Exponentially Weighted Imitation Learning for Batched Historical Data

Qing Wang[1]* **Jiechao Xiong**[1] **Lei Han**[1] **Peng Sun**[1] **Han Liu**[12] **Tong Zhang**[1]

Abstract

We consider deep policy learning with only batched historical trajectories. The main challenge of this problem is that the learner no longer has a simulator or "environment oracle" as in most reinforcement learning settings. To solve this problem, we propose a monotonic advantage reweighted imitation learning strategy that is applicable to problems with complex nonlinear function approximation and works well with hybrid (discrete and continuous) action space. The method does not rely on the knowledge of the behavior policy, thus can be used to learn from data generated by an unknown policy. Under mild conditions, our algorithm, though surprisingly simple, has a policy improvement bound and outperforms most competing methods empirically. Thorough numerical results are also provided to demonstrate the efficacy of the proposed methodology.

1 Introduction

In this article, we consider the problem of learning a deep policy with batched historical trajectories. This problem is important and challenging. As in many real-world tasks, we usually have numerous historical data generated by different policies, but is lack of a perfect simulator of the environment. In this case, we want to learn a good policy from these data, to make decisions in a complex environment with possibly continuous state space and hybrid action space of discrete and continuous parts.

Several existing fields of research concern the problem of policy learning from batched data. In particular, imitation learning (IL) aims to find a policy whose performance is close to that of the data-generating policy [Abbeel and Ng, 2004]. On the other hand, off-policy reinforcement learning (RL) concerns the problem of learning a good (or possibly better) policy with data collected from a *behavior* policy [Sutton and Barto, 1998]. However, to the best of our knowledge, previous methods do not have satisfiable performance or are not directly applicable in a complex environment as ours with continuous state and hybrid action space.

In this work, we propose a novel yet simple method, to imitate a better policy by monotonic advantage reweighting. From theoretical analysis and empirical results, we find the proposed method has several advantages that

- From theoretical analysis, we show that the algorithm as proposed has policy improvement lower bound under mild condition.
- Empirically, the proposed method works well with function approximation and hybrid action space, which is crucial for the success of deep RL in practical problems.
- For off-policy learning, the method does not rely on the knowledge of action probability of the behavior policy, thus can be used to learn from data generated by an unknown policy, and is robust when current policy is deviated from the behavior policy.

In our real-world problem of a complex MOBA game, the proposed method has been successfully applied on human replay data, which validates the effectiveness of the method.

*[1]Tencent AI Lab. [2]Northwestern University.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

The article is organized as follows: We firstly state some preliminary notations (Sec. 2) and related works (Sec. 3). Then we present our main method of imitating a better policy (Sec. 4). The strengths of the proposed method are discussed with theoretical analysis (Sec. 5) and empirical experiments (Sec. 6). Finally we conclude our analysis (Sec. 7).

2 Preliminaries

Consider a Markov decision process (MDP) with infinite-horizon, denoted by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, d_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, P is the transition probability defined on $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, r is the reward function $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$, d_0 is the distribution of initial state s_0, and $\gamma \in (0, 1)$ is the discount factor. A trajectory τ is a sequence of triplets of state, action and reward, i.e., $\tau = \{(s_t, a_t, r_t)\}_{t=1,\dots,T}$, where T is the terminal step number. A stochastic policy denoted by π is defined as $\mathcal{S} \times \mathcal{A} \to [0, 1]$. We use the following standard notation of state-value $V^\pi(s_t)$, action-value $Q^\pi(s_t, a_t)$ and advantage $A^\pi(s_t, a_t)$, defined as $V^\pi(s_t) = \mathbb{E}_{\pi|s_t} \sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l})$, $Q^\pi(s_t, a_t) = \mathbb{E}_{\pi|s_t,a_t} \sum_{l=0}^{\infty} \gamma^l r(s_{t+l}, a_{t+l})$, and $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, where $\mathbb{E}_{\pi|s_t}$ means $a_l \sim \pi(a|s_l)$, $s_{l+1} \sim P(s_{l+1}|s_l, a_l)$, $\forall l \geq t$, and $\mathbb{E}_{\pi|s_t,a_t}$ means $s_{l+1} \sim P(s_{l+1}|s_l, a_l)$, $a_{l+1} \sim \pi(a|s_{l+1})$, $\forall l \geq t$. As the state space $\mathcal{S}$ may be prohibitively large, we approximate the policy and state-value with parameterized forms as $\pi_\theta(s, a)$ and $V^\pi_\theta(s)$ with parameter $\theta \in \Theta$. We denote the original policy space as $\Pi = \{\pi | \pi(s, a) \in [0, 1], \sum_{a \in \mathcal{A}} \pi(s, a) = 1, \forall s \in \mathcal{S}, a \in \mathcal{A}\}$ and parametrized policy space as $\Pi_\Theta = \{\pi_\theta | \theta \in \Theta\}$.

To measure the similarity between two policies π and π', we consider the Kullback–Leibler divergence defined as

$$D^d_{\text{KL}}(\pi' || \pi) = \sum_s d(s) \sum_a \pi'(a|s) \log \frac{\pi'(a|s)}{\pi(a|s)}$$

and total variance as

$$D^d_{\text{TV}}(\pi', \pi) = (1/2) \sum_s d(s) \sum_a |\pi'(a|s) - \pi(a|s)|$$

where $d(s)$ is a probability distribution of states.

The performance of a policy π is measured by its expected discounted reward:

$$\eta(\pi) = \mathbb{E}_{d_0,\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

where $\mathbb{E}_{d_0,\pi}$ means $s_0 \sim d_0$, $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. We omit the subscript d_0 when there is no ambiguity. In [Kakade and Langford, 2002], a useful equation has been proved that

$$\eta(\pi') - \eta(\pi) = \frac{1}{1-\gamma} \sum_s d_{\pi'}(s) \sum_a \pi'(a|s) A^\pi(s, a)$$

where d_π is the discounted visiting frequencies defined as $d_\pi(s) = (1-\gamma)\mathbb{E}_{d_0,\pi} \sum_{t=0}^{\infty} \gamma^t \mathbf{1}(s_t = s)$ and $\mathbf{1}(\cdot)$ is an indicator function. In addition, define $L^{d,\pi}(\pi')$ as

$$L^{d,\pi}(\pi') = \frac{1}{1-\gamma} \sum_s d(s) \sum_a \pi'(a|s) A^\pi(s, a)$$

then from [Schulman et al., 2015, Theorem 1], the difference of $\eta(\pi')$ and $\eta(\pi)$ can be approximated by $L^{d_\pi,\pi}(\pi')$, where the approximation error is bounded by total variance $D^{d_\pi}_{\text{TV}}(\pi', \pi)$, which can be further bounded by $D^{d_\pi}_{\text{KL}}(\pi' || \pi)$ or $D^{d_\pi}_{\text{KL}}(\pi || \pi')$.

In the following sections, we mainly focus on maximizing $L^{d_\pi,\pi}(\pi_\theta)$ as a proxy for optimizing policy performance $\eta(\pi_\theta)$, for $\pi_\theta \in \Pi_\Theta$.

3 Related Work

Off-policy learning [Sutton and Barto, 1998] is a broad region of research. For policy improvement method with performance guarantee, *conservative policy iteration* [Kakade and Langford, 2002] or

safe policy iteration [Pirotta et al., 2013] has long been an interesting topic in the literature. The term "safety" or "conservative" usually means the algorithm described is guaranteed to produce a series of monotonic improved policies. Exact or high-probability bounds of policy improvement are often provided in these previous works [Thomas and Brunskill, 2016, Jiang and Li, 2016, Thomas et al., 2015, Ghavamzadeh et al., 2016]. We refer readers to [Garcıa and Fernández, 2015] for a comprehensive survey of safe RL. However, to the best of our knowledge, these prior methods cannot be directly applied in our problem of learning in a complex game environment with large scale replay data, as they either need full-knowledge of the MDP or consider tabular case mainly for finite states and discrete actions, with prohibitive computational complexity.

Constrained policy optimization problems in the parameter space are considered in previous works [Schulman et al., 2015, Peters et al., 2010]. In [Peters et al., 2010], they constrain the policy on the distribution of $p^\pi(s,a) = \mu^\pi(s)\pi(a|s)$, while in [Schulman et al., 2015], the constraint is on $\pi(a|s)$, with fixed state-wise weight $d(s)$. Also, in [Schulman et al., 2015], the authors have considered $D_{\text{KL}}^{d_\pi}(\pi||\pi_\theta)$ as a policy divergence constraint, while in [Peters et al., 2010] the authors considered $D_{\text{KL}}(\mu^\pi\pi||q)$. The connection with our proposed method is elaborated in Appendix B.1. A closely related work is [Abdolmaleki et al., 2018] which present the exponential advantage weighting in an EM perspective. Independently, we further generalize to monotonic advantage re-weighting and also derive a lower bound for imitation learning.

Besides off-policy policy iteration algorithm, value iteration algorithm can also be used in off-policy settings. For deep reinforcement learning, DQN [Mnih et al., 2013], DQfD [Hester et al., 2018] works primarily with discrete actions, while DDPG [Lillicrap et al., 2016] works well with continuous actions. For hybrid action space, there are also works combining the idea of DQN and DDPG [Hausknecht and Stone, 2016]. In our preliminary experiments, we found value iteration method failed to converge for the tasks in the HFO environment. It seems that the discrepancy between behavior policy and the target policy ($\arg\max$ policy in DQN) should be properly restrained, which we think worth further research and investigation.

Also, there are existing related methods in the field of imitation learning. For example, when expert data is available, we can learn a policy directly by predicting the expert action [Bain and Sommut, 1999, Ross et al., 2011]. Another related idea is to imitate an MCTS policy [Guo et al., 2014, Silver et al., 2016]. In the work of [Silver et al., 2016], the authors propose to use Monte-Carlo Tree Search (MCTS) to form a new policy $\tilde{\pi} = \text{MCTS}(\pi)$ where π is the base policy of network, then imitate the better policy $\tilde{\pi}$ by minimizing $D_{\text{KL}}(\tilde{\pi}||\pi_\theta)$. Also in [Guo et al., 2014], the authors use UCT as a policy improvement operator and generate data from $\tilde{\pi} = \text{UCT}(\pi)$, then perform regression or classification with the dataset, which can be seen as approximating the policy under normal distribution or multinomial distribution parametrization.

4 Monotonic Advantage Re-Weighted Imitation Learning (MARWIL)

To learn a policy from data, the most straight forward way is imitation learning (behavior cloning). Suppose we have state-action pairs (s_t, a_t) in the data generated by a behavior policy π, then we can minimize the KL divergence between π and π_θ. To be specific, we would like to minimize

$$D_{\text{KL}}^{d}(\pi||\pi_\theta) = -\mathbb{E}_{s\sim d(s), a\sim\pi(a|s)}(\log \pi_\theta(a|s) - \log \pi(a|s)) \tag{1}$$

under some state distribution $d(s)$. However, this method makes no distinction between "good" and "bad" actions. The learned π_θ simply imitates all the actions generated by π. Actually, if we also have reward r_t in the data, we can know the consequence of taking action a_t, by looking at future state s_{t+1} and reward r_t. Suppose we have estimation of the advantage of action a_t as $\widehat{A}^\pi(s_t, a_t)$, we can put higher sample weight on the actions with higher advantage, thus imitating good actions more often. Inspired by this idea, we propose a monotonic advantage reweighted imitation learning method (Algorithm 1) which maximizes

$$\mathbb{E}_{s\sim d_\pi(s), a\sim\pi(a|s)} \exp(\beta \widehat{A}^\pi(s,a)) \log \pi_\theta(a|s) \tag{2}$$

where β is a hyper-parameter. When $\beta = 0$ the algorithm degenerates to ordinary imitation learning. Ideally we would like to estimate the advantage function $A(s_t, a_t) = \mathbb{E}_{\pi|s_t,a_t}(R_t - V^\pi(s_t))$ using cumulated discounted future reward $R_t = \sum_{l=t}^{T} \gamma^{l-t} r_l$. For example, one possible solution is to use a neural network to estimate $A(s_t, a_t)$, by minimizing $\mathbb{E}_{\pi|s_t,a_t}(A_\theta(s_t, a_t) - (R_t - V_\theta(s_t)))^2$

Algorithm 1 Monotonic Advantage Re-Weighted Imitation Learning (MARWIL)

Input: Historical data $\mathcal{D}$ generated by π, hyper-parameter β.
For each trajectory τ in $\mathcal{D}$, estimate advantages $\widehat{A}^{\pi}(s_t, a_t)$ for time $t = 1, \cdots, T$.
Maximize $\sum_{\tau \in \mathcal{D}} \sum_{(s_t, a_t) \in \tau} \exp(\beta \widehat{A}^{\pi}(s_t, a_t)) \log \pi_\theta(a_t|s_t)$ with respect to θ.

for R_t computed from different trajectories, where $V_\theta(s_t)$ is also estimated with a neural network respectively. In practice we find that good results can be achieved by simply using a *single path* estimation as $\widehat{A}(s_t, a_t) = (R_t - V_\theta(s_t))/c$, where we normalize the advantage by its average norm c^2 in order to make the scale of β stable across different environments. We use this method in our experiments as it greatly simplifies the computation.

Although the algorithm has a very simple formulation, it has many strengths as

1. Under mild conditions, we show that the proposed algorithm has policy improvement bound by theoretical analysis. Specifically, the policy $\tilde{\pi}$ is uniformly as good as, or better than the behavior policy π.
2. The method works well with function approximation as a complex neural network, as suggested by theoretical analysis and validated empirically. The method is naturally compatible with hybrid action of discrete and continuous parts, which is common in practical problems.
3. In contrast to most off-policy methods, the algorithm does not rely on importance sampling with the value of $\pi(a_t|s_t)$ – the action probability of the behavior policy, thus can be used to learn from an unknown policy, and is also robust when current policy is deviated from the behavior policy. We validate this with several empirical experiments.

In Section 5 we give a proposition of policy improvement by theoretical analysis. And in Section 6 we give experimental results of the proposed algorithm in off-policy settings.

5 Theoretical Analysis

In this section, we firstly show that in the ideal case Algorithm 1 is equivalent to imitating a new policy $\tilde{\pi}$. Then we show that the policy $\tilde{\pi}$ is indeed uniformly better than π. Thus Algorithm 1 can also be regarded as *imitating a better policy* (IBP). For function approximation, we also provide a policy improvement lower bound under mild conditions.

5.1 Equivalence to Imitating a New Policy

In this subsection, we show that in the ideal case when we know the advantage $A^{\pi}(s_t, a_t)$, Algorithm 1 is equivalent to minimizing KL divergence between π_θ and a hypothetic $\tilde{\pi}$. Consider the problem

$$\tilde{\pi} = \arg\max_{\pi' \in \Pi} ((1-\gamma)\beta L^{d_\pi, \pi}(\pi') - D_{\text{KL}}^{d_\pi}(\pi'||\pi)) \tag{3}$$

which has an analytical solution in the policy space Π [Azar et al., 2012, Appendix A, Proposition 1]

$$\tilde{\pi}(a|s) = \pi(a|s) \exp(\beta A^{\pi}(s, a) + C(s)) \tag{4}$$

where $C(s)$ is a normalizing factor to ensure that $\sum_{a \in \mathcal{A}} \tilde{\pi}(a|s) = 1$ for each state s. Then

$$\begin{aligned} \arg\min_\theta D_{\text{KL}}^{d}(\tilde{\pi}||\pi_\theta) &= \arg\max_\theta \sum_s d(s) \sum_a \tilde{\pi}(a|s) \log \pi_\theta(a|s) \\ &= \arg\max_\theta \sum_s d(s) \exp(C(s)) \sum_a \pi(a|s) \exp(\beta A^{\pi}(s, a)) \log \pi_\theta(a|s) \end{aligned} \tag{5}$$

Thus Algorithm 1 is equivalent to minimizing $D_{\text{KL}}^{d}(\tilde{\pi}||\pi_\theta)$ for $d(s) \propto d_\pi(s) \exp(-C(s))$. [3]

[2] In our experiments, the average norm of advantage is approximated with a moving average estimation, by $c^2 \leftarrow c^2 + 10^{-8}((R_t - V_\theta(s_t))^2 - c^2)$.

[3] In the implementation of the algorithm, we omit the step discount in d_π, i.e., using $d'_\pi(s) = \mathbb{E}_{d_0, \pi} \sum_{t=0}^{T} \mathbf{1}(s_t = s)$ where T is the terminal step. Sampling from $d_\pi(s)$ is possible, but usually leads to inferior performance according to our preliminary experiments.

5.2 Monotonic Advantage Reweighting

In subsection 5.1, we have shown that the $\tilde{\pi}$ defined in 4 is the analytical solution to the problem 3. In this section, we further show that $\tilde{\pi}$ is indeed uniformly as good as, or better than π. To be rigorous, a policy π' is considered *uniformly as good as, or better* than π, if $\forall s \in \mathcal{S}$, we have $V^{\pi'}(s) \geq V^{\pi}(s)$. In Proposition 1, we give a family of $\tilde{\pi}$ which are uniformly as good as, or better than π. To be specific, we have

Proposition 1. *Suppose two policies π and $\tilde{\pi}$ satisfy*

$$g(\tilde{\pi}(a|s)) = g(\pi(a|s)) + h(s, A^{\pi}(s,a)) \tag{6}$$

where $g(\cdot)$ is a monotonically increasing function, and $h(s,\cdot)$ is monotonically increasing for any fixed s. Then we have

$$V^{\tilde{\pi}}(s) \geq V^{\pi}(s), \ \forall s \in S. \tag{7}$$

that is, $\tilde{\pi}$ is uniformly as good as or better than π.

The idea behind this proposition is simple. The condition (6) requires that the policy $\tilde{\pi}$ has positive advantages for the actions where $\tilde{\pi}(a|s) \geq \pi(a|s)$. Then it follows directly from the well-known *policy improvement theorem* as stated in [Sutton and Barto, 1998, Equation 4.8]. A short proof is provided in Appendix A.1 for completeness.

When $g(\cdot)$ and $h(s,\cdot)$ in (6) are chosen as $g(\pi) = \log(\pi)$ and $h(s, A^{\pi}(s,a)) = \beta A^{\pi}(s,a) + C(s)$, then we recover the formula in 4. By Proposition (1) we have shown that $\tilde{\pi}$ defined in 4 is as good as, or better than policy π.

We note that there are other choice of $g(\cdot)$ and $h(s,\cdot)$ as well. For example we can choose $g(\pi) = \log(\pi)$ and $h(s, A^{\pi}(s,a)) = \log((\beta A^{\pi}(s,a))_+ + \epsilon) + C(s)$, where $(\cdot)_+$ is a positive truncation, ϵ is a small positive number, and $C(s)$ is a normalizing factor to ensure $\sum_{a \in \mathcal{A}} \tilde{\pi}(s,a) = 1$. In this case, we can minimize $D^{d}_{\mathrm{KL}}(\tilde{\pi}||\pi_\theta) = \sum_s d(s) \exp(C(s)) \sum_a \pi(a|s)((\beta A^{\pi}(s,a))_+ + \epsilon) \log \pi_\theta(a|s) + C$.

5.3 Lower bound under Approximation

For practical usage, we usually seek a parametric approximation of $\tilde{\pi}$. The following proposition gives a lower bound of policy improvement for the parametric policy π_θ.

Proposition 2. *Suppose we use parametric policy π_θ to approximate the improved policy $\tilde{\pi}$ defined in Formula 3, we have the following lower bound on the policy improvement*

$$\eta(\pi_\theta) - \eta(\pi) \geq -\frac{\sqrt{2}}{1-\gamma} \delta_1^{\frac{1}{2}} M^{\pi_\theta} + \frac{1}{(1-\gamma)\beta} \delta_2 - \frac{\sqrt{2}\gamma \epsilon_{\pi}^{\tilde{\pi}}}{(1-\gamma)^2} \delta_2^{\frac{1}{2}} \tag{8}$$

where $\delta_1 = \min(D^{d_{\tilde{\pi}}}_{KL}(\pi_\theta||\tilde{\pi}), D^{d_{\tilde{\pi}}}_{KL}(\tilde{\pi}||\pi_\theta))$, $\delta_2 = D^{d_\pi}_{KL}(\tilde{\pi}||\pi)$, $\epsilon_{\pi}^{\pi'} = \max_s |\mathbb{E}_{a \sim \pi'} A^{\pi}(s,a)|$, and $M^{\pi} = \max_{s,a} |A^{\pi}(s,a)| \leq \max_{s,a} |r(s,a)|/(1-\gamma)$.

A short proof can be found in Appendix A.2. Note that we would like to approximate $\tilde{\pi}$ under state distribution $d_{\tilde{\pi}}$ in theory. However in practice we use a heuristic approximation to sample data from trajectories generated by the base policy π as in Algorithm 1, which is equivalent to imitating $\tilde{\pi}$ under a slightly different state distribution d as discussed in Sec.5.1.

6 Experimental Results

In this section, we provide empirical evidence that the algorithm is well suited for off-policy RL tasks, as it does not need to know the probability of the behavior policy, thus is robust when learning from replays from an unknown policy. We evaluate the proposed algorithm with HFO environment under different settings (Sec. 6.1). Furthermore, we also provide two other environments (TORCS and mobile MOBA game) to evaluate the algorithm in learning from replay data (Sec. 6.2, 6.3).

Denote the behavior policy as π, the desired parametrized policy as π_θ, the policy loss L_p for the policy iteration algorithms considered are listed as following: (C is a θ-independent constant)

- (**IL**) Imitation learning, minimizing $D^{d_\pi}_{\mathrm{KL}}(\pi||\pi_\theta)$.

$$L_p = D^{d_\pi}_{\mathrm{KL}}(\pi||\pi_\theta) = -\mathbb{E}_{s \sim d_\pi(s), a \sim \pi(a|s)} \log \pi_\theta(a|s) + C \tag{9}$$

- (**PG**) Policy gradient with baseline and $D_{\text{KL}}^{d_\pi}(\pi||\pi_\theta)$ regularization.

$$L_p = -\mathbb{E}_{s\sim d_\pi(s), a\sim\pi(a|s)}(\beta A^\pi(s,a)+1)\log\pi_\theta(a|s) + C \tag{10}$$

- (**PGIS**) Policy gradient with baseline and $D_{\text{KL}}^{d_\pi}(\pi||\pi_\theta)$ regularization, with off-policy correction by importance sampling (IS), as in TRPO [Schulman et al., 2015] and CPO [Achiam et al., 2017]. Here we simply use penalized gradient algorithm to optimize the objective, instead of using delegated optimization method as in [Schulman et al., 2015].

$$\begin{aligned} L_p &= D_{\text{KL}}^{d_\pi}(\pi||\pi_\theta) - (1-\gamma)\beta L^{d_\pi,\pi}(\pi_\theta) \\ &= -\mathbb{E}_{s\sim d_\pi(s), a\sim\pi(a|s)}\left(\frac{\pi_\theta(a|s)}{\pi(a|s)}\beta A^\pi(s,a) + \log\pi_\theta(s,a)\right) + C \end{aligned} \tag{11}$$

- (**MARWIL**) Minimizing $D_{\text{KL}}^{d}(\tilde{\pi}||\pi_\theta)$ as in (5) and Algorithm 1.

$$L_p = D_{\text{KL}}^{d}(\tilde{\pi}||\pi_\theta) = -\mathbb{E}_{s\sim d_\pi(s), a\sim\pi(a|s)}\log(\pi_\theta(a|s))\exp(\beta A^\pi(s,a)) + C \tag{12}$$

Note that IL simply imitates all the actions in the data, while PG needs the on-policy assumption to be a reasonable algorithm. Both PGIS and MARWIL are derived under off-policy setting. However, the importance ratio π_θ/π used to correct off-policy bias for PG usually has large variance and may cause severe problems when π_θ is deviated far away from π [Sutton and Barto, 1998]. Several methods are proposed to alleviate this problem [Schulman et al., 2017, Munos et al., 2016, Precup et al., 2000]. On the other hand, we note that the algorithm MARWIL is naturally off-policy, instead of relying on the importance sampling ratio π_θ/π to do off-policy correction. We expect the proposed algorithm to work better when learning from a possibly unknown behavior policy.

6.1 Experiments with Half Field Offense (HFO)

To compare the aforementioned algorithms, we employ Half Field Offense (HFO) as our primary experiment environment. HFO is an abstraction of the full RoboCup 2D game, where an agent plays soccer in a half field. The HFO environment has continuous state space and hybrid (discrete and continuous) action space, which is similar to our task in a MOBA game (Sec. 6.3). In this simplified environment, we validate the effectiveness and efficiency of the proposed learning method.

6.1.1 Environment Settings

Like in [Hausknecht and Stone, 2016], we let the agent try to goal without a goalkeeper. We follow [Hausknecht and Stone, 2016] for the settings, as is briefed below.

The observation is a 59-d feature vector, encoding the relative position of several critical objects such as the ball, the goal and other landmarks (See [Hausknecht, 2017]). In our experiments, we use a hybrid action space of discrete actions and continuous actions. 3 types of actions are considered in our setting, which correspond to {"Dash", "Turn", "Kick"}. For each type k of action, we require the policy to output a parameter $x_k \in \mathbb{R}^2$. For the action "Dash" and "Kick", the parameter x_k is interpreted as $(r\cos\alpha, r\sin\alpha)$, with r truncated to 1 when exceeding. Then $\alpha \in [0, 2\pi]$ is interpreted as the relative direction of that action, while $r \in [0, 1]$ is interpreted as the power/force of that action. For the action "Turn", the parameter x_k is firstly normalized to $(\cos\alpha, \sin\alpha)$ and then θ is interpreted as the relative degree of turning. The reward is hand-crafted, written as:

$$r_t = d_t(b,a) - d_{t+1}(b,a) + \mathbb{I}_{t+1}^{kick} + 3(d_t(b,g) - d_{t+1}(b,g)) + 5\mathbb{I}_{t+1}^{goal},$$

where $d_t(b,a)$ (or $d_t(b,g)$) is the distance between the ball and the agent (or the center of goal). $\mathbb{I}_t^{kick} = 1$ if the agent is close enough to kick the ball. $\mathbb{I}_t^{goal} = 1$ if a successful goal happens. We leverage *Winning Rate* to evaluate the final performance:

$$\text{Winning Rate} = \frac{N_{\text{G}}}{N_{\text{G}} + N_{\text{F}}},$$

where N_{G} is the number of goals (G) achieved, N_{F} is the number of failures (F), due to either out-of-time (the agent does not kick the ball in 100 frames or does not goal in 500 frames) or out-of-bound (the ball is out of the half field).

Algorithm 2 Stochastic Gradient Algorithm for MARWIL

Input: Policy loss L_p being one of 9 to 12. base policy π, parameter m, c_v.
Randomly initialize π_θ. Empty replay memory D.
Fill D with trajectories from π and calculate R_t for each (s_t, a_t) in D.
for $i = 1$ **to** N **do**
  Sample a batch $B = \{(s_k, a_k, R_k)\}_m$ from D.
  Compute mini-batch gradient $\nabla_\theta \widehat{L}_p$, $\nabla_\theta \widehat{L}_v$ of B.
  Update θ: $-\Delta\theta \propto \nabla_\theta \widehat{L}_p + c_v \nabla_\theta \widehat{L}_v$
end for

Table 1: Performance of PG and MARWIL in TORCS, where $\beta = 0$ is the case of IL. For consistent performance, β should be inversely proportional to the scale of (normalized) A^π. Different β are tested in the experiments. The performance is evaluated on the sum of rewards per episode.

β	0.0	0.25	0.5	0.75	1.0
PG	2710	6396	6735	6758	7152
MARWIL	(2710)	5583	6832	7670	9492

When learning from data, the historical experience is generated with a mixture of a perfect (100% winning rate) policy π_{perfect} and a random policy π_{random}. For the continuous part of the action, a Gaussian distribution of $\sigma = 0.2$ or 0.4 is added to the model output, respectively. The mixture coefficient ϵ is used to adjust the proportion of "good" actions and "bad" actions. To be specific, for each step, the action is taken as

$$a_t \sim \begin{cases} \pi_{\text{perfect}}(\cdot|s_t) + N(0, \sigma) & \text{w.p. } \epsilon \\ \pi_{\text{random}}(\cdot|s_t) + N(0, \sigma) & \text{w.p. } 1 - \epsilon \end{cases} \tag{13}$$

The parameter ϵ is adjusted from 0.1 to 0.5. Smaller ϵ means greater noise, in which case it is harder for the algorithms to find a good policy from the noisy data.

6.1.2 Algorithm Setting

For the HFO game, we model the 3 discrete actions with multinomial probabilities and the 2 continuous parameters for each action with normal distributions of known $\sigma = 0.2$ but unknown μ. Parameters for different types of action are modeled separately. In total we have 3 output nodes for discrete action probabilities and 6 output nodes for continuous action parameters, in the form of

$$\pi_\theta((k, x_k)|s) = p_\theta(k|s) N(x_k|\mu_{\theta,k}, \sigma), \quad k \in \{1, 2, 3\}, x_k \in \mathbb{R}^2$$

where $p_\theta(\cdot|s)$ is computed as a soft-max for discrete actions and $N(\cdot|\mu_\theta, \sigma)$ is the probability density function of Gaussian distribution.

When learning from data, the base policy (13) is used to generate trajectories into a replay memory D, and the policy network is updated by different algorithms, respectively. We denote the policy loss objective as L_p, being one of the formula (9) (10) (11) (12). Then we optimize the policy loss L_p and the value loss L_v simultaneously, with a mixture coefficient c_v as a hyper-parameter (by default $c_v = 1$). The value loss L_v is defined as $L_v = \mathbb{E}_{d,\pi}(R_t - V_\theta(s_t))^2$. A stochastic gradient algorithm is given in Algorithm 2. Each experiment is repeated 3 times and the average of scores is reported in Figure 1. Additional details of the algorithm settings are given in Appendix B.2.

We note that the explicit value $\pi(a_t|s_t)$ is crucial for the correction used by most off-policy policy iteration methods [Sutton and Barto, 1998], including [Munos et al., 2016, Wang et al., 2016, Schulman et al., 2017, Wu et al., 2017] and many other works [Geist and Scherrer, 2014]. Here for a comparable experiment between policy gradient method and our proposed method, we consider a simple off-policy correction by importance sampling as in (11). We test the performance of the proposed method and previous works under different settings in Figure 1. We can see that the proposed MARWIL achieves consistently better performance than other methods.

6.2 Experiments with TORCS

We also evaluate the imitation learning and the proposed method within the TORCS [Wymann et al., 2014] environment. In the TORCS environment, the observation is the raw screen with image size of

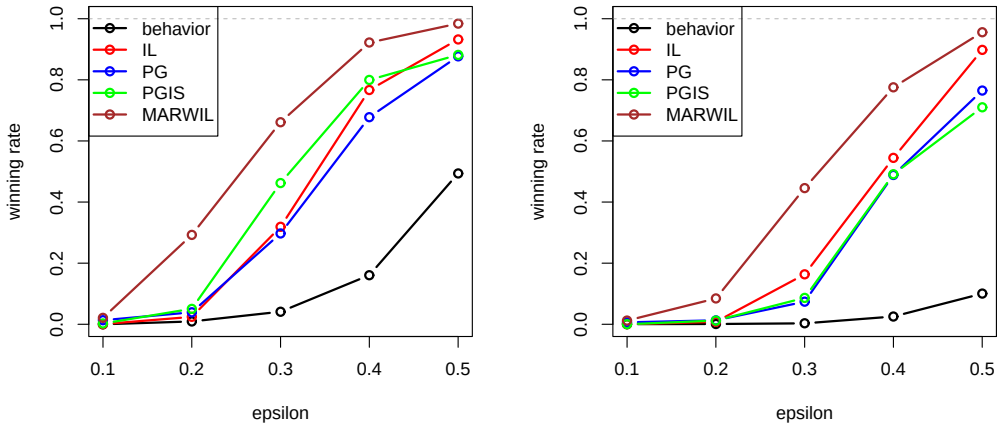

Figure 1: **Left:** Learning from data with additional noise $\sigma = 0.2$. **Right:** Learning from data with additional noise $\sigma = 0.4$. The data is generated with a mixture of a perfect (100% winning rate) policy π_{perfect} and a random policy π_{random}. For the continuous part of the action, a Gaussian noise of $\sigma = 0.2$ (left) or 0.4 (right) is added to the model output, respectively. The mixture coefficient ϵ is used to adjust the proportion of "good" actions and "bad" actions. Smaller ϵ means less "good" actions and harder problem. The performance of the behavior policy is plotted in black. We see that the performance of IL is stable, while PG and PGIS may be affected by the increasing noise in the data. In all settings we see that the proposed algorithm MARWIL performs best in this task.

$64 \times 64 \times 3$, the action is a scalar indicating the steering angle in $[-\pi, \pi]$, and the reward r_t is the momentary speed. When the car crushes, a -1 reward is received and the game terminates.

For the TORCS environment, a simple rule is leveraged to keep the car running and to prevent it from crushing. Therefore, we can use the rule as the optimal policy to generate expert trajectories. In addition, we generate noisy trajectories with random actions to intentionally confuse the learning algorithms, and see whether the proposed method can learn a better policy from the data generated by the deteriorated policy. We make the training data by generating 10 matches with the optimal policy and another 10 matches with the random actions.

We train the imitation learning and the proposed method for 5 epochs to compare their performance. Table 1 shows the test scores when varying the parameter β. From the results, we see that our proposed algorithm is effective at learning a better policy from these noisy trajectories.

6.3 Experiments with King of Glory

We also evaluate the proposed algorithm with *King of Glory* – a mobile MOBA (Multi-player Online Battle Arena) game popular in China. In the experiments, we collect human replay files in the size of millions, equaling to tens of billions time steps in total. Evaluation is performed in the "solo" game mode, where an agent fights against another AI in the opposite side. A DNN based function approximator is adopted. In a proprietary test, we find that our AI agent, trained with the proposed method, can reach the level of an experienced human player in a solo game. Additional details of the algorithm settings for King of Glory is given in Appendix B.3.

7 Conclusion

In this article, we present an off-policy learning algorithm that can form a better policy from trajectories generated by a possibly unknown policy. When learning from replay data, the proposed algorithm does not require the bahavior probability π over the actions, which is usually missing in human generated data, and also works well with function approximation and hybrid action space. The algorithm is preferable in real-world application, including playing video games. Experimental results over several real world datasets validate the effectiveness of the proposed algorithm. We note that the proposed MARWIL algorithm can also work as a full reinforcement learning method, when applied iteratively on self-generated replay data. Due to the space limitation, a thorough study of our method for full reinforcement learning is left to a future work.

Acknowledgement We are grateful for the anonymous reviewers for their detailed and helpful comments on this work. We also thank our colleagues in the project of King of Glory AI, particularly Haobo Fu and Tengfei Shi, for their assistance on the game environment and parsing replay data.

References

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *International Conference on Learning Representations*, 2018.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 22–31, 2017.

Mohammad Gheshlaghi Azar, Vicenç Gómez, and Hilbert J Kappen. Dynamic policy programming. *Journal of Machine Learning Research*, 13(Nov):3207–3245, 2012.

Michael Bain and Claude Sommut. A framework for behavioural cloning. *Machine intelligence*, 15(15):103, 1999.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Rémi Coulom. Bayesian elo rating. `https://www.remi-coulom.fr/Bayesian-Elo/`, 2005. [Online; accessed 9-Feb-2018].

Imre Csiszar and János Körner. *Information theory: coding theorems for discrete memoryless systems*. Cambridge University Press, 2011.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Matthieu Geist and Bruno Scherrer. Off-policy learning with eligibility traces: A survey. *The Journal of Machine Learning Research*, 15(1):289–333, 2014.

Mohammad Ghavamzadeh, Marek Petrik, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306, 2016.

Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in neural information processing systems*, pages 3338–3346, 2014.

Matthew Hausknecht. Robocup 2d half field offense technical manual. `https://github.com/LARG/HFO/blob/master/doc/manual.pdf`, 2017. [Online; accessed 9-Feb-2018].

Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-Arnold, John Agapiou, Joel Leibo, and Audrunas Gruslys. Deep q-learning from demonstrations. In *AAAI Conference on Artificial Intelligence*, 2018.

Daniel Jiang, Emmanuel Ekwedike, and Han Liu. Feedback-based tree search for reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 2284–2293, 2018.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 652–661, 2016.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, pages 267–274, 2002.

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. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

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.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 1607–1612, 2010.

Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315, 2013.

Doina Precup, Richard S Sutton, and Satinder P Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, pages 759–766. Citeseer, 2000.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

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

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pages 2380–2388, 2015.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5285–5294, 2017.

Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner". TORCS, The Open Racing Car Simulator. `http://www.torcs.org`, 2014.

Jiechao Xiong, Qing Wang, Zhuoran Yang, Peng Sun, Lei Han, Yang Zheng, Haobo Fu, Tong Zhang, Ji Liu, and Han Liu. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. *arXiv preprint arXiv:1810.06394*, 2018.

A Proofs

In this section, we provide proofs for the propositions appeared in the main text.

A.1 Proof to Proposition 1

Proof. For a fixed s, consider $A_1 = \{a \in \mathcal{A} \mid \tilde{\pi}(a|s) \geq \pi(a|s)\}$, $A_2 = \{a \in \mathcal{A} \mid \tilde{\pi}(a|s) < \pi(a|s)\}$. Since $g(\cdot)$ and $h(s, \cdot)$ is monotonically increasing, we have

$$\begin{aligned} h(s, A^{\pi}(s, a_1)) &= g(\tilde{\pi}(a_1|s)) - g(\pi(a_1|s)) \\ &\geq g(\tilde{\pi}(a_2|s)) - g(\pi(a_2|s)) \\ &= h(s, A^{\pi}(s, a_2)),\ \forall a_1 \in A_1, a_2 \in A_2 \end{aligned}$$

which means that $\exists\ q(s) \in \mathbb{R}$ s.t.

$$Q^{\pi}(s, a_1) \geq q(s) \geq Q^{\pi}(s, a_2), \quad \forall a_1 \in A_1, a_2 \in A_2$$

Thus

$$\begin{aligned} & \sum_a \tilde{\pi}(a|s) Q^{\pi}(s, a) - \sum_a \pi(a|s) Q^{\pi}(s, a) \\ = & \sum_{a \in A_1} (\tilde{\pi}(a|s) - \pi(a|s)) Q^{\pi}(s, a) + \sum_{a \in A_2} (\tilde{\pi}(a|s) - \pi(a|s)) Q^{\pi}(s, a) \\ \geq & \sum_{a \in A_1} (\tilde{\pi}(a|s) - \pi(a|s)) q(s) + \sum_{a \in A_2} (\tilde{\pi}(a|s) - \pi(a|s)) q(s) \\ = & q(s) \sum_a \tilde{\pi}(a|s) - q(s) \sum_a \pi(a|s) = 0 \end{aligned}$$

Define

$$V_l(s) = \begin{cases} \mathbb{E}_{a \sim \tilde{\pi}(s)} \left(\mathbb{E}_{s', r|s, a}(r + \gamma V_{l-1}(s')) \right), & l \geq 1 \\ V^{\pi}(s), & l = 0 \end{cases}$$

that is, the value of state s if we follow $\tilde{\pi}$ in the first l steps, and then follow π in subsequent steps. So we have just proved

$$V_1(s) \geq V_0(s), \quad \forall s \in S.$$

By induction, we assume that $V_l(s) \geq V_{l-1}(s),\ \forall s \in S$, then

$$\begin{aligned} V_{l+1}(s) &= \mathbb{E}_{a \sim \tilde{\pi}} \left(\mathbb{E}_{s', r|s, a}(r + \gamma V_l(s')) \right) \\ V_l(s) &= \mathbb{E}_{a \sim \tilde{\pi}} \left(\mathbb{E}_{s', r|s, a}(r + \gamma V_{l-1}(s')) \right) \end{aligned}$$

we have $V_{l+1}(s) \geq V_l(s),\ \forall s \in S$. For finite horizon MDP and infinite horizon MDP with $\gamma < 1$, we have

$$V^{\tilde{\pi}}(s) \geq V^{\pi}(s), \quad \forall s \in S$$

□

The proof is for discrete action space only. However it could be generalized to continuous actions and hybrid actions without much difficulties.

A.2 Proof to Proposition 2

Proof. In [Kakade and Langford, 2002] a useful equation is proved that

$$\eta(\pi') - \eta(\pi) = L^{d_{\pi'}, \pi}(\pi') = \frac{1}{1 - \gamma} \sum_s d_{\pi'}(s) \sum_a \pi'(a|s) A^{\pi}(s, a) \tag{14}$$

From Corollary 1 in [Achiam et al., 2017], we have

$$\eta(\pi') - \eta(\pi) \geq L^{d_{\pi}, \pi}(\pi') - \frac{2\gamma \epsilon_{\pi}^{\pi'}}{(1 - \gamma)^2} D_{\mathrm{TV}}^{d_{\pi}}(\pi', \pi) \tag{15}$$

where $\epsilon_{\pi}^{\pi'} = \max_s |\mathbb{E}_{a \sim \pi'} A^{\pi}(s, a)|$. Similarly, we also have

$$\begin{aligned} |\eta(\pi') - \eta(\pi)| &= \frac{1}{1 - \gamma} \sum_s d_{\pi'}(s) \left| \sum_a (\pi'(a|s) - \pi(a|s)) A^{\pi}(s, a) \right| & (16) \\ &\leq \frac{1}{1 - \gamma} \sum_s d_{\pi'}(s) \sum_a |\pi'(a|s) - \pi(a|s)|\, |A^{\pi}(s, a)| & (17) \\ &\leq \frac{2}{1 - \gamma} D_{\mathrm{TV}}^{d_{\pi'}}(\pi', \pi) M^{\pi} & (18) \end{aligned}$$

where $M^\pi = \max_{s,a} |A^\pi(s,a)| \le \max_{s,a} |r(s,a)|/(1-\gamma)$. Define

$$\mathcal{L}(\pi') = (1-\gamma)\beta L^{d_\pi,\pi}(\pi') - D_{\text{KL}}^{d_\pi}(\pi'||\pi) \tag{19}$$

Then

$$\tilde{\pi} = \arg\max_{\pi' \in \Pi} \mathcal{L}(\pi') \tag{20}$$

and $\mathcal{L}(\tilde{\pi}) \ge \mathcal{L}(\pi) = 0$. Now consider

$$\begin{aligned}
\eta(\pi_\theta) - \eta(\pi) &= (\eta(\pi_\theta) - \eta(\tilde{\pi})) + (\eta(\tilde{\pi}) - \eta(\pi)) && (21)\\
&\ge -\frac{2}{1-\gamma} D_{\text{TV}}^{d_{\tilde{\pi}}}(\tilde{\pi}, \pi_\theta) M^{\pi_\theta} + L^{d_\pi,\pi}(\tilde{\pi}) - \frac{2\gamma\epsilon_\pi^{\tilde{\pi}}}{(1-\gamma)^2} D_{\text{TV}}^{d_\pi}(\tilde{\pi}, \pi) && (22)\\
&\ge -\frac{2}{1-\gamma} D_{\text{TV}}^{d_{\tilde{\pi}}}(\tilde{\pi}, \pi_\theta) M^{\pi_\theta} + \frac{1}{(1-\gamma)\beta} D_{\text{KL}}^{d_\pi}(\tilde{\pi}||\pi) - \frac{2\gamma\epsilon_\pi^{\tilde{\pi}}}{(1-\gamma)^2} D_{\text{TV}}^{d_\pi}(\tilde{\pi}, \pi) && (23)
\end{aligned}$$

From Pinsker's inequality [Csiszar and Körner, 2011], we have

$$\begin{aligned}
D_{\text{TV}}^{d}(\pi', \pi) &= \sum_s d(s) D_{\text{TV}}(\pi'(\cdot|s), \pi(\cdot|s)) && (24)\\
&\le \sum_s d(s) \sqrt{\frac{1}{2} D_{\text{KL}}(\pi'(\cdot|s)||\pi(\cdot|s))} \le \sqrt{\frac{1}{2} D_{\text{KL}}^{d}(\pi'||\pi)} && (25)
\end{aligned}$$

where the last inequality comes from Jensen's inequality. Denote $\delta_1 = \min(D_{\text{KL}}^{d_{\tilde{\pi}}}(\pi_\theta||\tilde{\pi}), D_{\text{KL}}^{d_{\tilde{\pi}}}(\tilde{\pi}||\pi_\theta))$ and $\delta_2 = D_{\text{KL}}^{d_\pi}(\tilde{\pi}||\pi)$, we have

$$\eta(\pi_\theta) - \eta(\pi) \ge -\frac{\sqrt{2}}{1-\gamma} \delta_1^{\frac{1}{2}} M^{\pi_\theta} + \frac{1}{(1-\gamma)\beta} \delta_2 - \frac{\sqrt{2}\gamma\epsilon_\pi^{\tilde{\pi}}}{(1-\gamma)^2} \delta_2^{\frac{1}{2}} \tag{26}$$

□

B Discussion

B.1 Connection with regularized policy optimization

In this subsection, we show in a general form, the proposed procedure of Algorithm 1 can recover many interesting algorithms, which are related to previous works. We consider a general regularized policy optimization problem (RPO) as

$$\max_{\theta \in \Theta} \left((1-\gamma) L^{d_\pi,\pi}(\pi_\theta) - \frac{1}{\beta} D_{\text{app}}(\pi, \pi_\theta) \right) \tag{27}$$

where D_{app} is a divergence use for approximation (e.g. KL divergence D_{KL}, Bregman divergence D_ψ). A closely related formulation of Algorithm 1 is

$$\min_{\theta \in \Theta} D_{\text{app}}(\mathcal{O}(\pi), \pi_\theta), \quad \text{where } \mathcal{O}(\pi) = \arg\max_{\pi' \in \Pi} \left((1-\gamma) L^{d_\pi,\pi}(\pi') - \frac{1}{\beta} D_{\text{KL}}(\pi'||\pi) \right) \tag{28}$$

We call this generalized method as *imitating a better policy* (IBP).

B.1.1 RPO, with $D_{\text{app}}(\pi, \pi_\theta) = D_{\text{KL}}^{d_\pi}(\pi||\pi_\theta)$

When $D_{\text{app}}(\pi, \pi_\theta)$ is realized with $D_{\text{KL}}^{d_\pi}(\pi, \pi_\theta)$, the RPO problem is equivalent to the constrained policy optimization problem considered in [Schulman et al., 2015]. The optimization objective is

$$\arg\max_{\theta \in \Theta} (1-\gamma)\beta L^{d_\pi,\pi}(\pi_\theta) - D_{\text{KL}}^{d_\pi}(\pi||\pi_\theta) = \arg\max_{\theta \in \Theta} \mathbb{E}_\pi\left(\frac{\pi_\theta(a|s)}{\pi(a|s)} \beta A^\pi(a|s) + \log \pi_\theta(a|s)\right) \tag{29}$$

B.1.2 RPO, with $D_{\text{app}}(\pi, \pi_\theta) = D_{\text{KL}}^{d_\pi}(\pi_\theta||\pi)$

When $D_{\text{app}}(\pi, \pi_\theta)$ in RPO is set to the forward KL divergence $D_{\text{KL}}^{d_\pi}(\pi_\theta, \pi)$, the optimization objective becomes

$$\arg\max_{\theta \in \Theta} (1-\gamma)\beta L^{d_\pi,\pi}(\pi_\theta) - D_{\text{KL}}^{d_\pi}(\pi_\theta||\pi) = \arg\max_{\theta \in \Theta} \mathbb{E}_\pi\left(\frac{\pi_\theta(a|s)}{\pi(a|s)} (\beta A^\pi(a|s) - \log \frac{\pi_\theta(a|s)}{\pi(a|s)})\right) \tag{30}$$

B.1.3 IBP, with $D_{\text{app}}(\pi, \pi_\theta) = D_{\text{KL}}^{d}(\pi||\pi_\theta)$

For IBP as in (28), when $D_{\text{app}}(\pi, \pi_\theta) = D_{\text{KL}}^{d}(\pi||\pi_\theta)$, the optimization objective becomes

$$\arg\max_{\theta \in \Theta} -D_{\text{KL}}^{d}(\mathcal{O}(\pi)||\pi_\theta) = \arg\max_{\theta \in \Theta} \mathbb{E}_{s \sim d(s), a \sim \pi(a|s)} \exp(\beta A^\pi(s,a) + C(s)) \log(\pi_\theta(a|s)) \tag{31}$$

This is the main algorithm 12 discussed in our work.

Table 2: Connection between the proposed imitating a better policy (IBP) procedure and previous policy optimization methods

Method	$D_{\text{app}} = D_{\text{KL}}(\pi \| \pi_\theta)$	$D_{\text{app}} = D_{\text{KL}}(\pi_\theta \| \pi)$
RPO	$\approx$ TRPO (29)	IBP with KL (30)
IBP (28)	MARWIL (31)	IBP with KL (30)

B.1.4 IBP, with $D_{\textbf{app}}(\pi, \pi_\theta) = D^{d_\pi}_{\textbf{KL}}(\pi_\theta \| \pi)$

When $D_{\text{app}}(\pi, \pi_\theta) = D^{d_\pi}_{\text{KL}}(\pi \| \pi_\theta)$, the algorithm IBP is equivalent to RPO as in Formula (30). In general, for a family of functions $\psi(\cdot)$ we define the Bregman divergence as $D^d_\psi(\pi', \pi) = \sum_s d(s) \Delta_\psi(\pi'(\cdot|s), \pi(\cdot|s))$, where $\Delta_\psi(x, y) = \psi(x) - \psi(y) - \langle \nabla\psi(y), x - y \rangle$, with the inner product $\langle \cdot, \cdot \rangle$ taken on the action space $\mathcal{A}$. We then have

$$\max_{\theta \in \Theta} ((1-\gamma)\beta L^{d_\pi, \pi}(\pi_\theta) - D_\psi(\pi_\theta, \pi)) \tag{32}$$

equivalent to the problem

$$\min_{\theta \in \Theta} D_\psi(\pi_\theta, \mathcal{O}(\pi)), \quad \text{where } \mathcal{O}(\pi) = \arg\max_{\pi' \in \Pi} ((1-\gamma)\beta L^{d_\pi, \pi}(\pi') - D_\psi(\pi', \pi)) \tag{33}$$

To summarize, the proposed procedure is closely related to existing methods like regularized policy optimization. And for special cases, imitating a better policy is equivalent to regularized policy optimization. We present the relationship with previous method in Table 2.

B.2 Additional Details of the Algorithm Settings for HFO

To parametrize the policy and value function, we use a neural network with multiple outputs and shared basic layers. 3 fully connected layers are used as the shared base layers, each having 64 hidden nodes and followed by an ELU [Clevert et al., 2015] activation layer. For outputting probability for discrete actions $k = 1, 2, 3$, a small network of 2 fully connected layers with 32 hidden nodes and 3 soft-maxed outputting nodes are appended to the base layers. For outputting the mean of normal distribution for each action's parameter, we use a 32×6 fully connected 2 layer network after the base layers. We also use a third 32×1 network appended to the base layers to output the state value $V(s_t)$ for each state.

In our implementation, we distribute the algorithm over different servers to speed up the experiments. The "worker" processes which are responsible for generating trajectories to be filled in the replay memory D are deployed on a CPU server. A "trainer" process which is responsible for updating θ is deployed on a GPU server. The replay memory is distributed over the cluster to collect trajectories from workers in parallel and provide batches of data for the trainer.

For these experiments, we use 20 workers and 1 centralized trainer. The maximum capacity of the replay memory for each actor is set to 32 episodes, meaning a total of 640 episodes. In each iteration we randomly sample a batch of 1024 samples from D. The overall loss is the policy loss plus the squared Bellman error of V^π. The basic learning rate is set to 10^{-4}, with β set to 1.0. The learning rate decreases proportional to $1/\sqrt{0.0001T}$. We use RMSProp with weight decay set to 10^{-5} and no momentum. In the experiment, each run is allowed to iterate 100000 batches to converge.

B.3 Additional Details of the Algorithm Settings for King of Glory

The solo mode of King of Glory is similar to those in previous works [Jiang et al., 2018, Xiong et al., 2018], except that we use the hero *Diao Chan* in our experiments. For quantitatively measure, we use a pool of AI agents as opponents, and calculate the Elo ratings [Coulom, 2005] of the agents trained with/without the proposed technique. Experimental results are summarized in Table 3. As can be seen, the agents trained with our proposed method are significantly stronger than those trained with the baseline method (IL). Also, in a proprietary test with colleagues, the *Diao Chan* AI can defeat experienced *XingYao* and *WangZhe* level [4] players in a solo game. We conclude that the proposed method can be successfully used in training AI agents for complex video games with hybrid action space in real-world.

Feature For each frame, we extract 4 types of feature to represent the game state:

1. **Image-like Feature of Global View** The image-like feature covers the whole map of solo mode, with a resolution of 16×64 and 6 channels of allied hero position, allied soldiers' positions, allied towers' defense region, enemy hero position, enemy soldiers' positions, and enemy towers' defense region.

[4] The level of a player in the mobile game is ranked (from lowest to highest) by *QingTong*(Bronze), *BaiYin*(Silver), *HuangJin*(Gold), *BoJin*(Platium), *ZuanShi*(Diamond), *XingYao*(Starshine), and *WangZhe*(King).

Table 3: Performance of the AI agents trained with/without the proposed technique. A total of 40 AI agents trained with different methods are tested in roughly round-robin matches. For comparison, MARWIL1 and IL1 use the same state-features, network structure, and algorithm settings, except for the update formula when computing the gradient. Similarly, MARWIL2 and IL2 use the same settings except for the update formula. The Elo score (higher is better) measures the strength of agents, and the winning ratio is the percentage of games the agent has won. In the results, the best agent is trained with MARWIL method, which reaches an Elo score (higher is better) of 126, and a winning ratio of 64%.

AGENT	ELO	W.RATIO
MARWIL1	126	64%
MARWIL2	72	58%
IL1	-65	41%
IL2	-184	26%

2. **Image-like Feature for Local View** The image-like feature corresponds to the player's screen size of map. The resolution is 32×48 with 12 channels of allied hero position, allied hero attack region, allied soldiers' positions, allied soldiers' HP, allied towers' defense region, allied bullets' damage region, enemy hero position, enemy hero attack region, enemy soldiers' position, enemy soldiers' HP, enemy towers' defense region, and enemy bullets' damage region.
3. **Dense Feature** A 256 dimension dense feature is extracted for each frame. These features include allied and enemy heroes' basic attributes and properties, towers' status and soldiers' status, etc.
4. **Sparse Feature** Two sparse features are provided to indicate the allied and enemy hero types.

Action We use a hybrid of discrete and continuous action space. The action space is defined as $\mathcal{A} = K \times \mathbb{R}^2$, where $K = 6$. The 6 discrete action types are: `NoAction`, `Move`, `Attack`, `Skill1`, `Skill2`, `Skill3`. For $k \in \{$ `Move`, `Skill1`, `Skill2` $\}$, the environment also accepts a "direction" $x_k \in \mathbb{R}^2$ as the action parameter.

Reward We craft 14 dimension rewards as the optimization target, namely `ShortTimeGold`, `LongTimeGold`, `InstantHP`, `ShortTimeHP`, `Kill`, `Death`, `Exp`, `LevelUp`, `Damage`, `DamageToHero`, `TowerDestruct`, `HighTowerDestruct`, `CrystalDestruct`, and `WinLoss`. Different rewards $r^{(k)}$ may have different discount factors $\gamma^{(k)}$. A weighted sum of $R_t^{(k)}$ is used as the final cumulative reward $R_t = \sum_{k=0}^{13} w^{(k)} R_t^{(k)}$, where $R_t^{(k)} = \sum_{l=t}^{T} (\gamma^{(k)})^{l-t} r_l^{(k)}$.

Network Structure We adopt a VGG [Simonyan and Zisserman, 2014] like structure for image-like features: Each "block" consists of 5 layers in the order of Conv-ELU-Conv-ELU-Pooling, with kernel size of 3 for convolution and stride of 2 for max-pooling. By default we use ELU [Clevert et al., 2015] as activation layer for convolution layers and fully-connected layers. For global view image-like feature, 3 "blocks" of size $32\times16\times64$ [5] $\rightarrow 64\times8\times32 \rightarrow 64\times4\times16$ are stacked to extract information from raw image-like features. For local view image-like feature, 3 "blocks" of size $32\times16\times24 \rightarrow 64\times8\times12 \rightarrow 64\times4\times6$ are stacked. Each sparse feature is embedded to a vector of 32 dimension and concatenated together with the dense feature, followed by 2 fully connected layers with 512 hidden nodes. Then these preprocessed representations from image-like features and dense features are all concatenated, followed by 5 fully connected layers of 2048 hidden nodes. 3 final modules consisting of two fully-connected (FC) layers of hidden size 512 and 256 are appended for outputting discrete action probabilities, continuous action parameters, and value estimation for each dimension of rewards, respectively. The whole network structure is depicted as in Figure 2.

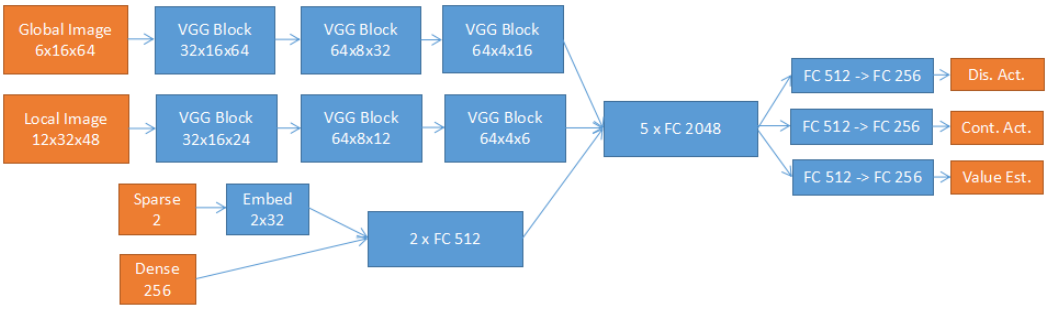

Figure 2: Network structure for our AI agent in King of Glory

[5] We write $C \times H \times W$ for brevity, where C is the number of channels, H is the height, and W is the width.