

Recurrent Neural Network.

KEYWORDS: RNN, LSTM, Attention

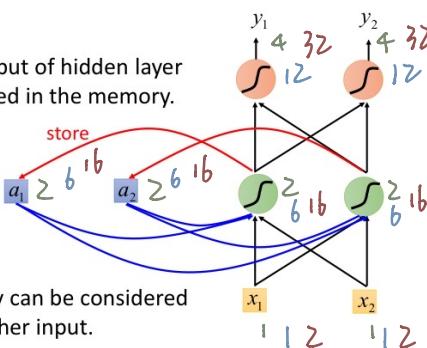
应用: Slot Filling (自动填充)



RNN:

The output of hidden layer are stored in the memory.

Memory can be considered as another input.



用 Feedforward N?

输入一个词的向量表示，输出该词属于每个slot的概率。

问题: "leave Taipei" 和 "arrive Taipei", 无法区分

Taipei 是出发地还是目的地



希望 NN 有记忆，在判断 Taipei 时记住看过的 arriver 还是 leave.

RNN: hidden layer 的 output 被存到 memory 中, memory 可作为输入.

下一轮 hidden layer 的

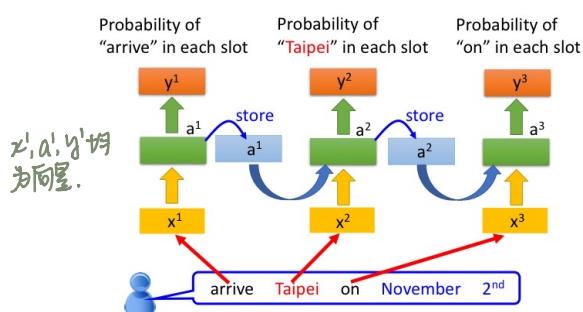
假设所有 weights $\neq 1$, bias $\neq 0$.

激活函数为线性。 输入相同输出也可能不同。

Input sequence: $[i][i][2] \dots$

Output sequence: $[4][12][32] \dots$

同一网络被 sequence 中的每个输入重叠使用。

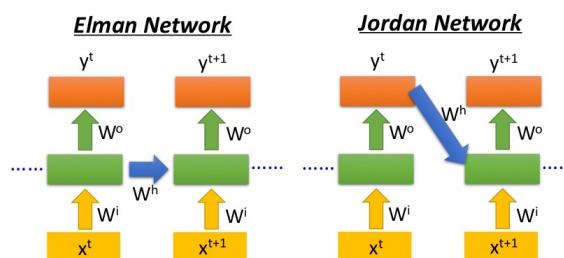


两种不同结构：

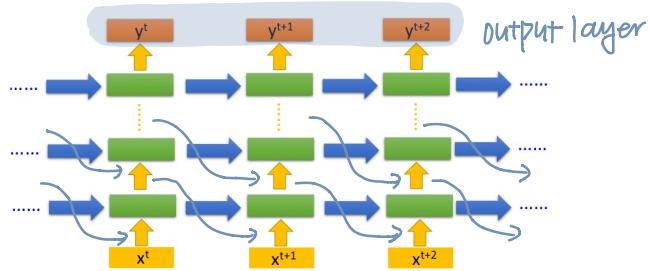
{ Elman N: 从 hidden layer 输出.

{ Jordan N: 从 output layer 输出.
(比较清楚放的是什么)

因为 "arrive Taipei" 和 "leave Taipei" 在处理 "Taipei" 时 memory 中内容不同，所以可区分两个 "Taipei"。

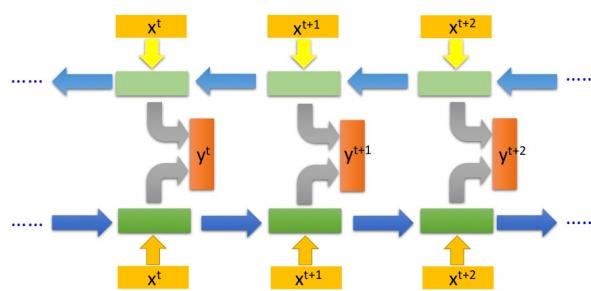


可以 deep: 隐藏层多于1个.



训练: Backpropagation through time (BPTT)

△ Bidirectional RNN

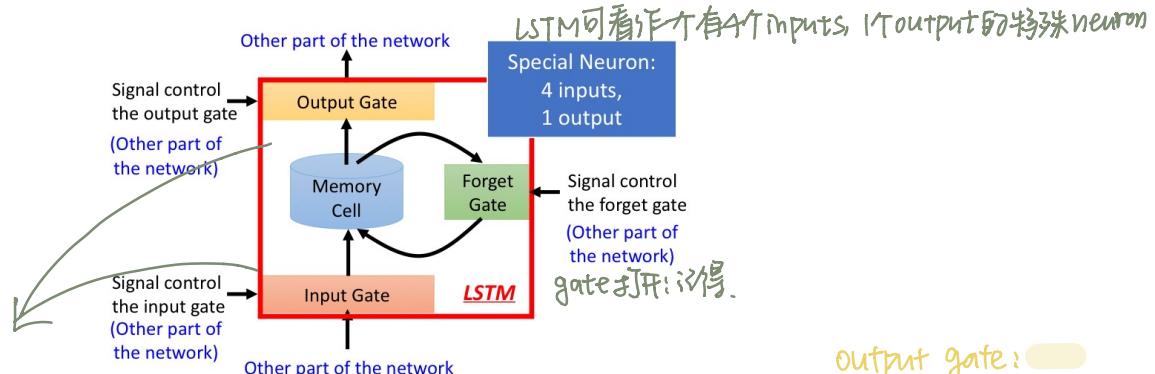


同时训练一对正向、反向RNN，将对应的 hidden layers的 output 拿出来，接给一个 output layer，得 y .

单向: 产生 y^{t+1} 时, 只有 x^t, \dots, x^1 信息
双向: 产生 y^{t+1} 时, 除了 x^t, \dots, x^1 外还看到 x^{t+2}, x^{t+3}, \dots

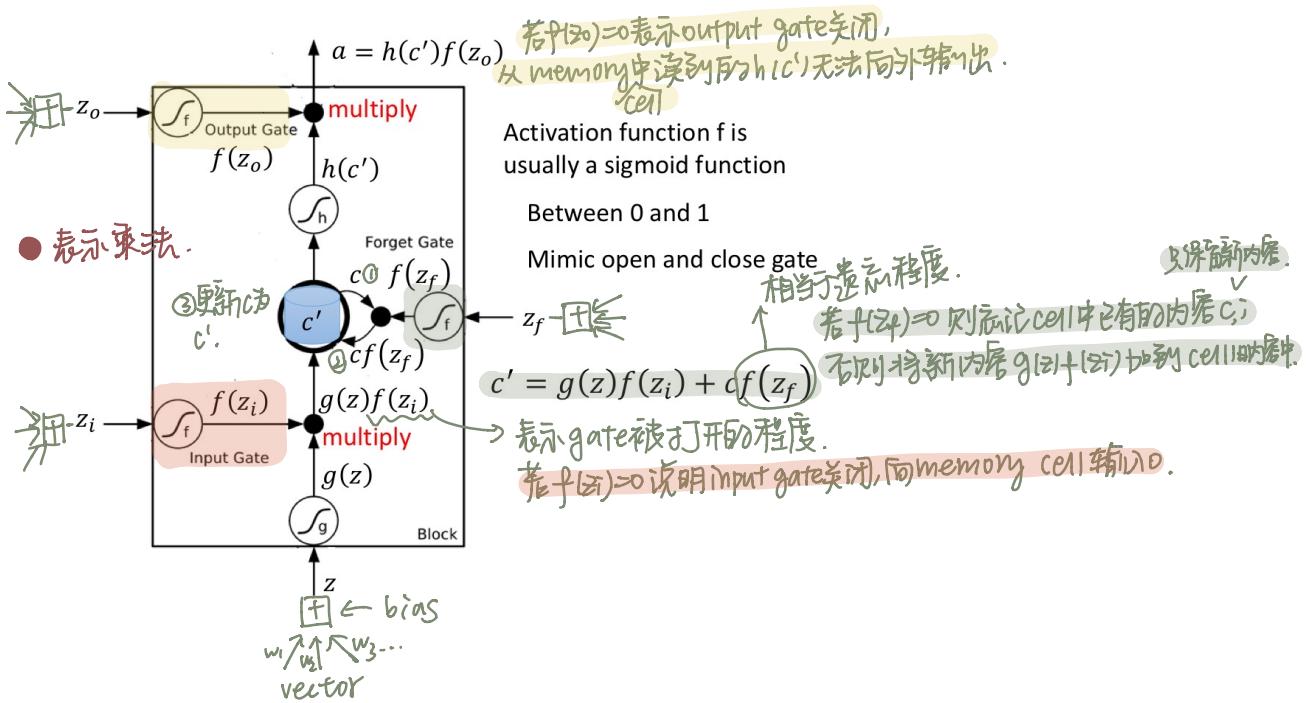
△ Long Short-term Memory (LSTM)

LSTM是训练序用的一种 memory.



gate 打开可 input/output
gate 关闭相当于 gate 车输出
为.

output gate:
forget gate:
input gate:



e.g.

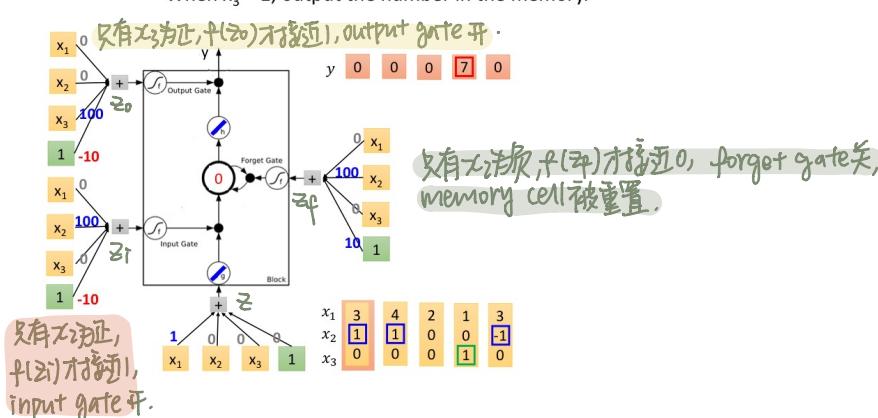
x_1	0	0	3	3	7	7	0	6
x_2	0	0	1	0	0	-1	1	0
x_3	0	0	0	0	1	0	0	1
y	0	0	0	0	0	7	0	0

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.

} 为何有这样的效果？和 NN 的 weights 及 bias 有关。

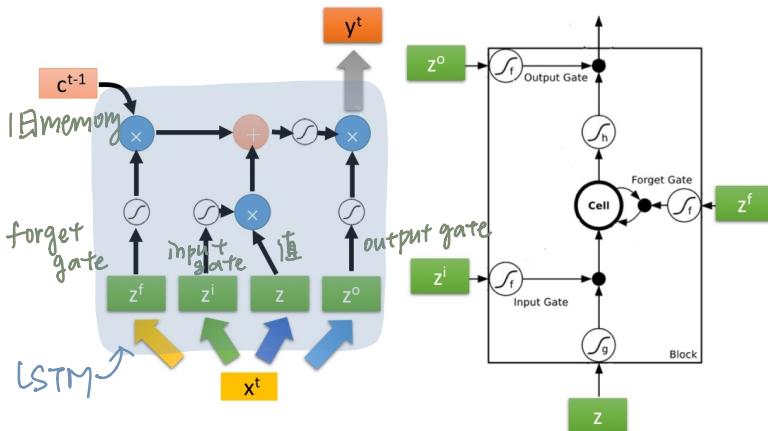
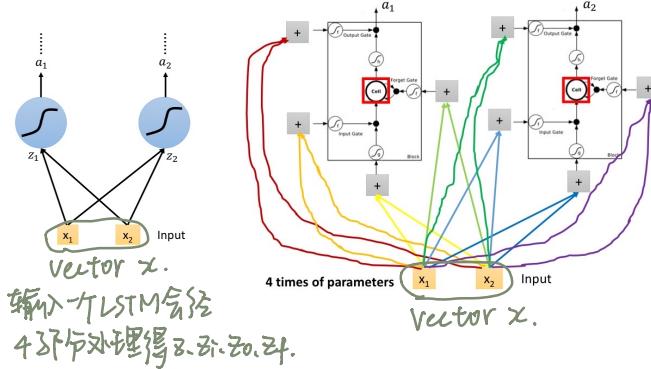


LSTM 的 inputs 由 vector 和 weights、bias 共同决定，3 个 inputs 为信号，一个 input 为输入值。4 个 inputs 均为单个值。

任何输入一个向量，会经不同 weights 和 bias 计算得 4 个值作为 inputs。

→ 和普通neuron没有区别，均以 vector 作输入。

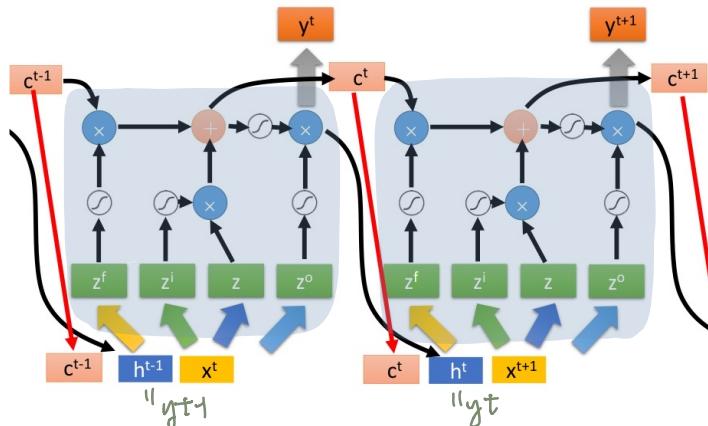
使用LSTM时，直接像使用普通neuron一样即可。（但会有普通neuron 4倍的参数）。



△ LSTM 扩展：

前一时刻

- ① peephole: 将 memory cell 里的值 c^{t-1} 和输出值 y^{t-1} (h^{t-1}) 作为当前时刻的输入。



- ② 多层LSTM：当前层 y^t 作为下一层 x^t 。

现在该用RNN和用LSTM是什么意思。Keras 支持 LSTM、GRU、Simple RNN.

↓
Gated Recurrent Unit

将 input gate 和 forget gate 联动。
input 打开，forget gate 关闭(漏掉)。

"新的进来，旧的不走"。

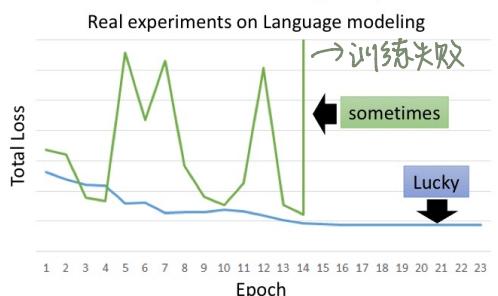
· 相当于只有 2 个 gates 是 LSTM 的简化，参数少，train 时更 robust.

△ 如何 train 及 train 中的问题

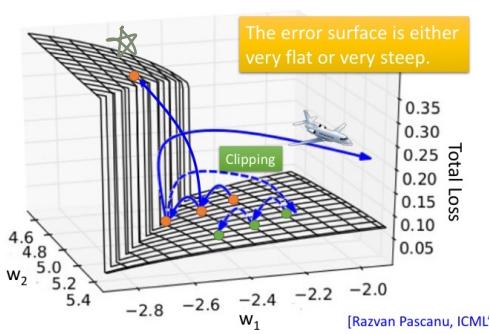
Back propagation through time (BPTT). (从 BP 引些时间维度的信息)。

实际还是用 gradient descent.

- RNN-based network is not always easy to learn



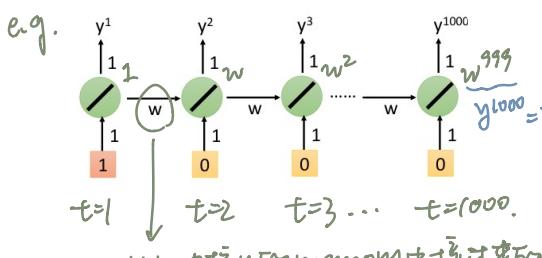
← RNN 较难训练：loss 会跳来跳去。



error surface: loss 和参数的关系。

如果一直在 flat 处，可能会将 learning rate 调大，
一旦踩到悬崖上，gradient 会梯度爆表，参数会被
更新到十万八千里，loss 梯度爆表无序大。

解决：Clipping：当 gradient 大于 threshold 时让值直接等于 threshold.



假设我们想要学习 w.

$$\text{case 1: } \begin{cases} w = 1 \Rightarrow y^{1000} = 1 \\ w = 1.01 \Rightarrow y^{1000} \approx 20000 \end{cases}$$

w 的变化对 output 影响

很大 \Rightarrow L 对 w 的 gradient 很大。

\Rightarrow 梯度爆炸

gradient explode

Case 2: $\begin{cases} w = 0.99 \Rightarrow y^{1000} \approx 0 \\ w = 0.01 \Rightarrow y^{1000} \approx 0 \end{cases}$ 对 w gradient 很小。 \rightarrow gradient vanishing \rightarrow 梯度消失

在 w 的很小区间内, $\frac{\partial L}{\partial w}$ 变化很大, 难以设置合适的 learning rate.

原因: 将同一参数 w 在不同时间反复使用, w 产生的影响被无限放大. (导致出现 w^{999}).

解决方法: LSTM (可解决梯度消失, 不能解决梯度爆炸).
 error surface 较平坦处 error surface 峰山区处.

$\begin{cases} \text{Simple RNN 中, 每个新时间步, memory 里的旧值会被新值覆盖. } (w \rightarrow w^2 \rightarrow w^3 \rightarrow \dots) \\ \text{LSTM 中, 每个新时间步, memory 里的旧值会乘上 } f(t) \text{ 和新值相加.} \end{cases}$

LSTM 最早提出就是为
 避免梯度消失, 因此当时没
 有 forget gate (相当于 forget
 gate一直开着). \Rightarrow train 时不要给 forget gate 大的 bias, 保证在大多数情况下开启.

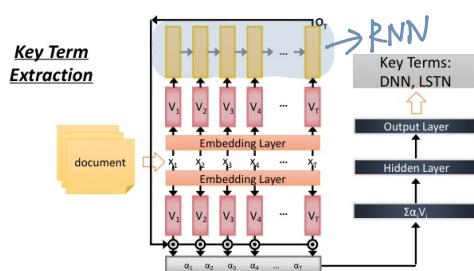
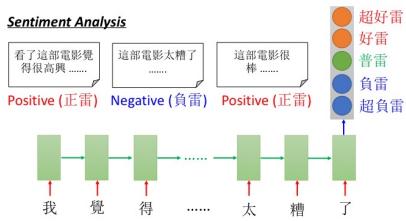
除非 forget gate 被关上, 对 cell 进行一次 format,
 否则之前 w 的影响会一直存在 cell 中.
 $(w \rightarrow c_0 + w^2 \rightarrow c_1 (c_0 + w^2) + w^3 \rightarrow \dots)$

总结 $\begin{cases} \text{梯度爆炸 (gradient explode) : 一直在梯度平缓处停滞不前.} \\ \text{梯度消失 (gradient vanish) : 梯度更新步伐太大导致直接飞出有效区间.} \end{cases}$

△ 应用

多对一: 输入 vector sequence, 输出只有一 \rightarrow 最后时刻输出向量. \rightarrow 最后时刻输出
 vector. 向量属于某类的权值.

① Sentiment Analysis: 输入字符序列 (每一时刻输入一个字), 最后时刻输出 sentiment.



② key term Extraction (关键词):

多对一: input和output均为vector sequence.

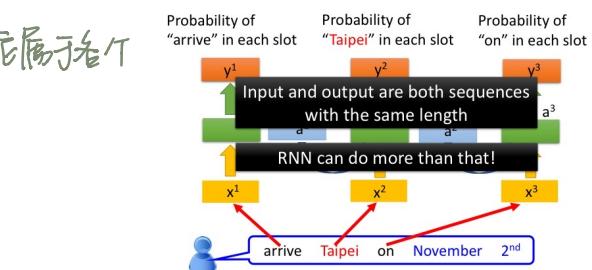
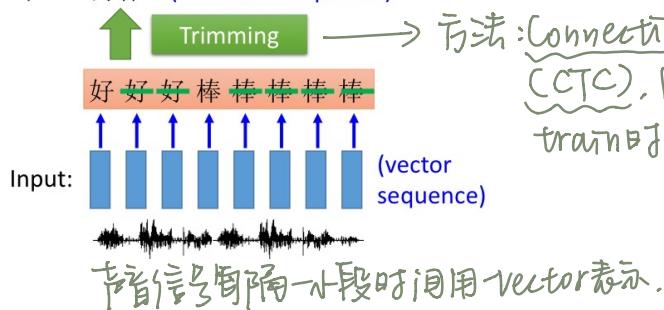
① 轮输出sequence和输入的一样长:

. **Slot Filling**: 对每一个word vector, 输出它属于各个slot的概率.

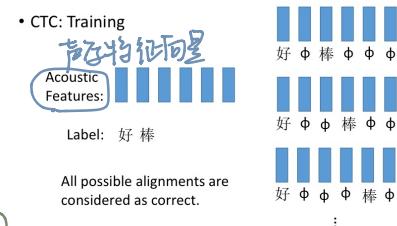
② 轮输出sequence比输入的短:

Speech Recognition (语音识别):

Output: "好棒" (character sequence)

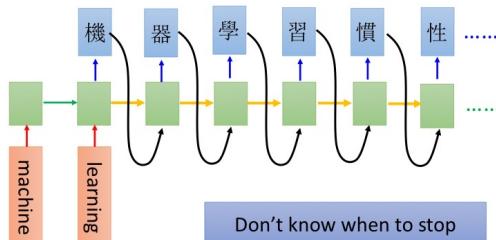


方法: Connectionist Temporal classification (CTC), 向序列中添加中表示null.
train时, 常常所有包含null的排列, 均被正例.



③ 输入、输出sequence长度不同 (Seq2Seq Learning)

a. **Machine Translation**



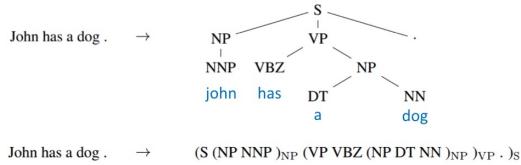
在这个时间点, memory存了整个句子"machine learning"的信息, RNN输出"机"后, 将"机"作为输入, 结合memory中信息可得"器".

问题: RNN不知何时停止输出.

解决方法: 加一个"==" symbol, 表示结束. 输出"=="后则停止输出.

b. **Syntactic Parsing (句法解析)**

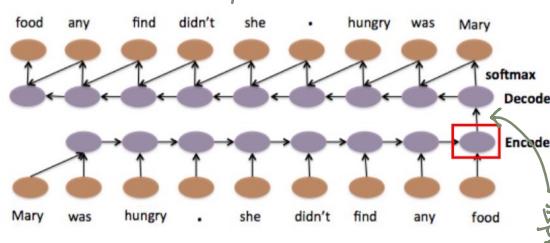
自动生成树状语法结构图.



c. Auto-encoder Text 文章编码. (把RNN当作编码器和解码器)

若用传统的 bag-of-word 表示 text, 会丢失语序信息.

可行) 想将文章编码成一个 vector:



encode 得到的 vector 能通过 decode 得到原文, 则说明 vector 成功包含了语序信息.

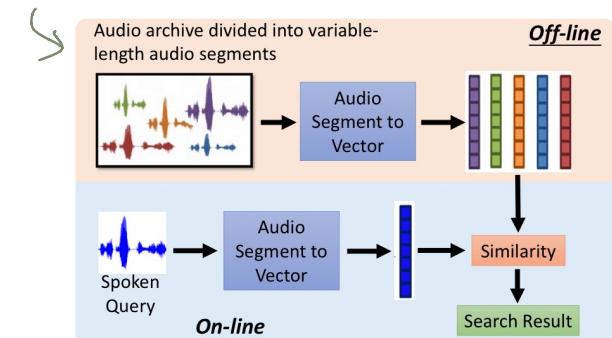
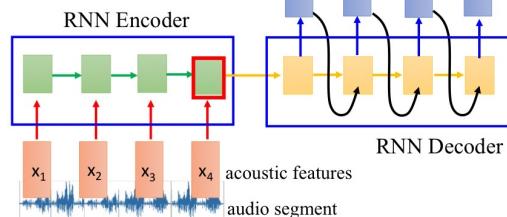
最后时刻输出即为所求 vector.

d. Auto-encoder Speech.

将一段语音编码成 vector. (可用 vector 间的相似度做语音搜索)

Sequence-to-sequence Auto-encoder

The RNN encoder and decoder are jointly trained.

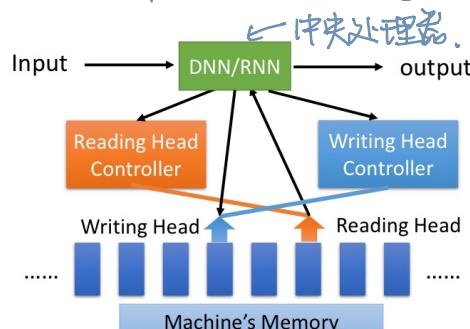


训练时希望 x_i 和 y_i 差距最小.

= Neural Turing Machine.

△ Attention-based Model (RNN 进阶, 也使用 memory)

NN 通过操控读写头去读取指定位置.



应用: 阅读理解; 一篇文章向量为 vector, 用户提问后, NN 调用读写头, 取出 memory 中与查询语句相关的信息, 处理后给出回答.