循环神经网络与LSTM

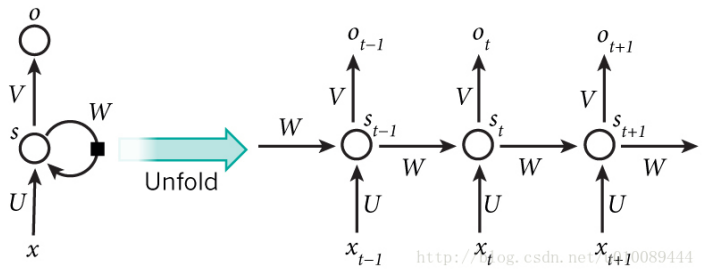
http://blog.csdn.net/u010089444/article/details/55211653

1. 循环神经网络

①基本结构

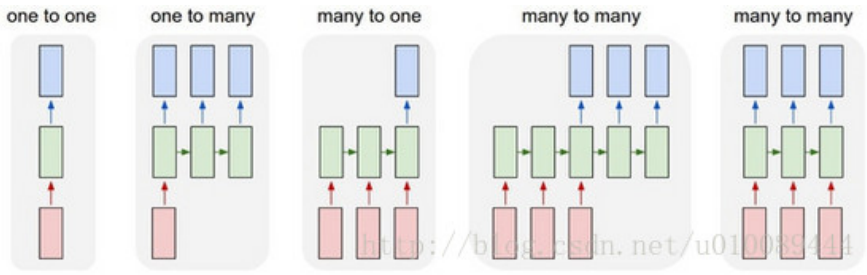
在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNN(Recurrent Neuron Network)是一种对序列数据建模的神经网络，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。下面是一个RNN模型的示例图，其中：

- Xt是t时刻的输入，例如单词中相应的one-hot向量
- St是t时刻的隐状态（memory），基于上一时刻的隐状态和当前输入得到：St=f(UXt+WSt-1)，其中f一般是非线性的激活函数，在计算S0时，即第一个单词的隐藏层状态，需要用到S-1，但是其并不存在，在实现中一般置为0
- Ot表示t时刻的输出，如下个单词的向量表示，Ot=softmax(VSt)
- 需要注意的是：在传统神经网络中，每一个网络层的参数是不共享的。而在RNN中，所有层次均**共享同样的参数**（例如上例中的U,V,W）。其反应出RNN中的每一步都在做相同的事，只是输入不同，因此大大地降低了网络中需要学习的参数。



图片名称

普通神经网络和卷积神经网络的一个显而易见的局限就是他们接收一个固定尺寸的向量作为输入（比如一张图像），并且产生一个固定尺寸的向量作为输出（比如针对不同分类的概率）。不仅如此，这些模型甚至对于上述映射的演算操作的步骤也是固定的（比如模型中的层数）。RNN的不同之处在于其允许我们对向量的序列进行操作：输入可以是序列，输出也可以是序列，在最一般化的情况下输入输出都可以是序列。下面是一些直观的例子：

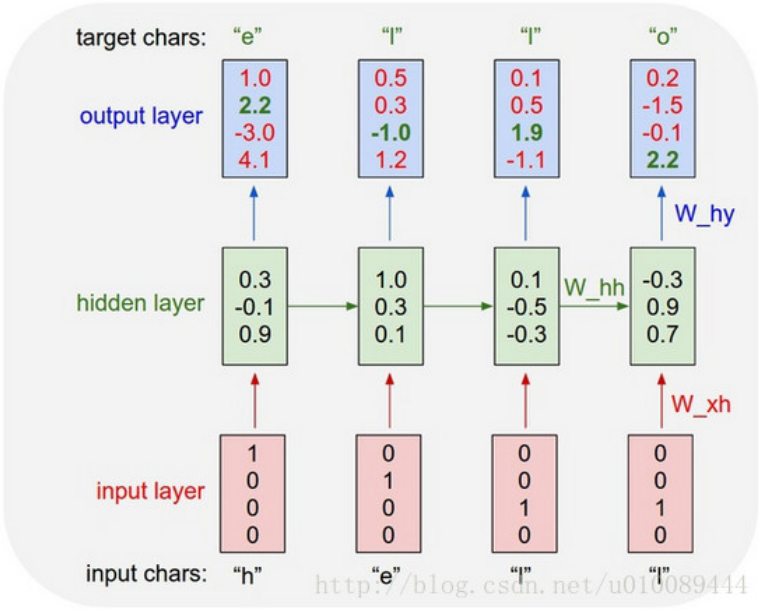


图片名称

上图中每个正方形代表一个向量，箭头代表函数（比如矩阵乘法）。输入向量是红色，输出向量是蓝色，绿色向量装的是RNN的状态。从左至右为：

1. 非RNN的普通过程，从固定尺寸的输入到固定尺寸的输出（比如图像分类）。
2. 输出是序列（例如图像标注：输入是一张图像，输出是单词的序列）。
3. 输入是序列（例如情绪分析：输入是一个句子，输出是对句子属于正面还是负面情绪的分类）。
4. 输入输出都是序列（比如机器翻译：RNN输入一个英文句子输出一个法文句子）。
5. 同步的输入输出序列（比如视频分类中，我们将对视频的每一帧都打标签）。

通过一个有趣的应用来更深入地加以体会：我们将利用RNN训练一个字母级别的语言模型。也就是说，给RNN输入巨量的文本，然后让其建模并根据一个序列中的前一个字母，给出下一个字母的概率分布。在下面的例子中，假设我们的字母表只由4个字母组成“helo”，然后利用训练序列“hello”训练RNN。该训练序列实际上是由4个训练样本组成：1.当h为上文时，下文字母选择的概率应该是e最高。2.l应该是he的下文。3.l应该是hel文本的下文。4.o应该是hell文本的下文。



图片名称

具体来说，该RNN的输入输出是4维，表示字典长度，隐层神经元数量是3个。我们将会把每个字母编码进一个1到k的向量（除对应字母为1外其余为0），然后利用 step 方法一次一个地将其输入给RNN。随后将观察到4维向量的序列（一个字母一个维度），我们希望绿色数字大，红色数字小。我们将这些输出向量理解为RNN关于序列下一个字母预测的信心程度。

②训练过程

循环神经网络的参数训练可以通过**随时间进行反向传播**（Backpropagation Through Time，BPTT）算法，BPTT算法是针对循环层的训练算法，它的基本原理和BP算法是一样的，也包含同样的三个步骤：

1. 前向计算每个神经元的输出值；
2. 反向计算每个神经元的误差项值，它是误差函数E对神经元j的加权输入的偏导数；
3. 计算每个权重的梯度，最后再用随机梯度下降算法更新权重。

（1）前向计算

我们假设输入向量x的维度是m，输出向量s的维度是n，则矩阵U的维度是n×m，矩阵W的维度是n×n。下面是前向计算展开成矩阵的样子，在这里我们用手写体字母表示向量的一个元素，它的下标表示它是这个向量的第几个元素，它的上标表示第几个时刻：

$$\begin{bmatrix} s_1^t \\ s_2^t \\ . \\ . \\ s_n^t \end{bmatrix} = f\left(\begin{bmatrix} u_{11} u_{12} \dots u_{1m} \\ u_{21} u_{22} \dots u_{2m} \\ . \\ . \\ u_{n1} u_{n2} \dots u_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_m \end{bmatrix} + \begin{bmatrix} w_{11} w_{12} \dots w_{1n} \\ w_{21} w_{22} \dots w_{2n} \\ . \\ . \\ w_{n1} w_{n2} \dots w_{nn} \end{bmatrix} \begin{bmatrix} s_1^{t-1} \\ s_2^{t-1} \\ . \\ . \\ s_n^{t-1} \end{bmatrix} \right) \quad (19)$$

（2）误差项的计算

在这里我们用手写体字母表示向量的一个元素，它的下标表示它是这个向量的第几个元素，它的上标表示第几个时刻。例如，s_j^t表示向量s的第j个元素在t时刻的值。u_{ji}表示输入层第i个神经元到循环层第j个神经元的权重。w_{ji}表示循环层第t-1时刻的第i个神经元到循环层第t个时刻的第j个神经元的权重。

误差项的计算

BTPP算法将第l层t时刻的误差项δ_t^l值沿两个方向传播，一个方向是其传递到上一层网络，得到δ_t^{l-1}，这部分只和权重矩阵U有关；另一个是方向是将其沿时间线传递到初始t_1时刻，得到δ_1^l，这部分只和权重矩阵W有关。

我们用向量net_t表示神经元在t时刻的加权输入，因为：

$$\mathrm{net}_t = Ux_t + Ws_{t-1} \tag{20}$$
$$\mathrm{s}_{t-1} = f(\mathrm{net}_{t-1}) \tag{21}$$

因此：

$$\frac{\partial \mathrm{net}_t}{\partial \mathrm{net}_{t-1}} = \frac{\partial \mathrm{net}_t}{\partial \mathrm{s}_{t-1}} \frac{\partial \mathrm{s}_{t-1}}{\partial \mathrm{net}_{t-1}} \tag{22}$$

我们用a表示列向量，用a^T表示行向量。上式的第一项是向量函数对向量求导，其结果为Jacobian矩阵：

$$\frac{\partial \mathrm{net}_t}{\partial \mathrm{s}_{t-1}} = \begin{bmatrix} \frac{\partial net_1^t}{\partial s_1^{t-1}} & \frac{\partial net_1^t}{\partial s_2^{t-1}} & \cdots & \frac{\partial net_1^t}{\partial s_n^{t-1}} \\ \frac{\partial net_2^t}{\partial s_1^{t-1}} & \frac{\partial net_2^t}{\partial s_2^{t-1}} & \cdots & \frac{\partial net_2^t}{\partial s_n^{t-1}} \\ & \vdots & & \\ \frac{\partial net_n^t}{\partial s_1^{t-1}} & \frac{\partial net_n^t}{\partial s_2^{t-1}} & \cdots & \frac{\partial net_n^t}{\partial s_n^{t-1}} \end{bmatrix} \tag{23}$$

$$= \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nn} \end{bmatrix} \tag{24}$$

$$= W \tag{25}$$

同理，上式第二项也是一个Jacobian矩阵：

$$\frac{\partial \mathrm{s}_{t-1}}{\partial \mathrm{net}_{t-1}} = \begin{bmatrix} \frac{\partial s_1^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_1^{t-1}}{\partial net_2^{t-1}} & \cdots & \frac{\partial s_1^{t-1}}{\partial net_n^{t-1}} \\ \frac{\partial s_2^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_2^{t-1}}{\partial net_2^{t-1}} & \cdots & \frac{\partial s_2^{t-1}}{\partial net_n^{t-1}} \\ & \vdots & & \\ \frac{\partial s_n^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_n^{t-1}}{\partial net_2^{t-1}} & \cdots & \frac{\partial s_n^{t-1}}{\partial net_n^{t-1}} \end{bmatrix} \tag{26}$$

$$= \begin{bmatrix} f'(net_1^{t-1}) & 0 & \cdots & 0 \\ 0 & f'(net_2^{t-1}) & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdots & f'(net_n^{t-1}) \end{bmatrix} \tag{27}$$

$$= diag[f'(\mathrm{net}_{t-1})] \tag{28}$$

其中，diag[a]表示根据向量a创建一个对角矩阵。最后，将两项合在一起，可得：

$$\frac{\partial \mathrm{net}_t}{\partial \mathrm{net}_{t-1}} = \frac{\partial \mathrm{net}_t}{\partial \mathrm{s}_{t-1}} \frac{\partial \mathrm{s}_{t-1}}{\partial \mathrm{net}_{t-1}} \tag{29}$$

$$= W diag[f'(\mathrm{net}_{t-1})] \tag{30}$$

$$= \begin{bmatrix} w_{11} f'(net_1^{t-1}) & w_{12} f'(net_2^{t-1}) & \cdots & w_{1n} f(net_n^{t-1}) \\ w_{21} f'(net_1^{t-1}) & w_{22} f'(net_2^{t-1}) & \cdots & w_{2n} f(net_n^{t-1}) \\ & \vdots & & \\ w_{n1} f'(net_1^{t-1}) & w_{n2} f'(net_2^{t-1}) & \cdots & w_{nn} f'(net_n^{t-1}) \end{bmatrix} \tag{31}$$

上式描述了将δ沿时间往前传递一个时刻的规律，有了这个规律，我们就可以求得任意时刻k的误差项δ_k：

$$\delta_k^T = \frac{\partial E}{\partial \text{net}_k} \tag{32}$$
$$= \frac{\partial E}{\partial \text{net}_t}\frac{\partial \text{net}_t}{\partial \text{net}_k} \tag{33}$$
$$= \frac{\partial E}{\partial \text{net}_t}\frac{\partial \text{net}_t}{\partial \text{net}_{t-1}}\frac{\partial \text{net}_{t-1}}{\partial \text{net}_{t-2}}\cdots\frac{\partial \text{net}_{k+1}}{\partial \text{net}_k} \tag{34}$$
$$= Wdiag[f'(\text{net}_{t-1})]Wdiag[f'(\text{net}_{t-2})]\ldots Wdiag[f'(\text{net}_k)]\delta_t^l \tag{35}$$
$$= \delta_t^T\prod_{i=k}^{t-1}Wdiag[f'(\text{net}_i)] \quad (式3) \tag{36}$$

式3就是将误差项沿时间反向传播的算法，循环层将误差项反向传递到上一层网络，与普通的全连接层是完全一样的

循环层的加权输入net^l与上一层的加权输入net^{l-1}关系如下：

$$\text{net}_t^l = U\text{a}_t^{l-1} + Ws_{t-1} \tag{37}$$
$$\text{a}_t^{l-1} = f^{l-1}(\text{net}^{l-1}) \tag{38}$$

上式中net_t^l是第l层神经元的加权输入(假设第l层是循环层)；net_t^{l-1}是第l-1层神经元的加权输入；a_t^{l-1}是第l-1层神经元的输出；f^{l-1}是第l-1层的激活函数。

$$\frac{\partial \text{net}^l}{\partial \text{net}^{l-1}} = \frac{\partial \text{net}^l}{\partial \text{a}^{l-1}}\frac{\partial \text{a}^{l-1}}{\partial \text{net}^{l-1}} \tag{39}$$
$$= Udiag[f'^{l-1}(\text{net}^{l-1})] \tag{40}$$

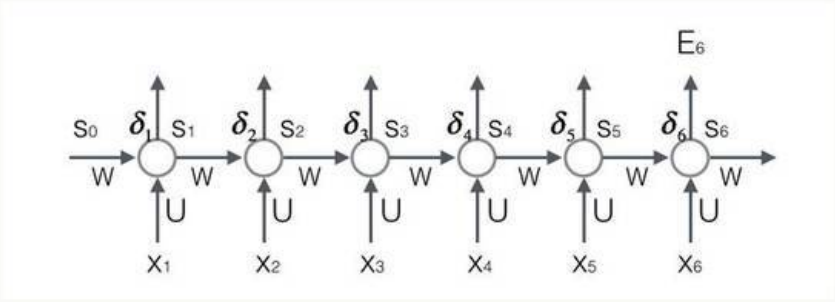
所以，

$$\delta_{l-1}^T = \frac{\partial E}{\partial \text{net}^{l-1}} \tag{41}$$
$$= \frac{\partial E}{\partial \text{net}^l}\frac{\partial \text{net}^l}{\partial \text{net}^{l-1}} \tag{42}$$
$$= \delta_l^T Udiag[f'^{l-1}(\text{net}^{l-1})] \quad (式4) \tag{43}$$

式4就是将误差项传递到上一层算法

（3）权重梯度的计算

首先，我们计算误差函数E对权重矩阵W的梯度$\frac{\partial E}{\partial W}$。



上图展示了我们到目前为止，在前两步中已经计算得到的量，包括每个时刻t**循环层**的输出值s_t，以及误差项δ_t。

回忆一下我们在文章零基础入门深度学习(3) - 神经网络和反向传播算法介绍的全连接网络的权重梯度计算算法：只要知道了任意一个时刻的误差项δ_t，以及上一个时刻循环层的输出值s_{t-1}，就可以按照下面的公式求出权重矩阵在t时刻的梯度$\nabla_{W_t}E$：

$$\nabla_{W_t}E = \begin{bmatrix} \delta_1^t s_1^{t-1} & \delta_1^t s_2^{t-1} & \cdots & \delta_1^t s_n^{t-1} \\ \delta_2^t s_1^{t-1} & \delta_2^t s_2^{t-1} & \cdots & \delta_2^t s_n^{t-1} \\ \cdot & & & \\ \cdot & & & \\ \delta_n^t s_1^{t-1} & \delta_n^t s_2^{t-1} & \cdots & \delta_n^t s_n^{t-1} \end{bmatrix} \quad (式5)$$

在式5中，δ_i^t表示t时刻误差项向量的第i个分量；s_i^{t-1}表示t-1时刻循环层第i个神经元的输出值。

我们下面可以简单推导一下式5。

我们知道：

$$\text{net}_t = Ux_t + Ws_{t-1} \tag{44}$$

$$\begin{bmatrix} net_1^t \\ net_2^t \\ . \\ . \\ net_n^t \end{bmatrix} = Ux_t + \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ . & & & \\ . & & & \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix} \begin{bmatrix} s_1^{t-1} \\ s_2^{t-1} \\ . \\ . \\ s_n^{t-1} \end{bmatrix} \tag{45}$$

$$= Ux_t + \begin{bmatrix} w_{11}s_1^{t-1} + w_{12}s_2^{t-1} \dots w_{1n}s_n^{t-1} \\ w_{21}s_1^{t-1} + w_{22}s_2^{t-1} \dots w_{2n}s_n^{t-1} \\ . \\ . \\ w_{n1}s_1^{t-1} + w_{n2}s_2^{t-1} \dots w_{nn}s_n^{t-1} \end{bmatrix} \tag{46}$$

因为对W求导与Ux_t无关，我们不再考虑。现在，我们考虑对权重项w_{ji}求导。通过观察上式我们可以看到w_{ji}只与net_j^t有关，所以：

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial net_j^t} \frac{\partial net_j^t}{\partial w_{ji}} \tag{47}$$

$$= \delta_j^t s_i^{t-1} \tag{48}$$

按照上面的规律就可以生成式5里面的矩阵。

我们已经求得了权重矩阵W在t时刻的梯度$\nabla_{W_t}E$，最终的梯度$\nabla_W E$是各个时刻的梯度之和：

$$\nabla_W E = \sum_{i=1}^{t} \nabla_{W_i} E \tag{49}$$

$$= \begin{bmatrix} \delta_1^t s_1^{t-1} & \delta_1^t s_2^{t-1} & \dots & \delta_1^t s_n^{t-1} \\ \delta_2^t s_1^{t-1} & \delta_2^t s_2^{t-1} & \dots & \delta_2^t s_n^{t-1} \\ . & & & \\ . & & & \\ \delta_n^t s_1^{t-1} & \delta_n^t s_2^{t-1} & \dots & \delta_n^t s_n^{t-1} \end{bmatrix} + \dots + \begin{bmatrix} \delta_1^1 s_1^0 & \delta_1^1 s_2^0 & \dots & \delta_1^1 s_n^0 \\ \delta_2^1 s_1^0 & \delta_2^1 s_2^0 & \dots & \delta_2^1 s_n^0 \\ . & & & \\ . & & & \\ \delta_n^1 s_1^0 & \delta_n^1 s_2^0 & \dots & \delta_n^1 s_n^0 \end{bmatrix} \tag{式6)(50}$$

式6就是计算循环层权重矩阵W的梯度的公式。

③梯度爆炸与梯度消失

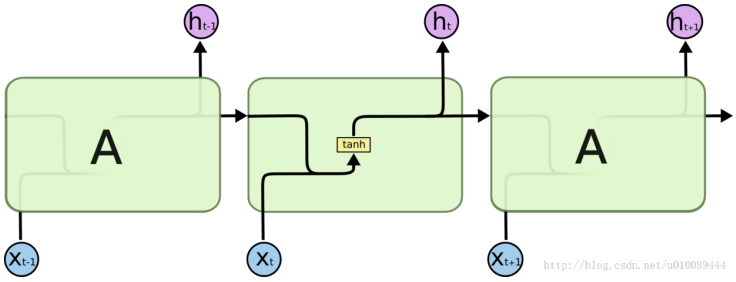
式3中令 $\gamma = ||\text{Wdiag}[f'(\text{net}_i)]||$ ，如果 $\gamma > 1$ ，当$k-t+1 \to \infty$会造成梯度爆炸问题；相反，如果 $\gamma < 1$ ，当$k-t+1 \to \infty$时会出现和深度前馈神经网络类似的梯度消失问题。换而言之，导数的链式法则导致了连乘的形式，从而造成梯度消失与梯度爆炸。而LSTM能避免RNN的梯度消失问题，其使用“累加”的形式计算状态，这种累加形式导致导数也是累加形式，因此避免了梯度消失。

3. 长短时记忆神经网络（LSTM）

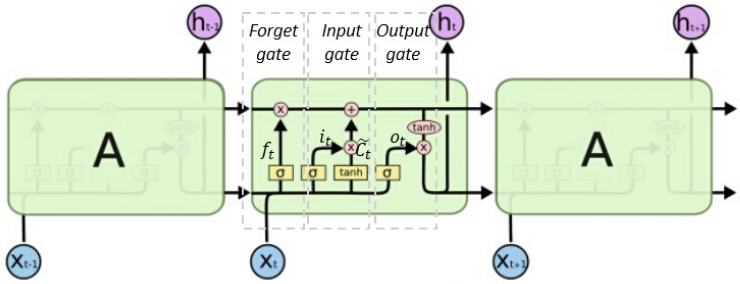
①LSTM 的核心思想

Long Short-Term Memory Neural Network—— 一般就叫做 LSTM ，是一种 RNN 特殊的类型，可以学习长期依赖信息。LSTM 由Hochreiter & Schmidhuber (1997)提出，并在近期被Alex Graves进行了改良和推广。在很多问题，LSTM 都取得相当巨大的成功，并得到了广泛的使用。LSTM 通过刻意的设计来避免长期依赖问题。记住长期的信息在实践中是 LSTM 的默认行为，而非需要付出很大代价才能获得的能力！

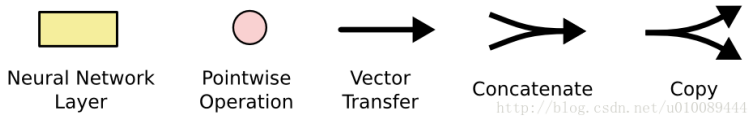
所有 RNN 都具有一种重复神经网络模块的链式的形式。在标准的 RNN 中，这个重复的模块只有一个非常简单的结构，例如一个 tanh 层。



LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于 单一神经网络层，这里是有四个，以一种非常特殊的方式进行交互。

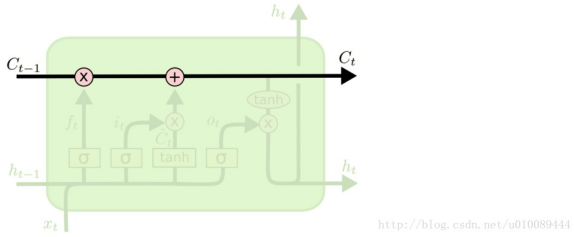


我们先来熟悉一下图中使用的各种元素的图标：在上面的图例中，每一条黑线传输着一整个向量，从一个节点的输出到其他节点的输入。粉色的圈代表 pointwise 的操作，诸如向量的和，而黄色的矩阵就是学习到的神经网络层。合在一起的线表示向量的连接，分开的线表示内容被复制，然后分发到不同的位置。

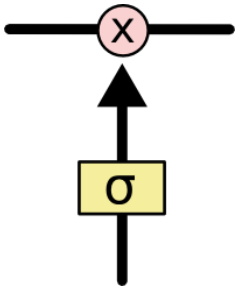


LSTM 的关键就是细胞状态，水平线在图上方贯穿运行。

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会很容易。



LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到细胞状态的能力。门是一种让信息选择式通过的方法。他们包含一个 sigmoid 神经网络层和一个 pointwise 乘法操作。



Sigmoid 层输出 0 到 1 之间的数值，描述每个部分有多少量可以通过。0 代表“不许任何量通过”，1 就指“允许任意量通过”！

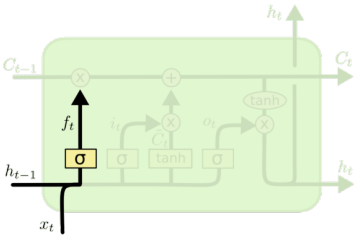
LSTM 拥有三个门（输入门，遗忘门，输出门），来保护和控制细胞状态。

②逐步理解 LSTM

在我们 LSTM 中的第一步是决定我们会从细胞状态中丢弃什么信息。这个决定通过遗忘层完成。该门会读取 h_{t-1}和 x_t，输出一个在 0 到 1 之间的数值给每个在细胞状态 C_{t-1} 中的数字。1 表示“完全保留”，0 表

示“完全舍弃”。

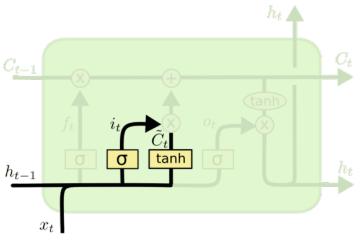
让我们回到语言模型的例子中来基于已经看到的预测下一个词。在这个问题中，细胞状态可能包含当前主语的性别，因此正确的代词可以被选择出来。当我们看到新的主语，我们希望忘记旧的主语。



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

下一步是确定什么样的新信息被存放在细胞状态中。这里包含两个部分。第一，sigmoid 层称 “输入门层” 决定什么值我们将要更新。然后，一个 tanh 层创建一个新的候选值向量，$\tilde{C}_t$，会被加入到状态中。下一步，我们会讲这两个信息来产生对状态的更新。

在我们语言模型的例子中，我们希望增加新的主语的性别到细胞状态中，来替代旧的需要忘记的主语。

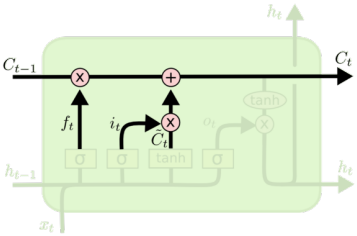


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

现在是更新旧细胞状态的时间了，C_{t-1} 更新为 C_t。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。

我们把旧状态与 f_t 相乘，丢弃掉我们确定需要丢弃的信息。接着加上 $i_t * \tilde{C}_t$。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。

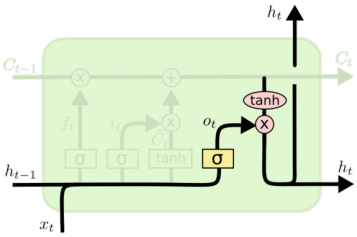
在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的性别信息并添加新的信息的地方。



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

最终，我们需要确定输出什么值。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

在语言模型的例子中，因为他就看到了一个 代词，可能需要输出与一个 动词 相关的信息。例如，可能输出是否代词是单数还是负数，这样如果是动词的话，我们也知道动词需要进行的词形变化。



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

3. 参考资料

- https://zhuanlan.zhihu.com/p/27485750
- http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- http://mt.sohu.com/20161110/n472772642.shtml
- http://www.jianshu.com/p/9dc9f41f0b29