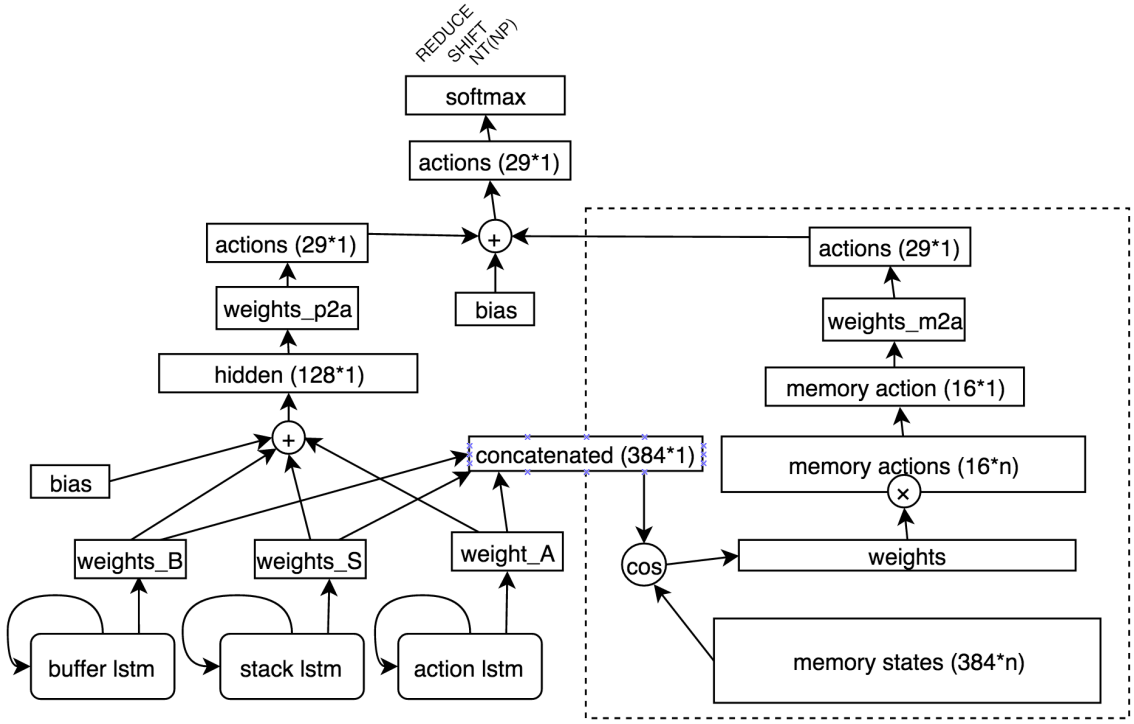
RNNG + Memory 实验报告

张诗悦

2016.11.1

尝试在 RNNG 中增加 memory，现在只考虑结构如图下图所示。虚线框内部分是增加的模型结构。



现在的实现比较简单，具体过程如下：

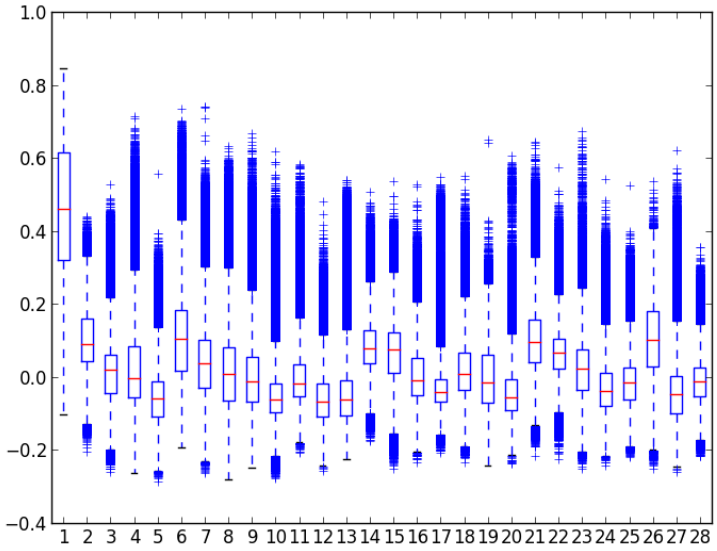
1. 取训练好的模型，重新输入训练集作为测试数据，获取每一步输出的 {stack, buffer, action}384 维隐状态，共有 2599716 个隐状态。
2. 按照隐状态对应的 29 种 action 分类，分布如下：

action	隐状态数
0,REDUCE	792794
1,SHIFT	1014128
2,NT(S)	100096
3,NT(PP)	95581
4,NT(NP)	350432
5,NT(PRN)	2420
6,NT(VP)	146319
7,NT(ADVP)	22447
8,NT(SBAR)	30532
9,NT(ADJP)	14607
10,NT(QP)	9444
11,NT(UCP)	497
12,NT(WHNP)	9115
13,NT(SINV)	2042
14,NT(FRAG)	514

action	隐状态数
15,NT(NAC)	411
16,NT(WHADVP)	2638
17,NT(PRT)	2641
18,NT(NX)	1344
19,NT(WHPP)	392
20,NT(SQ)	350
21,NT(SBARQ)	222
22,NT(CONJP)	302
23,NT(WHADJP)	59
24,NT(INTJ)	127
25,NT(X)	176
26,NT(RRC)	47
27,NT(LST)	38
28,NT(PRT|ADVP)	**1**

因为有些action经常出现，例如：reduce, shift。因此大部分的隐状态也集中在这些action上。也发现了一个异第28个 action应该被归到第17个action中，后续会来处理）。

3. 因为想利用模型的隐状态和memory states之间的cos距离来增加memory的影响，因此先做了一个统计工作，**验证同一个action对应的states之间的距离要比不同action对应的states之间的距离近** 。首先，求得0-27个action对应的states的center(平均)， 其次，对每个action下的states，求其与这28个center的cos距离。



上图表示 “REDUCE” 对应的states与28个center的cos距离。发现还是和REDUCE本身的center距离相对最近。同理，其他的action下的states都有类似的性质。也就是简单证实了做法的合理性。

4. 将28个action下的centers作为模型图中的memory states, 对应的action的embedding后的结果作为memory actions。
5. 重新训练模型，将每次stack, buffer, action三个lstm输出的hidden vector连接在一起，和memory中的28个states做cos，得到权重；按照权重组合对应的action向量；将得到的向量组合上原始模型输出的向量，具体请见模型结构图。

结果：

还在跑，暂时从模型收敛速度上看不出什么特别的提升。按理说应该会收敛的快一下才对，但是并没有。

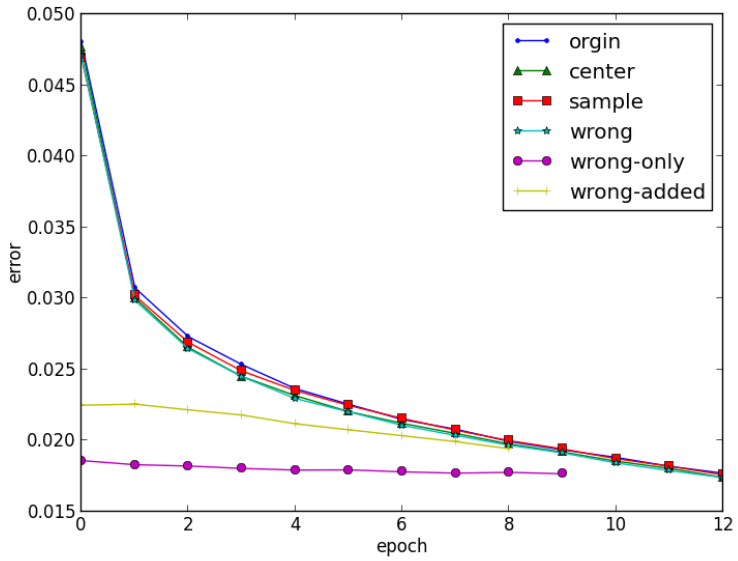
改进方向：

需要改进的地方很多，最容易想到的是memory太少了，简化的太多，可以不只取center，应该还要在每类action中多选择一些memory加进去。

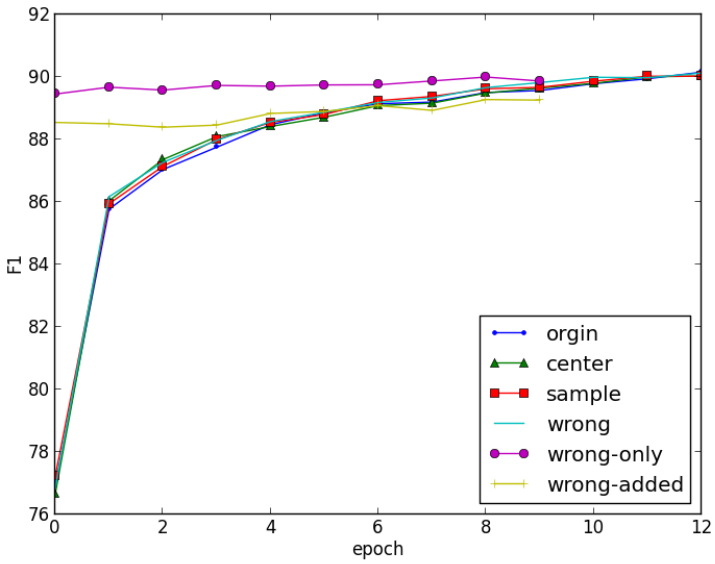
2016.11.7

1. 去除unexpected action **NT(PRT|ADVP)**
2. 重新跑论文模型，控制update 5110次 (12.8289 epoch, 一般已经基本达到最优)
3. 获取论文模型在训练集上做错的部分，共有2599716步输出，其中做错的有37447
4. 重新跑加入28个centered memory的模型, 控制update 5110次
5. 跑加入140个sampled memory的模型, 控制update 5110次
6. 跑加入140个wrong memory的模型, 控制update 5110次
7. 跑加入140个wrong memory, 且原始模型不更新，仅更新上层权重的模型,
8. 跑加入140个wrong memory, 且在原始最优模型基础上，增量更新的模型

不同模型在训练集上的训练过程：



不同模型在训练集上的效果：



不同模型在测试集上的效果：

模型	origin	center	sample	wrong	wrong-only	wrong-added
测试集上的F1 score	91.4	91.26	91.2	91.6	91.25	90.95

检查加入的wrong memory是否有用：

对wrong model, 也就是原模型和mm增加后的参数一起更新的模型，获取其在训练集上做错的部分，有35812个，其中输入进去140个的wrong memory中只有31个作对了。

对wrong-only model, 也就是原模型不变，仅mm增加后的参数更新的模型，获取其在训练集上做错的部分，有38723个，其中输入进去140个的wrong memory中只有12个作对了。

问题：
为什么wrong-only的效果比较不好，是因为memory太少，还是优化函数参数之类的问题？