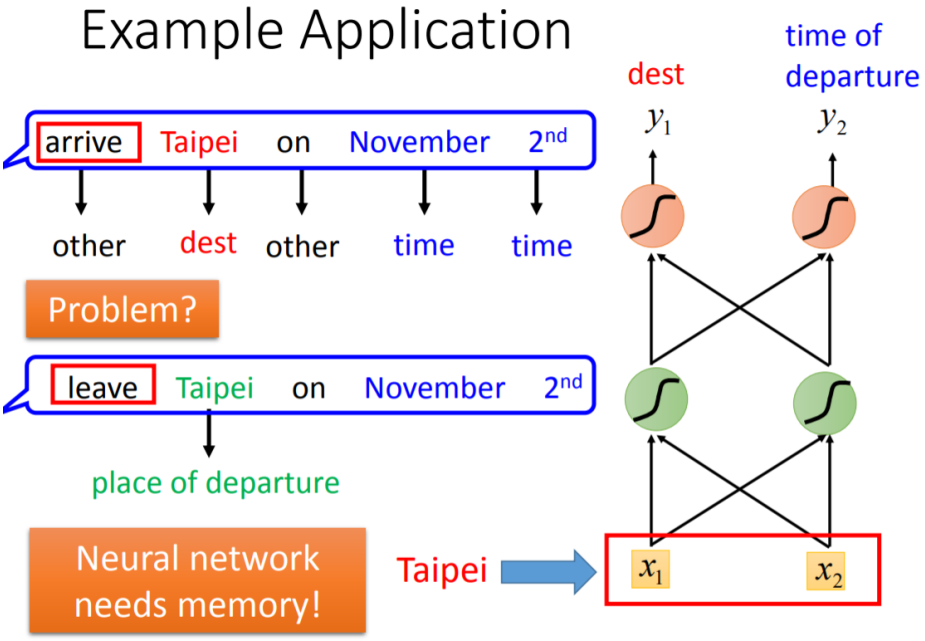
Recurrent Neural Network

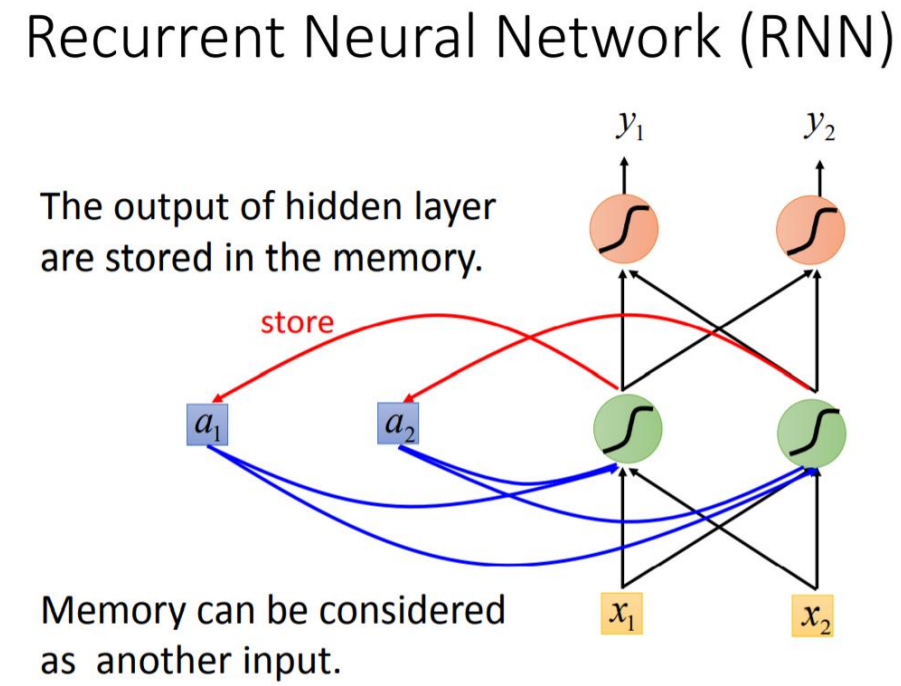
Example Application

将 Taipei 作为向量输入，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



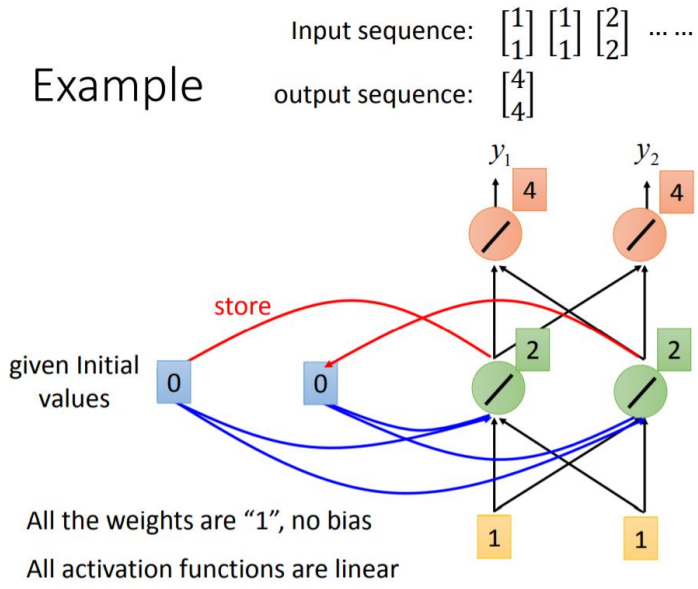
RNN Example

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 neural，本次 hidden neural 的输出又作为下一次 hidden neural 的部分输入

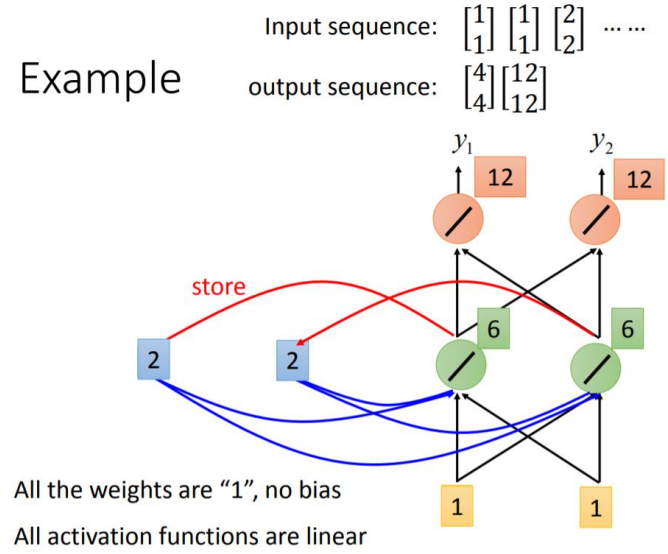


在下图中，对于给定的 input sequence 为$\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\2\end{bmatrix}$,所有的 weights 都为 1，没有 bias 项，所有的 activation function 都是 linear 的

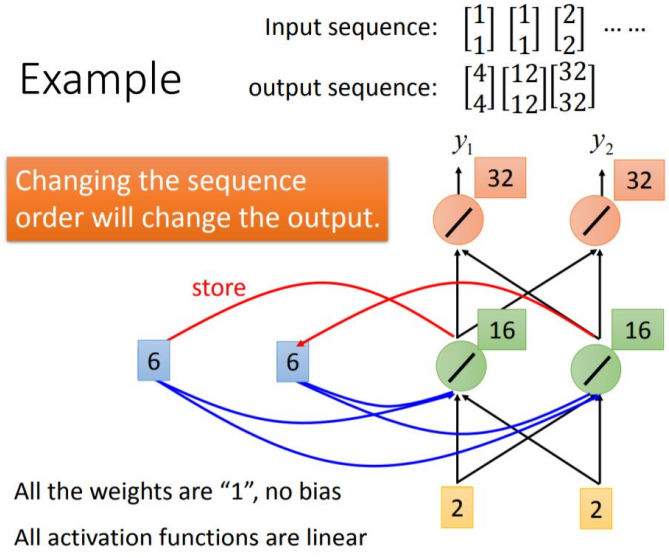
如果 input sequence 的第一个输入$\begin{bmatrix}1\\1\end{bmatrix}$,store neural 的初始值为 0，则 hidden neural 的输出为 2，把 hidden neural 的输出储存到 store neural 里(红色箭头)，output neural 的输出为$y_1 = 4, y_2 = 4$



第二个 input 为$\begin{bmatrix}1\\1\end{bmatrix}$,store neural 的值为$\begin{bmatrix}2\\2\end{bmatrix}$,那么 hidden layer 的输出 2+2+1+1=6，再把 hidden neural 的输出作为 store neural 的值，对应的$y_1 = 12, y_2 = 12$



第三个 input 为$\begin{bmatrix}2\\2\end{bmatrix}$,store neural 的值为$\begin{bmatrix}6\\6\end{bmatrix}$,那么 hidden layer 的输出 2+2+6+6=16，再把 hidden neural 的输出作为 store neural 的值，对应的$y_1 = 32, y_2 = 32$



Introduction

Basic Concept

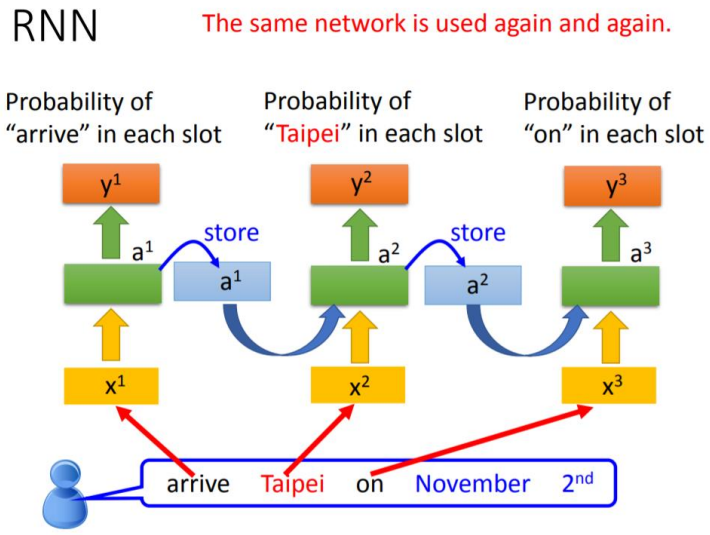
下面将在具体叙述 RNN 神经网络的原理

如果 input sequence 为 “arrive Taipei on November 2nd”，那么第一个 input 记为$x^1 = arrive$,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;

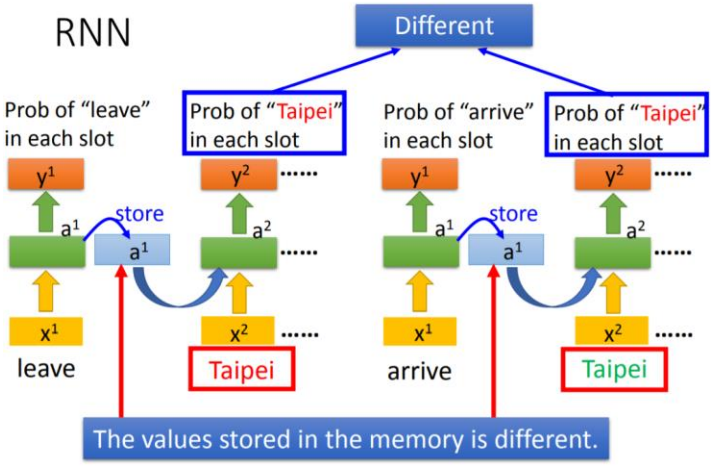
那么第二个 input 记为$x^2 = Taipei$,其中x^2, a^1都是 hidden neural 的输入，再通过 hidden neural 计算出a^2，这里需要把a^2存入到 store neural(作为下一次 hidden neural 计算的部分输入)，再通过 output neural 计算出y^2;

……..

RNN 的训练并不是多个神经网络的叠加，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



注意： 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 内存的值不同，因此 output neural 的输出值也不同

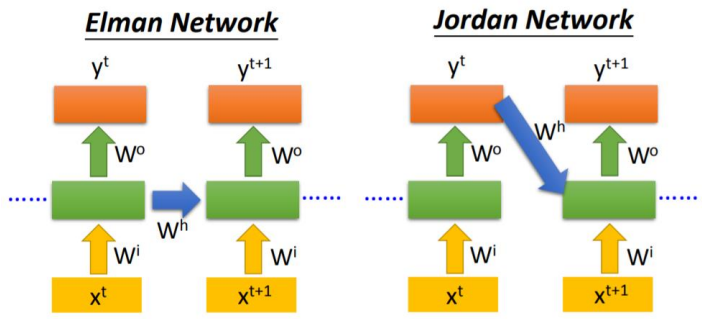


Elman Network & Jordan Network

下面将叙述两种不同的 RNN 神经网络

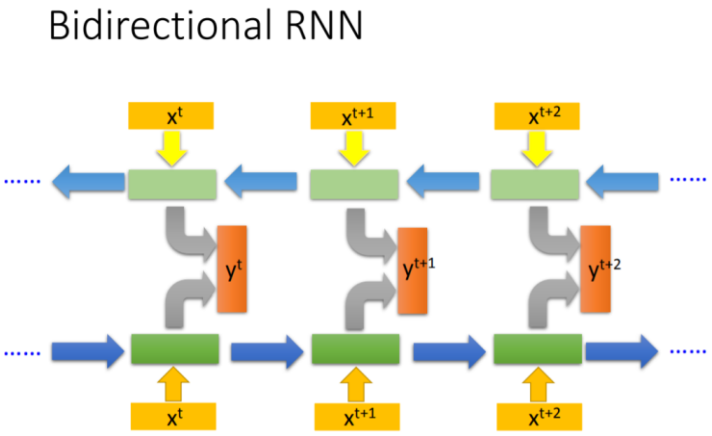
Elman Network 类似于前文讲的 RNN Network，将 hidden neural 的输出作为 store neural 的值；而 Jordan Network 则是将 output neural 的值存入 store neural;

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….,x^t, x^{t+1}, x^{t+2}, ……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…, y^t, y^{t+1}, y^{t+2}, ……



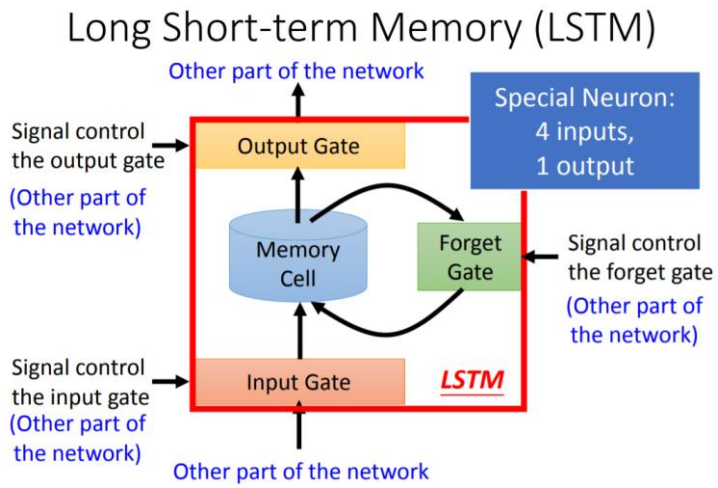
这种双向的 RNN 对我们的训练是有好处的，对于图中的 output y^{t+1}，如果只是单向的 RNN，那么y^{t+1}只可以看到$x^1, \ldots . x^{t+1}$的值；但现在是双向的 RNN，y^{t+1}不仅可以看到$x^1, \ldots . x^{t+1}$的值，还可以看到$x^{t+2}, \ldots$到句尾的值，即整个 input sequence 的值都可以被双向的 RNN 看到，从而得出更好的训练结果

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 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



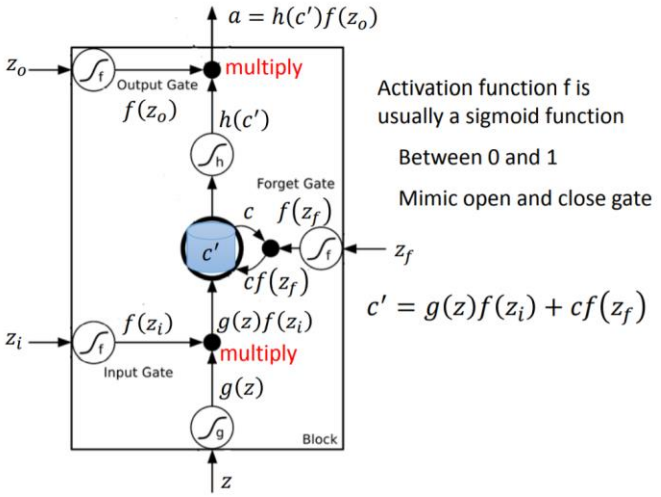
Formula

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 multiple；z_i, z_o, z_f分别操控 input、output、forget gate；z 表示外界存入的值

如果$f(z_f)$的输出值为 1，表示 open forget gate；输出值 0，表示 close forget gate

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

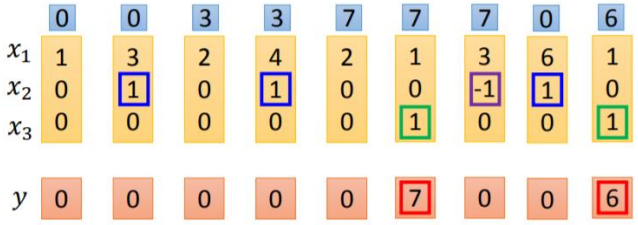
$c' = g(z)f(z_i) + cf(z_f)$



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

$a = \hbar(c')f(z_o)$

Example

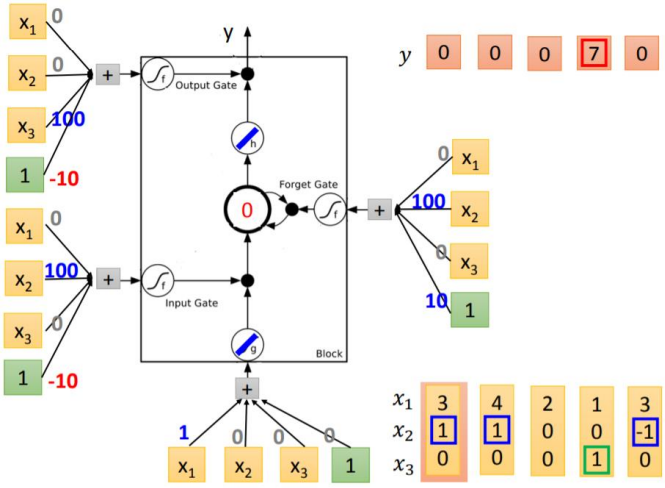


When x_2 = 1, add the numbers of x_1 into the memory

When x_2 = -1, reset the memory

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

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 $z_f = 100 \rightarrow f(z_f) \approx 1$, Memory Cell 内的$c = 0$,那么 memory cell 更新后的值$c'$

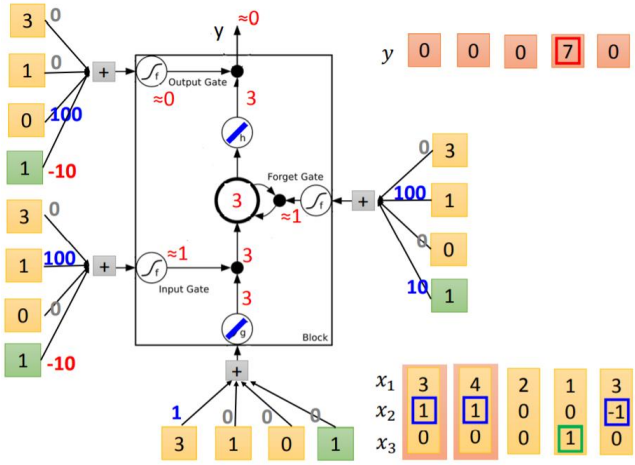
$c' = g(z)f(z_i) + cf(z_f)$

$= 3 * 1 + 0 * 1$

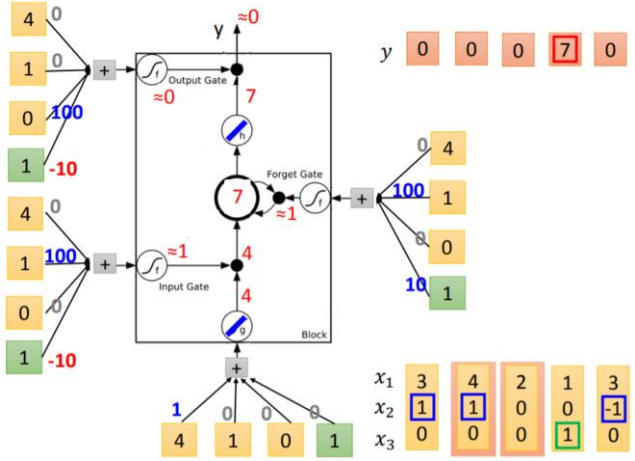
$= 3$

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

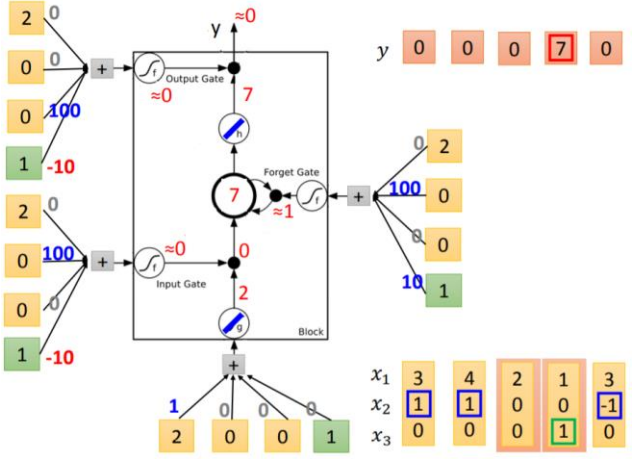
$a = h(c')f(z_o) = 0$



第二个 input vector 为$z = (x_1, x_2, x_3, bias)^T = (4,1,0,1)^T$，…….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$z = (x_1, x_2, x_3, bias)^T = (2,0,0,1)^T$，……，同理可得 a=0



第四个 input vector 为$z = (x_1, x_2, x_3, bias)^T = (1,1,0,1)^T$,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$c'$

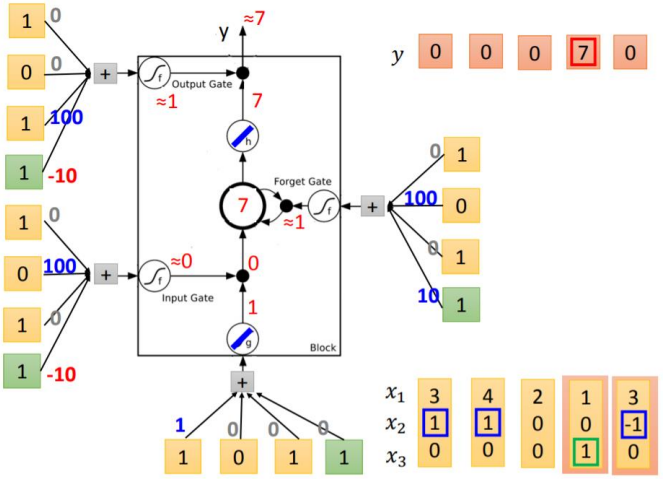
$c' = g(z)f(z_i) + cf(z_f)$

$= 1 * 0 + 7 * 1$

$= 7$

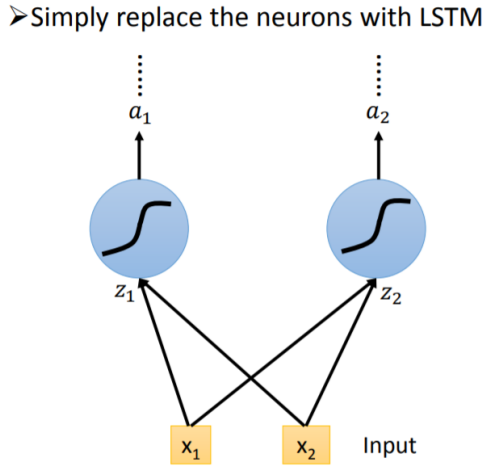
Memory cell 的输出再输入 activation function h(linear),输出$h(c') = 7$，Output Gate 为$z_o = 90 \rightarrow f(z_o) \approx 1$,即可得出该 neural 的 output 为 a=7

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

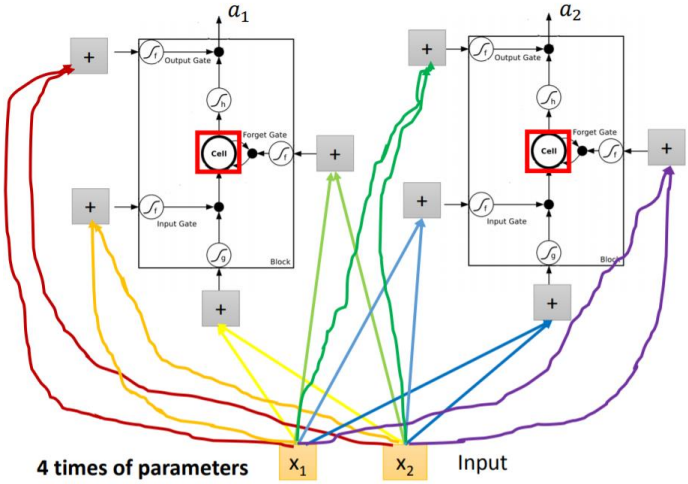


Original Network vs LSTM

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



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 neuron 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

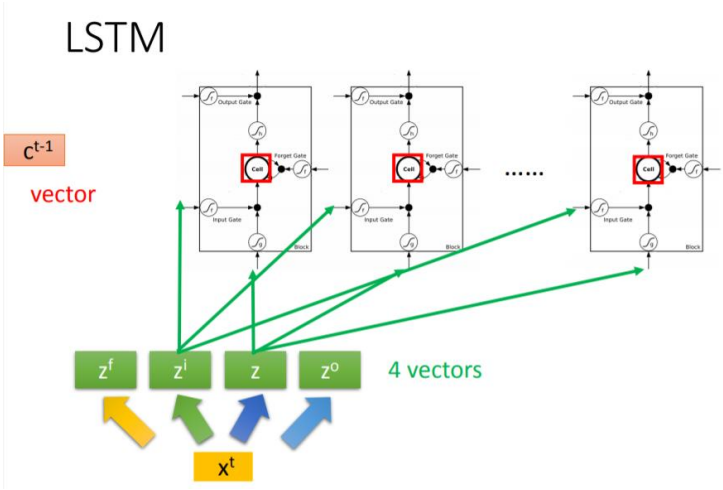


LSTM with neural network

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，……

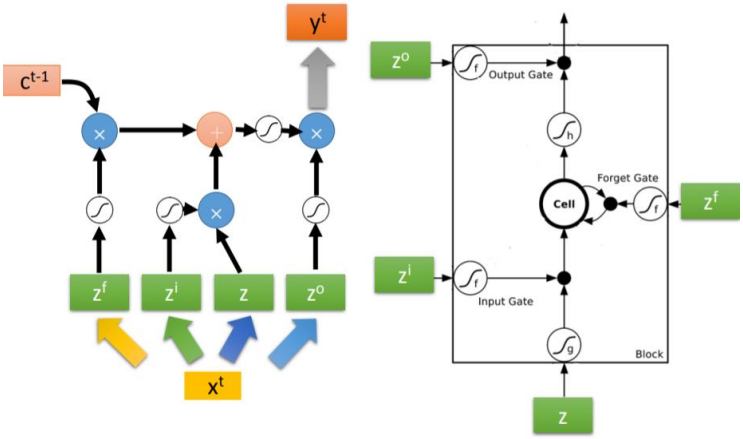
因此一共有 4 个不同的 vector z_f, z_i, z, z_o,作为 network 的 input，操控整个 LSTM 的运转，输入到第一个 LSTM

cell 的值 vector z_f, z_i, z, z_o中的第一维数据，每个 cell 的输入值都是不一样的



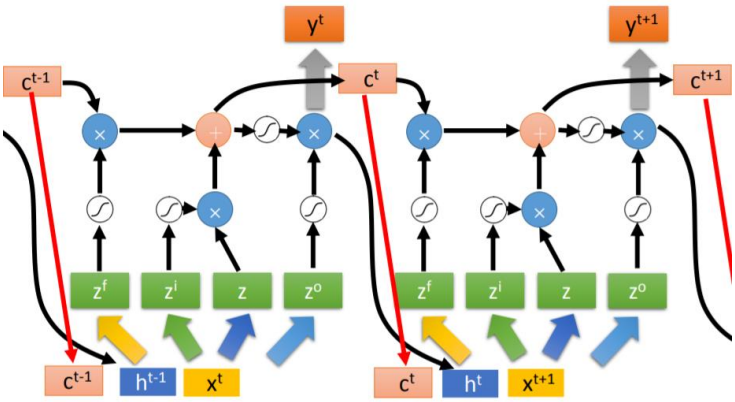
x^t进行不同的 transform，可以得出 4 个不同的 input z_f, z_i, z, z_o

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经过运算的$f(z_i)$，$g(z)$再和$f(z_i)$相乘(element-wise),memory cell 的初始值为c^{t-1},经过运算可得到的值$c^t = g(z)f(z_i) + c^{t-1}f(z_f)$，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 t 进行的计算，不仅包括x^{t+1}，还要包括上个时间点中 LSTM cell 算出来的值，即 memory cell 的输出c^t，以及c^t再输入 activation function 的输出值$h(c^t)$

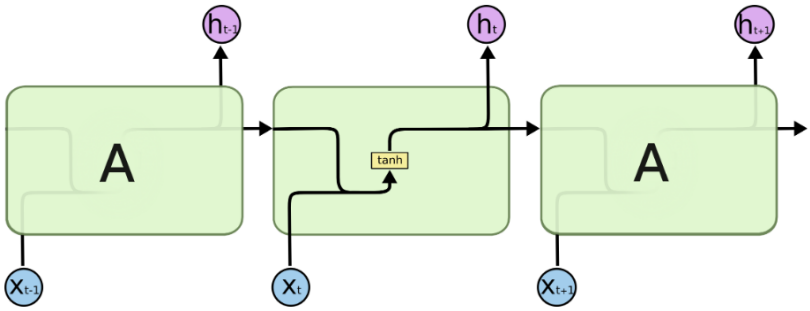


RNN vs LSTM

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点的计算结果，可以处理短期依赖问题，无法处理长期依赖问题；
- LSTM 通常采用了“累加”的形式，上一个时间点的计算结果会影响之后的计算结果，并不是直接覆盖；可以处理短期和长期依赖问题
- 对比两者的参数更新公式

首先是传统的 RNN，网络单元结构示意图

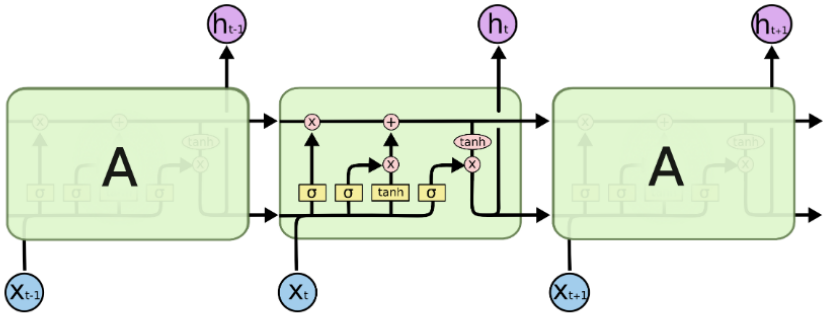


The repeating module in a standard RNN contains a single layer.

参数更新公式如下，RNN 此时的隐藏层信息只来源与当前输入和上一时刻的隐藏层信息，没有记忆功能

$h_t = \tanh\left(W_h \cdot [h_{t-1}, x_t] + b_h\right)$

对 LSTM 而言



The repeating module in an LSTM contains four interacting layers.

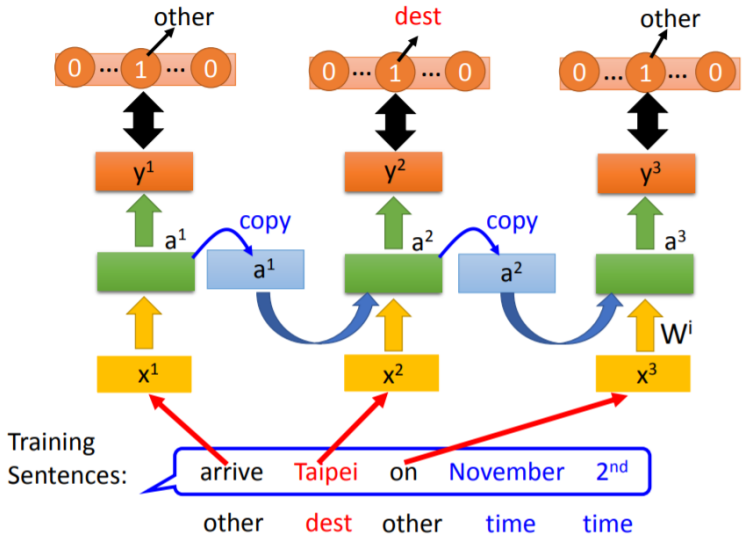
参数更新公式如下，

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

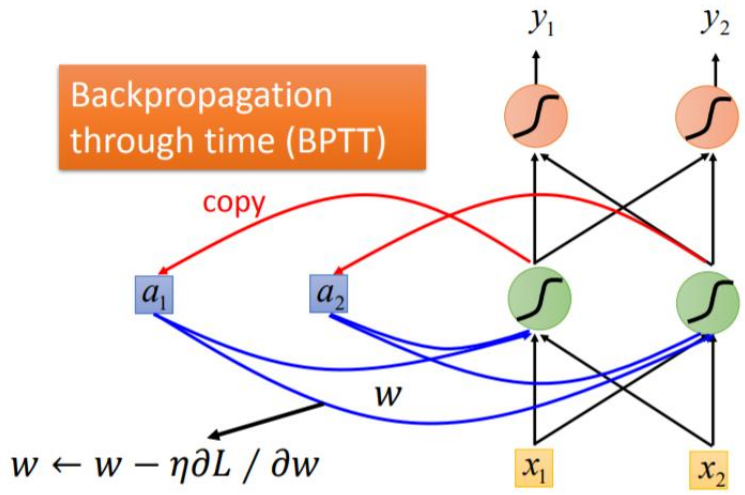
$a = h(c')f(z_o)$

Learning

Training Sentence 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

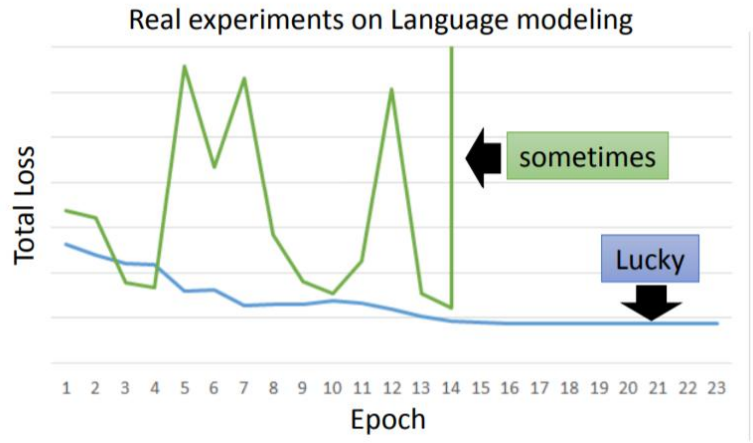


RNN 的训练也是前向和逆向传播，其中逆向传播为 Backpropagation through time (BPTT),也有一个参数更新公式



但基于 RNN 的 network 并不是很容易训练

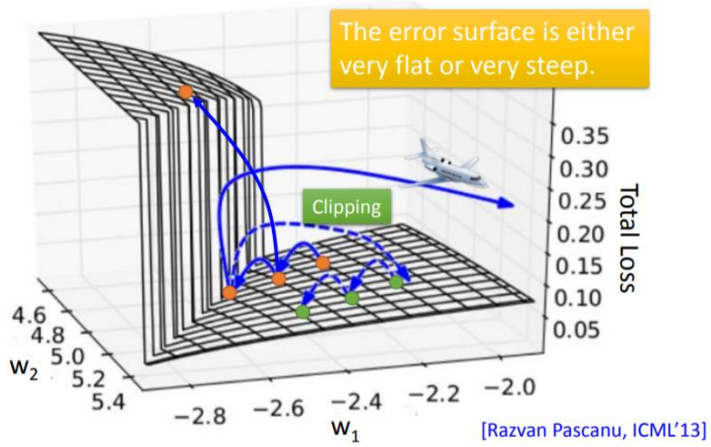
- RNN-based network is not always easy to learn



Problem

下图中的纵轴表示 total loss，横轴表示两个 weight w_1和w_2，而 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 可能就直接飞出去了

Clipping

改进措施是 clipping，当 gradient 大于 15 的时候，就看做 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，仍然尅继续做 RNN training

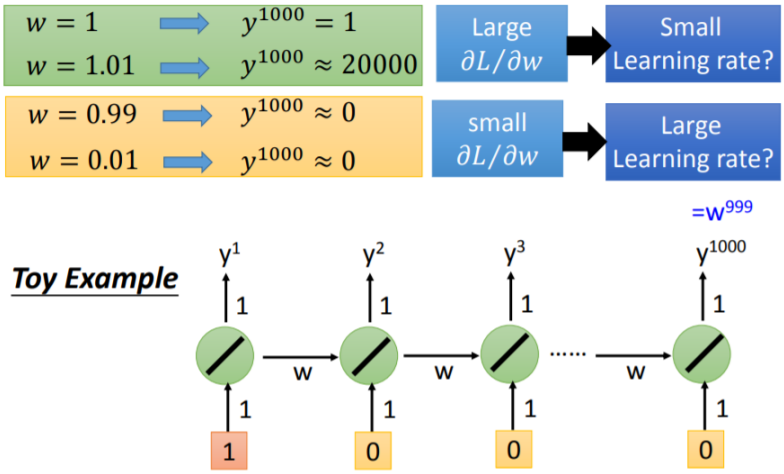
Why1?

Q:由于之前的的文章提到 sigmoid function 会产生 gradient vanishing 的问题，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 替换成 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，再上一个时间点的 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

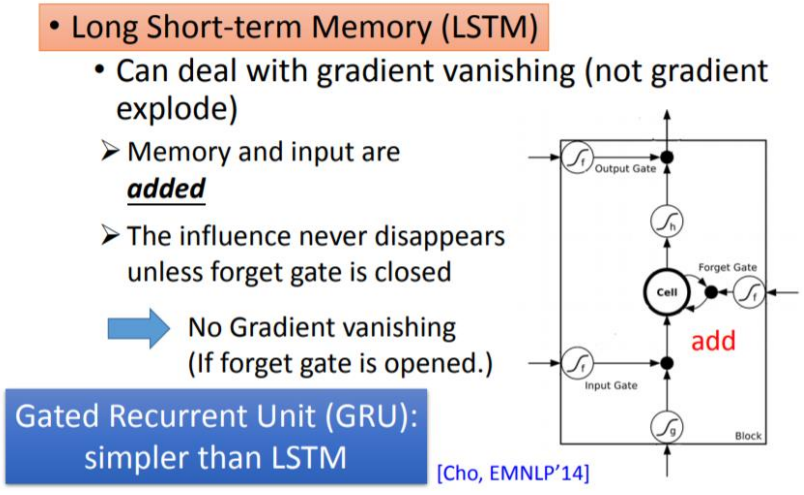
如果$w = 0.99$, 那么对应的输出$y^{1000} \approx 0$;如果$w = 0.01$，那么对应的输出$y^{1000} \approx 0$, w 虽然很大的变化，对 output 的的影响却很小，因此有很 small 的 gradient$\frac{\partial L}{\partial w}$,这时就需要把 learning rate 设置大一点

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

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

Helpful Technique

LSTM 可以让 error surface 不那么崎岖，可以把一些很平坦的地方拿掉，是 error surface 不再那么平坦的地方，这时可以把 learning rate 设置小一点，可以解决 gradient Vanishing 问题



LSTM 可以解决 gradient Vanishing 的原因：

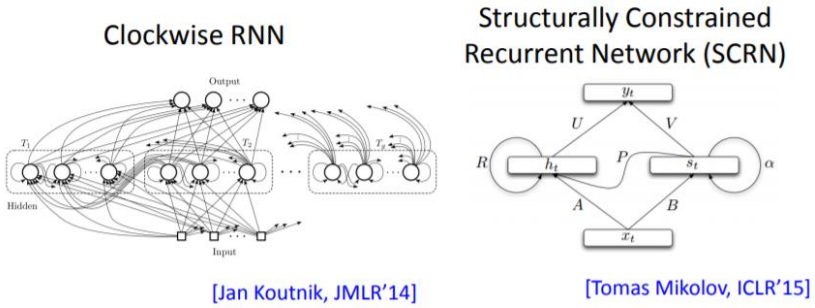
- 当前的 input 和 memory 里的的值是“累加”的
- 如果 forget gate 是 open state，上一个时间点的 memory 会对当前时间点造成影响，这个影响会一直持续下去

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 大部分情况是 forget 的

Gated Recurrent Unit(GRU):比 LSTM 更简单，gate 的数量只有两，即 forget gate 和 input gate，这两者有一个联动:

- 当 input gate open 时，forget gate close，forget 掉 memory 里的值
- 当 forget gate open 时，记住 memory 里的值，input gate close，不再进行输入

还有一些其他的技术可以解决这个问题



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

➢ Outperform or be comparable with LSTM in 4 different tasks

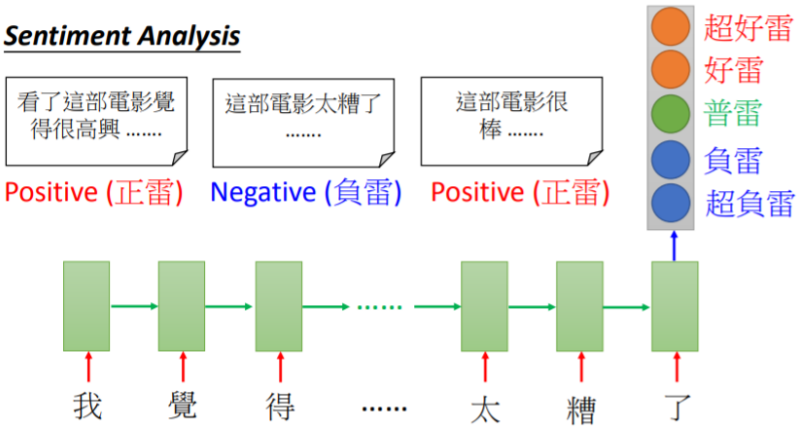
More Application

Many to one

Input 是一些 vector 的 sequence，output 可以说一个 vector

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

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



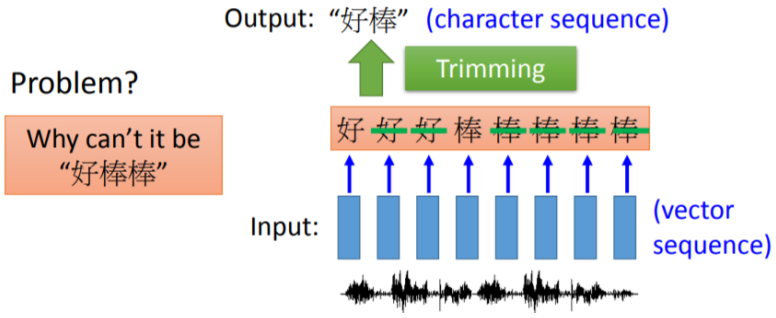
Many to Many(output is shorter)

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

对与原来的 RNN，每个 vector 对应一个 character，input 的 vector 通常是很短的，比如 0.01s，因此通常是很多 vector 对应同一个 character

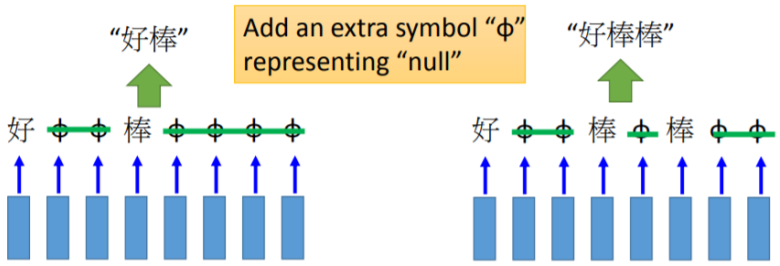
有一种解决方式是 trimming，去掉重复汉字，但这时就识别不出“好棒棒”，只能识别出“好棒”

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



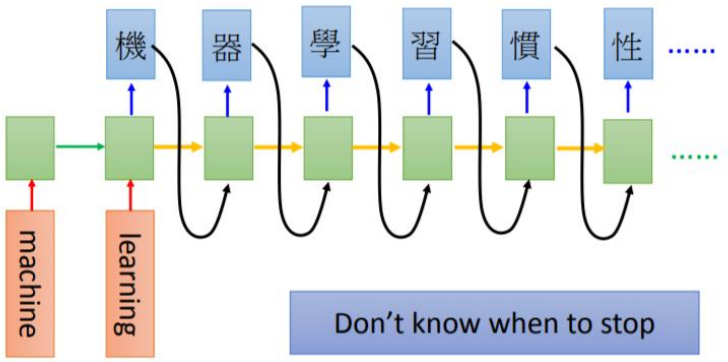
有学者提出了 CTC，添加了表示 null 的符号进来

- Both input and output are both sequences, ***but the output is shorter.***
- Connectionist Temporal Classification (CTC) [Alex Graves, ICML’06][Alex Graves, ICML’14][Haşim Sak, Interspeech’15][Jie Li, Interspeech’15][Andrew Senior, ASRU’15]



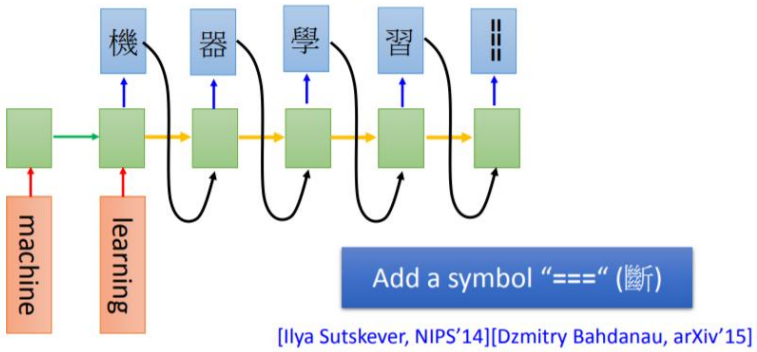
当输入“机”字，根据上一个时间点的 memory 和“机”，RNN 就可以学习到输出为“器”，….

- 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

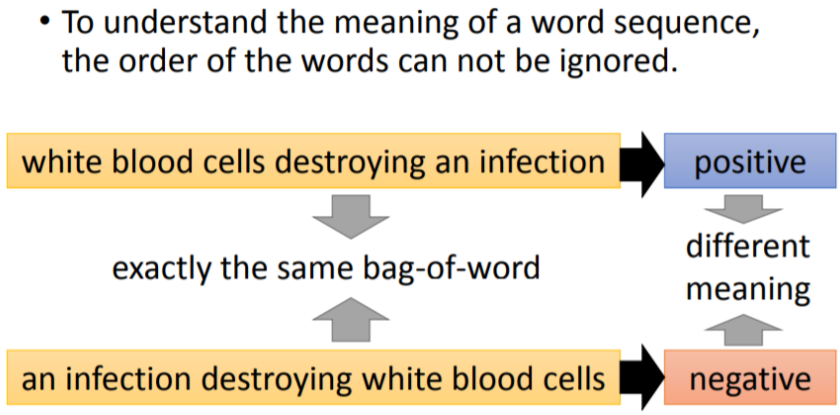
- Both input and output are both sequences ***with different lengths***. → ***Sequence to sequence learning***
  - E.g. ***Machine Translation*** (machine learning→機器學習)



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

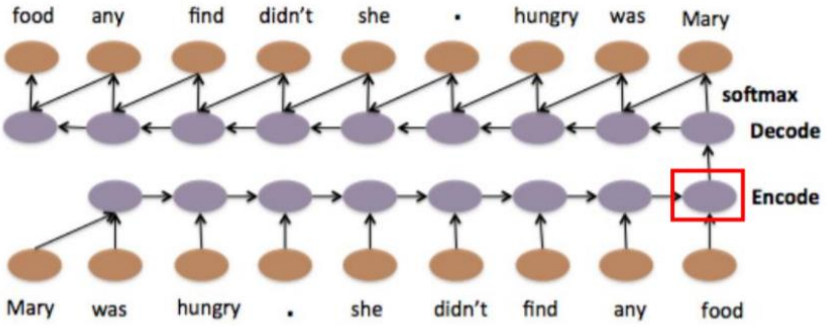
Sequence-to-Sequence Auto-encoder-Text

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 postive，一个是 negative

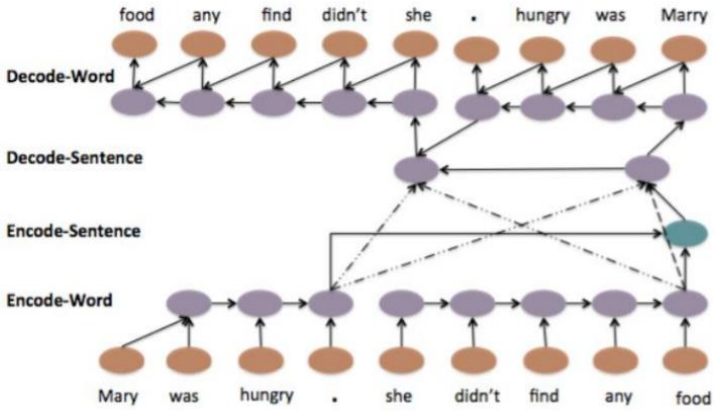


在考虑单词原有顺序的情况下，我们可以通过 sequence-to-sequence auto-encoder 来将一个 document 庄华伟 vector，那具体怎么做呢?

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



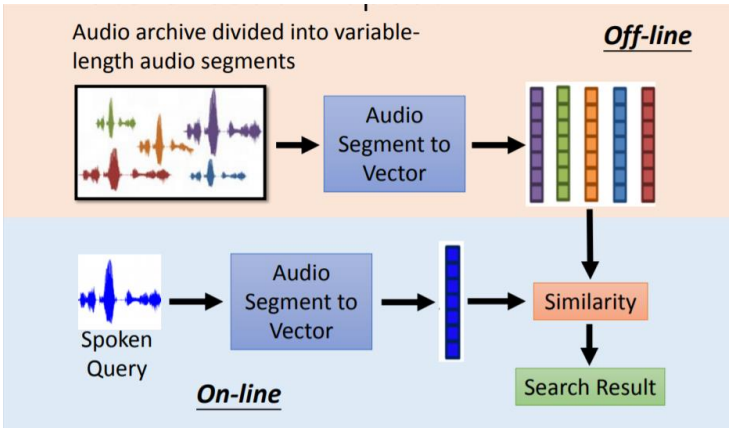
Li, Jiwei, Minh-Thang Luong, and Dan Jurafsky. "A hierarchical neural autoencoder for paragraphs and documents." *arXiv preprint arXiv:1506.01057*(2015).



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

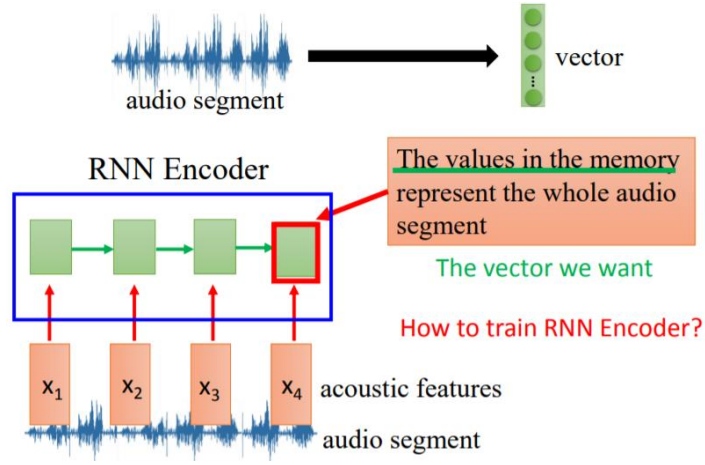
Sequence-to-Sequence Auto-encoder-speech

比如有一段演讲语音，现在我想要搜索 audio 中提到了白宫的部分，这时我们就需要作如下工作，把输入的 audio segment 转化为 vector，再把 spoken query 的 audio segment 转化为 vector，再计算两者的相似程度，就可以得到搜寻的结果

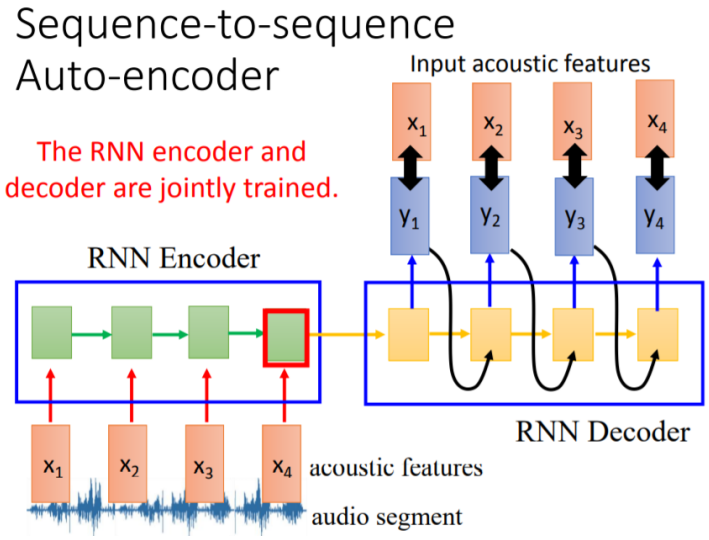


Audio->vector

把输入的 audio segment 分成 4 部分x_1, x_2, x_3, x_4,输入 RNN，这里 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 是没有办法 train 的，所以这里加入了 RNN 的 encoder。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



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