从前馈到反馈：解析循环神经网络（RNN）及其tricks

原创 夕小瑶 夕小瑶的卖萌屋 2017-08-20

好像已经有两周没有更新啦。最后这几天都不敢打开订阅号后台了，怕一打开发现掉了几百个粉丝的话就难过死了T_T。然而小夕发现你们并没有离开，感动的差点哭出来，都感觉再不认真写一篇文章就太对不起大家的等待啦。

而这两周，经历的事情蛮多的。为了凑下一季的房租，接了个私活，要死要活的做完了QAQ。而且还发现了一个特别好的学习平台，闭关修炼了一周，改天跟你们分享一下～下面开始正文啦，不要太激动哦。

为什么需要反馈?

在上一篇《前馈网络与卷积网络》中，小夕讲解了前馈网络中的战斗机——卷积神经网络（CNN）更关注局部特征的提取，比如在句子级情感分类的任务中，往往找出句子中可以表现情感的情感词就能完成这个句子的情感极性分类，而情感词之前的词并不会对决策结果起到多大的影响。还是习惯性的举出栗子 ：

比如 “夕小瑶 今天 不开心 了。”

为了判断这个句子的情感极性，只需要让分类器能识别出“不开心”是个负极性的词，其他词是偏中性词就可以了。而“不开心”前面的中性词是“今天”还是“有时”，对“不开心”的情感极性并不会有多大的影响，也不会对整个句子的情感极性有多大影响。因此，当我们把该句子中的各个词条依次输入给模型去分类时，并不需要去“瞻前顾后”，因此使用一个关注局部的前馈神经网络往往表现更佳。而从最近三四年的论文来看，在句子级情感分类问题上，也确实是卷积网络比递归网络和循环网络更容易引领state-of-arts。

然而，还有一些任务的labels之间会有很强的相关性，比如命名实体识别（NER）任务，我们想要识别出文本中的地址时：

“据报道，在 2016年，有 一只 可爱的 小狗狗 在 北京市 海淀区 番茄猫咪 小区 失踪。”（什么鬼栗子）

这句话中的地址是“北京市 海淀区 番茄 猫咪 小区”，因此我们的目标是给这几个词条贴上“这是地址”的标签，给其他词条贴上“这不是地址”的标签。

显然，如果用CNN去做的话，很容易把“番茄”识别成非地址词，毕竟想要识别出来它的话，就要同时考虑“海淀区”“猫咪”“小区”这三个相邻的上下文词条，也就是说CNN的卷积滤波器的长度要最少达到4才有较大的可能性正确标注这个句子的labels。而对更长的地址，那代价就更大了，总不能无限增长滤波器的长度呐。所以**一个更靠谱的办法是让模型在当前位置的输出再反馈给模型，来帮助模型做下一位置的决策！**

这样的有反馈单元的模型在做当前位置的决策的时候，会同时考虑前面所有位置/时间点的情况（直接考虑上一时间点，又由于上一时间点也考虑了上上时间点，因此间接考虑了上一时间点前面所有的时间点），因此有反馈单元的模型在判断“番茄”是不是地址时，它通过前面积累的 “据报道，在2016年，有 一只 可爱的 小狗狗 在 北京市 海淀区” 发现下一个词 “番茄” 是非地址词的概率并不大（因为训练集中很少会出现主语+“在”+地址1+地址2+非地址名词+...的情况，倒是经常出现主语+“在”+地址1+地址2+地址3+...的情况），从而可以完成正确决策。

相比之下，在卷积网络中，由于它视野有限，不瞻前顾后，所以它更可能觉得“地址+非地址名词”非常正常，因为比

如“北京糖葫芦”，“美国 猫”也很常见嘛～进而导致了错误决策。这也是为什么在该任务中，有反馈单元的神经网络比前馈网络更容易成为state-of-art。

数学上如何描述反馈?

显然这样模型就不再是前馈的了，而是将模型的输出也作为输入来丢进模型。回顾一下，前馈网络的时候简单的前馈公式是这样的：

$$O=f(X * W+b)$$

其中W和b是模型的参数，X是当前的输入，f(·)是激活函数，*是矩阵乘法，O是当前的输出。即输出 等于 输入经过线性与非线性映射后的结果。

加上反馈单元后，公式就变成了：

$$O_t=f(X * W + O_{t-1} * V+ b)$$

其中W、V、b是模型的参数，下标t代表当前的序列位置／时间点，t-1代表上个位置/上个时间点，X是当前的输入，f(·)是激活函数，*是矩阵乘法，O是模型输出。

简单的加上反馈单元的这个模型，就叫做**循环神经网络（RNN,R=Recurrent）**。这也是后续LSTM、GRU等门限循环网络的基础模型。

RNN是浅层的还是深层的?

从前向过程来看，貌似是浅层的。比如1000层的前馈神经网络，在做“xxxxx”的命名实体识别这个前文例子的时候，是每一个词条都要跑一遍1000层的网络才能出这个词条的标注结果。而简单的循环网络来说，跑一层就会出一个标注结果。因此看起来是浅层的。然而，在计算序列的后几个位置的label的时候，显然也积累了前面所有位置的计算结果，因此这样看又是深层的。

从反向过程看，也就是从优化的角度来看，RNN是深层的。因为在进行每一次误差反向传播的时候，误差要从前馈网络的第1000层开始逐层传回第一层，误差在循环网络中也要从序列的末端输出一直传回到序列的首端。因此序列是1000长度的话，在RNN中误差就相当于传递了1000层。

这个争论也被公认为是争论。小夕就不参与讨论了。我们只关注这样会带来什么特殊的问题。

RNN的问题

首先，从浅层的角度去看RNN的前向过程，就可以认为它只有一层权重矩阵W（我们先不管V）。由此可见从深层的角度去看RNN的前向过程，就可以认为RNN是**各个层的权重矩阵相同**的深层网络。我们忽略V和激活函数，就可以近似的认为网络一共有T层（T等于序列的长度），那么第t层的输出就是连乘t次W，也就是W^t！

在《线性代数（二）》中，小夕讲过**矩阵可以用它的特征值矩阵和特征向量矩阵去近似**，即

$$W\approx Vdiag(\lambda)V^{-1}$$

所以W^t就可以展开成$(Vdiag(\lambda)V^{-1})(Vdiag(\lambda)V^{-1})...$，约掉中间的一堆$V^{-1}V$，就是

$W^t=Vdiag(\lambda)^tV^{-1}$

也就是说，**特征值矩阵中的每个特征值都会随着t的增大发生指数级变化！**所以某个特征值大于1时，就容易导致这个维度的值爆炸性增长；当特征值小于1时，会就会导致这个维度的值指数级速度衰减为0！

前向过程如此，误差反向传播的过程也必然近似为输出层的误差会乘以W^t来得到倒数第t层的梯度，然而由于刚才讲的各个维度不是指数级衰减就是指数级爆炸，很容易看出当你去更新RNN的靠前的层的时候（即离着输出层较远的那些层，或者说与序列末端离得远的位置），计算出的梯度要么大的吓人，要么小的忽略。小的忽略的值不会对W的更新有任何指导作用，大的吓人的值一下子就把W中的某些值弄的大的吓人了，害得前面的优化都白做了...因此这显然是不靠谱的。哦对了，这个现象叫做**梯度消失**和**梯度爆炸**。

解决方法呢?

一个很简单的想法是进行**梯度截断**，我们在优化RNN的参数的时候，给剃度值设置一个上限，免得发生爆炸摧毁我们之前积累的结果。但是这样显然就会导致模型难以再顾及很靠前的历史信息了（梯度都被大幅阉割或者自己消失了），因此理论上RNN可以保存任意长的历史信息来辅助当前时间点的决策，然而由于在优化时（训练RNN时），梯度无法准确合理的传到很靠前的时间点，因此RNN实际上只能记住并不长的序列信息（在NLP中，经验上认为序列大于30的时候，RNN基本记不住多于30的部分，而从多于10的位置开始就变得记性很差了），因此RNN相比前馈网络，可以记住的历史信息要长一些，但是无法记住长距离的信息（比如段落级的序列甚至篇章级的序列，用RNN就鸡肋了）。

使用梯度截断的trick可以减轻参数矩阵连乘带来的问题。同样，如前所述，由于前向过程也会对参数矩阵进行连乘，显然乘的次数多了，就会容易产生很大的值（包括很大的正值与很小的负值），而很大的正值或者很小的负值会在反向传播时丢给激活函数的导数（忘了的回去复习《BP算法过程》），因此：

对于sigmoid的导数$\sigma'(z) = \sigma(z)(1-\sigma(z))$（有疑问的自己推导一下）来说，显然无论输入的z是超大正值也好，超小负值也好，都会导致其导数接近0！因此直接导致梯度传到这里后就消失了！而ELU 的导数和ReLU的导数则不会有这个问题。因此在RNN中，更要**注意一般情况下不要使用sigmoid作为激活函数**！

好了。标准的RNN讲完了。我们发现RNN的想法很简单，直接引入了反馈单元来记忆历史信息。然而由于训练时的梯度爆炸与消失，导致其难以学习远距离历史信息（或者说与当前时间点的远距离依赖关系），因此要注意使用梯度截断的trick来处理优化过程，同时要避开sigmoid这类容易带来梯度饱和的激活函数。那么如何让循环神经网络去学习**长距离依赖关系**呢？期待小夕的长短时记忆网络（LSTM）吧！

文章已于修改