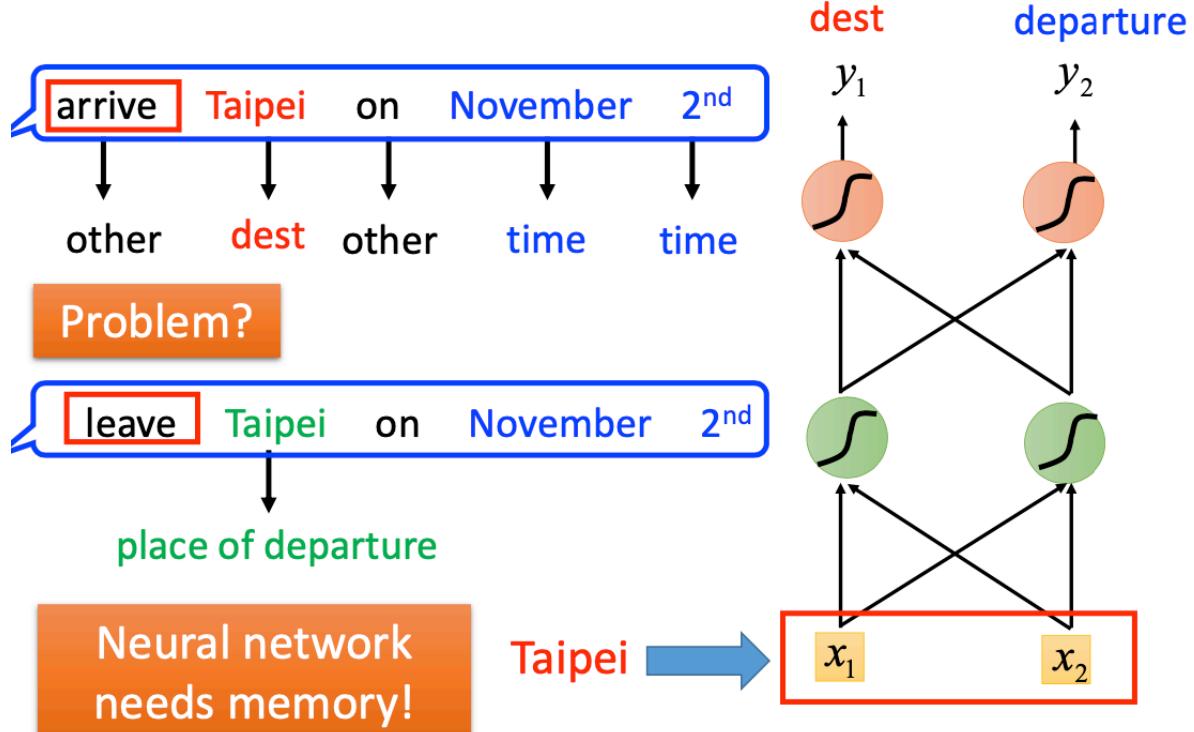


Example Application

将Taipei作为向量的输入，由于RNN网络是有记忆的，因此可以根据输入来预测目的地dest，以及离开的时间

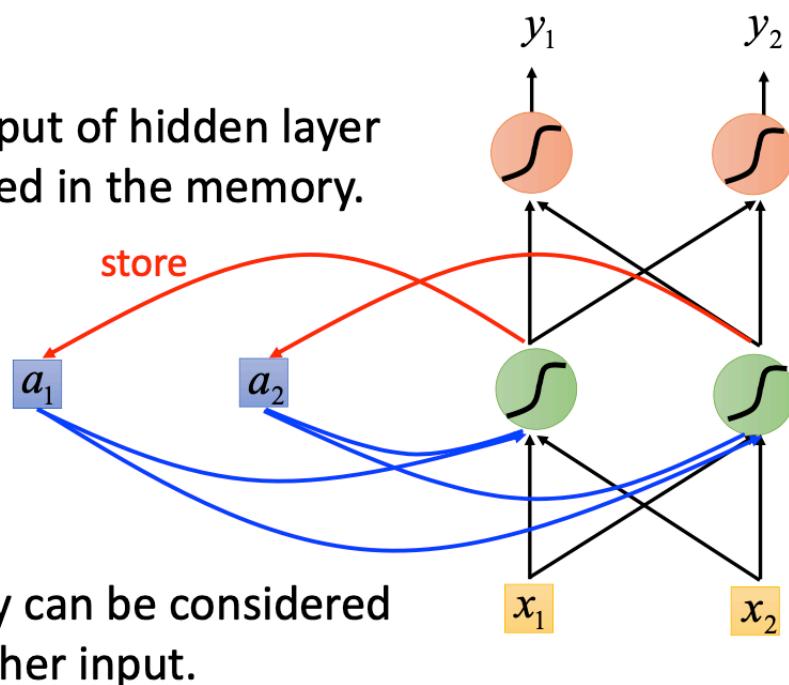
Example Application



RNN Example

RNN是一个有记忆的网络。对于输入的向量 (x_1, x_2) ，hidden neural的输入就包含store和input的神经，本次hidden neural的输出又作为下一次hidden neural的部分输入

The output of hidden layer are stored in the memory.



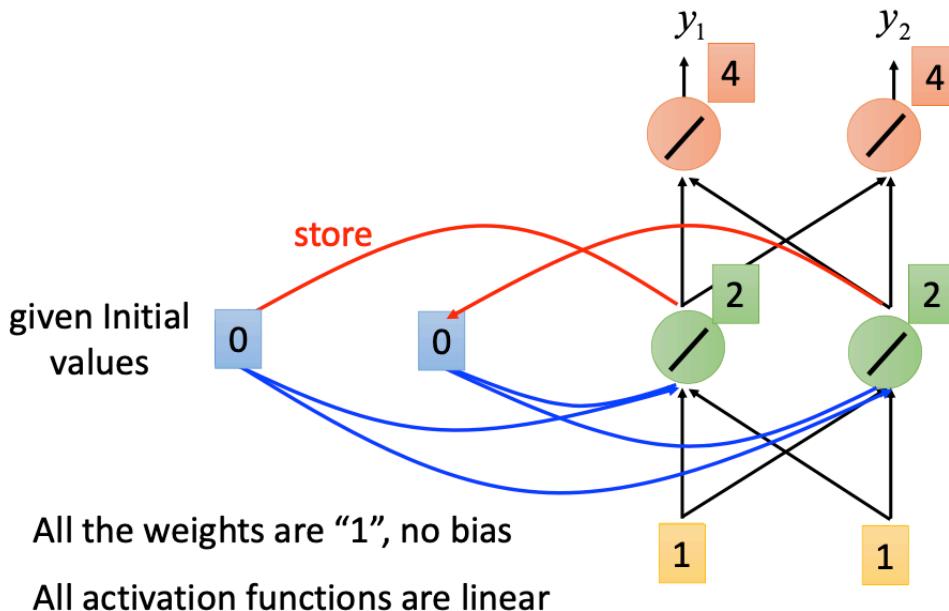
在下图中，对于给定的input sequence为 $(1,1), (1,1), (2,2)$ ，所有的weights都设为1，没有bias项，所有的activation function都是linear的

如果input sequence的第一个输入为 $(1,1)$, store neural的初始值为0，则hidden neural的输出为2，把hidden neural的值放到store neural里（红色箭头），output neural的输出为 $y_1 = 4, y_2 = 4$

Example

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

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

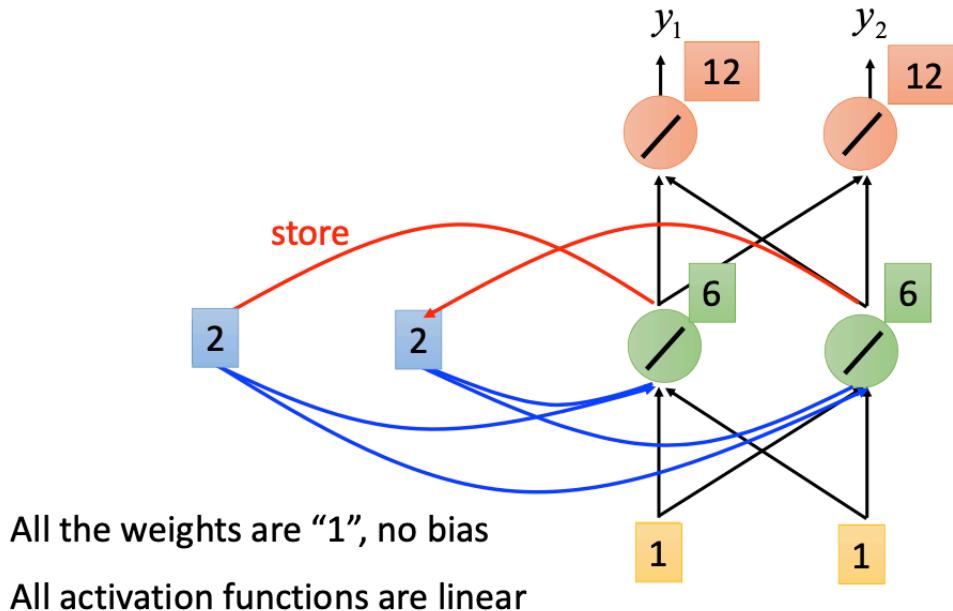


第二个input为 $(1,1)$, store neural的值为 $(2,2)$ ，那么hidden layer的输出为 $2 + 2 + 1 + 1 = 6$ ，再把hidden neural 的输出作为store neural的值，对应的 $y_1 = 2, y_2 = 12$

Example

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

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



第三个input为(2,2), store neural的值为(6,6), 那么hidden neural的输出为 $6 + 6 + 2 + 2 = 16$, 对应的 $y_1 = 32, y_2 = 32$

Example

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

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

Introduction

Basic Concept

下面将在具体叙述RNN网络的原理。

如果input sequence为"arrive Taipei on November 2nd", 那么第一个input记为 $x^1 = \text{arrive}$, 再通过hidden neural计算出 a^1 , 这里需要把 a^1 存入到store neural (作为下一次hidden neural计算的部分输入), 再通过output neural计算出 y^1 ;

那么第二个input记为 $x^2 = \text{Taipei}$, 其中 x^2, a^1 都是hidden neural的输入, 再通过hidden neural计算出 a^2 , 这里需要把 a^2 存入到store neural (作为下一次hidden neural计算的部分输入), 再通过output neural计算出 y^2 ;

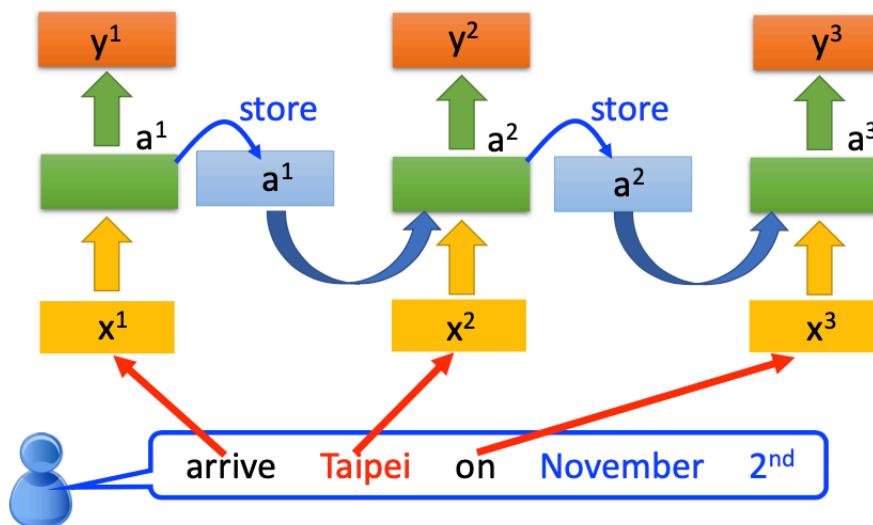
...

RNN的训练并不是多个网络的叠加, 即在下图中, RNN不是有三个不同的network, 只有一个network, 每次更新相对应的input和其他neural的值即可

RNN

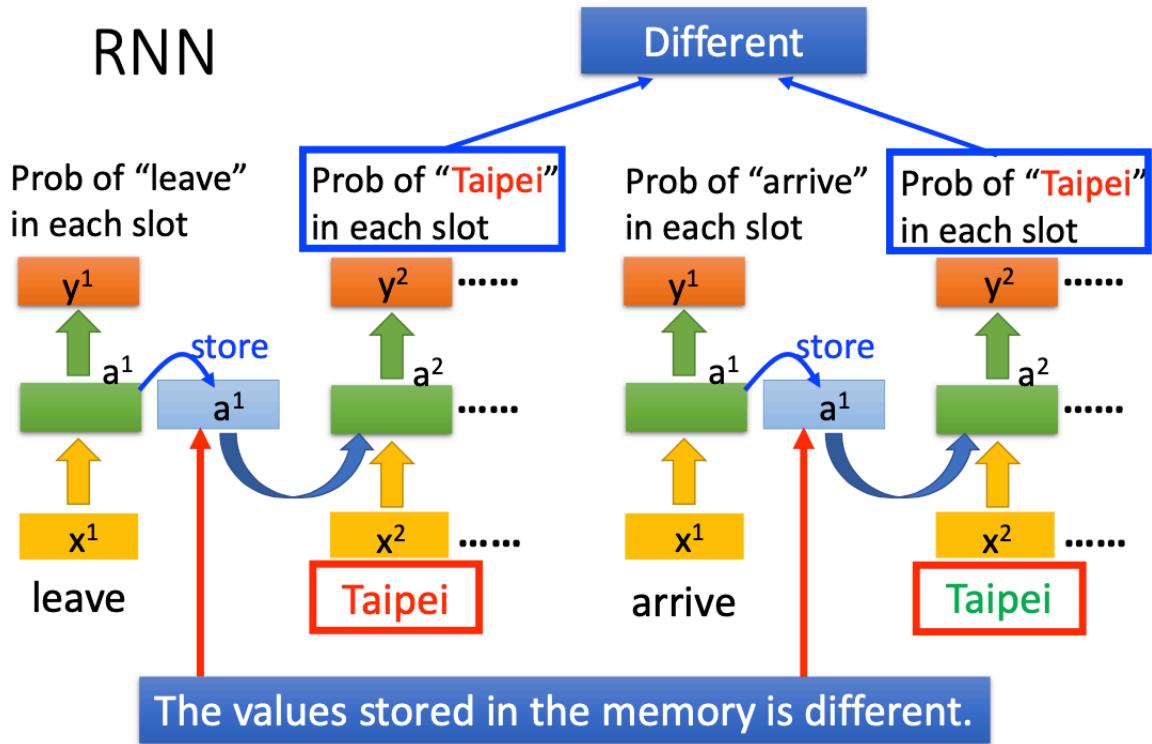
The same network is used again and again.

Probability of “arrive” in each slot	Probability of “Taipei” in each slot	Probability of “on” in each slot
---	---	-------------------------------------



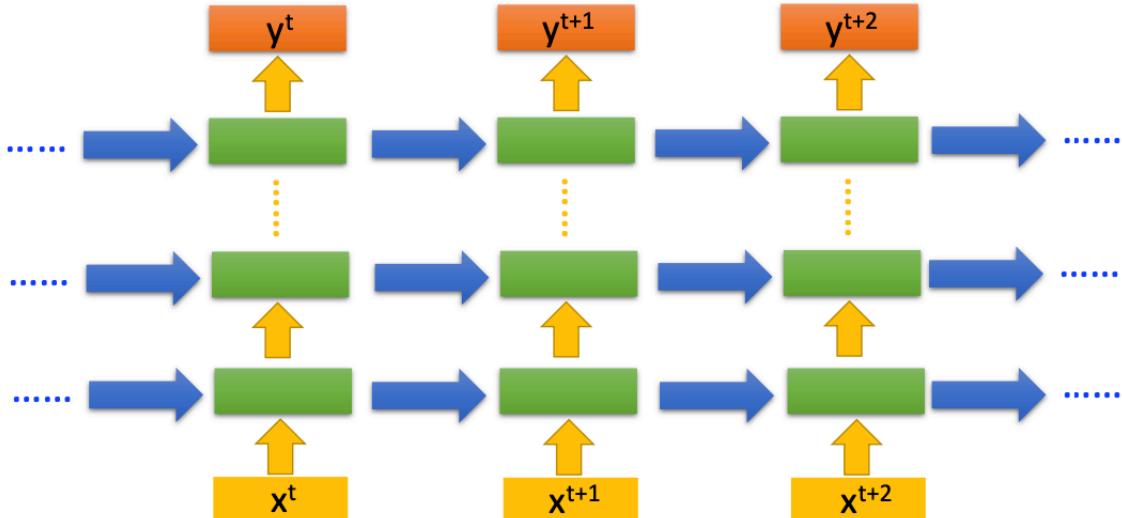
这里还有一个需要注意的点, 即使是input sequence中相同的单词输入(Taipei), 由于input sequence的顺序不同, store neural内存的值也不同, 因此output neural的输出值也不同

RNN



Of course it can be deep ...

和其他的神经网络一样，RNN也可以加深他的网络层次

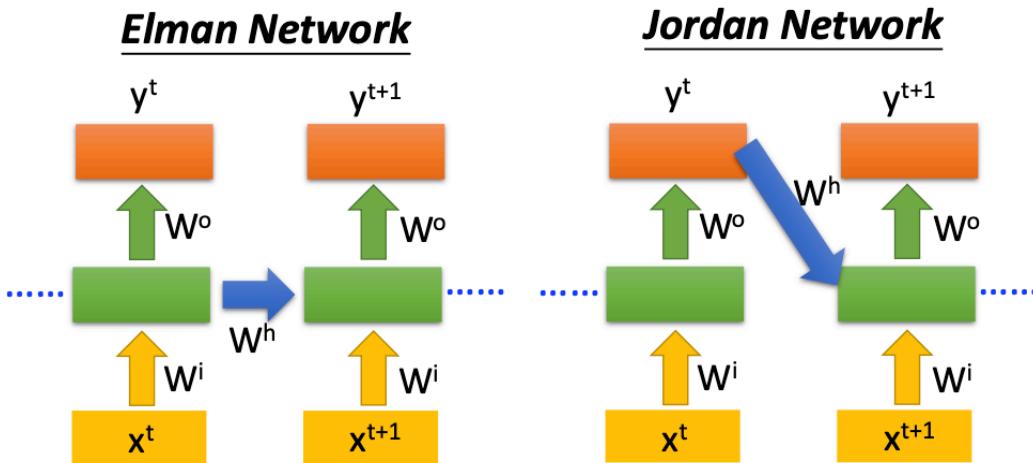


Elman Network & Jordan Network

下面将叙述两种不同的RNN网络。

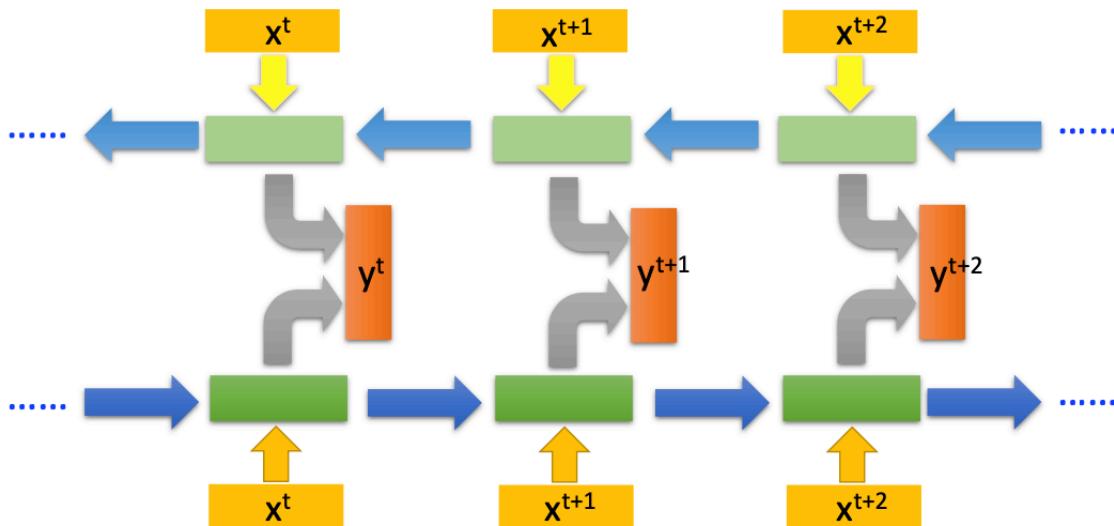
Elman Network类似于前文讲的rnn network，将hidden neural的输出作为store neural的值；而Jordan Network则是将output neural的值存入store neural；

其中Jordan Network的performance比较好，因为Jordan是直接将更清晰的值target存入store neural，而Elman是将学习到的值（hidden neural的输出）存入store neural



Bidirectional RNN

RNN不仅可以是单向的，也可以是双向的，可以train一个正向的network，也同时是train一个逆向的network。如下图所示，input sequence为 $\dots, x^t, x^{t+1}, x^{t+2}, \dots$ ，正向和逆向hidden neural的输出同时作为output neural的输入，从而输出target的值 $\dots, y^t, y^{t+1}, y^{t+2}, \dots$



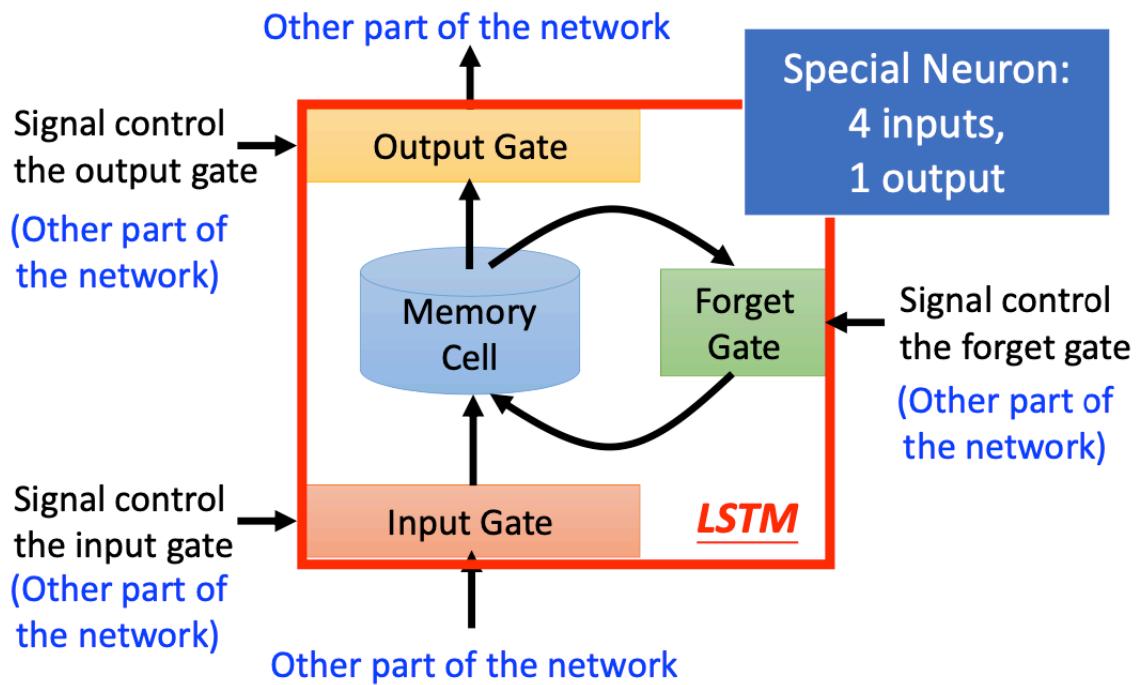
这种双向的RNN对我们的训练是有好处的。对于图中的output y^{t+1} ，如果只是单向RNN，那么 y^{t+1} 只可以看到 x^1, \dots, x^{t+1} 的值；但现在是双向的RNN， y^{t+1} 不仅可以看到 x^1, \dots, x^{t+1} 的值，还可以看到 x^{t+2}, \dots 到句尾的值，即整个input sequence的值都可以被双向的RNN看到，从而得出更好的训练结果

Long Short-term Memory (LSTM)

three gates

下图中的neural有4个input和1个output，一共有三个gate：

- Input Gate：决定其他neural的值是否可以输入这个neural；是否可以输入由neural的学习情况而定
- Forget Gate：什么时候要把memory里面的东西forget，打开表示memory，关闭表示forget；具体什么时候要自己学习
- Output Gate：其他的neural可不可以到这个neural把值取出来，打开的时候才可以取出来，关闭的时候则不可以；打不打开可以自己学习，自己决定



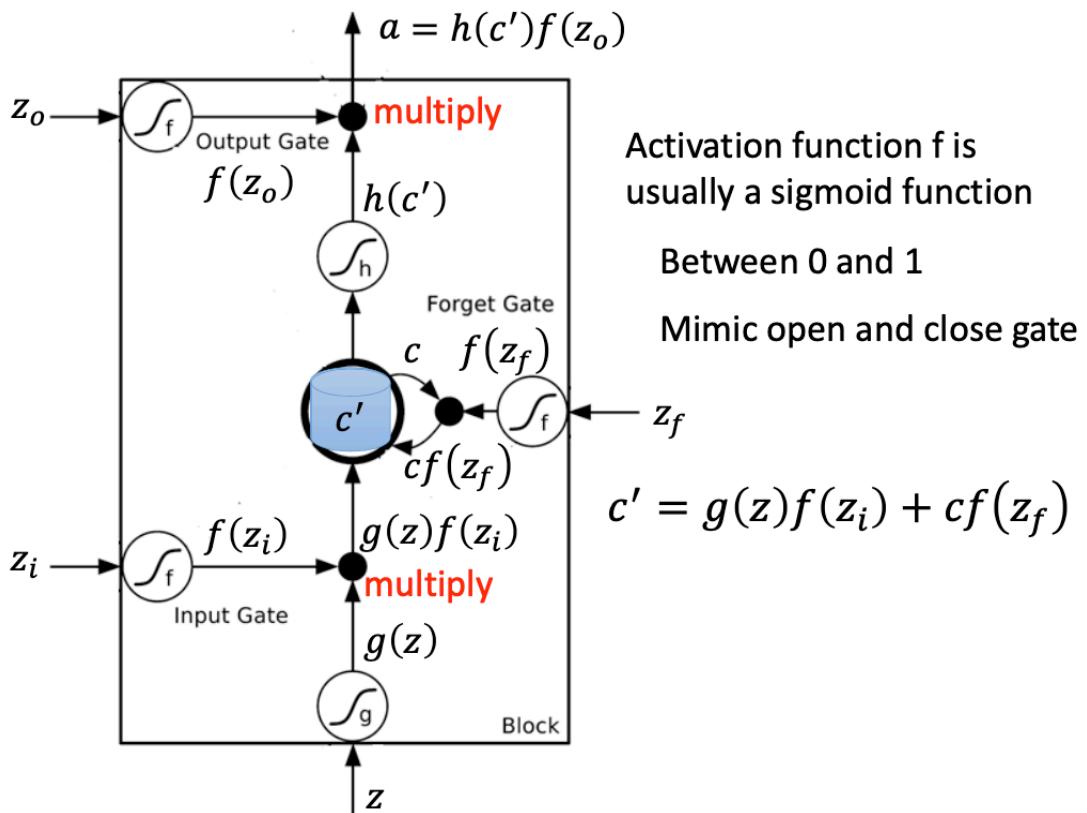
formula

上面只是对LSTM做了一个简要的叙述，下面将开始详细叙述。下图中的activation function f 为sigmoid function，黑色圆点表示multiply； z_i, z_o, z_f 分别操控input、output、forget gate； z 表示外界要存入的值，

如果 $f(z_f)$ 的输出值为1，表示打开forget gate (memory)；输出值为0，表示关闭forget gate (forget)

对于输入的 z 值，经过activation function g 运算后为 $g(z)$ ，相应的 z_i 经过运算得 $f(z^i)$ ， c 为Memory的初始值，当forget gate开启的时候， c' 表示经过一次forget gate打开时进行计算后更新的值

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



memory cell的输出再输入activation function h , 输出 $h(c')$, 即可得出该neural的output为a

$$a = h(c')f(z_0)$$

Example

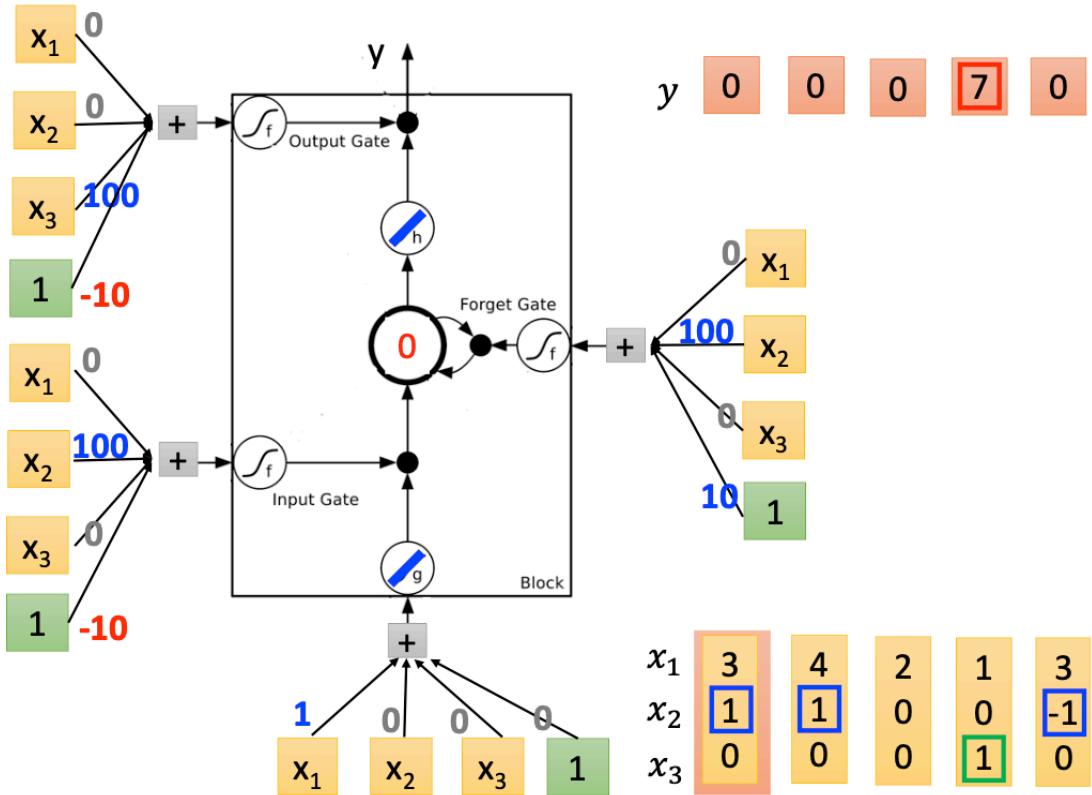
	0	0	3	3	7	7	0	6
x_1	1	3	2	4	2	1	6	1
x_2	0	1	0	1	0	-1	1	0
x_3	0	0	0	0	0	1	0	1
y	0	0	0	0	0	7	0	6

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.

如下图所示, 箭头上的数字表示weight, activation function g 和 h 都是linear function, LSTM初始状态如下

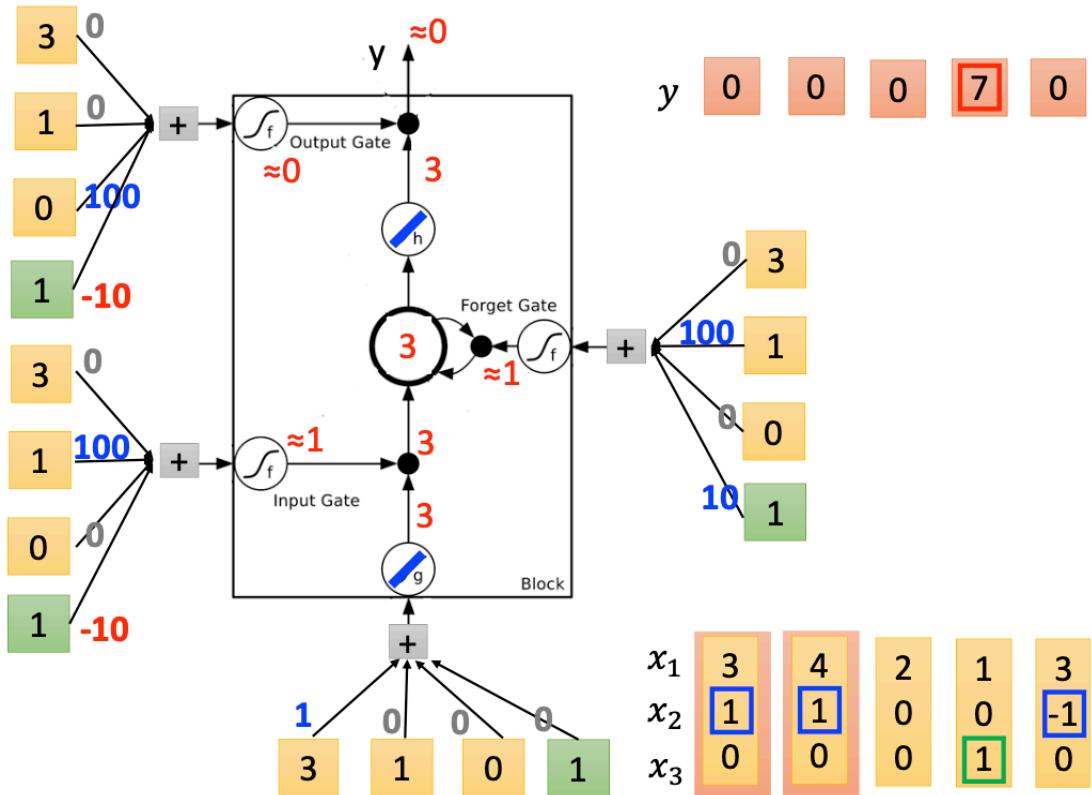


第一个input vector为 $z = (x_1, x_2, x_3, bias)^T = (3, 1, 0, 1)^T$, 则 $g(z) = 3$, Input Gate为 $z_i = 100 \rightarrow f(z_i) \approx 1$, Forget Gate为 $z_f = 110 \rightarrow f(z_f) \approx 1$, Memory Cell内的 $c = 0$, 那么memory cell更新后的值为 c'

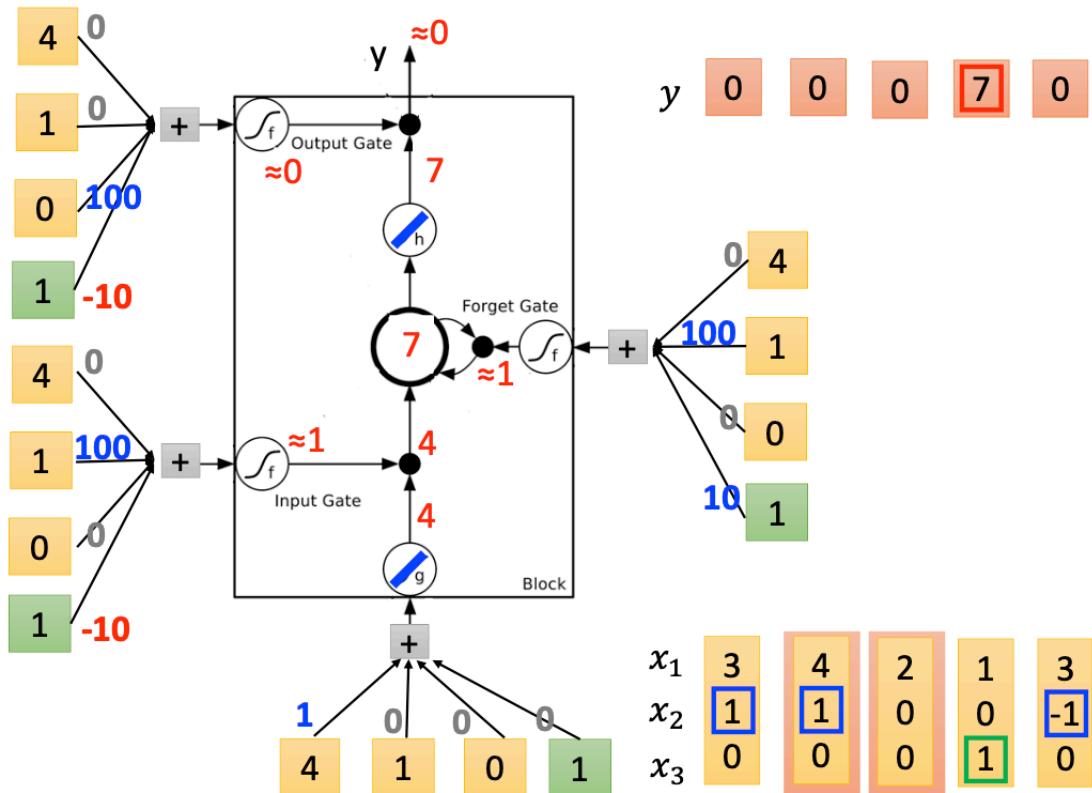
$$\begin{aligned} c' &= g(z)f(z_i) + cf(z_f) \\ &= 3 * 1 + 0 * 1 \\ &= 3 \end{aligned}$$

memory cell的输出再输入activation function h (linear), 输出 $h(c') = 3$, Output Gate为 $z_o = 0 \rightarrow f(z_o) \approx 0$, 即可得出该neural的output为a

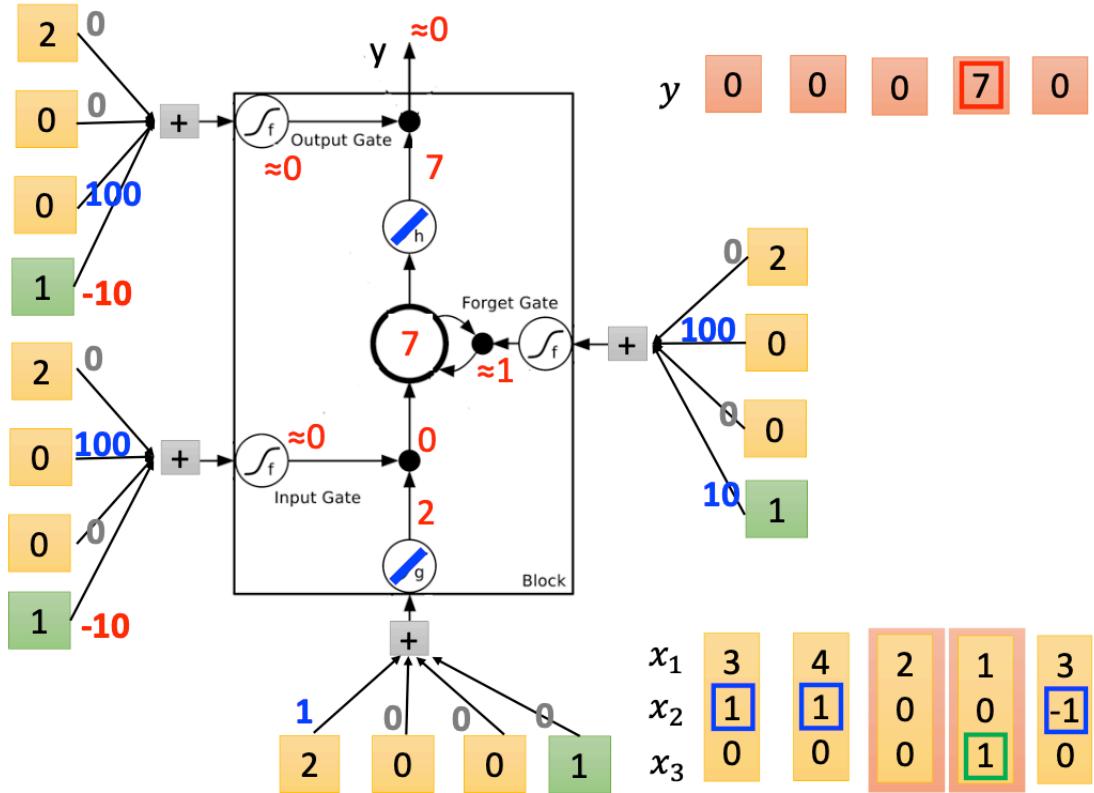
$$a = h(c')f(z_o) = 0$$



第二个input vector为 $z = (x_1, x_2, x_3, \text{bias})^T = (4, 1, 0, 1)^T$, ..., 其中Memory保存了上一次更新的值3, 同理可得 $a=0$



第三个input vector为 $z = (x_1, x_2, x_3, \text{bias})^T = (2, 0, 0, 1)^T$, ..., 同理可得 $a=0$

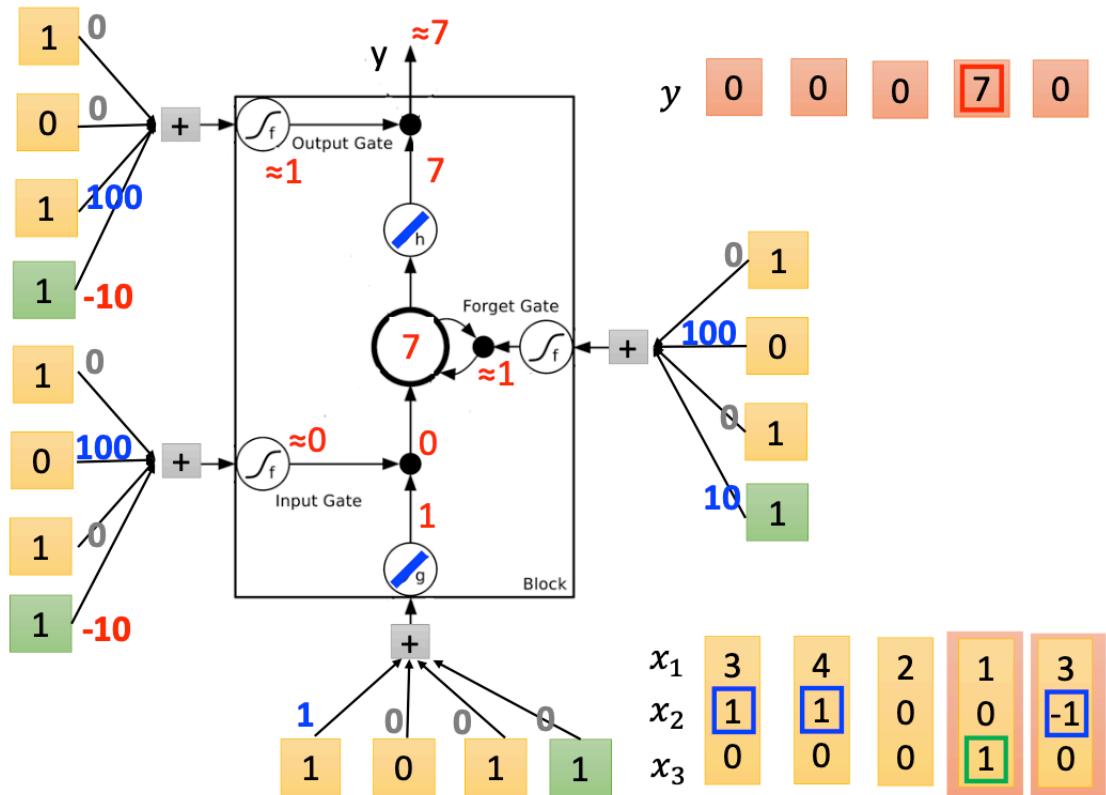


第四个input vector为 $z = (x_1, x_2, x_3, bias)^T = (1, 0, 1, 1)^T$, 则 $g(z) = 1$, Input Gate为 $z_i = 0 \rightarrow f(z_i) \approx 0$, Forget Gate为 $z_f = 10 \rightarrow f(z_f) \approx 1$, Memory Cell内的 $c = 7$, 那么memory cell更新后的值为 c'

$$\begin{aligned} c' &= g(z)f(z_i) + cf(z_f) \\ &= 1 * 0 + 7 * 1 \\ &= 7 \end{aligned}$$

memory cell的输出再输入activation function h (linear), 输出 $h(c') = 7$, Output Gate为 $z_o = 90 \rightarrow f(z_o) \approx 1$, 即可得出该neural的output为 $a=7$

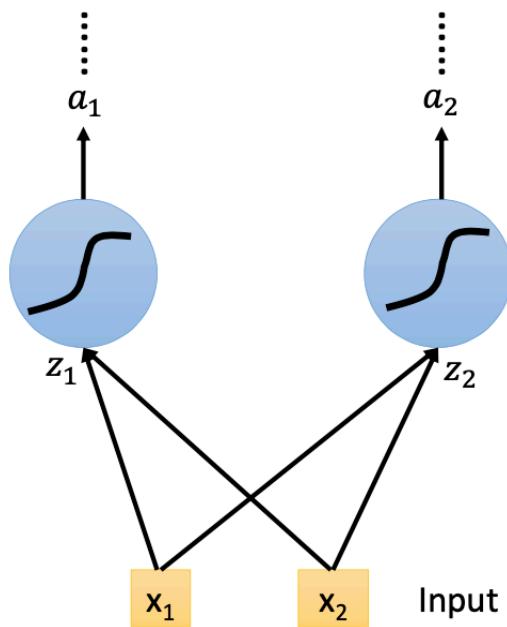
$$a = h(c')f(z_o) = 7$$



Original Network vs LSTM

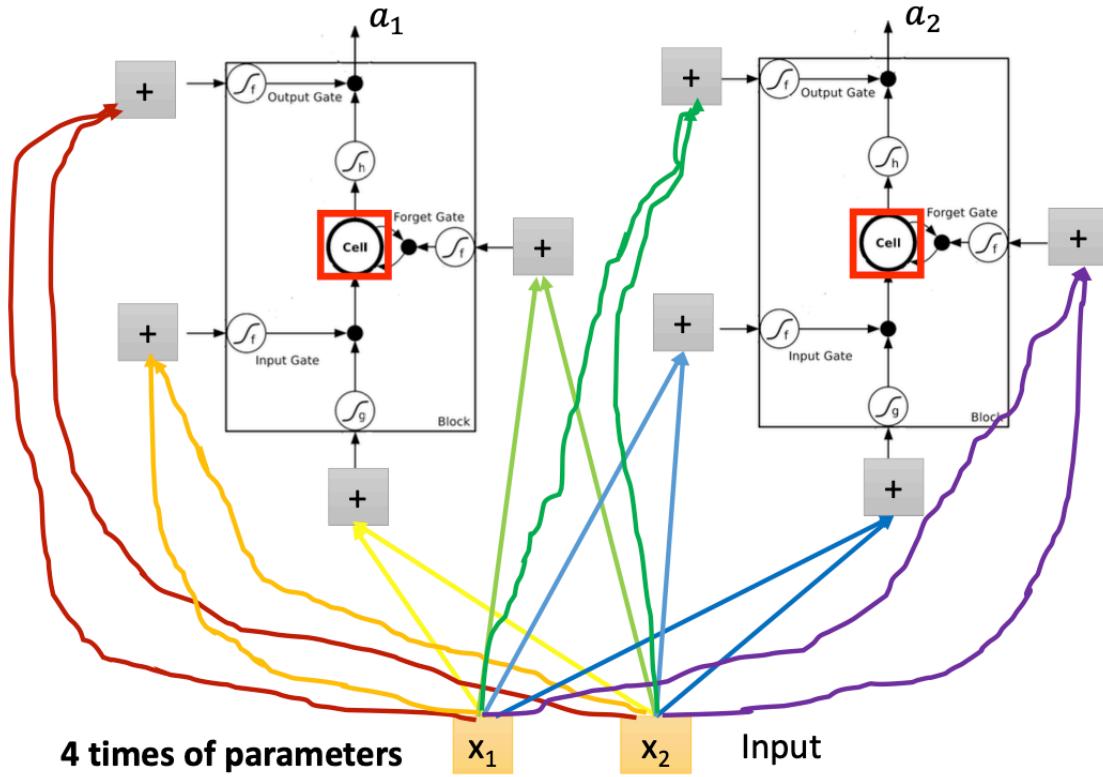
LSTM就是把原来的hidden layer中的neuron替换为LSTM cell

➤Simply replace the neurons with LSTM



将现在的 (x_1, x_2) 乘以不同的weight，从而可以作为4个不同输入，即input、input gate、forget gate、output gate

一般的neural network只需要操控 (x_1, x_2) ，但LSTM需要操控4个不同的input，因此参数量多了4倍

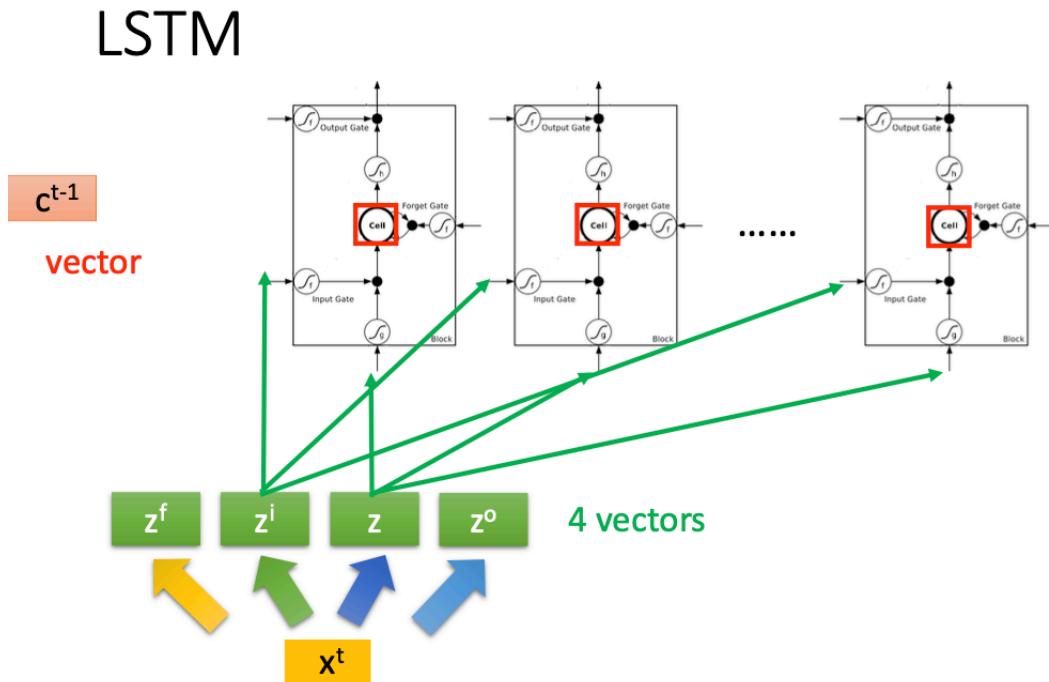


LSTM with neural network

z 的维数则表示LSTM中memory cell (LSTM中红色方框) 的个数, z 的第一维表示第一个LSTM cell的input, 第二维表示第二个LSTM cell的input,

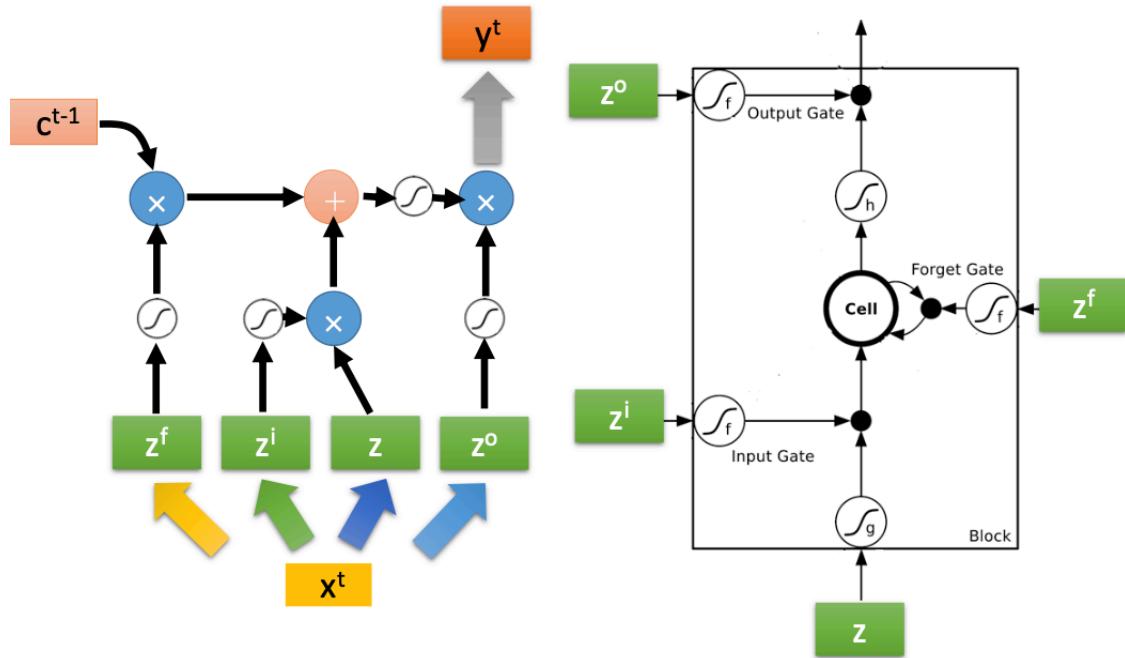
因此一共有4个不同的vector z_f, z_i, z_o , 作为network的input, 操控整个LSTM的运转

输入到第一个LSTM cell的值是vector z_f, z_i, z_o 中的第一维数据, 每个cell的输入值都是不一样的



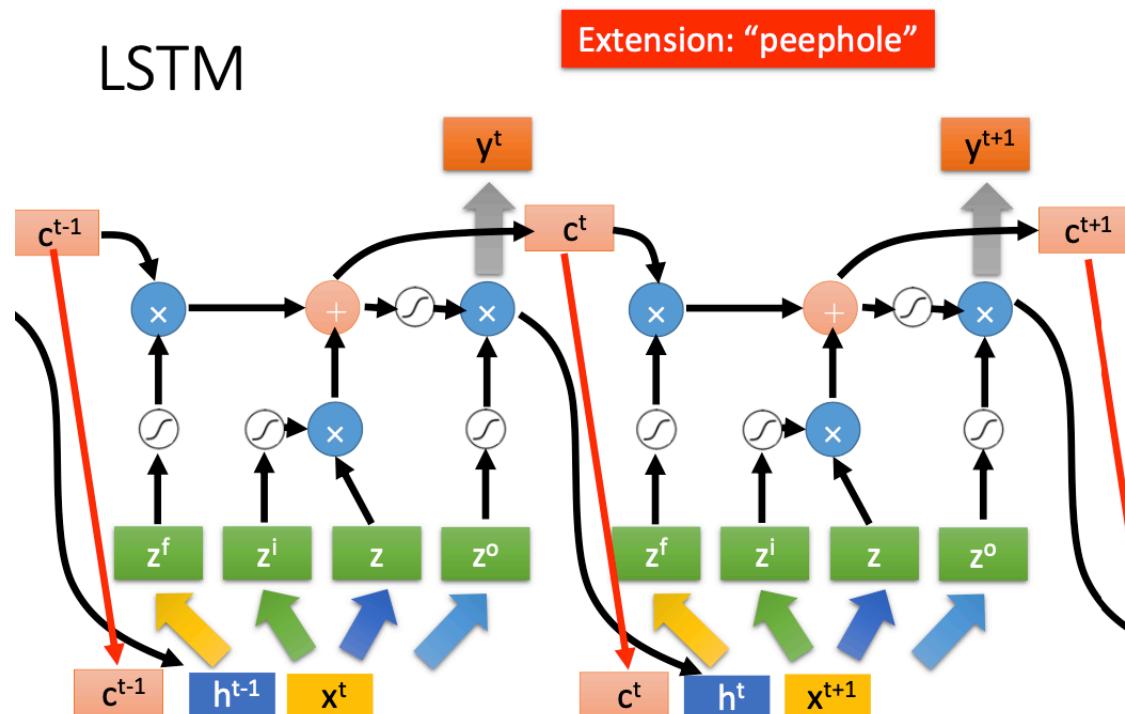
x^t 进行不同的transform, 可以得出4个不同的input z_f, z_i, z_o

首先对于输入的 z 值，经过activation function g 运算后为 $g(z)$ ，相应的 z_i 经过运算得 $f(z_i)$ ， $g(z)$ 再和 $f(z_i)$ 相乘 (element-wise) ， memory cell的初始值为 c^{t-1} ，经过运算可得当前的值 $c^t = g(z)f(z_i) + c^{t-1}f(z_f)$ ，经过activation function h 计算出 $h(c^t)$ ，再和output gate的输出值 $f(z_o)$ 结合，得出当前LSTM cell的输出值 $y^t = h(c^t)f(z_o)$ ，



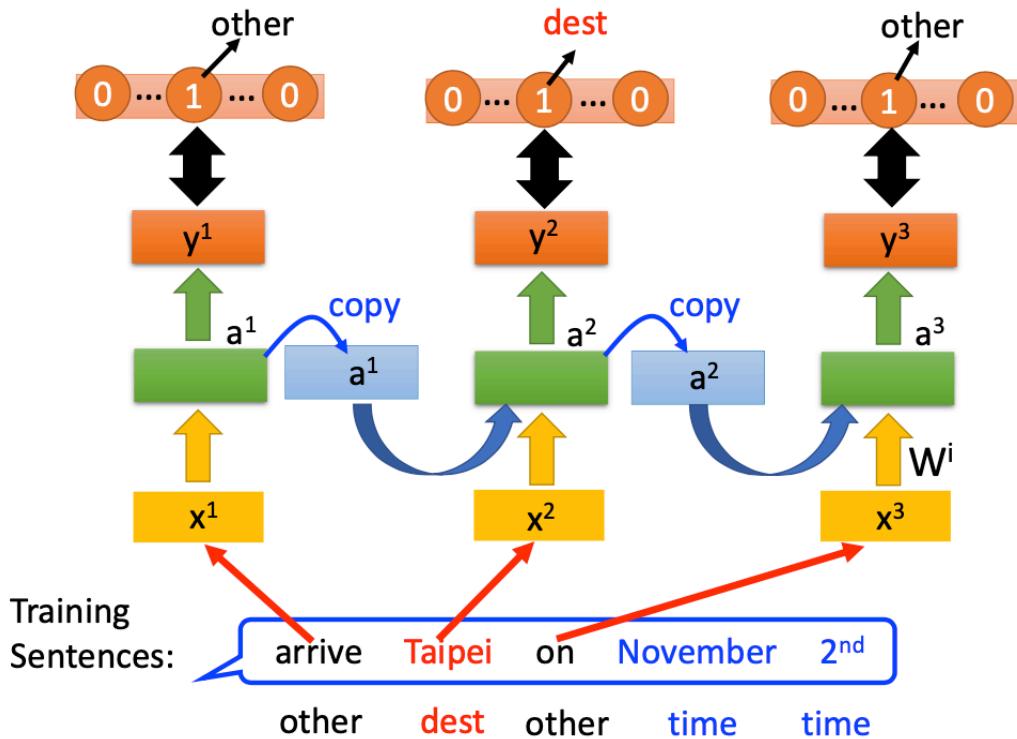
LSTM Extension

下图展示了两个LSTM cell的extension，在时间点t进行的计算，不仅要包括 x^{t+1} ，还要包括上个时间点中lstm cell算出的值，即memory cell的输出 c^t ，以及 c^t 再输入activation function的输出值 $h(c^t)$

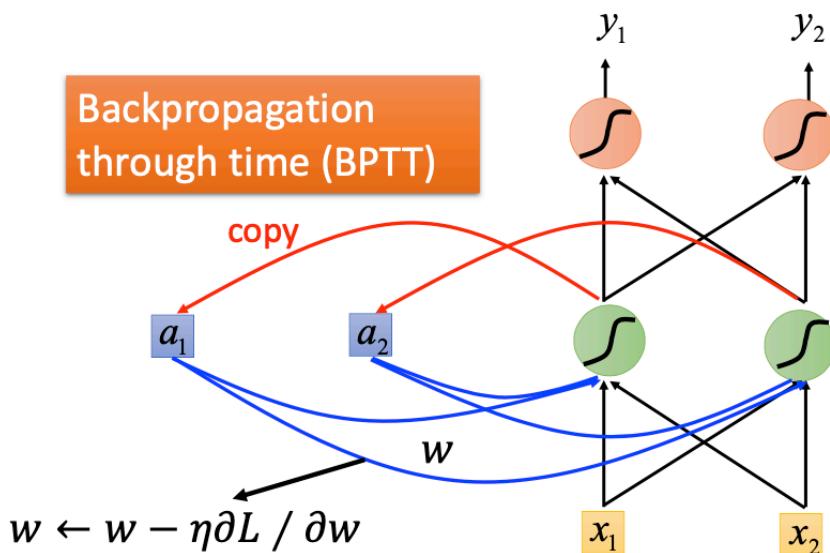


Learning

Training Sentences为"arrive Taipei on November 2nd"，当Taipei输入时，必须要先计算出前一层hidden layer的输出值



RNN的训练也是前向和逆向传播，其中逆向传播算法为Backpropagation through time (BPTT)，也有一个参数更新公式

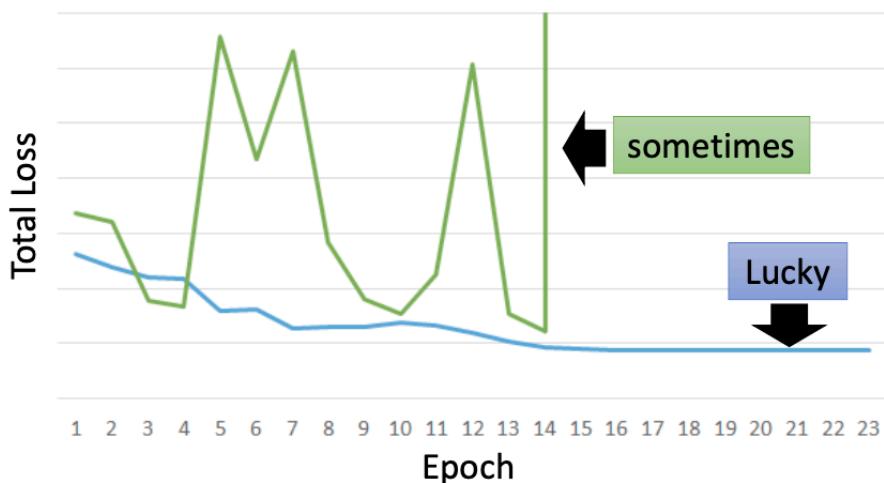


但基于RNN的network并不是很容易训练，

Unfortunately

- RNN-based network is not always easy to learn

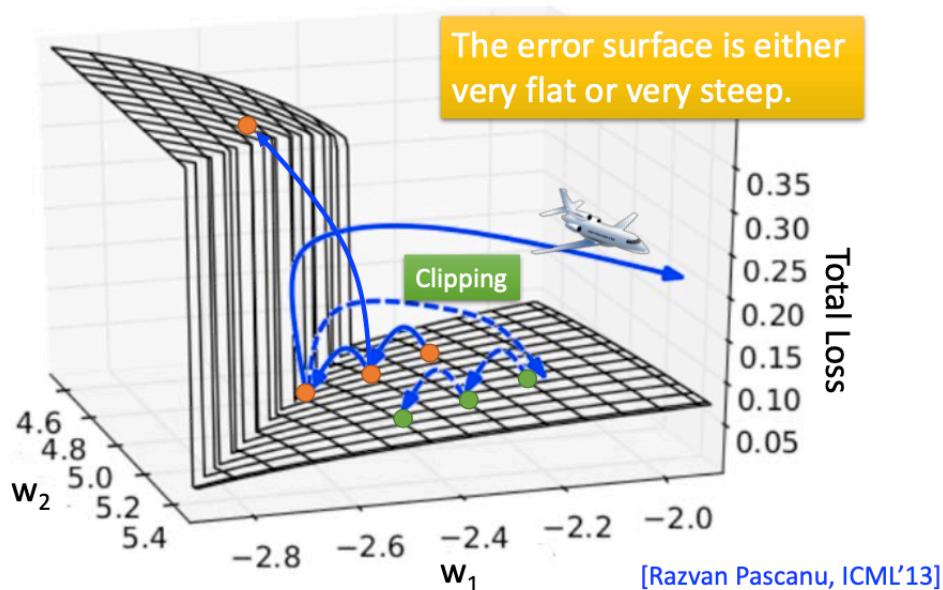
Real experiments on Language modeling



Problem

下图中的纵轴表示total loss， 横轴表示两个weight w_1 和 w_2 。error surface非常的rough， 图像要么很flat， 要么非常steep

The error surface is rough.



假设图中最右方的橙色点表示初始值，由于刚开始error surface比较flat，gradient也比较小，learning rate比较大；经过多次的训练后，如果这时候刚好到达悬崖和平原的交接处，这一次的gradient就超级大，再加上之前很大的learning rate，loss可能就会直接飞出去了

Clipping

改进措施是clipping，当gradient大于15的时候，就看做是15，不再继续增加，这时gradient的移动方向就是图中的蓝色虚线部分，就不会再飞出去了，仍然可以继续做RNN training

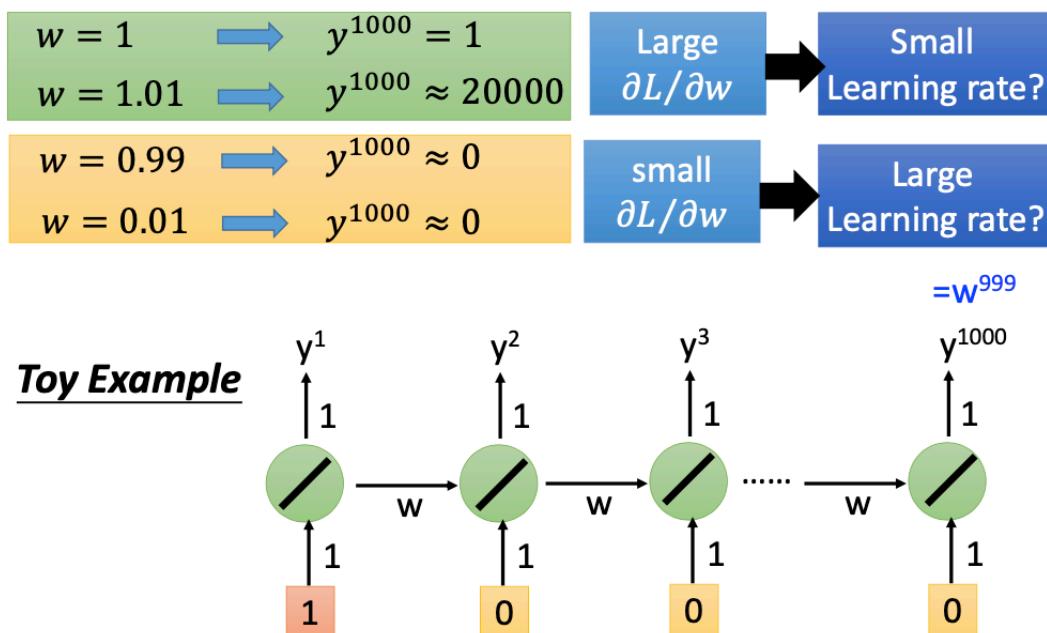
Why?

Q: 由于之前的文章中提到sigmoid function会产生gradient vanishing问题，那么造成RNN训练困难的原因是因为activation function吗？

A: 有学者将sigmoid function 替换为ReLU，发现RNN网络给出了更差的performance，因此这个问题的产生并不是因为activation function的问题

最直观的解决方式是：将某一个参数的gradient进行变化，观察这个变化会对output产生的影响

对于下图中的toy example，input sequence为1,0,0,0,...，hidden layer中的neural都是线性的，在上一个时间点的hidden layer的输出也会作为当前时间点的部分input，设其权重为w，那么如果输入了1000次，那么整个network的output $y^{1000} = w^{999}$



如果 $w = 1$ ，那么对应的输出 $y^{1000} = 1$ ；如果 $w = 1.01$ ，那么对应的输出 $y^{1000} \approx 20000$ ； w 虽然只有小小的变化，但由于蝴蝶效应，对output的影响却很大，因此有很large的gradient $\frac{\partial L}{\partial w}$ ，这时就需要把learning rate设置小一点

如果 $w = 0.99$ ，那么对应的输出 $y^{1000} \approx 0$ ；如果 $w = 0.01$ ，那么对应的输出 $y^{1000} \approx 0$ ； w 虽然有很大的变化，但对output的影响却很小，因此有很small的gradient $\frac{\partial L}{\partial w}$ ，这时就需要把learning rate设置大一点

在很小的区域内，gradient就会有很大的变化，learning rate的变化也很大

因此，RNN不好训练的原因并不是因为activation function的影响，而是由于weight在high frequency地被使用，weight在不同的位置、不同的时间点是会被反复地使用的

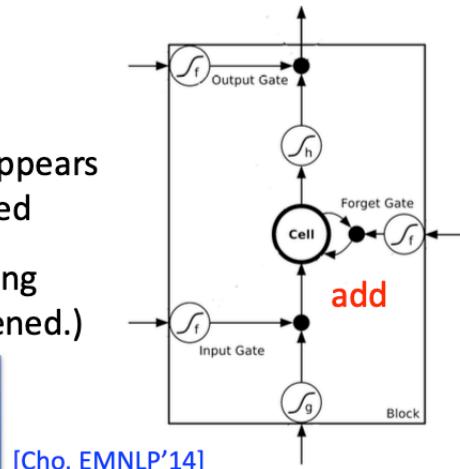
Helpful Techniques

LSTM可以让error surface不那么崎岖，可以把一些很平坦的地方拿掉，使error surface不再有那么平坦的地方，这时可以把learning rate设置得小一些，可以解决gradient vanishing问题

• 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]

传统的RNN和LSTM的区别：

- 传统的RNN通常是直接覆盖的形式，当前时间点的计算结果直接覆盖上一个时间节点的计算结果；
- LSTM通常采用了“累加”的形式，上一个时间节点的计算结果会影响之后的计算结果，并不是直接覆盖。

LSTM可以解决gradient vanishing的原因：

- 当前的input和memory里的值是“累加”的；
- 如果forget gate是打开的状态，上一个时间点的memory会对当前时间点造成影响，这个影响会一直持续下去

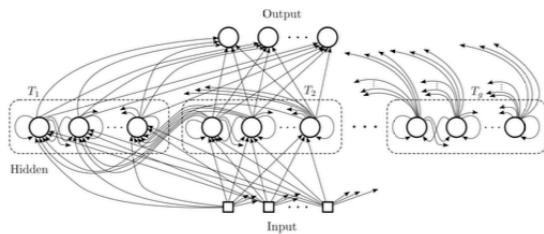
LSTM的第一个版本是为了解决gradient vanishing问题，forget gate是后面才加上去的。因此在LSTM的训练中，要给forget gate非常大的bias，要保证forget gate大部分情况下都是forget的

Gated Recurrent Unit (GRU)：比LSTM更简单，gate的数量只有俩，即forget gate和input gate，这两者有一个联动：

- 当input gate打开时，forget gate关闭，forget掉memory里的值；
- 当forget gate打开时，记住memory里的值，input gate关闭，不再进行输入

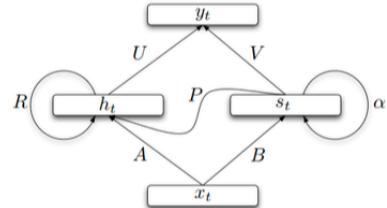
还有一些其他的技术可以解决这个问题，

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

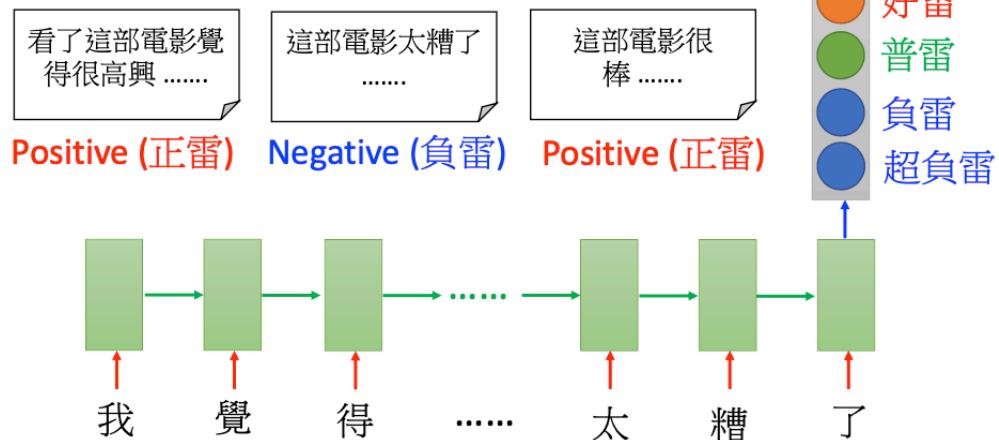
Many to one

input是一些vector的sequence, output可以只是一个vector

比如sentiment analysis, 输入电影评价的sequence, 输出为电影评论的类别

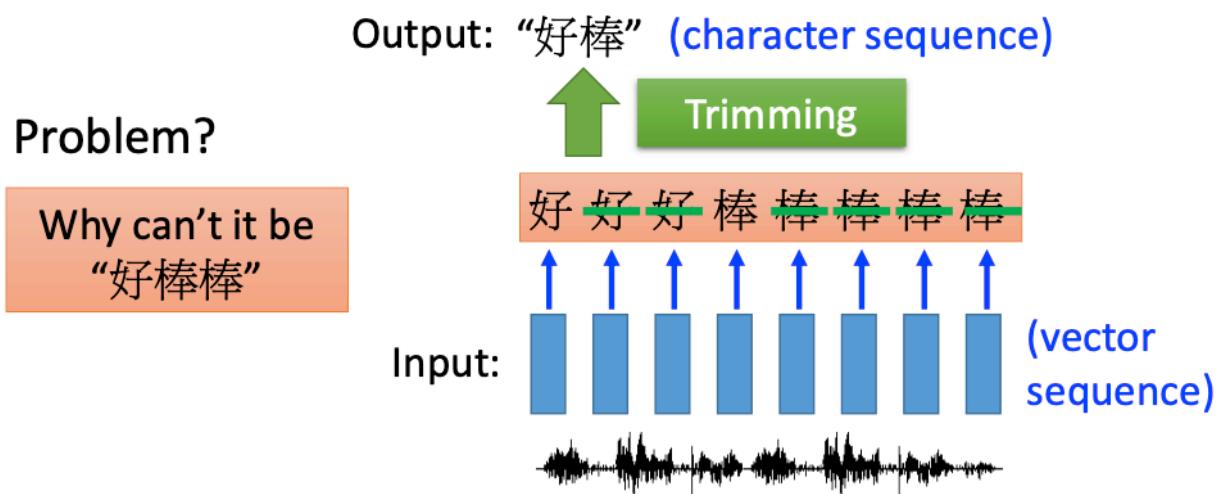
- Input is a vector sequence, but output is only one vector

Sentiment Analysis



- Both input and output are both sequences, but the output is shorter.

- E.g. Speech Recognition



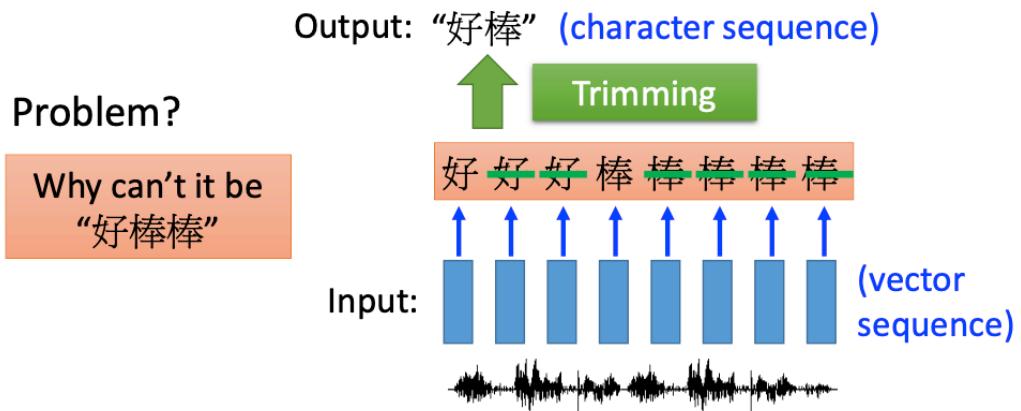
Many to Many (Output is shorter)

input和output也可以都是一个sequence, 比如语音识别speech recognition

对于原来的RNN, 每个vector对应到某一个character, input的vector通常是很短的, 比如0.01s, 因此通常是很多个vector对应到同一个character

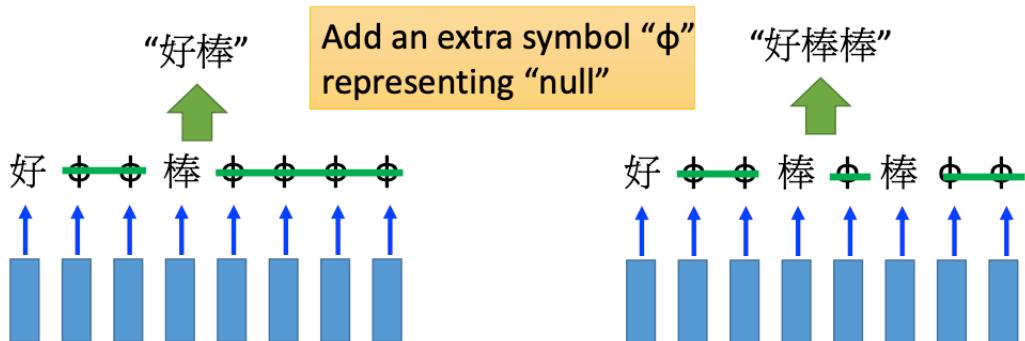
有一种解决方式是trimming, 去掉重复的汉字, 但这时就识别不出“好棒棒”, 只能识别出“好棒”

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



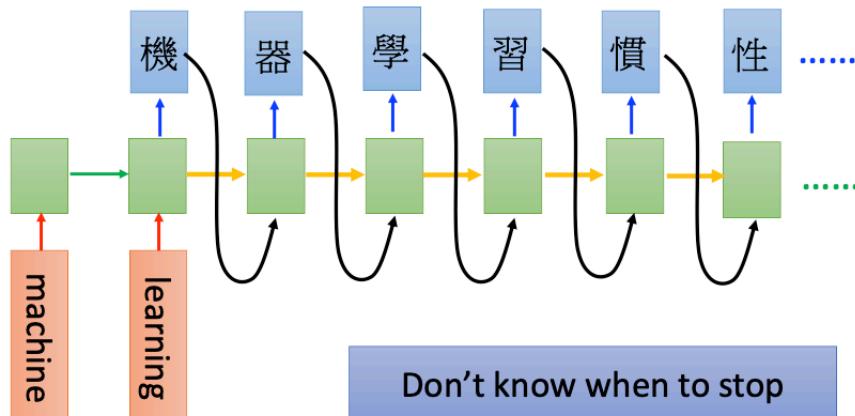
有学者提出了CTC，添加了表示null的符号进来

- Connectionist Temporal Classification (CTC) [Alex Graves, ICML'06][Alex Graves, ICML'14][Hasim Sak, Interspeech'15][Jie Li, Interspeech'15][Andrew Senior, ASRU'15]



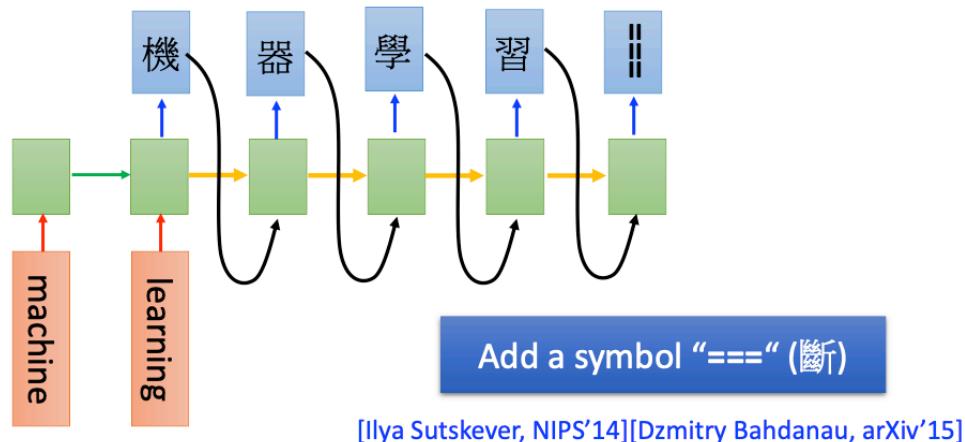
当输入“机”字，根据上一个时间点的memory和“机”，RNN就可以学习到输出为“器”，.....

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



如果不停止的话，rnn可能会一直生成下去，因此要加入一个“断”字，停止生成

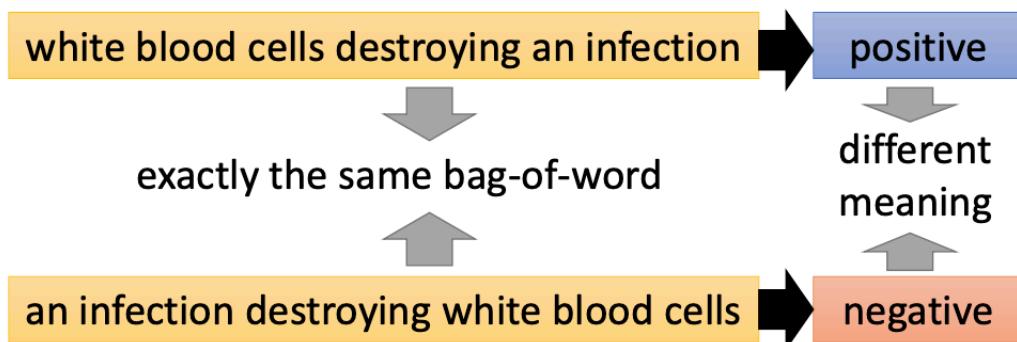
- Both input and output are both sequences with different lengths. → Sequence to sequence learning
- E.g. Machine Translation (machine learning → 機器學習)



Sequence-to-sequence Auto-encoder - Text

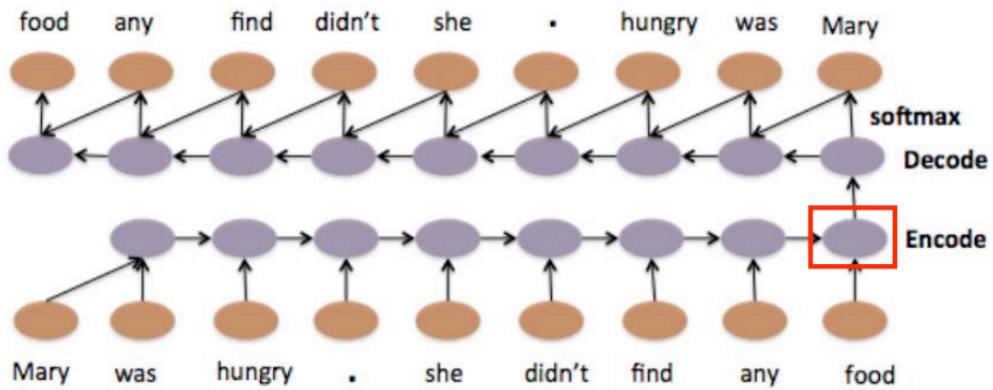
虽然这两句话有相同的单词，但这两句话中的单词却有不同的顺序，因此表示的含义也不一样，一个是positive，一个是negative

- To understand the meaning of a word sequence, the order of the words can not be ignored.

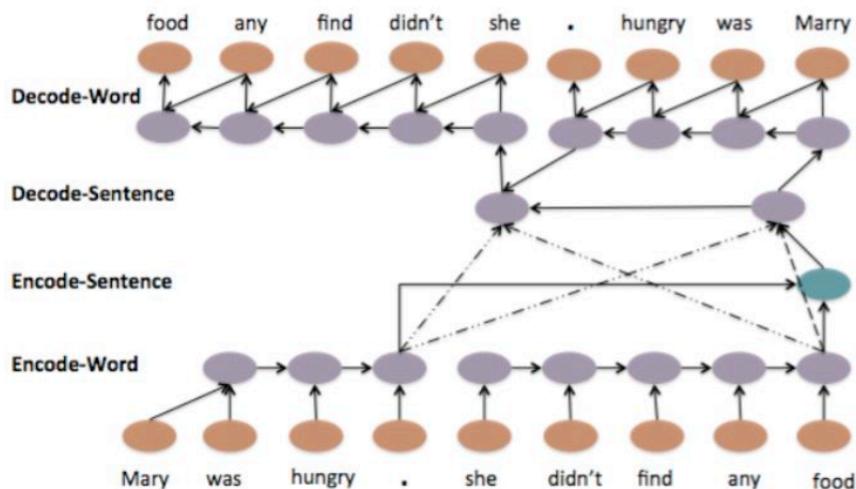


在考虑单词原有顺序的情况下，我们可以通过sequence2sequence auto-encoder来将一个document转化为vector，那么具体怎么做呢？

现在有一个input sequence, “mary was hungry. She didn't find any food”，先通过encoder找到一个vector，再把这个vector输入decoder，找出对应的sequence



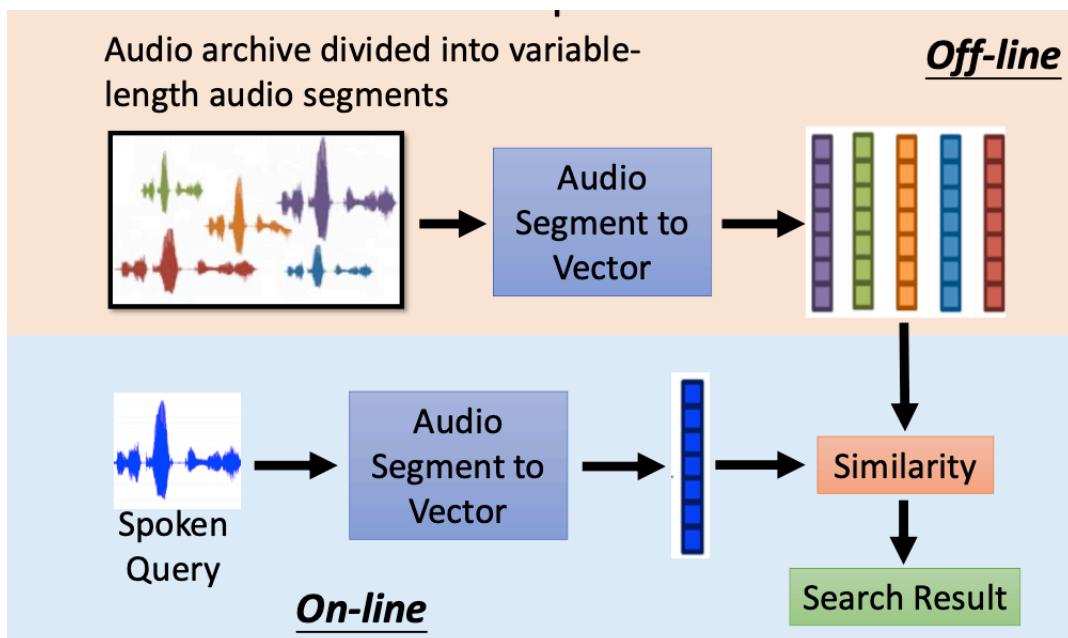
Li, Jiwei, Minh-Thang Luong, and Dan Jurafsky. "A hierarchical neural autoencoder for paragraphs and documents." *arXiv preprint arXiv:1506.01057*(2015).



Sequence-to-sequence Auto-encoder - Speech

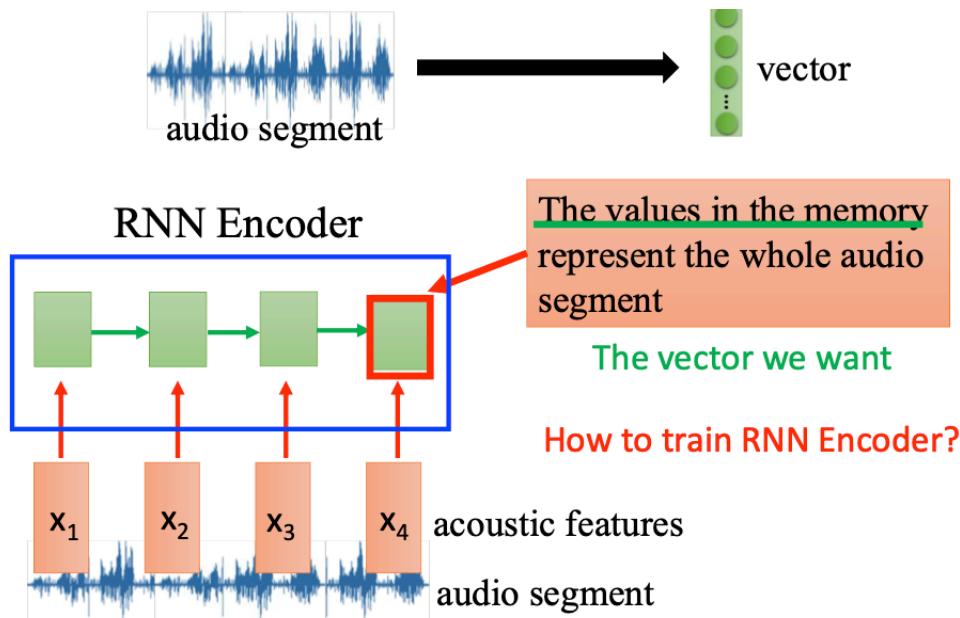
比如有一段演讲语音，现在我想搜索audio中提到了白宫的部分，这时我们就需要做如下工作

把输入的audio segment转化成vector，再把spoken query的audio segment转化成vector，再计算这两者的相似程度，就可以得到搜寻的结果



audio->vector

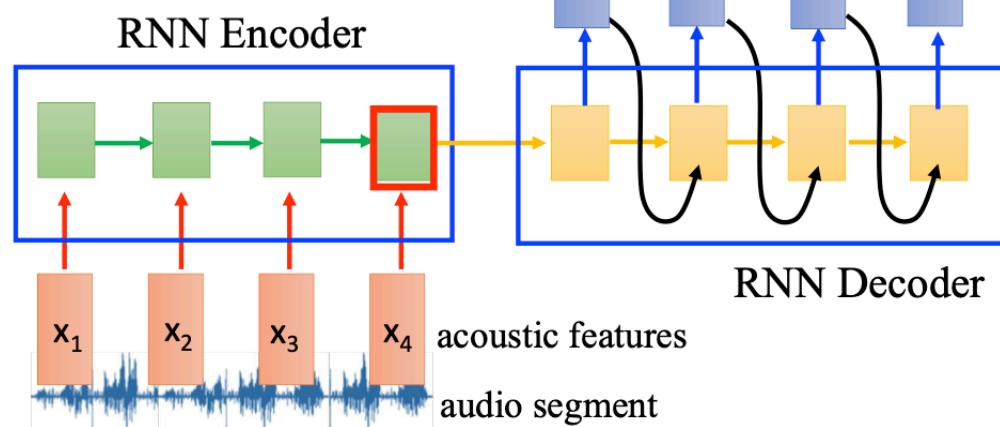
把输入的audio segment分成4部分 x_1, x_2, x_3, x_4 , 输入RNN, 这里的RNN充当了一个encoder的角色, 最后一个时间点在memory里存的值就是整段audio的information



但是这个RNN的encoder是没办法train的, 所以这里加入了RNN的decoder。encoder的memory里存的值作为decoder的第一个input, 产生一个sequence, 这个 y_1 和 x_1 越接近越好, 再根据 y_1 产生 y_2 , y_3 , y_4 。这个训练的target是 y_1, y_2, y_3, y_4 与 x_1, x_2, x_3, x_4 越接近越好

Sequence-to-sequence Auto-encoder

The RNN encoder and decoder are jointly trained.



这个RNN的encoder和decoder是一起 train, 其中任何一个都是没办法单独训练的, 一起train就有一个target

将speech转化为vector之后, 再进行可视化

- Visualizing embedding vectors of the words

