ECE 276 final report
Advantage Actor Critic (A2C) with Experience Replay

Chun Hu[1] and Po-Jung Lai[1] and Chih-Hui Ho[2]

***Abstract*— We propose a simple and lightweight framework for deep reinforcement learning that uses synchronous gradient descent for optimization of deep neural network controllers with experience replay. We incorporate synchronous, deterministic variant of reinforcement learning algorithms, named A2C, with experience replay and show that parallel actor-learners have a stabilizing effect on training. Experiment are conducted by solving games in OpenAI gym with the proposed algorithm. Moreover, different experience replay buffer size and sampling techniques are compared. Our result outperforms the baseline A2C without experience replay. Our code[1] are released for future research.**

I. INTRODUCTION

The goal of reinforcement learning (RL) is to learn the a policy by using an agent to interact with the environment. A better policy indicates higher reward will be accumulated [1]. Recently, reinforcement learning has been shown to perform extraordinarily well on a wide range of Atari games[2] and physical simulations[3], [4].

In the literature of reinforcement learning, several methods are proposed to improve the policy and those methods can be coarsely categorized into 2 categories. The first category finds the optimal policy indirectly through the surrogate optimal value function, such as Q-learning[5] and SARSA[6]. The other category, named policy iteration, optimizes the policy directly without any objective value function. Actor-critic[7] method is a well-known method that generalizes policy iteration. It iterates between the policy evaluation process and the policy improvement process. Two modules, an actor module and a critic module, are interacting with each other. The actor module aims at improving the current policy, while the critic module evaluates the current policy.

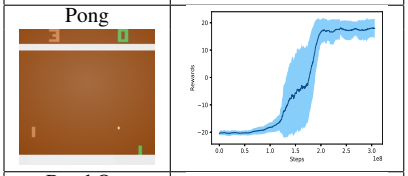
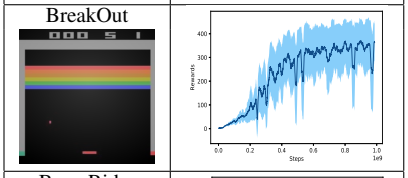
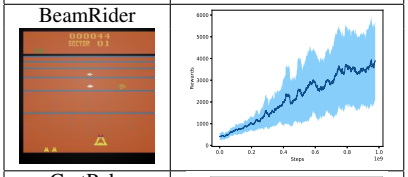
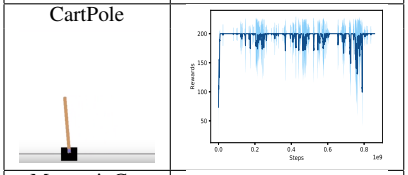
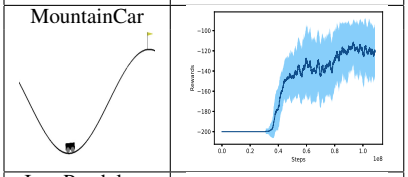
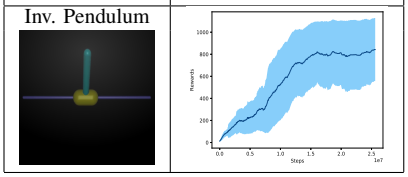
A recent paper advantage actor-critic method [8] discussed an alternative way to train the system by using synchronous gradient descent for optimization of deep neural network controllers. The proposed methods can improve stability of both the on-policy and off-policy algorithms with parallel actor learners during the training process.

In addition, inspired by paper Sample Efficient Actor-Critic with Experience Replay (ACER) [9], we would like to investigate the possibility to incorporate experience replay into A2C algorithm. Moreover, [8] also mentions that adding experience replay would possibility improve the data efficiency by reusing old data.

As a result, in this final project, the advantage actor-critic(A2C) method is implemented as our baseline. Several Atari games and traditional control problems were experimented using the algorithm. We extend A2C algorithm with experience replay technique and hopefully, it will boost and stabilize the performance of the network.

TABLE I

REWARD PLOTS FOR DIFFERENT GAMES

Game	Reward
Pong	
BreakOut	
BeamRider	
CartPole	
MountainCar	
Inv. Pendulum	

II. BACKGROUND

Reinforcement learning (RL) has performed effectively and efficiently on different tasks, including Atari games and

*This work was the final project of UCSD ECE 276C course

[1]Department of Electrical and Computer Engineering {chh281, polai}@eng.ucsd.edu

[2]Department of Computer Science chh279@eng.ucsd.edu

[1]https://bitbucket.org/peter_chun_hu/276_hw4/src/master/

traditional control problems. Although RL algorithms has good performance, its training procedure is unstable. As a result, experience-based RL methods are proposed to solve the unstable training procedure.

Experience replay is actually a useful tool for improving sample efficiency and, as we will see in our experiments, state-of-the-art deep Q-learning methods [10] [11] have been up to the efficient techniques on Atari by boosting reward.

The A2C method was inspired by the Asynchronous Advantage Actor Critic method (A3C). The algorithm combines a few key ideas as following. First, an updating scheme that operates on fixed-length segments of experience (say, 20 timesteps) and uses these segments to compute estimators of the returns and advantage function. Second, architectures that share layers between the policy and value function. Third, the networks are updated asynchronously.

Advantage actor-critic (A2C) method [8] proposed to train the network in synchronous way and apply to different RL learning algorithms, including SARSA, Q-learning and actor critic. The motivation of proposed method is to solve the instability when training the network in the same thread, due to the high correlation between the training data. This parallel training scheme also decorrelates agents data leading to a more stable outcome.

Many of these methods are restricted to continuous domains or to very specific tasks such as playing Go. The existing variants applicable to both continuous and discrete domains, such as the on-policy asynchronous advantage actor critic (A3C) of [12], are sample inefficient.

ACER [9] capitalizes on recent advances in deep neural networks, variance reduction techniques, the off-policy retrace algorithm [13] and parallel training of RL agent s[12]

Although adding experience replay techniques can alleviate the problem, it constraints the training procedure to be off-policy and the replay buffer will increase significantly as more experience is needed, which results in high memory usage.

III. METHOD

A. Project Idea

The baseline implementation is to replicate the A2C algorithm. Since the A2C only depends on the on-policy update, the algorithm discards all the trajectories after updating networks. To further extend the stability and capability of the network, we implement the algorithm Sample Efficient Actor-Critic with Experience Replay [9] by adding the experience replay to the original A2C algorithm. We wonder that the old trajectories may be still useful as for updating the networks. The experiment results are shown in Section IV.

B. Proposed Architecture

Advantage actor-critic (A2C) method [8] proposes to train the network in synchronous way and apply to different RL learning algorithms, including SARSA, Q-learning and actor critic. The motivation of proposed method is to solve the instability when training the network in the same thread, due to the high correlation between the training data. Although adding experience replay techniques can alleviate the problem, it constraints the training procedure to be off-policy and the replay buffer will increase significantly as more experience is needed, which results in high occupation of computing and memory resources.

The proposed method solves the problem by introducing multiple agent to interact with the environment in parallel at different threads. Every agent is randomly initialized, so the data correlation is reduced between experience from different agents. The method [8] allows on-policy learning without occupying great amount of computing resources and stability during training procedure is improved.

Our proposed architecture is similar to [8] as described in Algorithm 1. For each thread, a replay buffer is created to store the past experience of the agent. Again, the data correlation is minimized between agents, but the past experience of same agent are utilized. Each tuple in the replay buffer is a single trajectory that the agent has explored, while each trajectory are composed of several steps before reaching the terminated state. The reward for each step and the return of the entire trajectory are stored with the evaluation metric described in Section III-C

The proposed experience replay sampling procedure is similar to [9]. The latest trajectory is always sampled to ensure that the network update always take the latest trajectory into consideration. If the minibatch only contains 1 trajectory, which is the latest one, the proposed architecture then becomes the typical A2C algorithm, which is an on-policy algorithm. With appropriate number of past experience added, our proposed algorithm then become an off-policy algorithm without ignoring the latest trajectory.

Since we need to handle different environments in Open AI gym, different games come with different reward function which may lead to the negative reward. We applied the softmax to the reward function when we need to prioritize the replay buffer.

C. Evaluation Metric

Assume the network is trained with T threads. All the returns for each thread are recorded during the training procedure. When the training is done, the returns of each trajectory for each thread is sorted according to the time it is generated. The typical running average methods is applied on the sorted reward. The results are shown in Table II. The performance of our result is comparable to the state-of-the-art result.

IV. RESULTS

A. Baseline Architecture

We implement the A2C algorithm using OpenAI resource[2]. Referring to the OpenAI docker file, we have regenerated the docker file[3] for our experiments. Three Atari games, Pong, Breakout and BeamRider, and 3 control

[2] https://blog.openai.com/baselines-acktr-a2c/

[3] https://hub.docker.com/r/fraserlai/276_project/ Tag: gym_10_TA_v6

Algorithm 1 Advantage actor-critic with experience replay

//Assume global shared parameter vectors θ and θ_v and global shared counter T = 0
// Assume thread-specific parameter vectors θ' and θ_v
Initialize thread step counter $t \leftarrow 1$
for $T < T_{max}$ **do**
Reset gradient: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
Synchronize thread specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
$t_{start} = t$
Get state s_t
while not terminate or $t - t_{start} < t_{max}$ **do**
Perform a_t according to policy $\pi(a_t|s_t;\theta')$
Receive reward r_t and new state s_{t+1}
$t \leftarrow t + 1$,
$T \leftarrow T + 1$
end while
Add the trajectory into replay buffer
$X = 0 \begin{cases} 0, & \text{for terminal} s_t \\ V(s_t, \theta'_v), & \text{for non-terminal} s_t \end{cases}$
for $i \in t-1, \ldots, t_{start}$ **do**
$R \leftarrow r_i + \gamma R$
Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \Delta_{\theta'} log\pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
end for
Replay the mini-batch and update the network
end for

simulations, cart pole, mountain car and invert pendulum are experimented. The reward plot are shown in Table I. We have successfully run the algorithm on different environments and the videos are provided in the supplementary material.

B. Proposed Architecture

Different from the game experimented in the baseline architecture, we explore 3 extra environments to test the proposed architecture, including humanoid, half-cheetah and hopper. The hyperparameters setting are identical to the baseline architecture.

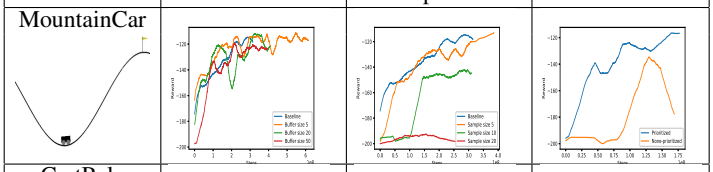
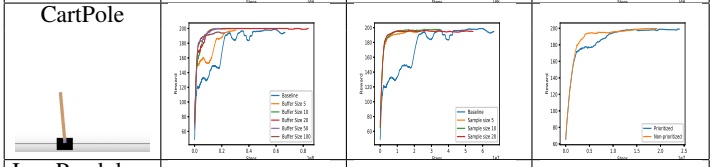
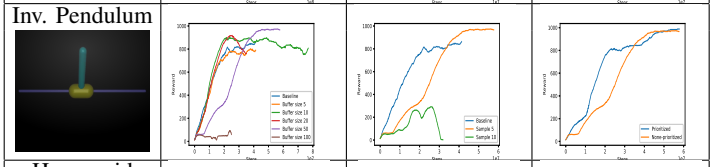
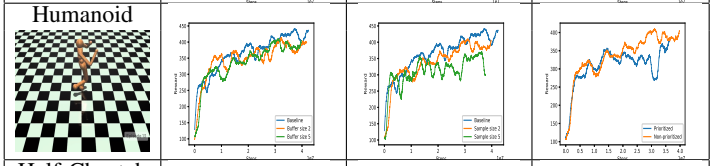
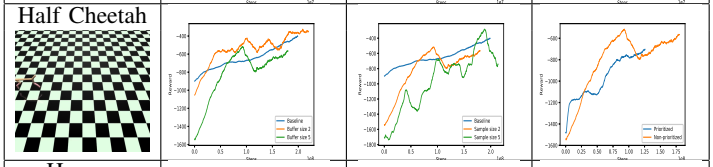
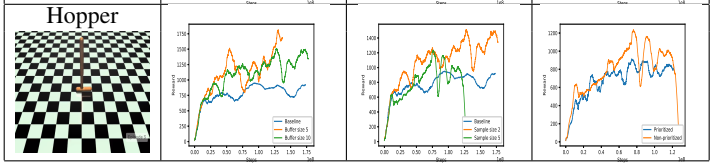
The proposed architecture are experimented with different replay buffer settings, including 3 different factors. While a single factor is altered, the others are fixed in the experiment.

1) **Buffer size** This is the maximum number of past trajectories stored in the replay buffer. Note that the latest trajectory is always sampled.
2) **Sample size** This is the number of past trajectories sampled from the replay buffer. Note that the latest trajectory is always sampled.
3) **Prioritized** Assume that the importance of a certain trajectory is determined by its return. With each trajectory weighted by its return, the trajectory with higher return has higher probability to be sampled. Note that the weighted of probability is measure by using softmax function.

*a) **Experience Replay Buffer Size Comparison**:* Different replay buffer sizes are experimented in this experiment, as illustrated in the first column of Table II. With appropriate replay buffer size, the performance with replay

TABLE II

REWARD PLOTS FOR DIFFERENT ENVIRONMENTS UNDER DIFFERENT EXPERIENCE REPLAY SETTINGS. (A) DIFFERENT BUFFER SIZE (SAMPLE SIZE IS FIXED) (B) DIFFERENT SAMPLE SIZE (BUFFER SIZE IS FIXED) (C) SORT THE REPLAY BUFFER BY EACH TRAJECTORY RETURN (PRIORITIZED VS NON-PRIORITIZED)

Game	Comparison		
	Buffer size	Sample size	Prioritized
MountainCar			
CartPole			
Inv. Pendulum			
Humanoid			
Half Cheetah			
Hopper			

buffer outperforms the baseline network. It is observed that the received reward is sensitive to the replay buffer size. The result shows that large buffer size will harm the performance significantly, because more unrelated past experiences are likely to be sampled.

For the traditional control and mountain car environment, the appropriate buffer size are around five to twenty. With the replay buffer, these three problem can converge more stable than baseline (A2C).

Moreover, we also observe that the appropriate buffer size of humanoid, hopper and half-cheetah environments are smaller than the other three environments. This shows that past experiences are not helpful in those tasks.

*b) **Experience Replay Sample Size Comparison**:* This experiment shows the relationship between reward and sample size under fixed buffer size, as shown in the second column of FigureII. As the sample size increases, more past

experience are sampled. Since some of the old trajectories deviate too far from the trajectory generated from current policy, sampling too many past experience will degrade the performance of the network.

For MountainCar and Inverted Pendulum problem, adding small replay buffer can make the converge more stable after updating certain iterations. The reward grows faster than the baseline. For the CarPole problem, any sample size of replay buffer makes the result converge much faster and more stable than the baseline model.

Again, for humanoid, hopper and half-cheetah environments, the reward are more sensitive to the number of sample size. For hopper environment, with small experience, the reward can surpass the baseline model and keep growing. However, the reward of half-cheetah environment is more unstable as we adding little number of experience replay.

c) ***Prioritized vs Non-prioritized Experience Replay****:* This experiment investigates the effect of prioritizing the importance of different trajectories, as demonstrated in the third column of FigureII. Interestingly, we found the prioritized sampling improves the performance of the agent in mountain car, cart pole and invert pendulum environments, but deteriorates the performance on the rest.

V. DISCUSSIONS AND CONCLUSIONS

In this project, we reproduce the A2C result using the OpenAI resources as our baseline. 6 different environments are tested, which include Atari games, control problems and physical simulation. Moreover, A2C algorithm is extended by adding the replay buffer. We borrow the idea from [9] and combine the concept with A2C algorithm. Experiments show that A2C with experience replay, can make the convergence speed much faster and more stable than the baseline model (A2C). With the advantage of A2C and experience replay, we can boost the training speed in many environment in OpenAI gym. Moreover, we compare the effect of different buffer size and sample size settings. Many testing environment are sensitive to the size of replay buffer and both large buffer size and sample size hurt the result of the experiment. With adequate experience replay, our algorithm can surpass the performance from the baseline. We also demonstrate two new environment, half-cheetah and hopper, which have not tested in both [12] and [9] before. Our code and docker are released future research.

VI. FUTURE WORK

In [14], they show that applying Kronecker-factored approximate curvature with trust region on on-policy actor-critic methods can achieve higher rewards and improve the sample efficiency. Therefore, we would like to apply trust region on off-policy methods like our framework and to see whether it will improve the performance of off-policy methods or not.

ACKNOWLEDGMENT

We thank the assist from the instructors in ECE 276C for setting up the GPU cluster and providing the suggestion for the final project.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[3] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015. [Online]. Available: http://arxiv.org/abs/1509.02971

[5] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, 1992, pp. 279–292.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[7] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K. Müller, Eds. MIT Press, 2000, pp. 1008–1014. [Online]. Available: http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[9] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *CoRR*, vol. abs/1611.01224, 2016. [Online]. Available: http://arxiv.org/abs/1611.01224

[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[11] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[12] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[13] R. Munos, T. Stepleton, A. Harutyunyan, and M. Bellemare, "Safe and efficient off-policy reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 1054–1062.

[14] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Advances in neural information processing systems*, 2017, pp. 5285–5294.