## 报告：基于Tensorflow手工实现简易LSTM算法

* **LSTM简述**

LSTM全名为长短期记忆（Long short-term memory），实际上是对循环神经网络（Recurrent Neural Network，RNN）的一种改进，旨在解决长序列训练过程中的梯度消失和梯度爆炸的问题。简而言之，LSTM在更长的序列中比普通RNN的变现更出色。Wikipedia上LSTM的方程为：

ft = sigmoid(Wfxt + Ufht-1 + bf)

it = sigmoid(Wixt + Uiht-1 + bi)

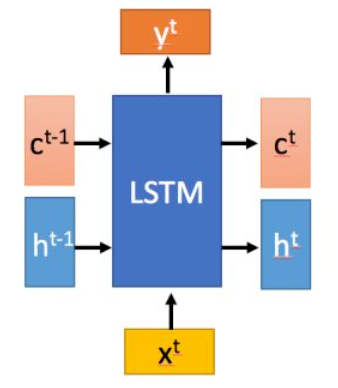
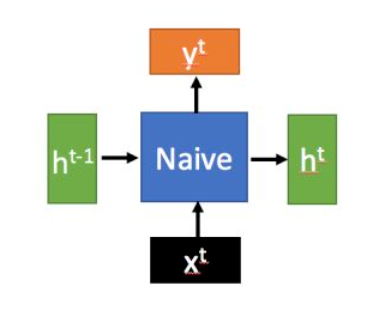
ot = sigmoid(Woxt + Uoht-1 + bo)

cht = tanh(Wcxt + Ucht-1 +bc)

ct = ft \* ct-1 + it \* cht

ht = ot \* tanh(ct)

其中，W、U和b代表参数阵，也就是模型要训练的东西；x为输入张量；h为隐状态（hidden state），也是模型结果； \* 代表矩阵对应元素相乘。ft为遗忘门，it为输入/更新门，ot为输出门。为了更好的理解LSTM为何在长序列中如此优秀，首先看下普通RNN（左）和LSTM（右）一个小单元的模型对比图：



相比于RNN只有一个传递状态ht，LSTM有两个传输状态，一个ct（cell state），和一个ht（hidden state）（RNN中的ht对于LSTM中的ct）。其中ct改变得很慢，通常输出的ct是上一个状态传过来的ct加上一些数值，而ht则在不同节点下往往会有很大的区别。

根据LSTM的计算公式我们可以看到，LSTM首先使用当前输入xt和上一个状态传下来的ht-1拼接训练得到四个状态。其中ft、it和ot是由拼接向量乘以权重矩阵之后再通过一个sigmoid激活函数转换为0到1之间的一个数，作为门控状态；而cht则是将结果通过一个tanh激活函数转换为-1到1之间的值，这里使用tanh是因为要将其作为输入数据而不是门控信号。LSTM通过这些计算公式主要实现了三个阶段的过程：

1. **遗忘阶段**

这个阶段主要是对上一个节点传进来的输入进行选择性忘记。简单说就是忘记不重要的，记住重要的。具体来说是通过计算得到的ft遗忘门控与上一个节点的输出ct-1做矩阵元素对应相乘来控制哪些需要保留，哪些需要忘记。

1. **选择记忆阶段**

这个阶段主要对输入xt进行选择性的记忆，重要的多记点，不重要的少记点。这个阶段的输入内容由cht表示，通过cht与输入/更新门控it矩阵元素对应相乘来进行控制。遗忘阶段与选择记忆阶段的结果相加即可得到传递给下一个状态的ct，也就是LSTM公式中的ct = ft \* ct-1 + it \* cht。可见ct此时已经是“遗忘了不重要的、重点记忆了重要的”的状态信息。

1. **输出阶段**

这个阶段将决定哪些将会被当成当前状态的输出。主要是通过输出门控ot(包含有本状态输入xt的信息)与来自上一个状态的信息ct矩阵元素对应相乘来进行控制，并且还对上一阶段得到的ct进行了放缩（通过一个tanh激活函数进行变化）。与普通RNN类似，最终输出（图中的yt）往往也是通过ht变化得到。

以上就是LSTM的内部结构。LSTM通过门控状态来控制传输状态，记住需要长时间记忆的，忘记不重要的信息；而不像普通的RNN那样只能够“呆萌”的使用一种记忆叠加方式。因此对很多需要“长期记忆”的任务来说尤其好用。

* **TensorFlow简述**

TensorFlow是一个开放源代码软件库，用于进行高性能的数值计算。用户可以借助其灵活的架构轻松的将计算工作部署到多种平台（CPU、GPU、TPU）和设备（桌面设备、服务器集群、移动设备、边缘设备等）。

TensorFlow采用数据流图（data flow graphs）来计算，所以我们首先得创建一个数据流流图，然后再将数据（以张量“tensor”的形式存在）放在数据流图中计算。在图中节点（Nodes）表示数学操作，边（edges）则表示在节点间相互联系的多维数据数组，即张量（tensor）。训练模型时tensor会不断的从数据流图中的一个节点flow到另一节点，这就是TensorFlow名字的由来。

在机器学习中，数值通常由4种类型构成：

（1）标量（scalar）：即一个数值，它是计算的最小单元，如“1”或“3.2”等。

（2）向量（vector）：由一些标量构成的一维数组，如[1, 3.2, 4.6]等。

（3）矩阵（matrix）：是由标量构成的二维数组。

（4）张量（tensor）：由多维数组构成的数据集合，可理解为高维矩阵。

TensorFlow中张量有多种，其中零阶张量为纯量或标量（scalar）；一阶张量为向量（vector）；二阶张量为矩阵（matrix）。以此类推，三阶张量即三维矩阵，等等。张量从流图的一端流动到另一端的计算过程，生动形象地描述了复杂数据结构在人工神经网中的流动、传输、分析和处理模式。

使用TensorFlow的优点主要表现在以下几个方面：

（1）TensorFlow有一个非常直观的构架，它有一个“张量流”。用户借助TensorBoard可以很容易地看到张量流动的每一个部分。

（2）TensorFlow可轻松地在CPU/GPU上部署，进行分布式计算。

（3）TensorFlow跨平台性高，灵活性强，不但可以在Linux、Mac和Windows系统下运行，甚至还可以在移动终端下工作。

* **Python+TensorFlow的简易LSTM实现**

TensorFlow中有封装的更好的LSTM算法，因为我刚看了Bidirectional LSTM-CRF Models for Sequence Tagging这篇论文，为了更好的理解LSTM的内部结构和原理就萌生了自己实现一个简单LSTM算法的想法，也为自己之后复现高级算法打打基础摸摸路数。期间我上网查阅了一些资料，看了些视频讲解，使用了tensorflow中的一些激活函数和张量处理函数，最终实现了一个简单的LSTM算法。我把这个算法封装成类，并用随机生成的4000\*5\*30（batch\_size\*sequence\_length\*input\_size）的输入张量x\_data去拟合一个随机生成的二分类任务y\_data，结果发现损失函数确实在降低。随机数去拟合随机任务我觉得挺神奇的，解释我认为是模型记住了随机数，然后产生了过拟合，不过LSTM算法应该是成功了，以下为代码运行截图：



以下为代码在git上的链接：

https://github.com/helldog-star/SimpleLSTM.git

* **优化提升**

这个简洁版跟官方的LSTM算法肯定是比不了的，向着官方LSTM算法的功能还有非常大的优化空间。比如如何实现双向的LSTM；再比如这个简单LSTM第一步的隐变量h-1这里是按照空（也就是全零）来开始计算的，如果第一部的输入是别的模型的结果要如何处理（诸如此类的不同情况貌似有很多）；甚至说每个unit里面的计算（每个计算单元的公式都涉及到了8次矩阵乘法（为了算出门控和传递信息）、3次矩阵对应元素相乘（为了算出状态信息）和多次的激活函数运算），当输入张量的数据很大时会有很大的计算量（应该也会很耗时），这个能不能优化？为此我找到个叫GRU的使用效果和LSTM相当但参数更少，可以用来构建大训练量的模型，之后去了解下！:)