

循环神经网络

主讲：林涛



循环神经网络

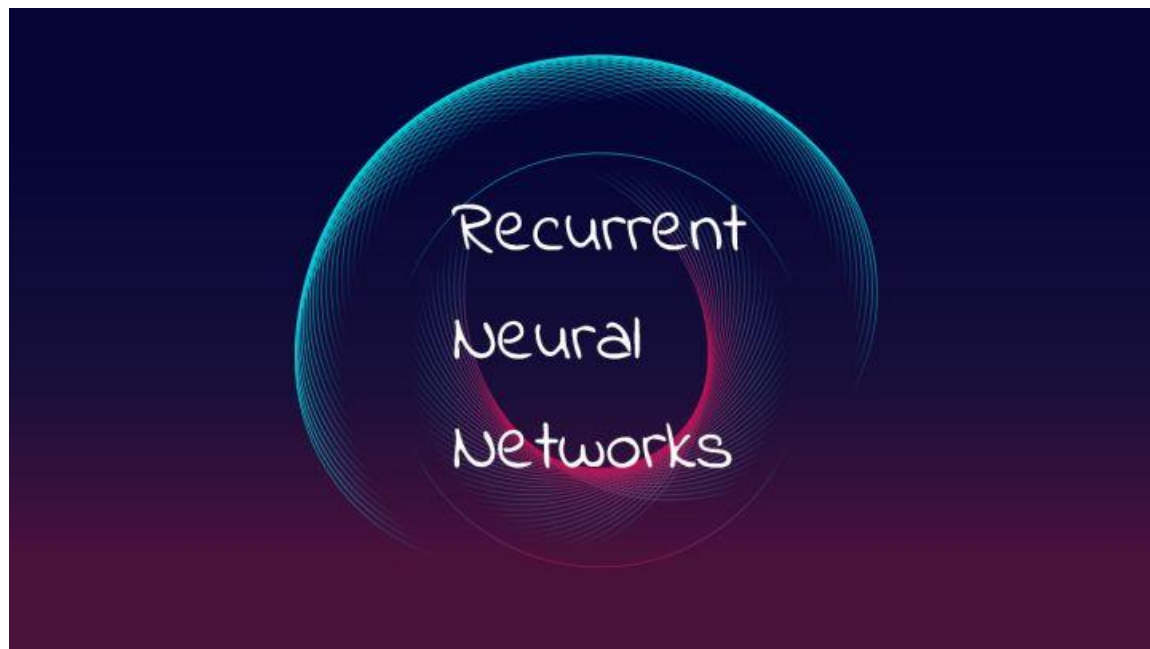
主讲：林涛





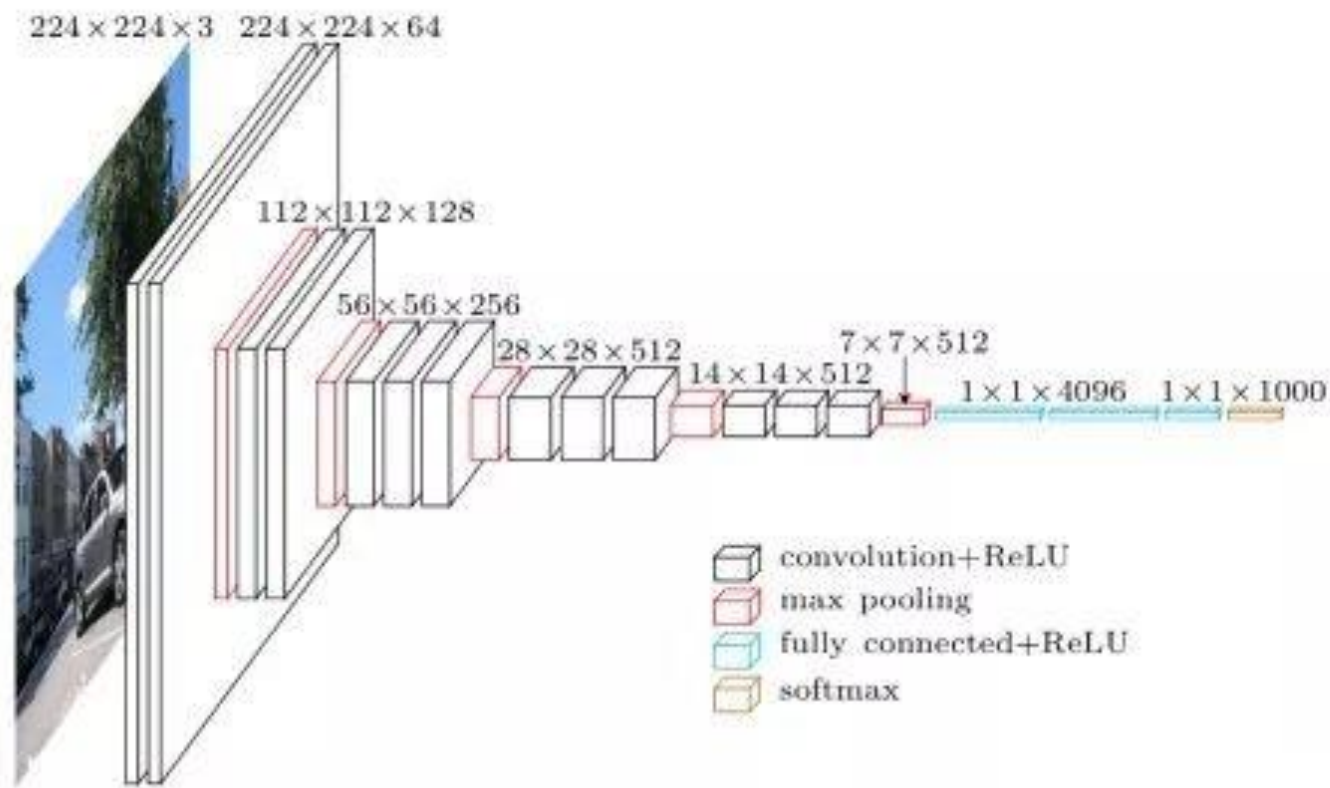
RNN-循环神经网络

1



RNN

传统的CNN，样本的顺序没有体现



但是有些场景，样本出现的顺序是有关联的

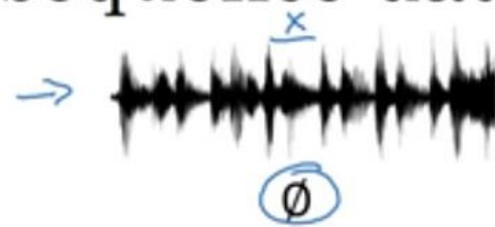
汉字的序顺并不定一能影阅响读

RNN

但是有些场景，样本出现的顺序是有关联的

Examples of sequence data

Speech recognition



“The quick ^ybrown fox jumped
over the lazy dog.”



Music generation

Sentiment classification

“There is nothing to like
in this movie.”



DNA sequence analysis → AGCCCCTGTGAGGAACTAG

AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec
moi?

Do you want to sing with
me?

Video activity recognition



Running



Name entity recognition → Yesterday, Harry Potter
met Hermione Granger.

Yesterday, **Harry Potter**
met **Hermione Granger**.

<https://blog.csdn.net/AndrewNg>

RNN

时序有关的应用

小明在上海一所大学奋斗，每到花开时，他就跑得很快，毕竟还是太年轻，有时显得太幼稚，因为他担心制造业在大城市难以立足。学长告诉他，东川路男子职业技术大学的学生的个人奋斗固然重要，但也要考虑历史的进程，我们国家将集中力量在信息科学、神经科学、人工智能、数学等方面取得新进展。所以小明坚定信念，选择了人工智能的数学基础，为新一代制造业努力奋斗。

问题

- 1 “他跑得特别快”的“他”指谁？
- 2 “一所大学”指什么大学？
- 3 “因为”和“所以”是因果关系吗？

情感分类

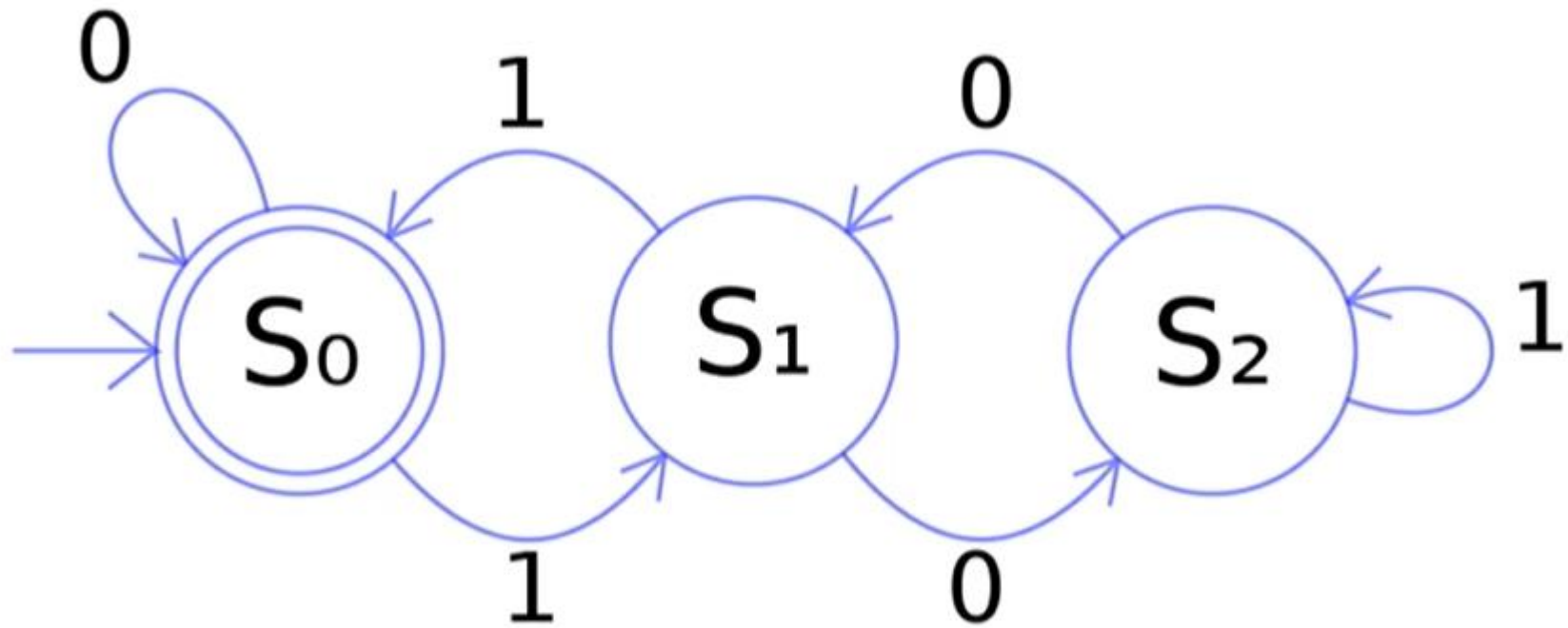
“这家店装修不错” -RNN-正

“这家店不太行” -RNN-负

股票分析

有限状态自动机

有限状态自动机



图灵机

图灵机是一种数学计算模型，定义了一个抽象的机器，该机器可以根据指定的规则表在纸带上进行操作。虽然图灵机模型非常简单，但是图灵机可以模拟任何给定的计算机算法



TDNN（时延神经网络）

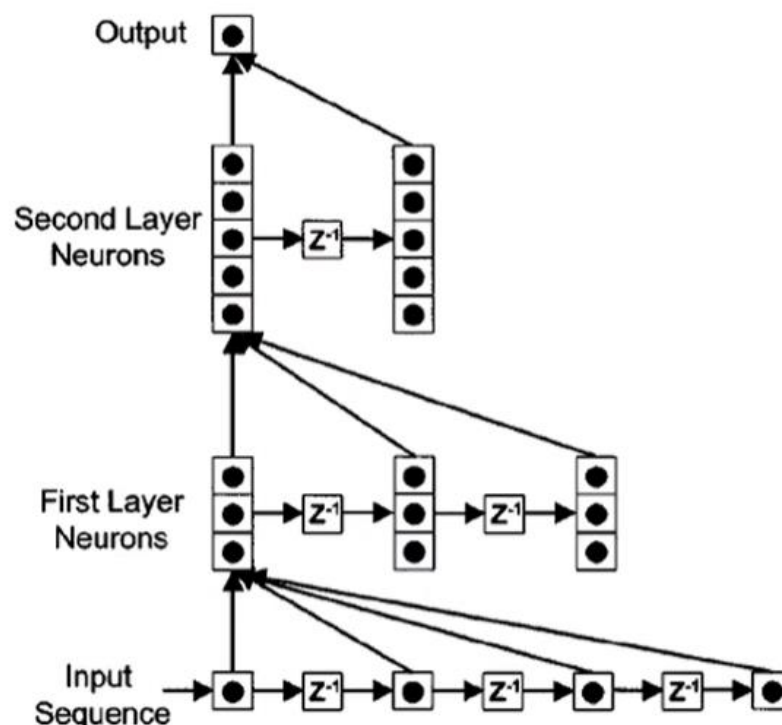
增加延时单元

► 时延神经网络 (Time Delay Neural Network, TDNN)

- 建立一个额外的延时单元，用来存储网络的历史信息（可以包括输入、输出、隐状态等）

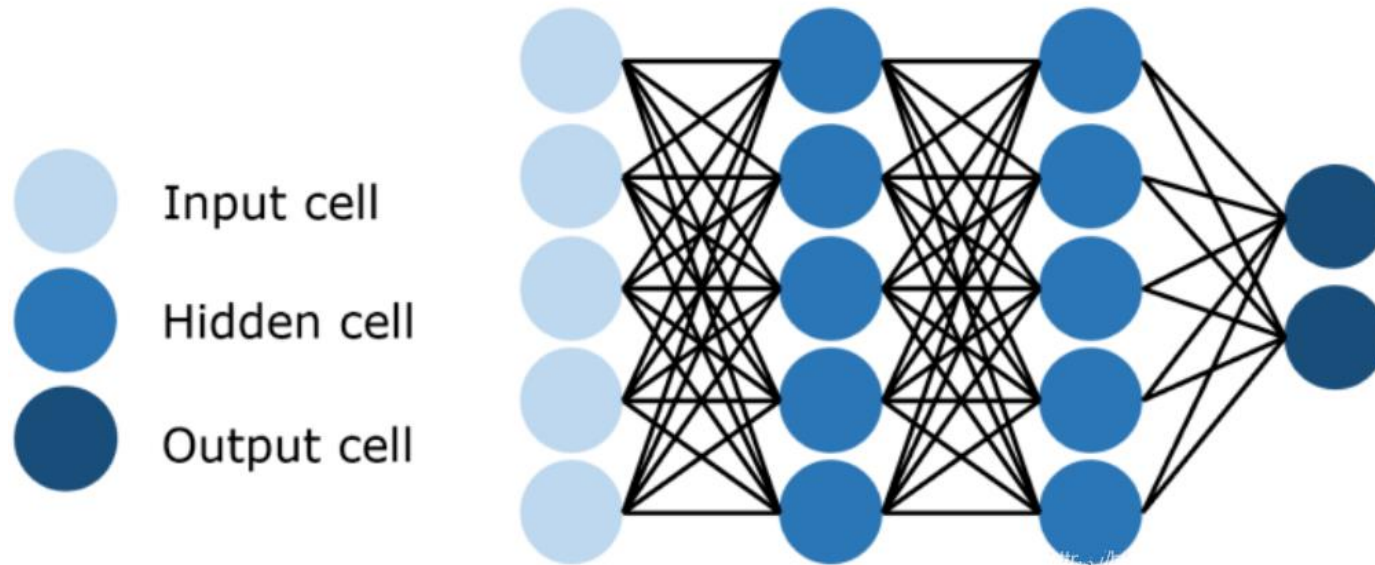
$$h_t^{(l)} = f(h_t^{(l-1)}, h_{t-1}^{(l-1)}, \dots, h_{t-K}^{(l-1)})$$

- 这样，前馈网络就具有了短期记忆的能力。



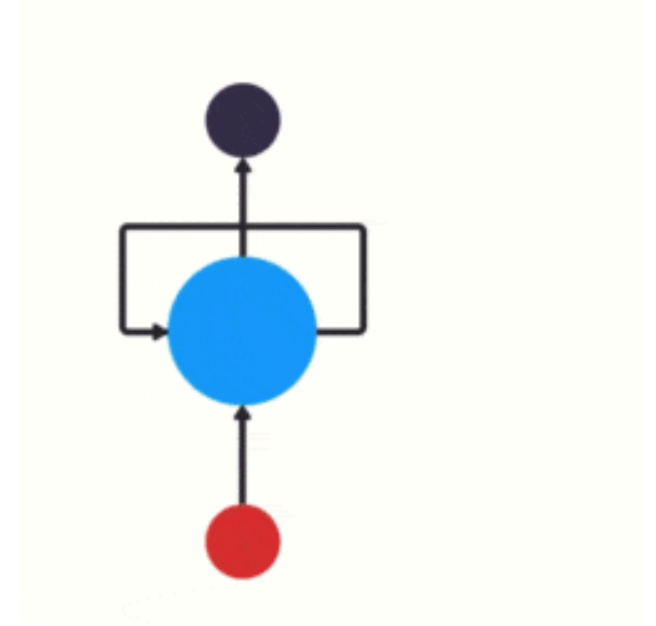
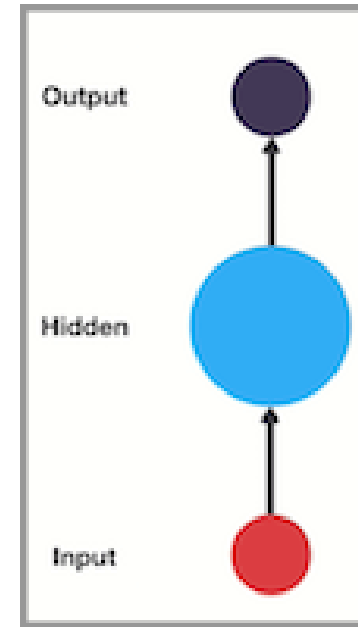
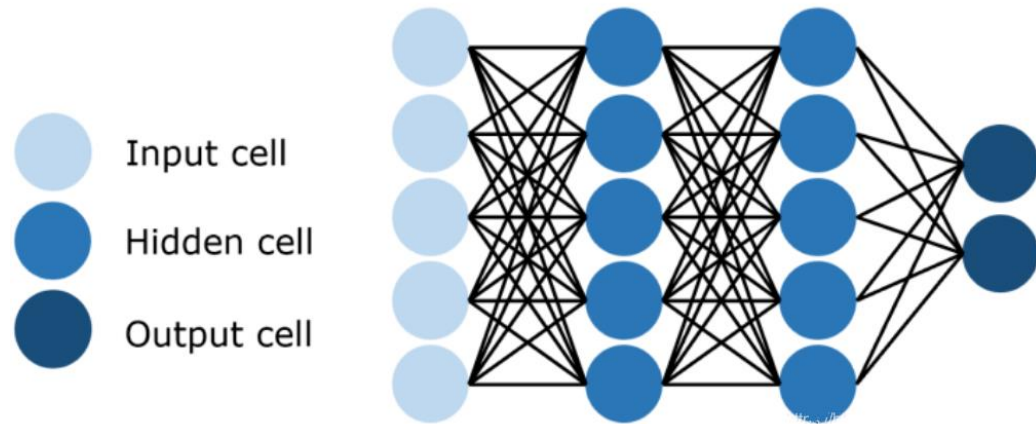
RNN

为何传统的神经网络无法处理序列数据
传统的神经网络结构是固定的



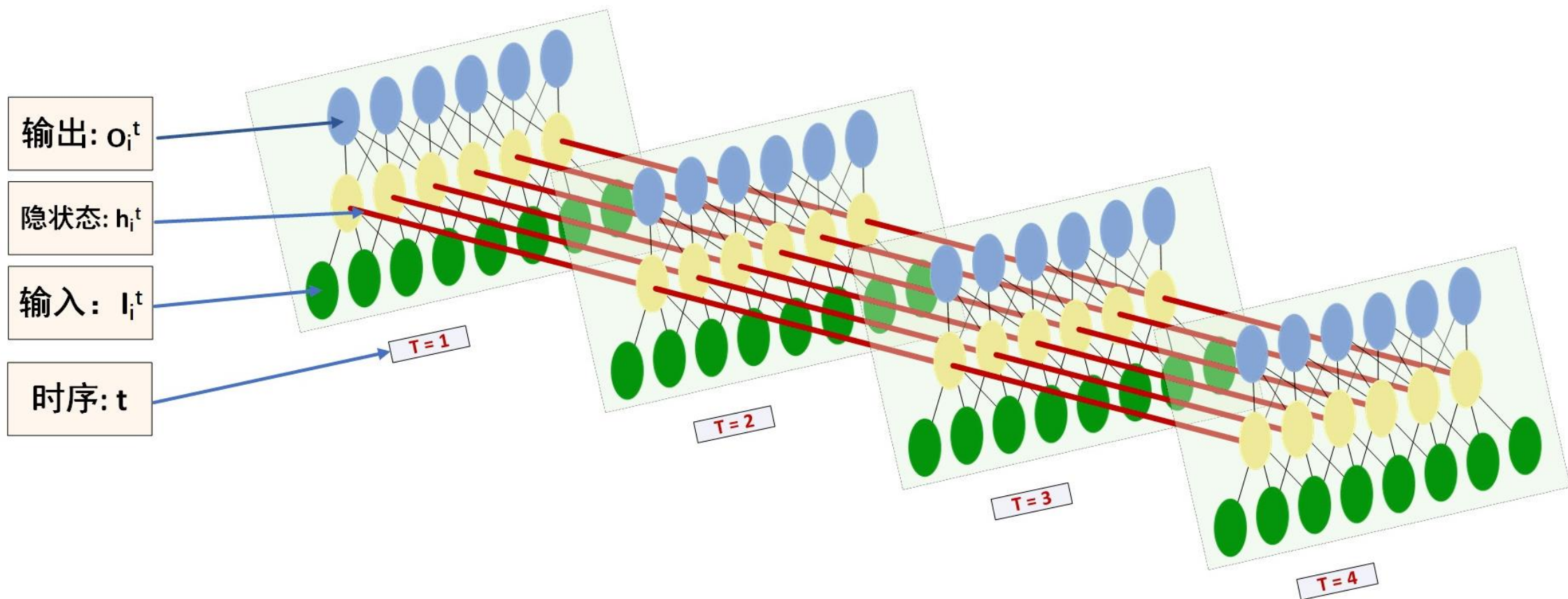
RNN

RNN是如何处理可变输入的。让连续样本之间建立联系



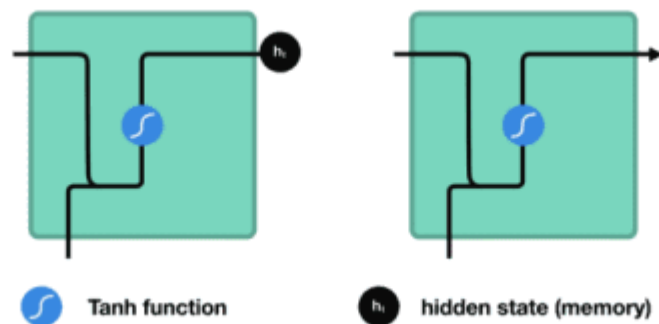
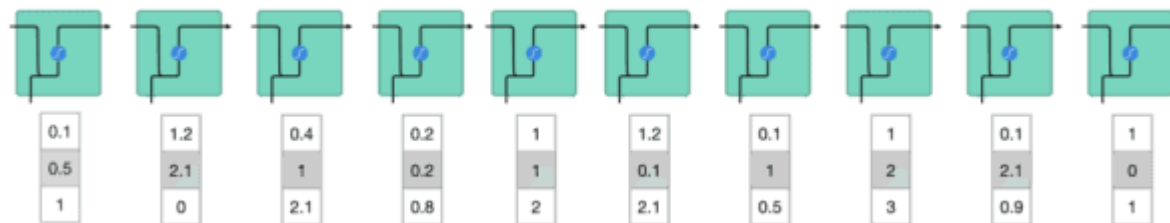
RNN

RNN中，让连续样本之间建立联系



RNN

RNN中，让连续样本之间建立联系

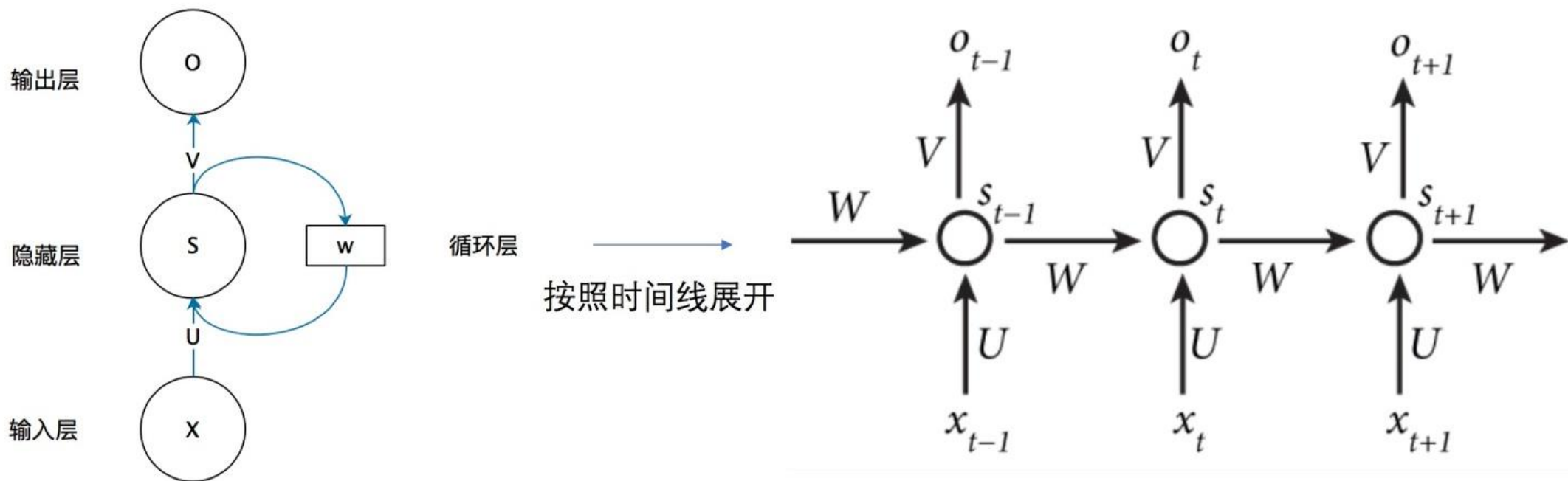


RNN

RNN中，让连续样本之间建立联系



RNN



x 是一个向量，它表示输入层的值， s 是一个向量，它表示隐藏层的值， U 是输入层到隐藏层的权重矩阵， o 也是一个向量，它表示输出层的值； V 是隐藏层到输出层的权重矩阵；循环神经网络的隐藏层的值 s 不仅仅取决于当前这次的输入 x ，还取决于上一次隐藏层的值 s 。权重矩阵 W 就是隐藏层上一次的值作为这一次的输入的权重。

标准RNN的还有以下特点：

- 1、权值共享，图中的 W 全是相同的， U 和 V 也一样。
- 2、每一个输入值 x_t 都只与它本身的那条路线建立权连接，不会和别的神经元连接。

RNN的反向传播

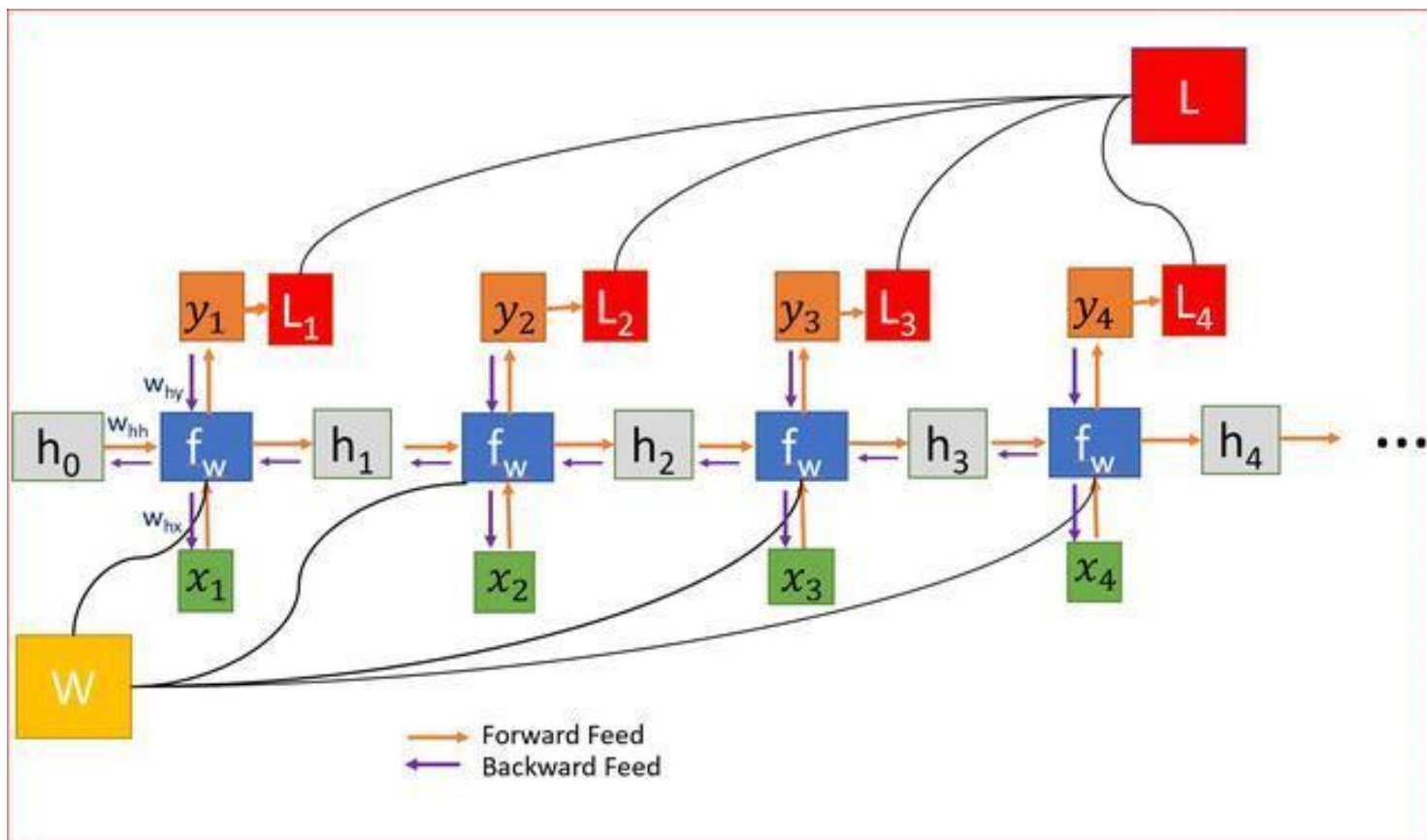
将RNN展开之后，前向传播就是依次按照时间的顺序计算一次就好了，反向传播就是从最后一个时间将累积的损失传递回来即可，这与普通的神经网络训练本质上是相似的。即通过梯度下降法一轮轮的迭代，得到合适的RNN模型参数 U, W, V, b, c 。由于基于时间反向传播，所以RNN的反向传播有时也叫做BPTT(back-propagation through time)。这里的BPTT和普通神经网络也有很大的不同点，即这里所有的 U, W, V, b, c 在序列的各个位置是共享的，反向传播时更新的是相同的参数

$$E = \sum_t e_t$$
$$\nabla U = \frac{\partial E}{\partial U} = \sum_t \frac{\partial e_t}{\partial U}$$
$$\nabla V = \frac{\partial E}{\partial V} = \sum_t \frac{\partial e_t}{\partial V}$$
$$\nabla W = \frac{\partial E}{\partial W} = \sum_t \frac{\partial e_t}{\partial W}$$

<https://blog.csdn.net/qin032244189>

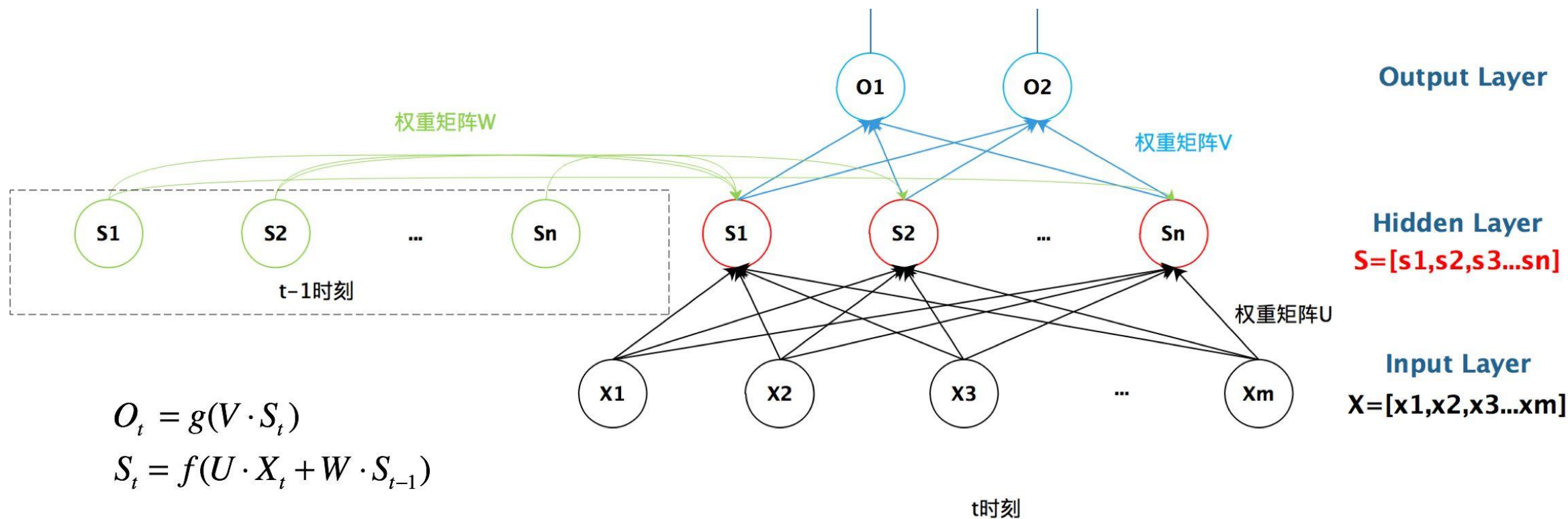
RNN

RNN前向及反向传播示意图



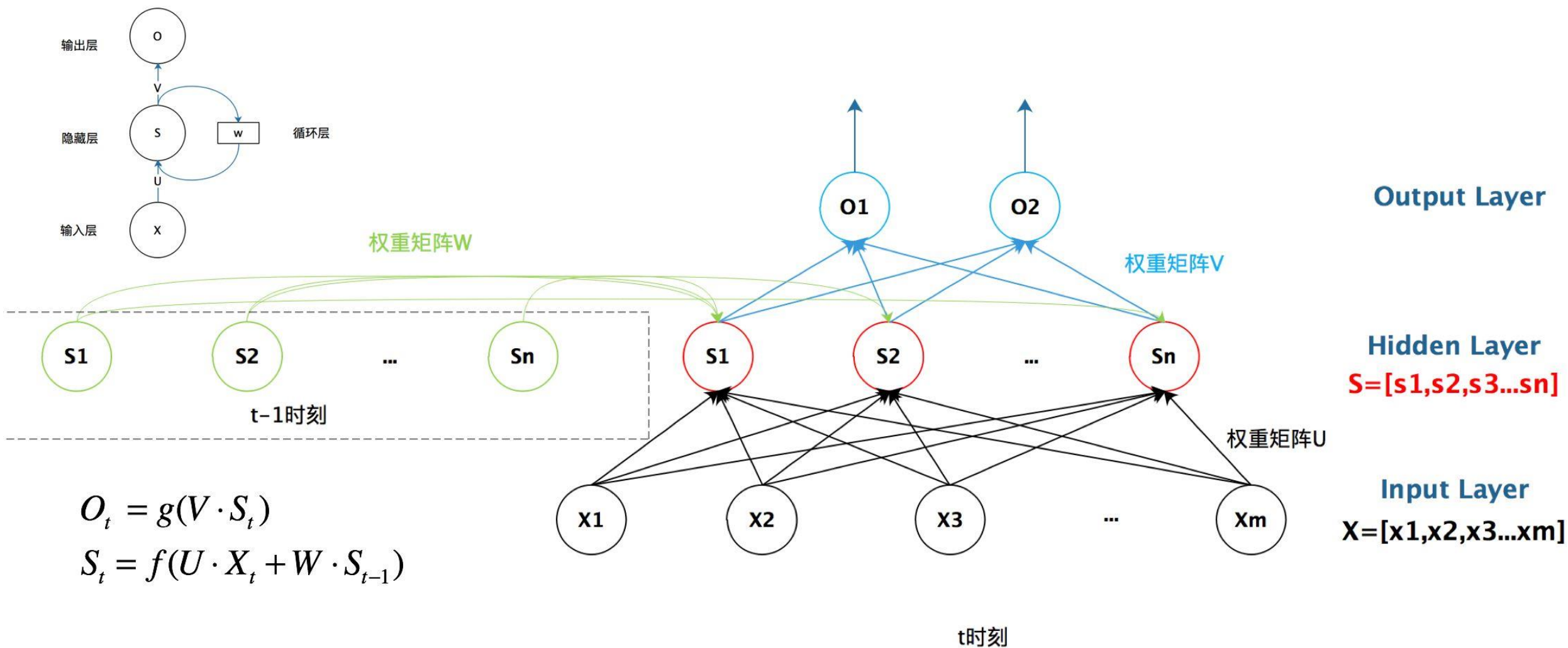
RNN

RNN中，t-1时刻的权重 w 与t时刻的关系



RNN

RNN结构图及理论公式全图



RNN

RNN中，输入和输出的关系可以是多种情况

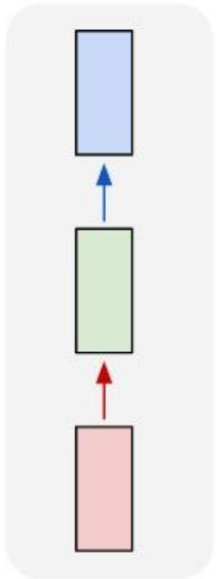
Image Captioning: image \rightarrow sequence of words (one to many)

Sentiment Classification: sequence of words \rightarrow sentiment (many to one)

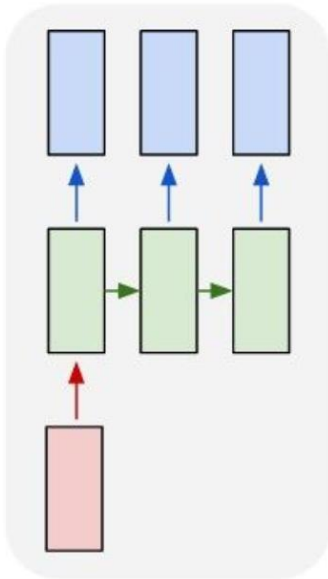
Machine Translation: seq of words \rightarrow seq of words (many to many)

Video classification on frame level : many to many

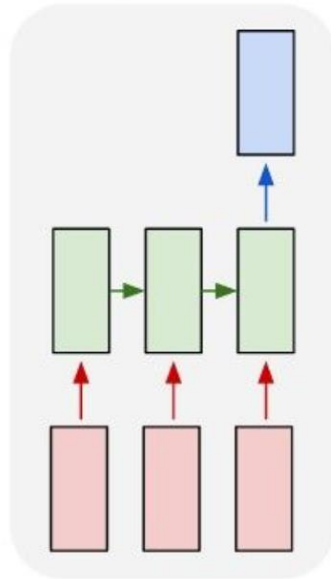
one to one



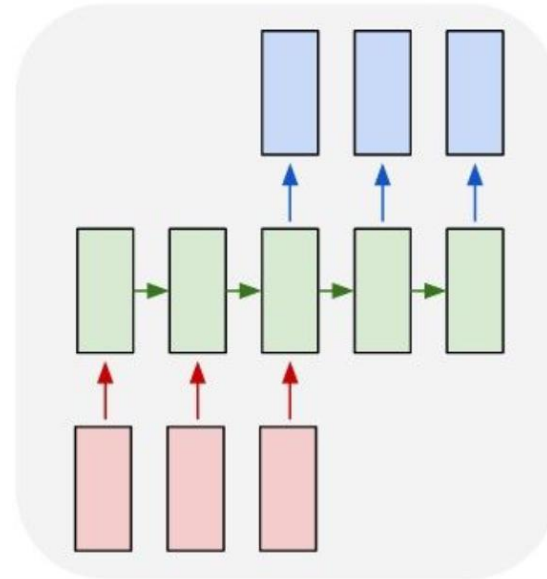
one to many



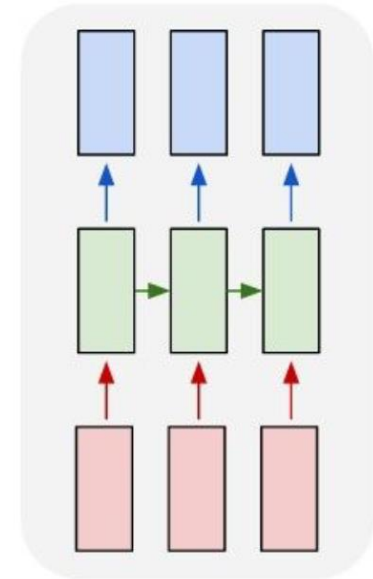
many to one



many to many



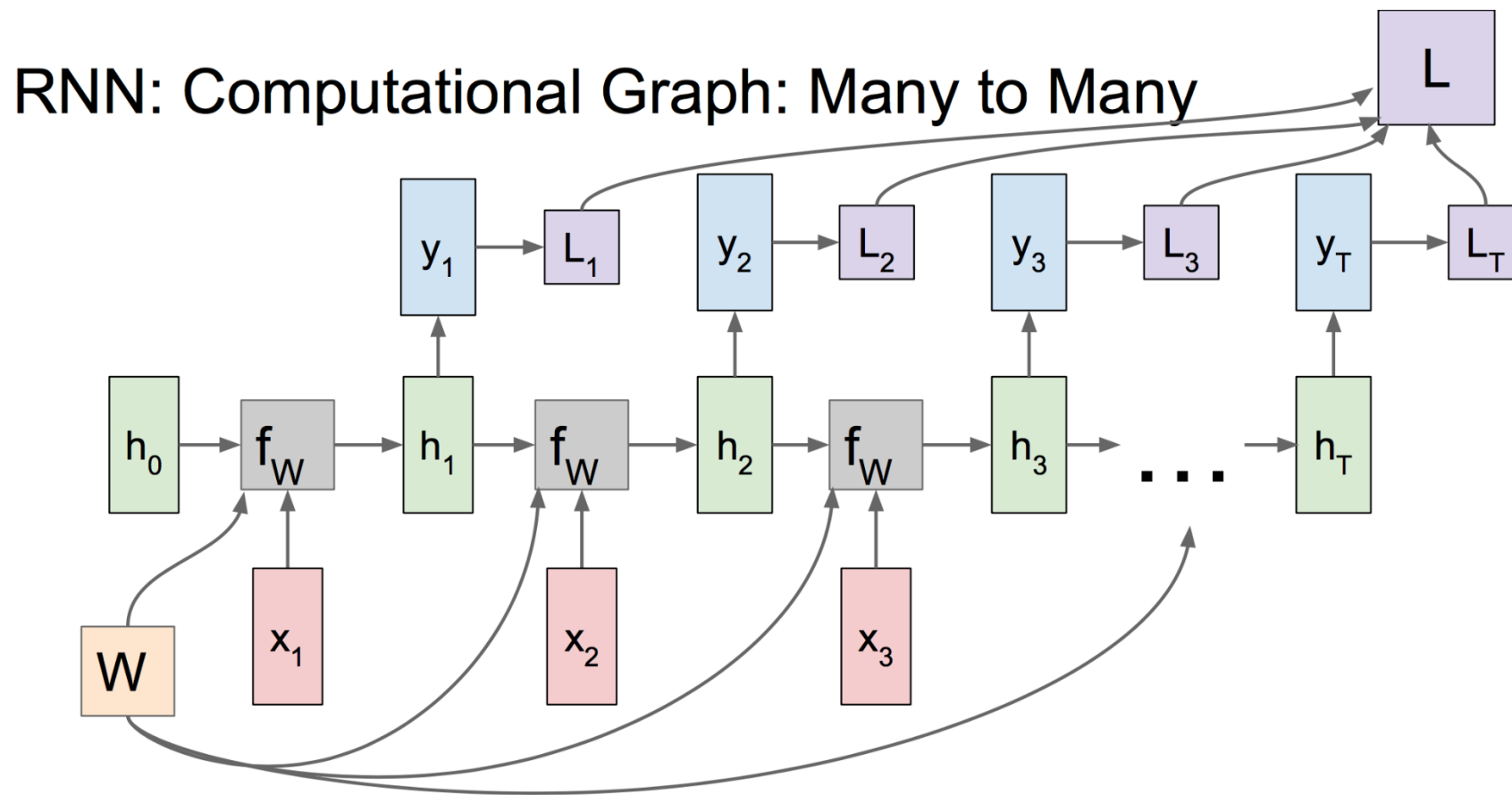
many to many



RNN

输出定长的many to many的RNN

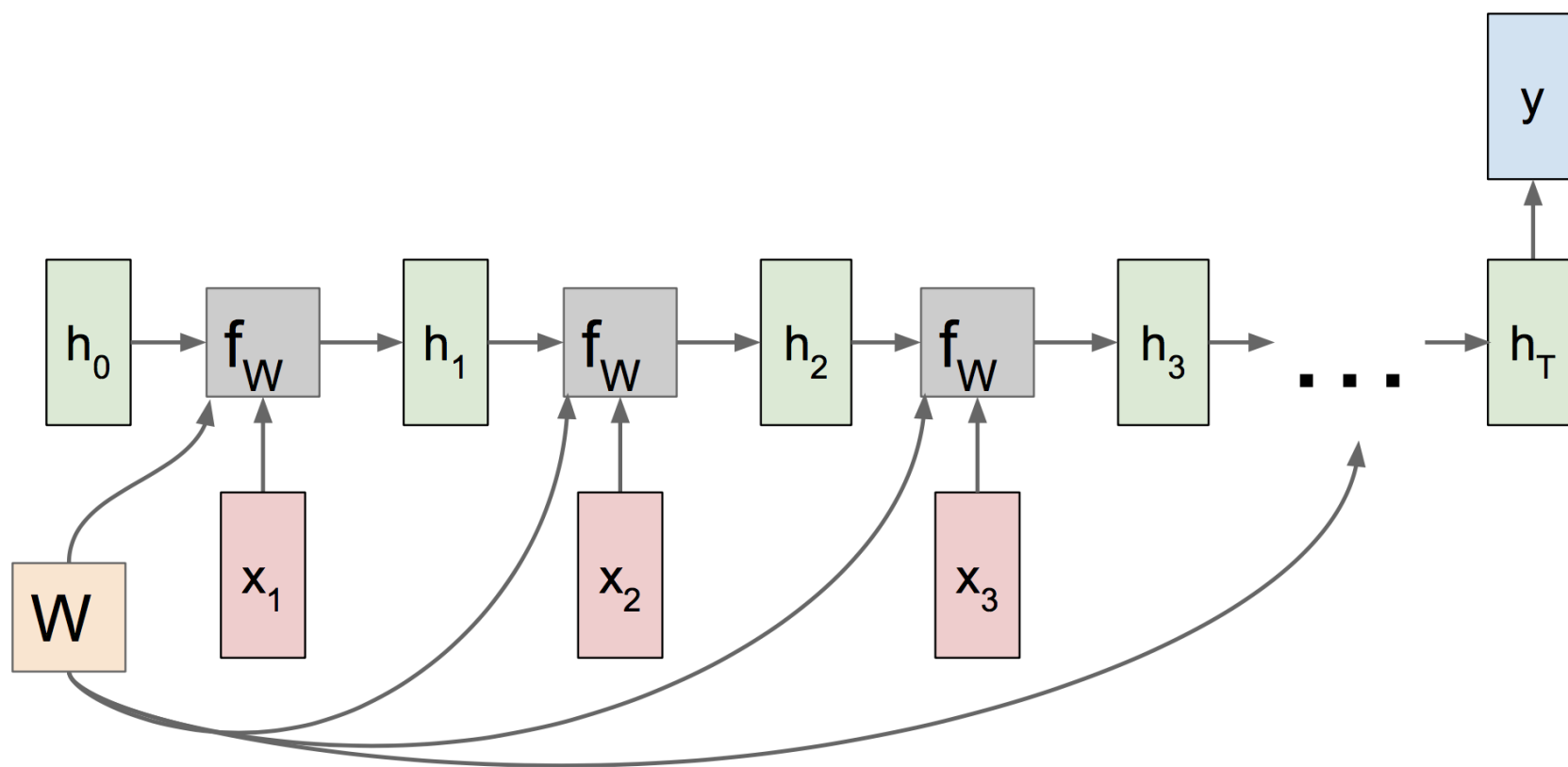
参数 W 在各个时刻是共享的，因此当反向传播时， W 的梯度应该是各个时刻传过来的梯度之和。



RNN

many2one的计算图:

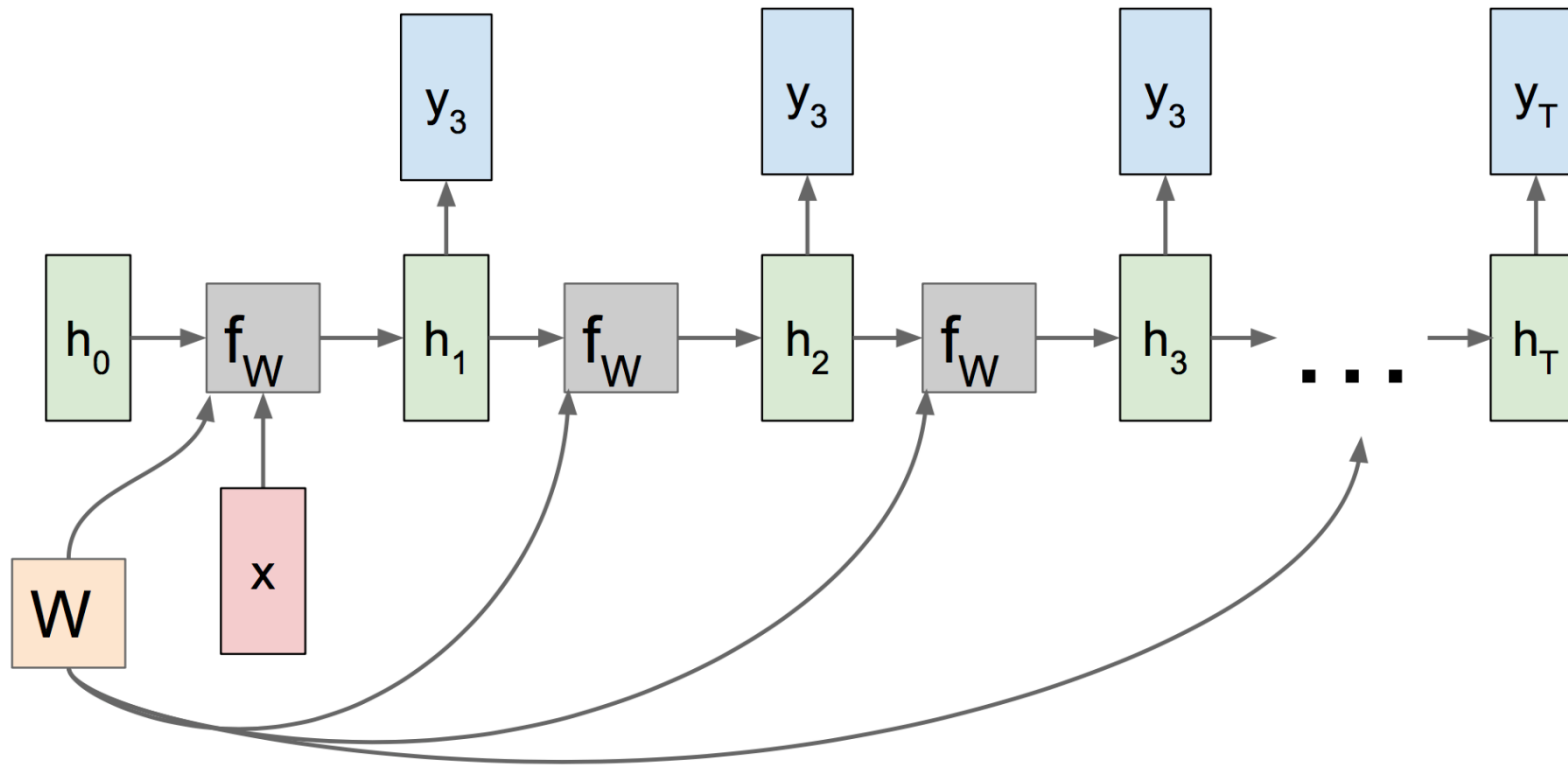
输入是一段话，输出是对这段话的语气判断



RNN

one2many的计算图:

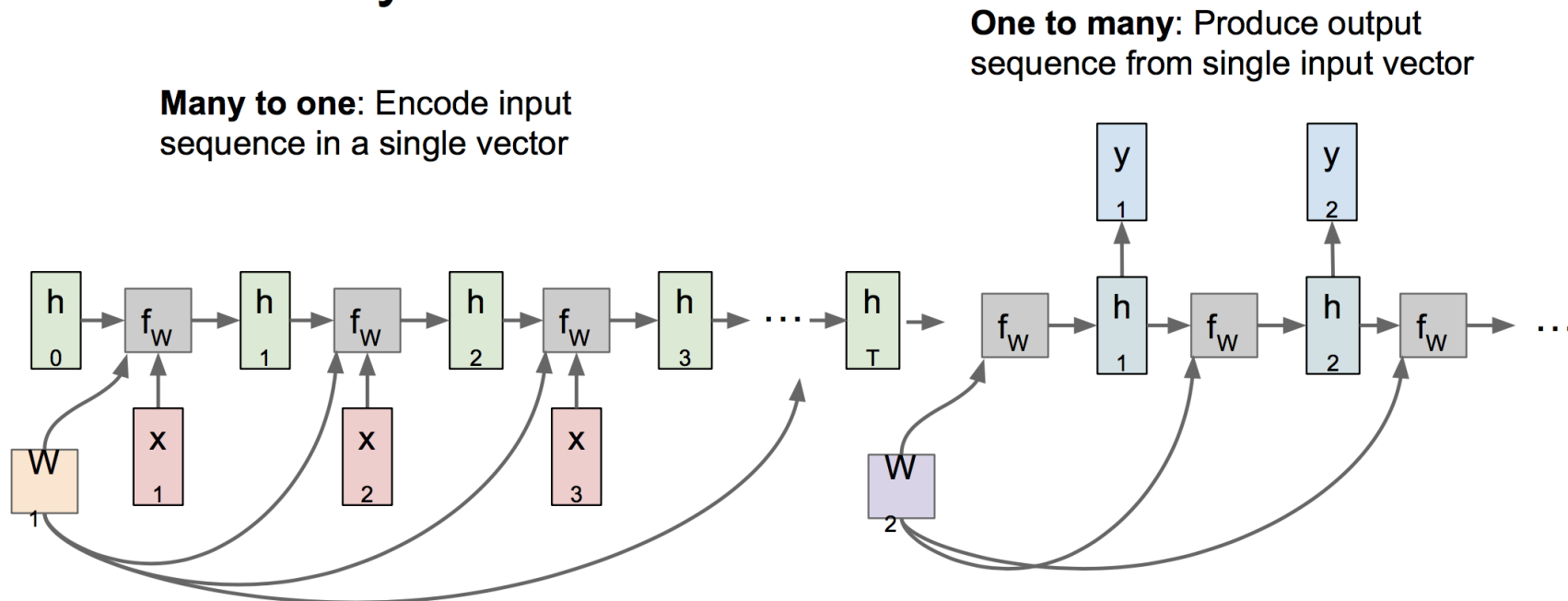
输入一张图片，输出是一段对图片的描述，看图说话



RNN

seg2seg的计算图，可以被拆解为many2one和one2many两部分。其中many2one部分被称为编码器，将输入 x 编码成一个定长的向量。 $one2many$ 被称为解码器，负责将该向量转化成输出序列。如下：

Sequence to Sequence: Many-to-one + one-to-many



RNN

RNN模型的数学表示：

每个时刻函数 f 接收上一时刻的隐藏态 h_{t-1} 和当前时刻的输入 x_t ，产生当前时刻的隐藏态 h_t 。其中函数 f 是关于参数 w 的函数，在每个时刻参数 w 和函数 f 都应该是相同的。输出 y 可以由当前的隐藏态产生。

We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

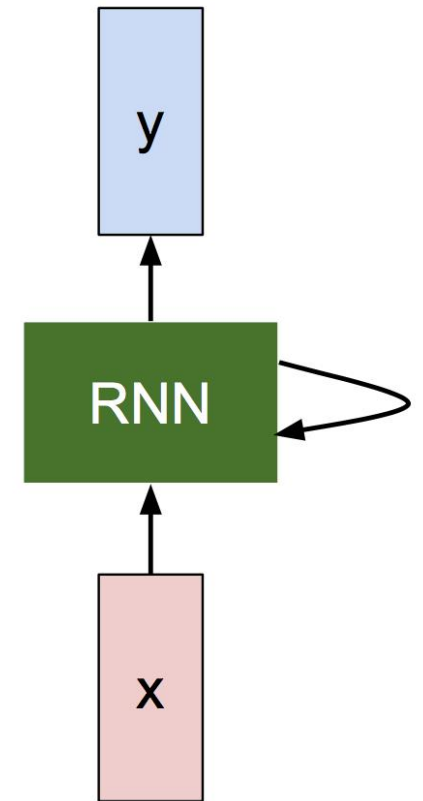
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

some function with parameters W

old state

input vector at some time step



RNN

一个RNN模型的简单例子如下：

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

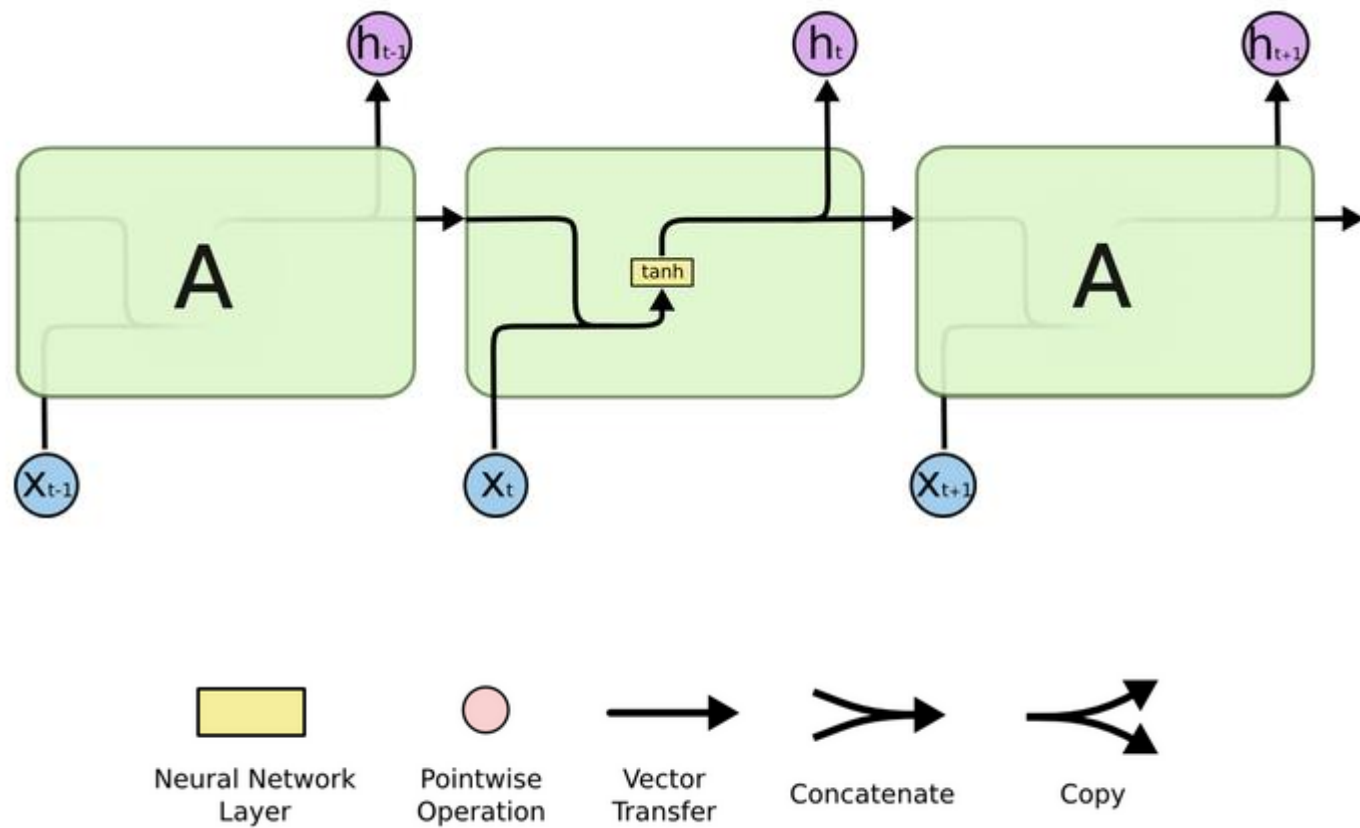


$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

RNN

RNN神经元内部结构：



RNN

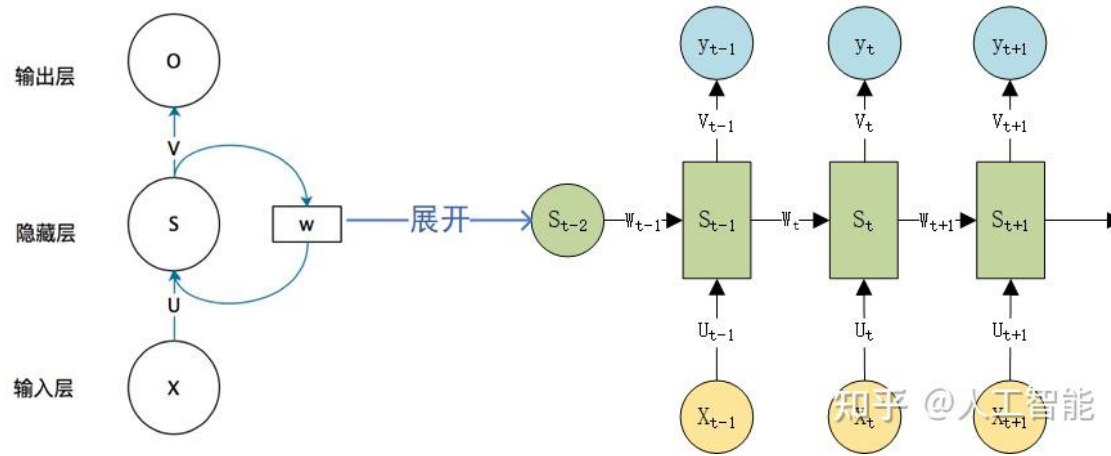
梯度消失和梯度爆炸

$$1.01^{365} = 37.8$$

$$0.99^{365} = 0.03$$

RNN

梯度消失和梯度爆炸



对应的前向传播公式和每个时刻的输出公式

$$S_t = \tanh(UX_t + WS_{t-1}) \quad y_t' = \text{softmax}(VS_t)$$

使用交叉熵为损失函数，对应的每个时刻的损失和总的损失。通常将一整个序列（一个句子）作为一个训练实例，所以总的误差就是各个时刻（词）的误差之和

$$L_t = -y_t \log y_t' = - \sum_i y_{t,i} \log(y'_{t,i})$$
$$L = \sum_t L_t = - \sum_t y_t \log(y_t')$$

RNN

RNN使用交叉熵作为损失函数，为什么？

交叉熵是什么？

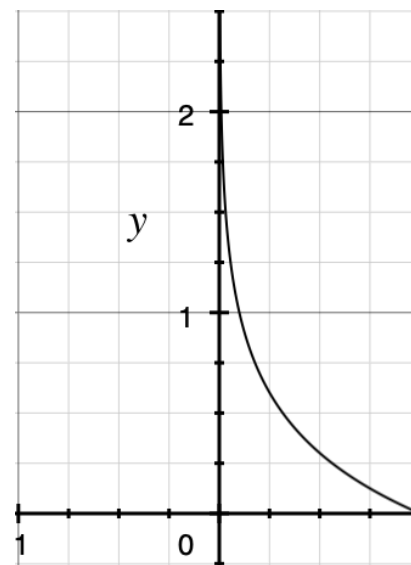
熵是什么？

混乱程度，越是混乱，熵值越高

信息熵，用来衡量信息的不确定性，不确定性越高，信息熵值越大

事件 x_0 的信息量 $I(x_0)$ ， $p(x_0)$ 表示事件 x_0 发生的概率

$$I(x_0) = -\log(p(x_0))$$



RNN

信息量是对于单个事件来说的，但是实际情况一件事有很多种发生的可能，比如掷骰子有可能出现6种情况，明天的天气可能晴、多云或者下雨等等。熵是表示随机变量不确定的度量，是对所有可能发生的事件产生的信息量的期望。公式如下：

$$H(X) = - \sum_{i=1}^n p(x_i) \log(p(x_i))$$

n表示事件可能发生的情况总数

其中一种比较特殊的情况就是掷硬币，只有正、反两种情况，该种情况（二项分布或者0-1分布）熵的计算可以简化如下： $p(x)$ 代表掷正面的概率， $1-p(x)$ 则表示掷反面的概率（反之亦然）

$$\begin{aligned} H(X) &= - \sum_{i=1}^n p(x_i) \log(p(x_i)) \\ &= -p(x) \log(p(x)) - (1 - p(x)) \log(1 - p(x)) \end{aligned}$$

RNN

相对熵又称KL散度，用于衡量对于同一个随机变量 x 的两个分布 $p(x)$ 和 $q(x)$ 之间的差异。在机器学习中， $p(x)$ 常用于描述样本的真实分布，例如 $[1,0,0,0]$ 表示样本属于第一类，而 $q(x)$ 则常常用于表示预测的分布，例如 $[0.7,0.1,0.1,0.1]$ 。显然使用 $q(x)$ 来描述样本不如 $p(x)$ 准确， $q(x)$ 需要不断地学习来拟合准确的分布 $p(x)$ 。KL散度的公式

$$D_{KL}(p||q) = \sum_{i=1}^n p(x_i) \log\left(\frac{p(x_i)}{q(x_i)}\right)$$

n 表示事件可能发生的情况总数

KL散度的值越小表示两个分布越接近。

将KL散度的公式进行变形

$$\begin{aligned} D_{KL}(p||q) &= \sum_{i=1}^n p(x_i) \log(p(x_i)) - \sum_{i=1}^n p(x_i) \log(q(x_i)) \\ &= -H(p(x)) + \left[-\sum_{i=1}^n p(x_i) \log(q(x_i))\right] \end{aligned}$$

RNN

前半部分就是 $p(x)$ 的熵，后半部分就是交叉熵：

$$H(p, q) = - \sum_{i=1}^n p(x_i) \log(q(x_i))$$

机器学习中，常常使用KL散度来评估predict和label之间的差别，但是由于KL散度的前半部分是一个常量，所以常常将后半部分的交叉熵作为损失函数，其实二者是一样的。

用交叉熵做损失函数的好处是误差越大，梯度下降的越快，而MSE（均方差）做损失函数，则会出现误差越大，学习的越慢的情况。

RNN

将RNN前向传播的各公式分布展开如下：

$$S_t = \tanh(UX_t + WS_{t-1})$$
$$z_t = VS_t$$
$$y_t' = \text{softmax}(z_t)$$
$$L_t = -y_t \log y_t' = -\sum_i y_{t,i} \log(y_{t,i}') \quad (4)$$
$$L = \sum_t L_t \quad (5)$$

| 符号 | 解释 |
|---------------------------------|--|
| K | 词汇表的大小 |
| T | 句子长度 |
| H | 隐藏层大小 |
| z_t | 长度为K的vector |
| y_t | 长度为K的vector,表示真实的标签，一般是one-vector |
| $y_{t,i}$ | 对应的第i个词的标签值 |
| y_t' | 长度为K的vector,表示预测的向量 |
| $y_{t,i}'$ | 表示生成的词在是词表的第i个词的概率 |
| L_t | 当前时刻的损失 |
| L | 一个句子的损失，由各个时刻的损失求和得到， $L = \sum_t L_t$ |
| $V \in \mathbb{R}^{K \times H}$ | 隐藏层到输出层的权重 |
| $W \in \mathbb{R}^{H \times K}$ | 上一个隐藏层状态到当前层的输入的权重 |
| $U \in \mathbb{R}^{H \times H}$ | 输入的权重 |

RNN

对V的导数:

$$\left\{ \begin{array}{l} \frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V} \end{array} \right. \quad (6)$$

$$\left\{ \begin{array}{l} L_t = -y_t \log y_t' = -\sum_i y_{t,i} \log(y'_{t,i}) \end{array} \right. \quad (7)$$

$$\left\{ \begin{array}{l} y'_{t,i} = \frac{e^{z_{t,i}}}{\sum_k e^{z_{t,k}}} \end{array} \right. \quad (8)$$

为了得到公式(6)的结果, 通过链式求导法则, 拆解为如下两个公式, 我们真正要求解的是、
、
是这3项的值

$$\left\{ \begin{array}{l} \frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial z_t} \frac{\partial z_t}{\partial V} \end{array} \right. \quad (9)$$

$$\left\{ \begin{array}{l} \frac{\partial L_t}{\partial z_t} = \frac{\partial L_t}{\partial y_t'} \frac{\partial y_t'}{\partial z_t} \end{array} \right. \quad (10)$$

对V的导数:

$\frac{\partial L_t}{\partial y_{t,i}'}$ 【公式(4)对应的求导】和 $\frac{\partial z_t}{\partial V}$ 【公式(2)对应的求导】的值如下:

$$\begin{cases} \frac{\partial L_t}{\partial y_{t,i}'} = - \sum_{t,i} \frac{y_{t,i}}{y_{t,i}'} & (11) \\ \frac{\partial z_t}{\partial V} = S_t & (12) \end{cases}$$

$\frac{\partial y_{t,i}'}{\partial z_{t,i}}$ 求解相对复杂一些。由于 z_t 是一个向量，所以求导分为两种情况:

- 如果当前的label是第*i*个类别，那么*i*对应的位置的交叉熵。
- 如果当前的label不是第*i*个类别，那么对应另外一种情况。

1)如果 $i = j$:第*i*位置的交叉熵，同时需要将公式(8)带入到如下公式中

$$\frac{\partial y_{t,i}'}{\partial z_{t,i}} = \frac{e^{z_{t,i}} \sum_k e^{z_{t,k}} - e^{z_{t,i}} * e^{z_{t,i}}}{(\sum_k e^{z_{t,k}})^2} = \frac{e^{z_{t,i}}}{\sum_k e^{z_{t,k}}} \left(1 - \frac{e^{z_{t,i}}}{\sum_k e^{z_{t,k}}}\right) = y_{t,i}' (1 - y_{t,i}') \quad (13)$$

2)如果 $i \neq j$:其他位置的交叉熵

$$\frac{\partial y_{t,j}'}{\partial z_{t,i}} = - \frac{e^{z_{t,j}} e^{z_{t,i}}}{(\sum_k e^{z_{t,k}})^2} = - \frac{e^{z_{t,j}}}{\sum_k e^{z_{t,k}}} \frac{e^{z_{t,i}}}{\sum_k e^{z_{t,k}}} = -y_{t,j}' y_{t,i}' \quad (14)$$

对v的导数:

偏导数的值, 带入【公式(11)、(13)、(14)】得到如下公式, 对于由【公式(c)到(d)的过程】, $\sum_{t,i} y_{t,i} = y_{t,i}$, 因为除了 $y_{t,i}$ 为1, 其他label均为0, 所以等式成立。

$$\left\{ \begin{array}{l} \frac{\partial L_t}{\partial z_t} = \left(- \sum_{t,i} \frac{y_{t,i}}{y'_{t,i}} \right) \frac{\partial y'_{t,i}}{\partial z_{t,i}} + \sum_{i,i \neq j} \frac{y_{t,i}}{y'_{t,j}} y'_{t,i} y'_{t,j} \quad (a) \\ = \left(- \sum_{t,i} \frac{y_{t,i}}{y'_{t,i}} \right) y'_{t,i} (1 - y'_{t,i}) + \sum_{i,i \neq j} \frac{y_{t,i}}{y'_{t,j}} y'_{t,i} y'_{t,j} \quad (b) \\ = - \sum_{t,i} y_{t,i} + \sum_{t,i} y_{t,i} y'_{t,i} + \sum_{i,i \neq j} y_{t,i} y'_{t,i} \quad (c) \\ = -y_{t,i} + y'_{t,i} \sum_i y_{t,i} \quad (d) \\ = y'_{t,i} - y_{t,i} \quad (15) \end{array} \right.$$

RNN

对V的导数:

在t时刻对V的偏导,带入【公式(12)、(15)】,通过这么大费周章的计算,最终的t时刻的导数就是如下公式,多么的简单。

$$\frac{\partial L_t}{\partial V} = \frac{\partial L_t}{\partial z_t} \frac{\partial z_t}{\partial V} = (y'_{t,i} - y_{t,i}) S_t$$

最终的损失,把各个时刻的相加则可得到。整个循环一遍,会改变参数,并不是每个时刻更新。

$$\frac{\partial L}{\partial V} = \sum_t \frac{\partial L_t}{\partial V}$$

RNN

对U的导数:

对U的导数和对V的导数相似,

$$\begin{aligned}\frac{\partial L}{\partial U} &= \sum_t \frac{\partial L_t}{\partial U} \\ \frac{\partial L_t}{\partial U} &= \frac{\partial L_t}{\partial z_t} \frac{\partial z_t}{\partial S_t} \frac{\partial S_t}{\partial U}\end{aligned}$$

由V得到如下值:

$$\begin{aligned}\frac{\partial L_t}{\partial z_t} &= (y'_{t,i} - y_{t,i}) \\ \frac{\partial z_t}{\partial S_t} &= V \\ \frac{\partial S_t}{\partial U} &= \tanh' X_t\end{aligned}$$

所以

$$\frac{\partial L_t}{\partial U} = (y'_{t,i} - y_{t,i}) V \tanh' X_t$$

RNN

对W的导数:

通过导数, 能够发现, v 和 u 的梯度并不会变得很小, 所以真正会导致梯度消失的只有 W , 因为 W 的梯度有连乘项 $\frac{\partial S_t}{\partial S_{t-1}}$ 。所以当两个词距离较远, 互相影响就很微弱。

对W的导数会有依赖项, 故而需要求解依赖项。

$$\begin{aligned}\frac{\partial L}{\partial W} &= \sum_t \frac{\partial L_t}{\partial W} \\ \frac{\partial L_t}{\partial W} &= \frac{\partial L_t}{\partial z_t} \frac{\partial z_t}{\partial S_t} \frac{\partial S_t}{\partial W}\end{aligned}$$

由V得到如下值:

$$\begin{aligned}\frac{\partial L_t}{\partial z_t} &= (y'_{t,i} - y_{t,i}) \\ \frac{\partial z_t}{\partial S_t} &= V \\ \frac{\partial S_t}{\partial W} &= \frac{\partial S_t}{\partial W} + \frac{\partial S_t}{\partial S_{t-1}} \frac{\partial S_{t-1}}{\partial W} + \frac{\partial S_t}{\partial S_{t-1}} \frac{\partial S_{t-1}}{\partial S_{t-2}} \frac{\partial S_{t-2}}{\partial W} \dots\end{aligned}$$

最终对W的导数:

$$\begin{aligned}\frac{\partial S_t}{\partial W} &= \sum_k^T \prod_{t=k+1}^T \frac{\partial S_t}{\partial S_{t-1}} \frac{\partial S_k}{\partial S_W} \\ \frac{\partial L_t}{\partial W} &= \frac{\partial L_t}{\partial z_t} \frac{\partial z_t}{\partial S_t} \frac{\partial S_t}{\partial W} = \frac{\partial L_t}{\partial z_t} \frac{\partial z_t}{\partial S_t} \sum_k^T \prod_{t=k+1}^T \frac{\partial S_t}{\partial S_{t-1}} \frac{\partial S_k}{\partial S_W}\end{aligned}$$

LSTM (Long Short-Term Memory)

梯度消失和梯度爆炸

长短期记忆 (Long short-term memory, LSTM) 是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。

RNN只是将相邻的两个样本关联, LSTM能够在更长的序列中有更好的表现

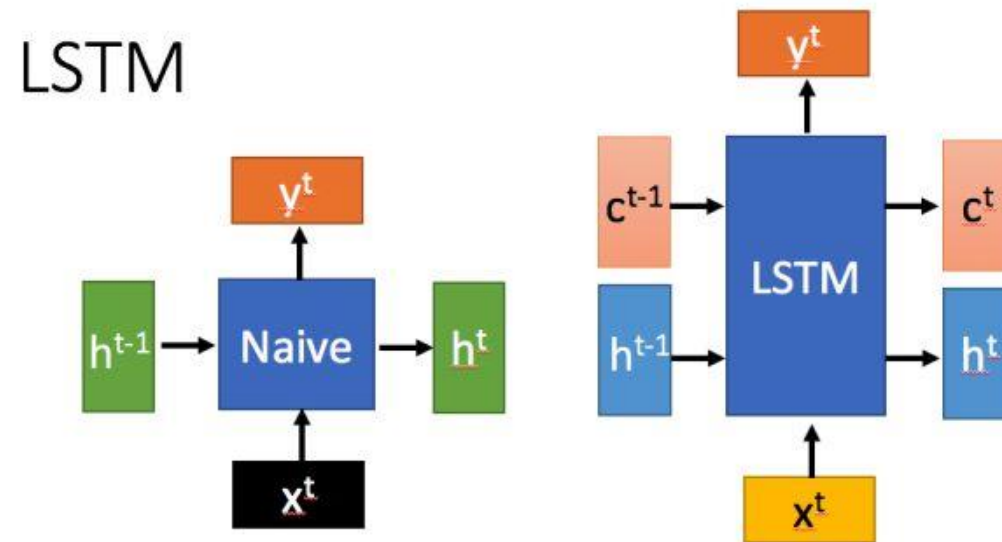
“the clouds are in the sky” 预测sky比较容易

“I grew up in France... I speak fluent French.” 预测French比较难

理论上RNNs是能够处理这种“长依赖”问题的。通过调参来解决这种问题。但是在实践过程中RNN无法学习到这种特征

LSTM

长短期记忆（Long short-term memory, LSTM）的结构
相比于原始的RNN的隐层(hidden state), LSTM增加了一个细胞状态(cell state)

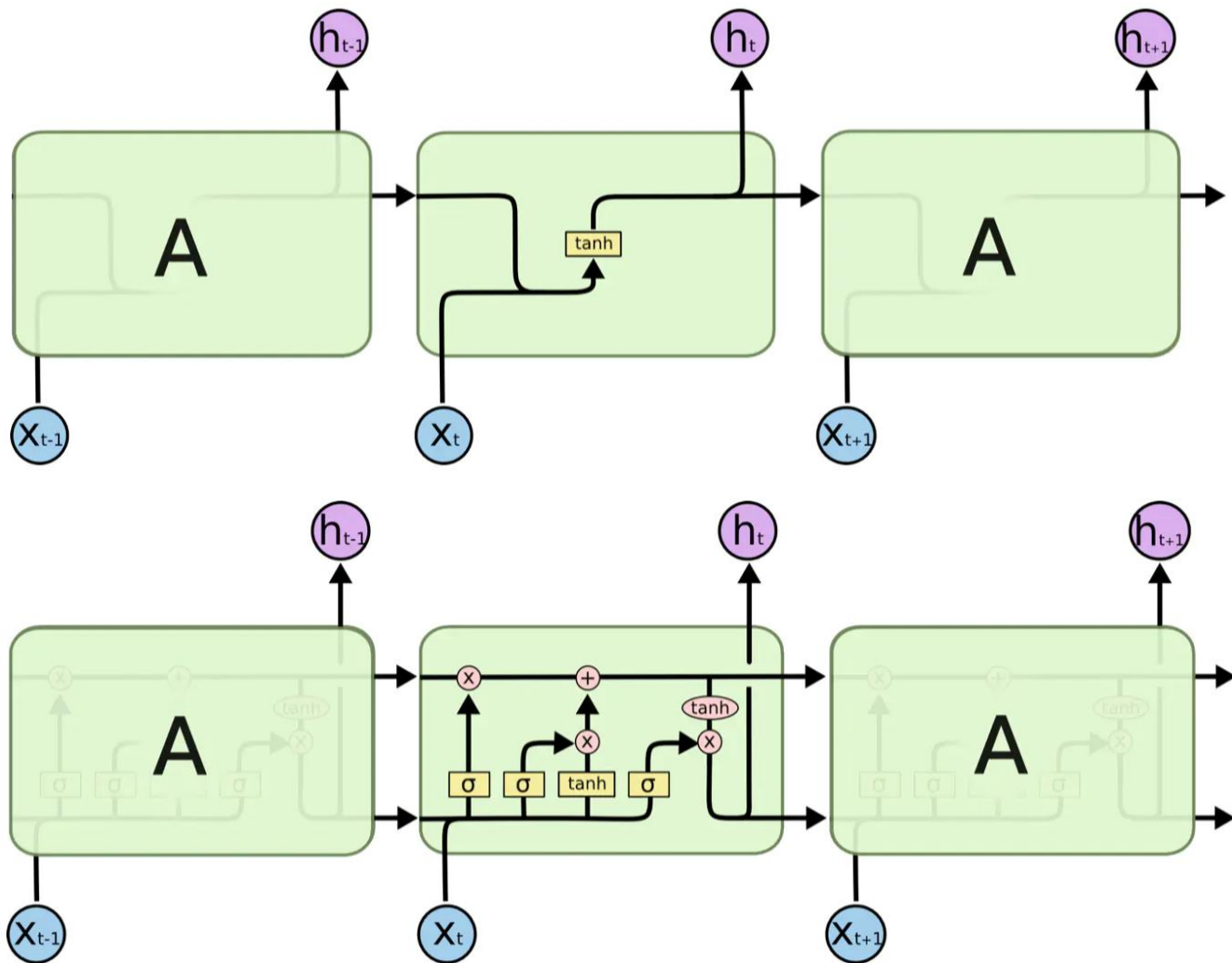


c change slowly ➡ c^t is c^{t-1} added by something

h change faster ➡ h^t and h^{t-1} can be very different

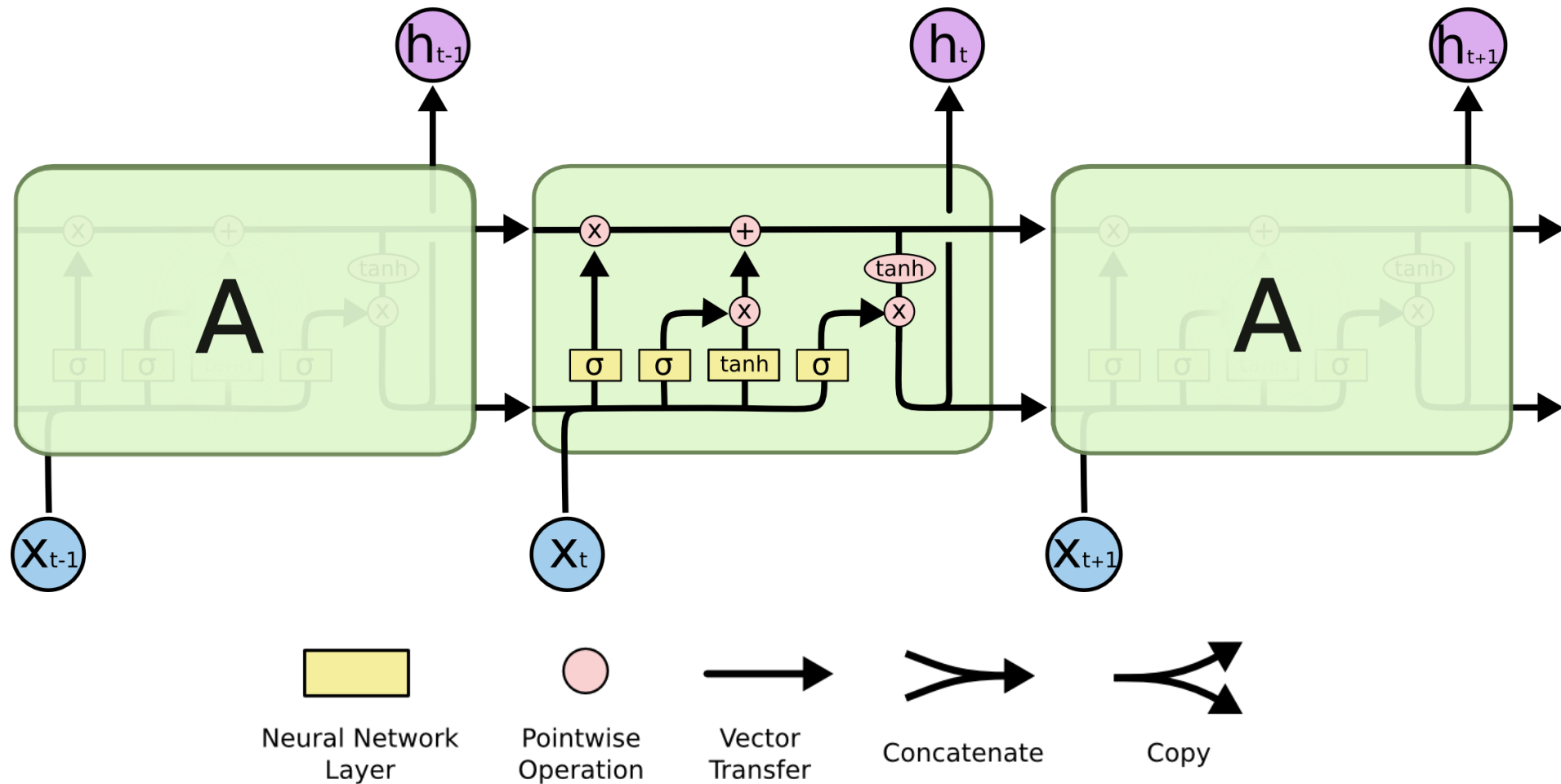
LSTM

长短期记忆（Long short-term memory, LSTM）的结构与RNN结构对比



LSTM

长短期记忆 (Long short-term memory, LSTM)



LSTM

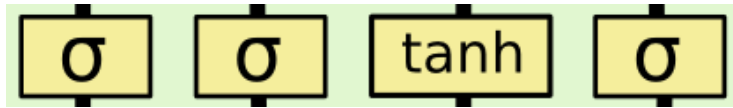
长短期记忆 (Long short-term memory, LSTM)

输入：细胞状态 C_{t-1} ，隐层状态 h_{t-1} ， t 时刻输入向量 X_t

输出：细胞状态 C_t ，隐层状态 h_t 。其中 h_t 还作为 t 时刻的输出。

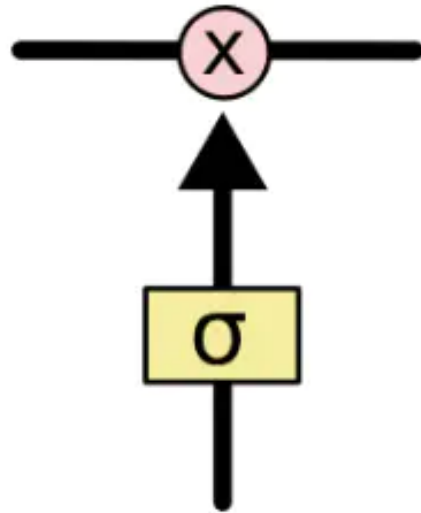
C_0 与 h_0 的值，也就是两个隐层的初始值，一般是用全0初始化

$\text{Shape}(C_t) = \text{Shape}(h_t) = \text{HiddenSize}$



LSTM

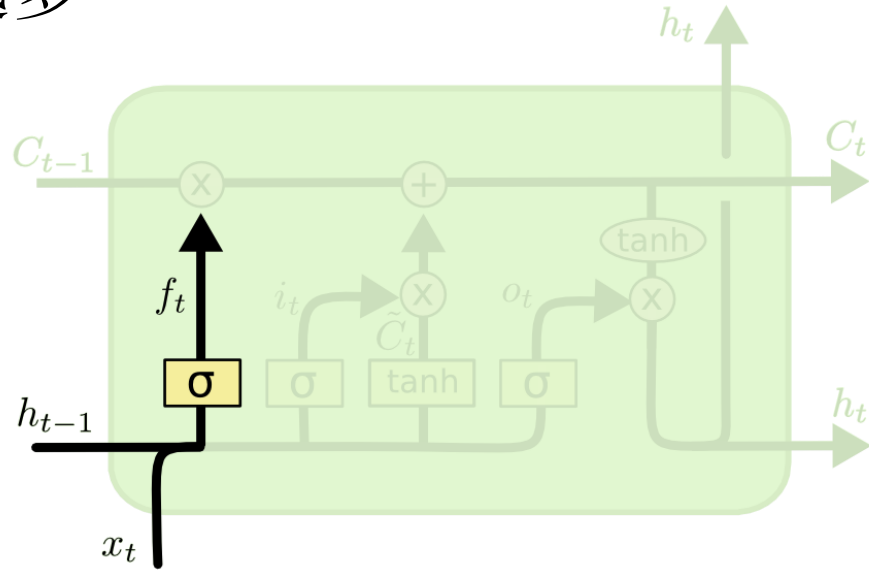
LSTM网络能通过一种被称为门的结构对细胞状态进行删除或者添加信息，门能够有选择性的决定让哪些信息通过，就是一个sigmoid层和一个点乘操作的组合



LSTM

LSTM由三个门来控制细胞状态，这三个门分别称为遗忘门、输入门和输出门。

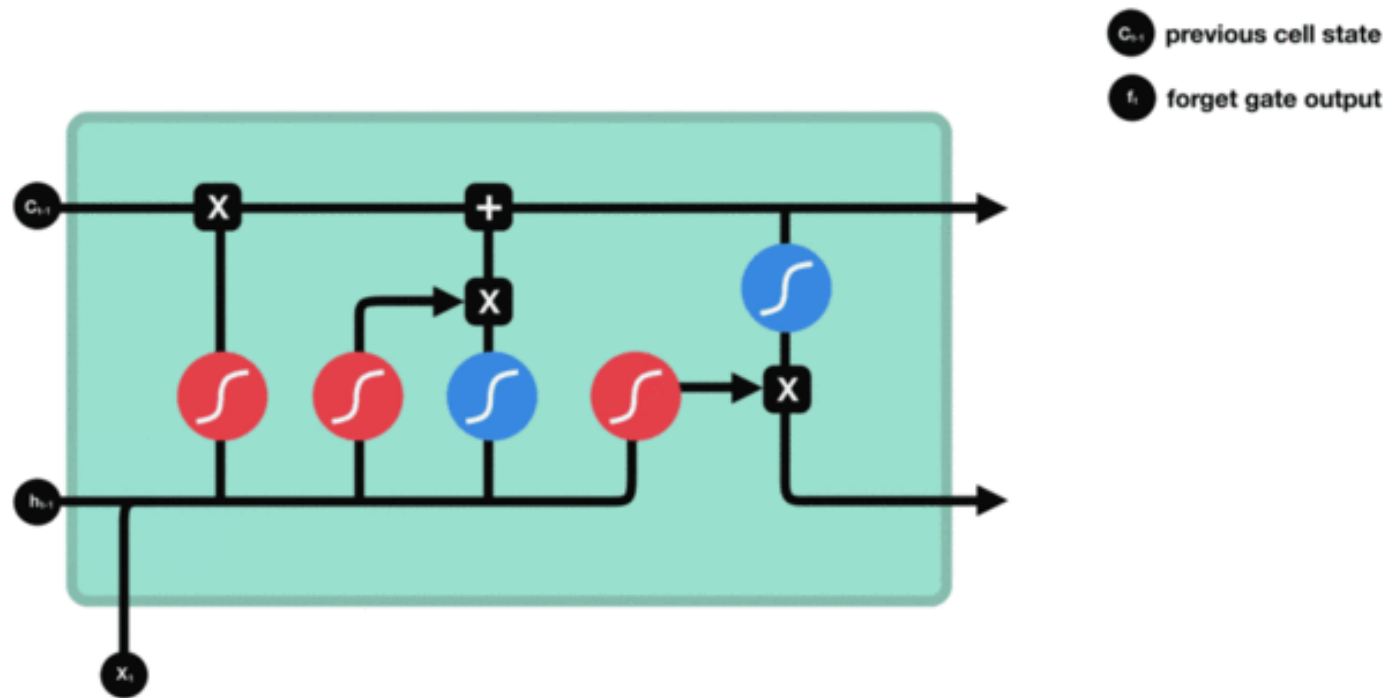
遗忘门： σ 是sigmoid函数，输出在0到1之间，越是靠近0则表示需要删除的越多



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

LSTM

遗忘门

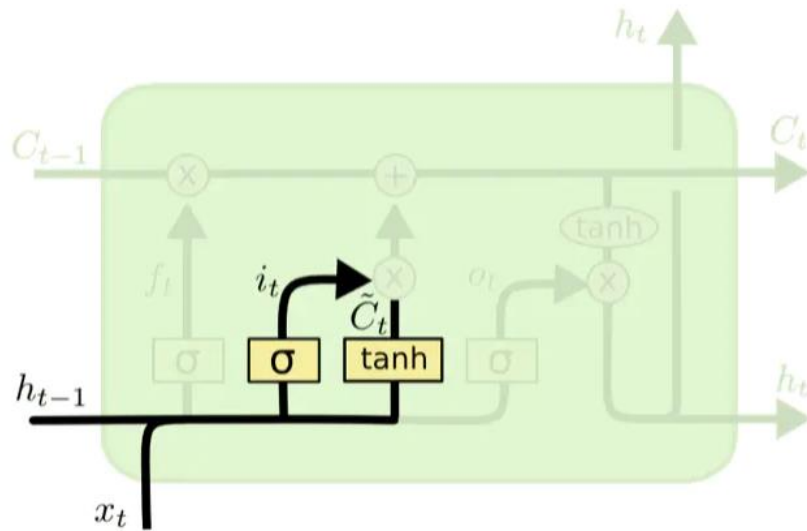


LSTM

输入门：分为两个步骤，

第一，利用 h_{t-1} 和 x_t 决定更新哪些信息，。

第二，然后利用 h_{t-1} 和 x_t 来通过 \tanh 函数得到新的候选细胞信息 \tilde{C}_t ，这些信息可能会被更新到细胞信息中



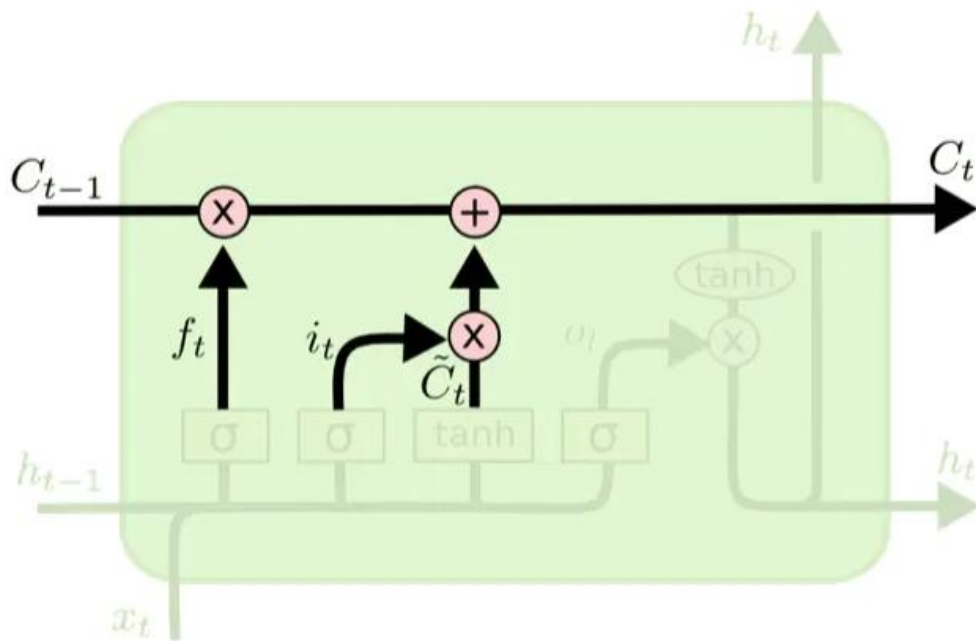
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

LSTM

输入门:

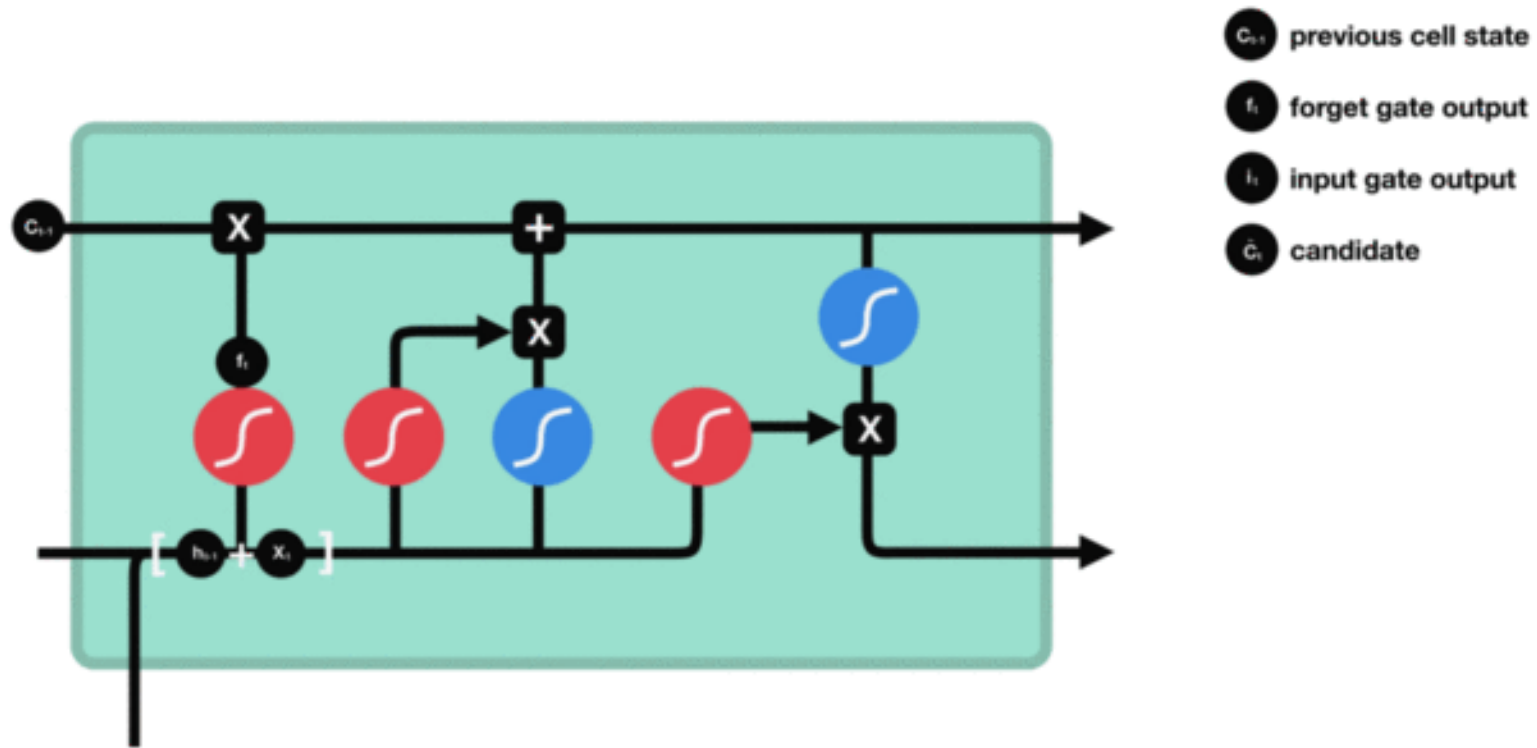
更新 C_t 的值，更新的方法是用遗忘门将旧的细胞中的信息删掉，并通过输入门中的候选细胞 \tilde{C}_t 的一部分，得到新的 C_t



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM

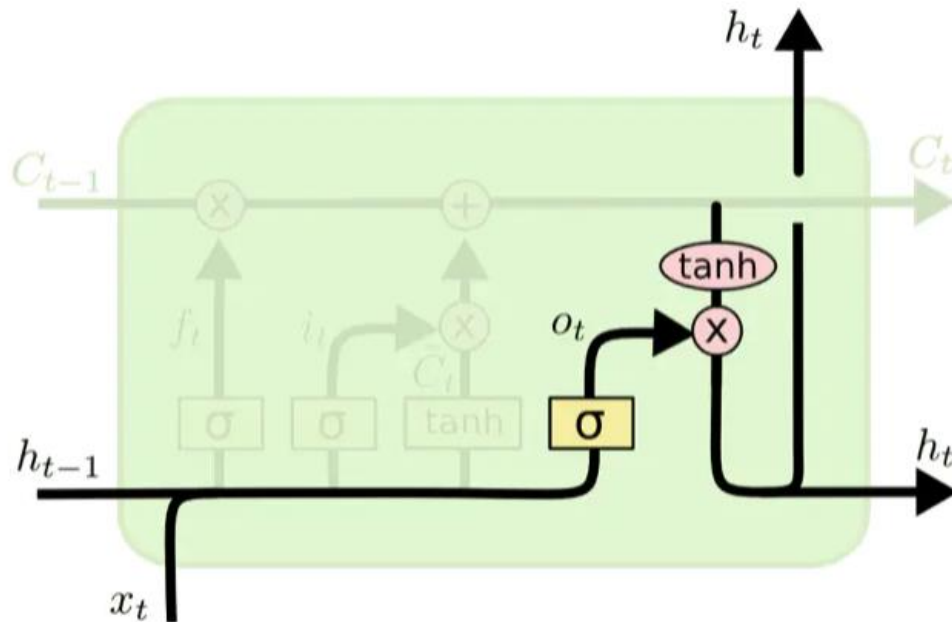
输入门



LSTM

输出门:

根据 h_{t-1} 和 x_t 决定输出哪些信息，需要将输入经过一个称为输出门的sigmoid层得到判断条件，然后将细胞状态经过tanh层得到一个-1~1之间值的向量，该向量与输出门得到的判断条件相乘就得到了最终该RNN单元的输出

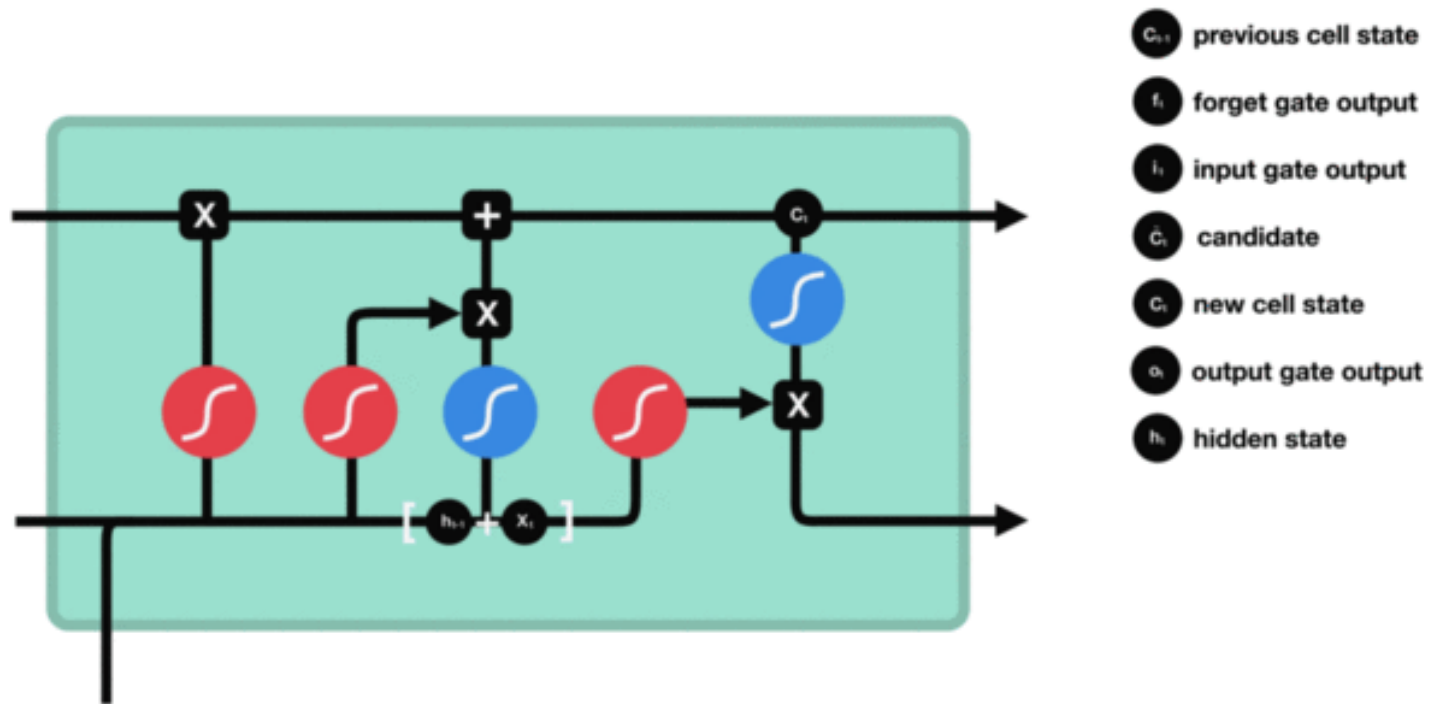


$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

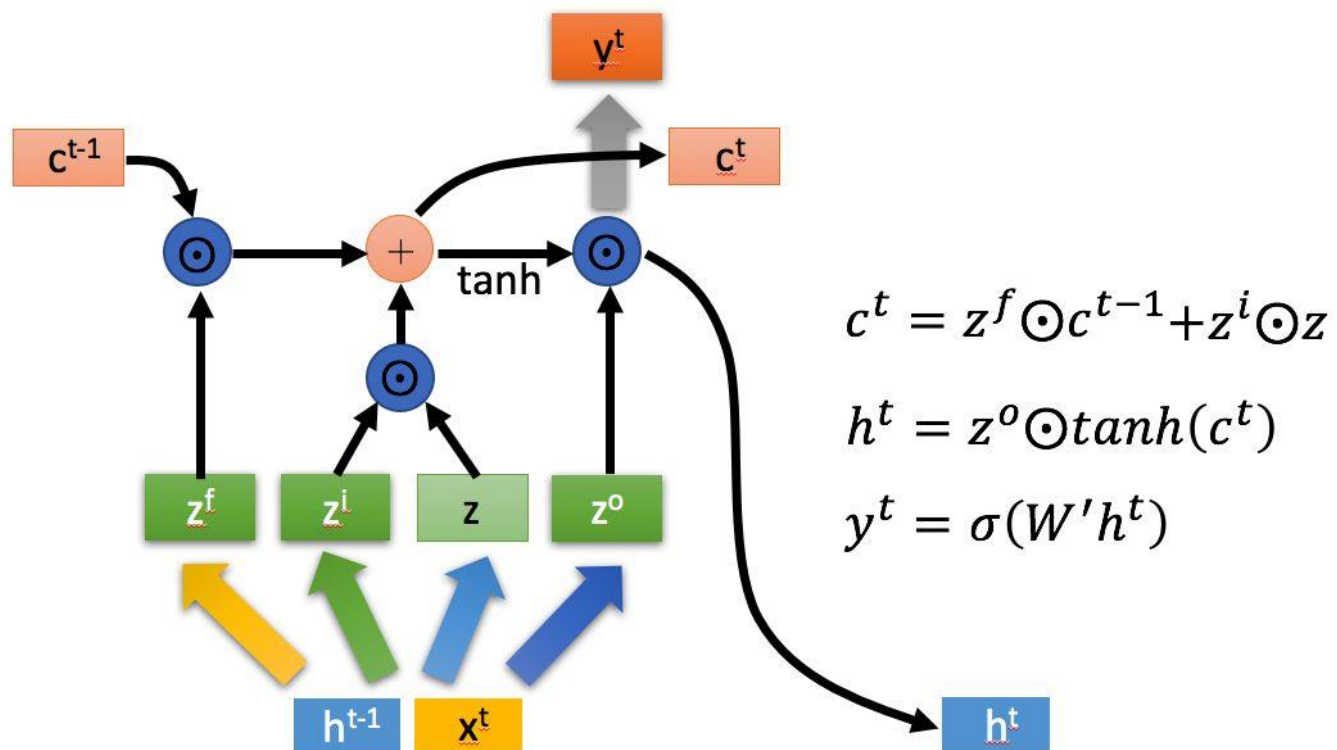
LSTM

输出门



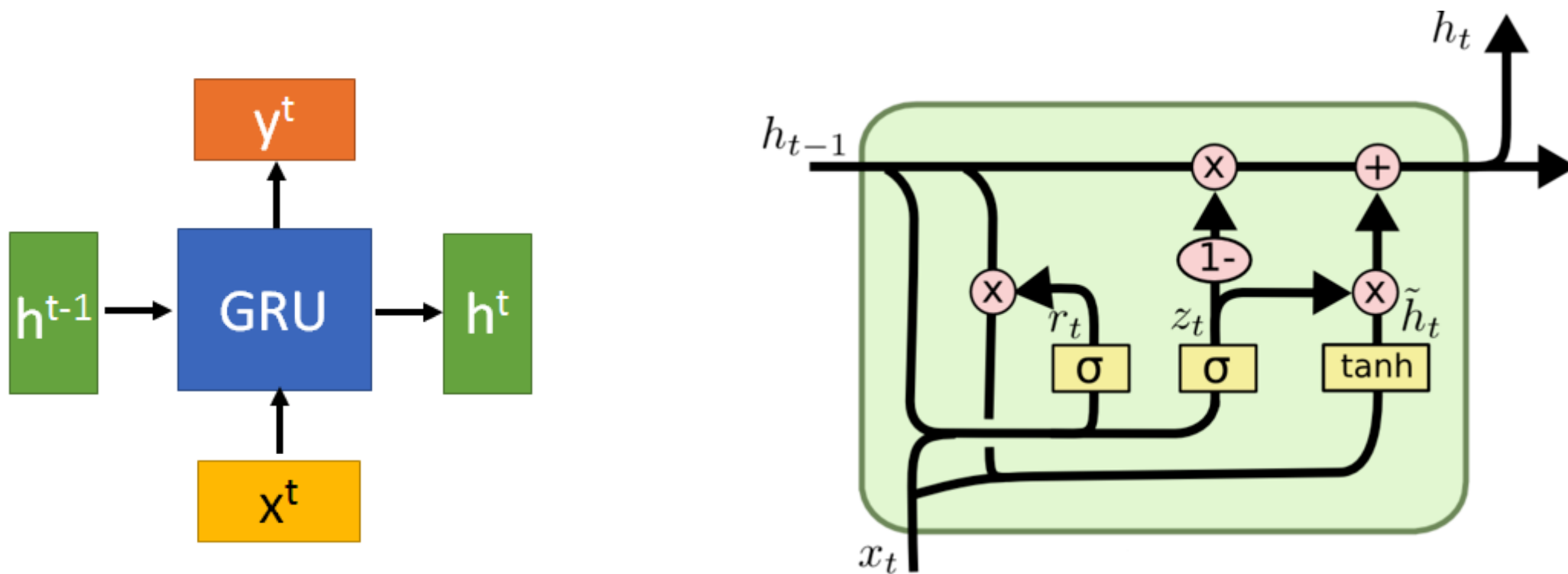
LSTM

LSTM的整体结构



GRU (Gate Recurrent Unit) 门控循环单元

与LSTM类似，是一个简单变体，只有两个门，重置门和一个更新门。可处理序列数据的一种模型，是循环神经网络的一种，LSTM有很多可以精简改进的地方，因此GRU就诞生了。相比较LSTM内部结构进行了简化，同时准确率也得到了提升。

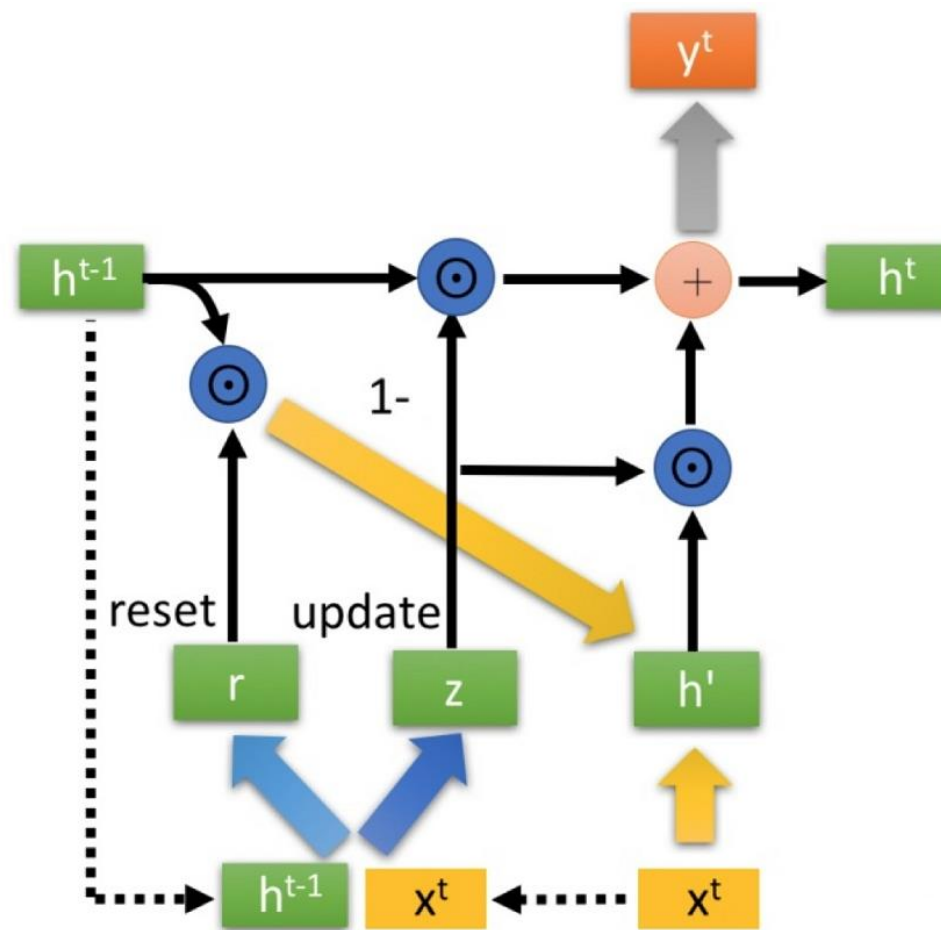


GRU (Gate Recurrent Unit)

GRU的结构解析

通过上层传下来的 h^{t-1} ，和当前节点的输入 x^t 来计算门控状态 r ， z ， σ 为sigmoid，取值0-1之间

$$r = \sigma(W^r \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$
$$z = \sigma(W^z \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix})$$

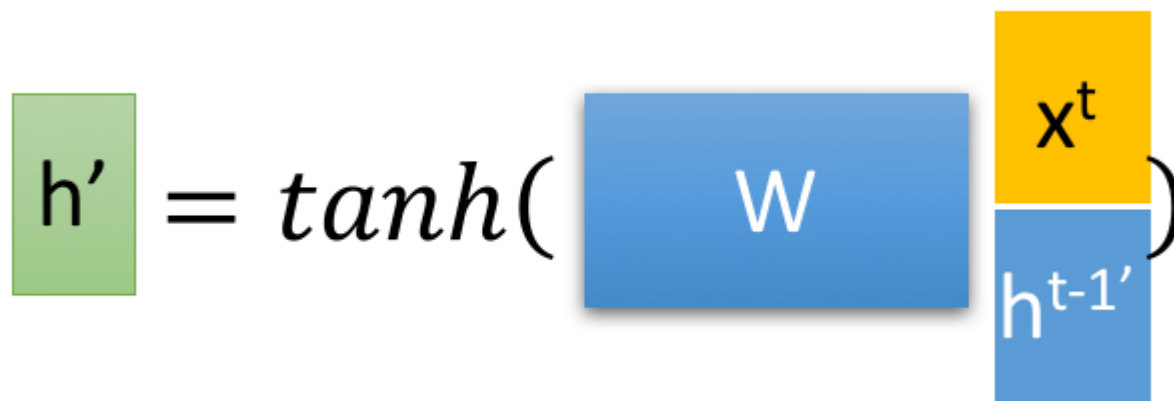


GRU (Gate Recurrent Unit)

GRU的结构解析

重置门：使用重置门控 r 得到重置之后的数据 $h^{t-1'}$ ， $h^{t-1'} = h^{t-1} \odot r$ 节点，再将 $h^{t-1'}$ 与输入 x 进行拼接，然后通过 \tanh 激活函数来将数据缩放到-1到1之间，得到 h' 。（ \odot 是Hadamard Product，也就是操作矩阵中对应的元素相乘，因此要求两个相乘矩阵是同型的）

这里的 h' 主要是包含了当前输入的 x^t 数据。有针对性地对 h' 添加到当前的隐藏状态，相当于”记忆了当前时刻的状态“。类似于LSTM的选择记忆阶段。



The diagram illustrates the calculation of the reset gate output h' . On the left, a green box contains the symbol h' . This is followed by an equals sign and the word \tanh in an italicized font. To the right of \tanh is a large blue box containing the letter W . To the right of the W box is a vertical stack of two boxes: a yellow box on top containing x^t and a blue box on the bottom containing $h^{t-1'}$. A closing parenthesis $)$ is located to the right of the stacked boxes.

$$h' = \tanh(W \begin{bmatrix} x^t \\ h^{t-1'} \end{bmatrix})$$

GRU (Gate Recurrent Unit)

GRU的结构解析

更新门：用来更新记忆。

首先使用更新门控对 h^t 进行更新，

$$h^t = (1-z) \odot h^{t-1} + z \odot h'$$

z 的值是0-1之间，越是接近1，越是能被保留下来，接近0则遗忘的更多，通过该门控 z 可以同时进行选择和遗忘。

$(1-z) \odot h^{t-1}$ ：表示对原本隐藏状态的选择性“遗忘”，这里的 $1-z$ 可以想象成遗忘门（forget gate），忘记 h^{t-1} 维度中一些不重要的信息

$z \odot h'$ ：表示对当前节点信息 h' 进行选择性的“记忆”，这里的 $1-z$ 可以想象成遗忘门（forget gate），同理会忘记 h' 维度中的一些不重要的信息。或者，这里我们更应当看做是对 h' 维度中的某些信息进行选择。

GRU

与LSTM的区别

1. LSTM有三个门，而GRU有两个门，利用更新门合并了LSTM的遗忘门和输入门
2. 去掉了细胞单元C
3. 输出的时候取消了二阶的非线性函数

重置门：

作用对象是前边的隐藏状态，作用是决定了有多少过去信息需要遗忘

更新门：（可以理解为LSTM中的遗忘门和输入门相结合）

作用对象是当前时刻和上一时刻的隐藏单元，作用是上一时刻，以及当前时刻总共多少有用的信息需要接着向下传递

GRU

GRU与LSTM的对比

GRU是在2014年提出来的，而LSTM是1997年。他们的提出都是为了解决相似的问题，那么GRU难免会参考LSTM的内部结构

都通过了门控去保留重要的特征

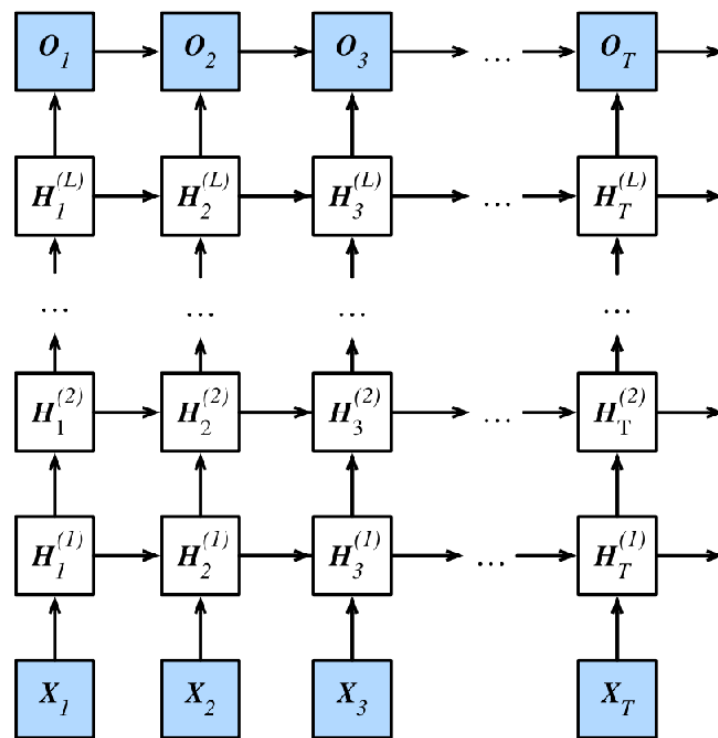
准确率不相上下

但是GRU更快因为它拥有更少的参数，少了一个门就少了很多的矩阵的乘法，在数据上训练的时候GRU就能节省下来很多的时间。

深度循环神经网络

如果将深度定义为网络中信息传递路径长度的话，循环神经网络可以看作既“深”又“浅”的网络。一方面来说，如果我们把循环网络按时间展开，长时间间隔的状态之间的路径很长，循环网络可以看作一个非常深的网络。从另一方面来说，如果同一时刻网络输入到输出之间的路径 $\mathbf{x}_t \rightarrow \mathbf{y}_t$ ，这个网络是非常浅的。

L个隐藏层的堆叠循环神经网络



$$\mathbf{H}_t = f(\mathbf{H}_{t-1}, \mathbf{X}_t)$$

$$\mathbf{O}_t = g(\mathbf{H}_t)$$

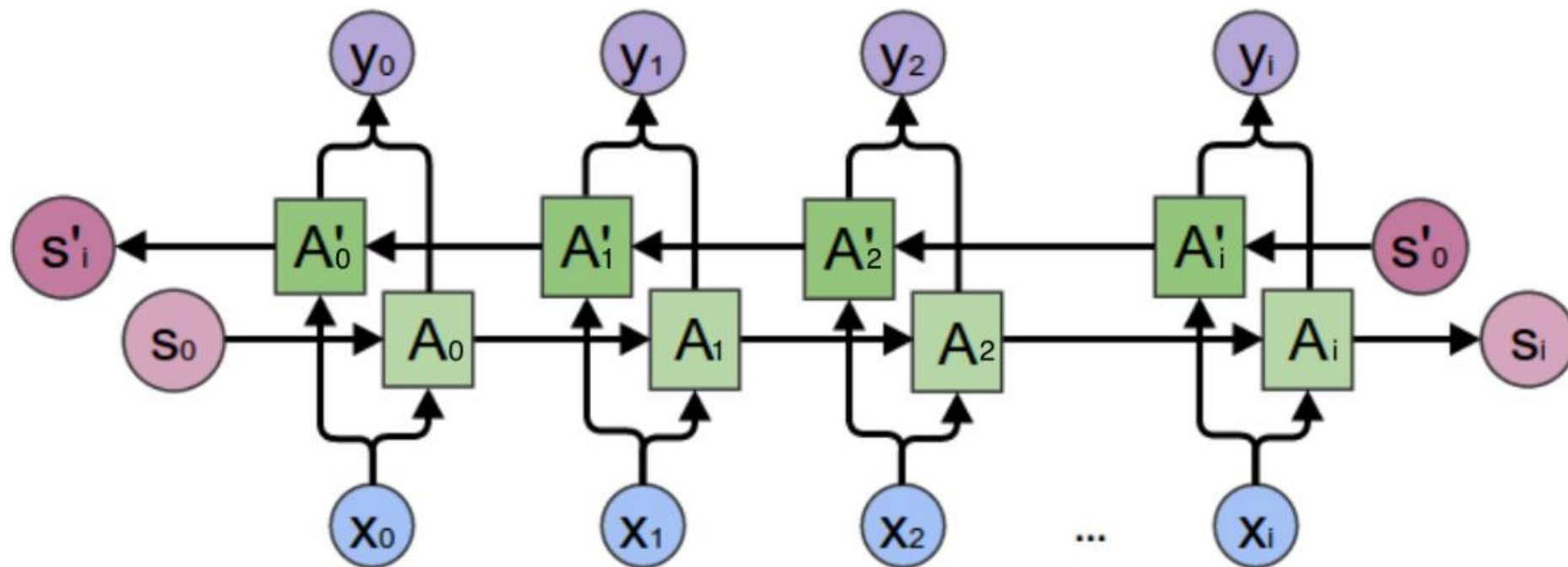
$$\mathbf{H}_t^1 = f_1(\mathbf{H}_{t-1}^1, \mathbf{X}_t)$$

$$\mathbf{H}_t^j = f_j(\mathbf{H}_{t-1}^j, \mathbf{H}_t^{j-1})$$

$$\mathbf{O}_t = g(\mathbf{H}_t^L)$$

双向循环神经网络

即可以从过去的时间点获取记忆，又可以从未来的时间点获取信息
至于网络单元到底是标准的RNN还是GRU或者是LSTM是没有关系的，都可以使用



循环神经网络如何使用

影评的情感分析

