

On Reinforcement Learning for Full-length Game of StarCraft

Zhen-Jia Pang, Ruo-Ze Liu, Zhou-Yu Meng, Yi Zhang, Yang Yu[†], Tong Lu

National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China

[†]To whom correspondence should be addressed; E-mail: yuy@nju.edu.cn.

【强化学习算法 16】NJUStarCraft



张楚珩 💙

清华大学 交叉信息院博士在读

5 人赞同了该文章

以前在俞扬老师那里做毕设的时候,俞扬老师还讲过算法起名字的重要性,但这篇工作好像并没有官方给的名字。为了便于本专栏区分,就把它叫做NJUStarCraft吧。

原文传送门:

Pang, Zhen-Jia, et al. "On Reinforcement Learning for Full-length Game of StarCraft." arXiv preprint arXiv:1809.09095 (2018).

特色:

StarCraft属于目前强化学习解决起来非常困难的一类游戏了,之前的算法有针对局部战斗的,而针对整个游戏的算法通常不能打败最简单的内置AI。这个工作通过expert trajectory、hierarchical RL和curriculum learning来训练,最后能够以相当的胜率打败内置的AI,甚至是内置的困难AI。

Table 2: Evaluation the policy against Protoss and Zerg without re-training.

Opponent's Type	Non-cheating (No-Training)							Cheating (No-training)		
Difficulty Level	1	2	3	4	5	6	7	8	9	10
vs Zerg	1	0.99	1	0.98	0.98	0.99	0.94	0.89	0.82	0.35
vs Protoss	1	1	0.92	0.76	0.46	0.40	0.45	0.47	99 F	@佛楚珩

分类:

Hierarchical RL (using PPO) for specific task

过程:

1. Hierarchical RL

主要分为三个层级,controller、sub-policy和macro action。

Controller: controller是最上层的策略 π ,其状态空间 s_a 是一些宏观的游戏特征,而其行动空间 a_a 是下一层的所有sub-policy。这一层每K步会调用一次,而这之间每步控制权都交给sub-policy。这一层主要用来管理游戏的宏观策略。

Sub-policy: 一共有n个sub-policy π_i $i \in \mathbb{N}$,每个sub-policy的状态空间 g_i 和行动空间 g_i 可以不相同,其行动空间是下一层次macro action的集合。一共有两类sub-policy,一类主要负责建造各种建筑或者可移动单元,另一类负责战斗。

Macro action: macro action主要通过已有的expert trajectory使用prefix-span算法来做数据挖掘,挖掘出来常见的操作序列,通过筛选得到 \mathbf{x} 个macro action。举个例子,macro action就是把常见操作打包,比如造建筑操作就打包成"选择一个农民-选择一个地点让其建造-建造完成之后把它拉回来"。

2. Reward

有三个可能的选择:每一局比赛的胜负奖励,这个奖励就十分稀疏;暴雪算出来的得分;自定义的奖励。本文就是使用的结合了前两者自己定义出来的奖励,对于每一步macro action步骤都会有奖励,sub-policy的更新就使用这些一步步的奖励;对于controller来讲,使用的奖励是在sub-policy的K步内得到的总奖励。

3. Training

训练方法使用增加了entropy loss项的PPO算法,对于controller和每一个sub-policy都使用各自独立的experience replay来分别使用PPO来更新权重。

4. Curriculum Learning

内置AI难度由易到难有9个等级,训练AI的时候就由易到难使用这些内置的AI来进行对战训练。在AI 难度切换(环境发生变化)的时候,相当于是一个transfer learning,如果仍然再同时训练controller 和sub-policy的话会不稳定,因此就固定一个训另外一个。

算法:

```
Algorithm 1 RL training algorithm
Input: Number of sub-policys N, time interval K, reward function R_c, R_1, R_2, ..., R_n, max
episodes M, max iteration steps Z
  Initialize replay buffer \langle D_c, D_1, D_2, ..., D_n \rangle, controller policy \Pi_{\phi}, each sub-policy \pi_{\theta_i}
  for i = 1 to Z do
     clear data buffer \langle D_c, D_1, D_2, ..., D_n \rangle
     for k = 1 to M do
        collect \tau_c = \{(s_1^c, a_1^c, r_1^c, s_2^c), ...\} in 1/K timescale
        D_c \leftarrow D_c \cup \tau_c
        for i = 1 to N do
           collect \tau_i = \{(s_1^i, a_1^i, r_1^i, s_2^i), ...\} in full timescale
            D_i \leftarrow D_i \cup \tau_i
        end for
     end for
     using D_c to update \phi to maximize expected return
     for i = 1 to N do
        using D_i to update \theta_i to maximize expected return
     end for
   end for
```

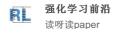
其他工程细节?

- 1. 作战的方式:对于作战环境,使用了combat network + combat rule的混合方式。combat rule就是把兵直接拖到离基地最远的地方,然后遇到敌人自动攻击;combat network则是使用了CNN网络,输入小地图的信息,输出一个位置向量和一个行动,行动从所有单元攻击某个位置、所有单元撤退到某个位置和不动这三者中间选择一个。
- 2. 分布式训练:使用了10个worker,每个worker使用了5个线程,这50个线程独立模拟游戏并且收集数据到replay buffer,每个worker独立计算梯度,所有梯度发送到一个parameter server进行参数更新。训练使用了48个CPU和8个K40 GPU,训练时间不超过两天,模拟游戏8万场。

编辑于 2018-10-09



文章被以下专栏收录



进入专栏