Multi-Agent Reinforcement Learning

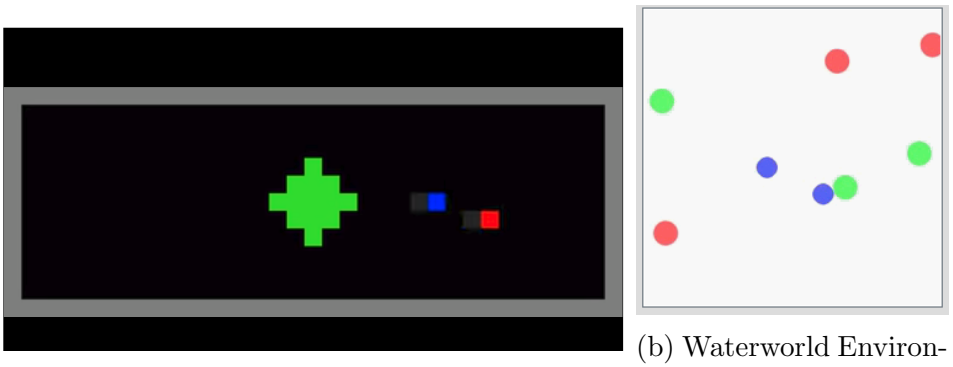
Course Project Proposal

Denis Osipychev*
Yizhi Fang†

*deniso2@illinois.edu
†yfang10@illinois.edu

1 Introduction

The recent advantages of Reinforcement Learning (RL) algorithms already demonstrated the ability of RL agents learn and perform many complex tasks previously attained by a human only. Agents involving deep RL have been successfully applied to many challenging problems from playing a range of Atari 2600 video games at a superhuman level to defeating a human world champion in game of Go. However, traditional reinforcement learning approaches such as Q-Learning are poorly suited to multi-agent problems. Actions performed by third agents are usually observed as transition noise that makes the learning very unstable[7]. Policy gradient methods, usually exhibit very high variance when coordination of multiple agents is required[6]. Nevertheless, multi-agent systems are finding applications in a high demand problems including resource allocation, robotic swarms, distributed control, collaborative decision making, data mining, etc. For instance, in task planning, identifying each resource with an agent may provide a helpful, distributed perspective on initially fully centralized system.



(a) Gathering Environment

(b) Waterworld Environment

Figure 1: Multi-Agent Environments

Being inspired by multi-agent RL (MARL) problems[5] in robotics and resource management, we want to start with an actor-critic algorithm which adopts an idea of using centralized critic decentralized actor. There are several implementations of that method, for instance, Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm proposed in Lowe's paper [6] utilizing individual DDPG policy networks for actor and shared Q-value network for critic. Artificial agents trained by deep multi-agent reinforcement learning to study the complex sequential social dilemmas have demonstrated feasibility of solution in complex virtual environments like Waterworld [1] shown in Fig. 1b and Gathering game shown in Fig 1a where two agents share world and collect apples to receive positive rewards[2]. We are going to start with a simplified environment like LunarLander modified for MARL purpose.

2 Environment

To minimize the effort of tuning other environments, we modified the environment that we used in one of the homeworks called LunarLander. LunarLander is a original OpenAI

GYM environment built using Box2D package [4]. In that environment, the agent can control a lander with a choice of discrete or continuous action sets. In discrete set, there are 4 actions - left and right engine's thrust and main engine thrust pointing down. In continuous version, the control is provided by a 2-dimensional vector controlling left/right thrust independently from main engine thrust. The observation space is given by the tuple of various parameters shown in Eq. 1. The reward system in the original environment given by continuous reward given by squared distance to landing zone and error in additional state parameters such as vector velocity, angular orientation and velocity, and discrete additional reward for successful landing or penalty for leaving the screen or crash as demonstrated in Eq. 2.

$$\text{State} = [X_{\text{err}}, Y_{\text{err}}, V_x, V_y, A_{\text{rot}}, W_{\text{rot}}, C_{\text{leg1}}, C_{\text{leg2}}] \tag{1}$$

$$\begin{aligned} \text{Reward} =& X_{\text{err}}^2 + Y_{\text{err}}^2 + V_x^2 + V_y^2 + A_{\text{rot}}^2 + W_{\text{rot}}^2 + \text{R}_{\text{step}} && (2)\\ \text{R}_{\text{step}} =& +100 \text{ if landed} && (3)\\ & -100 \text{ if crashed or left} && (4) \end{aligned}$$

To fit our needs, we modified the existing environment by adding the second lander that can be controlled independently. Both landers works completely parallel without collisions sharing state space and the landing zone. The only interaction between landers occurs in a new reward system that mixes rewards of both landers into one. That makes the problem very complex since the agents don't really understand if the change in the reward is made by themselves or the opponents.

$$\begin{aligned} \text{State} =& [[X_{\text{err}}, Y_{\text{err}}, V_x, V_y, A_{\text{rot}}, W_{\text{rot}}, C_{\text{leg1}}, C_{\text{leg2}}]_{\text{lander1}}, && (5)\\ & [X_{\text{err}}, Y_{\text{err}}, V_x, V_y, A_{\text{rot}}, W_{\text{rot}}, C_{\text{leg1}}, C_{\text{leg2}}]_{\text{lander2}}] && (6) \end{aligned}$$

$$\begin{aligned} \text{Reward} =& R_{\text{lander1}} + R_{\text{lander2}} + R_{\text{step}} \ \textit{if} \ (R_{\text{lander1}} R_{\text{lander2}}) \geq 0 && (7)\\ & R_{\text{step}} \ \textit{else} && (8)\\ R_{\text{step}} =& +100 \ \textit{if} \text{ both landed} && (9)\\ & -100 \ \textit{if} \text{ both crashed or left} && (10) \end{aligned}$$

The reward system has been modified several times during the main set of experiments in order to maximize the effect of training. We found that sum of rewards from two lander as shown in Eq. 7 may positively reinforce the wrong behavior explored by the agents that affects the whole training progress. To resolve the issue, we came up with an idea of providing the reward only when both agents doing well or both doing bad. In this case, it allows to reinforce the policy using a proper gradient of payoff and avoid the misleading reinforcement.

3 Models

Our model is based on GA3C by Babaeizadeh *et. al.*[3] (hybrid GPU/CPU version of Asynchronous Advantage Actor-Critic algorithm)[1] and adopts the idea of multi-agent decentralized actor, centralized critic approach where the critic is augmented with extra information about the policies of other agents, while the actor only has access to local information[6].

The policy gradient for each agent is described in Eq. 11: Each agent has its own policy $\pi_i(a_{t,i}|s_t, \theta)$ with regard of its own action $a_{t,i}$ and current state s_t but the advantage function $A(s_t, a_{t,1}, ..., a_{t,N}$ is shared by all agents. The advantage function is estimated by the temporal difference function δ_t (Eq. 12).

$$\nabla_\theta \eta_i(\pi_\theta) = E_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_i(a_{t,i}|s_t, \theta) A(s_t, a_{t,1}, ..., a_{t,N}) + \beta \sum_{t=0}^{T-1} \nabla_\theta H(s_t|\theta)] \tag{11}$$

$$\hat{A}(s_t, a_{t,1}, ..., a_{t,N}) = \sum_{t=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{12}$$

Because our LunarLander environment is continuous, the policy for each agent is chosen to be Gaussian distribution as shown in Eq. 13. Action mean $\mu_\theta(s_t)$ is a two hidden layers feedforward neural network of which output is 2-dimensional and standard deviation σ is given.

$$\begin{aligned} \pi_i(a_{t,i}|s_t) =& \frac{1}{\sqrt{2\pi}\sigma_{1,i}} \exp(-\frac{(a_{t,i}^{(1)} - \mu_{\theta,i}(s_t)^{(1)})^2}{2\sigma_{1,i}^2}) \\ & \cdot \frac{1}{\sqrt{2\pi}\sigma_{2,i}} \exp(-\frac{(a_{t,i}^{(2)} - \mu_{\theta,i}(s_t)^{(2)})^2}{2\sigma_{2,i}^2}) \end{aligned} \tag{13}$$

In addition to the policy gradient, we also trained the critic function using linear regression where the value is also a two hidden layers feedforward neural network of which output is a single value. By default, the number of hidden units of value network are twice of that in action mean network.

4 Results

As mentioned before, in order to compute the action mean network, we tried two strategies: (a) using 2 independent network for each agent's action mean with same state as input (so-called 2 actor) or (b) using the same network for both action means but each agent has different state input (so-called 1 actor). We expected to see similar performance between these two strategies because the two agents should have very similar policy when trained successfully; however, in contrast to what we thought, the single network for action mean (1 actor) doesn't perform well due to its simplicity (see yellow and purple curves in Fig. 2).

[1]Original code can be found at https://github.com/NVlabs/GA3C

The purple curve completely failed to train while the yellow curve also diverged at the end of train.

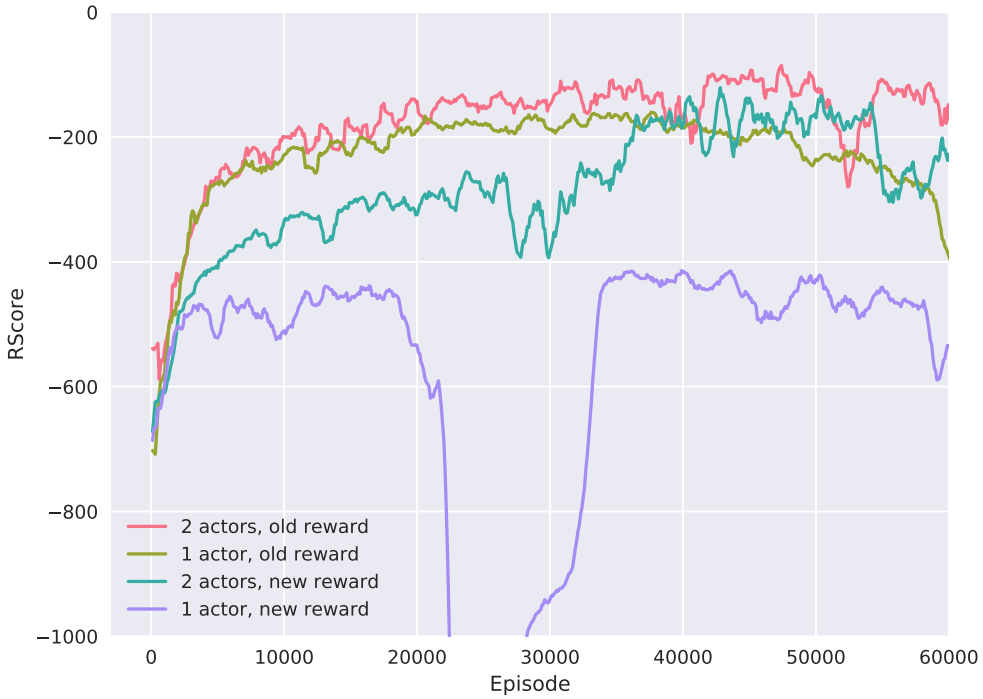


Figure 2: Learning curves with different actors and reward system

The old reward system has a higher payout than the new one therefore it reaching a similar score as with the new reward system actually means the old reward system was stuck in some local minimum where one of the agent quickly falls down to the target landing zone while the other one keeps hovering around in the air (Fig 3). The Fig. 2 shows the best model (blue curve) is using 2 independent network for each action mean with the new reward system.

In order to tune the hyper-parameters, we tested λ and β which is bias-variance trade-off and regularization factor respectively (Fig. 4). Unlike the single agent problem, the λ plays a more important role than β such that the model failed to train when λ is slightly away from 1. The best λ value agrees with our approximation of the advantage function: when $\lambda = 1$ the advantage function takes all future rewards into consideration and yields much lower bias (Eq. 14).

$$\hat{A}(s_t, a_t) = \sum_{t=0}^{\infty} \gamma^l r_{t+l} - V(s_t) \tag{14}$$

We also tested the impacts of number of hidden units in action mean network (the number of hidden units in value network is simultaneously twice of that in value network) as shown in Fig. 5. The optimal number is 64 among the three numbers we tried.

Finally, our best model of two agents with all tuned hyper-parameters is shown in Fig. 6.

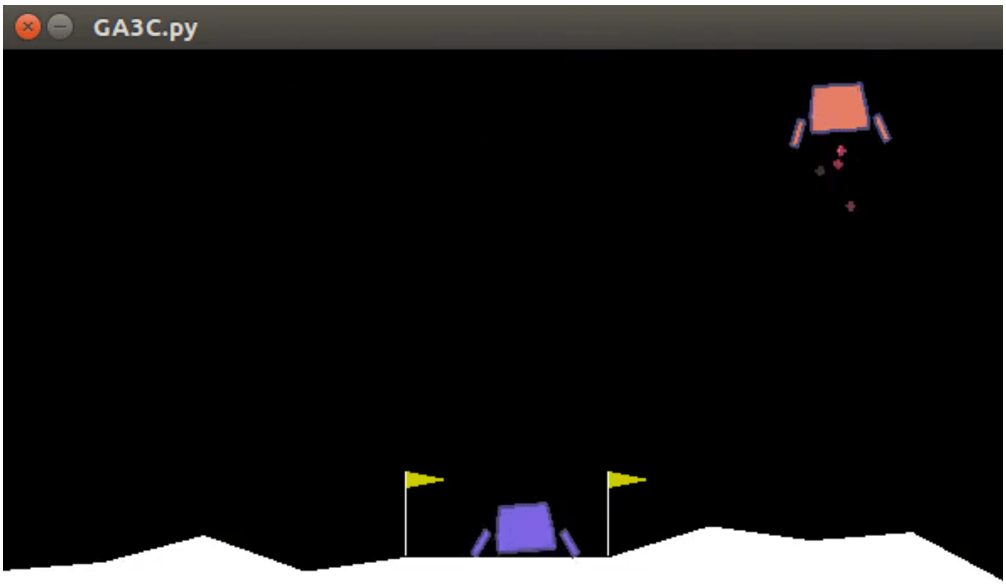


Figure 3: Failed policy with old reward system

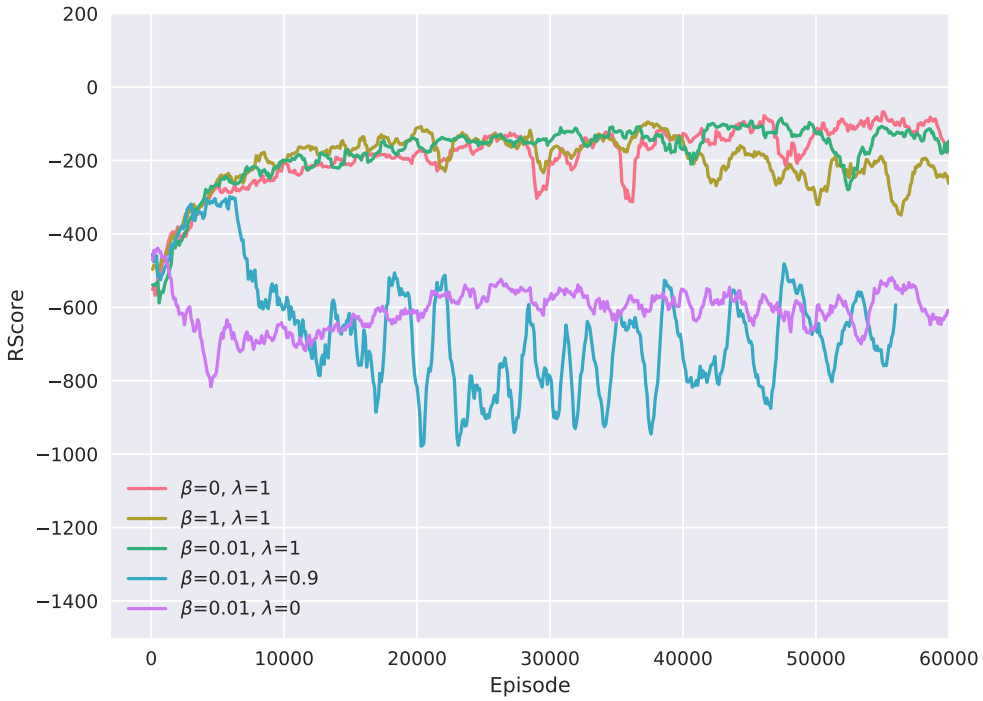


Figure 4: Learning curves with different λ and β

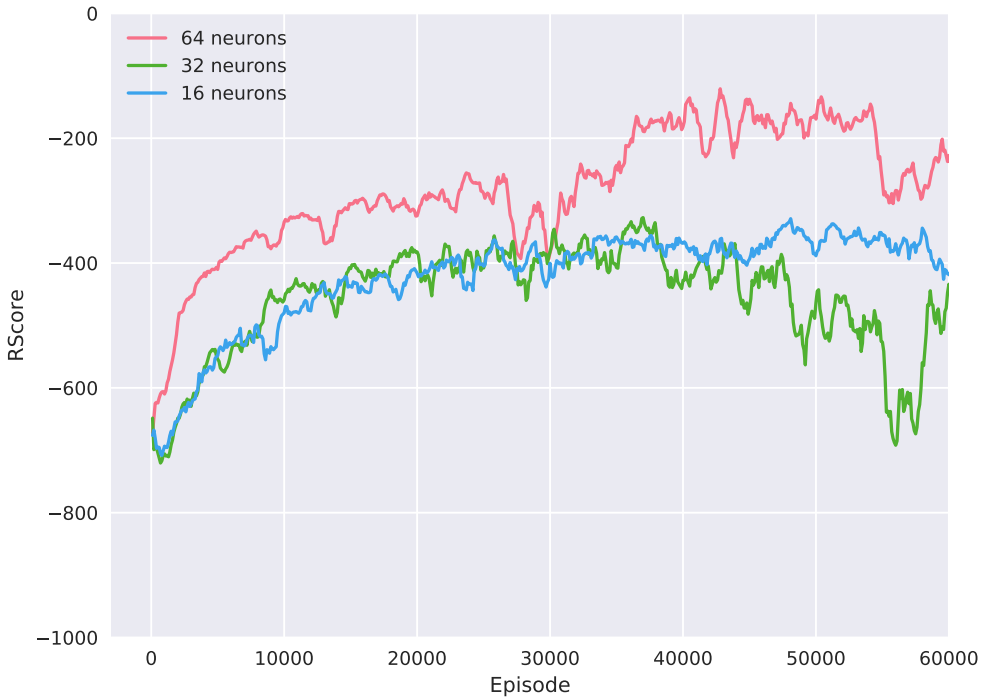


Figure 5: Learning curves with different number of hidden units in action mean network

Figure 6: Final successful policy best model and tuned parameters

5 Conclusion

In this project, we demonstrated that the complex MARL task may be solved using the shared critic approach. The choice of the algorithms for actor is not limited to DDPG algorithm demonstrated in the literature, but any policy based network would provide a feasible solution. This work demonstrates hyper-parameters that achieved the best rate of convergence for our problem. Local policy actor with shared critic is a possible solution for MARL type of problems that are very complex problems due to noisy and unstable solutions. This result fits the similar difficulties discussed in the original paper experiencing noisy transitions that caused high variance in solutions.

Success of training is impossible without careful reward system planning. Adding more agents causing additional very strong sub optimal policies that attracts the algorithm to converge to the wrong policy. Therefore, each task requires careful thinking of the reward and stopping criteria to allows the convergence. However, those changes are environment specific and cannot be easily extended to other environments. Each task requires much fine tuning and not transferable from problem to problem.

References

[1] A pytorch implementation of maddpg. https://github.com/xuehy/pytorch-maddpg.

[2] Understanding agent cooperation. https://deepmind.com/blog/understanding-agent-cooperation/.

[3] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz. GA3C: gpu-based A3C for deep reinforcement learning. *CoRR*, abs/1611.06256, 2016.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[5] L. Buoniu, R. Babuŝka, and B. D. Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE*, 38, 2008.

[6] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *CoRR*, abs/1706.02275, 2017.

[7] A. Nowé, P. Vrancx, and Y.-M. De Hauwere. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning*, pages 441–470. Springer, 2012.