RNN、LSTM及其变种

Li Junli 李军利 / June 2nd 2019

1. RNN

循环神经网络（RNN: Recurrent Neural Network）源自于1982年由 Saratha Sathasivam提出的霍普菲尔德网络。霍普菲尔德网络由于实现困难，在其提出时并没有被合适地应用。该网络结构也于1986年后被全连接神经网络以及一些传统的机器学习算法（ML: Machine Learning）所取代。然而，传统的ML算法非常依赖人工提取的特征，使得基于传统ML算法的图像识别、语音识别以及自然语言处理等问题存在**特征提取**的瓶颈。而基于全连接神经网络也存在**参数太多**、**无法利用数据中时间序列信息**等问题。随着更加有效的 RNN 结构被提出，RNN挖掘**时序信息**以及**语义信息**的深度表达能力被充分利用，并在语音识别、语言模型、机器翻译以及时序分析等方面实现了突破。

RNN的主要用途是处理和预测**序列数据**。百度百科对时间序列的解释是：时间序列数据是指在不同时间点上收集到的数据，这类数据反映了某一事物、现象等随时间的变化状态或程度。当然除了时间序列还有文字序列等，但总归序列数据有一个特点——后面的数据跟前面的数据有关系。

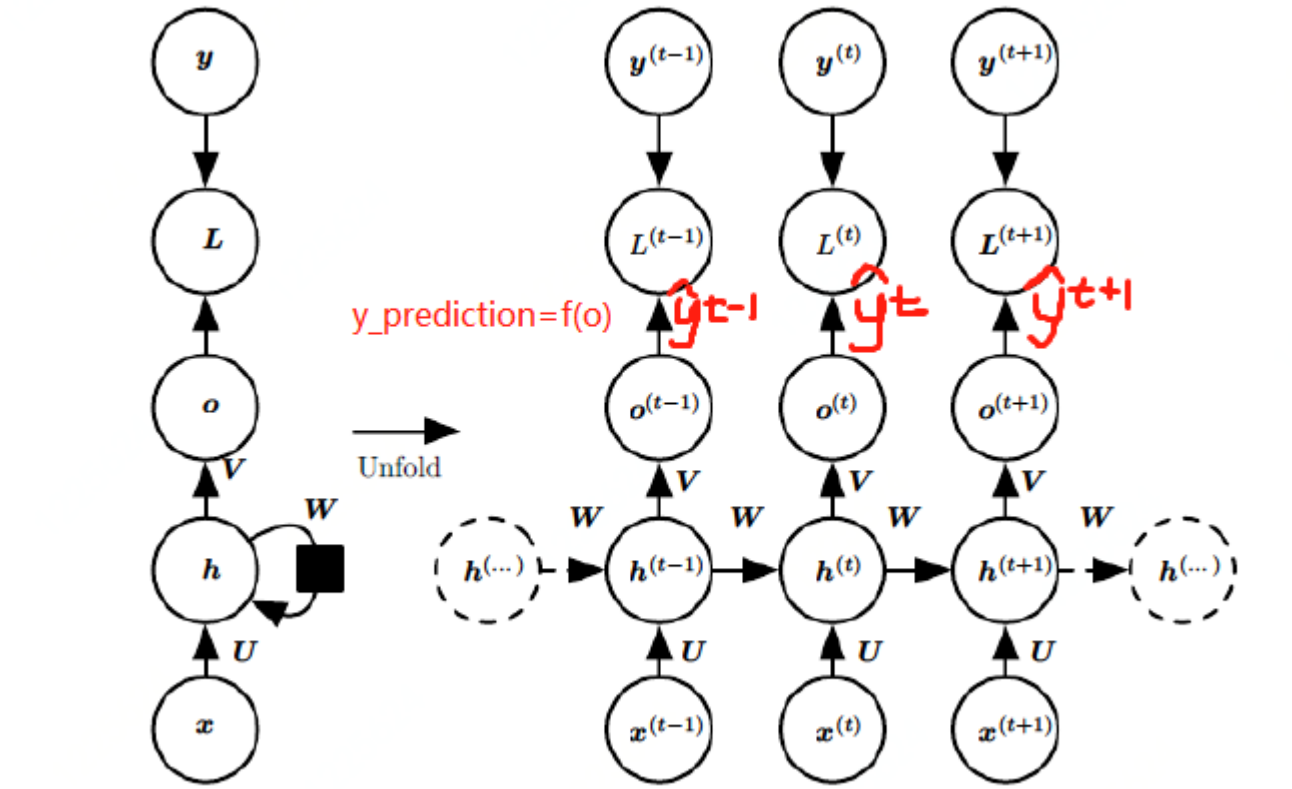
全连接网络或CNN的层与层之间是全连接或者部分连接的，但是每层之间的神经元是没有连接的（即假设各个数据之间是**独立的**）。这种结构不善于处理序列化的问题。比如要预测句子中的下一个单词是什么，这往往与前面的单词有很大的关联，因为句子里面的单词并不是独立的。比如当前word是“很”，前一个word是“天空”，那么下一个word很大概率是“蓝”。

从网络结构来看，RNN会**记忆**之前的信息，并利用之前的信息影响后面结点的输出。也就是说，RNN的隐藏层之间的结点是有连接的，隐藏层的输入不仅包括输入层的输出，还包括该隐藏层上一时刻的输出。

1.1 经典RNN结构

RNN有很多变种，首先介绍经典的RNN模型结构

经典RNN结构图



上图描述了在序列索引号 t 附近 RNN 的模型结构，我觉得加上预测值 y_prediction比较好理解，因此就在原图画蛇添足了一把。下面详细解释图中的结构，我比较喜欢用下标，因此用 $\mathbf{x_t}$ 表示 图中的 $\mathbf{x_t}$：

变量解释
$\mathbf{x_t}$ 代表在序列索引号**t**时训练样本的输入，$\mathbf{x_{t-1}}$和$\mathbf{x_{t+1}}$代表在序列索引号**t−1**和**t+1**时训练样本的输入
$\mathbf{h_t}$ 代表在序列索引号**t**时模型的隐藏状态，$\mathbf{h_t}$由$\mathbf{x_t}$和$\mathbf{h_{t-1}}$共同决定，很多教程把这个隐藏层写成 $\mathbf{s_t}$
$\mathbf{o_t}$ 代表在序列索引号**t**时模型的输出，$\mathbf{o_t}$只由模型当前的隐藏状态$\mathbf{h_t}$决定
$\mathbf{y\_prediction_t}$ 代表预测值，由$\mathbf{f(o_t)}$ 得到，**f**通常是**softmax**函数
$\mathbf{y_t}$ 代表在序列索引号*t*时训练样本序列的真实输出
$\mathbf{L_t}$ 代表在序列索引号**t**时模型的损失函数，由所有的$\mathbf{y_prediction_t}$ 和 $\mathbf{y_t}$ 计算得到
U、**W**、**V** 这三个矩阵是模型的线性关系参数，与CNN类似，这些参数在整个RNN网络中是共享的，也正因为是共享了，它体现了RNN的模型的“循环反馈”的思想，并且大大降低了计算量

假设x_{t-1}，x_t，x_{t+1} 对应输入 "我"，"是"，"中国"，那么y_{t-1}，y_t就应该对应 "是" ，"中国"这两个词，预测下一个词最有可能是什么？就是$y_prediction_{t+1}$应该是”人”的概率比较大。

1.2 前向传播

对于任意一个序列索引号t，有：

$$h_t = f(U * x_t + W * h_{t-1} + b)$$

对于序列的第一个时刻**t_0**，没有上一个隐层状态，则给予一个初始化的**h_t**。

f()是激活函数，做一个非线性映射，来过滤信息，处理非线性问题，RNN中一般是为**tanh**，也可以是Relu等，但是Sigmoid不太适合，容易**梯度消失**，这个后面再说。

h_t 除了利用当前信息 **x_t** 外，还捕捉了之前时间点上的信息，不过，美中不足的是，**h_t**并不能捕捉之前所有时间点的信息，这个缺点暂时也按下不表，之后再说。

b是线性关系的偏置项。

得到**h_t** 后，就可以计算 t 时刻的输出**o_t** 了

$$o_t = V * h_t + c$$

最终的预测输出为

$$\hat{y}_t = f(o_t)$$

由于RNN通常是分类模型，所以上面这个激活函数一般是softmax。

1.3 反向传播(BPTT)

RNN的反向传播就是利用损失函数（一般是**交叉熵损失**函数）对3个共享参数U、V、W求偏导，然后梯度下降更新各个权重。

各个权重的更新的递归公式：

$$V(t+1) = V(t) + \alpha \cdot \bigtriangledown V$$

$$U(t+1) = U(t) + \alpha \cdot \bigtriangledown U$$

$$W(t+1) = W(t) + \alpha \cdot \bigtriangledown W$$

一次训练任务的损失函数是每一时刻损失值的累加：

$$L = \frac{1}{t}\sum_{t=0}^{t} L_t = \frac{1}{t}\sum_{t=0}^{t} f(y_t, \hat{y}_t)$$

假设序列只有三段，t_1，t_2，t_3，对 t_3 时刻的U、V、W求偏导

对V求偏导，有：

$$\frac{\partial L_3}{\partial V} = \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial V}$$

对U、W求偏导，有：

$$\frac{\partial L_3}{\partial U} = \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial U} + \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial U} + \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial U}$$

$$\frac{\partial L_3}{\partial W}=\frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial W}+\frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W}+\frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial o_3}\frac{\partial o_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W}$$

可以看出对于 **W** 求偏导并没有长期依赖，但是对于 **U**、**V** 求偏导，会随着序列产生长期依赖。因为 h_t 随着序列向前传播，而 h_t 又是 **U** 、**V** 的函数。

根据上述求偏导的过程，我们可以得出任意时刻对 **U** 或者 **V** 求偏导的公式，：

$$\frac{\partial L_t}{\partial U}=\sum_{k=0}^{t}\frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}(\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}})\frac{\partial h_k}{\partial U}$$

任意时刻对**W**求偏导的公式同上。若加上双曲正切的激活函数，

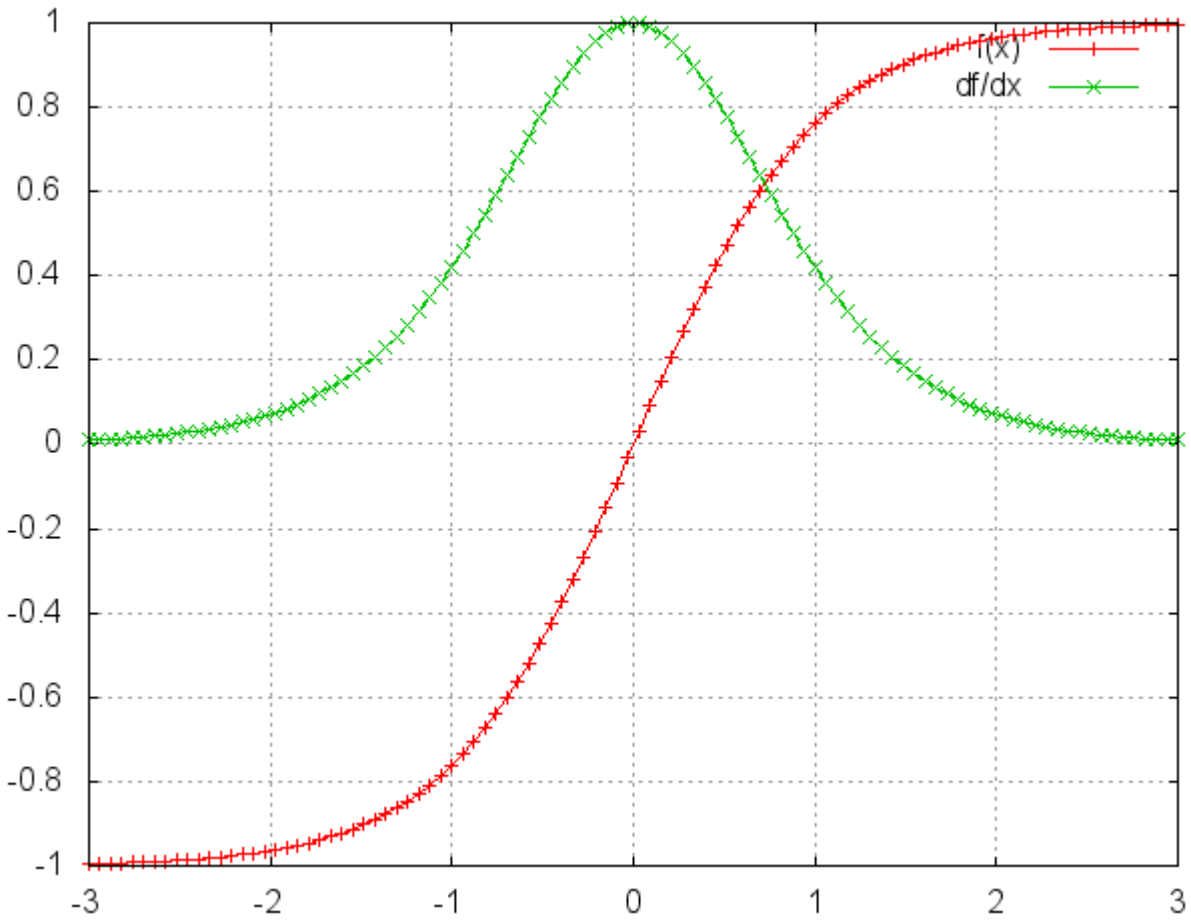
$$h_j=tanh(U*x_t+W*h_{j-1}+b)$$

则对于 W

$$\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}=\prod_{j=k+1}^{t}tanh'W$$

$$tabh(x)=\frac{sinh(x)}{cosh(x)}=\frac{e^x-e^{-x}}{e^x+e^{-x}}$$

tanh和它的导数



训练过程大部分情况下tanh的导数是小于1的，如果 **W** 的各个值也大于0，小于1，当 t 很大时

$$\prod_{j=k+1}^{t} tanh'W \to 0, \text{ 就像 } 0.01^{50} \to 0 \text{一样}$$

这就是上面提过的**梯度消失**的原因，梯度消失，参数也就不更新了；同理碰巧当 W 很大时，会出现 **梯度爆炸** 的情况。 **LSTM**、**GRU** 等变种 RNN 对这个问题有了优化。

下面也是两种常用的技术：

1、对于隐层态的权重矩阵whh使用单位矩阵进行初始化而不是随机初始化；

2、使用RELU激活函数而不是Sigmoid激活函数，因为RELU函数的导数为0或者1，在反向传播的过程中不会衰减至0。

当

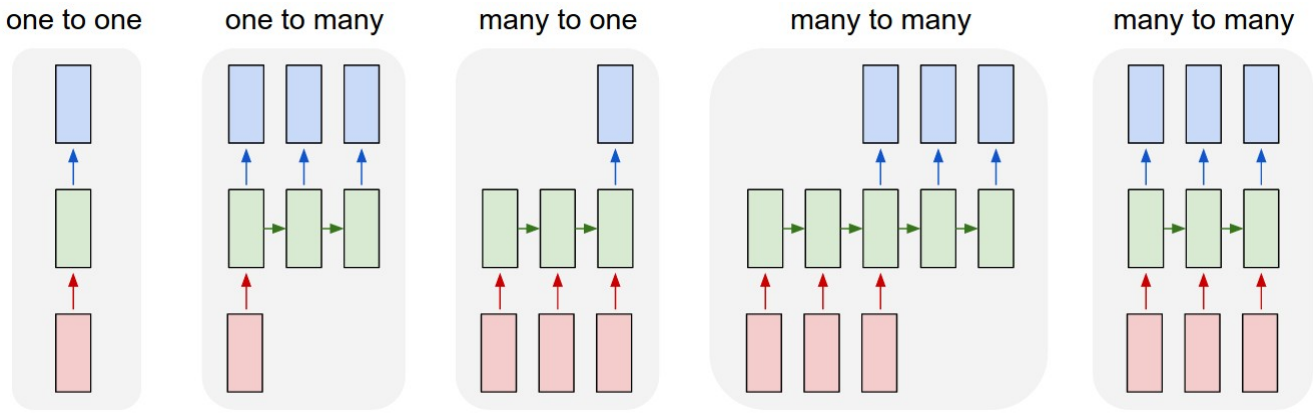
$$\frac{\partial h_j}{\partial h_{j-1}} \approx 1 \text{ 或者 } \frac{\partial h_j}{\partial h_{j-1}} \approx 0$$

时

可以解决**梯度消失**或者**梯度爆炸**的问题。

1.4 RNN 的输入输出

RNN允许人们对向量序列进行操作（无论是输入还是输出）



上图 红、绿、蓝矩形分别代表输入、隐藏层、输出，箭头代表向量flow以及特定函数（矩阵乘法等等）

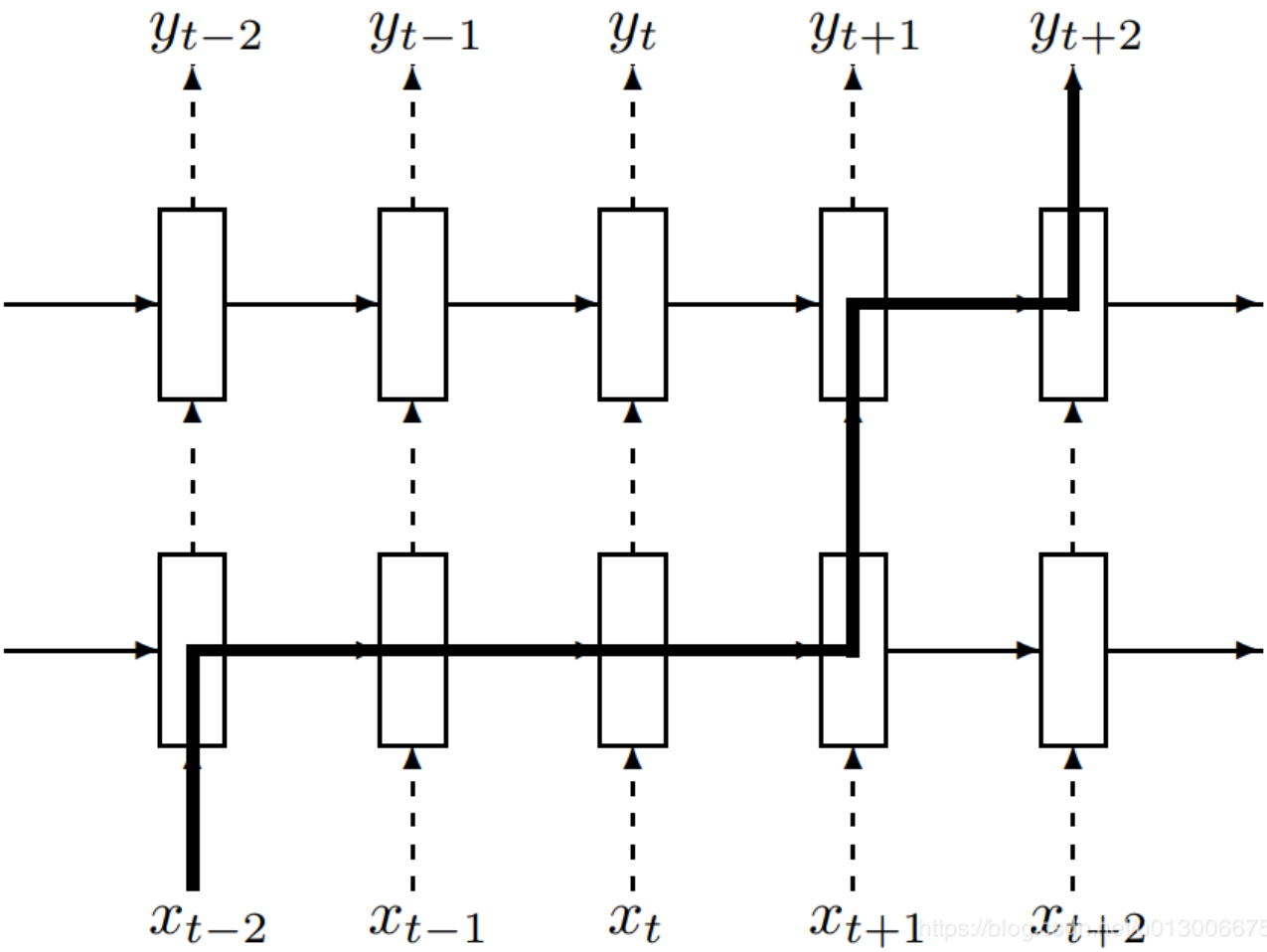
上图中的5个例子从左到右分别是：

1. 没有使用RNN，从固定大小的输入得到固定大小输出（比如图像分类）
2. 序列输出（比如图片描述，输入一张图片输出一段文字序列）
3. 序列输入（比如情感分析，输入一段文字然后将它分类成积极或者消极情感）
4. 序列输入和序列输出（比如机器翻译）
5. 同步序列输入输出（比如视频分类，对视频中每一帧打标签）

1.5 RNN的dropout

dropout 可以让神经网络的模型更加健壮，类似**CNN**中只在最后的全连接层使用**dropout**，RNN也只在不同层的循环体之间使用。就是说，从时刻 t-1 传递到 时刻 t 时，RNN不会进行 状态的 dropout，而在同一个时刻 t ，不同层的循环体 之间会使用 dropout （等等，上面只有一个循环体啊，多个循环体是deep RNN，下一节马上介绍）。

RNN的dropout



如图所示，**dropout**要设置在网络的非循环部分，否则信息将会因循环步的进行而逐渐丢失。比如我们将dropout设置在隐状态上(上图每一个实线位置)，那么每经过一次循环，剩余的信息就被做一次dropout，如果是长序列，那么循环到最后，前面的信息早就丢失殆尽了；反之，我们把dropout设置在输入神经元上(上图虚线位置)，，那么因dropout造成的信息损失则与循环的次数无关，而仅仅与网络的层数相关。

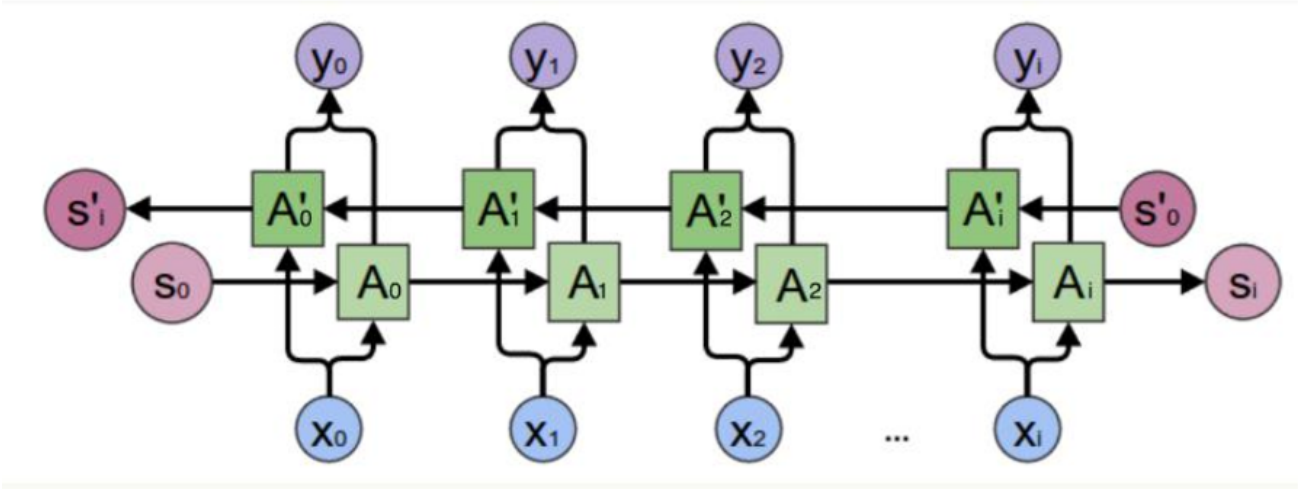
TensorFlow 中使用 **tf.nn.rnn_cell.DropoutWrapper** 类可以很容易地实现dropout功能，如果你是个tfboy，肯定感觉很亲切。

2. 深层双向RNN

2.1 双向RNN

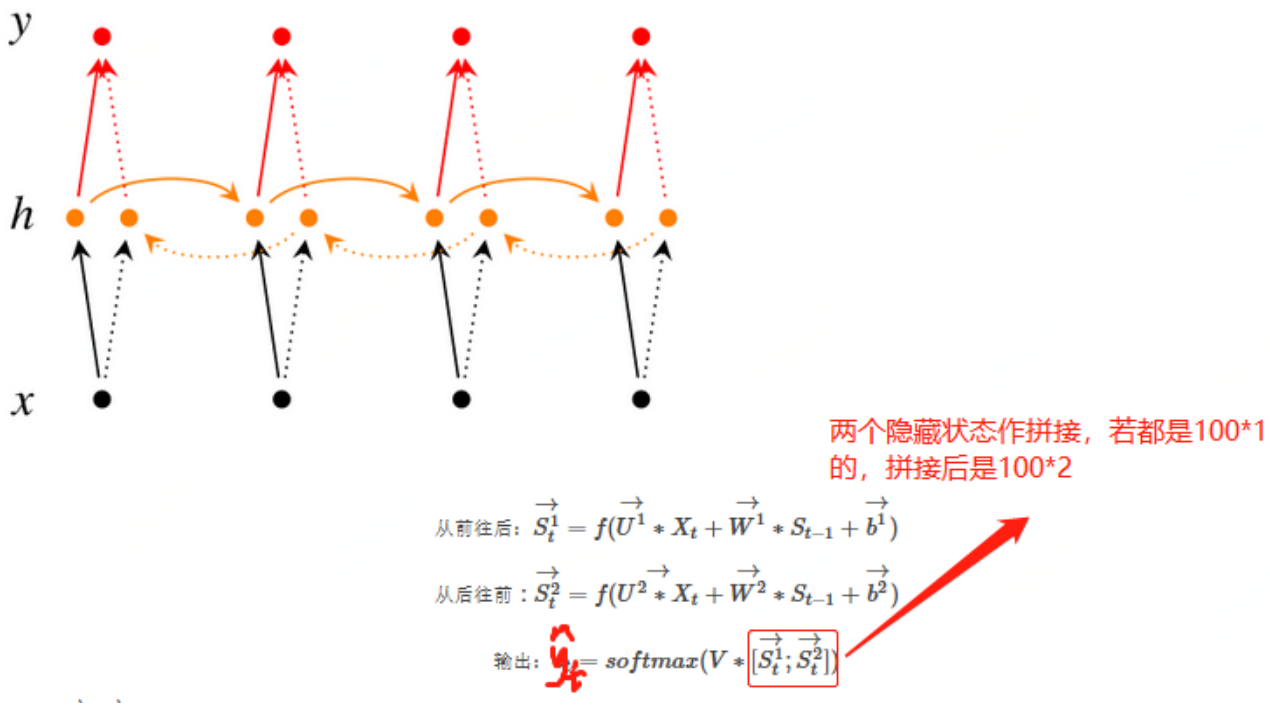
经典的循环神经网络，状态的传输是从前往后的，然而，在有些问题中，当前时刻的输出不仅和之前的状态有关系，也和之后的状态相关，这时候就需要**bidirectional RNN**（双向循环神经网络）解决这类问题。例如预测一个语句中缺失的单词 不仅需要根据前文来判断， 还需要结合后面的内容，这时 双向RNN就可以发挥它的作用。

双向RNN是由两个RNN上下叠加在一起组成的，输出由这两个RNN共同决定，如下图



在每一个时刻 t，输入会同时提供给这两个方向相反的循环神经网络 S_0 和 S'_0 是隐藏状态的初始化值（前文用的是符号h_0 表示，主要是个人觉得h 正好是 hidden 的首字母，比较好记，哈哈！）而输出则是由两个单向循环网络共同决定。

前向传播过程与单向RNN类似，至少多了一个方向而已，如下图所示：



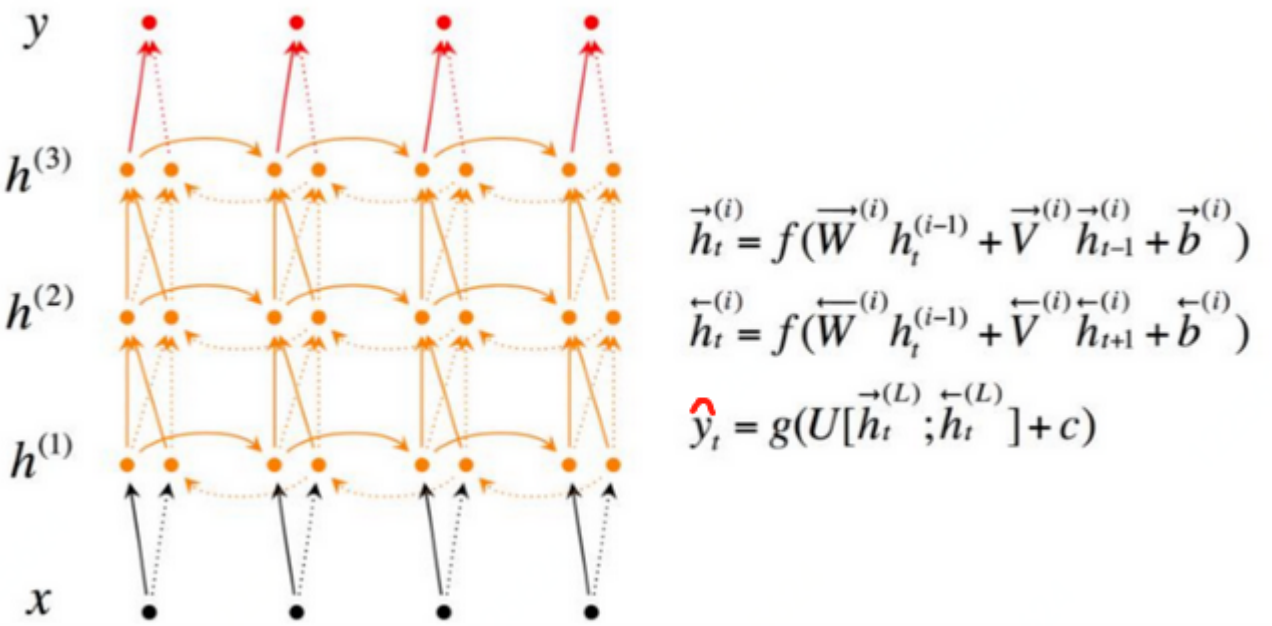
双向RNN需要的内存是单向RNN的两倍，因为在同一时间点，双向RNN需要保存两个方向上的权重参数，在分类的时候，需要同时输入两个隐藏层输出的信息。

Tensorflow 中实现双向RNN 的API是：bidirectional_dynamic_rnn； 其本质主要是做了两次reverse：

```
第一次reverse:将输入序列进行reverse,然后送入dynamic_rnn做一次运算
第二次reverse:将上面dynamic_rnn返回的outputs进行reverse,保证正向和反向输出的time是对上的
```

2.2 深层RNN

深层循环神经网络（**deep RNN**）是RNN的另一个变种。为了增强模型的表达能力，可以将每一个时刻的**循环体**重复多次，如下图：



深层神经网络在每一个时刻将 循环体复制了 多次，同一层的**参数 共享**，但是不同层的参数 可以**不同**。为更好地支持 **deep RNN** ， TensorFlow中提供了 **MultiRNNCell** 类来实现 deep RNN 的前向传播过程。

注：由于图片来源不同，所以符号表示会有不同

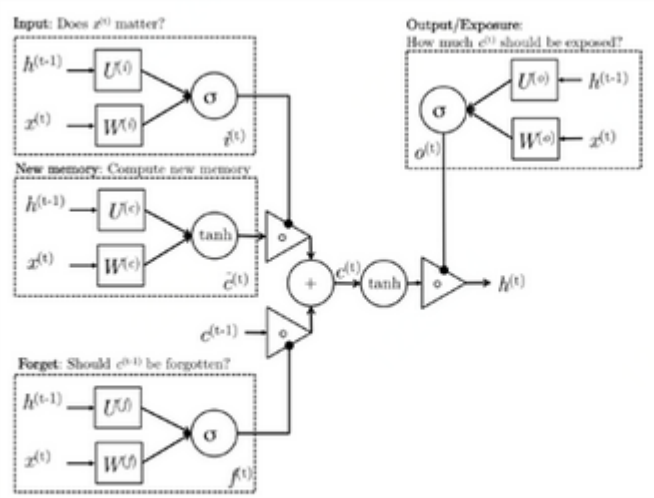
3. LSTM

因为RNN的信息只能传递给相邻的后继者（从循环展开之后的表示来看），因此当输出与其相关的输入信息相隔较近的时候，普通的RNN是可以胜任的。而当这个间隔很长的时候，虽然理论上RNN是可以处理这种**长期依赖 (Long Dependencies)** 的问题，但是实践中并没有成功。Bengio, et al. (1994)等人对该问题进行了深入的研究，他们发现了使训练 RNN 变得非常困难的根本原因（梯度消失/梯度爆炸）。因此，Hochreiter & Schmidhuber (1997) 提出了 Long Short Term Memory 网络 (LSTM，长短时记忆网络)，LSTM解决了RNN的梯度消失问题，可以处理长序列数据，成为流行的RNN变体，后面又被Alex Graves 进行了改良和推广。

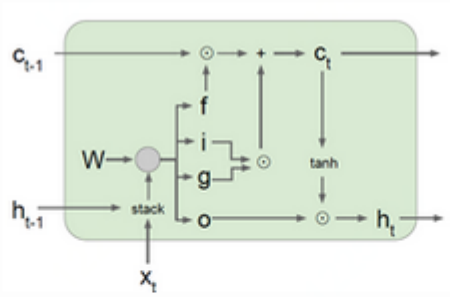
3.1 LSTM 前向传播综述

LSTM 表示图

一种表示



另一种表示



前向传播

$$
\begin{aligned}
i_t &= \sigma(W^{(i)}x_t + U^{(i)}h_{t-1}) && \text{(Input gate)} \\
f_t &= \sigma(W^{(f)}x_t + U^{(f)}h_{t-1}) && \text{(Forget gate)} \\
o_t &= \sigma(W^{(o)}x_t + U^{(o)}h_{t-1}) && \text{(Output/Exposure gate)} \\
\tilde{c}_t &= \tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) && \text{(New memory cell)} \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t && \text{(Final memory cell)} \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

1．产生新的记忆：使用t时刻的输入和上一时刻的隐藏态来计算新的记忆

2．输入门：在产生新的记忆时，我们发现网络并不会去检查新的输入对于新时刻的记忆有多重要，而输入门的意义就在这里，它将新的输入和上一时刻的隐藏态作为输入来计算i_t，i_t被用来控制新产生的记忆对新时刻下最终记忆的贡献程度

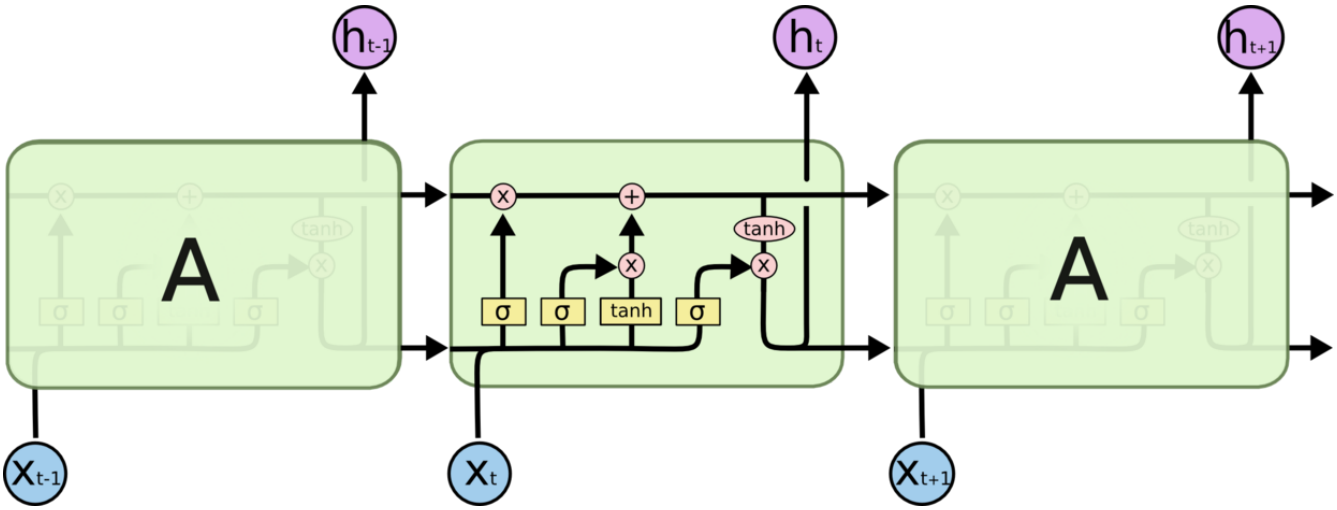
3．遗忘门：遗忘门的作用与输入门类似，但是遗忘门不是用来检查新时刻的输入，而是用来控制上一时刻的最终记忆对于新时刻的最终记忆状态的贡献程度

4．最终记忆：最终记忆的产生依赖于输入门和遗忘门的输出以及上一时刻的最终记忆和新时刻产生的新记忆，它是将上一时刻的最终记忆和新时刻产生的新记忆综合后的结果。这里遗忘门的输出f_t和上一时刻的记忆状态C_*{t-1}是逐元素相乘，而不是矩阵相乘，输入门i_t与新产生的记忆 之间也是逐元素相乘的*

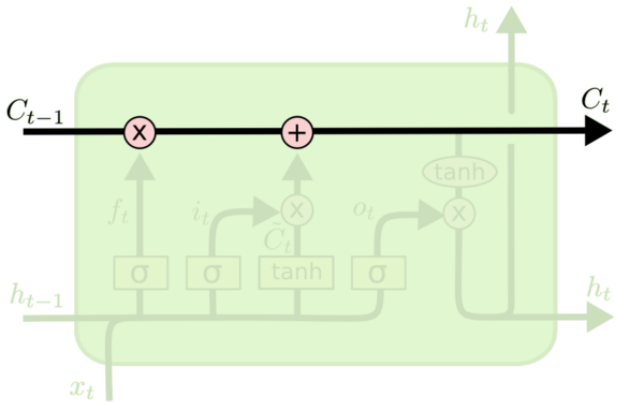
5．输出门：这个门(在GRU里不存在)的作用是将最终记忆与隐藏态分离开来；最终记忆里包含了大量的不需要在隐藏态里保存的信息，因此输出门的作用是决定最终记忆里哪些部分需要被暴露在隐藏态中。

3.2 LSTM 图分解

LSTM的网络结构大体上和传统RNN比较相似，但是重复结构A里面的东西要复杂一些，我们就一步一步来看。



首先，LSTM中最重要的核心思想就是所谓的cell state，也就是贯穿每个重复结构的上面这条flow

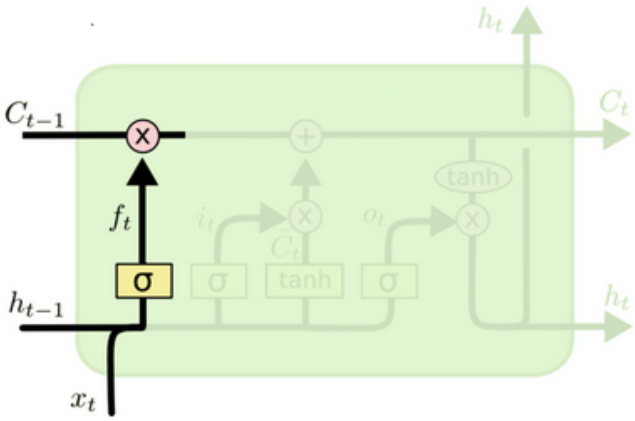


这条**flow**其实就承载着之前所有状态的信息，每当flow流经一个重复结构A的时候，都会有相应的操作来决定**舍弃哪些旧的信息**以及**添加哪些新的信息**。对**cell state**的信息增减进行控制的结构称为**门 (gates)**。一个LSTM block中有三个这样的门，分别是**遗忘门 (forget gate)**、**输入门 (input gate)**、**输出门 (output gate)**。

遗忘门 (forget gate):

遗忘门 决定了前一时刻中memory中的是否会被记住，当遗忘门打开时，前一刻的记忆会被保留，当遗忘门关闭时，前一刻的记忆就会被清空。

其通过输入 [上一状态的输出$\mathbf{h_{t-1}}$ 和当前状态输入信息 $\mathbf{x_t}$] 到一个**Sigmoid** 中，产生一个介于0到1之间的数值，与cell state相乘之后来决定舍弃（保留）多少前一时刻中的memory。0 表示“完全舍弃”，1 表示“完全保留”。



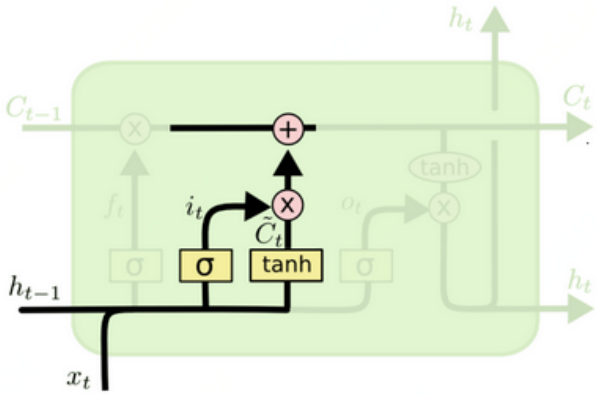
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

遗忘门 在连接 h_{t-1}和 x_t 之后会乘以一个权值 W_f 并加上偏置 b_f，这就是网络需要学习的参数。若hidden state的大小（也就是size of hidden layer of neurons）为 h_{size}，那么 W_f 的大小就为 h_{size} * h_{size}。h_size的数值是人工设定的。

输入门 (input gate):

输入门 决定当前的输入有多少被保留下来，因为在序列输入中，并不是每个时刻的输入的信息都是同等重要的，当输入完全没有用时，输入门关闭，也就是此时刻的输入信息被丢弃了。

与**遗忘门**相同，他通过输入 上一状态的输出h_{t-1} 和当前状态输入信息 x_t 到一个**Sigmoid** 中，产生一个介于0到1之间的数值，来确定我们需要保留多少的新信息。同时，一个**tanh**层会通过 h_{t-1} 和 x_t 来得到一个将要加入到cell state中的 “候选新信息” 。将刚才得到的数值 i_t 与“候选新信息“ $\tilde{C}_t$ 相乘得到我们真正要加入到cell state中的更新信息。



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

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

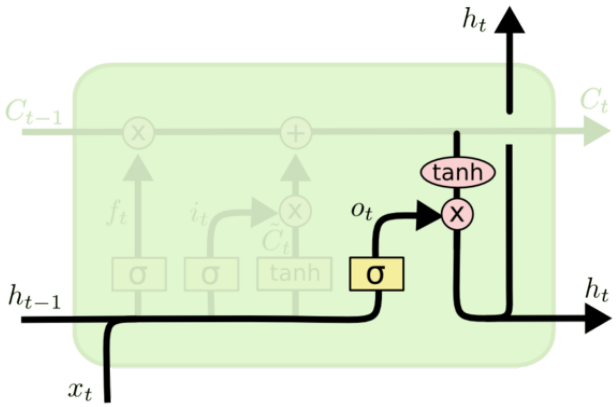
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

输入门（一个Sigmoid函数层）与tanh层，两个神经网络层都会和之前遗忘门一样学习各自的参数

输出门 (output gate):

输出门决定当前memroy的信息有多少会被立即输出，输出门打开时，会被全部输出，当输出门关闭时，当前memory中的信息不会被输出。

与之前类似，会先有一个Sigmoid函数产生一个介于0到1之间的数值 $\mathbf{o_t}$ 来确定我们需要输出多少cell state中的信息。cell state的信息在与 $\mathbf{o_t}$ 相乘时首先会经过一个tanh层进行"激活"（非线性变换）。得到的就是这个LSTM block的输出信息 $\mathbf{h_t}$ 。



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

输出门 (output gate)，同样有自己的权值参数需要学习。

```
注意：一个LSTM block中可能有不止一个cell state（也就是最上面的Ct会有不止一条），它们共享输入、输出和遗忘门：
S memory cells sharing the same input, output and forget gates form a structure called "a memory cell block of size S". This means that each cell might hold a different value in its memory, but the memory within the block is written to, read from and erased all at once. Memory cell blocks facilitate information storage as with conventional neural nets, it is not so easy to code a distributed input within a single cell. Since each memory cell block has as many gate units as a single memory cell (namely two), the block architecture can be even slightly more efficient. A memory cell block of size 1 is just a simple memory cell.
```

LSTM能够解决梯度弥散和梯度爆炸问题的原因

```
1. 遗忘门与上一时刻的记忆之间是矩阵元素相乘，而不是矩阵相乘；
2. 遗忘门在不同的时间步长会取不同的值
3. 遗忘门的激活函数是sigmoid函数，值域范围为(0,1)
```

与经典RNN一样，也可以构造深层双向的**LSTM**，不多说。

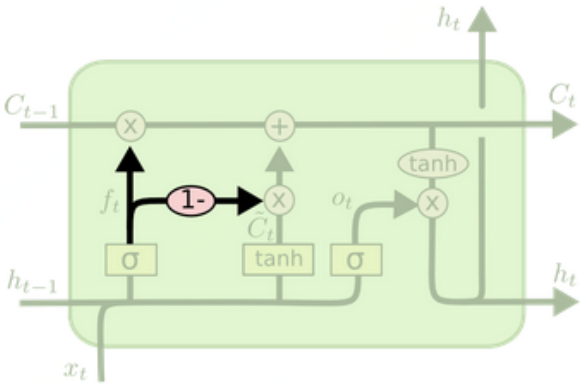
3. LSTM变种（GRU、CIFG）

很多人关注 LSTM各种变体的对比、以及LSTM内部的各个模块对于训练效果到底有什么样的影响。Jozefowicz, et al. (2015) 在超过 1 万种 RNN 架构上进行了测试，发现一些架构在某些任务上也取得了比 LSTM 更好的结果。

Greff, et al. (2015) 探讨了基于Vanilla LSTM (Graves & Schmidhube (2005))之上的8个变体，并比较了它们之间的性能差异，包括：

1. 没有输入门（No Input Gate, NIG）
2. 没有遗忘门（No Forget Gate, NFG）
3. 没有输出门（No Output Gate, NOG）
4. 没有输入激活函数（No Input Activation Function, NIAF）（也就是没有输入门对应的tanh层）
5. 没有输出激活函数（No Output Activation Function, NOAF）（也就是没有输出门对应的tanh层）
6. 没有"peephole connection"（No Peepholes, NP）
7. 遗忘门与输出门结合（Coupled Input and Forget Gate, CIFG）
8. Full Gate Recurrence（FGR）
   The FGR variant adds recurrent connections between all the gates (nine additional recurrent weight matrices).

CIFG



遗忘门与输出门结合
(Coupled Input and Forget Gate, CIFG)

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

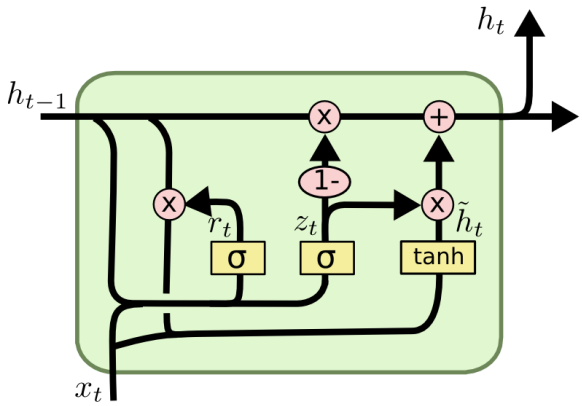
文章使用了将这8种LSTM变体与基本的Vanilla LSTM在TIMIT语音识别、手写字符识别、复调音乐建模三个应用中的表现情况进行了比较，得出了几个有趣的结论：

1. Vanilla LSTM在所有的应用中都有良好的表现，其他8个变体并没有什么性能提升
2. 将遗忘门与输出门结合（Coupled Input and Forget Gate, CIFG)以及没有"peephole connection"（No Peepholes, NP)简化了LSTM的结构，而且并不会对结果产生太大影响
3. 遗忘门和输出门是LSTM结构最重要的两个部分，其中遗忘门对LSTM的性能影响十分关键，输出门 is necessary whenever the cell state is unbounded （用来限制输出结果的边界）；
4. 学习率和隐含层个数是LSTM最主要的调节参数，而动量因子被发现影响不大，高斯噪音的引入对TIMIT实验性能提升显著，而对于另外两个实验则会带来反效果；
5. 超参分析表明学习率与隐含层个数之间并没有什么关系，因此可以独立调参，另外，学习率可以先使用一个小的网络结构进行校准，这样可以节省很多时间。

GRU:

由 Cho, et al. (2014) 提出的Gated Recurrent Unit (GRU)是现在相当流行的一种LSTM变体。它首先将cell state和hidden state进行了合并，然后将原来的三个门替换为了**更新门 (update gate)**（图中的z_t）和**重置门 (reset gate)**（r_t）。重置门用来决定新的hidden state $\tilde{h}_t$中上一 hidden state h_{t-1}所占的比重，新的hidden state $\tilde{h}_t$ 由重置门"过滤"后的h_{t-1}和输入x_t 经过tanh层激活（非线性变换）后得到；更新门用来决定当前hidden state h_t 中新的hidden state $\tilde{h}_t$ 所占的比重，控制了最后输出的h_t 中 $\tilde{h}_t$和h_{t-1}所占的比例。

GRU



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

声明

本文写作过程中参考了（尤其是图片的引用）一些博客、书籍、课程和论文，在此对作者们表示感谢！

转载请标明来源 github地址

References

[1] 郑泽宇, 顾思宇.TensorFlow实战谷歌深度学习框架.中国工信出版集团, 电子工业出版社

[2] 梯度消失

[3] 循环神经网络原理通俗解释

[4] 赵卫东深度学习MOOC

[5] RNN与前向反向传播算法

[6] 几种常见的循环神经网络结构

[7] 从RNN到LSTM

[8] RNN & LSTM 网络结构及应用

[9] 一步步理解LSTM

[10] RNN变种

[11] Kyunghyun Cho, Bart van Merrienboer etc.Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, 2014.

[12] Klaus Greff, Rupesh K. Srivastava etc.LSTM: A Search Space Odyssey, 2015.