第十一讲：强化学习前沿

主讲人 陈达贵

清华大学自动化系

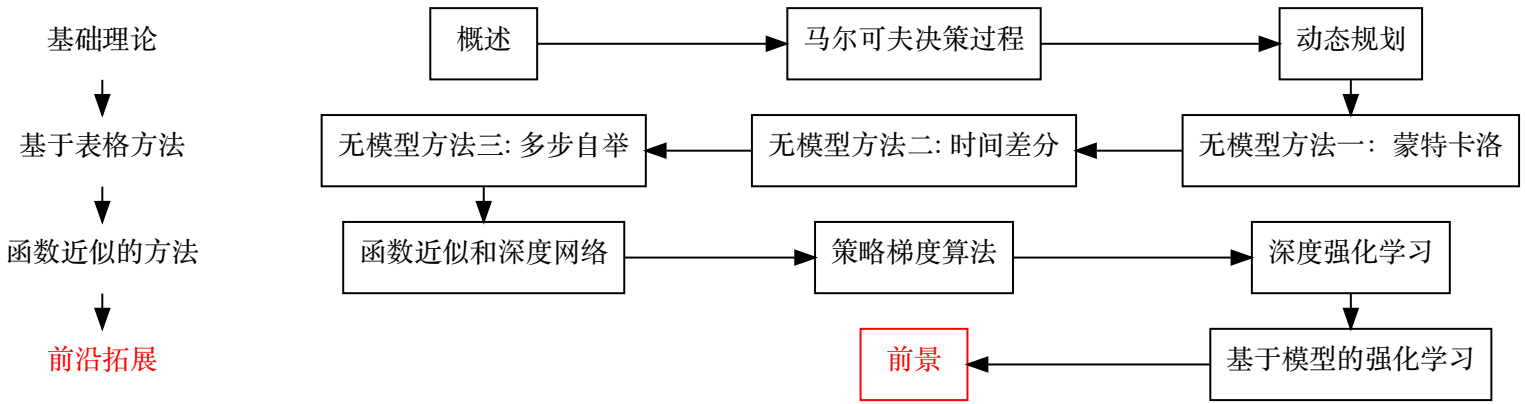
在读硕士

深蓝学院
shenlanxueyuan.com

强化学习理论与实践

目录

章节目录


基础理论
基于表格方法
函数近似的方法
前沿拓展
概述
马尔可夫决策过程
动态规划
无模型方法一：蒙特卡洛
无模型方法二: 时间差分
无模型方法三: 多步自举
函数近似和深度网络
策略梯度算法
深度强化学习
基于模型的强化学习
前景

本章目录

课程小结

- 从传统强化学习出发
  - 数学化地介绍了强化学习问题——MDP
  - 着重介绍了无模型方法 (表格情况下)
  - 以及在无模型下的函数近似的方法
  - 主要介绍了基于值函数的方法
  - 同时补充了策略梯度方法
- 概要式地梳理了 DRL
- 简要介绍了基于模型的强化学习

课程目标

希望大家学习完这门课后，能够回答以下问题：

- 知道强化学习究竟在解决什么问题？
- 其数学基础是什么？
- 主要的方法基础是什么？
- 每种方法背后的原理是什么？
- 不同的方法之间的优缺点是什么？
- 是否有一个清晰的知识树?

目录

参考文献

A Brief Survey of Deep Reinforcement Learning
Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, Anil Anthony Bharath

Reward-Driven Behaviour

- 强化学习的所有行为都是由奖励驱动的。
  - 最大化期望回报值
  - 奖励函数的设置是强化学习中的关键
- RL 所面临的一些独特的挑战
  - 最优的策略只能通过试错来获得，唯一的学习信号就是奖励
  - 观测数据和动作相关，具有时间相关性
  - 序列化的动作导致了奖励信号，因此存在信度分配问题

RL 算法

- 值函数方法
  - 动态规划，蒙特卡洛，Sarsa, Q 学习，采样
- 策略搜索
  - 进化算法，策略梯度，REINFORCE
- Actor-Critic
  - 方差减小，优势函数
- 规划
  - Dyna，MCTS
- DRL
  - DQN，A2C, DPPG，PPO

研究方向和挑战

- Model-based RL
- Exploration vs. Exploitation
- Hierarchical RL
- Imitation Learning and Inverse RL
- Multi-agent RL
- Memory and Attention
- Transfer Learning
- Benchmarks

Model-based RL

- 大幅减少真实样本复杂度
- 精确的模型可以降低策略的复杂度
- 可以融入人为先验
- 学习的环境模型可以指导探索

Exploration vs. Exploitation

- RL 中的基本问题
- ε-greedy
- Noise Net
- Upper confidence bound
- 好奇心算法

Hierarchical RL

- 分层策略
- 分层的动作：action 和 option
- 高层策略负责选择子策略，子策略负责执行
- 高层策略负责生成子目标，子策略负责完成子目标
- 大大减小学习的复杂度
- 能够具有更高的可解释性和迁移性

Imitation Learning and Inverse RL

- 使用监督学习从专家示例中学习
- Behaviour cloning
- 学习奖励函数
- 使用对抗学习来学习奖励函数

Multi-agent RL

- 考虑多个智能体
- 博弈论
- 完全信息和不完全信息
- 允许交流和不允许交流
- 竞争，合作

Memory and Attention

- CNN→RNN，整合序列信息，处理 POMDP
- 使用 Attention 选择下个关注的输入部分
- 除了处理序列信息外，可以增加可微分的 Memory
- 学习可微分的读写操作
- 给智能体增加了利用记忆的功能

Transfer Learning

- 迁移学习，多任务学习，课程学习
- 从一个任务迁移到另一个任务
- 从虚拟环境迁移到真实的环境
- 辅助任务
- distillation

Benchmarks

- Atari
- Vizdoom
- Mujoco
- StarCraft
- Deepmind Lab

目录

参考文献

Deep Reinforcement Learning that Matters

Peter Henderson[1]*, Riashat Islam[1,2]*, Philip Bachman[2]
Joelle Pineau[1], Doina Precup[1], David Meger[1]
[1] McGill University, Montreal, Canada
[2] Microsoft Maluuba, Montreal, Canada
{peter.henderson,riashat.islam}@mail.mcgill.ca, phbachma@microsoft.com
{jpineau,dprecup}@cs.mcgill.ca, dmeger@cim.mcgill.ca

概要

- Reproducing existing work
- Accurately judging the improvements offered by novel methods
- Investigate challenges posed by reproducibility, proper experimental techniques, and reporting procedures
- Reproducibility can be affected by:
  - Extrinsic factors (e.g. hyperparameters and codebases)
  - Intrinsic factors (e.g. random seeds and environments properties)
- Focus on DDPG, PPO, TRPO, ACKTR

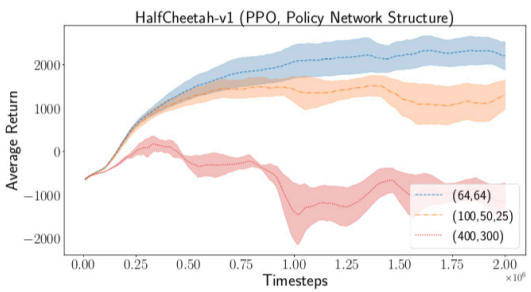
Hyperparameters

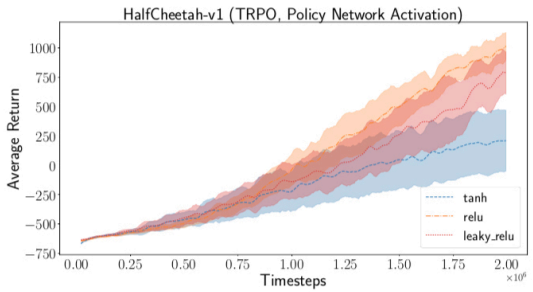
- What is the magnitude of the effect hyperparameter settings can have on baseline performance?

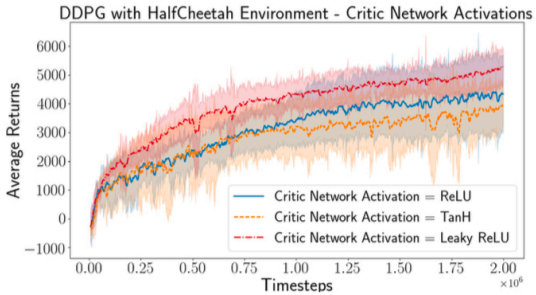
- PPO
  - Policy Network: (64, tanh, 64, tanh, Linear) + Standard Deviation variable; Value Network (64, tanh, 64, tanh, linear)
  - Normalized observations with running mean filter
  - Timesteps per batch 2048
  - clip param = 0.2
  - entropy coeff = 0.0
  - Optimizer epochs per iteration = 10
  - Optimizer step size $3e-4$
  - Optimizer batch size 64
  - Discount $\gamma = 0.995$, GAE $\lambda = 0.97$
  - learning rate schedule is constant

Network Architecture

- How does the choice of network architecture for the policy and value function approximation affect performance?







Network Architecture

- Policy Network

Algorithm	Environment	400,300	64,64	100,50,25	tanh	ReLU	LeakyReLU
TRPO	Hopper-v1	2980 ± 35	2674 ± 227	3110 ± 78	2674 ± 227	2772 ± 211	-
(Schulman et al. 2015a)	HalfCheetah-v1	1791 ± 224	1939 ± 140	2151 ± 27	1939 ± 140	3041 ± 161	-
TRPO	Hopper-v1	1243 ± 55	1303 ± 89	1243 ± 55	1303 ± 89	1131 ± 65	1341± 127
(Duan et al. 2016)	HalfCheetah-v1	738 ± 240	834 ± 317	850±378	834 ± 317	784 ± 352	1139 ±364
TRPO	Hopper-v1	2909 ± 87	2828 ± 70	2812 ± 88	2828 ± 70	2941 ± 91	2865 ± 189
(Schulman et al. 2017)	HalfCheetah-v1	-155 ± 188	205 ± 256	306 ± 261	205 ± 256	1045 ± 114	778 ± 177
PPO	Hopper-v1	61 ± 33	2790 ± 62	2592 ± 196	2790 ± 62	2695 ± 86	2587 ± 53
(Schulman et al. 2017)	HalfCheetah-v1	-1180 ± 444	2201 ± 323	1314 ± 340	2201 ± 323	2971 ± 364	2895 ± 365
DDPG	Hopper-v1	1419 ± 313	1632 ± 459	2142 ± 436	1491 ± 205	1632 ± 459	1384 ± 285
(Plappert et al. 2017)	HalfCheetah-v1	5579 ± 354	4198 ± 606	5600 ± 601	5325 ± 281	4198 ± 606	4094 ± 233
DDPG	Hopper-v1	600 ± 126	593 ± 155	501 ± 129	436 ± 48	593 ± 155	319 ± 127
(Gu et al. 2016)	HalfCheetah-v1	2845 ± 589	2771 ± 535	1638 ± 624	1638 ± 624	2771 ± 535	1405± 511
DDPG	Hopper-v1	506 ± 208	749 ± 271	629 ± 138	354 ± 91	749 ± 271	-
(Duan et al. 2016)	HalfCheetah-v1	850 ± 41	1573 ± 385	1224 ± 553	1311 ± 271	1573 ± 385	-
ACKTR	Hopper-v1	2577 ± 529	1608 ± 66	2287 ± 946	1608 ± 66	2835 ± 503	2718 ± 434
(Wu et al. 2017)	HalfCheetah-v1	2653 ± 408	2691 ± 231	2498 ± 112	2621 ± 381	2160 ± 151	2691 ± 231

Table 1: Results for our policy architecture permutations across various implementations and algorithms. Final average ± standard error across 5 trials of returns across the last 100 trajectories after 2M training samples. For ACKTR, we use **ELU** activations instead of leaky ReLU.

Network Architecture

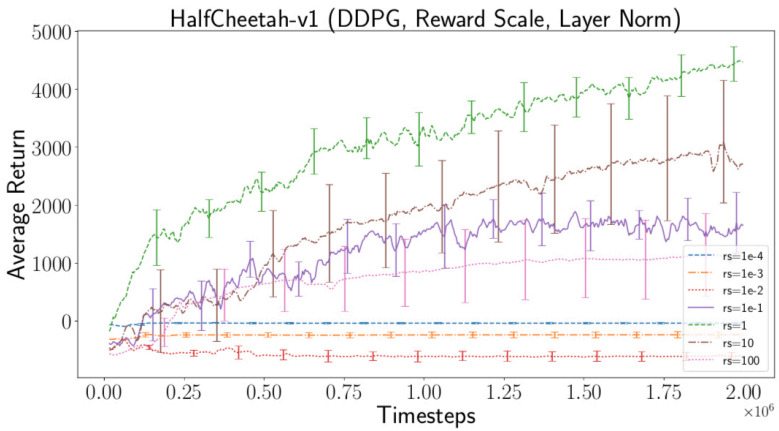
- Value Network

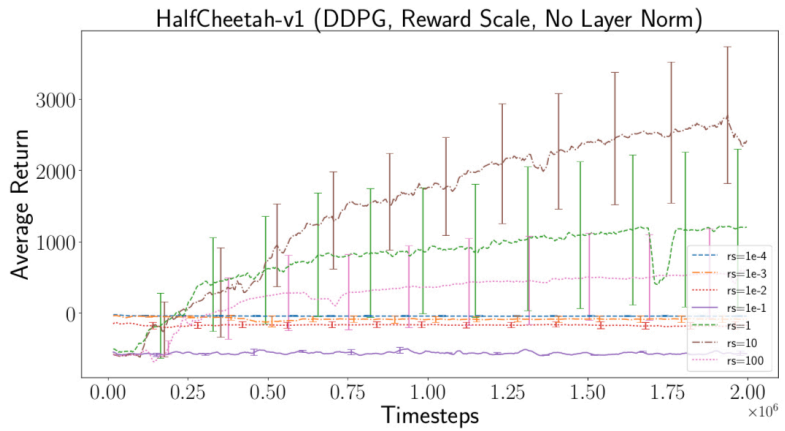
Algorithm	Environment	400,300	64,64	100,50,25	tanh	ReLU	LeakyReLU
TRPO	Hopper-v1	3011 ± 171	2674 ± 227	2782 ± 120	2674 ± 227	3104 ± 84	-
(Schulman et al. 2015a)	HalfCheetah-v1	2355 ± 48	1939 ± 140	1673 ± 148	1939 ± 140	2281 ± 91	-
TRPO	Hopper-v1	2909 ± 87	2828 ± 70	2812 ± 88	2828 ± 70	2829 ± 76	3047 ± 68
(Schulman et al. 2017)	HalfCheetah-v1	178 ± 242	205 ± 256	172 ± 257	205 ± 256	235 ± 260	325 ± 208
PPO	Hopper-v1	2704 ± 37	2790 ± 62	2969 ± 111	2790 ± 62	2687 ± 144	2748 ± 77
(Schulman et al. 2017)	HalfCheetah-v1	1523 ± 297	2201 ± 323	1807 ± 309	2201 ± 323	1288 ± 12	1227 ± 462
DDPG	Hopper-v1	1419 ± 312	1632 ± 458	1569 ± 453	971 ± 137	852 ± 143	843 ± 160
(Plappert et al. 2017)	HalfCheetah-v1	5600 ± 601	4197 ± 606	4713 ± 374	3908 ± 293	4197 ± 606	5324 ± 280
DDPG	Hopper-v1	523 ± 248	343 ± 34	345 ± 44	436 ± 48	343 ± 34	-
(Gu et al. 2016)	HalfCheetah-v1	1373 ± 678	1717 ± 508	1868 ± 620	1128 ± 511	1717 ± 508	-
DDPG	Hopper-v1	1208 ± 423	394 ± 144	380 ± 65	354 ± 91	394 ± 144	-
(Duan et al. 2016)	HalfCheetah-v1	789 ± 91	1095 ± 139	988 ± 52	1311 ± 271	1095 ± 139	-
ACKTR	Hopper-v1	152 ± 47	1930 ± 185	1589 ± 225	691 ± 55	500 ± 379	1930 ± 185
(Wu et al. 2017)	HalfCheetah-v1	518 ± 632	3018 ± 386	2554 ± 219	2547 ± 172	3362 ± 682	3018 ± 38

Table 2: Results for our value function (Q or V) architecture permutations across various implementations and algorithms. Final average ± standard error across 5 trials of returns across the last 100 trajectories after 2M training samples. For ACKTR, we use **ELU** activations instead of leaky ReLU.

Reward Scale

- How can the reward scale affect results? Why is reward rescaling used?





Random Seeds and Trials

- Can random seeds drastically alter performance? Can one distort results by averaging an improper number of trials?

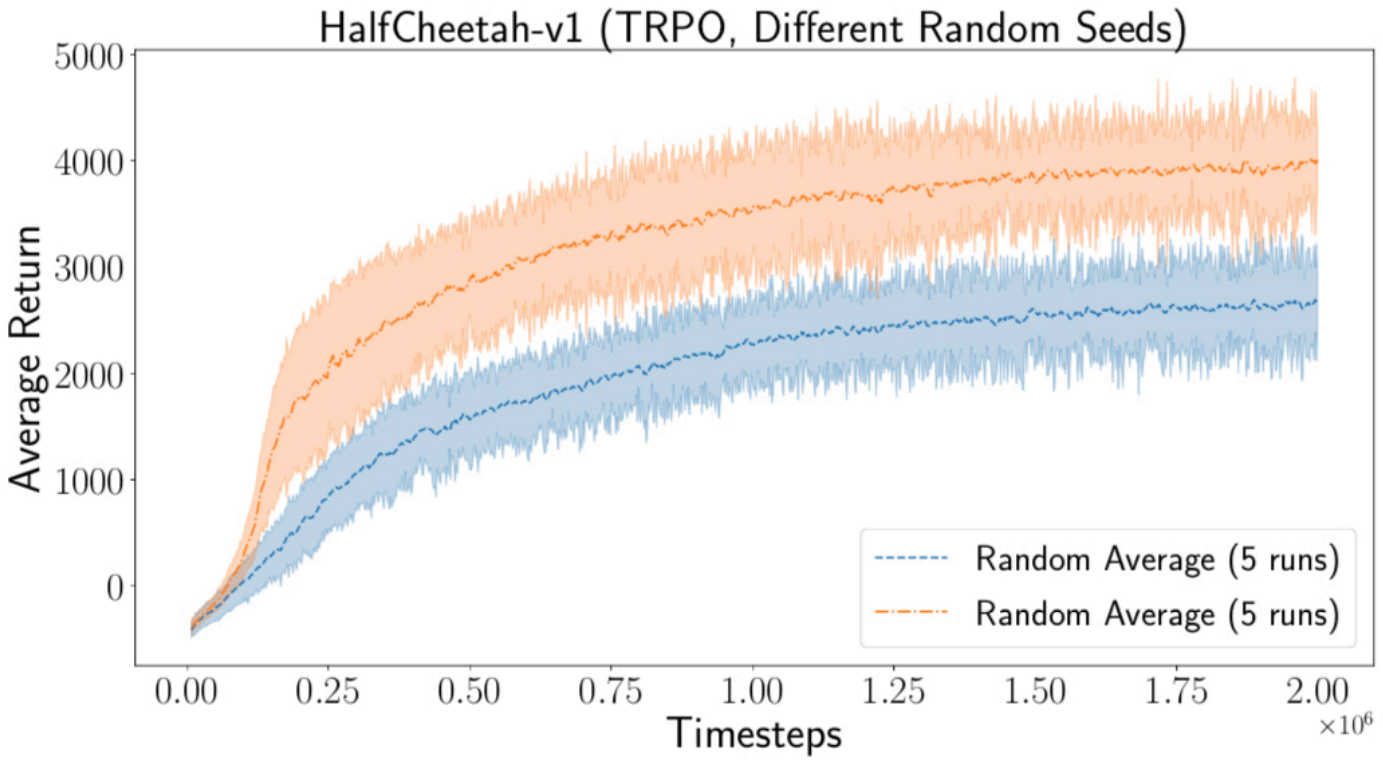


Figure 5: TRPO on HalfCheetah-v1 using the same hyperpa-

Environments

- How do the environment properties affect variability in reported RL algorithm performance?

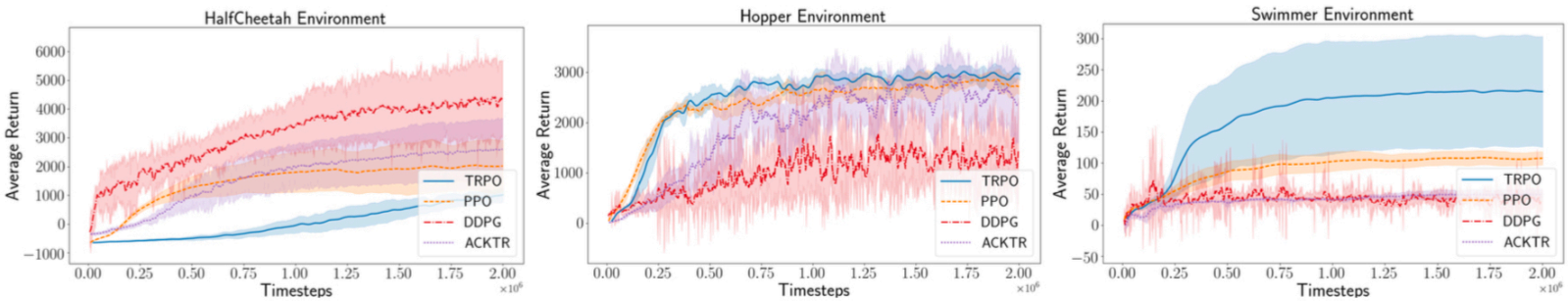


Figure 4: Performance of several policy gradient algorithms across benchmark MuJoCo environment suites

Environment	DDPG	ACKTR	TRPO	PPO
HalfCheetah-v1	5037 (3664, 6574)	3888 (2288, 5131)	1254.5 (999, 1464)	3043 (1920, 4165)
Hopper-v1	1632 (607, 2370)	2546 (1875, 3217)	2965 (2854, 3076)	2715 (2589, 2847)
Walker2d-v1	1582 (901, 2174)	2285 (1246, 3235)	3072 (2957, 3183)	2926 (2514, 3361)
Swimmer-v1	31 (21, 46)	50 (42, 55)	214 (141, 287)	107 (101, 118)

Table 3: Bootstrap mean and 95% confidence bounds for a subset of environment experiments. 10k bootstrap iterations and the pivotal method were used.

Codebases

- Are commonly used baseline implementations comparable?

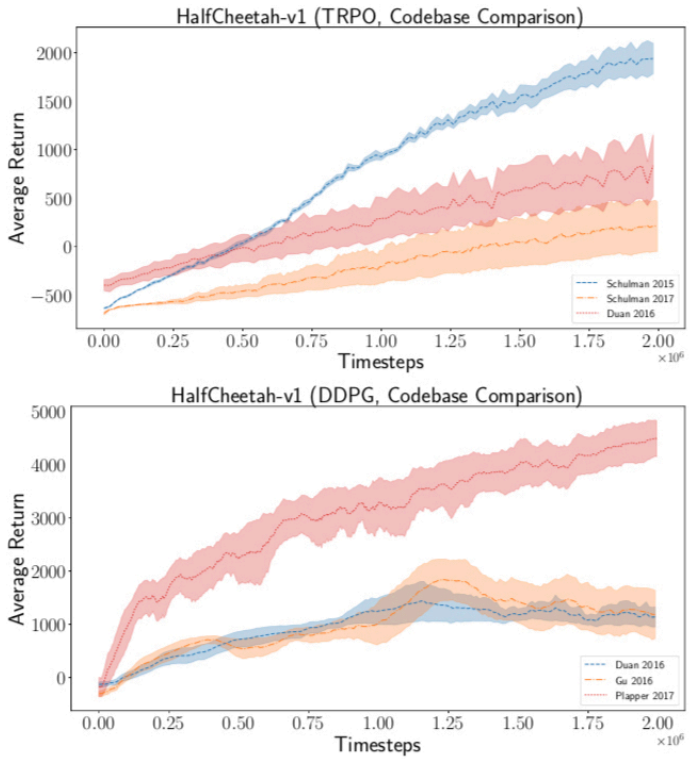
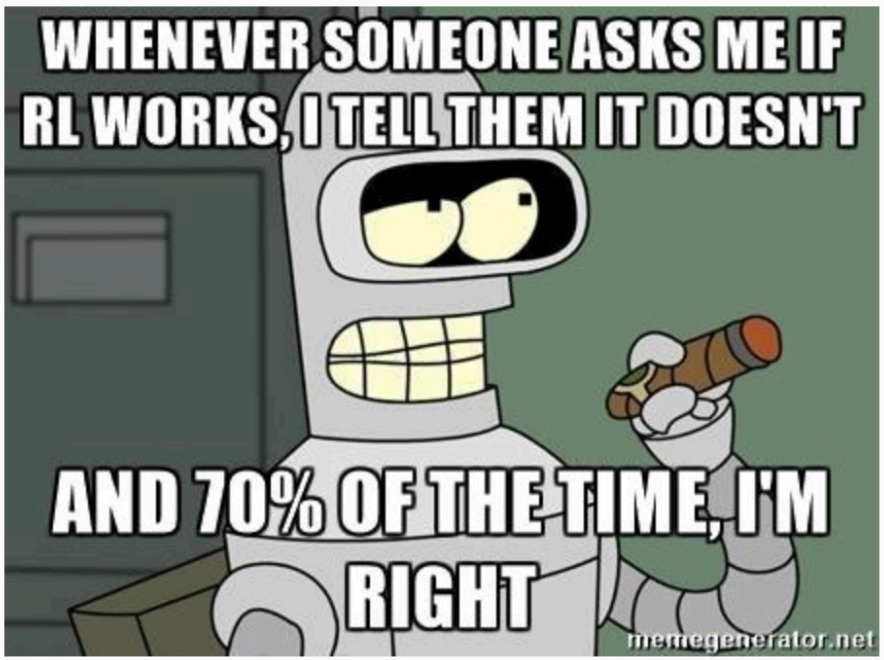


Figure 6: TRPO codebase comparison using our default set of hyperparameters (as used in other experiments).

目录

Deep Reinforcement Learning Doesn't Work Yet

- https://www.alexirpan.com/2018/02/14/rl-hard.html
- Author: Irpan, Alex



Sample Inefficient

- Deep Reinforcement Learning Can Be Horribly Sample Inefficient
- Atari:
  - Rainbow: 18 million frames: 83 hours of play experience
  - Distributional DQN: 70 million frames
  - Nature DQN: 200 million frames
- Mujoco benchmarks: $10^5 \sim 10^7$ steps
- DeepMind parkour paper: using 64 workers for over 100 hours

Poor Performance

- If You Just Care About Final Performance, Many Problems are Better Solved by Other Methods
- For purely getting good performance, deep RL's track record isn't that great
- Model based control and Linear Quadratic Regulator can achieve higher performance and lower sample complexity
- MCTS

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
DQN	4092	168	470	20	1952	1705	581
-best	5184	225	661	21	4500	1740	1075

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
UCT	7233	406	788	21	18850	3257	2354

Reward Function

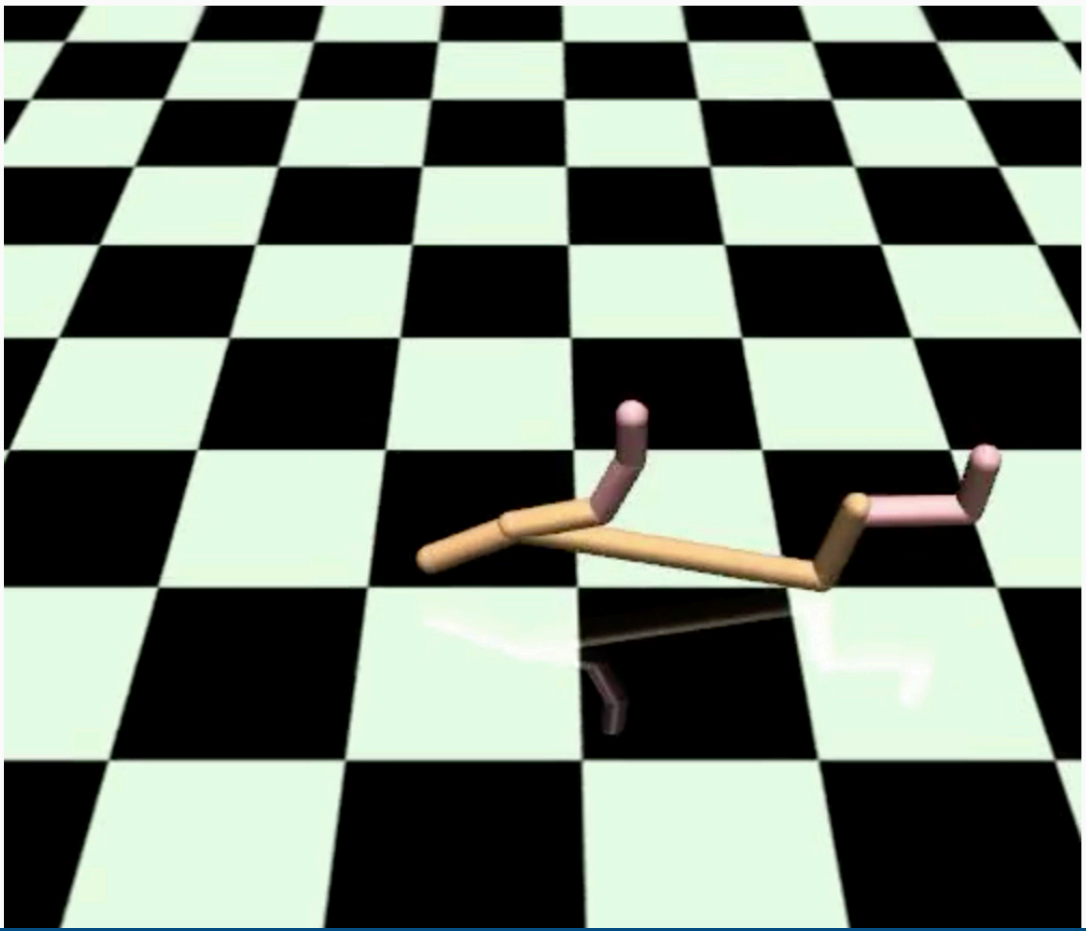
- Reinforcement Learning Usually Requires a Reward Function
- Reward Function Design is Difficult

例子

- A coworker is teaching an agent to navigate a room. The episode terminates if the agent walks out of bounds. He didn't add any penalty if the episode terminates this way. The final policy learned to be suicidal, because negative reward was plentiful, positive reward was too hard to achieve, and a quick death ending in 0 reward was preferable to a long life that risked negative reward.
- A friend is training a simulated robot arm to reach towards a point above a table. It turns out the point was defined with respect to the table, and the table wasn't anchored to anything. The policy learned to slam the table really hard, making the table fall over, which moved the target point too. The target point just so happened to fall next to the end of the arm.
- A researcher gives a talk about using RL to train a simulated robot hand to pick up a hammer and hammer in a nail. Initially, the reward was defined by how far the nail was pushed into the hole. Instead of picking up the hammer, the robot used its own limbs to punch the nail in. So, they added a reward term to encourage picking up the hammer, and retrained the policy. They got the policy to pick up the hammer···but then it threw the hammer at the nail instead of actually using it.

Local Optima

- Even Given a Good Reward, Local Optima Can Be Hard To Escape
- Exploration-exploitation trade-off wrong.



Overfitting to Weird Patterns

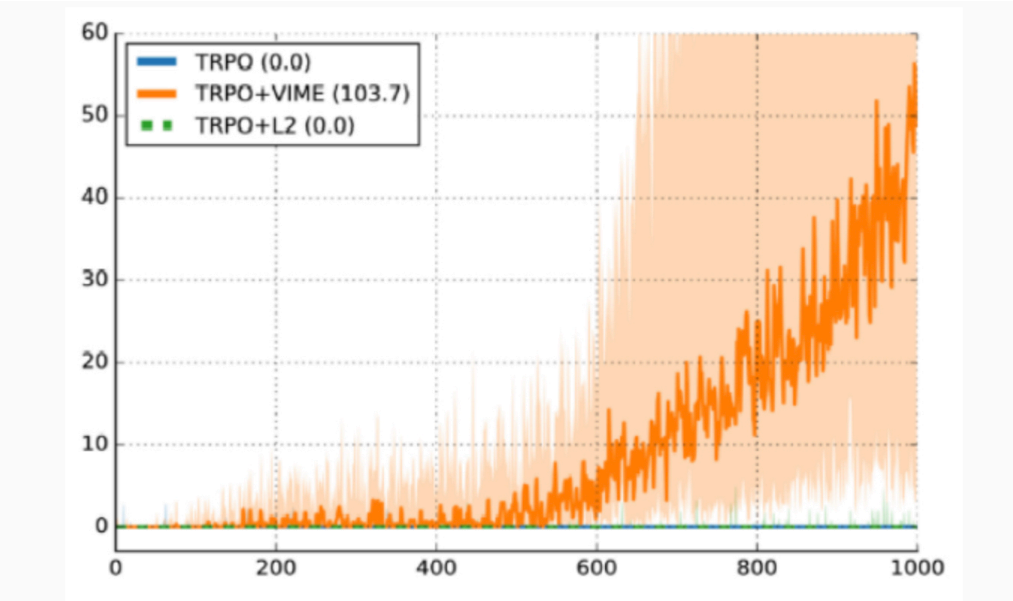
- Even When Deep RL Works, It May Just Be Overfitting to Weird Patterns In the Environment
- The final model won't generalize to other games,

例子

We studied a toy 2-player combinatorial game, where there's a closed-form analytic solution for optimal play. In one of our first experiments, we fixed player 1's behavior, then trained player 2 with RL. By doing this, you can treat player 1's actions as part of the environment. By training player 2 against the optimal player 1, we showed RL could reach high performance. But when we deployed the same policy against a non-optimal player 1, its performance dropped, because it didn't generalize to non-optimal opponents.

Hard to reproduce

- Even Ignoring Generalization Issues, The Final Results Can be Unstable and Hard to Reproduce



Common properties that make learning easier

- It is easy to generate near unbounded amounts of experience.
- The problem is simplified into an easier form.
- There is a way to introduce self-play into learning.
- There's a clean way to define a learnable, ungameable reward.
- If the reward has to be shaped, it should at least be rich.

Looking to The Future

- Local optima are good enough
- Hardware solves everything
- Add more learning signal
- Model-based learning unlocks sample efficiency
- Use reinforcement learning just as the fine-tuning step
- Reward functions could be learnable
- Transfer learning saves the day
- Good priors could heavily reduce learning time
- Harder environments could paradoxically be easier

目录

小结

- 挑战与机遇并存!

致谢

谢谢支持!