Compound Asynchronous Exploration and Exploitation

Jie Bai[1,a], Li Liu[2,b], Yaobing Wang[1,c], Haoyu Zhang[3,d], Jianfei Li[1,e]

[1]Beijing Key Laboratory of Intelligent Space Robotic Systems Technology and Applications, Beijing Institute of Spacecraft System Engineering, Beijing, China

[2]College of Engineering, Peking University, Beijing, China

[3]Science and Technology on Space Intelligent Control Laboratory, Beijing Institute of Control Engineering, Beijing, China

[a]baijierobot@163.com, [b]m55852@126.com, [c]iamwyb@163.com, [d]Haoy_Zhang@163.com, [e]lijianfei_hit@foxmail.com

Keywords: Deep Reinforcement Learning, Exploration and Exploitation, Asynchronous Methods

Abstract: Data efficiency has always been a significant key topic for deep reinforcement learning. The main progress has been on sufficient exploration and effective exploitation. However, the two are often discussed separately. Profit from distributed systems, we propose an asynchronous approach to deep reinforcement learning by combining exploration and exploitation. We apply our framework to off-the-shelf deep reinforcement learning algorithms, and experimental results show that our algorithm is superior in final performance and efficiency.

1. Introduction

Deep Reinforcement Learning (DRL) has been demonstrated on a series of challenge domains, from games [1-2] to robotic control [3]. However, there exist a central challenge to contribute DRL algorithms on real-world platforms. The main-stream DRL algorithms remain to be model-free DRL algorithms, which suffer from high sample complexity. Either relatively simple tasks or complex behaviors with high-dimensional observations might require millions of steps of data collection or even substantially more [4].

Effective use of computational resources [5] and more powerful and robust models [6] have recently achieved some successes for data efficiency. Deep learning frameworks such as DistBelief [5] can make large-scale neural networks efficient to implement, which place massive amounts of data on a distributed learning system. Some progress has been made on effective use of computational resources such as Gorila [7], A3C [8], GA3C [9] and Ape-X [10].

Despite the profit from the parallelism, the comprehensive utilization of available resources becomes a novel perspective, such as exploration and exploitation. Research on exploration and exploitation has always been decoupled, with some research focused on exploration [11-12] and others on exploitation [13]. More details can be seen in Section III. A combining and/or unified approach to exploration and exploitation remains to be ambiguous.

In this paper, we propose an asynchronous approach to deep reinforcement learning by combining exploration and exploitation, to generate more data and learn efficiently. Unlike general methods of distributed framework for deep learning algorithms focusing on parallelizing the computation of the models [5] or decoupled actors and learners [10], we contribute distributed systems on data generation.

We investigate the application of our framework integrated with off-the-shelf deep reinforcement learning algorithms such as Deep Q-Networks (DQN) [14] and Deep Deterministic Policy Gradient (DDPG) [15]. Experiments show this combination of exploration and exploitation is available to many types of both continuous control tasks and discrete environments. Our results indicate much faster training time and better final performance.

2. Background

In this section, we formulate the standard reinforcement learning problem, and introduce the necessary algorithmic foundations on which we demonstrate the methods for this work.

2.1 Reinforcement learning

We consider a reinforcement learning process where an agent interacts with an environment modeled as a Markov Decision Process (MDP) [16]. An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, R, \gamma, P)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, R is the reward function, γ is the discount factor and P is the transition dynamics. At each state s, the agent selects an action a according to the policy π, consisting of the stochastic distribution $\pi(s|a)$ or a deterministic mapping $a = \pi(s)$, transitions to new state s' according to the dynamics $P(s'|s, a)$, and receives a reward $R(s, a)$. Here, the goal of reinforcement learning is to learn a policy maximizing the expected discounted reward over the agent's trajectory $\tau = (s_0, a_0, s_1, a_1, \dots)$.

2.2 Actor-Critic framework

The approach to reinforcement learning problems can be available to two alternative methods. The first one *value function approaches* (Critic-only), is an estimate of the expected future reward according to the policy π, where the value-action function Q^{π} is defined as:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[\textstyle\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)]$$

The policy is implicitly derived from Q^{π} as $\pi(s) = \text{argmax}_{a\in\mathcal{A}} Q^{\pi}(s, a)$. The other available method is *policy search* (Actor-only). In the policy search methods, policies are represented by a variety of approaches and can be directly optimized to maximize the cumulative reward, given:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta}(\tau)}[r(\tau)] = \int_{\tau\sim\pi_{\theta}(\tau)} \pi_{\theta}(\tau) r(\tau)\, \mathrm{d}\tau$$

Where $r(\tau)$ represents the total reward of the trajectory. Actor-Critic framework [17] is composed of value function approaches and policy search methods, where the Critic estimates the value function according to the temporal difference (TD) learning, while the Actor updates the policy parameters according to the learned value function.

3. Related Works

3.1 Off-policy DRL Methods

We demonstrated that, mature off-policy algorithms, such as Deep Deterministic Policy Gradient (DDPG) [15] and Normalized Advantage Function (NAF) [4], can achieve a well-used data efficiency. In contrast, some general-purpose on-policy DRL algorithms, such as Trust Region Policy Optimization (TRPO) [18] and Asynchronous Advantage Actor-Critic (A3C) [8], require new samples to be collected for each learning step on the policy, which occurs to the data inefficiency. In this work, we only focus on off-policy methods. Off-policy can allow to learn based on data according to the arbitrary policy, where we can demonstrate experience replay on improving data efficiency. Furthermore, Prioritized experience replay [19] extends the experience replay, which show its superiority to contributing the agent's final performance, such as UNREAL [20], DQfD [21], and Rainbow [22].

3.2 Distributed DRL framework

Parallel computation is not uncommon in the field of machine learning. The idea of distributed system for training large-scale neural networks has recently been induced into deep reinforcement learning, which benefits from the parallelism architecture DistBelief and the learning architecture (Actor-Critic). Inspired by these, Gorila based on multi-actors for data collection and multi-learners for parameter update, A3C based on multi-cores for multi-actors, GA3C based on GPU for multi-actors [23] have been proposed, and more details can be referred to Ape-X.

3.3 Exploration and exploitation

A plethora of methods have been proposed to improve the trade-off exploration and exploitation, however, they are limited to the one-side. Some of the art-of-the-state exploration algorithms include Noisy networks, Parameter space noise, which represent the model as neural networks and add the noise to parameter space, while utilize parameter perturbations for more efficient exploration. On the contrary, we can think about the exploitation, such as Deep Q-learning from Demonstrations (DQfD) [13], which leverages even very small amounts of demonstration data to massively accelerate learning. This method can inspire us to make use of these pre-prepared high-quality sampling trajectories to accelerate the training speed of the model. Moreover, we can also utilize the data that well-trained models proposed previously can provide the test data for exploitation. Because we have the baseline algorithms offered by OpenAI, we can stand on the shoulders of the giants.

4. Algorithm

In this work, our contribution is a compound method of parallel computation, effective exploration and prior exploitation, and we propose a novel framework called Compound Asynchronous Exploration and Exploitation (CA2E). This paper investigates how CA2E algorithm can be effectively combined with off-the-shelf DRL algorithms to improve the final performance and data efficiency.

Similar to Gorila, we decompose the deep reinforcement learning algorithms from both architecture and learning, where we decouple acting from learning and adopt the asynchronous methods. The first component acting (policy evaluation), interacts with the environment, generates its own trajectories of experience and stores the data in a memory buffer. The second component learning (policy improvement), samples a minibatch of experience (off-policy algorithms) from the memory and update the network parameters, which shows in Figure 1.

In the DistBelief, both the actors and the learners can be possessed of multiple distributed workers. We adopt the GA3C to our architecture, where multiple actors run on multi-core CPUs to generate the experience, while a single learner runs on a GPU to sample from the memory buffer and update network parameters.

In Ape-X, it prioritizes the experience, to sample the most useful data more often. Remarkably, it still adopts ϵ-greedy to collect the data, by giving the different actors different exploration policies. In our work, we decompose the actors into nature actors and prior actors. Prior actors can make use of these pre-prepared high-quality sampling trajectory to sample the most useful data.

In general, since data generation can hardly discover the high prioritized data, we propose, prior actors leverage demonstration data from trained models, to accelerate the training speed of the model.

By off-policy methods, we can demonstrate experience replay on improving data efficiency, and our methods can improve data efficiency from two aspects: data collection and data sampling. We make further use of CA2E to collect the data from many distributed actors, by giving the different actors different exploration/exploitation policies, sampling prioritized data from the experience replay. Pseudocode for the actors, prioritized memory replay and the learners is shown in Algorithms 1, 2 and 3.

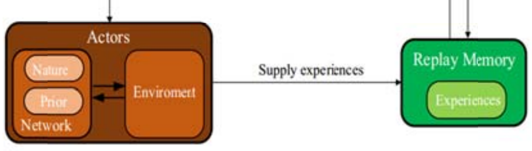


Fig. 1. The CA2E architecture in a container with the Multi-Actors-Single-Learner (MASL) system: multiple actors, generate exploration data (nature actors) and exploitation data (prior actors) from the interaction with the environment, and add them to a shared experience buffer. The learner samples from the experience replay memory and updates the network parameters. Meanwhile, the

actors' network parameters are updated from the learner periodically.

```
Algorithm 1.1 Prior Actor
Given the baseline average reward $\bar{r}$ of current prior actor
for episode = 1 to M do
    Receive an initial state $s_1$
    for t = 1 to T do
        Select the action $a_t = \pi(s_t)$
        Execute the action $a_t$ and observe the reward $r_{t+1}$ and the next state $s_{t+1}$
        Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in LocalBuffer
    end for
    Periodically receive the average reward $r$ from Learner
    if several times $r \geq \alpha\bar{r}$ $(\alpha > 1)$ then
        Replace the current prior actor with a new nature actor
    end if
end for
```

```
Algorithm 1.2 Nature Actor
for episode = 1 to M do
    Receive an initial state $s_1$
    Initialize network parameters $\theta_1$
    for t = 1 to T do
        Select the action $a_t = \pi(s_t)$
        Execute the action $a_t$ and observe the reward $r_{t+1}$ and the next state $s_{t+1}$
        Store $(s_t, a_t, r_{t+1}, s_{t+1})$ in LocalBuffer
    end for
    Periodically receive network parameters $\theta_t$ from Learner
end for
```

```
Algorithm 2 Prioritized Memory Replay
if LocalBuffer's capacity $\geq B$ then
    Get batch data $B$ of multi-step transitions $\tau$ from LocalBuffer
    Calculate priorities for experience $p$ of buffered data $\tau$
    Add prioritized experience $(\tau, p)$ to Replay Memory
end if
```

```
Algorithm 3 Learner
for episode = 1 to M do
    Initialize network parameters $\theta_1$
    for t = 1 to T do
        Sample a prioritized batch of transitions $(id, \tau)$ from Replay Memory
        Learn and update the network parameters $\theta_t$
        Calculate and update priorities for experience $p$
        Periodically remove old experience from replay memory
    end for
end for
```

5. Experiments

5.1 AC2E-DQN on Discrete Environments

The CA2E architecture we propose can be integrated with off-the-shelf deep reinforcement learning algorithms. First, we combined it with DQN on Atari using the standard reinforcement learning benchmark.

Before we formally introduce the experiment setup, let's review DQN algorithm briefly. We adopt several settings in the state-of-the-art DQN algorithm (Rainbow): double Q-learning and multi-step learning. We can compute the loss function with:

$$L(\boldsymbol{\theta}) = \mathbb{E}_{\pi}[\tfrac{1}{2}(\textstyle\sum_{i=0}^{n-1}\gamma^{i} R_{t+i} + \gamma^{n} q(s_{t+n}, \underset{a}{\operatorname{argmax}}\, q(s_{t+n}, a; \boldsymbol{\theta}); \boldsymbol{\theta}^{-}) - q(s_t, a_t; \boldsymbol{\theta}))^2]$$

Where the networks are represented by the function approximator $q(\cdot\,,\,\cdot\,;\boldsymbol{\theta})$, and $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{-}$ denotes parameters of the behavior network and the target network, respectively.

Here we show the setups of reinforcement learning, and deep networks' setups are in the appendix A. First, we use a single machine with multi-core CPU (8 cores) to run different actors for a variety of data generation, where 5 nature actors use ϵ-greedy policies with different values ϵ for an abundant exploration in the environment, while 3 prior actors use different pre-training DQN algorithms to supply effective experiences for exploitation.

Secondly, we set shared experience replay memory. Q-learning methods, as the off-policy methods, can be available to experience replay with prioritization mechanism. For better data efficiency, we abandon the approach to waiting for the learner to update priorities, we may benefit from the computation of asynchronous actors. This means that with more actors, experiences in the replay memory are more closed to the latest learner being optimized, and the network parameters are more similarly generated by policies with the newest network parameters (on-policy).

Thirdly, we set the learner. In principle, both actors and learners can be set by multiple distributed workers. Limited by the hardware equipment, we cannot know the effect of parallel training multiple GPUs, however, we trust the demonstration of Ape-X on multiple actors with single learner (MASL). Updated network parameters can be communicated to the actors by the asynchronously response of the learner.

Finally, we describe the environments. We evaluate CA2E-DQN by conducting experiments on 2 Atari Environment. More details about the environments can be found in Appendix B.

5.2 AC2E-DDPG on Continuous Control Tasks

The setting of AC2E-DDPG is basically similar to that of AC2E-DQN, but according to Actor-Critic algorithms, the model is represented as a separate actor-network with a critic-network, where the parameters of two networks are ω and θ, respectively. We represent the actor-network (policy network) to output an action $A_t = \pi(S_t, \theta)$, and the parameter update uses policy gradient descent on the estimated Q-value, and depends on the following target function:

$$\max_{\theta} \mathbb{E}_{\pi}[q(S_t, A_t, \boldsymbol{\omega})]$$

While we represent critic-network (Q-network) to output an estimate of an action-value function $q(S_t, A_t, \boldsymbol{\omega})$, and the parameter update is similar to DQN. Here we use a multi-step bootstrap target. The loss function can be written as:

$$\min_{\omega} \mathbb{E}_{\pi}[\tfrac{1}{2}(G_t - q(S_t, A_t, \boldsymbol{\omega}))^2]$$

Where multi-step return can be denoted as:

$$G_t = \textstyle\sum_{i=0}^{n-1}\gamma^{i} R_{t+i} + \gamma^{n} q(s_{t+n}, \pi(s_{t+n}, \boldsymbol{\theta}^{-}); \boldsymbol{\omega}^{-})$$

Other Settings are the same as CA2E-DQN, and the setting of deep network is referred to appendix A. In addition, we setup the benchmarking performed in two continuous control domains, which are implemented in the MuJoCo physics simulator (Todorov et al. 2012). An introduction to the environment can be found in Appendix B.

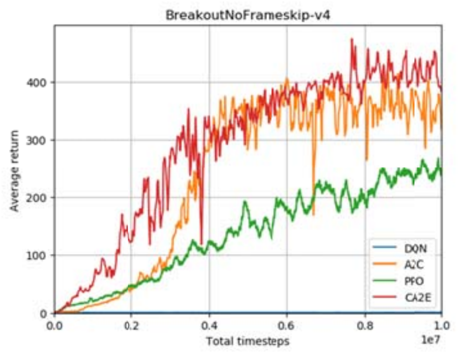
6. Results

To showcase the performance of CA2E on high-dimensional continuous control tasks and discrete environments, we have trained on a set of Atari and MuJoCo problems involving Breakout, Pong, Hopper and Reacher. See Figure 2 for the mean returns of Atari environments, and Figure 3 for the mean returns of MuJoCo environments with different algorithms.

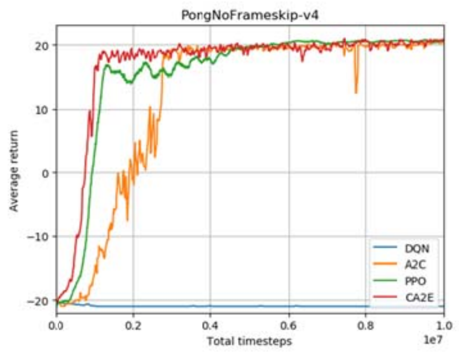
As can be seen from Figure 2, CA2E shows a faster improvement in the average return than the best of the three baseline algorithms (A2C), reflecting the prior actors' role in improving data efficiency. Limited by the training cycle and the performance of the workstation, there are several improvements in the final performance when the algorithm converges.

On the MuJoCo environments, we also consider DDPG, A2C and PPO2 with 10M frames, respectively. As can be seen from Figure 3, compared with the best algorithm (PPO2) among the three baseline algorithms, CA2E shows a faster improvement in the average return, and a certain improvement in the final performance when the algorithm converges.

Due to time and energy, we have not tested more algorithms and tasks for horizontal and vertical comparison, which is the future work of improving and perfecting the CA2E algorithm.

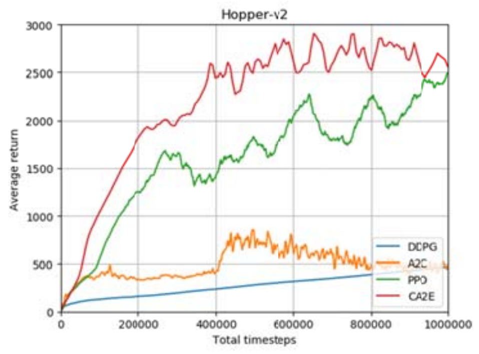


(a) BreakoutNoFrameskip-v4

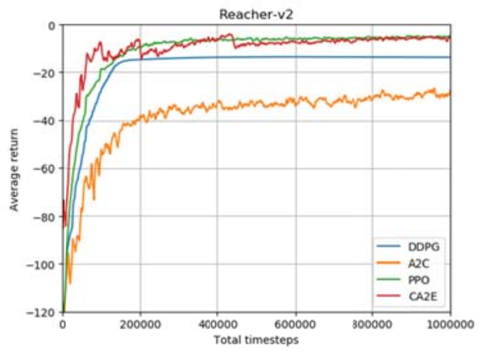


(b) PongNoFrameskip-v4

Fig. 2. Comparison of several algorithms on several Atari environments, training for ten million timesteps. (a) BreakoutNoFrameskip-v4 (b) PongNoFrameskip-v4



(a) Hopper-v2



(b) Reacher-v2

Fig. 3. Comparison of several algorithms on several MuJoCo environments, training for one million timesteps. (a) Hopper-v2 (b) Reacher-v2

7. Conclusion

In this paper, we propose an asynchronous approach to deep reinforcement learning by combining exploration and exploitation. This architecture has achieved progressive results in a range of discrete environments and continuous control tasks, both in terms of data efficiency and final performance.

Many deep reinforcement learning algorithms limit their ability to effectively exploration on the whole. CA2E uses a compound method of parallel computation, effective exploration and prior exploitation to address the issue: generating a diverse set of experiences and leveraging useful demonstrations for learning. Profit from the distributed system, our algorithm perfectly integrates the substantially exploration and effective exploitation.

CA2E is mainly designed for large quantities of data generation in parallel, where it includes simulated environments but also a series of real-world applications. In this work, we focus on applying the CA2E architecture to DQN and DDPG, but it can be also integrated with other off-policy deep reinforcement learning algorithms.

CA2E, as a scalable architecture for deep reinforcement learning, is gradually practical for research and applications. We hope our algorithm can speed up the efforts in distributed systems and compound algorithms for deep reinforcement learning.

Acknowledgment

Thanks to the Baseline algorithms provided by OpenAI.

References

[1] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[2] Silver D, Huang A, Maddison C J, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. *arXiv preprint arXiv*:1707.06347, 2017.

[4] Gu S, Lillicrap T, Sutskever I, et al. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, 2016.

[5] Dean J, Corrado G, Monga R, et al. Large scale distributed deep networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems, 2012.

[6] Kaiser L, Gomez A N, Shazeer N, et al. One model to learn them all. *arXiv preprint arXiv:*1706.05137, 2017.

[7] Nair A, Srinivasan P, Blackwell S, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:*1507.04296, 2015.

[8] Mnih V, Badia A P, Mirza M, et al. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 2016.

[9] Babaeizadeh M, Frosio I, Tyree S, et al. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:*1611.06256, 2016.

[10] Horgan D, Quan J, Budden D, et al. Distributed prioritized experience replay. *arXiv preprint arXiv:*1803.00933, 2018.

[11] Fortunato M, Azar M G, Piot B, et al. Noisy networks for exploration. *arXiv preprint arXiv:*1706.10295, 2017.

[12] Plappert M, Houthooft R, Dhariwal P, et al. Parameter space noise for exploration. *arXiv preprint arXiv:*1706.01905, 2017.

[13] Hester T, Vecerik M, Pietquin O, et al. Deep Q-learning from Demonstrations. *arXiv preprint arXiv:*1704.03732, 2017.

[14] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. Nature, 2015.

[15] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:*1509.02971, 2015.

[16] Sutton R S, Barto A G. Reinforcement learning: An introduction. *MIT press*, 1998.

[17] Peters J, Vijayakumar S, Schaal S. Natural Actor-Critic. *Neurocomputing*, 71(7):1180-1190, 2008.

[18] Schulman J, Levine S, Abbeel P, et al. Trust region policy optimization. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

[19] Schaul T, Quan J, Antonoglou I, et al. Prioritized experience replay. *arXiv preprint arXiv:*1511.05952, 2015.

[20] Jaderberg M, Dalibard V, Osindero S, et al. Population based training of neural networks. *arXiv preprint arXiv:*1711.09846, 2017.

[21] Hasselt H V. Double Q-learning. In Advances in Neural Information Processing Systems, 2010.

[22] Hessel M, Modayil J, Van Hasselt H, et al. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:*1710.02298, 2017.

[23] Bellemare M G, Naddaf Y, Veness J, et al. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253-279, 2013.

Appendix A

1. CA2E-DQN

For Discrete environments in gym, we use a similar network architecture as described in DQN, which consists of 3 convolutional layers (32 filters of size 8×8 with stride 4, 64 filters of size 4×4 with stride 2, 64 filters of size 3×3 with stride 1, in sequence), 1 hidden layer with 512 units and a fully connected linear output layer with one unit for each action. ReLUs are used in each layer, while layer normalization is used in the fully connected part of the network.

The policy network with the same architecture as the Q-value network, except for a softmax output layer. The Q-value network is trained by the Adam optimizer with a learning rate of 1e-4 and a batch size of 32. The target networks are updated every 10 K timesteps. The replay buffer holds 1 M transitions. For the ϵ-greedy, actors use different values ϵ for an abundant exploration in the environment, where we linearly anneal ϵ from the initial values to 0.1 over the first 1 M timesteps. As for observations, each frame is down-sampled to 84×84 pixels with converted to grayscale. The actual observation consists of a concatenation of 4 subsequent frames. Additionally, we use up to 30 noop actions at the beginning of the episode.

2. CA2E-DDPG

For continuous tasks, since the setting features of the agents in MuJoCo simulator are low-dimensional states, we use 3 dense layers to replace convolutional layers in DQN where the neural network is represented to process the pixel image inputs. A similar network architecture as described by DDPG is used: both the actor and the critic use 2 hidden (dense) layers with 64 ReLU units each, while layer normalization is applied to all layers. The actor network is followed by an output layer while the critic network is followed by a softmax output layer. Both the actor and critic are updated using the Adam optimizer with batch sizes of 128, where the critic is trained with a learning rate of 1e-3 while the actor uses a learning rate of 1e-4. The target networks are soft-updated with τ = 1e-3. In addition, the critic is regularized using an L2 penalty with 1e-2.

Appendix B

1. Discrete Environments

BreakoutNoFrameskip-v4 and PongNoFrameskip-v4 are to maximize your score in the Atari 2600 games, which were simulated through the Arcade Learning Environment (ALE). In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3).

2. Continuous Tasks

Benchmarking tasks have been performed in two continuous control domains ((a) Hopper-v2 and (b) Reacher-v2), which were implemented in the MuJoCo physics simulator. Hopper-v2 is a hopper walker with action, state dimensionalities $|\mathcal{S}| = 13$ and $|\mathcal{A}| = 3$ respectively. Reacher-v2 is a manipulator with $|\mathcal{S}| = 13$ and $|\mathcal{A}| = 2$, which receives reward for catching a randomly-initialized moving ball.