人工智能之深度学习

循环神经网络RNN

主讲人：刘老师(GerryLiu)

课程要求

- 课上课下“九字”真言
  - 认真听，**善摘录，勤思考**
  - **多温故，乐实践**，再发散
- 四不原则
  - 不懒散惰性，不迟到早退
  - 不请假旷课，不拖延作业
- 一点注意事项
  - 违反“四不原则”，不推荐就业

课程内容

- 递归神经网络(RNN)
  - 什么是递归神经网络
  - 应用场景
  - 层次结构
  - RNN描述
- LSTM
- GRU

什么是递归神经网络

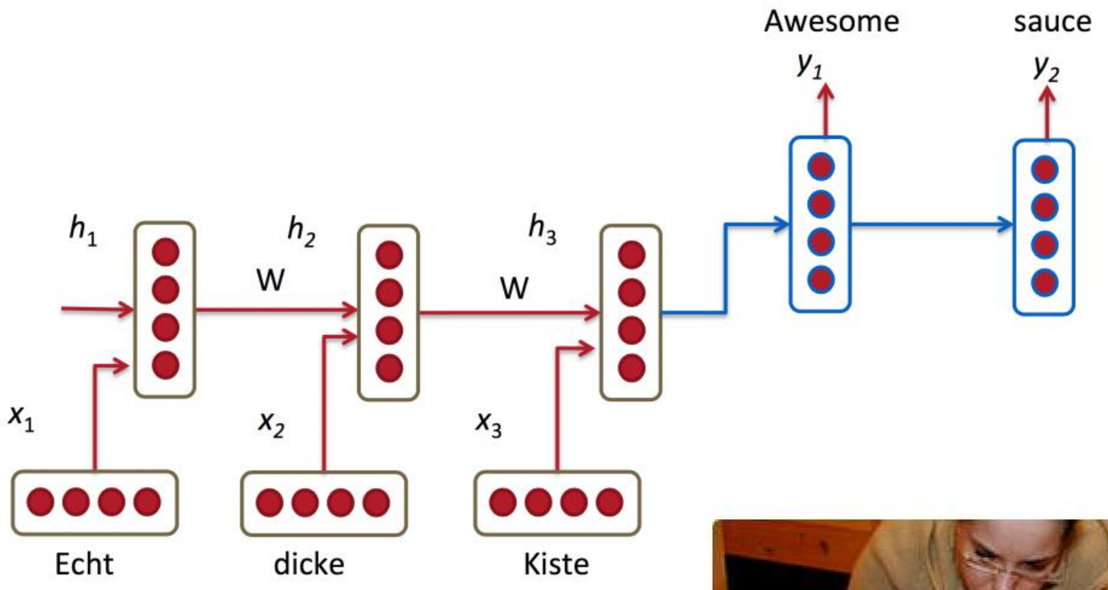
- 为什么有BP神经网络、CNN，还需要RNN?
  - BP神经网络和CNN的输入输出都是互相独立的；但是实际应用中有些场景输出内容和之前的内容是有关联的。
  - RNN引入“记忆”的概念；递归指其每一个元素都执行相同的任务，但是输出依赖于输入和“记忆”。

什么是递归神经网络

- 我们已经学习了前馈网络的两种结构——BP神经网络和卷积神经网络，这两种结构有一个特点，就是假设输入是一个独立的没有上下文联系的单位，比如输入是一张图片，网络识别是狗还是猫。但是对于一些有明显的上下文特征的序列化输入，比如预测视频中下一帧的播放内容，那么很明显这样的输出必须依赖以前的输入， 也就是说网络必须拥有一定的”记忆能力”。为了赋予网络这样的记忆力，一种特殊结构的神经网络——递归神经网络(Recurrent Neural Network)便应运而生了。

递归神经网络RNN-应用场景

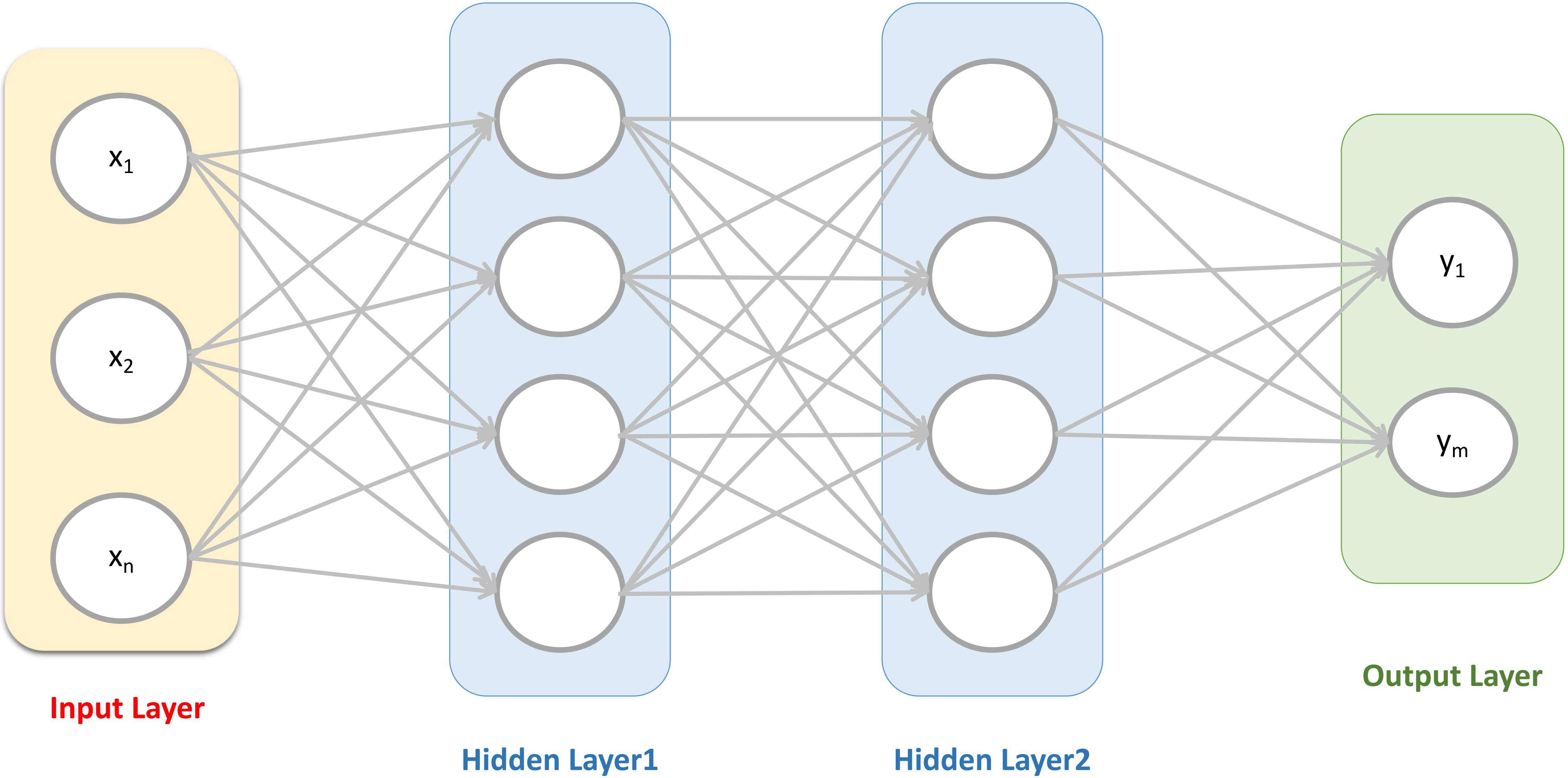
- 自然语言处理(NLP)
    - 语言模型与文本生成
- 机器翻译
- 语音识别
- 图像描述生成
- 文本相似度计算等



"man in black shirt is playing guitar."

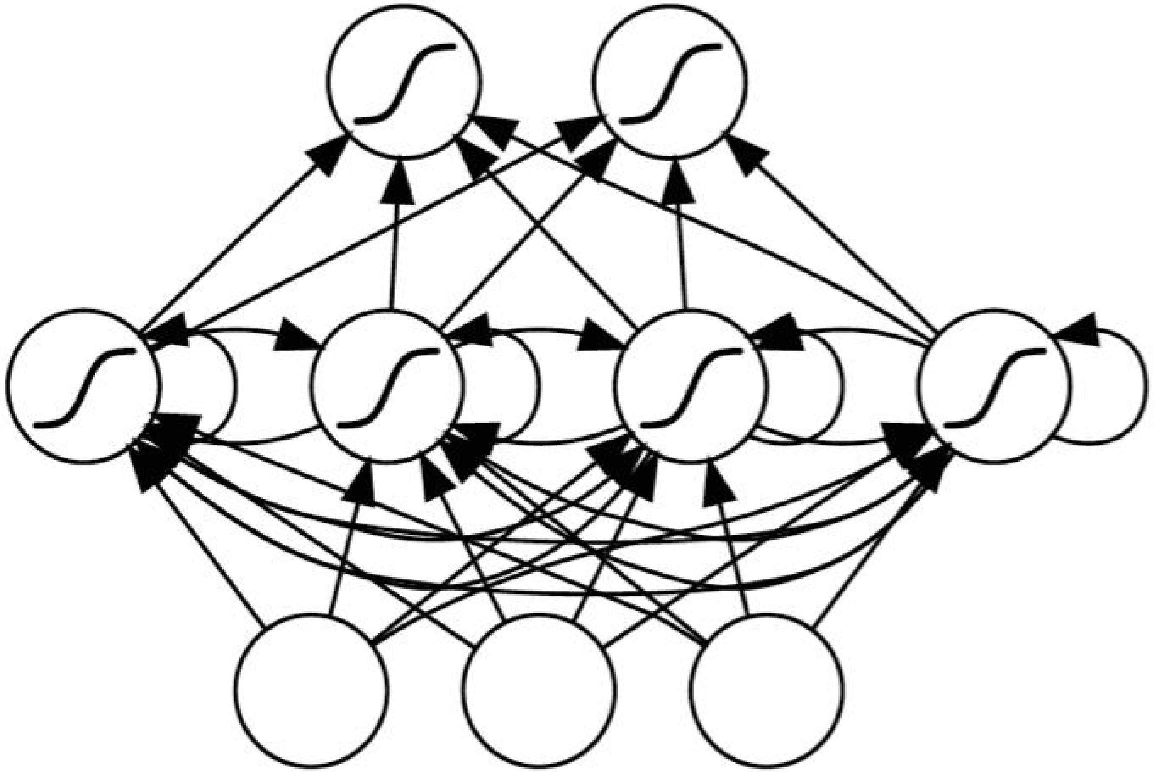
"two young girls are playing with lego toy."

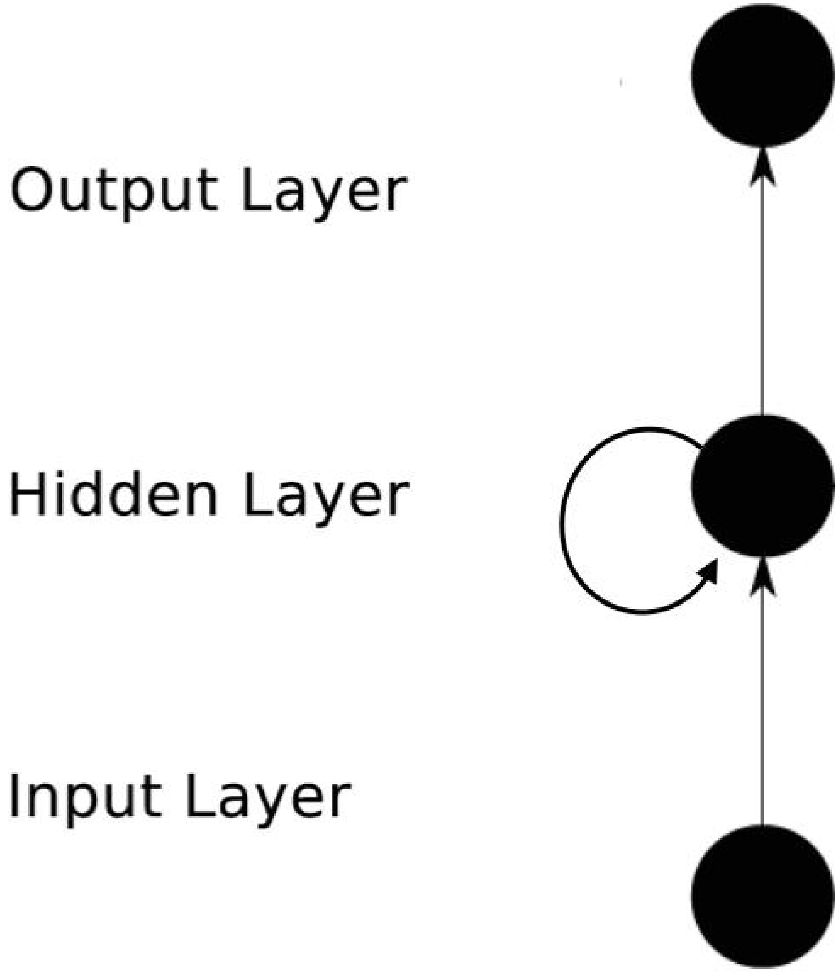
神经网络之结构



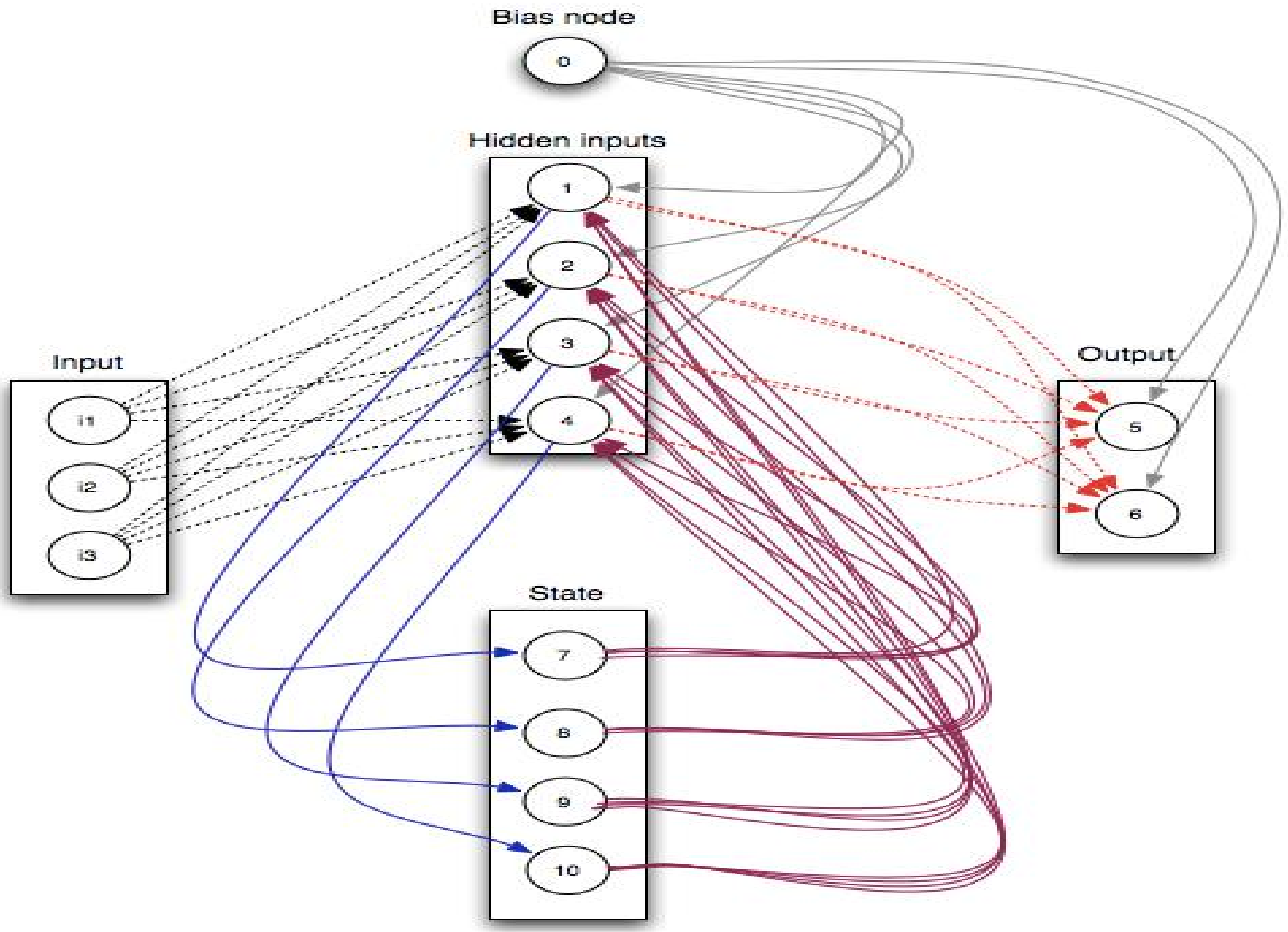
x1
x2
xn
y1
ym
Input Layer
Hidden Layer1
Hidden Layer2
Output Layer

递归神经网络RNN-结构





循环神经网络RNN-结构

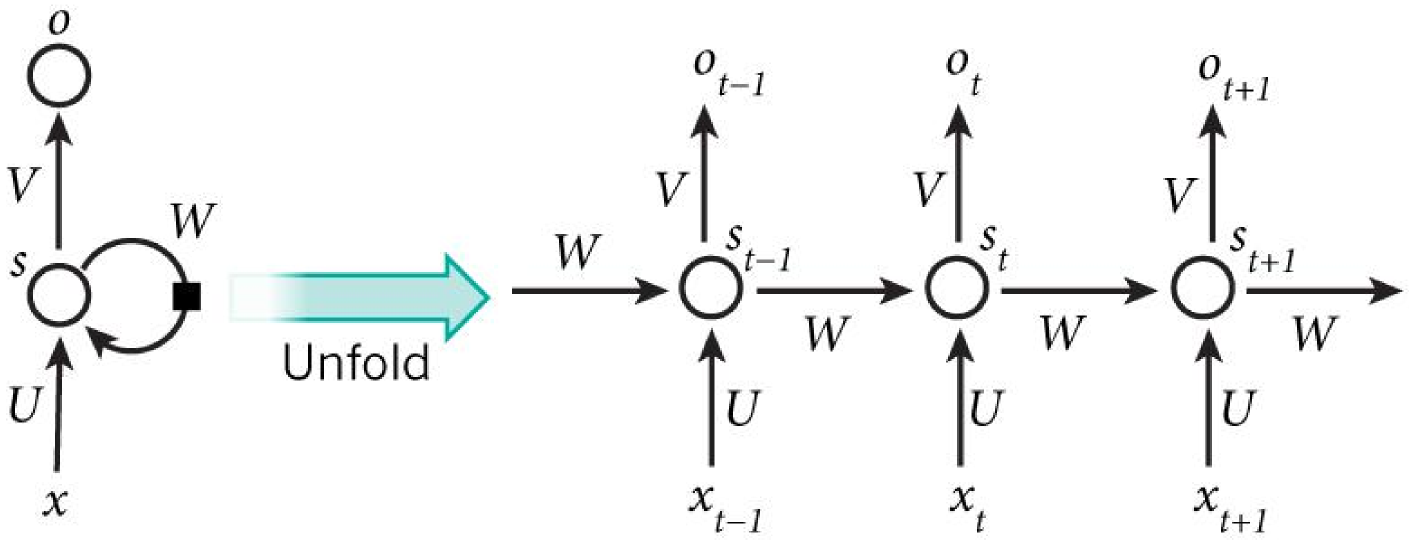


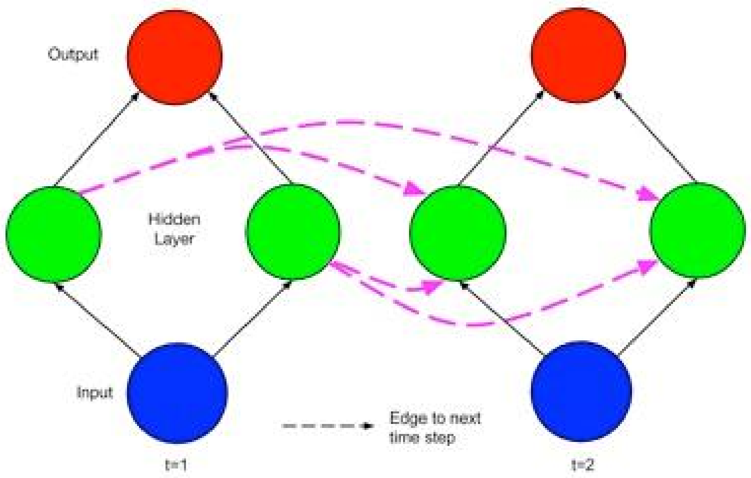
循环神经网络RNN-结构

- 网络某一时刻的输入x_t，和之前介绍的bp神经网络的输入一样，x_t是一个n维向量，不同的是递归网络的输入将是一整个序列，也就是$x=[x_1,...,x_{t-1},x_t,x_{t+1},...x_T]$，对于语言模型，每一个$x_t$将代表一个词向量，一整个序列就代表一句话。
- h_t代表时刻t的隐藏状态
- o_t代表时刻t的输出
- 输入层到隐藏层之间的权重由U表示，它将我们的原始输入进行抽象作为隐藏层的输入
- 隐藏层到隐藏层的权重W，它是网络的记忆控制者，负责调度记忆。
- 隐藏层到输出层的权重V，从隐藏层学习到的表示将通过它再一次抽象，并作为最终输出。

循环神经网络RNN-结构

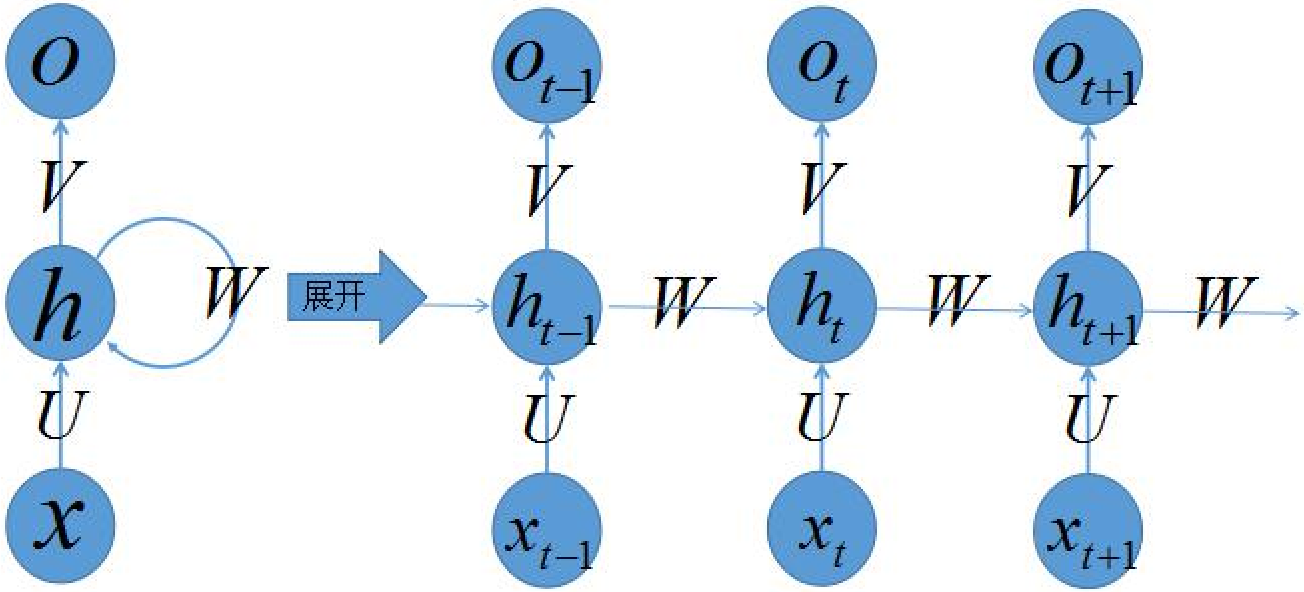
- 将序列按时间展开就可以得到RNN的结构
  - X_t是时间t处的输入
  - S_t是时间t处的“记忆”，$S_t=f(UX_t+WS_{t-1})$，f可以是非线性转换函数，比如tanh等
  - O_t是时间t处的输出，比如是预测下一个词的话，可能是sigmoid/softmax输出的属于每个候选词的概率，$O_t=softmax(VS_t)$





循环神经网络RNN-结构

按照一定的时间序列规定好计算顺序，于是实际上我们会将这样带环的结构展开成一个序列网络，也就是上图右侧被“unfold”之后的结构。



递归神经网络RNN正向传播阶段

- 在t=1的时刻，U,V,W都被随机初始化好，h0通常初始化为0，然后进行如下计算：

$$s_1 = Ux_1 + Wh_0$$
$$h_1 = f(s_1)$$
$$o_1 = g(Vh_1)$$

- 时间就向前推进，此时的状态h_1作为时刻1的记忆状态将参与下一次的预测活动，也就是：

$$s_2 = Ux_2 + Wh_1$$
$$h_2 = f(s_2)$$
$$o_2 = g(Vh_2)$$

递归神经网络RNN正向传播阶段

- 以此类推，可得

$$s_t = Ux_t + Wh_{t-1}$$
$$h_t = f(Ux_t + Wh_{t-1})$$
$$o_t = g(Vh_t)$$

- 其中f可以是tanh,relu,sigmoid等激活函数，g通常是softmax也可以是其他。
  值得注意的是，我们说递归神经网络拥有记忆能力，而这种能力就是通过W将以往的输入状态进行总结，而作为下次输入的辅助。可以这样理解隐藏状态：h=f(现有的输入+过去记忆总结)

递归神经网络RNN反向传播阶段

- bp神经网络用到的误差反向传播方法将输出层的误差总和，对各个权重的梯度∇U,∇V,∇W，求偏导数，然后利用梯度下降法更新各个权重。
- 对于每一时刻t的RNN网络，网络的输出o_t都会产生一定误差e_t，误差的损失函数，可以是交叉熵也可以是平方误差等等。那么总的误差为$E=\sum_t e_t$，我们的目标就是要求取

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

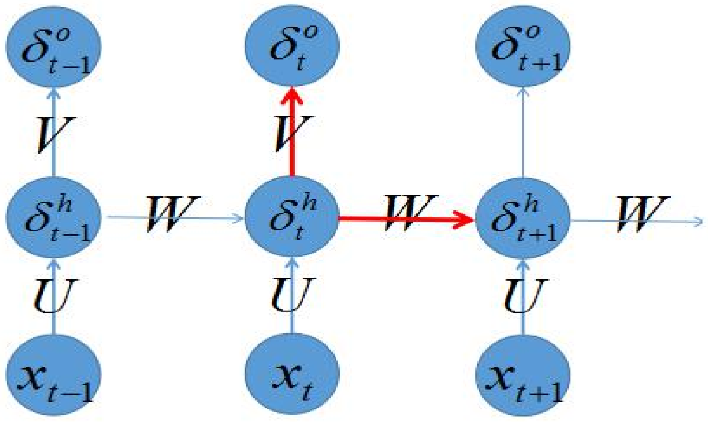
$$\nabla V = \frac{\partial E}{\partial V} = \sum_t \frac{\partial e_t}{\partial V}$$

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

递归神经网络RNN反向传播阶段

- 对于输出$o_t = g(Vs_t)$,对于任意损失函数，求取▽V将是简单的，我们可以直接求取每个时刻的$\partial e_t / \partial V$，由于它不存在和之前的状态依赖，可以直接求导取得，然后简单地求和即可。对于▽W,▽U的计算不能直接求导，因此需要用链式求导法则。

  为了使得误差e能够对U和W求偏导数，定义一个中$\delta = \partial e / \partial s$，首先计算出输出层的$\delta^L$，再向后传播到各层$\delta^{L-1}, \delta^{L-2}, \ldots$，那么如何计算δ呢?



递归神经网络RNN反向传播阶段

- 关注当前层次发射出去的链接即可，也就是

$$\delta_t^h = (V^T \delta_t^o + W^T \delta_{t+1}^h) .* f'(s_t)$$

只要计算出所有的δ^o_t,δ^h_t，就可以通过以下计算出∇W,∇U:

递归神经网络RNN反向传播阶段

- 关注当前层次发射出去的链接即可，也就是

$$\delta_t^h = (V^T \delta_t^o + W^T \delta_{t+1}^h) .* f'(s_t)$$

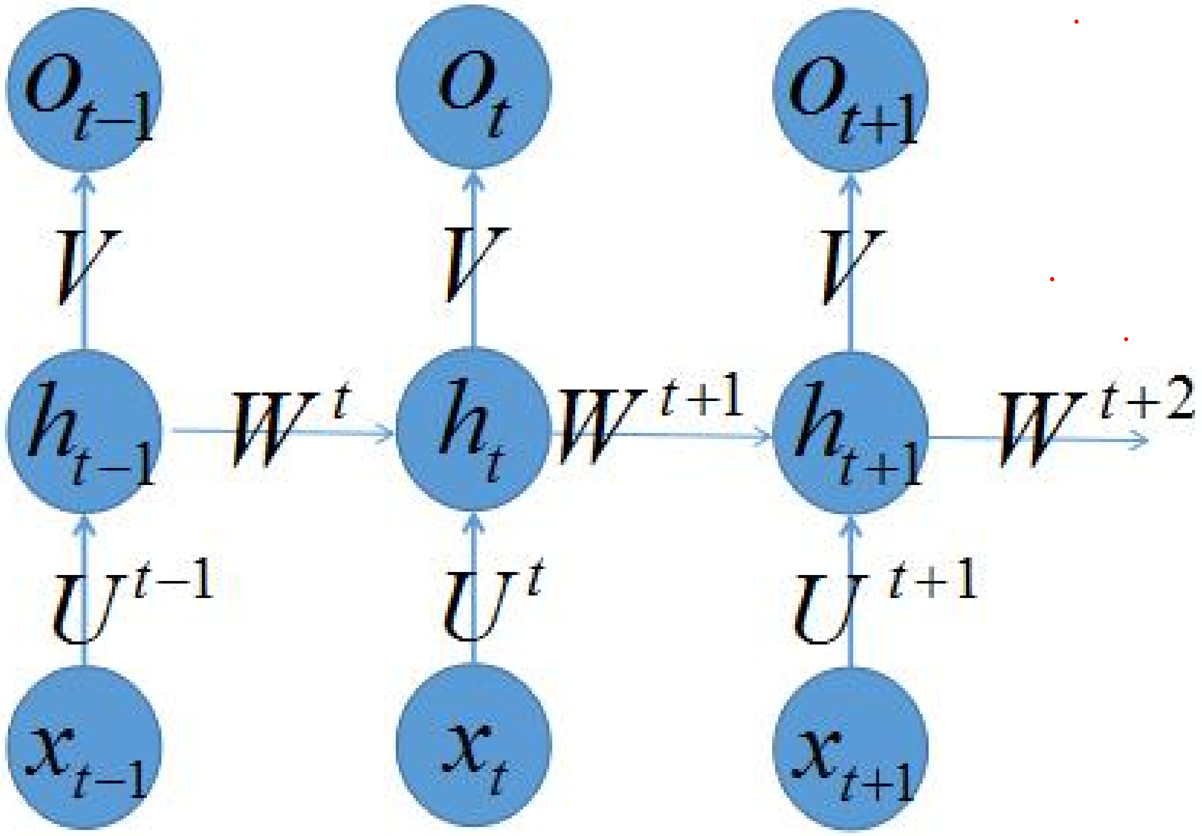
只要计算出所有的δ^o_t,δ^h_t，就可以通过以下计算出∇W,∇U:

$$\nabla W = \sum_t \begin{bmatrix} \delta_{0,t}^h h_{0,t-1}, \ldots, \delta_{0,t}^h h_{i,t-1}, \ldots, \delta_{0,t}^h h_{m,t-1} \\ \cdots \\ \delta_{j,t}^h h_{0,t-1}, \ldots, \delta_{j,t}^h h_{i,t-1}, \ldots, \delta_{j,t}^h h_{m,t-1} \\ \cdots \\ \delta_{n,t}^h h_{0,t-1}, \ldots, \delta_{n,t}^h h_{i,t-1}, \ldots, \delta_{n,t}^h h_{m,t-1} \end{bmatrix} = \sum_t \delta_t^h \times h_{t-1}$$

$$\nabla U = \sum_t \begin{bmatrix} \delta_{0,t}^h x_{0,t}, \ldots, \delta_{0,t}^h x_{i,t}, \ldots, \delta_{0,t}^h x_{m,t} \\ \cdots \\ \delta_{j,t}^h x_{0,t}, \ldots, \delta_{j,t}^h x_{i,t}, \ldots, \delta_{j,t}^h x_{m,t} \\ \cdots \\ \delta_{n,t}^h x_{0,t}, \ldots, \delta_{n,t}^h x_{i,t}, \ldots, \delta_{n,t}^h x_{m,t} \end{bmatrix} = \sum_t \delta_t^h \times x_t$$

递归神经网络RNN反向传播阶段

- 举个详细的例子计算W梯度的例子：



递归神经网络RNN反向传播阶段

- 举个对于时刻t+1产生的误差e_{t+1}，我们想计算它对于$W^1, W^2, ..., W^t$，W^{t+1}的梯度，可以如下计算：

$$\frac{\partial e_{t+1}}{\partial W^{t+1}} = \frac{\partial e_{t+1}}{\partial h^{t+1}} \frac{\partial h_{t+1}}{\partial W^{t+1}}$$

$$\frac{\partial e_{t+1}}{\partial W^{t}} = \frac{\partial e_{t+1}}{\partial h^{t+1}} \frac{\partial h_{t+1}}{\partial h^{t}} \frac{\partial h_{t}}{\partial W^{t}}$$

$$\frac{\partial e_{t+1}}{\partial W^{t-1}} = \frac{\partial e_{t+1}}{\partial h^{t+1}} \frac{\partial h_{t+1}}{\partial h^{t}} \frac{\partial h_{t}}{\partial h^{t-1}} \frac{\partial h_{t-1}}{\partial W^{t-1}}$$

$$\cdots\cdots$$

- 反复运用链式法则，我们可以求出每一个$\nabla W_1, \nabla W_2, ..., \nabla W_t$，$\nabla W_{t+1}$，在不同时刻都是共享同样的参数，这样可以大大减少训练参数，和CNN的共享权重类似。对于共享参数的RNN，我们只需将上述的一系列式子抹去标签并求和，就可以得到：

递归神经网络RNN反向传播阶段

- 推导出来的公式为：

$$\frac{\partial e_t}{\partial W} = \sum_{1\le k\le t} \frac{\partial e_t}{\partial h^t} \prod_{k<i\le t} \frac{\partial h_i}{\partial h^{i-1}} \frac{\partial^+ h_k}{\partial W}$$

- 其中 $\frac{\partial^+ h_k}{\partial W}$ 表示不利用链式法则直接求导，也就是假如对于函数f(h(x))，对其直接求导结果如下：∂f(h(x))/∂x=f'(h(x))，也就是求导函数可以写成x的表达式，也就是将h(x)看成常数了。

在Yoshua Bengio 论文中（ http://proceedings.mlr.press/v28/pascanu13.pdf ）

证明了 $\left\| \prod_{k<i\le t} \frac{\partial h_i}{\partial h^{i-1}} \right\| \le \eta^{t-k}$ ，从而说明了这是梯度求导的一部分环节是一个指数模型，

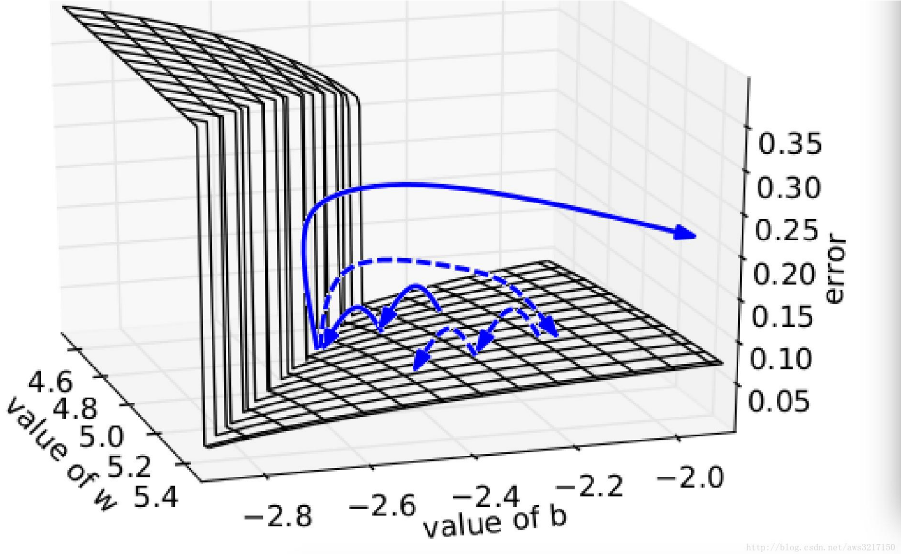
当η<1时，就会出现梯度消失问题，而当η>1时，梯度爆炸也就产生了。

递归神经网络RNN反向传播阶段

- 为了克服梯度消失的问题，LSTM和GRU模型便后续被推出了，为什么LSTM和GRU可以克服梯度消失问题呢？由于它们都有特殊的方式存储”记忆”，那么以前梯度比较大的”记忆”不会像简单的RNN一样马上被抹除，因此可以一定程度上克服梯度消失问题。（问题描述：**在普通RNN中于长序列而言，很早之前时刻输入的信息，对于当前时刻是不会产生影响的。**）
  另一个简单的技巧可以用来克服梯度爆炸的问题就是gradient clipping，也就是当你计算的梯度超过阈值c的或者小于阈值−c时候，便把此时的梯度设置成c或−c。

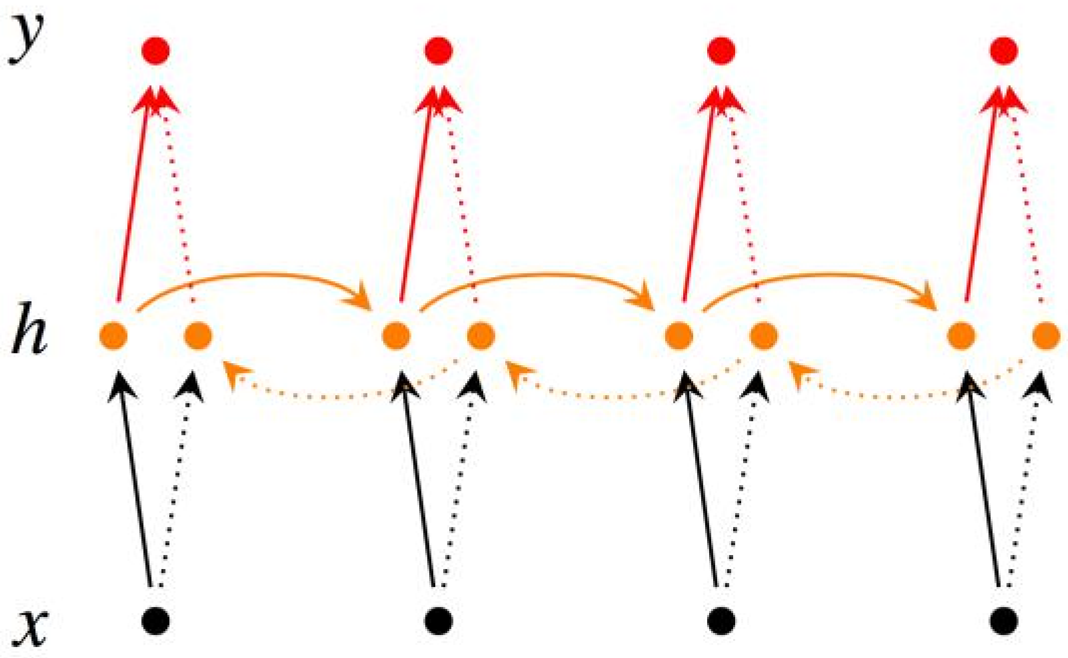
递归神经网络RNN反向传播阶段

- 下图所示是RNN的误差平面，可以看到RNN的误差平面要么非常陡峭，要么非常平坦，如果不采取任何措施，当你的参数在某一次更新之后，刚好碰到陡峭的地方，此时梯度变得非常大，那么你的参数更新也会非常大，很容易导致震荡问题。而如果你采取了gradient clipping这个技巧，那么即使你不幸碰到陡峭的地方，梯度也不会爆炸，因为梯度被限制在某个阈值c。



循环神经网络RNN-Bidirectional RNN

- Bidirectional RNN(双向RNN)假设当前t的输出不仅仅和之前的序列有关，并且还与之后的序列有关，例如：预测一个语句中缺失的词语那么需要根据上下文进行预测；Bidirectional RNN是一个相对简单的RNNs，由两个RNNs上下叠加在一起组成。输出由这两个RNNs的隐藏层的状态决定。

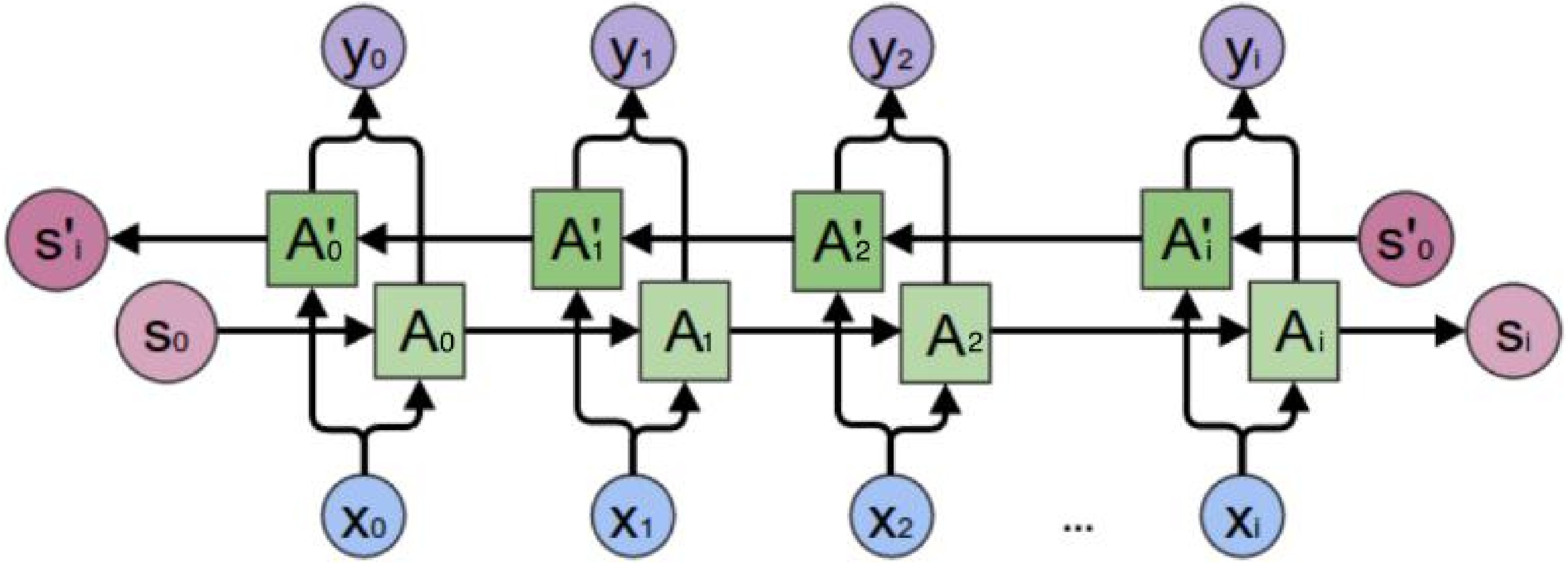


$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

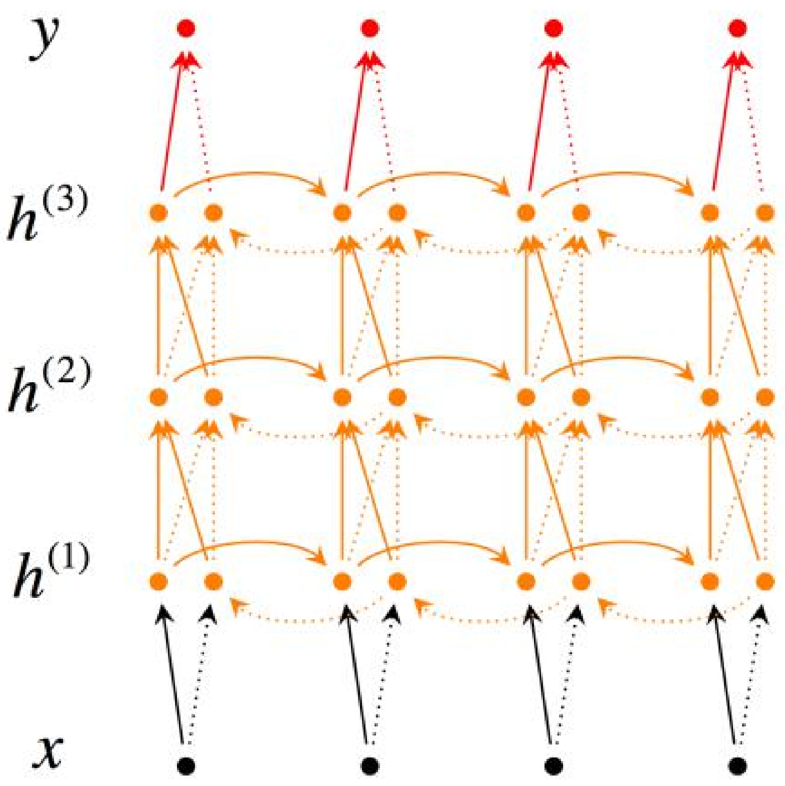
$$y_t = g(U[\overrightarrow{h}_t;\overleftarrow{h}_t] + c)$$

循环神经网络RNN-Bidirectional RNN



递归神经网络RNN-Deep(Bidirectional) RNN

- Deep Bidirectional RNN(深度双向RNN)类似Bidirectional RNN，区别在于每个每一步的输入有多层网络，这样的话该网络便具有更加强大的表达能力和学习能力，但是复杂性也提高了，同时需要训练更多的数据。



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)};\overleftarrow{h}_t^{(L)}]+c)$$

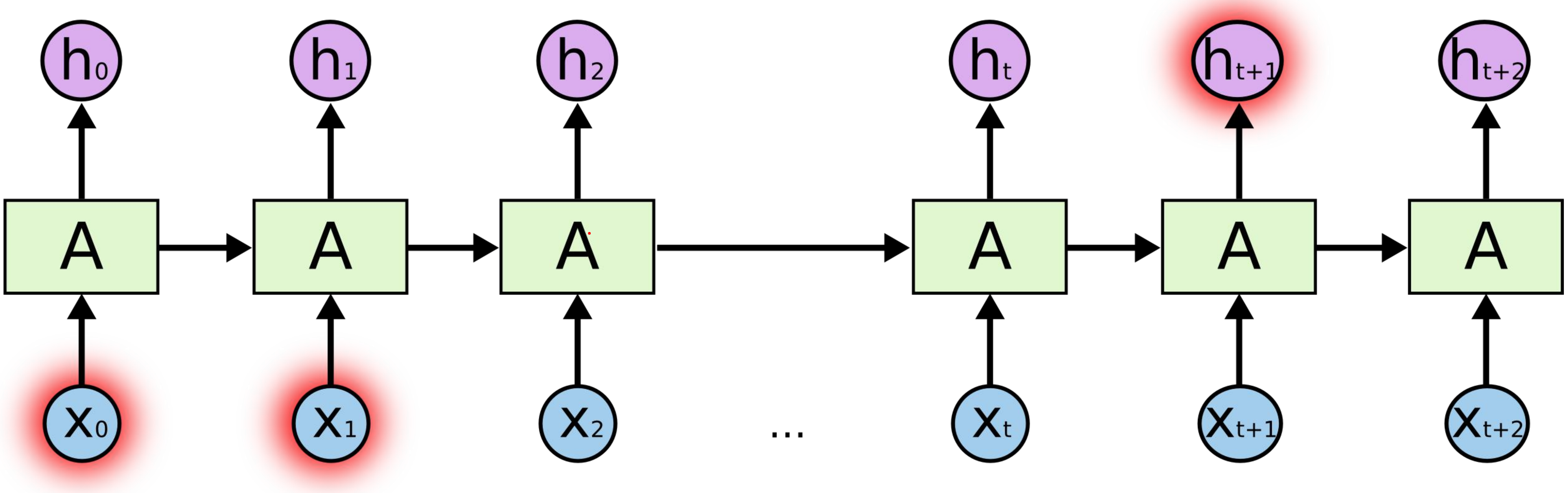
循环神经网络RNN-BPTT

- RNN的训练和CNN/ANN训练一样，同样适用BP算法误差反向传播算法。区别在于：RNN中的参数U\V\W是共享的，并且在随机梯度下降算法中，每一步的输出不仅仅依赖当前步的网络，并且还需要前若干步网络的状态，那么这种BP改版的算法叫做**Backpropagation Through Time**(**BPTT**)；BPTT算法和BP算法一样，在多层(多个输入时刻)训练过程中(长时依赖<即当前的输出和前面很长的一段序列有关，一般超过10步>)，可能产生梯度消失和梯度爆炸的问题。
- BPTT和BP算法思路一样，都是求偏导，区别在于需要考虑时间对step的影响

递归神经网络变形之LSTM

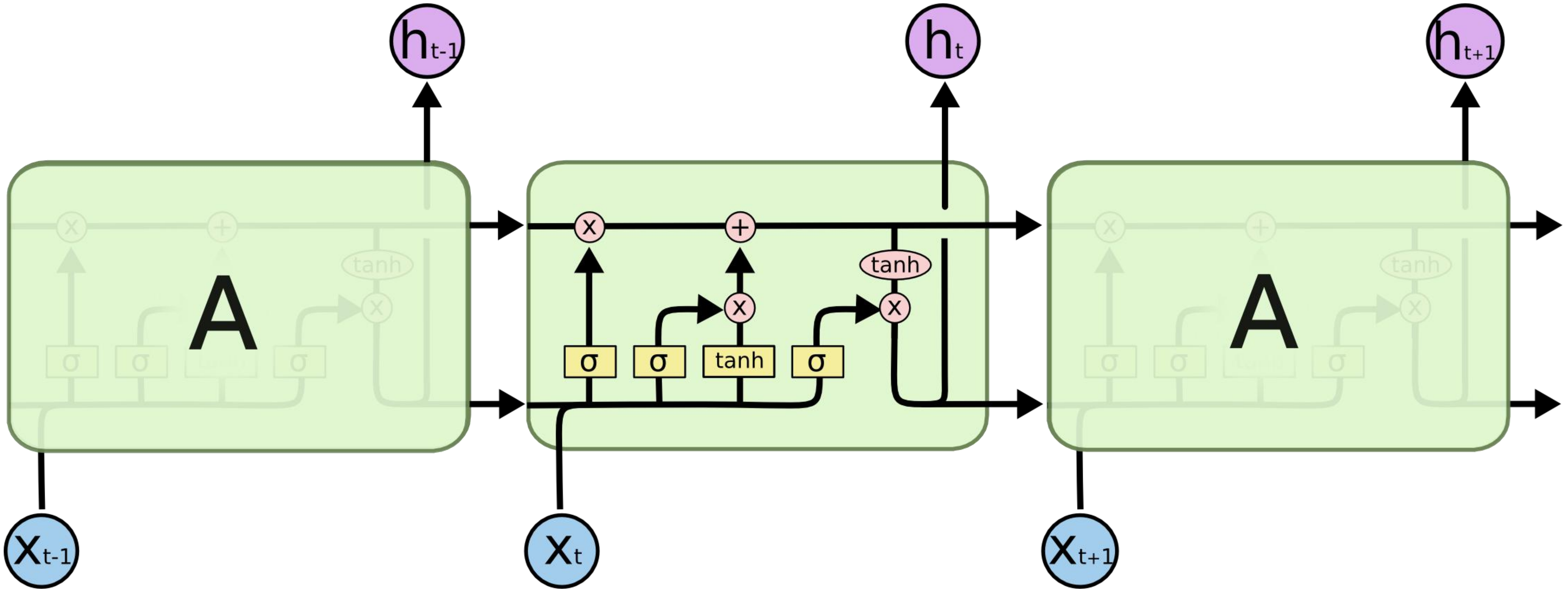
- 在RNN计算中，介绍到对于长期/**长时依赖**的问题，没法进行解决，可能产生梯度消失和梯度爆炸的问题；LSTM特别适合解决这类需要长时间依赖的问题。
- LSTM是RNN的一种变种，大体结构一致，区别在于：
  - LSTM的“记忆细胞”是改造过的
  - 该记录的信息会一直传递，不该记录的信息会被截断掉

递归神经网络变形之LSTM

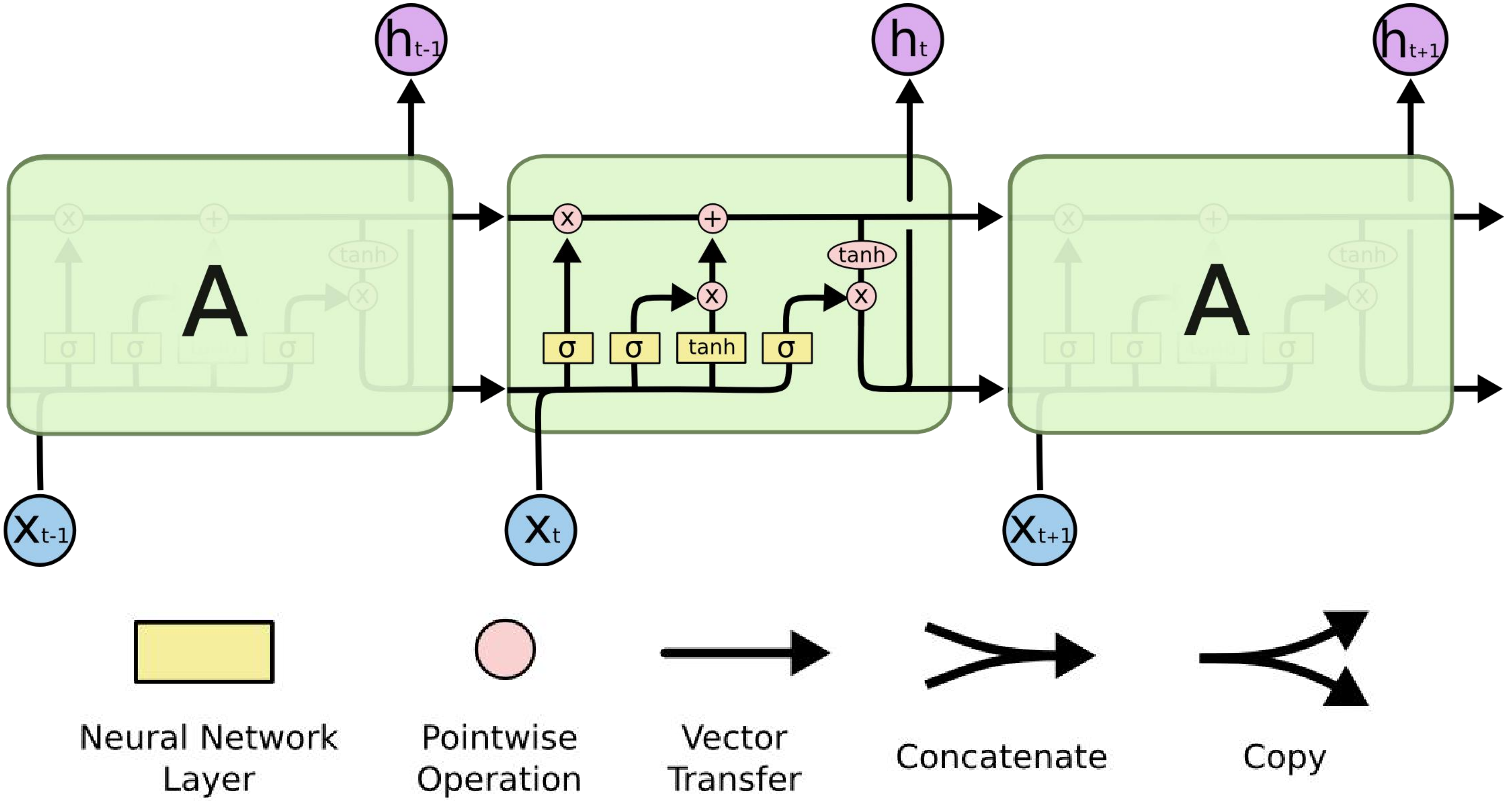


递归神经网络变形之LSTM

- 将“记忆细胞”变得稍微复杂一点点

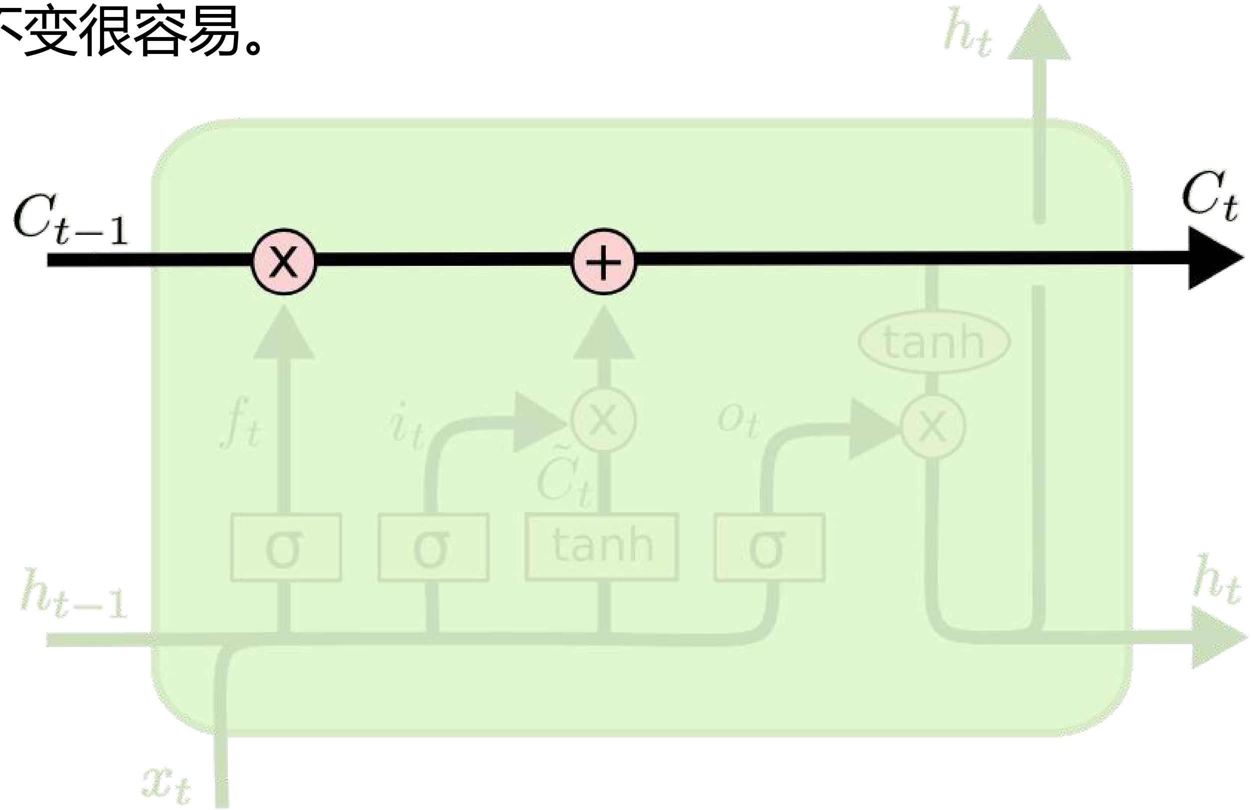


递归神经网络变形之LSTM



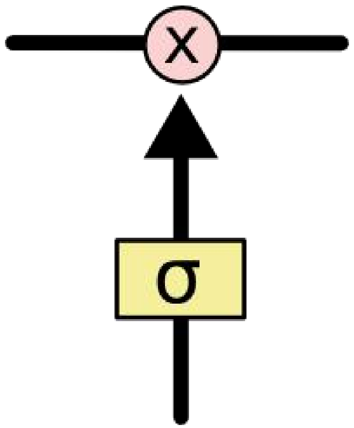
递归神经网络变形之LSTM

- LSTM关键：“细胞状态”
  - 细胞状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传保持不变很容易。



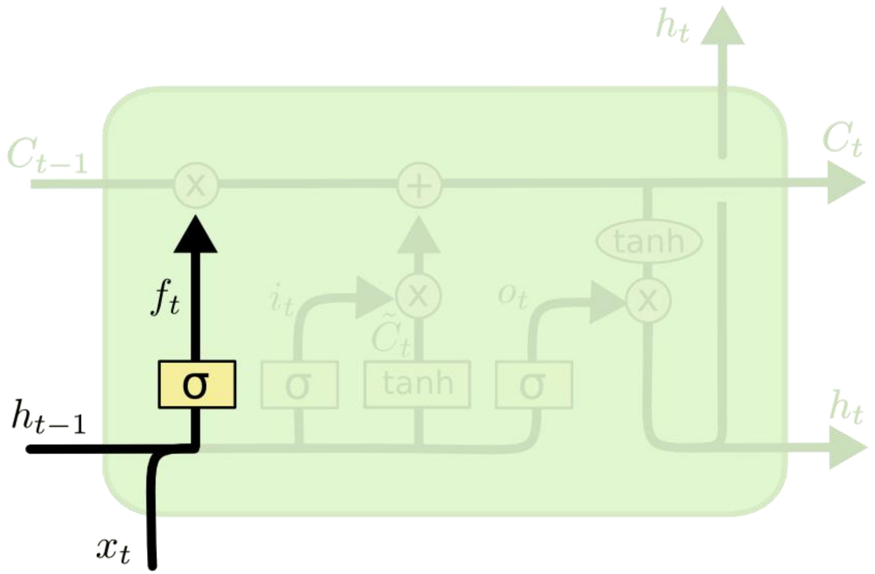
递归神经网络变形之LSTM

- LSTM怎么控制“细胞状态”？
  - LSTM可以通过gates(“门”)结构来去除或者增加“细胞状态”的信息
  - 包含一个sigmoid神经网络层次和一个pointwist乘法操作
  - Sigmoid层输出一个0到1之间的概率值，描述每个部分有多少量可以通过，0表示“不允许任务变量通过”，1表示“运行所有变量通过”
  - LSTM中主要有三个“门”结构来控制“细胞状态”



递归神经网络变形之LSTM

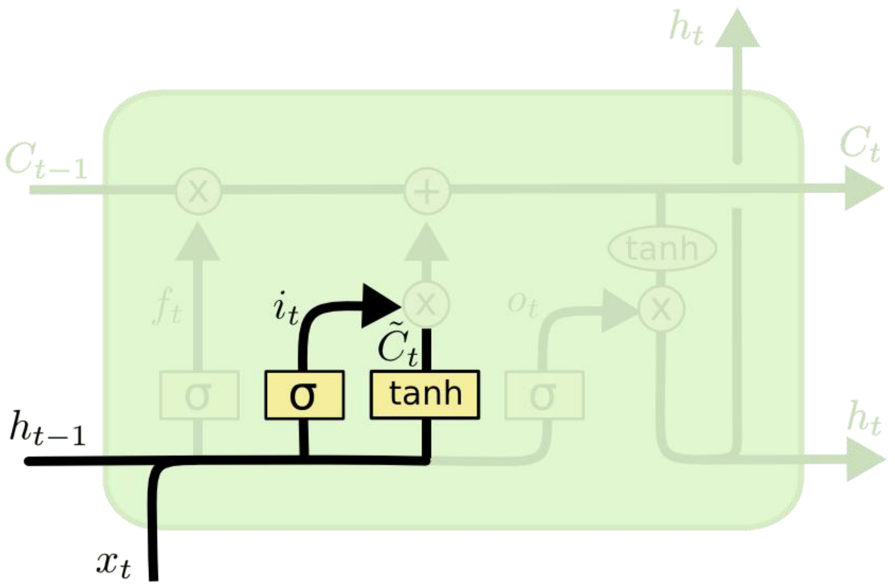
- 第一个“门”==>“忘记门”/“遗忘门”：决定从“细胞状态”中丢弃什么信息；比如在语言模型中，细胞状态可能包含了性别信息(“他”或者“她”)，当我们看到新的代名词的时候，可以考虑忘记旧的数据



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

递归神经网络变形之LSTM

- 第二个“门”==>“信息增加门”：决定放什么新信息到“细胞状态”中；
  - Sigmoid层决定什么值需要更新；
  - Tanh层创建一个新的候选向量C_t；
  - 主要是为了状态更新做准备

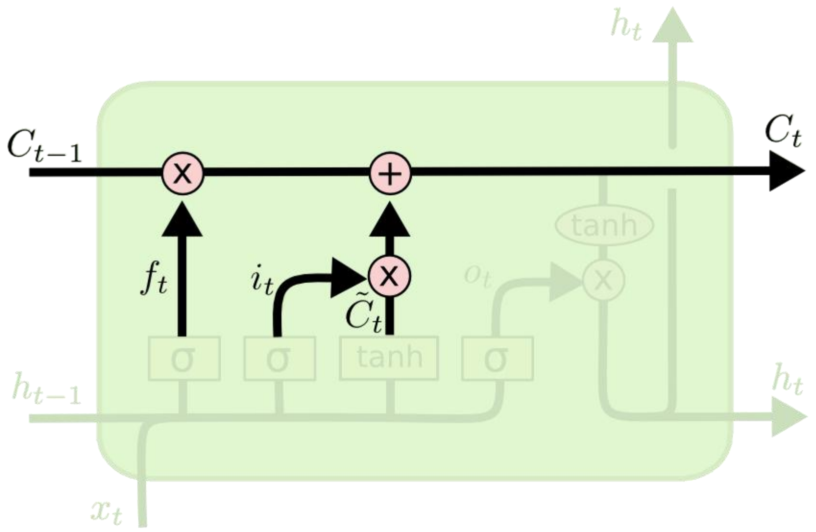


$$i_t = \sigma\left(W_i\cdot[h_{t-1}, x_t] \ + \ b_i\right)$$

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

递归神经网络变形之LSTM

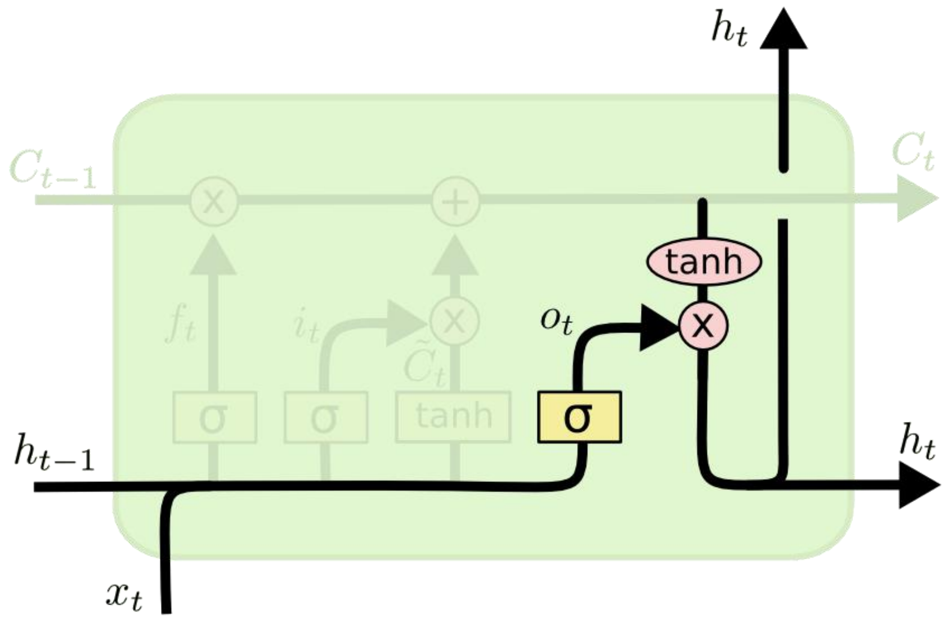
- 经过第一个和第二个“门”后，可以确定传递信息的删除和增加，即可以进行“细胞状态”的更新
  - 更新C_{t-1}为C_t;
  - 将旧状态与f_t相乘，丢失掉确定不要的信息；
  - 加上新的候选值i_t*C_t得到最终更新后的“细胞状态”



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

递归神经网络变形之LSTM

- 第三个“门”==>基于“细胞状态”得到输出，也就是“输出门”；
  - 首先运行一个sigmoid层来确定细胞状态的那个部分将输出
  - 使用tanh处理细胞状态得到一个-1到1之间的值，再将它和sigmoid门的输出相乘，输出程序确定输出的部分。

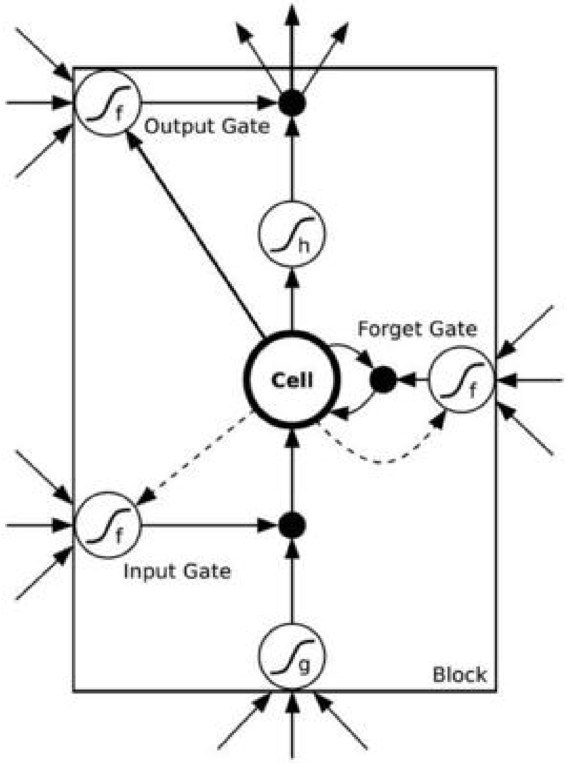


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

递归神经网络变形之LSTM

另外一种理解方式，图中方框我们称为记忆单元，其中实线箭头代表当前时刻的信息传递，虚线箭头表示上一时刻的信息传递。从结构图中我们看出，LSTM模型共增加了三个门：**输入门、遗忘门和输出门**。进入block的箭头代表输入，而出去的箭头代表输出。



递归神经网络变形之LSTM

LSTM的前向传播公式如下：

Input Gate:

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1}$$

$$b_\iota^t = f(a_\iota^t)$$

Forget Gate:

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

$$b_\phi^t = f(a_\phi^t)$$

递归神经网络变形之LSTM

LSTM的前向传播共公式如下：

Output Gate:

$$a_w^t = \sum_{i=1}^{I} w_{iw} x_i^t + \sum_{h=1}^{H} w_{hw} b_h^{t-1} + \sum_{c=1}^{C} w_{cw} s_c^t$$

$$b_w^t = f(a_w^t)$$

Cell:

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_l^t g(a_c^t)$$

Output:

$$b_c^t = b_w^t h(s_c^t)$$

递归神经网络变形之LSTM

上图有带h的权重矩阵均代表一种泛指，为LSTM的各种变种做准备，表示任意一条从上一时刻指向当前时刻的边，本文暂不考虑。与上篇公式类似，a代表汇集计算结果，b代表激活计算结果，**WiI**代表输入数据与输入门之间的权重矩阵，**WcI**代表上一时刻Cell状态与输入门之间的权重矩阵，**WiΦ**代表输入数据与遗忘门之间的权重矩阵，**WcΦ**代表上一时刻Cell状态与遗忘门之间的权重矩阵，**Wiω**代表输入数据与输出门之间的权重矩阵，**Wcω**代表Cell状态与输出门之间的权重矩阵，**Wic**代表输入层原有的权重矩阵。**需要注意的是，图中Cell一栏描述的是从下方输入到中间Cell输出的整个传播过程。**

递归神经网络变形之LSTM

反向传播

输出门不牵扯时间维度，我们可以直接写出输出门**Wiω**和**Wcω**的迭代公式为：

$$
\frac{\partial L}{\partial W_{iw}} = \frac{\partial L}{\partial b_c^t} \cdot \frac{\partial b_c^t}{\partial a_w^t} \cdot \frac{\partial a_w^t}{\partial W_{iw}} = L' \cdot h(S_c^t) \cdot f'(a_w^t) \cdot x_i^t
$$

$$
W_{iw} = W_{iw} - \eta \cdot \frac{\partial L}{\partial W_{iw}}
$$

$$
\frac{\partial L}{\partial W_{cw}} = \frac{\partial L}{\partial b_c^t} \cdot \frac{\partial b_c^t}{\partial a_w^t} \cdot \frac{\partial a_w^t}{\partial W_{cw}} = L' \cdot h(S_c^t) \cdot f'(a_w^t) \cdot S_c^t
$$

递归神经网络变形之LSTM

反向传播

$$W_{cw} = W_{cw} - \eta \cdot \frac{\partial L}{\partial W_{cw}}$$

遗忘门的权重矩阵**WiΦ**也可以直接给出，如下图：

$$\frac{\partial L}{\partial W_{i\phi}} = \frac{\partial L}{\partial b_c^t} \cdot \frac{\partial b_c^t}{\partial b_\phi^t} \cdot \frac{\partial b_\phi^t}{\partial W_{i\phi}}$$

$$= L' \cdot b_w^t \cdot h'(s_c^t) \cdot s_c^{t-1} \cdot f'(a_\phi^t) \cdot x_i^t$$

$$W_{i\phi} = W_{i\phi} - \eta \cdot \frac{\partial L}{\partial W_{i\phi}}$$

反向传播

对于遗忘门的权重矩阵**WcΦ**，由于是和上一时刻Cell状态做汇集计算，残差除了来自当前Cell，还来自下一时刻的Cell，因此需要写出下一时刻Cell传播至本时刻遗忘门的时间维度前向传播公式，如下图：

$$a_\phi^{t+1} = \sum_{c=1}^{C} W_{c\phi} S_c^t + \sum_{i=1}^{I} W_{i\phi} x_i^{t+1} + \sum_{h=1}^{H} W_{h\phi} b_h^t$$

$$b_\phi^{t+1} = f(a_\phi^{t+1})$$

递归神经网络变形之LSTM

反向传播

有了上面的公式，我们就能完整写出**WcΦ**的梯度公式如下：

$$
\begin{aligned}
\frac{\partial L}{\partial W^{o}_{c\phi}} &= \frac{\partial L}{\partial b^{t}_{c}} \cdot \frac{\partial b^{t}_{c}}{\partial b^{t}_{\phi}} \cdot \frac{\partial b^{t}_{\phi}}{\partial W_{c\phi}} \\
&= L' \cdot b^{t}_{w} \cdot h'(s^{t}_{c}) \cdot s^{t-1}_{c} \cdot f'(a^{t}_{\phi}) \cdot s^{t-1}_{c} \\
&= L' \cdot b^{t}_{w} \cdot h'(s^{t}_{c}) \cdot f'(a^{t}_{\phi}) \cdot {s^{t-1}_{c}}^{2}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial W^{h}_{c\phi}} &= \frac{\partial L}{\partial b^{t+1}_{\phi}} \cdot \frac{\partial b^{t+1}_{\phi}}{\partial b^{t}_{\phi}} \cdot \frac{\partial b^{t}_{\phi}}{\partial W_{c\phi}} \\
&= L'' \cdot f'(a^{t+1}_{\phi}) \cdot W_{c\phi} \cdot f'(a^{t}_{\phi}) \cdot {s^{t-1}_{c}}^{2}
\end{aligned}
$$

$$
\frac{\partial L}{\partial W_{c\phi}} = \frac{\partial L}{\partial W^{o}_{c\phi}} + \frac{\partial L}{\partial W^{h}_{c\phi}}
$$

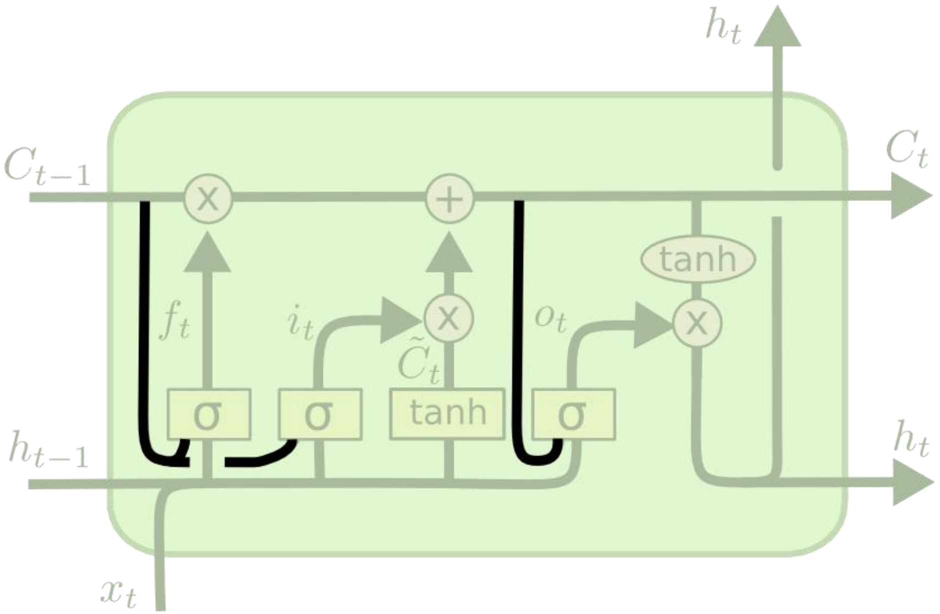
$$
W_{c\phi} = W_{c\phi} - \eta \cdot \frac{\partial L}{\partial W_{c\phi}}
$$

反向传播

推完遗忘门公式，就可以此类推输入门与Cell的公式。

递归神经网络变形之变种

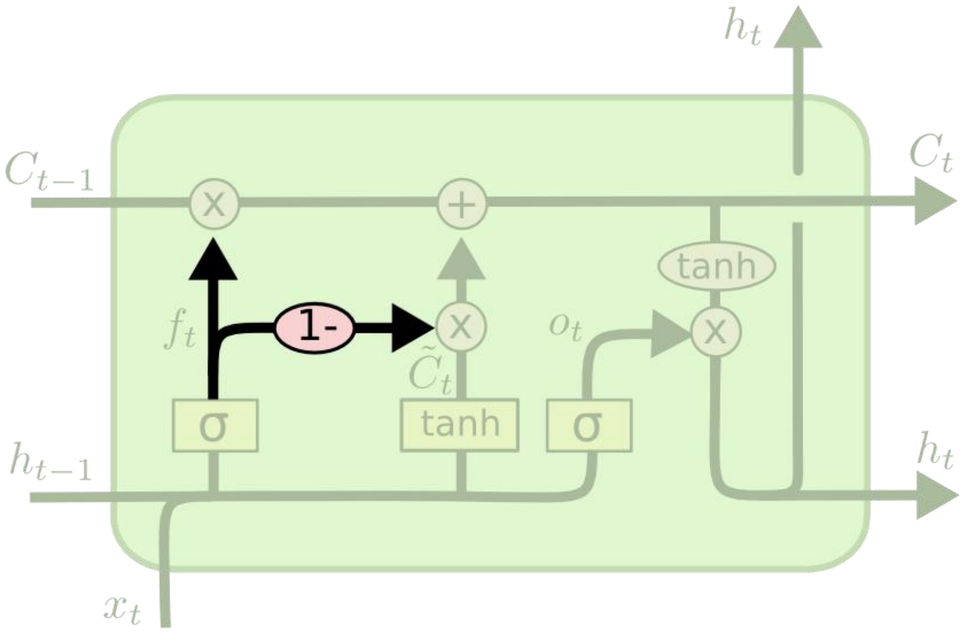
- 变种1
  - 增加“peephole connections”层
  - 让门层也接受细胞状态的输入



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

递归神经网络变形之变种

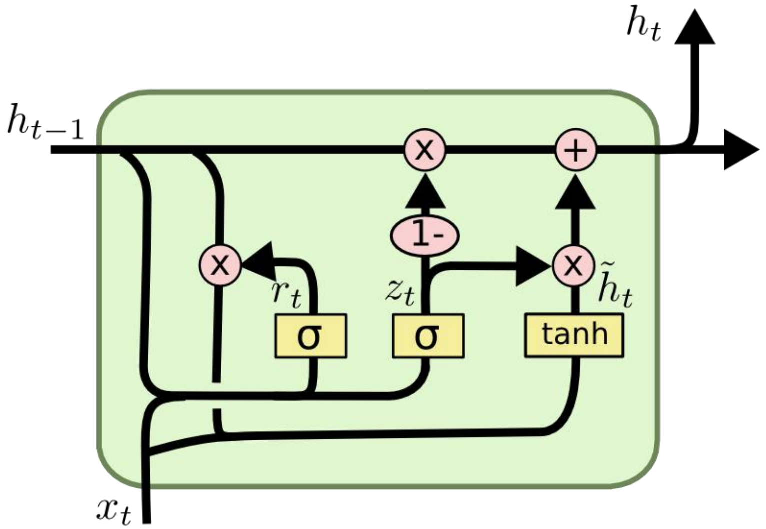
- 变种2
  - 通过耦合忘记门和更新输入门(第一个和第二个门)；也就是不再单独的考虑忘记什么、增加什么信息，而是一起进行考虑。



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

递归神经网络变形之变种

- Gated Recurrent Unit(GRU)，2014年提出
  - 将忘记门和输出门合并成为一个单一的更新门
  - 同时合并了数据单元状态和隐藏状态(细胞状态和输出状态)
  - 结构比LSTM的结构更加简单



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

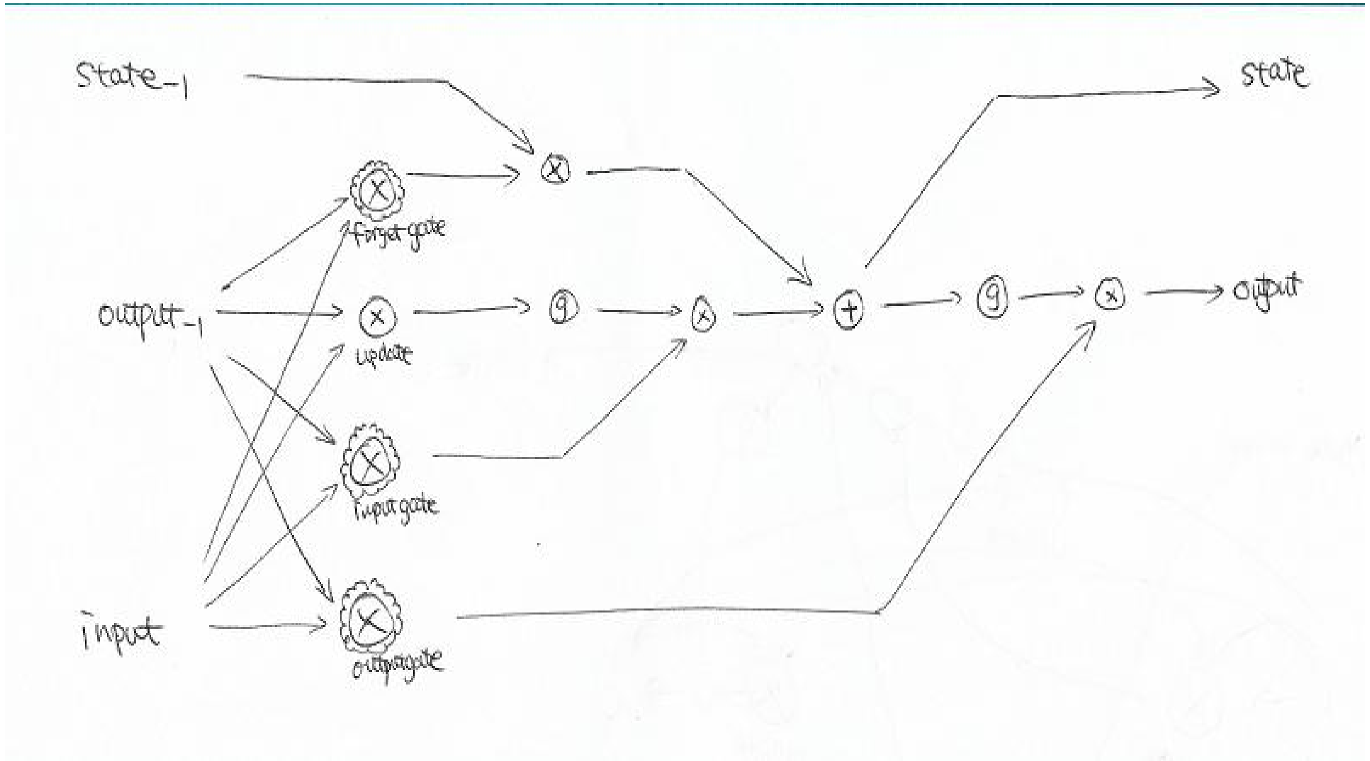
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

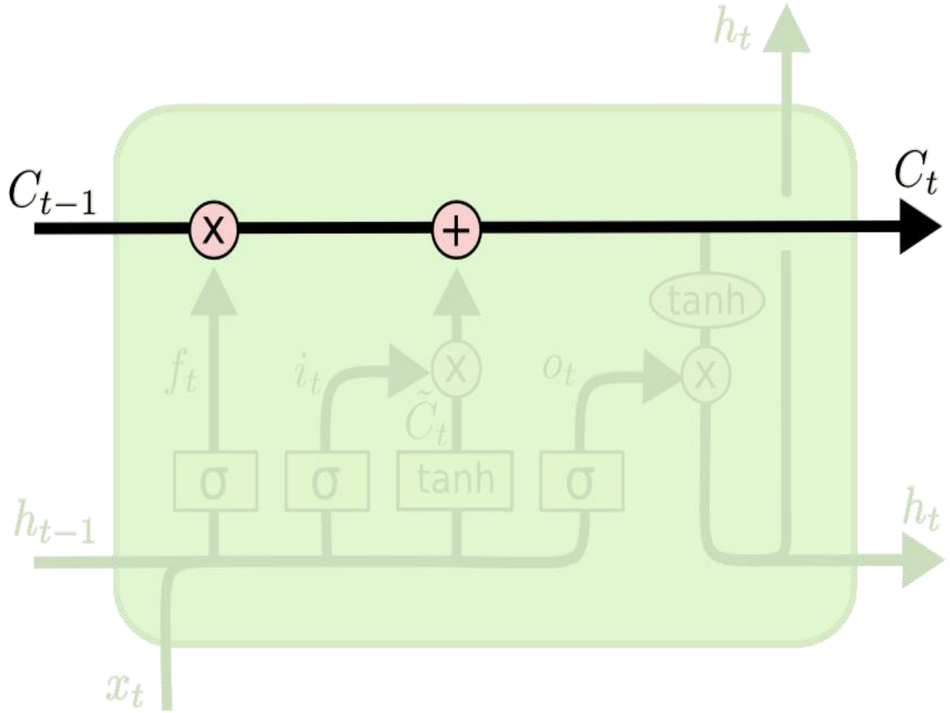
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

递归神经网络变形之变种

- 论文http://arxiv.org/pdf/1402.1128v1.pdf中定义的 LSTM Cell 似乎并不是我们平时熟悉的那种，而是如下图：





递归神经网络变形之变种

- (1) $\otimes$ 代表两个数据源乘上参数后相加。$\oplus$代表两个数据源相加。
- (2) $\otimes$ 外面再加花边的，代表两个数据源相乘后再取 sigmoid 。
- (3) 圆圈里是 gg 的，代表取 tanh 。
- (4) State−1 下标-1代表这是上一次迭代时的结果。