Improve the Training Efficiency for Multi-Agent RL using Master-Slave Framework

Minghao Guo
Department of Information Engineering
The Chinese University of Hong Kong
Sha Tin, Hong Kong
gm019@ie.cuhk.edu.hk

Haodong Duan
Department of Information Engineering
The Chinese University of Hong Kong
Sha Tin, Hong Kong
dh019@ie.cuhk.edu.hk

Abstract

Multi-Agent reinforcement learning (MAML) provides a framework that involves the participation of more than one single agent to solve the sequential decision-making problems. It improves the exploration ability of the reinforcement learning system, so can be exploited as a powerful technique in the situations consisting large search space. However, a major challenge in MAML is the training efficiency, since the difficulty in convergence and large gradient variance exposed in single-agent reinforcement learning are magnified in the multi-agent cases. When it comes to complex environments, typically with high dimension, continuous actions and infinite number of degree of freedom, the high training cost raises as a major bottleneck. In this paper, we propose a novel framework to improve the training efficiency for multi-agent reinforcement learning, which composes a *master* expert to produce rational but high-cost demonstrations, and several *slave* agents to exploit the information from the master while fast explore the policy space. By introducing the master-slave framework, the training for Markov games with high-dimensional state-action space and unknown dynamics becomes more effective and scalable. Experimental results on continuous control and neural architecture search validate the effectiveness of our master-slave framework.

1 Introduction

Reinforcement learning (RL) is a general and powerful framework for decision making under uncertainty. Recent advances in deep learning have enabled a number of RL applications including games, robotics, automated machine learning etc. Basically, reinforcement learning works really well in the simulated environment. However, when it comes to the real world, the training of the agent becomes much more difficult, always needs a large amount of training data and carefully designed training tricks. The underlying reason of this phenomenon is that the simulated environment is low dimensional, the action space is discrete and has a finite number of degree of freedom, while in the real world setting, the environment becomes much more complicated, with higher dimension, continuous actions and infinite number of degree of freedom. As the environment becomes more complicated, the training efficiency becomes lower and lower, making the convergence of the algorithm harder. This intrinsic difference on the complexity of environment causes the gap between simulation and the real world.

One possible solution to improve the training efficiency of reinforcement learning is multi-agent reinforcement learning, as shown in Figure 3a. Instead of using one single agent, MARL exploits several agents to simultaneously explore the whole environment. The agents communicate with each other through message channels. Although MARL increases the training efficiency to some extent, there are still several major drawbacks. First, the message channels introduce additional parameters,

Preprint. Under review.

making the training even harder. Second, though the exploring space of each agent is reduced, the dimension of freedom is still high. Furthermore, despite the paralleled training amortizes the training efficiency for each agent, the overall training cost still remains the same. To address these issues, the goal of our project is to work out a method that can efficiently explore the complicated environment and reduce the overall training cost.

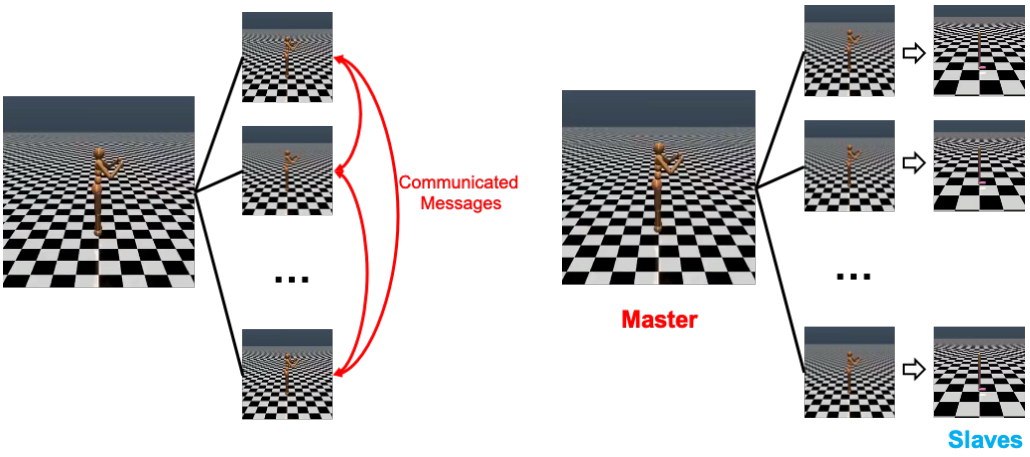
Our proposed method is called the *master-slave* framework. Instead of using multiple agents to explore the same complicated environment, we want these agents to explore in a relatively easier environment, with fewer dimension or degree of freedom (Dof). Note that this easy environment has the same or similar target of the training task with the original environment, which means that they share basic principles to achieve the target goal. For example, as shown in Figure 3b, in the context of human walking, the original complicated environment has a degree of joints say 20, where in the reduced environment, the degree of joints is 5. But the key components to make a human body to walk are shared, so are the basic principles. We call the agents operated on the reduced environment as slaves, and the agent of the original environment as master. For the master agent, since the environment has richer information, we can use relatively low update frequency for better exploiting the observations from the environment. And for the slave agents, since the Dof of environment is reduced, it is much easier to train the slave agent and we use a high update frequency for fast exploration. By combining the master and the slaves, we can get a better trade off between exploitation and exploration, and finally improve the training efficiency.

For the master-slave framework, we design several methods to aggregate the information from these two kinds of agents, as shown in Figure 2. The first one is called pre-training. We first train the slave agent on the reduced environment using typical reinforcement learning algorithm, and then the master agent directly inherits the parameters from slave agents and continues training on the full environment. The knowledge transfer is achieved by the network parameter initialization. The second method, is called multi-tasking. In this setting, the master agent and the slave agent share a common network backbone, and have separate stems and heads for taking corresponding observations as input and output corresponding actions. The two kinds of agents are trained simultaneously, while each of the agent operates on their own environment. The knowledge transfer is achieved by the shared network backbone. The third method is gradient momentum. The basic idea is that the two kinds of agents are trained separately on their corresponding environment, but the update/learning of the agent parameters is not only determined by their solely observation, but also affected by the gradients of the other agents. To validate the effectiveness of the master-slave framework, we perform experiments on the continuous control of human walking, and experiment on Neural Architecture Search. Experimental results show our framework significantly increases the training efficiency of MARL, and can be easily generalized across different problem settings.

2 Related Work

2.1 Model Free Reinforcement Learning

Most RL algorithms can be divided into two categories: model-based or model-free. Since a ground-truth model of the environment is usually not available to the agent, model-free algorithms are more popular and frequently used to solve problems. **Q-Learning** [19] is a model-free off-policy algorithm, it estimates the long-term return $Q(s, a)$ of executing an action a from a given state s. The estimated returns are Q-values and can be used for decision making. A lot of deep RL algorithms are based on Q-Learning. DQN [10] is the first deep learning model to successfully learn control policies directly from high-dimension input using RL, which use convolution neural network as the estimator of Q-values. A lot of work further improve DQN, including DRQN[6], Double DQN[17], Dueling DQN[18], etc.. Another family of model free RL algorithms is called **Policy Optimization**. The policy is represented explicitly as $\pi_\theta(a|s)$, while the performance objective J should be a differentiable function of θ. Sutton *et al.*first proposed policy gradient method with value function approximation in [14]. TRPO[11] and PPO[12] further improve vanilla policy gradient by improving the update policy. Another variant of policy gradient is called deterministic policy gradient[9, 13], which can only be used on continuous action space.



(a) (b)

Figure 1: (a) A typical framework of multi-agent reinforcement learning (MARL). MARL exploits several agents to simultaneously explore the whole environment. The agents communicate with each other through message channels. (b) Our proposed Master-slave framework. Instead of using multiple agents to explore the same complicated environment, we want these agents to explore in a relatively easier environment, with fewer dimension or degree of freedom. Note that this easy environment has the same or similar target of the training task with the original environment, which means that they share basic principles to achieve the target goal.

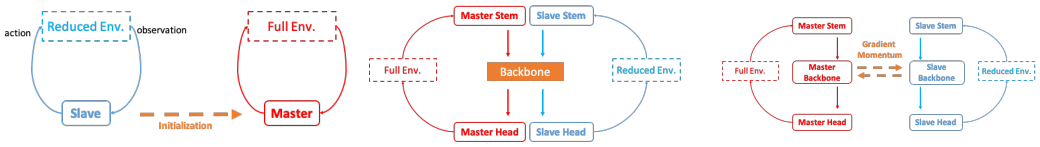
2.2 Multi-Agent Reinforcement Learning

Intriguingly, most of the successful applications for reinforcement learning involve the participation of more than one single agent/player , which should be modeled systematically as multi-agent reinforcement learning problems. Specifically, multi-agent reinforcement learning addresses the sequential decision-making problem of multiple autonomous agents that operate in a common environment, each of which aims to optimize its own long-term return by interacting with the environment and other agents [1]. By sharing experiments, agents with similar goals can learn faster and get better performance. There are several different kinds of methods. One representative way is to exchange information using communication [15]. The message channels and the agent are trained simultaneously to automatically learn the useful messages transferred between agents. A speed-up can be realized in MARL thanks to parallel computation, when the agents exploit the decentralized structure of the task. Several methods have explored along this direction, including [2, 3, 4, 7, 8]. The agent can take over tasks when one or more agents fail in a multi-agent system. Furthermore, most multi-agent systems also allow the easy insertion of new agents into the system, leading to a high degree of scalability.

3 Methods

3.1 Problem Setting

There are two agents in our reinforment learning system, named the master agent and the slave agent respectively. The two agents may be in different environments and may have different goals. However, they share the same basic principles. The difference between the master and the slave is: 1. The master agent usually has higher degrees of freedom, which makes it closer to the reality while also makes the exploration less efficient; 2. The master agent may have additional constraints comparing to the slave agent, which is also an obstacle during observation. What we care about is the performance of the master agent, but the inefficient exploration makes it very difficult to gain a good policy without the help of the slave agent. Our master-slave framework aims at exploiting the efficient exploration of the slave agent, transfering knowledge of the slave agent to the master agent to make training of the master agent more efficient.



(a) (b) (c)

Figure 2: Three different methods to aggregate information in the master-slave framework. (a) Pre-training. The slave agent is trained on the reduced environment using typical reinforcement learning algorithm, and then the master agent directly inherits the parameters from slave agents and continue training on the full environment. The knowledge transfer is achieved by the network parameter initialization. (b) Multi-tasking. The master agent and the slave agent share a common network backbone, and have separately stems and heads for taking corresponding observations as input and output corresponding actions. The two kinds of agent are trained simultaneously, while each of the agent operates on their own environment. The knowledge transfer is achieved by the share network backbone. (c) Gradient momentum. Two kinds of agents are trained separately on their corresponding environment, but the update/learning of the agent parameters is not only determined by their solely observation, but also affected by the gradients of the other agents.

3.2 Master-Slave Framework

We develop several methods to transfer the knowledge from the slave agent to the master agent. In experiments, we find that different methods work in different cases.

3.2.1 Pre-training

One straight-forward strategy to transfer knowledge from the slave agent to the master is to use the slave agent's weight as initialization for the master agent. It can provide the master agent a good starting point for further exploration. During experiments, we find that the pre-training strategy works well when the slave agent is a dof-reduced version of the master agent (which means its observation and action space are sub-spaces of the master's). When the low exploration efficency of the master agent is due to additional constraints, the pre-training strategy doesn't work.

3.2.2 Multi-tasking

Another way to transfer knowledge from the slave to the master is to make them share the same middle representation. In the Actor-Critic framework, we divide both actor and critic into three components: stem, backbone, head. The stem takes observations as input and outputs state representation. The backbone predicts middle representation based on state representation. Then the heads of Actor and Critic networks predict action or Q-value respectively based on the middle representation. In multi-task learning, we enforce two agents to share the backbone for both Actor and Critic networks. Two agents can also share the stem part, if their observation spaces are exactly the same. In experiments, we find that Multi-Tasking strategy works well when two agents share the same observation space. Otherwise, it will be extremely hard to optimize the multi-stem multi-head Actor-Critic, which leads to inferior performance eventually.

3.2.3 Gradient Momentum

In this method, the master and slaves start at an initial point in the policy space and find their optimal policy using gradient descent. A naïve idea is to sum up the gradient of two kinds of agents. However, since the environments that the agents operate on are different, and the corresponding optimal policies are also not the same. Directly summing up the gradients will introduce bias to the final result.

To address this, at each step of the gradient descent of the slave agents, we first use the gradient of the master agent as the initialization. Then the slave agent explore the policy space, find a reasonable gradient direction and use it to update the master's policy. We call this scheme as the master momentum of slaves. In this way, the gradient descent direction is dominated by the master, and the final optimized result will have no bias. The mathematical formulation is as follows. For the

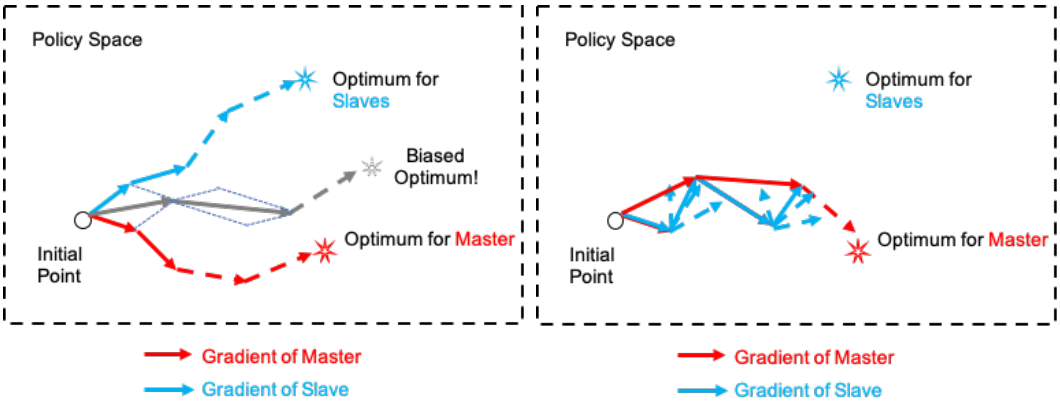


Figure 3: Comparison with different methods to aggregate gradient information for master-slave framework. (a) Naïve summing up the gradient of two kinds of agents will introduce bias to the final result. (b) The proposed gradient momentum method. The gradient descent direction is dominated by the master, and the final optimized result will have no bias.

master agent, the network parameter W is updated at the t-th iteration by,

$$\begin{aligned} V_t &= \beta V_{t-1} + (1-\beta) S_t, \\ W &= W - \alpha V_t, \end{aligned} \tag{1}$$

where S_t denotes the vanilla SGD gradient for the master agent, α denotes the learning rate, β denotes the momentum rate. For the slave agent, we replace the term of momentum in the optimizer with the gradient of the master agent,

$$\begin{aligned} v_t &= \beta v_{t-1} + (1-\beta) s_t, \\ w &= w - \alpha V_t, \end{aligned} \tag{2}$$

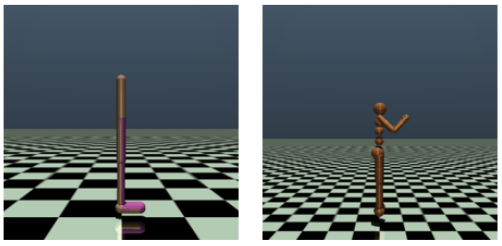
where s_t denotes the vanilla SGD gradient for the slave agent, and w denotes the network parameters of the slave agent.

4 Experiments on continuous Control

4.1 Environments

Our experiments on continuous control are based on two environments provided by mujoco [16]: `Walker2d-v3` and `Humanoid-v3`. The tasks of both environments are making the agent to move forward without falling down, while the structures of two agents are completely different. In Fig.4, we visualize agents in the two environments. For each environment, we define different master and slave agents. Among methods in our Master-Slave framework, we test Pre-training method and Multi-tasking methods on these two environments. We use the Actor-Critic framework as our baselines, in which the actor and the critic are all instantiated with four layer multi-layer perceptrons. We use PPO [12] as the optimizing algorithm.

Figure 4: Visualization of agents in Walker2d and Humanoid. Humanoid agents are with much larger degrees of freedom.



4.1.1 Walker2d

The Walker2d is a relatively easy environment to solve. The dimensions of the observation and action space are 17 and 6 respectively, which makes the exploration much more efficient. The height z of

Benchmark	Walker2d-Slave	Walker2d-Master	Pre-training	Multi-tasking
Success Rate	0.9	0.56	0.6	0.86
Avg. Episode Reward	3178.9	2151.3	1746.3	2787.7
Avg. Successful Episode Reward	3503.7	2972.2	2777.5	3233.4

Table 1: Results on Walker2d. A testing episode lasts at most 1000 steps. If the agent walks 1000 steps without falling down, we call that a successful episode. We provide average rewards of all episodes and successful episodes. The result is the average of 100 testing episodes with the same random seed set at the begining of testing.

the mass center is the indicator of falling down. For Walker2d-Slave, the acceptable range of z is very large: $z \in (0.8, 2.0)$. For Walker2d-Master, we add a strict constraint about the height of mass center: $z \in (1.1, 1.4)$. The strict constraint makes the agent easy to fail during exploration, thus makes the exploration less efficient.

4.1.2 Humanoid

On the opposite, for Humanoid, the degrees of freedom is far larger than Walker2d. The observation dimension of Humanoid is 376 and the action dimension is 17. Training makes progress extremely slow with such huge observation and action space. We reduce the dimension of observation and action spaces to form the environment of the slave agent, only keep those most essential dimensions. For observation, we keep dimensions about joints locations and velocities(46 dim). For action, we keep dimensions which are related to lower body movement(17 dim). The Humanoid-Master interacts with the full environment, and the Humanoid-Slave interacts with the reduced environment.

4.2 Experiments on Walker2d

We explore different methods to transfer knowledge from Walker2d-Slave to Walker2d-Master. The results are demonstrated in Table.1 and Fig.5. For the Pre-training method, no additional work is needed since two agents have exactly the same network structure. For the Multi-tasking method, two agents share the same stem, backbone, and have individual head for both actor and critic.

First of all, since an additional constraint is added to Walker2d-Master, it has much lower episode success rate and average episode reward comparing to Walker2d-Slave if the policy is learned directly. Using Multi-tasking strategy leads to faster convergence, and the performance is also much higher than the Walker2d-Master baseline, even competitive to the Walker2d-Slave (though the agent is evaluated in a harder environment). With Multi-tasking strategy, the Walker2d-Master learns a better middle representation to exploit. The Pre-training strategy, however, doesn't work well. From Fig.5, we see that with Walker2d-Slave weights as initialization, the Walker2d-Master has better performance at the beginning of the training. However, as training goes on, the knowledge is no longer useful due to the additional constraint and is quickly forgotten.

4.3 Experiments on Humanoid

We demonstrates results on Humanoid in Table.2 and Fig.6. The difference of Humanoid-Slave and Humanoid-Master is that Humanoid-Slave uses reduced observation and action spaces. It makes the training converge much faster, while also leads to inferior performance due to the reduction. Since the observation space and action space of two agents differ, for the Pre-training method, the master agent is partially initialized with the slave agent's weight. Weights related to the reduced observation and action dimensions are initialized as 0. For the Multi-tasking method, Humanoid-Master and Humanoid-Slave have their own stem and head, while share the same backbone.

From Fig.7, we see that due to the large degrees of freedom and low efficiency exploration, Humanoid-Master can hardly make any progress. The Humanoid-Slave, with reduced observation and action space, converges quickly and achieves reasonable performance, if not optimal. Pre-training is a good strategy to transfer knowledge from the slave to the master. From Fig.8, we see that Humanoid-Master initialized with the slave's weights can achieve good performance at the begining of training. After

Benchmark	Humanoid-Slave	Humanoid-Master	Pre-training	Multi-tasking
Avg. Episode Reward	523.7	223.9	608.0	106.0
Avg. Episode Length	190.0	108.9	203.2	86.9

Table 2: Results on Humanoid. We report average reward and length of testing episodes. The result is the average of 100 testing episodes with the same random seed set at the begining of testing.

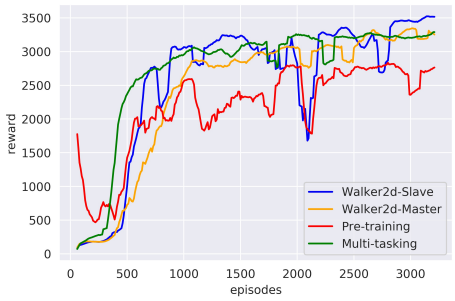


Figure 5: Validating Results on Walker2d.

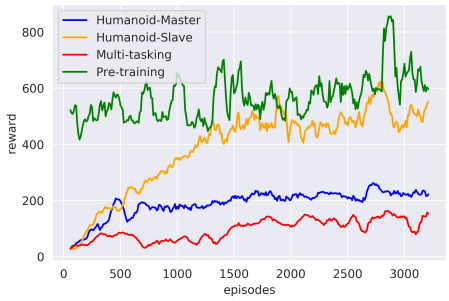


Figure 6: Validating Results on Humanoid.

that, it gradually refine the policy by taking more observation into consideratino, and by adding upper body actions. From the supplementary videos, we see that the Humanoid-Master agent learned by Pre-training strategy can achieve better performance comparing to the Humanoid-Slave. Regular upper body movement makes it keep balance better. The Multi-tasking method, however, gives out even worse performance comparing to the Humanoid-Master baseline. We suppose that it's because optimizing the multi-stem multi-head Actor-Critic is extremely hard.

5 Experiments on neural architecture search

For the inverse reinforcement experiment, we use the momentum gradient method to validate the master-slave framework. We follow the settings in [5]. We use ResNext as our expert policy. The full environment which is also the search space has a maximal number of 25 layers and 15 candidate operations, resulting a search space of size 15^{24}. The reduced environment has 8 layers and 11 candidate operations, resulting a search space with 11^8. We use 1 master 2 slaves and perform momentum gradient to the master-slave. The experimental results are illustrated here. We visualize

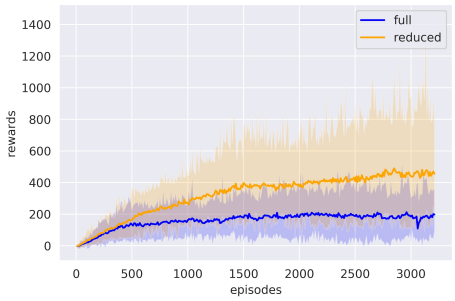


Figure 7: Comparing the average training rewards in the full environment and reduced environment. Shadow denotes the range of training rewards.

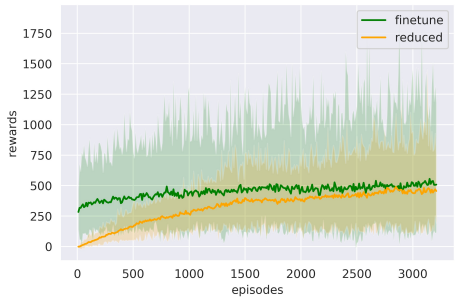


Figure 8: Comparing the average training rewards of the Pre-training method and the plain Humanoid-Slave. Shadow denotes the range of training rewards

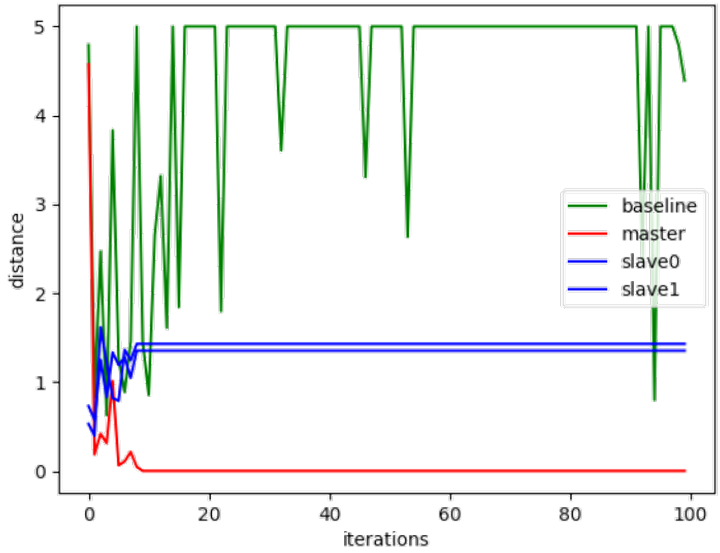


Figure 9: Experimental results of master-slave framework on neural architecture search. We can see that with the help of slave agents, the master agent can converge very easily compared to the baseline method, which we perform the inverse reinforcement learning directly on the full environment.

the loss between inverse policy and the expert policy, as shown in Figure 9. The lower the better. We can see that with the help of slave agents, the master agent can converge very easily compared to the baseline method, which we perform the inverse reinforcement learning directly on the full environment.

6 Conclusion

In this paper, we have proposed a novel framework to improve the training efficiency for multi-agent reinforcement learning, which composes a master expert to produce rational but high-cost demonstrations, and several slave agents to exploit the information from the master while fast explore the policy space. we have designed several methods to aggregate the information from these two kinds of agents, including pre-training, multi-tasking, and gradient momentum. Experiments on the continuous control of human walking and on neural architecture search show our framework can significantly increase the training efficiency of MARL, and can be easily generalized across different problem settings. By introducing the master-slave framework, the training for Markov games with high-dimensional state-action space and unknown dynamics becomes more effective and scalable.

References

[1] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[2] Lucian Busoniu, Bart De Schutter, and Robert Babuska. Multiagent reinforcement learning with adaptive state focus. In *BNAIC*, pages 35–42, 2005.

[3] Robert Fitch, Bernhard Hengst, Dorian Šuc, Greg Calbert, and Jason Scholz. Structural abstraction experiments in reinforcement learning. In *Australasian Joint Conference on Artificial Intelligence*, pages 164–175. Springer, 2005.

[4] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234. Citeseer, 2002.

[5] Minghao Guo, Zhao Zhong, Wei Wu, Dahua Lin, and Junjie Yan. Irlas: Inverse reinforcement learning for architecture search. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[6] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*, 2015.

[7] Jelle R Kok, Matthijs TJ Spaan, and Nikos Vlassis. Non-communicative multi-robot coordination in dynamic environments. *Robotics and Autonomous Systems*, 50(2-3):99–114, 2005.

[8] Jelle R Kok and Nikos Vlassis. Sparse cooperative q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 61, 2004.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[13] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014.

[14] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[15] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[16] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[17] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[18] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[19] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.