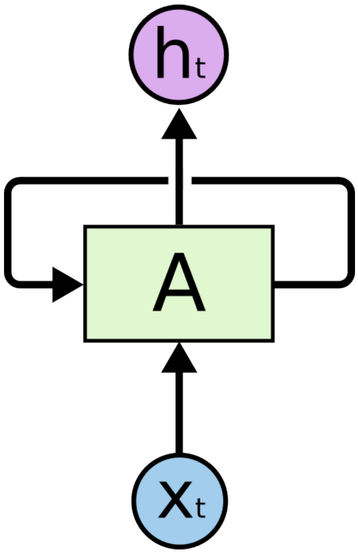
LSTM

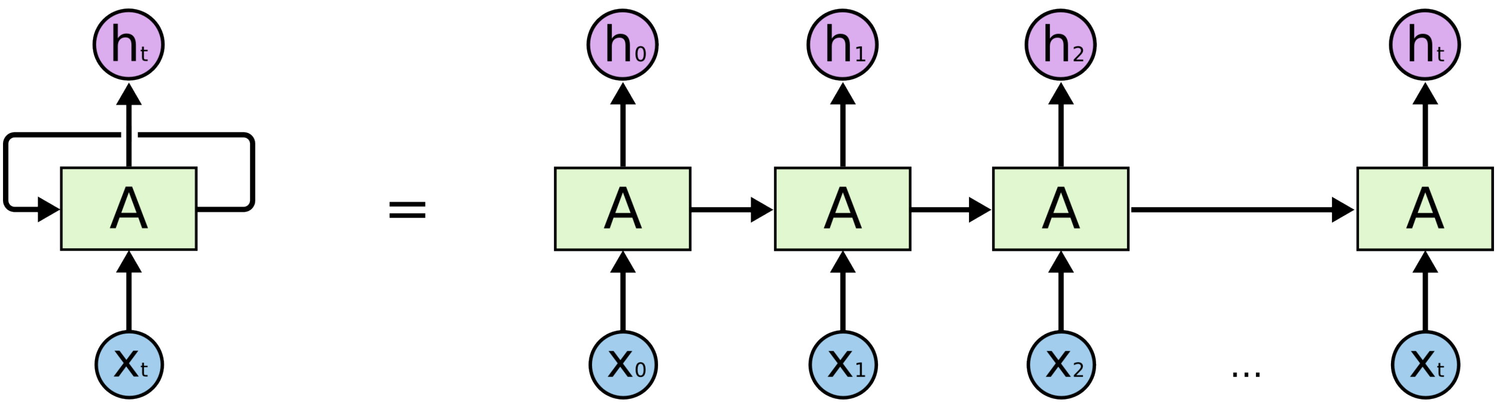
Recurrent Neural Networks 循环神经网络

示意图：



Recurrent Neural Networks have loops.

受到人类理解语言的启发，循环神经网络设计了一个循环的结构：网络的每一个单元都可以向下一个传递一些信息：



An unrolled recurrent neural network.

RNN 网络的运作方式决定了它擅长处理前后有关联的信息，比如语音识别、语言建模、翻译等。

RNN 网络的不足： Long-Term Dependencies

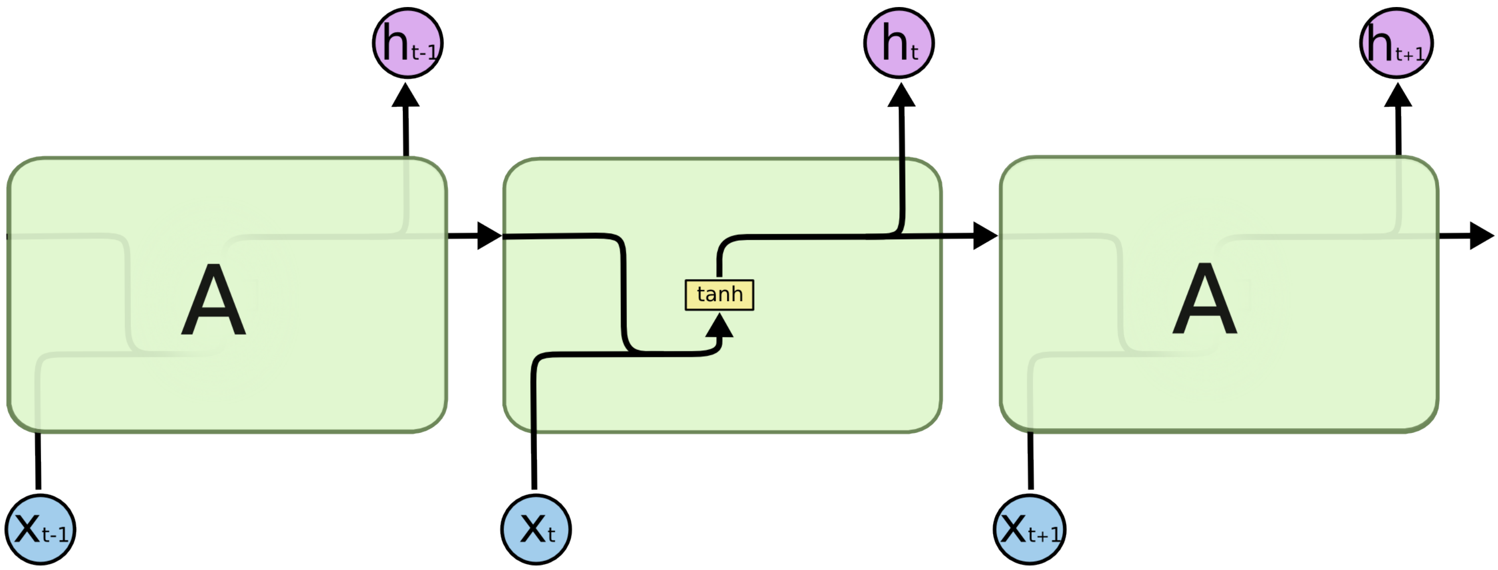
举个例子，假设我们要判断“天空中飘着 __" 的下一个字，显然这个词应该是云。但是如果我们的程序只着眼“飘着”这个动词，它得出的结论很可能是（水面上飘着）”垃圾袋“。因此，我们的程序应该能够利用过去的信息进行预测

理论上说，RNN 神经网络应该能够很好地学习这些规律，但实际上并非如此。

LSTM Networks

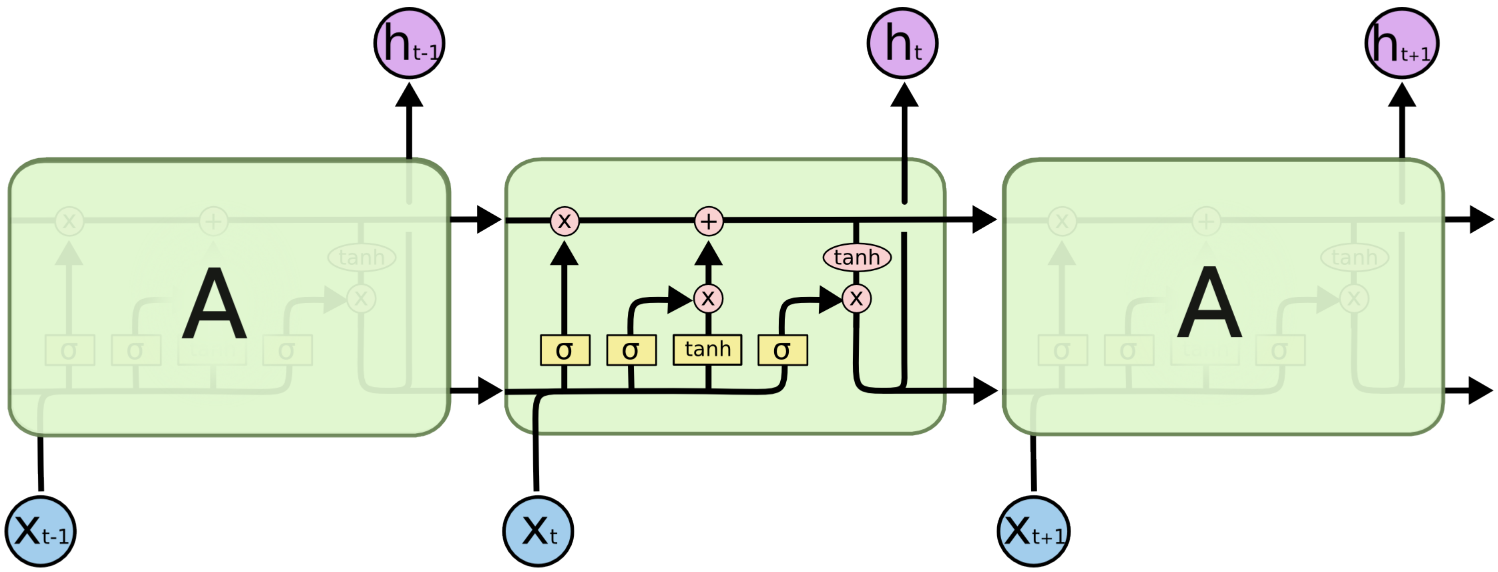
Long Short Term Memory networks，即 LSTM 网络，因此被提出来。它被证实在很多问题上有出色的表现。

RNN 网络可以很简单（如下图，只有一个 tanh 层）

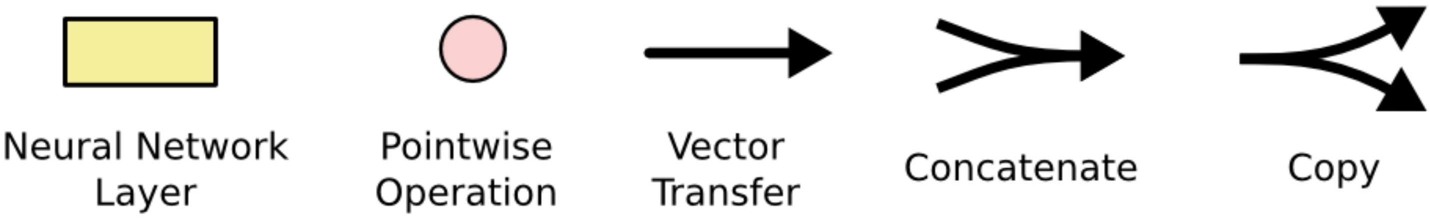


The repeating module in a standard RNN contains a single layer.

而 LSTM 网络有着不同的结构

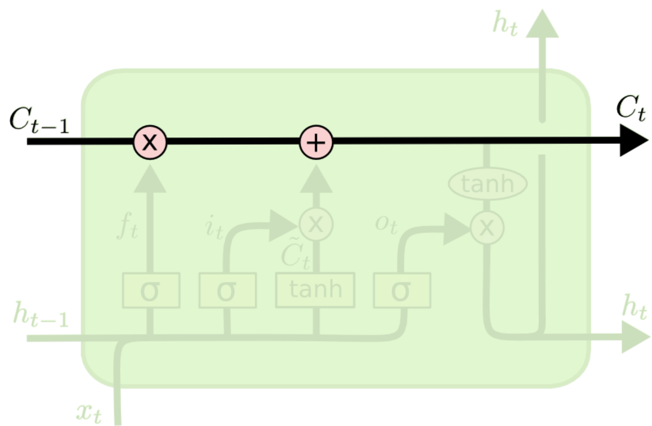


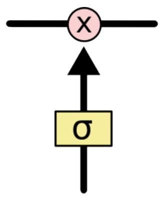
The repeating module in an LSTM contains four interacting layers.



核心精神

LSTM 网络的核心在于由 `Sigmoid` 函数构成的Control Gate

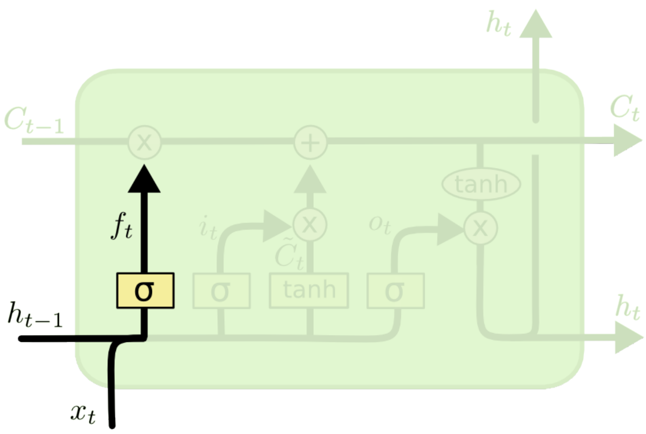




Sigmoid 函数输出 0–1 的值，0 代表不通过，1代表通过。这些门可以控制状态的传递，起到保护各个module的作用

详细分析

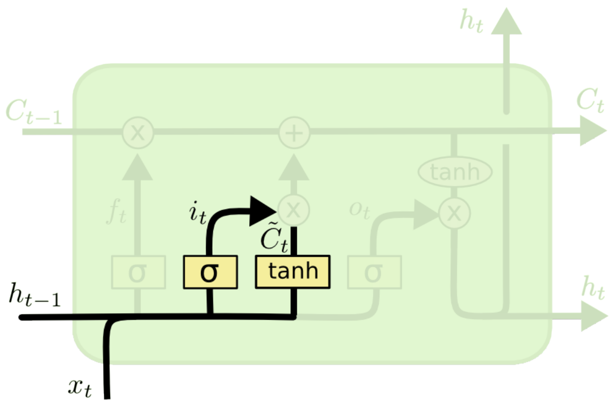
1. 确定需要舍弃的信息



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

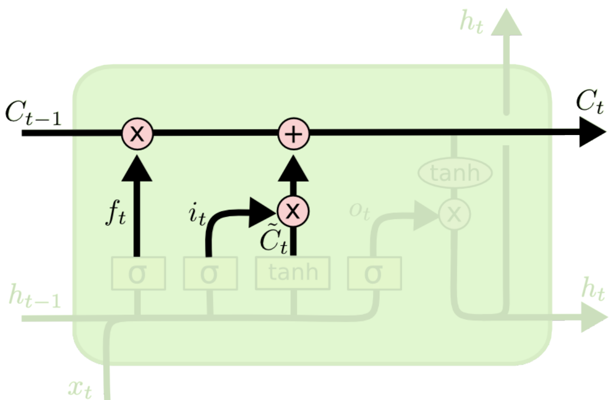
2. 确定当前单元要保存的信息

   1. 从输入值出发，决定候选值$\tilde{C}_t$



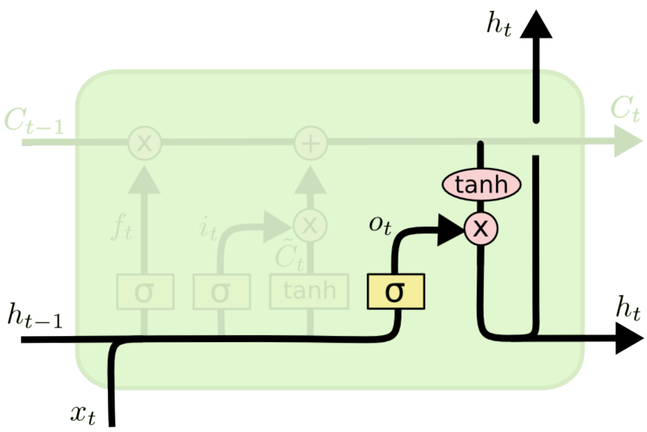
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

   2. 从C_{t-1}出发， 更新C_t



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

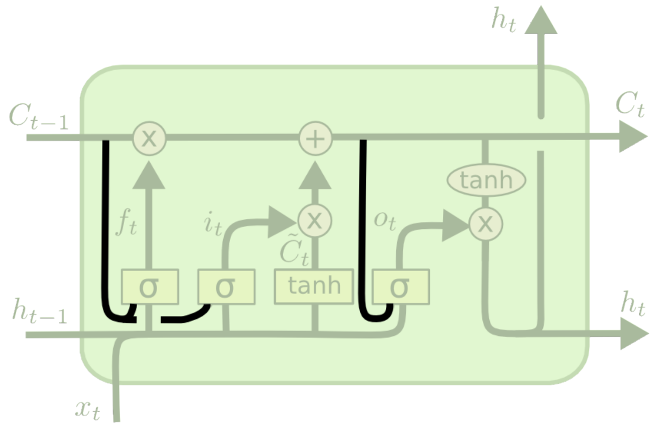
3. 确定要输出的信息



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

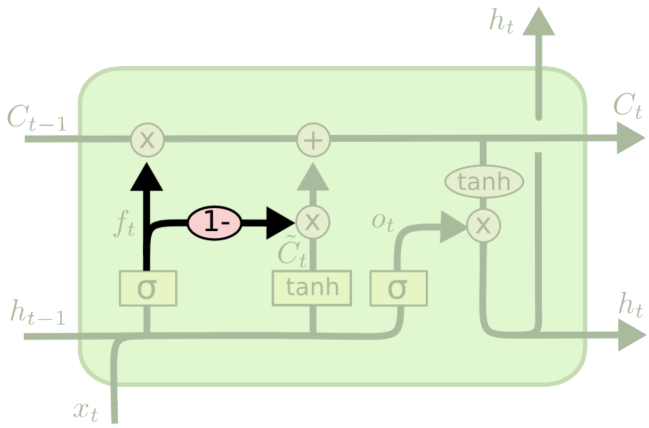
LSTM 的变体

by Gers & Schmidhuber (2000) 加入了“peephole connection”



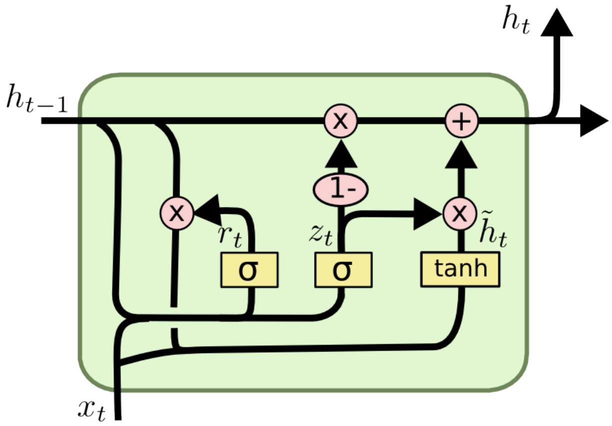
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \ + \ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \ + \ b_o\right)$$

加入遗忘单元



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

by Cho, et al. (2014) 更加复杂的模型



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

Pytorch 实现

```
torch.nn.LSTM(*args, **kwargs)
```

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.
For each element in the input sequence, each layer computes the following function:

$$
\begin{aligned}
i_t &= \sigma(W_{ii} x_t + b_{ii} + W_{hi} h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if} x_t + b_{if} + W_{hf} h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig} x_t + b_{ig} + W_{hg} h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io} x_t + b_{io} + W_{ho} h_{(t-1)} + b_{ho}) \\
c_t &= f_t * c_{(t-1)} + i_t * g_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

Parameters:

- *input_size*
- *hidden_size*
- *num_layers*
- *bias*
- *batch_first*
- *dropout*
- *bidirectional*

Inputs: (h_0,c_0)
Outputs:(h_n,c_n)
Variables:

- *~LSTM.weight)ih_l[k]*
- *~LSTM.weight_hh_l[k]*
- *~LSTM.bias_ih_l[k]*
- *~LSTM.bias_hh_l[k]*

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```