NasNet 复现报告

安东 andong2019@ia.ac.cn

简述：

没有写过 lstml 和 RL 相关的东西，参照了台大强化学习课程的策略梯度部分，这次的复现还是花了不少功夫，也不知道对不对。有一些问题，主要集中在 controller 实现的板块。

代码地址：https://github.com/MarSaKi/nasnet

问题：

• controller 的设计问题

Controller 被设计为一个 lstml，每个 lstml cell 控制的单元是不同的，一个 block 中：lstml cell_1 控制第一个输入 node，lstml cell_2 控制第二个输入 node，lstml cell_3 控制 node_1 的 op，lstml_cell_4 控制 node_2 的 op……那么每个 lstml cell 输出的 logits 的维度是不一样的。有两个维度的不一样：选择 node 的 cell，选择 op 的 cell，选择 combination 的 cell 输出维度不一样；同一个 normal cell 中，不同 node 的可以选择不同数目的 previous node，例如 node2 可以选择 node0，node1 作为输入，node3 可以选择 node0，node1，node2 作为输入。那么就有一个问题，怎样确保每个 lstml cell 的输入维度是一样的？我想到了两种解决方案：

1. 每个 lstml cell 输出 logits 后，在输入下一个 lstml cell 之前用一个全连接层把它们处理成维度一样的，但这样又有一个问题，全连接层应该是 node 共享，op 共享，combination 共享还是 node，op，combination 都共享？因此实验中我采用了第二种方向。
2. 实验中我定义 normal cell，reduce cell 中只有 4 个 intermediate node。3 个 op [identity, 3x3_sep_conv, 3x3_max_pooling], 2 个 combination [add, concat]，然后我给每个操作都标上序号，node 连接的为[0, 1, 2, 3, 4]，op 的为[5, 6, 7], combination 的为[8, 9]。然后每次 lstml 输出一个 logits，我依据 logits 概率选择一个标号，再输入下一个 lstml cell 之前我把这个标号 embedding 成固定长度的向量(torch.nn.Embedding)。但 lstml 的初始输入又是一个问题，整个 lstml 先控制 normal cell，再控制 reduce cell，最后一个 normal cell 控制的是 reduce cell 的最后一个 node 的 combination，因此我把 lstml 的初始输入固定为 torch.LongTensor([[8]]), 对应 add 操作。
3. 我不知道方案 2 是否是正确的操作，如果每个 lstml 输出的 logits 包含了 node 选择，op 选择，combination 的选择，那么在 sample 的时, 就有可能选 node

的时 sample 到 op/combination，或者之后出现的 node 被 sample 到了。

• 惩罚项超参数的调节问题

训练 controller 的时候我用的是 2017 年的 policy gradient(PPO 还没实现)，学习率固定为 0.00035，还有一个 entropy 的惩罚项，它的系数固定为 0.00001，比较疑惑的是 entropy 的惩罚参数怎么选择，训练过程中我观察到 entropy 并没有改变。Controller 的优化目标应该是最大化 policy grading 的期望项，最大化 entropy？

• batch size，episode size 等调节问题

1. 我做了一个实验对比每个 episode 中 batch size 的对比，一个的 batch size 是 128，另一个 batch size 是 1024（为了提高速度），每个 episode 中网络训练 2 个 epoch，发现 batch size 128 得到的 valid acc 普遍比 batch size 1024 得到的 valid acc 大。
2. episode 的问题，episode 应该是越大越好？但那样 controller 训练一个 epoch 的时间比较长，因此实验中我吧 episodes 固定为 8，也就是说 controller 每更新一次都需要先跑 8 个 episode。或许有并行的写法？我尝试了用 multiprocessor，把一个 epoch 中的所有 model 都 map 到 valid acc 的计算函数上，但是不 work，nn.Module 好像是 unpickle 的。写法如下：

```
pool = multiprocessing.Pool(12)
list_valid_acc = pool.map(self.get_valid_acc, list_genotype)
pool.close()
pool.join()
```

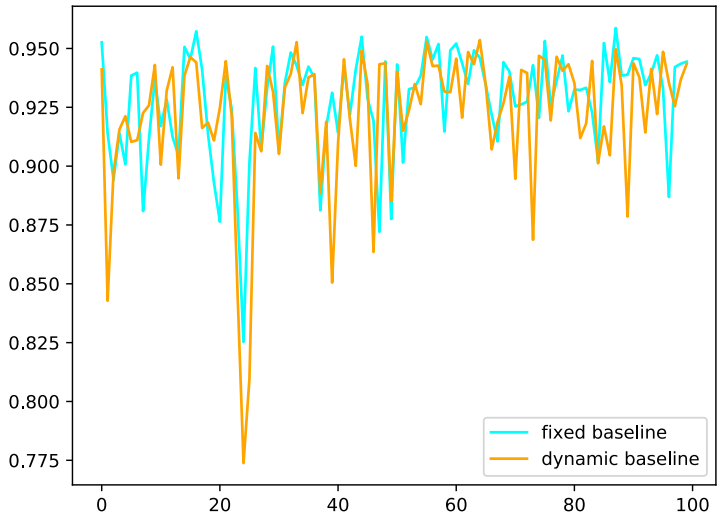
• RL 对比 random search 的优势体现？

简而言之 RL 的 pipline 是：根据 controller 的 logits 分布采样出 model，训练 model 得到 valid acc 作为 reward 去修改 contoller 参数，使得 controller 采样出“好 model”的概率越来越大。那么采样次数足够多的情况下，random search 和 RL 采出的最好 model 性能应该是差不多的？RL 做的事情可能是加速 search，使得它每次采出的 model 都尽可能是好的？

实验：

• 对比计算 reward 时固定 baseline 和动态 baseline。

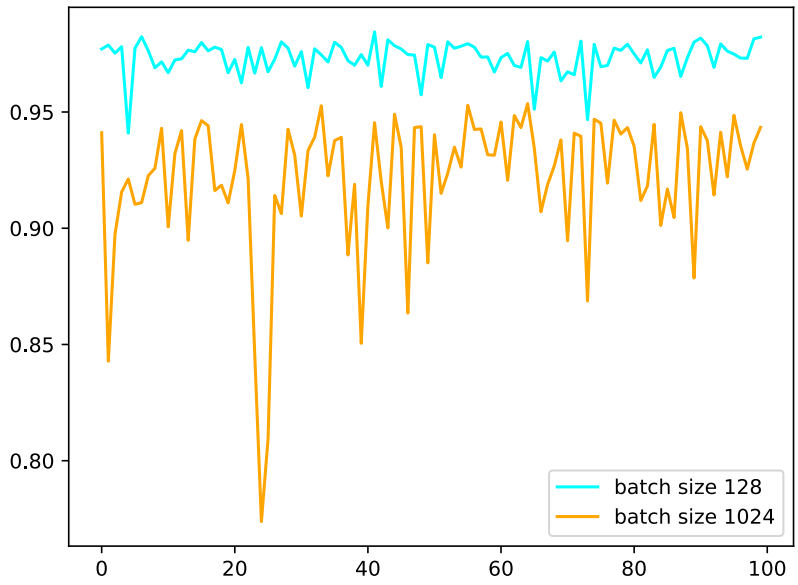
Controller 的 reward=valid_acc – baseline，针对 mnist 数据集(比较简单)，固定 baseline 中我把 baseline 固定为 0.95，也就是说 controller 采样出的 model 要达到 0.95 的正确率才能获得正向 reward；动态 baseline 中记录了之前所有 episode 的 valid_acc 均值，将其作为 baseline。Controller 跑了 100 个 epoch，每个 epoch8 个 episode，统计每个 epoch 的平均 valid acc 作为衡量两个方法的指标。实验结果如下：



1. 两种方法效果是差不多的，或许 fixed baseline 要稍好一点？
2. 100epoch 搜索下来，model 的表现在震荡，没有观察到效果提升，可能 mnist 数据集太简单了？又或者代码哪里出了问题？

• 对比 model 评估中的 batch size 影响，batch size=128，batch size=1024

定义的 model 比较简单，图片->preprocess->normal cell->reduce cell->normal cell->reduce cell->global pooling->classifier。对比结果如下：



由于 batch size 1024 和 batch size 128 时每个 episode 中 model 训练的 epoch 都一样为 2，所以导致效果可能差距比较大。或许 batch size 1024 的时候每个 model 应该多训练几轮？值得注意的是 batch size 128 的时候仍然没有看到训练 contorller 有效果提升（他们的 acc 已经很高了，0.98+）（感觉和 random search 差不多？），或许因为 mnist 太简单了？

Batch size 128 的最好 model acc 为 0.9865， batch size 1024 的最好 model acc 为 0.9757.

Idea：

• darts 中的退化问题

“UNDERSTANDING AND ROBUSTIFYING DIFFERENTIABLE ARCHITECTURE SEARCH“ 中提到 darts search 阶段，validation loss 对 alpha 的 Hessian 矩阵的特征值增过大的时候，darts 搜索出的网络会开始退化，那么是否可以在 search 的 loss 中设计一个有关 hessian 矩阵特增值的损失项来优化搜索过程？

• RL 中 reward 传给 controller 的每个节点

controller 的 loss 计算中，只用到了最后一个 lstml cell 的 reward，并把它作为所有 lstml cell 的 reward，RL 中还可以在 reward 反传的过程中增加一个 gamma 系数（比如 0.95），最后一个 lstml cell 的 reward 是 R，倒数第二个是 0.95*R，倒数第三个是 0.95*0.95*R…我怀疑这样做对每个 lstml cell 的 reward 的估计是否是有效的（中间 action 有可能做出一个重大的决策导致整体 reward 上升很多/下降很多）。但 R(a_t|a_t-1)也和前面 t-1 个，后面 T-t 个操作都有关，我不确定是否可以找到一种方法把 R(a_t|a_t-1)正确估计出来，或许 RL 有解决这个问题？或许可以从隐马尔科夫链的维特比算法中找点灵感？