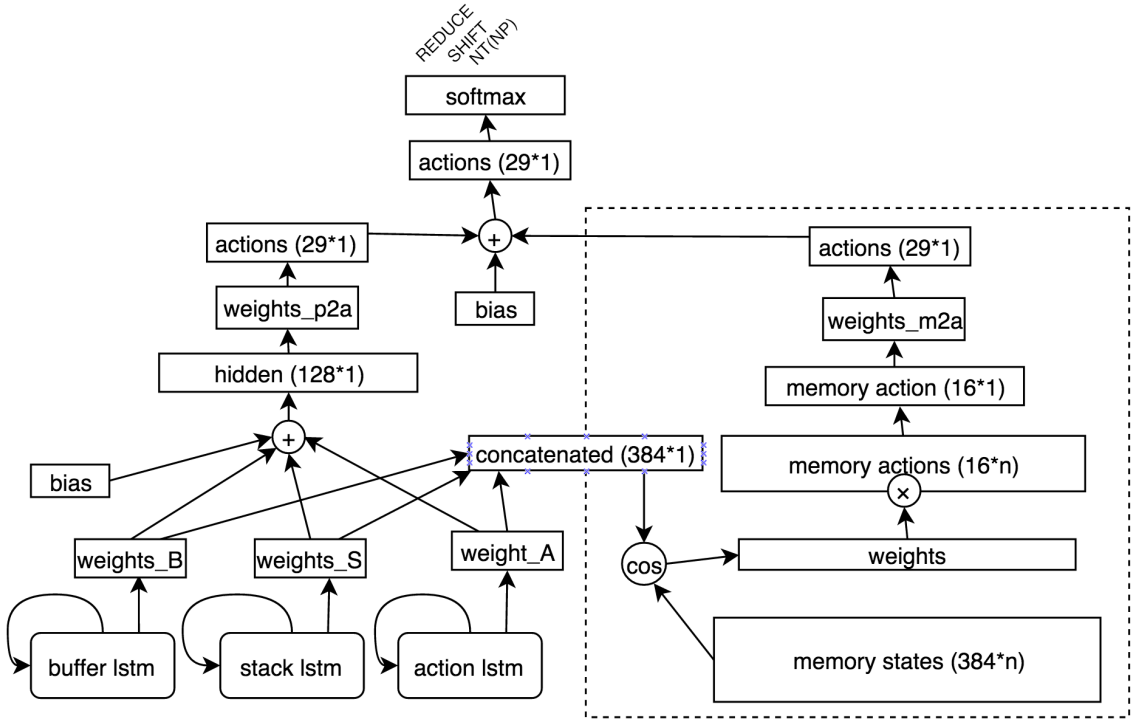
RNNG + Memory 实验报告

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：

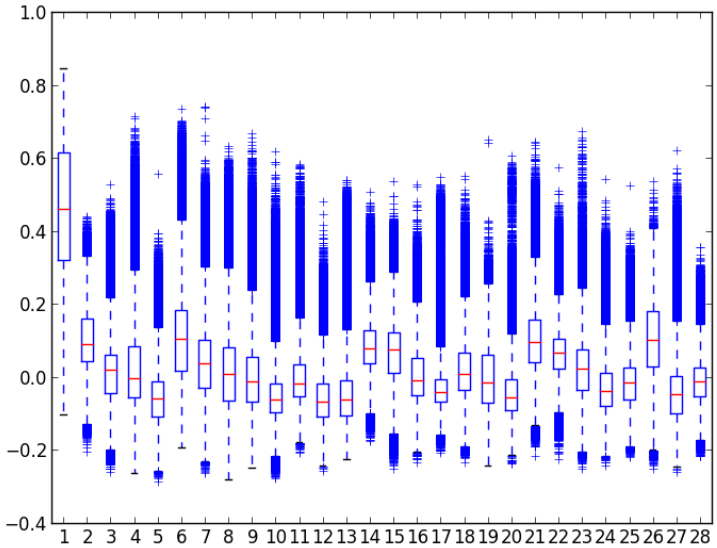
1. 取训练好的模型，重新输入训练集作为测试数据，获取每一步输出的 {stack, buffer, action}384 维隐状态，共有 2599716 个隐状态。
2. 按照隐状态对应的 29 种 action 分类，分布如下：

action	隐状态数	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
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。

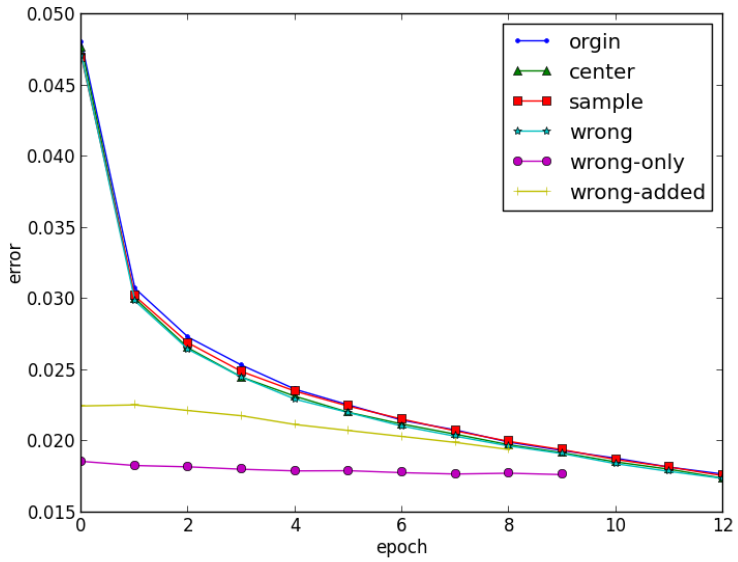
改进方向：

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。

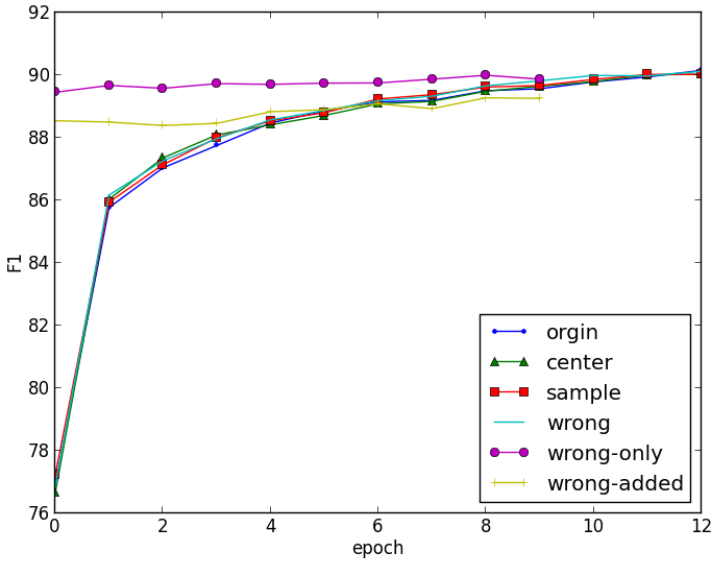
2016.11.7

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：



不同模型在验证集上的效果：



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	91.54

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。

2016.11.14

1. 尝试 GPU

结果：RNNG无法利用GPU得到速度提升：（

Glad you figured it out. Generally we use a hidden dim of 256 (optionally you can also extend it to two-layer LSTM) for the RNNLM.

For the RNNLM, it can be easily batched because the computation graph is linear (you just process the sequence from left to right). In the RNNG, we have composition functions within each phrase (e.g. NP the hungry cat would be composed into a single entry), so the computation graph no longer becomes linear, since the LSTM must also be "rewinded" after each composition, see the paper "Transition-based dependency parsing with stack long short-term memory" (Dyer et al., 2015). Note that the composition depends on the phrase size within the sentence, where a sentence might have a phrase of length 3 that will be composed, while another sentence would have a phrase of length 4.

For this reason the computation graph for each sentence would be different, and this is not easy to batch in GPU as each sentence has a unique graph.

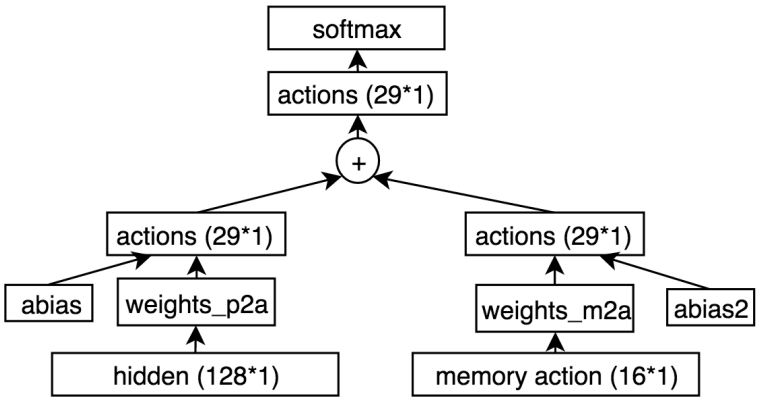
Hope this helps.

下一步准备尝试MKL利用CPU的多核资源。

2. 看了冯老师的代码
帮助做速度上的优化
其他？

3. 模型尝试

sturc2:

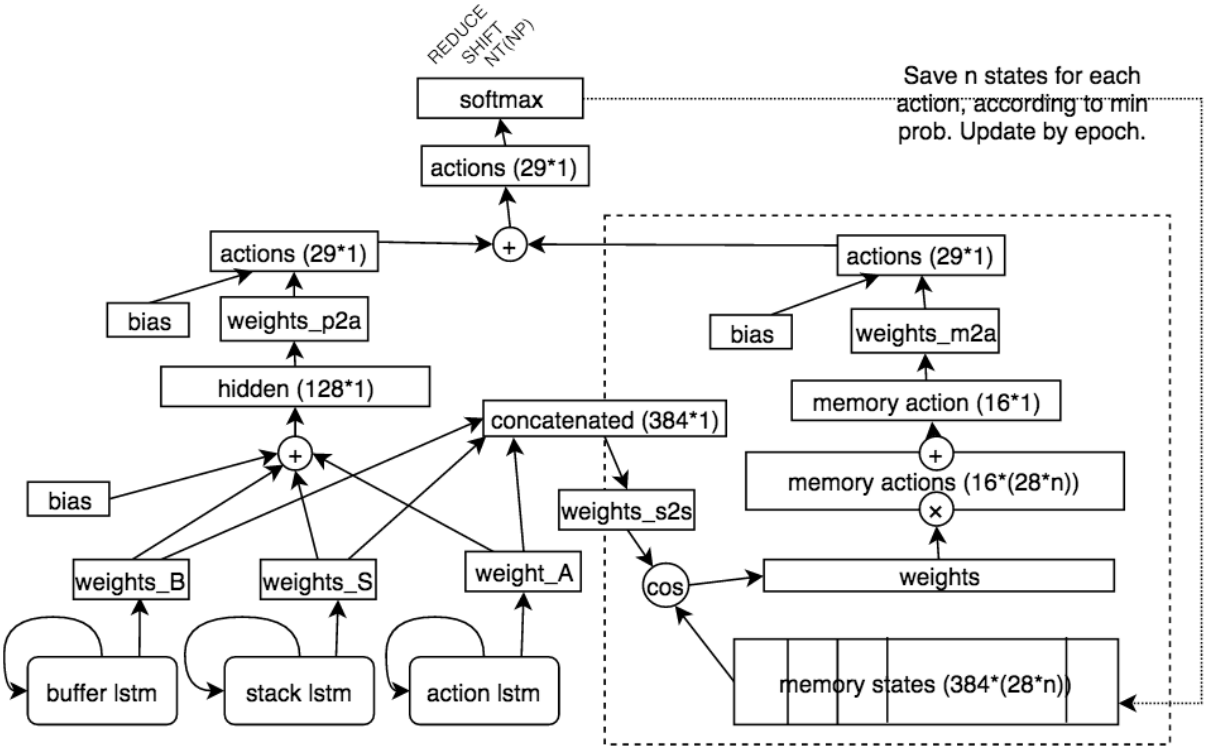


Discriminative model								
model	Name	origin	origin	wrong	wrong-only	wrong2-only		
rnng	If train	yes	yes	yes	no	no	no	no
	epoch	12.2	34+	12.3	12.2	12.2	12.2	12.2
mem	model	no	no	yes	yes	yes	yes	yes
	struc	no	no	struc1	struc1	struc2	struc2	struc2
	size	no	no	140	140	140	140	140
	epoch	no	no	12.3	1.8	3.6	439.7	37.2
other	data	all	all_old	all	all	all	209	18000
Res	F1 on test	91.4	92.26	**91.6**	91.25	91.42	91.39	91.48
	corrected wrong			31	12	16	63	

model	Name	wrong_s2s	wrong_bm		
rnng	If train	yes	yes		
	epoch	33.8	15.3		
mem	model	yes	yes		
	struc	struc1 +s2s	struc1		
	size	140	2417		
	epoch	33.8	15.3		
other	data	all	all		
Res	F1 on test	**92.32**	**91.91**		
	corrected wrong				

下一周的计划：

1. MKL，多核运行提升速度
2. 把baseline等模型跑到确保收敛，对比效果
3. Dynamic memory



2016.11.21

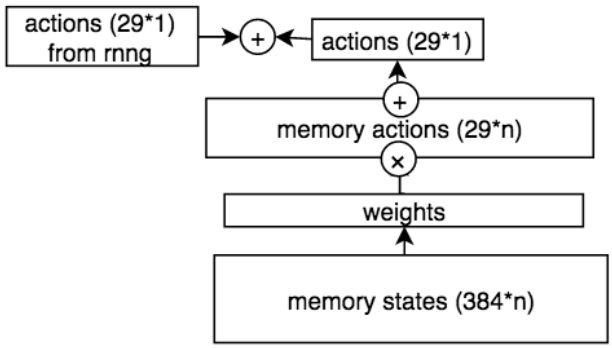
1. 尝试 MKL

成功用 MKL 运行代码，可以使用多核资源，但是最快是在使用 2-3 个核的时候达到，最快可以提升 2-3 倍。把配置过程更新到 RNNG User Guide 文档里。

2 重新跑之前跑过的模型，确保收敛

Discriminative model								
model	Name	origin	static memory			dynamic memory		
rnng	If train	yes	yes	yes		yes		
mem	model	no	yes	yes		yes		
	struc	no	struc1	struc1		struc2 no W before cos		
	size	no	5	100		10		
other	data	all	all	all		all		
Res	F1 on test	92.32	running	running		92.54		

3 尝试另一种结构的 MEM 。



尝试了不同的组合方式，保持原 rnng 模型不变， memory 是 2099 个（每种 action<=100）。

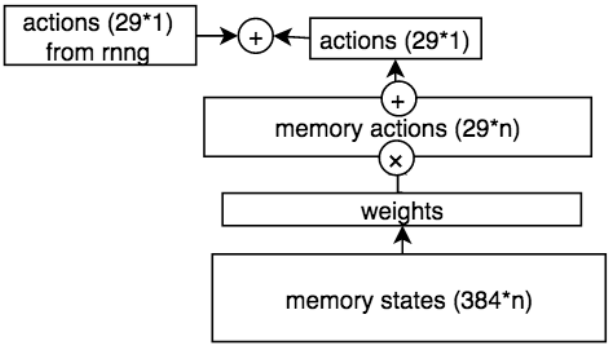
1. 直接加：test f1 92.37， 输入的 memory 判对了 260
2. 手动加权重：
    a. 0.01：test f1 92.32
    b. 0.1：test f1 92.34
    c. 2：test f1 92.16
3. 加序列权重，在全部训练集上训练：test f1 92.2，输入的 memory 判对了 260
4. 加矩阵权重，在全部训练集上训练：test f1 92.13，输入的 memory 判对了 275

2016.11.28

1. Debug

发现了代码中 cos 算的有问题，实际是数据输入有问题。

2. 尝试第二种 mem 结构



不同的组合方式，保持原 rnng 模型不变， memory 是 140 个，每个 action 5 个

1. 直接加 test f1: 92.31 输入的 memory 判对了 0 个
2. 手动调整权重
    a. 10： test f1: 92.32
    b. 100：test f1: 92.29 输入的 memory 判对了 38 个
3. 加 gate，大于某个门限之后直接切换到 memory
    a. gate>0.9: test f1: 92.12 输入 memory 判对了 109
    b. gate>0.95：test f1: 92.22 输入 memory 判对了 109
    c. gate>0.99：test f1: 92.31 输入 memory 判对了 109
    d. gate>0.999：test f1: 92.32 输入 memory 判对了 109

发现了一个 mem 的问题，和记住的 mem 距离近的不一定是同样的类别，导致记住的 men 带来了很大的负面作用。

3. 重新跑模型

Discriminative model seed=4176871112									
model	Name	origin	origin again	static memory			dynamic memory		
rnng	If train	yes	yes	no	no	no	yes	yes	
mem	model	no	no	yes	yes	yes	yes	yes	
	struc	no	no	struc1	struc1	struc2	struc1	struc2	
	size	no	no	5*28	5*28	5*28	10*28	10*28	
other	data	all	all	all	all	all	all	all	
	optimize	sgd1.0	Sgd1.0	Sgd1.0	Sgd0.1	Sgd0.1	Sgd1.0	Sgd1.0	

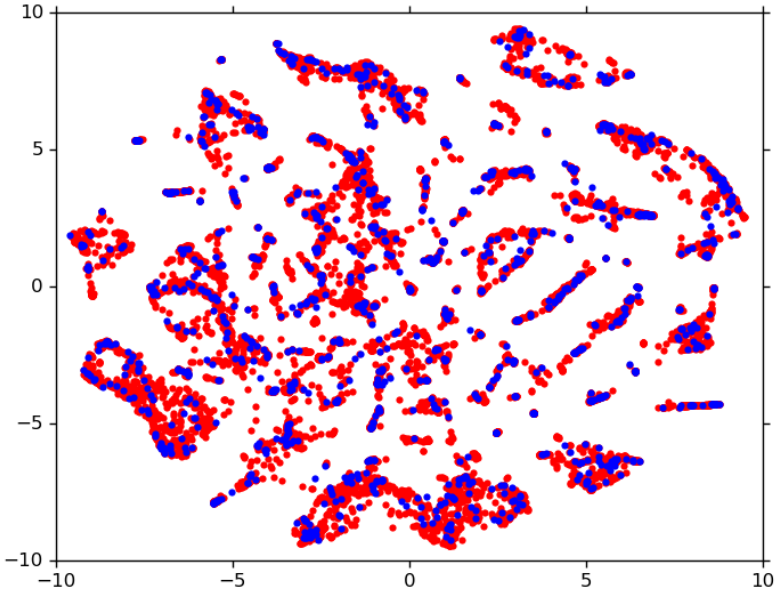
Res	F1 on test	92.32	92.32	91.95	92.41	92.37	92.05	91.99	

总体看来效果都不是很好：原来添加 mem 的方式似乎比 one hot 向量作为 mem 的方式好一些；之前尝试的动态模型的效果也不好；只有在保证原模型不变加 mem 同时调整 learning rate=0.1 时的效果要比 baseline 好个 0.09.

TSNE

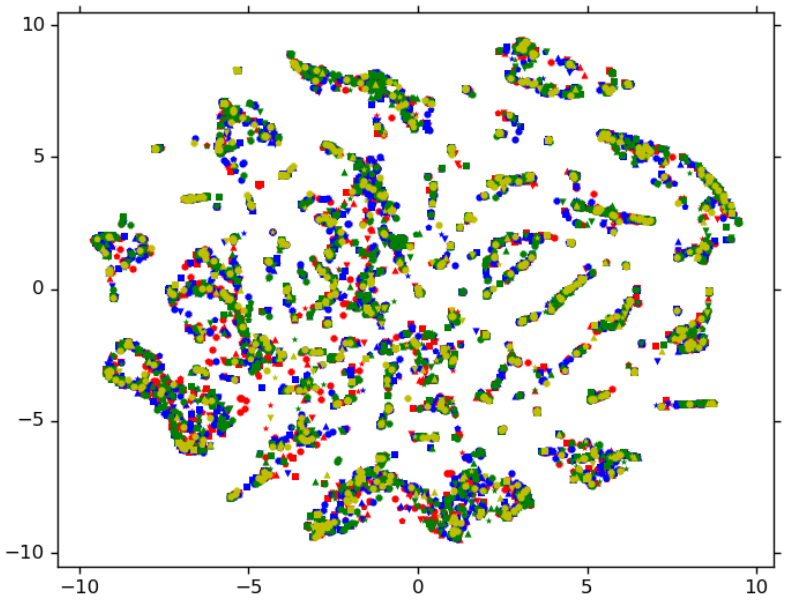
由于原始数据量很大（250 多万）不好画，采样了一部分。对 28 种 action，每个 action 采样 500 个作对的样本，50 个做错的样本。一共 11409 个样本（有些 action 总体的样本数不足 550）。

对错样本的比较，红色为做对的样本，蓝色为做错的样本：



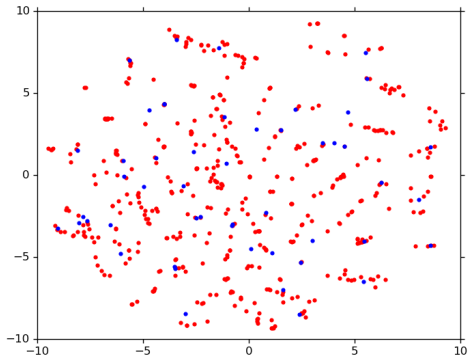
right vs. wrong

不同 action 的样本分布，28 种 action 用 4 种颜色*7 种形状区分：

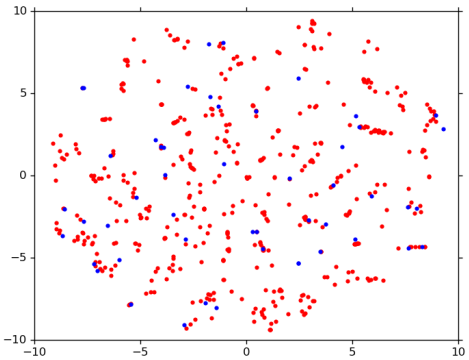


Different actions

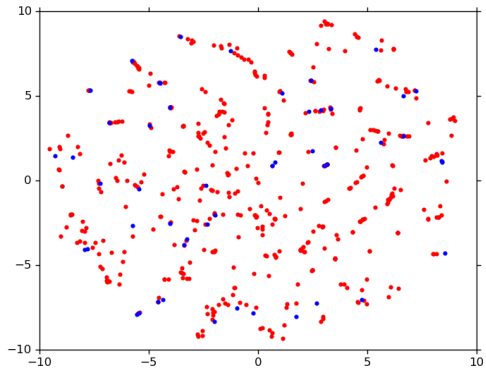
不同 action 下，对错样本的比较，红色为对，蓝色为错：



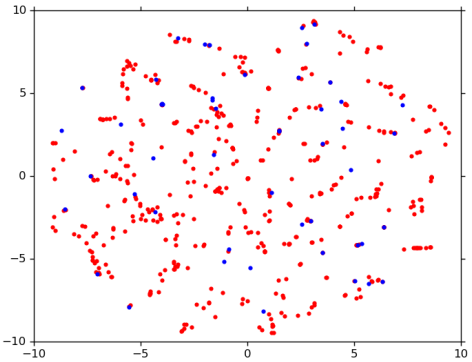
action 0



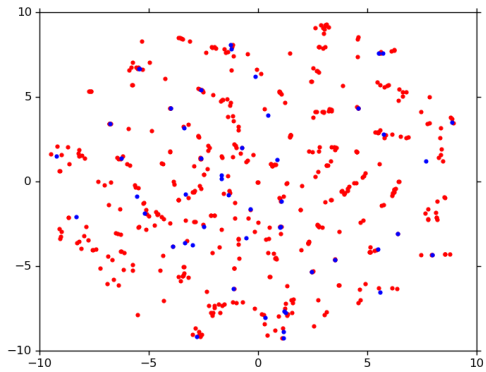
action 1



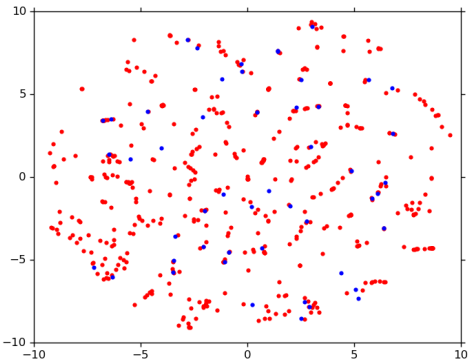
action 2



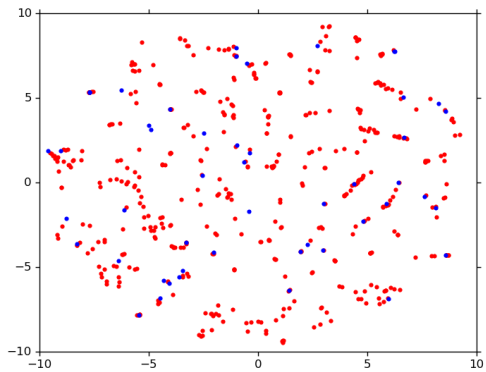
action 3



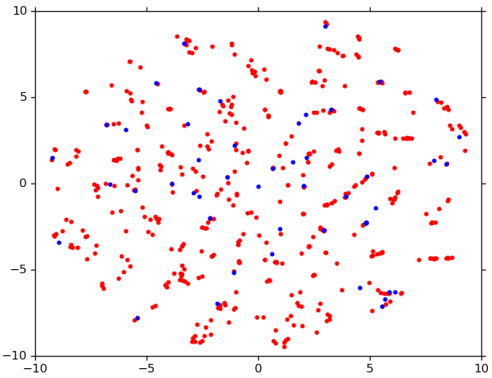
action 4



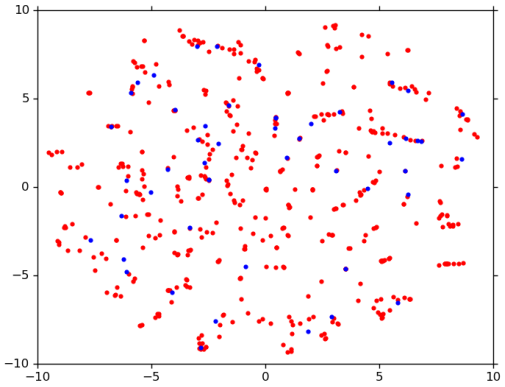
action 5



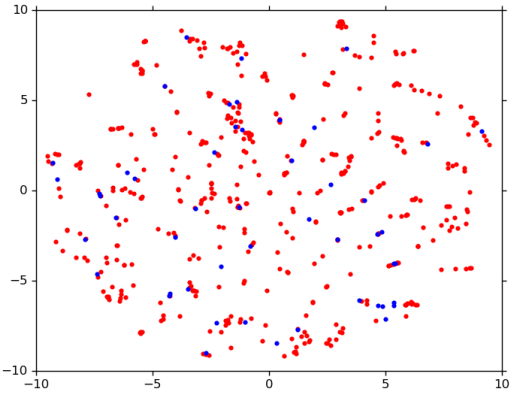
action 6



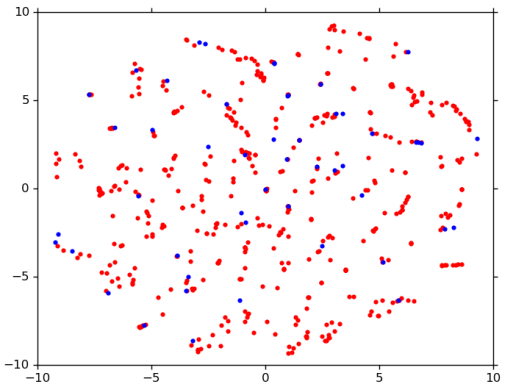
action 7



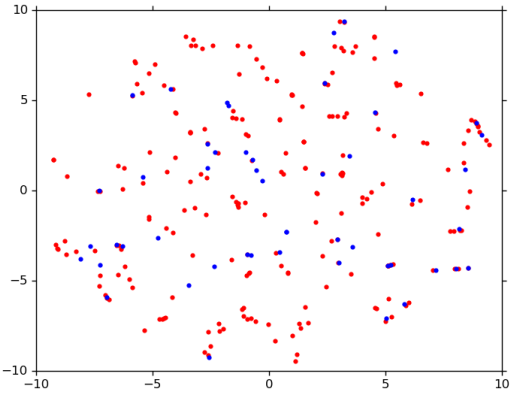
action 8



action 9



action 10



action 11

其他 action 的图基本一致，这里就不贴出来了。。