深入理解RNN结构

近年来,自然语言处理成为了人工智能界的一个热门话题, LSTM, Attention, Transformer 等模型结构大火,更基于这些理论衍生出了各种强大的预训练模型,如BERT,GPT3等.

这些算法的共有的核心思想之一便是RNN (Recurrent Neural Network),本文将尽可能详细的介绍RNN的逻辑和实现原理(会包含核心公式,不会介绍具体训练过程的公式推导)

传统神经网络的局限性

在神经网络算法中,大部分算法(MLP,CNN,...)都是输入(x)与输出(y)独立对应的,也就是说

$$x_1 \rightarrow y_1, x_2 \rightarrow y_2, ..., x_n \rightarrow y_n$$

但是在某些场景中,独立的输入就变得不够了,例如我们想对一句不完整的话填词

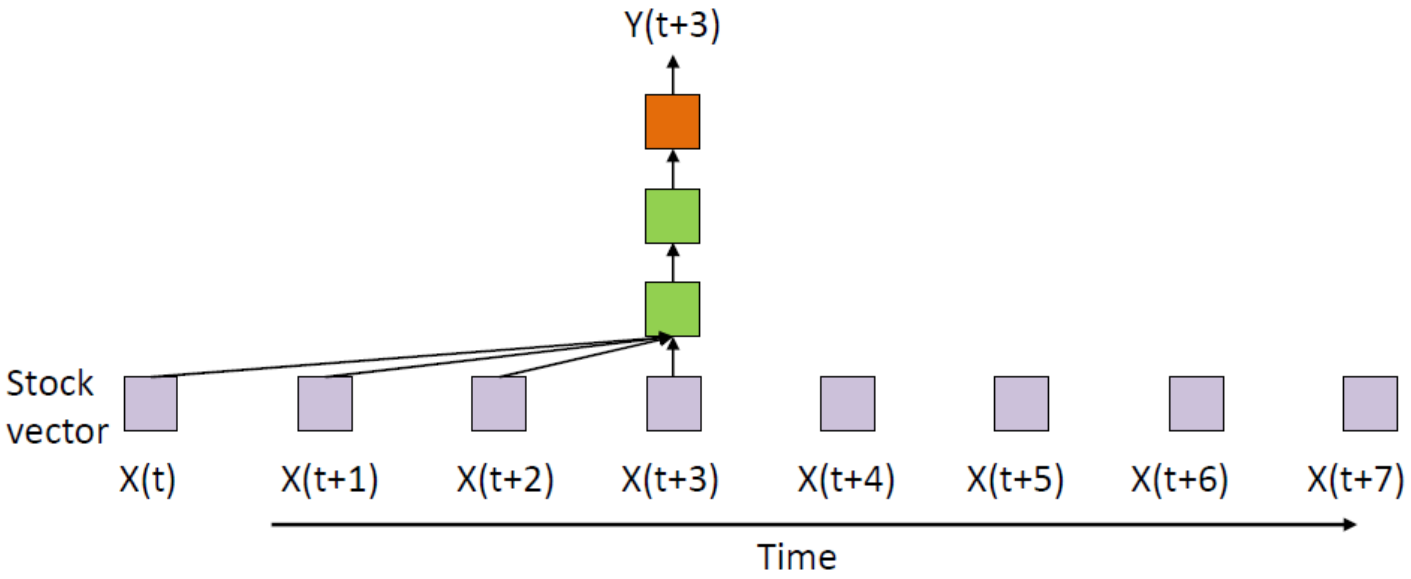
制作回锅肉所用的肉是__

显而易见,这里的回答应该是"猪肉". 但是利用神经网络的情况下,我们即使对上述句子做了分词操作,仅仅基于某一个字或者词,显然是没办法预测结果的. 这时候我们就需要处理**具有互相依赖性质的时序数据**,这种场景下,便需要基于其他方式来实现了.

The Sliding Predictor

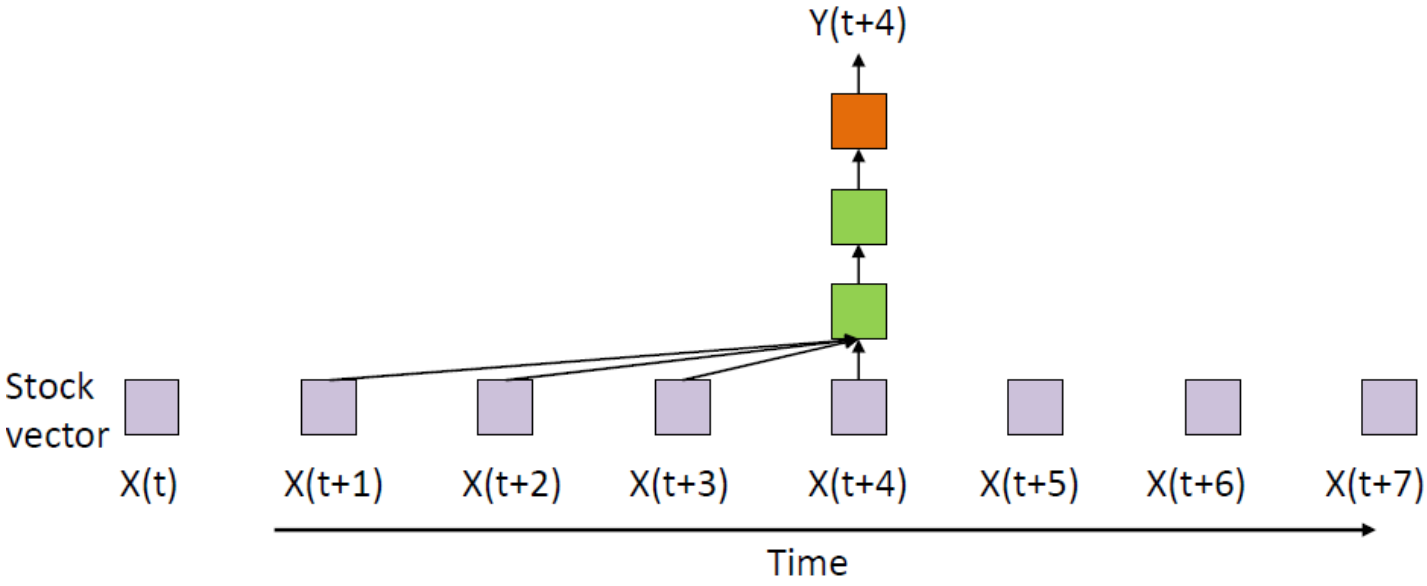
Sliding Predictor & CNN

为了解决上述问题,最简洁易懂的方法之一便是滑动预测模型(Sliding Predictor),通过观测前几个时间节点的输入和当前时间节点的输入,做为模型的整体输入. 以下为了更清楚的展示模型结构,我们以一个更经典的Use Case来做为例子: 预测股票价格



Sliding Predictor 样例

如上图Sliding Predictor会以$t, t+1, t+2, t+3$时刻的的股票向量(stock vector)做为整体的输入来预测$t+3$时刻的股票价格



Sliding Predictor 样例

以此类推,Sliding Predictor会以一样的逻辑去预测$t+4$时刻的股票价格

不难看出,以上的计算方法很类似与计算机视觉算法CNN中的卷积过程(Convolution Step),所以 Sliding Predictor 实际上就是应用在序列数据上的CNN. 这样的算法也被称为 *Time-Delay neural network*

Finite-response Model

这样的模型属于*Finite-response Model*,更形象的来说就是,今天发生的事情只会影响未来N天以内的结果,N就是整个系统的宽度

$$Y_t = f(X_t, X_{t-1}, ..., X_{t-N})$$

Problems

上面的模型看起来非常合理,但是当我们的影响辐射宽度变大了怎么办?如果今天发生的事情会影响未来10,000天内的结果呢?这时候模型会变得更加复杂

> "不用担心,我们的CPU够用" --> Do we?

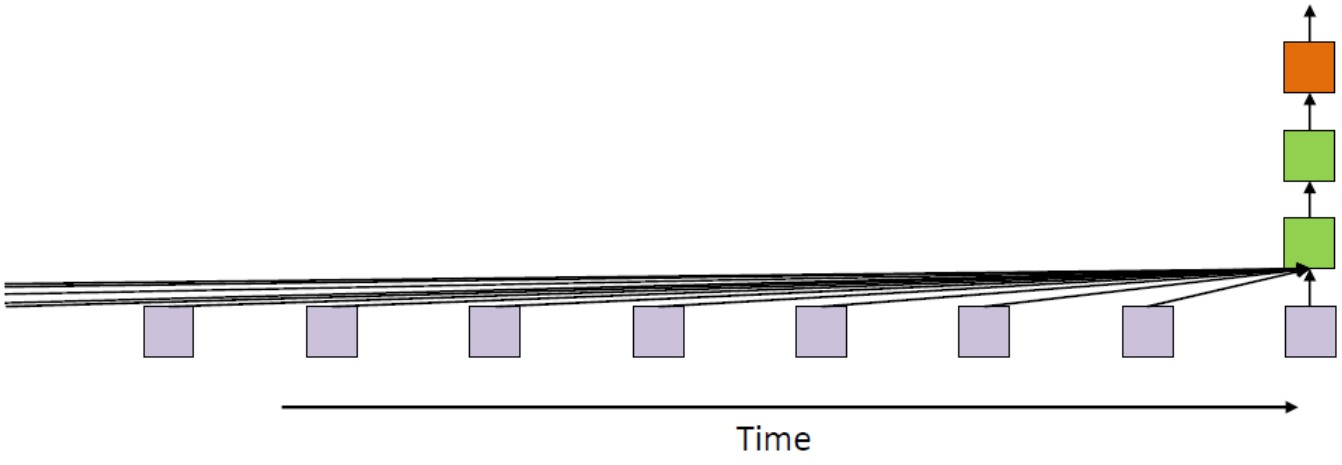
Long Term Dependency

很多场景下,我们需要预测的结果会基于长期依赖(long term dependency),比如在股票预测中,我们可能会考虑:

- 一周内的股市趋势
- 一个月内的股市趋势
- 整年的股市趋势
- ...

NARX Network

如果今天发生的事情会影响未来所有的结果?我们需要无限的记忆:



Infinite-response Model

那么,

$$Y_t = f(x_t, x_{t-1}, ..., x_{t-\infty})$$

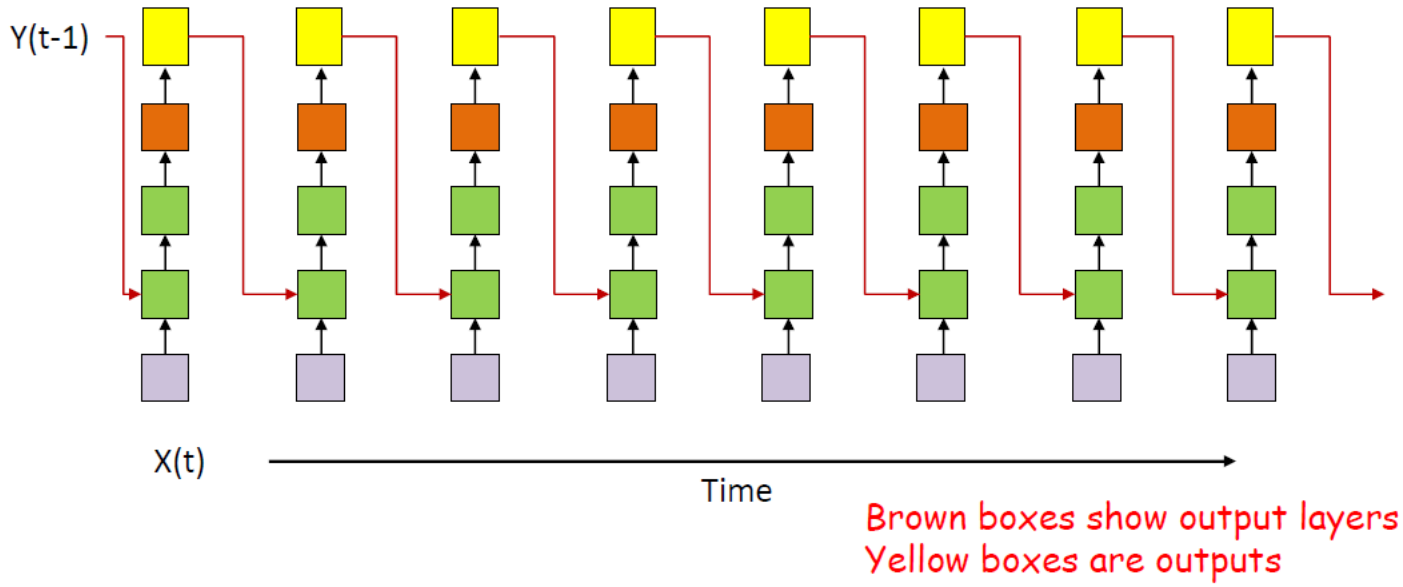
或者我们可以这样理解,

$$Y_t = f(X_t, Y_{t-1})$$

- 这样的假设需要对初始状态进行定义,也就是 $t = 0$ 所对应的 Y_{-1},
- 这时候$t = 0$所对应的输入X_0会综合Y_{-1}得到Y_0,
- 接下来通过Y_0得到Y_1,Y_2,...,Y_∞, 甚至在X_1, ..., X_∞为0的情况下
  - i.e.对应时刻的没有X的输入

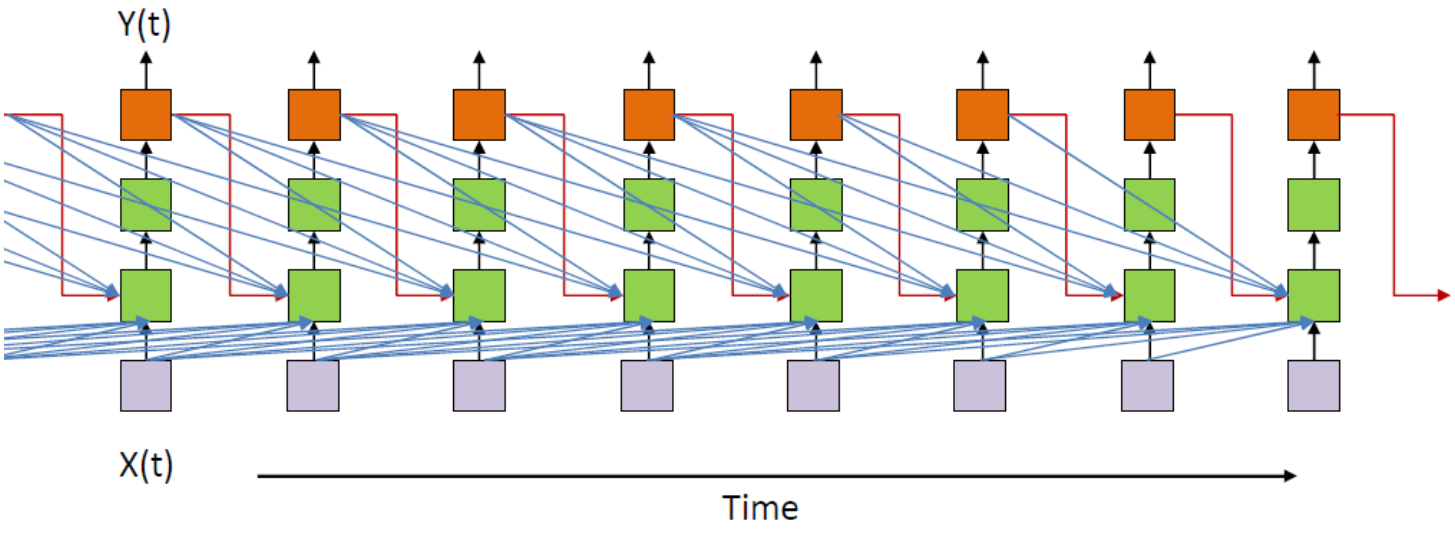
这样的模型结构被称为*NARX network (nonlinear autoregressive network with exogenous inputs)*

$$Y_t = f(X_{0:t}, Y_{0:t-1})$$



NARX模型基础结构

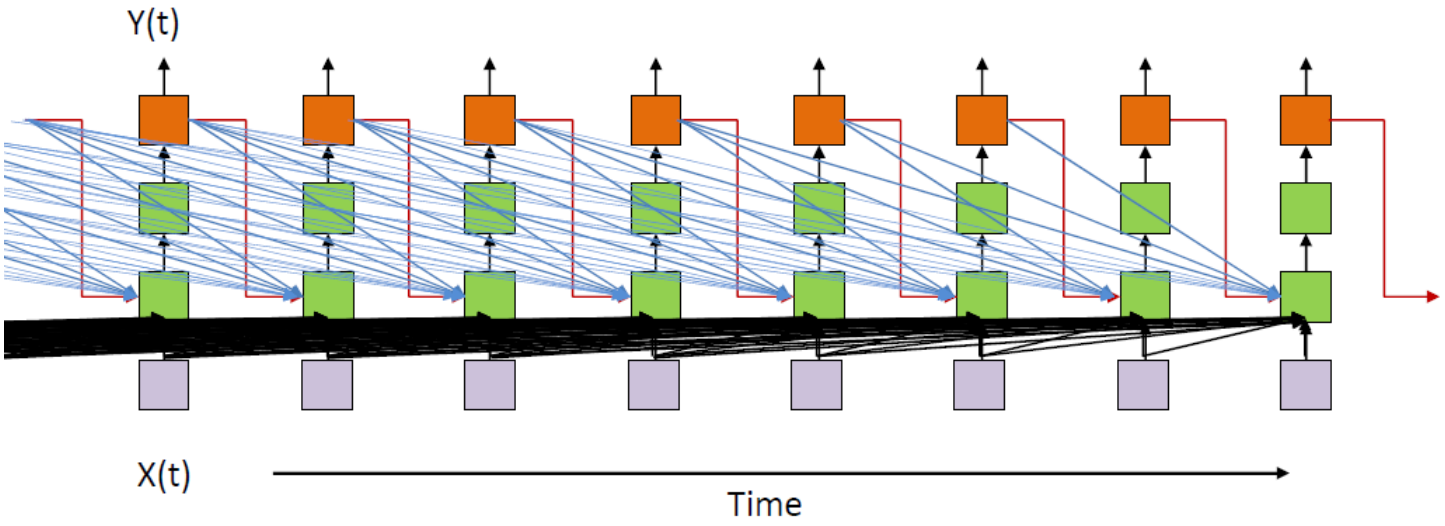
更加通用的NARX如下:



NARX模型结构

在t时刻的Y_t会基于以往的K个输出$Y_{t-1}, ..., Y_{t-k}$和L个输入$X_t, ..., X_{t-L}$计算

"完整"的NARX如下:



NARX“完整”模型结构

在t时刻的Y_t会基于以往的所有输出$Y_{t-1}, ..., Y_{-1}$和所有输入$X_t, ..., X_0$计算,这种模型由于算力的原因不能做因为一个实用的模型.

系统的定义memory

在时序模型中,明确的记忆的定义方式是去让他记住:

$$m_t = r(y_{t-1}, h_{t-1}, m_{t-1})$$

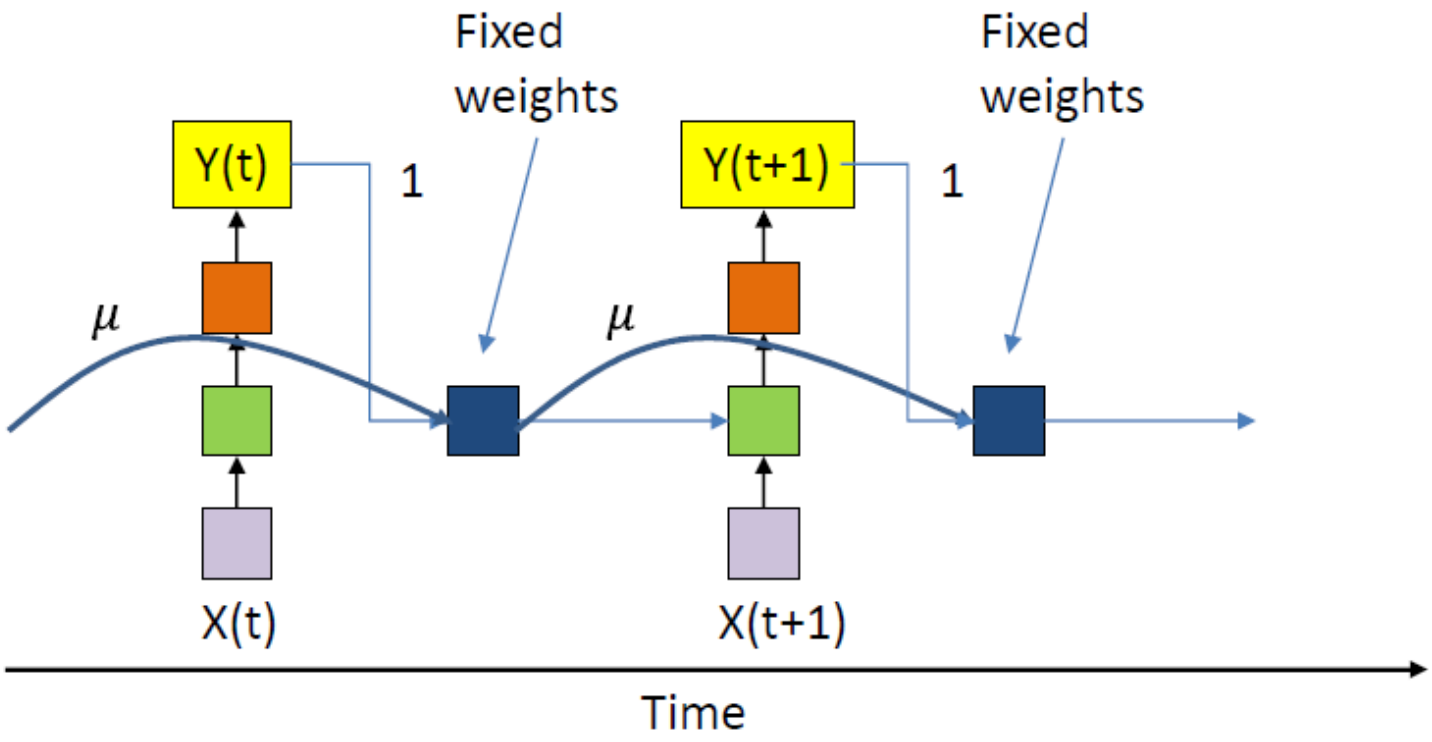
$$h_t = f(x_t, m_t)$$

$$y_t = g(h_t)$$

其中m_t代表的就是"memory"变量,用于"记住"过去,一般会储存在模型的"memory unit"中

Jordan Network

1986年,M.I.Jordan的论文"Serial order: A parallel distributed processing approach",定义的memory unit为历史所有输出的均值(running average)

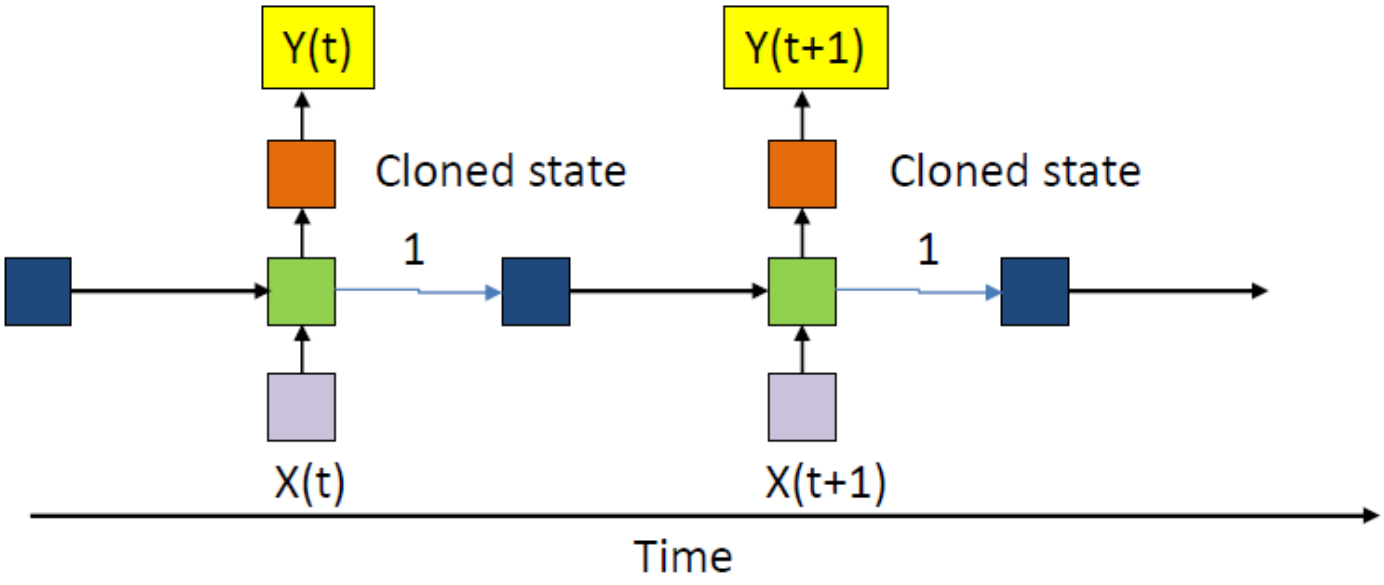


Jordan Network

该模型的"Memory"实际上是一个固定的结构,并不是"学习"记忆,且储存的是所有历史的均值,并不是附近的历史.

Elman Networks

1990年, Jeffrey Elman的论文"Finding structure in time"提出的方式通过学习memory unit到hidden unit的权重的方法



Elman Network

The State-space Model & RNN

State-space Model & Single Hidden Layer RNN

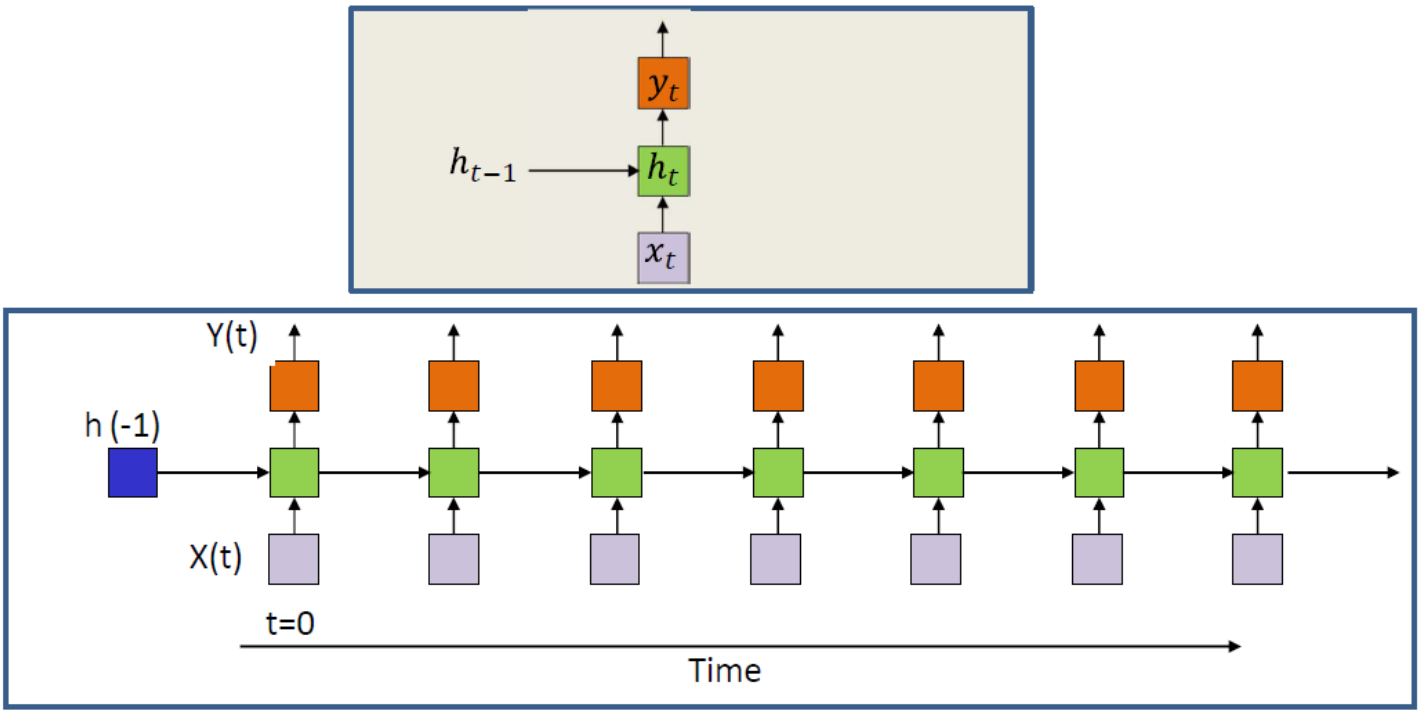
另外一种针对 infinite response system的模型: the state-sapce model

$$h_t = f(x_t, h_{t-1})$$

$$y_t = g(t_t)$$

- 其中h_t表示的神经网络的状态(State), 对历史的所有信息进行了一个整合
  - 模型会直接将memory嵌入该层
- 该模型需要定义初始状态h_{-1}
- 这就是一个完整的RNN模型

最简单的state-space model结构如下,

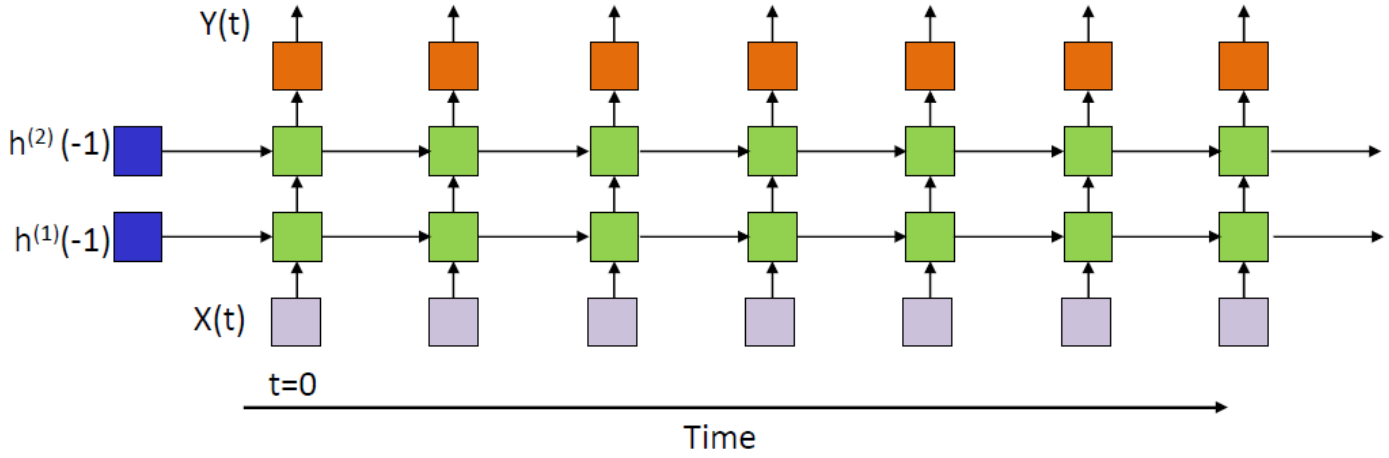


The simple state-space model

- 绿色block表示任意时刻基于input和前序state所定义的state
- $t = 0$时刻的输入会永久影响后续的输出
- 此模型其实就是标准的 `Single Hidden Layer RNN`

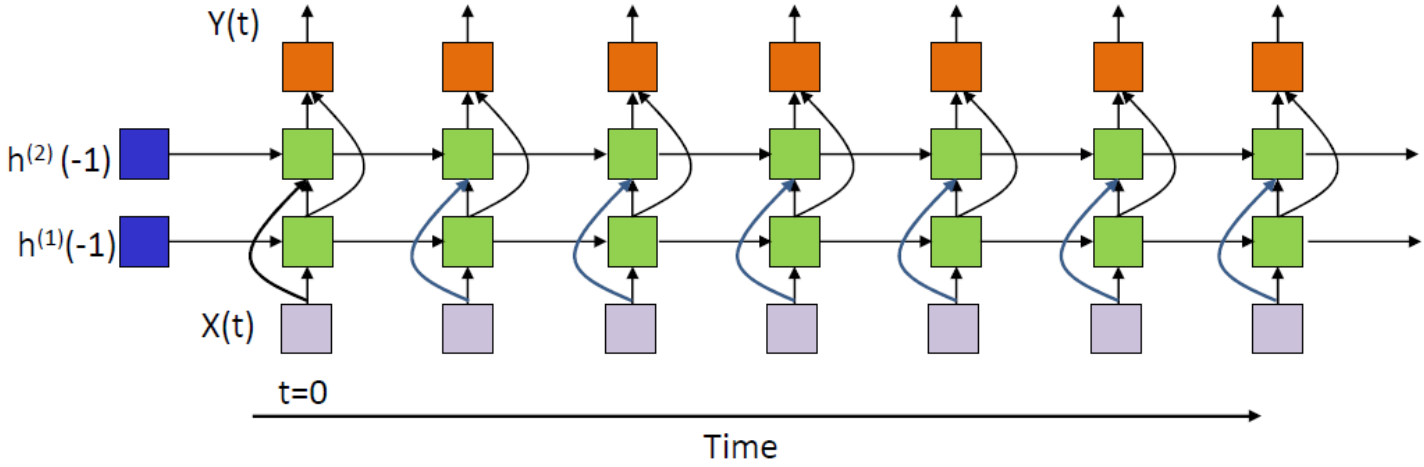
Multiple recurrent layer RNN

多层循环RNN结构如下



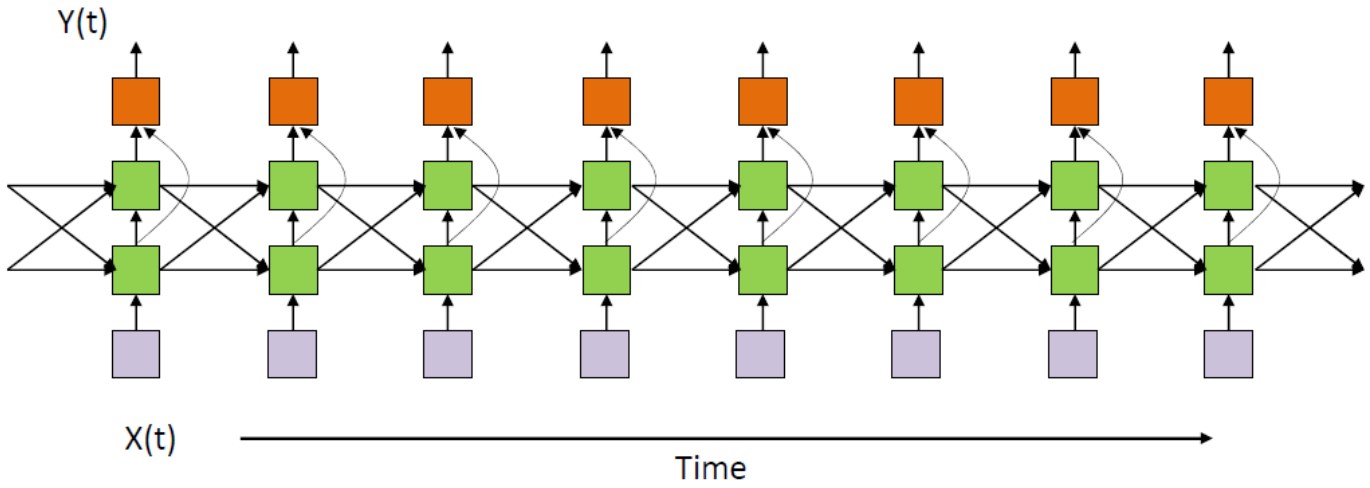
Multiple recurrent layer RNN

或者,



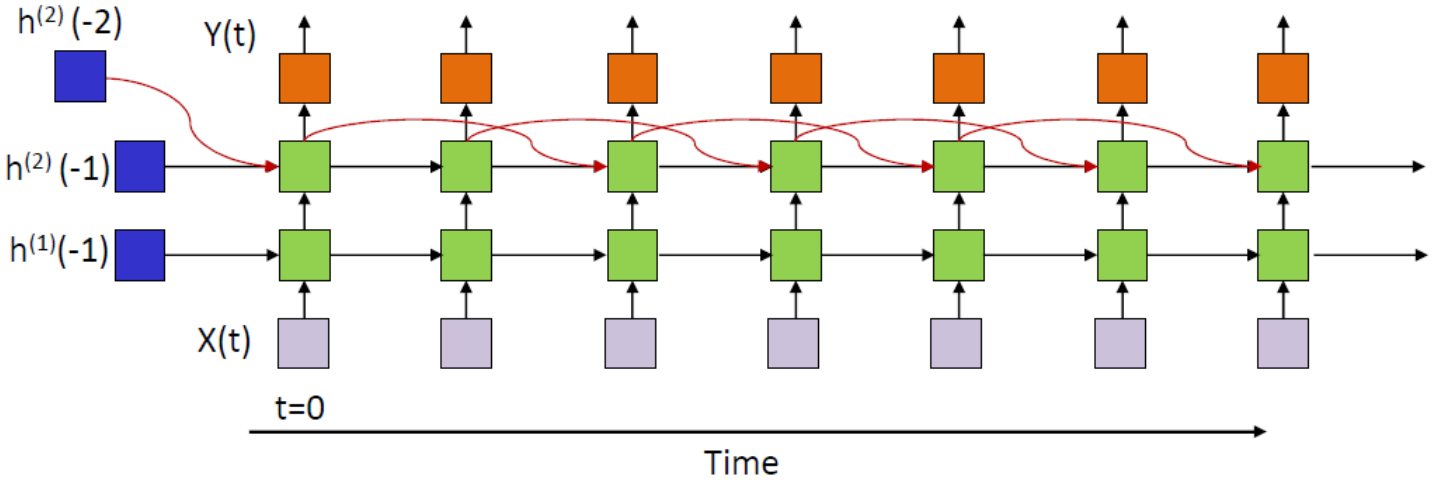
Multiple recurrent layer RNN

甚至是,



Multiple recurrent layer RNN

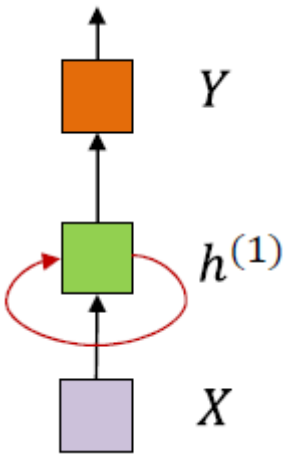
还可以基于其他递归进行泛化,



Generalization with other recurrences

核心公式

针对以下结构,



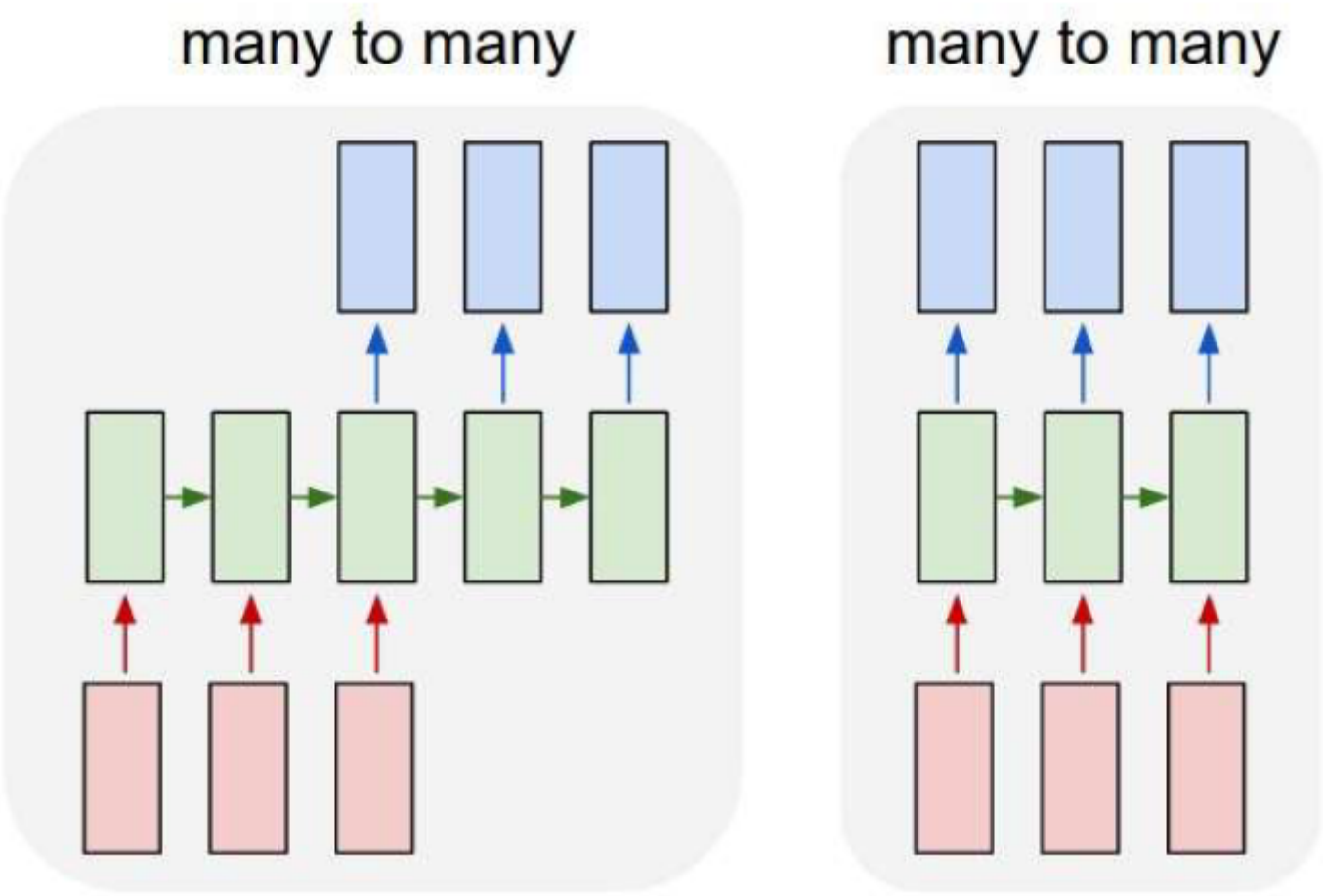
$$h_i^1(-1) = part \quad of \quad network \quad parameters$$

$$h_i^1(t) = f_1(\sum_j w_{ji}^1 X_j(t) + \sum w_{ji}^1 h_i^{(1)}(t-1) + b_i^{(1)})$$

$$Y(t) = f_2(\sum w_{ji}^2 h_i^{(1)}(t-1) + b_i^{(2)}, k = 1..M)$$

RNN的变种

RNN模型通常还有两个变种,



具体应用如下,

1. Delayed Sequence to Sequence, 例如机器翻译(Machine Translation)
2. Sequence to Sequence, 例如股票预测,标签预测等

总结

本文主要总结了RNN的由来和整体的结构细节,具体的训练步骤(如back propagation等)由于公式繁琐的原因,就不在这里阐述,感兴趣的朋友们可以下去自己了解推一推,整体的梯度下降方式和传统的神经网络不会有太大的差异.