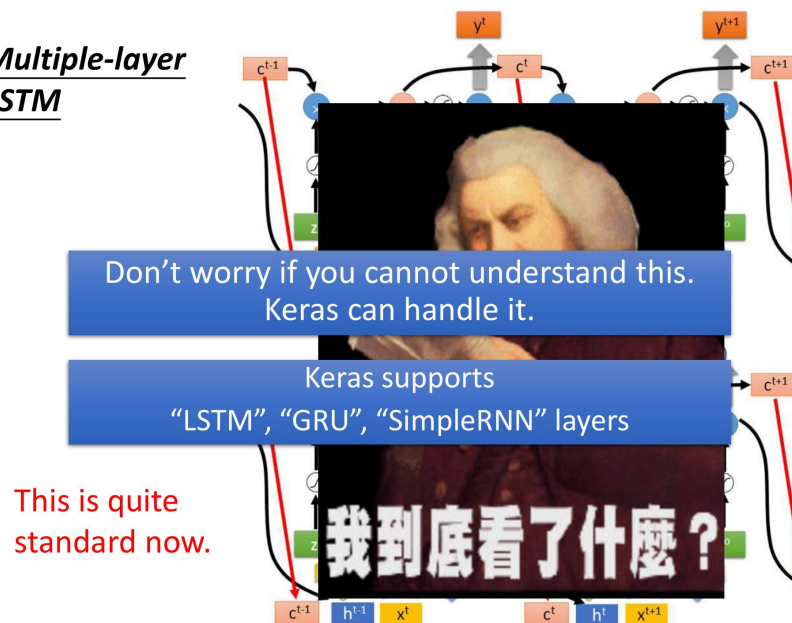
LSTM 的最終型態

LSTM



- 以上只是一個 simplified 的 version
- 真正的 LSTM 會怎麼做呢：
 - 把前一個 hidden layer 的輸出接進來當作下一個時間點的 input（如上圖紅色線，指到的c）
 - 還會加一個東西呢，叫做 "peephole"
 - 把存在 memory cell 裡面的值也拉過來（上圖藍色h）
- 讓人傻眼的複雜模型：
 - 同時考慮了 x , 同時考慮了 h , 同時考慮了 c
 - 把這 3 個 vector 並在一起，乘上4個不同的 transform，得到這4個不同的 vector，再去操控 LSTM
 - 通常不會只有一層，都胡亂疊個五、六層

Multiple-layer
LSTM



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

Keras 支援三種 Recurrent neural networks

- 一個是 LSTM
- 一個是 GRU
 - LSTM 的一個稍微簡化的版本，它只有兩個 gate
 - performance 跟 LSTM 差不多，而且少了 1/3 的參數，所以比較不容易 over-fitting
- 一個是 SimpleRNN