Automatic Successive Reinforcement Learning with Multiple Auxiliary Rewards

Zhao-Yang Fu , **De-Chuan Zhan** , **Xin-Chun Li** and **Yi-Xing Lu**
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
{fuzy,lixc}@lamda.nju.edu.cn, zhandc@nju.edu.cn, YixingLu97@gmail.com

Abstract

Reinforcement learning has played an important role in decision making related applications, e.g., robotics motion, self-driving, recommendation, etc. The reward function, as a crucial component, affects the efficiency and effectiveness of reinforcement learning to a large extent. In this paper, we focus on the investigation of reinforcement learning with more than one auxiliary reward. It is found that different auxiliary rewards can boost up the learning rate and effectiveness in different stages, and consequently we propose the Automatic Successive Reinforcement Learning (ASR) for auxiliary rewards grading selection for efficient reinforcement learning by stages. Experiments and simulations have shown the superiority of our proposed ASR on a range of environments, including OpenAI classical control domains and video games; Freeway and Catcher.

1 Introduction

Reinforcement learning [Sutton and Barto, 1998] has played an important role in many realistic domains, such as robotics, self-driving and recommendation systems. In reinforcement learning, agents interact with the environment by trial and error. Meanwhile, a reward signal, indicating how well they are performing, is given. In general cases, the goal of an agent is to maximize the expected cumulative reward.

The reward function is crucial to reinforcement learning [Ng *et al.*, 1999]. For policy-based reinforcement learning methods, the reward provided by environment determines the search directions of policies which will eventually affect the final policies obtained. For example, considering the reward signal in FetchReach environment [Plappert *et al.*, 2018], which becomes higher as the gripper gets closer to the target, then the policy obtained will lead the gripper to approach the target location. For more general cases, the choices of reward functions can be reflected in the efficiencies of general reinforcement learning approaches, e.g., the shaping reward is more efficient than the original reward in Pathfinding environment [Brys *et al.*, 2014a]. Though various rewards may lead to the final results, a reward function without elaborate designing may take more exploration.

A severe problem of reinforcement learning is how to train fast and efficiently using information provided by rewards. Although reinforcement learning has achieved great success and has been applied in many domains, existing methods have difficulty exploring effectively to learn a good policy when the reward is sparse and rare. They usually need to interact with the environment millions of times, especially in complicated real-world environments, and consequently leading to the whole procedure intractable. Delayed rewards and sparse rewards build a barrier to the widespread applicability of reinforcement learning.

It is a natural desire for selecting or designing an appropriate reward for better reinforcement learning with efficiency considered, especially those with sparse and rare reward functions. In real-world scenarios, rewards can be designed in many aspects. A typical example is the traffic light problem, in which car delay and system throughout are two correlated reward signals [Brys *et al.*, 2014b]. More generally, a reinforcement learning task in the real world is usually accompanied by a lot of rewards and domain knowledge. Therefore, the optimal policy can be calculated in many ways.

Auxiliary rewards have attracted much attention recently. Inspired by human learning, Bengio *et al.* proposed curriculum learning, i.e., when training reinforcement learning models, we can start with easier tasks and gradually increase the difficulty level of the tasks, and this brings benefits on learning effectiveness [Bengio *et al.*, 2009]. Reward shaping [Ng *et al.*, 1999] provides us another way to modify the original reward function for optimal policy preserving as well as boosting the performance.

Making use of auxiliary rewards can definitely improve the model. Previous work has shown that auxiliary rewards are useful to the learning rate since the multiple auxiliary rewards are expected to capture more information during learning. However, existing multiple auxiliary rewards methods are limited to simple linear combinations [Brys *et al.*, 2014a] or ensemble of multiple rewards [Brys *et al.*, 2017], which neglect the stage influence of rewards and reward selection we considered. In this paper, we focus on the investigation of reinforcement learning with multiple auxiliary rewards exploitation and selection in different stages.

It is notable that in reinforcement learning the guidance provided by rewards is various according to the changes of stages. For example, a simple ball moving task contains re-

wards about approaching a ball, grasping it, moving to a location; the reward based on the distance between the gripper and the ball is more important at first, and consequently as the gripper approaches the ball, another reward about grasping becomes more important, etc. Curriculum reinforcement learning manually designs the stages and invokes different reward functions and learning strategies for better results. However, in most cases, the hierarchical stages are hard to design.

In this paper, we propose the Automatic Successive Reinforcement Learning (ASR) with multiple auxiliary rewards. ASR performs reward selection automatically at each training step, which is the "atom" of the training stages. It is expected that ASR can achieve faster and better training. We empirically investigate the effectiveness of ASR, and it achieves superior performance on various environments.

In the following of this paper, we start with a brief review of related work, then give the ASR approach and the experimental results. Finally, we conclude the paper.

2 Related Work

The exploitation of different rewards has attracted much attention recently. In this paper, our method concentrates on taking advantage of multiple auxiliary rewards. There are two kinds of methods using multiple rewards: curriculum learning and multi-objective reinforcement learning (MORL).

Curriculum learning breaks a complicated problem down to a sequence of manageable stages and tasks manually. Since the learning process of curriculum learning is similar to human learning, it has been widely used in robotics. Most practical curriculum learning approaches use manual task sequences [Karpathy and van de Panne, 2012]. Some general frameworks aim to generate increasingly difficult tasks [Schmidhuber, 2013]. Unlike ASR, all existing studies in curriculum learning rely on curriculum design from easy to difficult, and the agent starts with easier tasks and learns all tasks in order of difficulty. Designing an effective curriculum by human efforts is a complex problem. However, ASR makes use of multiple auxiliary rewards simultaneously to help training, and doesn't require human efforts for curriculum designing. The multiple auxiliary rewards only need to encode some information for learning the task in ASR.

MORL, from another aspect, focuses on handling tasks with inherent multiple opposite objectives. By how many policies are computed in a single run, there are two types of methods. The first type to solve the MORL involves the use of scalarization function, known as the single policy methods, including the min-max method [Lin, 2005], weighted sum method[Kim and De Weck, 2006], and Chebyshev scalarization method [Moffaert *et al.*, 2013]. More and more complicated scalarization functions are proposed to approximate the Pareto frontier better. Other approaches seek to find multiple policies in a single run, e.g., radial method and Pareto following method for discrete Pareto frontier approximation [Parisi *et al.*, 2014], and the manifold method for continuous Pareto frontier approximation [Pirotta *et al.*, 2015]. Solutions of such problems often need to make a trade-off between multiple objectives since there are conflictions among objectives. Different from our scenario, MORL aims at the Pareto frontier, which actually is a preprocessing for missing constraints situations. The goal of ASR is to learn the optimal policy faster with multiple auxiliary rewards.

Therefore, many researchers have devoted to taking advantage of multiple rewards. However, there is no previous research performing reward selection for deep reinforcement learning. In this paper, a novel ASR framework is proposed, which utilizes all rewards to learn an agent and can determine the importance of each reward automatically. Specifically, by maximizing the improvement of objective function or size of parameter update at each training step, ASR could utilize the gradient information from multiple auxiliary rewards to find a better update direction. Consequently, faster and better performance can be achieved.

3 Preliminaries

In this section, we introduce the required concepts and methods for the derivation of ASR. We start from the notations of reinforcement learning.

3.1 Reinforcement Learning

A reinforcement learning environment usually consists of four components: state space S, action space A, state transition function T and scalar reward function R. At time t, an agent observes the state $s_t \in S$, then takes action $a_t \in A$ and receives a real-valued reward r_t. A probabilistic policy π is defined as a mapping from the state space to probability distributions over the action space.

In general, the state value function V^π and state-action value function Q^π for policy π are defined by discounted cumulative reward: $V^\pi(s) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t | s_0 = s\right]$ and $Q^\pi(s,a) = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t | s_0 = s, a_0 = a\right]$, where γ is the discount factor. The advantage function is usually defined as: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. The reinforcement learning algorithms aim to maximize the following expected total reward: $J_\pi = E_\pi\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]$.

3.2 Reward Shaping

Reward shaping is a useful method to incorporate auxiliary knowledge safely. The purpose of reward shaping is to explore how to modify the native reward function without misleading the agent. Let F be the shaping function, then $R + F$ is the new reward. Ng *et al.* point out that when F is an arbitrary potential-based shaping function, the optimality of policies will not be changed [Ng *et al.*, 1999]. According to this, many shaping functions can be constructed based on expert demonstrations or domain knowledge. Shaping function F is potential-based if there exists a real-valued function $\Phi : S \to \mathbb{R}$ such that $\forall s, s' \in S$, the Equation 1 holds.

$$F(s,a,s') = \gamma\Phi(s') - \Phi(s)\,. \tag{1}$$

Typically, the potential function indicates how good a state is, hence it's beneficial for learning a policy. Potential-based reward shaping has been successfully applied in many complex environments, such as StarCraft [Efthymiadis and Kudenko, 2013], RoboCup TakeAway [Devlin *et al.*, 2011] and Mario [Brys *et al.*, 2014a].

4 Proposed Method

Detailed approach and its optimization strategies are presented in this section. We restrict the discussion in the policy gradient approaches and propose the method concretely.

4.1 Automatic Successive Reinforcement Learning

Most of the recent deep reinforcement learning algorithms aim to optimize an objective function using a gradient descent method. Multiple auxiliary rewards can provide more gradient information, and the improvement of objective function or size of parameter update could tell us how much an agent learns at each iteration. Thus we can get a better gradient direction by combining the multiple gradient directions. Consequently, we propose the Automatic Successive Reinforcement Learning (ASR) framework which focuses on the exploitation of multiple auxiliary rewards. At each training step, we determine the weight of each reward automatically, and then a single reward algorithm is applied to learn an agent using the weighted sum of multiple rewards.

It's worthy to emphasize that the setting of ASR is different from MORL's. MORL attempts to approximate the Pareto frontier, and usually, the optimal policy is not unique. However, in this paper, auxiliary rewards are constructed by potential-based reward shaping, and thus the optimal policy holds for each auxiliary reward. ASR aims to learn the optimal policy faster and better with the help of auxiliary rewards.

The trust region policy optimization method (TRPO) is one of the state-of-the-art policy gradient methods [Schulman *et al.*, 2015]. TRPO is an effective method for optimizing large nonlinear policies such as neural networks with guaranteed monotonic improvement. In ASR, we choose TRPO as the base learner.

Standard TRPO optimization problem is defined by:

$$\begin{aligned} \max_{\theta} \quad & L_{\theta_{\text{old}}}(\theta) = \hat{E}_t \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} Q^{\pi_{\theta_{\text{old}}}}(s,a) \right] \\ s.t. \quad & \hat{E}_t \left[D_{KL}(\pi_{\theta_{\text{old}}}(\cdot|s) || \pi_\theta(\cdot|s)) \right] \leq \delta \,, \end{aligned} \tag{2}$$

where θ and θ_{old} are parameters of policy network, $\pi_{\theta_{\text{old}}}$ is the current policy, π_θ is the policy to be optimized, $\hat{E}_t[\cdot]$ represents the empirical average over several sampling trajectories like $(s_0, a_0, s_1, a_1, \ldots, s_T)$, and δ is the desired KL divergence. TRPO maximizes an objective function subject to a constraint on the KL divergence of new policy and old policy. Such constraint could avoid changing policy parameters too much. Consequently, the training process is more stable.

In practice, Equation 2 is hard to handle, and consequently, we usually deal with the approximation problem instead. Let H be the Hessian matrix of KL divergence. After making a linear approximation to the objective and a quadratic approximation to the constraint, the approximation problem is:

$$\begin{aligned} \max_{\theta} \quad & \left(\nabla L_{\theta_{\text{old}}}(\theta) \right)^T (\theta - \theta_{\text{old}}) \\ s.t. \quad & \tfrac{1}{2} (\theta - \theta_{\text{old}})^T H (\theta - \theta_{\text{old}}) \leq \delta \,. \end{aligned} \tag{3}$$

Now Equation 3 becomes a natural gradient problem [Amari, 1998]. The search direction z is given by:

$$z = H^{-1} g \,, \tag{4}$$

where $g = \nabla L_{\theta_{\text{old}}}$ is the gradient of the objective function. The search direction can be computed by approximately solving the equation $Hz = g$, which can be efficiently solved using the conjugate gradient algorithm [Schulman *et al.*, 2015]. Suppose the step size is β, the parameter θ is updated by:

$$\theta = \theta_{\text{old}} + \beta z \,. \tag{5}$$

From the KL divergence constraint $\frac{1}{2}\beta^2 z^T H z \leq \delta$, the maximum value of β is represented as:

$$\beta = \sqrt{2\delta / (z^T H z)} \,. \tag{6}$$

For the idea maximization of Equation 3 should be along with the gradient direction of the objective of Equation 2. In the multiple rewards scenarios, affections on gradient directions include the importance of each gradient and the differences between the old policy and the new one. Policy update is guided by multiple search directions. However, how to select the rewards using the information of all reward signals as possible is not an obvious problem. It is notable that the linear combination of potential-based shaping functions is also potential-based. We can formulate this problem as calculating a weighted sum of multiple rewards. At each training step, the optimal weight of each reward is determined by solving an optimization problem.

In scenarios where there are multiple reward functions, the multiple rewards can be denoted as a vector $\boldsymbol{R} = [R+F_1, R+F_2, \ldots, R+F_n]$, where R is the original reward and F_i is the potential-based shaping function. Corresponding to the i-th reward signal, L_i represents the TRPO's objective function, g_i is the gradient of L_i, z_i is the search direction and w_i is the weight of the i-th reward. Their vector forms are denoted by $L = [L_1, L_2, \ldots, L_n]^T, G = [g_1, g_2, \ldots, g_n], Z = [z_1, z_2, \ldots, z_n]$, and $w = [w_1, w_2, \ldots, w_n]^T$. According to Equation 4, we get each $z_i = H^{-1} g_i$, and it holds $HZ = G$. The objective function of the weighted sum of multiple rewards is given by $L^T w$, the gradient is Gw, and the search direction is Zw. For the fixed weights w, the optimization problem for the weighted sum of multiple rewards is:

$$\begin{aligned} \max_{\theta} \quad & \left(\nabla (L^T w) \right)^T (\theta - \theta_{\text{old}}) \\ s.t. \quad & \tfrac{1}{2} (\theta - \theta_{\text{old}})^T H (\theta - \theta_{\text{old}}) \leq \delta \,. \end{aligned} \tag{7}$$

Note that the solution of Equation 7 is related to weights w. ASR aims to determine the weight of each reward automatically, and the weights w is a variable to be optimized. When w has some specific properties, ASR can select or weight the auxiliary rewards in different stages of reinforcement learning. Consequently, ASR can automatically "generate" new combined reward for better reinforcement learning. In this paper, we advocate three strategies to determine the weight of each reward based on maximizing the improvement of objective function or size of parameter update, including reward selection, reward shrinking and maximum gradient.

In the ASR framework, each reward's gradient and search direction are computed at each iteration first, and then we find the optimal combinations of rewards by some strategies. A summary of our ASR framework is shown in Algorithm 1.

Implementation 1: Reward Selection

The target of reward selection strategy is to find the optimal weights w which can maximize the improvement of objective function with ℓ_1 constraint on w. Then the optimization problem becomes:

$$\begin{aligned} \max_{w} \max_{\theta} \ & \left(\nabla(L^T w)\right)^T (\theta - \theta_{\text{old}}) \\ s.t. \ & \tfrac{1}{2}(\theta - \theta_{\text{old}})^T H (\theta - \theta_{\text{old}}) \le \delta, \\ & \|w\|_1 = 1, \\ & w_i \ge 0, \ \ i = 1, 2, \ldots, n\,. \end{aligned} \tag{8}$$

Substitute the Equation 5 and Equation 6 into Equation 8. We get the ℓ_1 constraint ASR problem:

$$\begin{aligned} \max_{w} \ & w^T Z^T G w \\ s.t. \ & \|w\|_1 = 1, \\ & w_i \ge 0, \ \ i = 1, 2, \ldots, n\,. \end{aligned} \tag{9}$$

ℓ_1 constraint produces sparse solutions, therefore inherently performing reward selection. At each training step, the reward could obtain maximal improvement is more likely to be selected to perform parameter update.

Implementation 2: Reward Shrinking

We propose the reward shrinking strategy by applying the ℓ_2 constraint on weights. The ℓ_2 constraint has a different effect from ℓ_1 constraint; namely, it forces the weights to be spread out more equally. The ℓ_2 constraint is more likely to get a dense solution. Hence it can take advantage of more rewards at each iteration. The ℓ_2 constraint ASR problem is:

$$\begin{aligned} \max_{w} \ & w^T Z^T G w \\ s.t. \ & \|w\|_2 = 1, \\ & w_i \ge 0, \ \ i = 1, 2, \ldots, n\,. \end{aligned} \tag{10}$$

Implementation 3: Maximum Gradient

In general case, agents learn the target policy by gradient descent methods. The learning process of an agent is reflected in the parameter update. An uninformative reward will return a small gradient, leading to slow learning. Our maximum gradient strategy is the radical one intending to find the search direction with the biggest change in parameter space. That is, we maximize the parameter update $\|\theta - \theta_{\text{old}}\|_2$. The optimization problem of maximum gradient strategy is:

$$\begin{aligned} \max_{w} \ & \frac{w^T Z^T Z w}{w^T Z^T H Z w} \\ s.t. \ & \|w\|_2 = 1, \\ & w_i \ge 0, \ \ i = 1, 2, \ldots, n\,. \end{aligned} \tag{11}$$

Here we could remove the ℓ_2 constraint because the objective has nothing to do with the weight's magnitude. Equation 11 can be rewritten as Equation 12.

$$\begin{aligned} \max_{w} \ & w^T Z^T Z w \\ s.t. \ & w^T Z^T H Z w = 1, \\ & w_i \ge 0, \ \ i = 1, 2, \ldots, n\,. \end{aligned} \tag{12}$$

Algorithm 1 The pseudo code of ASR framework

```
Initialize policy π
for Iteration i = 0, 1, ... until convergence do
    Run policy π_{θ_old} in environment for T time steps
    for Reward k = 1, 2, ..., n do
        Compute all advantage values A_k^{π_{θ_old}}
        Compute gradient g_k of L_k and search direction z_k
    end for
    Find optimal weight w
    θ ← θ + βZw
    θ_old ← θ
end for
```

4.2 Optimization

In this section, we mainly focus on the optimization of our proposed ASR methods. Note that $Z^T G, Z^T Z$ and $Z^T H Z$ are all $n \times n$ matrices, and n usually is very small compared to parameter size. Thus, solving such problem doesn't take much time.

Firstly, Equation 9 is a quadratic programming problem with linear constraints which is like the optimization problem of SVM. The SMO algorithm breaks the optimization problem of SVM into a series of sub-problems that could be solved analytically [Platt, 1998]. Inspired by the SMO algorithm, we design a similar algorithm to solve Equation 9 efficiently. At every iteration, a pair (w_i, w_j) is selected first, and then we compute the maximum of a one-dimensional quadratic function analytically.

When ignoring the positive weight constraints, Equation 10 is a Rayleigh quotient problem and Equation 12 is a general Rayleigh quotient problem which could transfer to Rayleigh quotient problem, so we only need to take into account Equation 10. Furthermore, the Rayleigh quotient problem can be solved analytically by matrix eigenvalue decomposition. The optimal solution is given by the eigenvector v corresponding to the largest eigenvalue of $Z^T G$.

Suppose the feasible region of Equation 10 is denoted by $\mathcal{D}$, and $P_{\mathcal{D}}$ is a projection operator that projects a vector to region $\mathcal{D}$. Even though the eigenvector v may be no longer the optimal solution of Equation 10, $P_{\mathcal{D}}(v)$ is an excellent initial point. After initializing, projected gradient descent is performed to find a better point.

5 Experiments

In this section, empirical experiments and investigations are conducted to validate the effectiveness of ASR framework. We compare our approach to state-of-the-art single reward methods and multiple rewards methods. Since there are less ready-made multiple rewards environments, we extend some popular environments to multiple rewards environments, i.e., OpenAI classical control domains MountainCar, CartPole and Acrobot [Brockman *et al.*, 2016], PLE game Catcher[1] and Atari game Freeway. Firstly, we show the comparison results on three classical control environments and two more

[1] https://github.com/ntasfi/PyGame-Learning-Environment

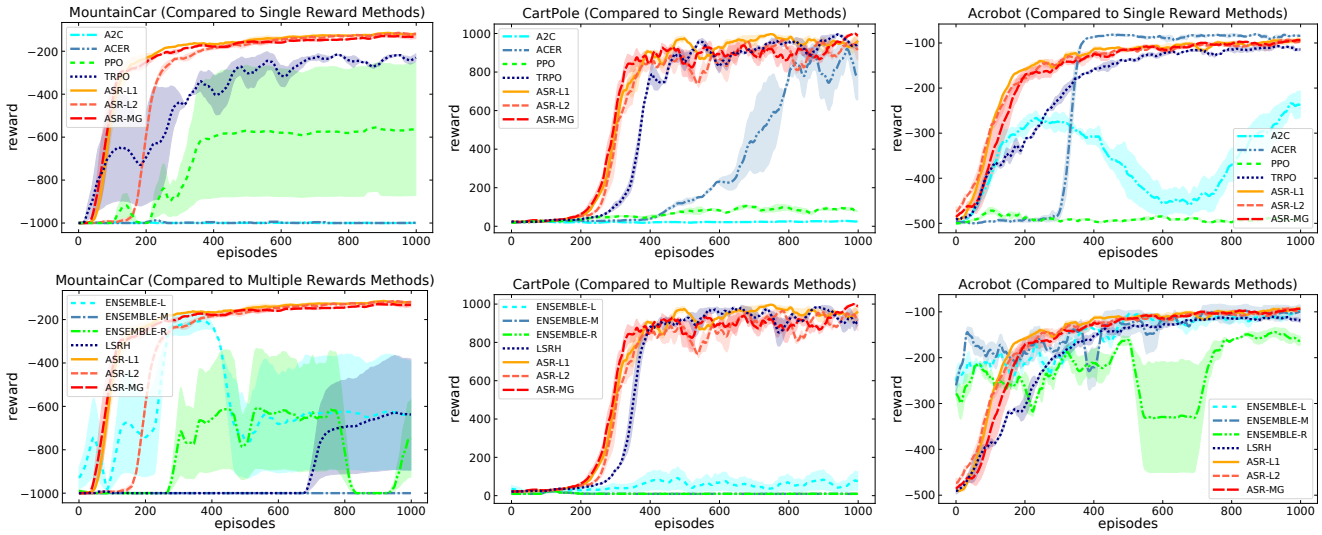

Figure 1: Comparisons on classical control domains

complicated video games. Then we demonstrate the visualization on weights during the training process.

5.1 Compared Methods and Configurations

We compare ASR to two kinds of methods. First is the single reward policy gradient methods, i.e., A2C [Mnih *et al.*, 2016], ACER [Wang *et al.*, 2017], TRPO [Schulman *et al.*, 2015], and PPO [Schulman *et al.*, 2017]. The second kind of methods is multiple rewards methods, i.e., linear scalarization reward shaping [Brys *et al.*, 2014a] and ensemble methods with different voting strategies [Brys *et al.*, 2017]. In detail, the compared methods are listed as follows:

Single Reward Methods

- **A2C** is a synchronous advantage actor-critic approach, which gives an equal performance with A3C;
- **ACER** is an off-policy actor-critic model with experience replay, greatly increasing the sampling efficiency;
- **TRPO** applies a KL divergence constraint to avoid large parameter update, improving training stability;
- **PPO** simplifies TRPO by using a clipped surrogate objective while retaining similar performance.

Multiple Rewards Methods

- **LSRH** applies a fixed linear scalarization of multiple shaping rewards;
- **ENSEMBLE-L** is an ensemble method with linear voting;
- **ENSEMBLE-M** is an ensemble method with majority voting;
- **ENSEMBLE-R** is an ensemble method with rank voting.

Single reward methods are implemented by OpenAI Baselines,[2] which are high-quality implementations of reinforcement learning algorithms. Since our proposed ASR methods are based on TRPO, for fairness, the base learner of multiple rewards methods is TRPO too.

For classical control environments, we perform one million time steps of training for each method. For video games, we perform ten million time steps of training for each method. Each method is run with five random seeds. We demonstrate the average training curves of two runs with the highest average cumulative reward over the entire training period, and the shade in figures shows the error in the estimate of the mean.

5.2 Comparisons on Classical Control Domains

Classical control is a collection of classical reinforcement learning environments implemented by OpenAI. We extend three classical control environments to multiple rewards environments, i.e., CartPole, MountainCar, and Acrobot. Details of auxiliary reward designs for the selected environments are:

- **MountainCar** The goal of MountainCar environment is to drive the car up the mountain. The original reward is -1 for every step taken. Therefore, the reward function is very uninformative especially when the car doesn't arrive at the goal location. We suggest using height, speed and position as potential functions to construct auxiliary rewards;
- **CartPole** A pole is attached by an unactuated joint to a cart, which moves along a frictionless track. The original reward is 1 for every step taken. We suggest using cart position and pole angle to construct the potential functions, since keeping cart position and pole angle small is conducive to preventing the pole from falling over;
- **Acrobot** Acrobot is a 2-link pendulum with only the second joint actuates. The goal is to swing the end-effector at a height at least the length of one link above the base. Similar to MountainCar environment, the original reward function is very uninformative. Heuristically, we use the height of end-effector as a potential function, since encouraging the agent to move up will help to reach the target height.

[2] https://github.com/openai/baselines

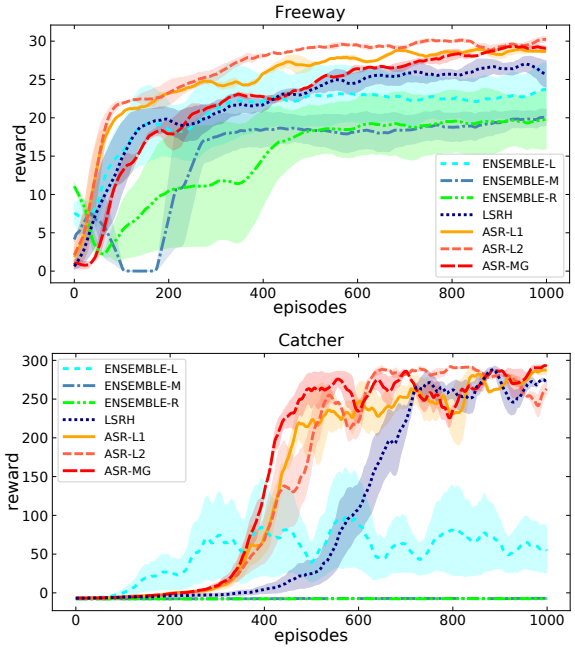

Figure 2: Comparisons on video games

The learning curves of all methods are shown in Figure 1. We plot the first one thousand episodes of each method, and the vertical axis is the cumulative reward of each episode. From the results, we can see some learning curves are flat because of the low learning rate. And the multiple rewards methods, especially ASR and LSRH, are faster than single reward methods, which verifies that auxiliary rewards could improve the training speed. Moreover, compared to multiple rewards methods, especially LSRH with fixed linear scalarization of rewards, ASR methods are faster and more stable. The learning curves of ASR show that ASR can provide monotonic and fast improvement during the learning process.

5.3 Comparisons on Video Games

In addition to classical control environments, much more complex pixel input environments are considered. Atari and PLE games are widely used video games. We extend two environments of Atari and PLE to multiple rewards environments, including Freeway and Catcher.

- **Freeway** In Freeway, the agent controls chickens run across a ten lane highway filled with traffic. The goal is to control chickens to move up and keep away from cars. The chicken is forced back if hit by a car. If the chicken gets across, the reward is 1, otherwise 0. The potential function of this environment is the chicken's height;
- **Catcher** In Catcher, the agent controls a paddle to move left or right to catch falling fruit. The agent receives a positive reward for each successful fruit catch. If the fruit is not caught, it receives a negative reward. Since the original reward is a little sparse, we introduce a denser reward in Catcher, i.e., the horizontal distance between paddle and fruit is used as a potential function.

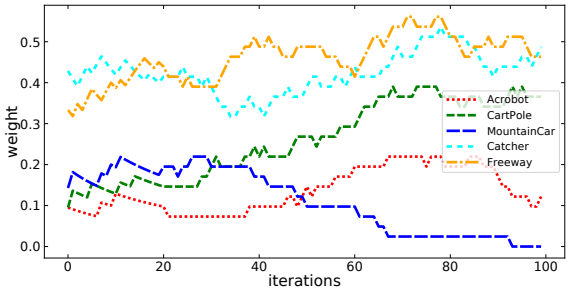

Figure 3: Weights of original rewards in ASR

We plot the first one thousand episodes of each method in Figure 2. From the results, our ASR approaches can achieve the best performance in Freeway and Catcher, which attributes to the automatic reward selection. Compared to LSRH, which applies a fixed linear scalarization of shaping rewards, the superiority of ASR validates the effectiveness of determining the weights automatically.

5.4 Visualization on Weights

The weight of a reward could reflect the importance of this reward. As mentioned before, the original reward functions are a little uninformative, especially at the preliminary stage. Therefore, it's reasonable to give the original reward a lower weight first. We demonstrate the smoothed curves of weights during the training process in Figure 3. From the results, the ASR can assign weights to original rewards lower than averages, which shows that our ASR methods can find a meaningful and helpful combination at each training step. Consequently, ASR can achieve faster and better performance.

6 Conclusion

Researchers have paid great attention to efficient reinforcement learning, while delayed rewards or uninformative rewards usually lead to low efficiency. Considering that multiple rewards are usually available in many real-world environments and can provide more information to the agent, we make use of multiple auxiliary rewards for speeding up the training process and develop Automatic Successive Reinforcement Learning (ASR) framework with the help of multiple auxiliary rewards. Investigations show ASR can increase the learning rate compared to existing methods. Experimental results also show that ASR can assign a low weight to the sparse reward and can take advantage of informative rewards at the preliminary stage. This observation validates that our reward selection is meaningful.

Acknowledgements

This work is supported by National Key R&D Program of China (2018YFB1004300), NSFC(61773198), and Collaborative Innovation Center of Novel Software Technology and Industrialization of NJU, Jiangsu. De-Chuan Zhan is the corresponding author. We thank Prof. Yang Yu for his valuable comments and suggestions on this work. Yi-Xing Lu is an undergraduate student of Dept. EE, NJU.

References

[Amari, 1998] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

[Bengio *et al.*, 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48, Montréal, Canada, 2009.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Brys *et al.*, 2014a] Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew E. Taylor, Daniel Kudenko, and Ann Nowé. Multi-objectivization of reinforcement learning problems by reward shaping. In *Proceedings of the 2014 International Joint Conference on Neural Networks*, pages 2315–2322, Beijing, China, 2014.

[Brys *et al.*, 2014b] Tim Brys, Ann Nowé, Daniel Kudenko, and Matthew E. Taylor. Combining multiple correlated reward and shaping signals by measuring confidence. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1687–1693, Québec, Canada, 2014.

[Brys *et al.*, 2017] Tim Brys, Anna Harutyunyan, Peter Vrancx, Ann Nowé, and Matthew E. Taylor. Multi-objectivization and ensembles of shapings in reinforcement learning. *Neurocomputing*, 263:48–59, 2017.

[Devlin *et al.*, 2011] Sam Devlin, Daniel Kudenko, and Marek Grzes. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(2):251–278, 2011.

[Efthymiadis and Kudenko, 2013] Kyriakos Efthymiadis and Daniel Kudenko. Using plan-based reward shaping to learn strategies in StarCraft: Broodwar. In *Proceedings of the 2013 IEEE Conference on Computational Inteligence in Games*, pages 1–8, Niagara Falls, Canada, 2013.

[Karpathy and van de Panne, 2012] Andrej Karpathy and Michiel van de Panne. Curriculum learning for motor skills. In *Advances in Artificial Intelligence*, pages 325–330, Heidelberg, Germany, 2012.

[Kim and De Weck, 2006] I. Y. Kim and O. L. De Weck. Adaptive weighted sum method for multiobjective optimization: A new method for Pareto front generation. *Structural and Multidisciplinary Optimization*, 31(2):105–116, 2006.

[Lin, 2005] JiGuan G. Lin. On min-norm and min-max methods of multi-objective optimization. *Mathematical Programming*, 103(1):1–33, 2005.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1928–1937, New York, NY., 2016.

[Moffaert *et al.*, 2013] Kristof Van Moffaert, Madalina M. Drugan, and Ann Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *Proceedings of the 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 191–199, Singapore, 2013.

[Ng *et al.*, 1999] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, pages 278–287, Bled, Slovenia, 1999.

[Parisi *et al.*, 2014] Simone Parisi, Matteo Pirotta, Nicola Smacchia, Luca Bascetta, and Marcello Restelli. Policy gradient approaches for multi-objective sequential decision making. In *Proceedings of the 2014 International Joint Conference on Neural Networks*, pages 2323–2330, Beijing, China, 2014.

[Pirotta *et al.*, 2015] Matteo Pirotta, Simone Parisi, and Marcello Restelli. Multi-objective reinforcement learning with continuous Pareto frontier approximation. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2928–2934, Austin, TX., 2015.

[Plappert *et al.*, 2018] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*, 2018.

[Platt, 1998] John C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods - Support Vector Learning*, 208:185–208, 1998.

[Schmidhuber, 2013] Jürgen Schmidhuber. PowerPlay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in Psychology*, 4:313, 2013.

[Schulman *et al.*, 2015] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1889–1897, Lille, France, 2015.

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

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA., 1998.

[Wang *et al.*, 2017] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *Proceedings of 5th International Conference on Learning Representations*, Toulon, France, 2017.