面向博弈类游戏的深度强化学习算法框架

李亦琛

南方科技大学

2023 年 5 月 25 日

Outline

1 绪论

2 深度强化学习

3 框架设计与实现

4 框架实验

5 总结

6 参考文献

研究背景

博弈类游戏是人工智能应用领域的重要研究方向之一，具有很高的复杂度与挑战性。在传统人工智能算法面临限制的情况下，深度强化学习算法的逐步成熟为博弈类游戏的智能化提供了强有力的支持。当前的 DRL 算法分为基于值函数、策略梯度和行动者评论家三类，已有许多成功的游戏 AI 案例，如 AlphaGo[1] 和斗地主 AI DouZero[2] 等。DRL 算法在游戏智能研究领域有广泛的应用前景和研究价值。

研究意义

本项目设计了一个面向博弈类游戏的 AI 算法框架，旨在帮助用户更好地应用深度强化学习算法。该框架集成了多种深度强化学习算法，并提供了经典的博弈类游戏环境供用户使用。用户可以根据自己的需求和实际情况，选择不同的算法和游戏环境进行训练和测试，以不断提高模型的性能。该框架具有上手容易、接口友好等特点，具有良好的应用前景，在游戏领域具有重要的实际意义。

Outline

1 绪论

2 深度强化学习

3 框架设计与实现

4 框架实验

5 总结

6 参考文献

基于值函数的 DRL 算法

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

DQN[3]

将强化学习与卷积神经网络相结合，分别用两个神经网络来近似表示值函数以及产生目标 Q 值，并使用经验池存储转移样本。

DDQN[4]

解决 DQN 过度估计的问题，对目标 Q 值的动作选择和策略评估解耦。

Prioritized DDQN[5]

对 DQN 中经验池等概率随机采样进行改进，提出将样本的时间差分 TD 误差作为优先级标准进行采样。

Dueling DDQN[6]

对 DQN 网络结构进行优化，将神经网络提取出的特征分流为两个支路，分别为价值函数和优势函数两个部分。

基于策略梯度的 DRL 算法

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

REINFORCE[7]

通过采样并利用蒙特卡洛方法来估计策略函数梯度，并使用梯度上升方法进行优化，最终目的是使得输出动作序列的期望收益最大化。梯度更新公式为 $\theta_{t+1} = \theta_t + \alpha(G_t - b(s_t))\nabla log\pi(a_t|s_t, \theta_t)$。

PPO-clip[8]

鉴于标准策略梯度方法对每个数据样本执行一次梯度更新，PPO 提出了一种新的目标函数，该函数支持多个 epochs 的小批量更新：$L(\theta) = \sum_{(s_t,a_t)} min(\frac{p_\theta(a_t|s_t)}{p_{\theta_{old}}(a_t|s_t)}A(s_t,a_t), clip(\frac{p_\theta(a_t|s_t)}{p_{\theta_{old}}(a_t|s_t)}, 1+\epsilon, 1-\epsilon)A(s_t,a_t))$。

基于行动者-评论家的 DRL 算法

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

A2C[9]

使用 Actor-Critic 架构，其中 Actor 表示策略网络，负责生成一个动作的分布，Critic 表示价值网络，它估计当前状态的值函数。A2C 算法使用优势函数来衡量某个状态下采取某个动作的收益优劣，并使用基于梯度的方法来同时优化 Actor 和 Critic 网络。

A3C[10]

将 Actor-Critic 框架与并行计算结合起来，通过多个 Agent 在各自的环境下同时学习，提高学习效率。每个 Agent 都有自己的 Actor 和 Critic，通过共享梯度来更新全局模型。

Outline

1 绪论

2 深度强化学习

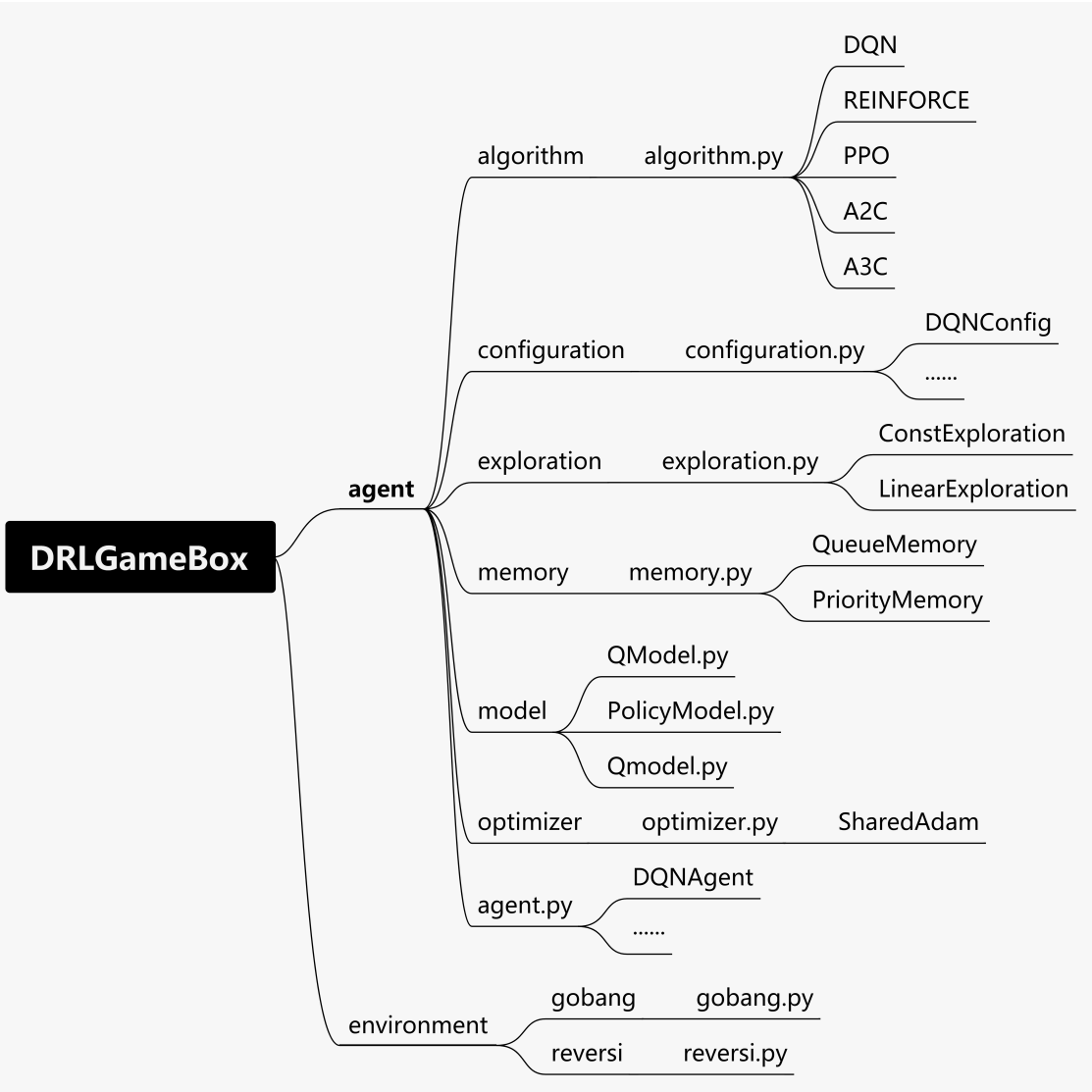
3 **框架设计与实现**

4 框架实验

5 总结

6 参考文献

框架概览


DRLGameBox
agent
algorithm
algorithm.py
DQN
REINFORCE
PPO
A2C
A3C
configuration
configuration.py
DQNConfig
.....
exploration
exploration.py
ConstExploration
LinearExploration
memory
memory.py
QueueMemory
PriorityMemory
model
QModel.py
PolicyModel.py
Qmodel.py
optimizer
optimizer.py
SharedAdam
agent.py
DQNAgent
.....
environment
gobang
gobang.py
reversi
reversi.py

模块实现

算法	内容	功能
Algorithm	DQN 等一系列 DRL 算法类	算法核心
Configuration	DQNConfig 等算法配置类	算法所需的参数配置
Exploration	ConstExploration 类、 LinearExploration 类	探索策略
Memory	QueueMemory 类、 PriorityMemory 类	经验池
Model	QModel 等一系列网络类	算法所需模型
Optimizer	SharedAdam	为 A3C 算法提供 梯度同步
Agent	DQNAgent 等一系列 Agent 类	封装 DRL 算法
Environment	Gobang、Reversi	参考 Gym 库的接口 实现的游戏环境

应用实例

Listing 1: **PriotizedDuelingDDQN.py**

```
1  from agent import DQNAgent
2  from agent.configuration import DQNConfig
3  from agent.exploration import LinearExploration
4  from agent.memory import PriorityMemory
5  from agent.model import LinearQModel
6  from experiment_gobang.code.experiment import do_exp
7
8  config = DQNConfig(
9      eval_model=LinearQModel(450, 255, 512, dueling=True),
10     target_model=LinearQModel(450, 255, 512, dueling=True),
11     ckpt_path='../checkpoint/dueling_ddqn_p.ckpt',
12     batch_size=32,
13     lr=0.00001,
14     gamma=0.99,
15     target_replace_frequency=100,
16     capacity=50000,
17     max_grad_norm=1.0,
18     exploration=LinearExploration(1.0, 0.1, 0.995),
19     ddqn=True,
20     memory=PriorityMemory(50000)
21 )
22 agent = DQNAgent(config)
23 do_exp(agent, 15, 1000, 50, 50, '../data/dueling_ddqn_p')
```

Outline

1 绪论

2 深度强化学习

3 框架设计与实现

4 框架实验

5 总结

6 参考文献

训练环境（五子棋）

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

环境设置

选择棋盘大小为 15×15 的五子棋作为游戏环境。将棋盘转化为黑方视角，并将黑白棋子分别编码至第一层和第二层，作为状态 s。

奖励设置

每轮游戏结束前每步奖励为-1，最后一步若导致平局则奖励为-0.2，赢棋则奖励为 0。

训练和测试

每个 Agent 共训练 1000 轮，每训练 50 轮进行一次测试。测试内容为与随机算法 Agent 对打 50 轮。

参数配置

每个算法的参数配置参考了原论文等资料，并进行了相应的调整。

统计曲线

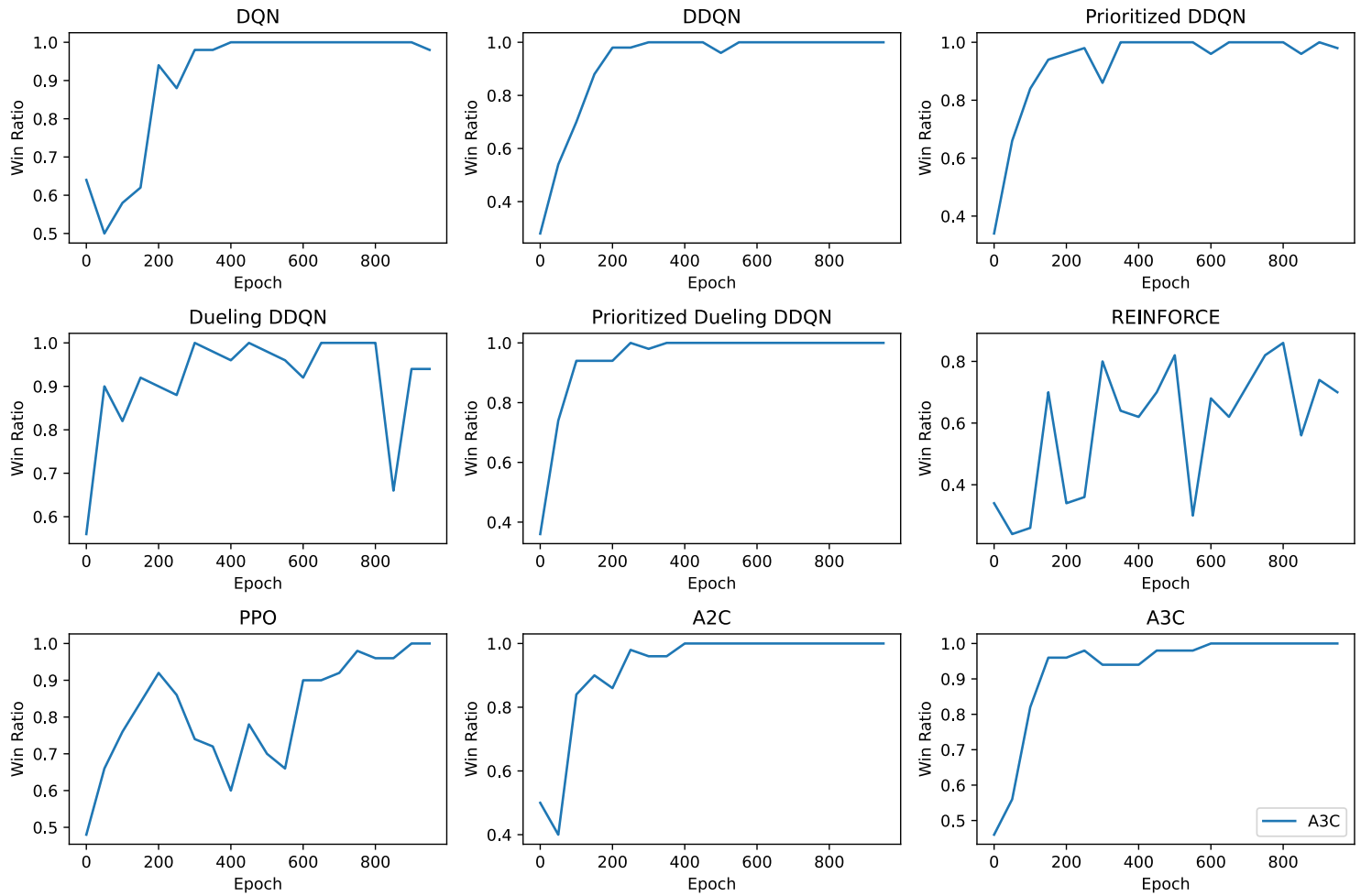


图: DRL 算法在五子棋环境中与随机算法智能体对打的胜率

结果分析

- DQN、DDQN、Prioritized DDQN、Prioritized Dueling DDQN、A2C、A3C 算法比较稳定。
- 在实验初期，DDQN 收敛快于 DQN，而 Prioritized DDQN 与 DDQN 差距不明显。
- Dueling DDQN 可能不适合该游戏环境。
- PPO 虽然在实验末期达到 100% 胜率，可能由于训练轮数受限或参数设置问题导致性能一般。
- A2C 和 A3C 均为 AC 架构，尽管 A3C 采用了异步架构，但最终结果差别不大。

实验结论

- DRL 算法相对于 AlphaGo[1]等树搜索基础架构在棋类游戏上仍有所欠缺。
- DRL 算法对参数配置极其敏感。在训练时损失函数容易发散，必须调小学习率，设置梯度裁剪等。
- DRL 算法受奖励函数影响极大。在实验过程中，采用赢棋奖励 1，其余为 0 的策略会导致训练结果发散，而采用赢棋奖励 0，其余为-1 的策略时能够收敛。
- DRL 算法不宜使用较复杂的网络结构。在 DQN 系列算法的网络结构中使用一层卷积层，在训练过程中始终发散。而后参考了 DouZero[2]论文实验中 DQN 的网络架构，采用 MLP 模型，发现调整好参数后可以收敛并能够取得较好的结果。

Outline

1 绪论

2 深度强化学习

3 框架设计与实现

4 框架实验

5 总结

6 参考文献

总结

本项目基于深度强化学习算法，设计了针对博弈类游戏的 AI 算法框架，通过对前沿 DRL 算法的学习和复现，在框架的构建中充分运用了其核心原理。实验测试和性能评估表明，该框架具有良好的应用效果和灵活性。同时，该项目提供了博弈类游戏智能化和人工智能研究的重要支持和参考，并提供方便易懂的操作接口。然而，在算法和性能效率的优化方面存在不足之处，需要进一步研究探索和完善。在应用场景和领域方面，也需要不断拓展和完善以适应不同需求和实际情况。未来，我将继续优化和扩展该框架，并在实际应用中开展进一步探索和验证。

Outline

1 绪论

2 深度强化学习

3 框架设计与实现

4 框架实验

5 总结

6 参考文献

参考文献 I

[1] SILVER D, HUANG A, MADDISON C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.

[2] ZHA D, XIE J, MA W, et al. Douzero: Mastering doudizhu with self-play deep reinforcement learning[C]//International Conference on Machine Learning. 2021: 12333-12344.

[3] MNIH V, KAVUKCUOGLU K, SILVER D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[4] VAN HASSELT H, GUEZ A, SILVER D. Deep reinforcement learning with double q-learning[C]//Proceedings of the AAAI conference on artificial intelligence: vol. 30: 1. 2016.

[5] SCHAUL T, QUAN J, ANTONOGLOU I, et al. Prioritized experience replay[J]. arXiv preprint arXiv:1511.05952, 2015.

参考文献 II

[6] WANG Z, SCHAUL T, HESSEL M, et al. Dueling network architectures for deep reinforcement learning[C]//International conference on machine learning. 2016: 1995-2003.

[7] WILLIAMS R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning[J]. Reinforcement learning, 1992: 5-32.

[8] SCHULMAN J, WOLSKI F, DHARIWAL P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[9] KONDA V, TSITSIKLIS J. Actor-critic algorithms[J]. Advances in neural information processing systems, 1999, 12.

[10] MNIH V, BADIA A P, MIRZA M, et al. Asynchronous methods for deep reinforcement learning[C]//International conference on machine learning. 2016: 1928-1937.

Thanks!