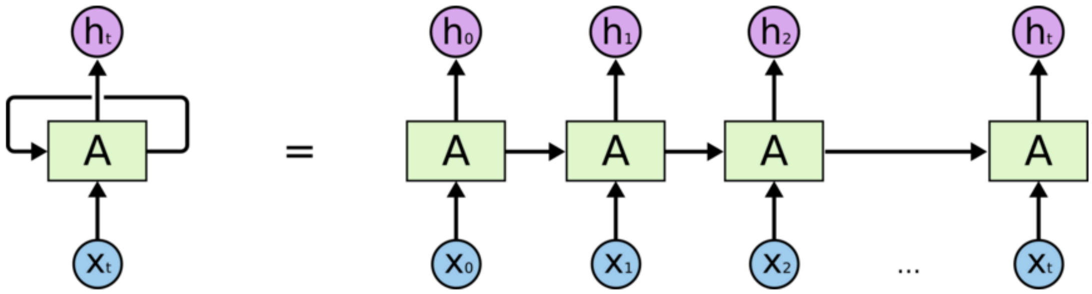
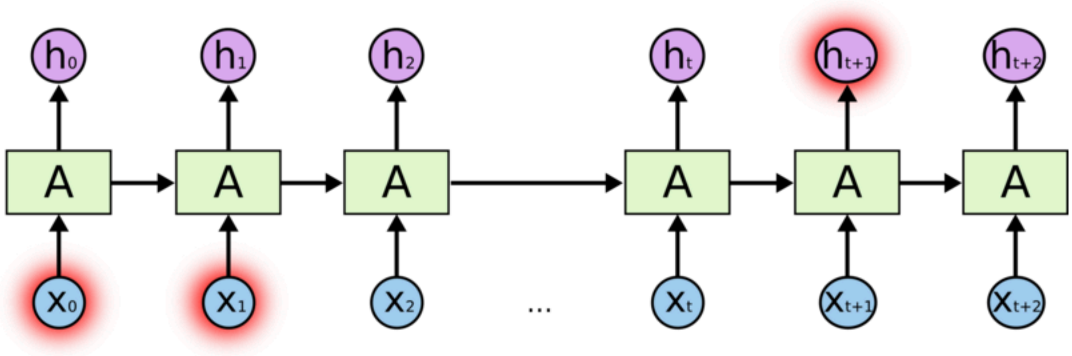
LSTM（Long Short Term Memory）

LSTM（Long-Short Term Memory/LSTM）是一种时间递归神经网络，论文首次发表于 1997 年。由于独特的设计结构，LSTM 适合于处理和预测时间序列中间隔和延迟非常长的重要事件。

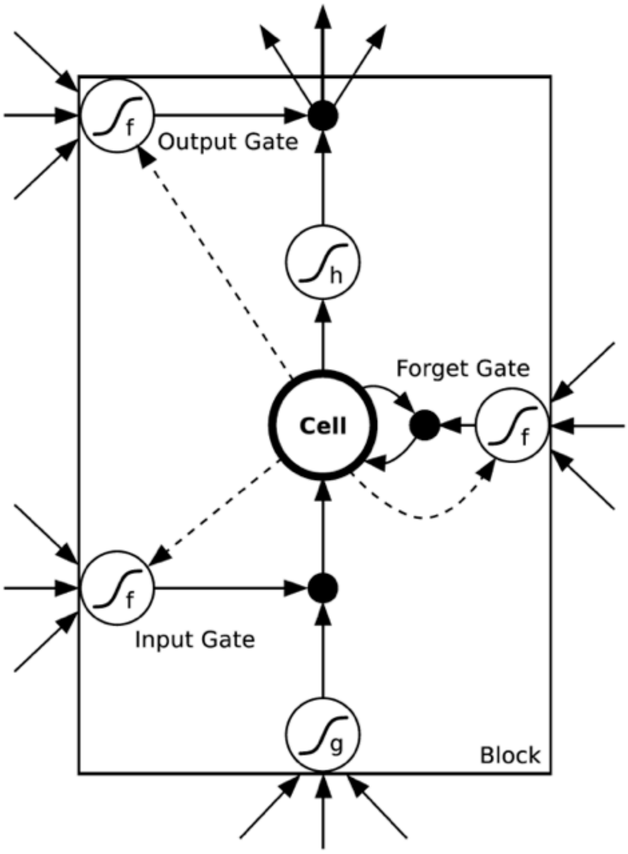
LSTM 是 RNN(Recurrent Neural Network)的特例，RNN 是一种节点定向连接成环的人工神经网络，其简化结构及展开形式如下。



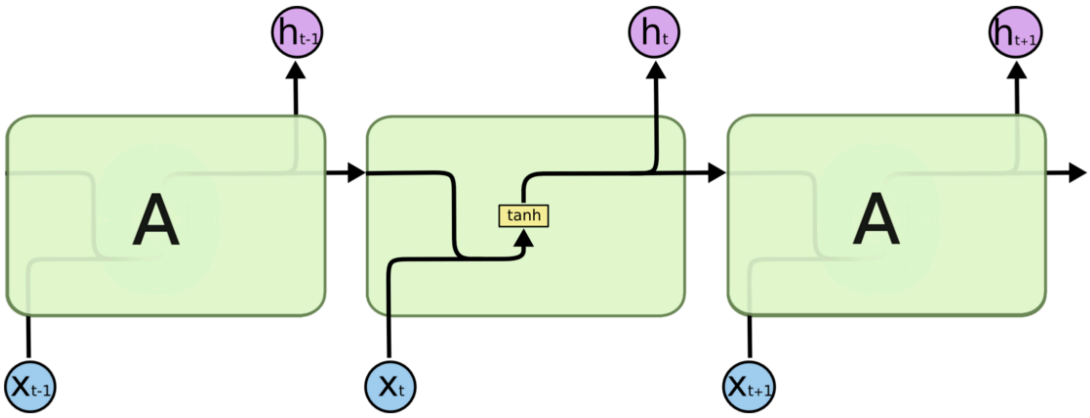
RNN 的优势在于能够通过先前的信息来指导当前的任务，而关键问题在于长期依赖（Long-Term Dependencies），所谓“长期依赖”是指当所需信息跨度过大时，如下图中 h(t+1)需要依赖于 x(0)和 x(1)，RNN 在计算过程中，后面时间节点相对前面时间节点的感知力下降（或根据链式法则推导求取梯度时所导致的梯度消失或爆炸），因此会导致方法失效。为了解决这个问题，有研究提出了 LSTM 算法。

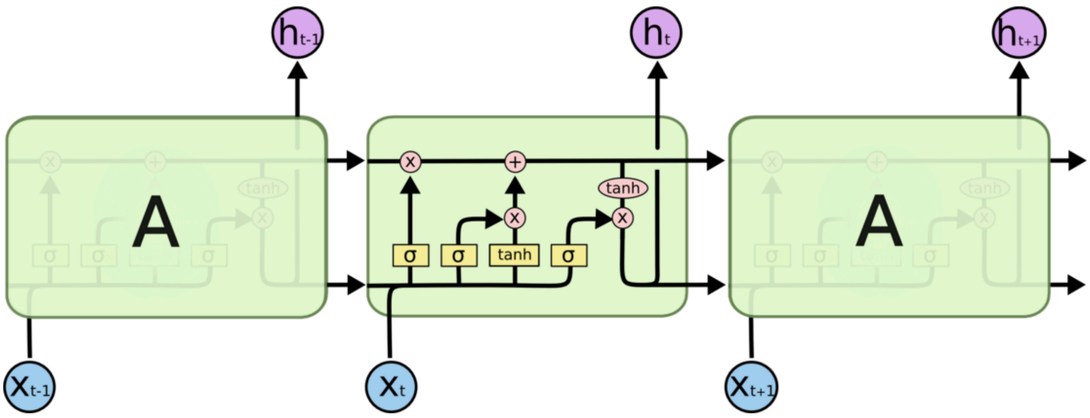


相比 RNN，LSTM 将隐含单元修改成为了如下所示的 block 单元。

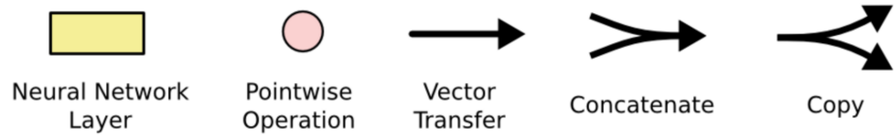


如下图所示分别是典型的 RNN 和 LSTM 结构。

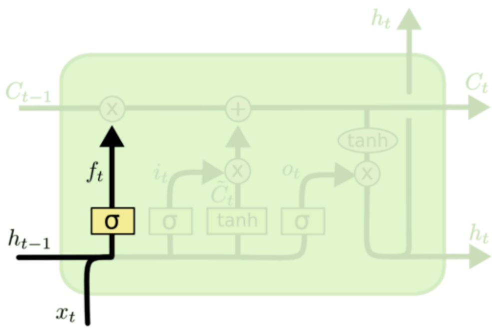




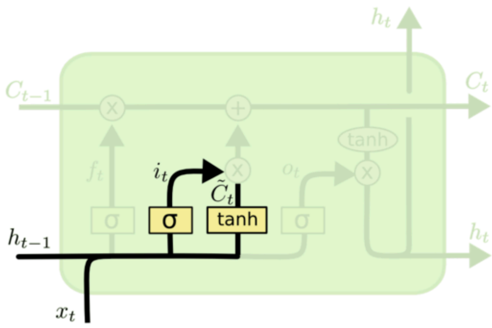
其中图中基本元素的含义如下。



由此可以推导出 LSTM 正向传播的公式如下。

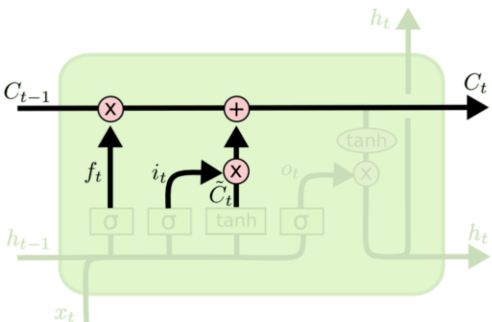


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

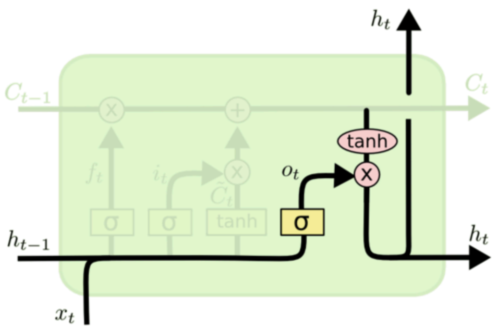


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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



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

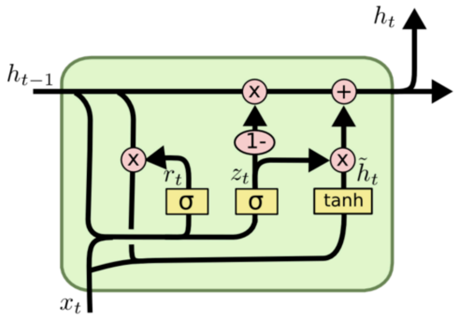


$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

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

其中，C(t)是隐含单元所传递的状态，f(t)是“forget gate layer”，控制上一阶段的隐含状态哪些需要保留，i(t)是“input gate layer”，控制本阶段的输入哪些需要对状态进行更新，$\widetilde{C(t)}$是本阶段候选状态，基于此，可以出计算本阶段的状态 C(t)，系统的输出和隐含单元输出分别是 o(t)和 h(t)。

LSTM 内部结构相对复杂，因此还有一种简化形式称为 GRU(Gated Recurrent Unit)，其将“forget gate”和“input gate”合并成为“update gate”，结构和正向传播公式如下所示。



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

为了理解 LSTM 的工作原理，需要对反向传播算法进行推导，可以参考这篇文章 http://nicodjimenez.github.io/2014/08/08/lstm.html。

LSTM 正向传播的公式重写如下。

$$g(t) = \phi(W_{gx}x(t) + W_{gh}h(t-1) + b_g)$$
$$i(t) = \sigma(W_{ix}x(t) + W_{ih}h(t-1) + b_i)$$
$$f(t) = \sigma(W_{fx}x(t) + W_{fh}h(t-1) + b_f)$$
$$o(t) = \sigma(W_{ox}x(t) + W_{oh}h(t-1) + b_o)$$
$$s(t) = g(t) * i(t) + s(t-1) * f(t)$$
$$h(t) = s(t) * o(t)$$

为了简化形式，将 x(t)和 h(t-1)连接成向量 xc(t)。

$$x_c(t) = [x(t), h(t-1)]$$

公式可简化为。

$$g(t) = \phi(W_g x_c(t) + b_g)$$
$$i(t) = \sigma(W_i x_c(t) + b_i)$$
$$f(t) = \sigma(W_f x_c(t) + b_f)$$
$$o(t) = \sigma(W_o x_c(t) + b_o).$$

选取方差函数为损失函数。

$$l(t) = f(h(t), y(t)) = \|h(t) - y(t)\|^2.$$

下面推导 LSTM 反向传播公式。假设 w 是模型参数，则根据链式法则，梯度计算如下。为了简化，引入 L(t)。

$$L(t) = \sum_{s=t}^{s=T} l(s)$$

$$\frac{dL}{dw} = \sum_{t=1}^{T} \sum_{i=1}^{M} \frac{dL}{dh_i(t)} \frac{dh_i(t)}{dw}$$, $$\frac{dL}{dh_i(t)} = \sum_{s=t}^{T} \frac{dl(s)}{dh_i(t)} = \frac{dL(t)}{dh_i(t)}$$

因此有：

$$\frac{dL}{dw} = \sum_{t=1}^{T} \sum_{i=1}^{M} \frac{dL(t)}{dh_i(t)} \frac{dh_i(t)}{dw}$$

下面结合源码进行分析，代码中变量的含义如下所示。

- `top_diff_h` $= \frac{dL(t)}{dh(t)} = \frac{dl(t)}{dh(t)} + \frac{dL(t+1)}{dh(t)}$
- `top_diff_s` $= \frac{dL(t+1)}{ds(t)}$.

- `self.state.bottom_diff_s` $= \frac{dL(t)}{ds(t)}$
- `self.state.bottom_diff_h` $= \frac{dL(t)}{dh(t-1)}$

- `self.param.wi_diff` $= \frac{dL}{dW_i}$
- …
- `self.param.bi_diff` $= \frac{dL}{db_i}$
- …

- `dxc` $= \frac{dL}{dx_c(t)}$

代码中 ds 的计算如下。

$$
\begin{aligned}
\frac{dL(t)}{ds_i(t)} &= \frac{dL(t)}{dh_i(t)}\frac{dh_i(t)}{ds_i(t)} + \frac{dL(t)}{dh_i(t+1)}\frac{dh_i(t+1)}{ds_i(t)} \\
&= \frac{dL(t)}{dh_i(t)}\frac{dh_i(t)}{ds_i(t)} + \frac{dL(t+1)}{dh_i(t+1)}\frac{dh_i(t+1)}{ds_i(t)} \\
&= \frac{dL(t)}{dh_i(t)}\frac{dh_i(t)}{ds_i(t)} + \frac{dL(t+1)}{ds_i(t)} \\
&= \frac{dL(t)}{dh_i(t)}\frac{dh_i(t)}{ds_i(t)} + [\texttt{top\_diff\_s}]_i.
\end{aligned}
$$

$$
h(t) = s(t) * o(t) \quad \Rightarrow \quad \frac{dL(t)}{dh_i(t)}\frac{dh_i(t)}{ds_i(t)} = o_i(t) * [\texttt{top\_diff\_h}]_i.
$$

因此有，ds = self.state.o * top_diff_h + top_diff_s。

其他推导和 RNN 类似，完整代码如下。

```python
def top_diff_is(self, top_diff_h, top_diff_s):
    # notice that top_diff_s is carried along the constant error carousel
    ds = self.state.o * top_diff_h + top_diff_s
    do = self.state.s * top_diff_h
    di = self.state.g * ds
    dg = self.state.i * ds
    df = self.s_prev * ds

    # diffs w.r.t. vector inside sigma / tanh function
    di_input = (1. - self.state.i) * self.state.i * di
    df_input = (1. - self.state.f) * self.state.f * df
    do_input = (1. - self.state.o) * self.state.o * do
    dg_input = (1. - self.state.g ** 2) * dg

    # diffs w.r.t. inputs
    self.param.wi_diff += np.outer(di_input, self.xc)
    self.param.wf_diff += np.outer(df_input, self.xc)
    self.param.wo_diff += np.outer(do_input, self.xc)
    self.param.wg_diff += np.outer(dg_input, self.xc)
    self.param.bi_diff += di_input
    self.param.bf_diff += df_input
    self.param.bo_diff += do_input
    self.param.bg_diff += dg_input

    # compute bottom diff
    dxc = np.zeros_like(self.xc)
    dxc += np.dot(self.param.wi.T, di_input)
    dxc += np.dot(self.param.wf.T, df_input)
    dxc += np.dot(self.param.wo.T, do_input)
    dxc += np.dot(self.param.wg.T, dg_input)

    # save bottom diffs
    self.state.bottom_diff_s = ds * self.state.f
    self.state.bottom_diff_x = dxc[:self.param.x_dim]
    self.state.bottom_diff_h = dxc[self.param.x_dim:]
```

参考连接

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- http://nicodjimenez.github.io/2014/08/08/lstm.html
- https://github.com/nicodjimenez/lstm