Addressing Action Oscillations through Learning Policy Inertia

Chen Chen[1*], **Hongyao Tang**[2,1*], **Jianye Hao**[1,2†], **Wulong Liu**[1], **Zhaopeng Meng**[2]
[1]Noah's Ark Lab, Huawei
[2]College of Intelligence and Computing, Tianjin University
chenchen9@huawei.com, bluecontra@tju.edu.cn, {haojianye,liuwulong}@huawei.com, mengzp@tju.edu.cn

Abstract

Deep reinforcement learning (DRL) algorithms have been demonstrated to be effective in a wide range of challenging decision making and control tasks. However, these methods typically suffer from severe action oscillations in particular in discrete action setting, which means that agents select different actions within consecutive steps even though states only slightly differ. This issue is often neglected since the policy is usually evaluated by its cumulative rewards only. Action oscillation strongly affects the user experience and can even cause serious potential security menace especially in real-world domains with the main concern of safety, such as autonomous driving. To this end, we introduce Policy Inertia Controller (PIC) which serves as a generic plug-in framework to off-the-shelf DRL algorithms, to enables adaptive trade-off between the optimality and smoothness of the learned policy in a formal way. We propose Nested Policy Iteration as a general training algorithm for PIC-augmented policy which ensures monotonically non-decreasing updates under some mild conditions. Further, we derive a practical DRL algorithm, namely Nested Soft Actor-Critic. Experiments on a collection of autonomous driving tasks and several Atari games suggest that our approach demonstrates substantial oscillation reduction in comparison to a range of commonly adopted baselines with almost no performance degradation.

Introduction

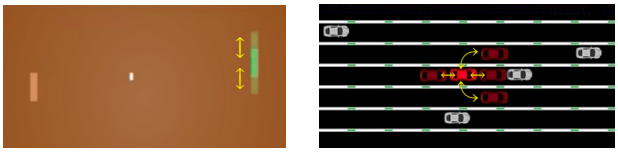
Deep reinforcement learning (DRL) has been widely considered to be a promising way to learn optimal policies in a wide range of practical decision making and control domains, such as Game Playing (Mnih et al. 2015; Silver et al. 2016), Robotics Manipulation (Hafner et al. 2020; Lillicrap et al. 2015; Smith et al. 2019), Medicine Discovery (Popova et al. 2019; Schreck, Coley, and Bishop 2019; You et al. 2018) and so on. One of the most appealing characteristics of DRL is that optimal policies can be learned in a model-free fashion, even in complex environments with high-dimensional state and action space and stochastic transition dynamics.

However, one important problematic phenomenon of DRL agent is *action oscillation*, which means that a well-trained agent selects different actions within consecutive steps during online execution though the states only differ slightly, which leads to shaky behaviors and jerky trajectories. Albeit the agent can achieve good task-specific rewards in simulation, the action oscillation may strongly affect the user experience in many practical interactive applications and exacerbate the wear and tear of a real physical agent. More crucially, the induced abnormal behavior can cause potential security menace in such as autonomous driving scenarios, where safety is the very first requirement. In a nutshell, action oscillation inhibits the deployment of DRL agents in many real-world domains.

Action oscillation can be widely observed for both deterministic policies and stochastic policies. For deterministic policies like Deep Q-Network (DQN) (Mnih et al. 2015), the underlying causes may come from the complexity of deep function approximation with high-dimensional inputs and stochastic noises due to partial observation and random sampling. This issue can be more inevitable for stochastic policies. For example, an entropy regularizer is often adopted in policy-based approaches; moreover, maximum entropy approaches, e.g., Soft Actor-Critic (SAC) (Haarnoja et al. 2018b), take policy entropy as part of optimization objective. Such approaches encourage diverse behaviors of policy for better exploration and generalization, thus aggravate the oscillation in actions in turn. Figure 1 shows two exemplary scenarios in which 'unnatural' and unnecessary oscillations in actions are often observed for learned policies. It deserves to note that the action oscillation issue we study in this paper is different from the inefficient shaky or unstructured exploration behaviors studied in previous works (Sutton and Barto 2018; Haarnoja et al. 2018b; Korenkevych et al. 2019; Haarnoja et al. 2018a; Kendall et al. 2019). On the contrary to exploration, we care about how to address action oscillation during online execution (i.e., exploitation).

As previously explained, action oscillation is often neglected since mainstream evaluations of DRL algorithms are based on expected returns, which might be sufficient for games yet ignores some realistic factors in practical applications. We are aware that the idea of action repetition (Durugkar et al. 2016; Lakshminarayanan, Sharma, and Ravindran 2016; Sharma, Lakshminarayanan, and Ravin-

[*]Equal contributions. This work is done when Hongyao Tang is an intern at Noah's Ark Lab, Huawei.
[†]Corresponding author.



(a) *Atari-Pong* (b) *Highway-Overtaking*

Figure 1: Examples of action oscillation of DRL policies. (a) In *Atari-Pong*, a well-performing DQN agent often shows unnecessary up-down shakes in actions when controlling the bat; (b) In *Highway-Overtaking*, a car agent learned by SAC can have frequent shifts between lane-change (i.e., left and right) and speed control (i.e., accelerate and decelerate) actions when driving on the lane.

dran 2017; Metelli et al. 2020) that repeatedly executes a chosen action for a number of timesteps to improve policy performance, can potentially be used to alleviate action oscillation. To be specific, beyond the usual static frame skip mechanism, Lakshminarayanan et al. (2016) propose dynamic frame skip to jointly learn dynamic repetition numbers through extending the action space with different repetition configurations. Similarly in (Durugkar et al. 2016), an action repetition policy is learned from the perspective of macro-actions that constitutes the same action repeated a number of times. Further, Sharma et al. (2017) introduces FiGAR structure to address the scalability issue of above works by learning factored policies for actions and action repetitions separately. Recently, Metelli et al. (2020) analyze how the optimal policy can be affected by different fixed action repetition configurations, and present an algorithm to learn the optimal value function under given configurations.

In general, action repetition utilizes temporal abstraction, which offers potential advantage in obtaining smooth policies. However, temporal abstraction can decrease the sample efficiency since more simulation steps are needed to meet the same number of transition experiences when compared to a flat policy. Besides, it is unclear whether the dynamic changes in underlying model (i.e., Semi-Markov Decision Process) will hamper the learning process. From another angle, Shen et al. (Shen et al. 2020) propose to enforce smoothness in the learned policy with smoothness-inducing regularization which can relieve the action oscillation problem in some degree. However, such constrains cannot guarantee the smoothness of one action sequence during execution, due to the essential difference of policy smoothness and action sequence smoothness.

In this paper, we propose Policy Inertia Controller (PIC) to address the action oscillation of DRL algorithms in discrete action space setting. PIC serves as a general accessory to conventional DRL policies (called *policy core* in the following) that adaptively controls the persistence of last action (i.e., *Inertia*) according to the current state and the last action selected. A PIC-augmented policy is built in a form of the linear combination between the distribution of the policy core and a Dirac distribution putting all mass on the last action. Rather than introducing extra regularization and reward shaping in learning process, such a structure provides a more direct and fine-grained way of regulating the smoothness of the policy core at each time step. We also theoretically prove the existence of a family of smoother policies among PIC-augmented policies with equivalent or better performance than the policy core.

Further, we introduce Nested Policy Iteration (NPI) as a general training algorithm for PIC-augmented policy, consisting of an outer policy iteration process for the PIC module and an inner one for policy core that nested in the former. We show that NPI can achieve monotonically non-decrease policy updates under some conditions. Finally, we derive a practical DRL algorithm, namely Nested Soft Actor-Critic (NSAC), as a representative implementation of PIC framework with Soft Actor-Critic (SAC) algorithm (Haarnoja et al. 2018b). We demonstrate the effectiveness and superiority of our approach in addressing action oscillation among a variety of autonomous driving environments in *Highway* simulator and several Atari games in OpenAI Gym.

Our main contributions are summarized as follows:

- We propose Policy Inertia Controller (PIC) as a generic framework to address action oscillation issue, which is of significance to practical applications of DRL algorithms.
- We propose a novel Nested Policy Iteration as a general training algorithm for the family of PIC-augmented policy, as well as analyze the conditions when non-decrease policy improvement can be achieved.
- Our extensive empirical results in a range of autonomous driving tasks and Atari game tasks show that our proposed approach can achieve substantial oscillation reduction at almost no performance degradation.

Background

Markov Decision Process

We consider a Markov Decision Process (MDP) $\mathcal{M} := \langle S, A, r, P, \rho_0, \gamma, T \rangle$ where S is the state space, A is the finite action space, $r : S \times A \to \mathbb{R}$ the bounded reward function, $P : S \times A \times S \to \mathbb{R}_{\in[0,1]}$ is the transition probability distribution, $\rho_0 : S \to \mathbb{R}_{\in[0,1]}$ is the initial state distribution, γ is the discount factor that we assume $\gamma \in [0, 1)$ and T is the episode horizon. The agent interacts with the MDP at discrete timesteps by performing its policy $\pi : S \times A \to \mathbb{R}_{\in[0,1]}$, generating a trajectory of states and actions, $\tau = (s_0, a_0, s_1, a_1, \dots s_T, a_T)$, where $s_0 \sim \rho_0(s)$, $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim P(\cdot|s_t, a_t)$. The objective of a reinforcement learning agent is to find a policy that maximize the expected cumulative discounted reward:

$$J(\pi) = \mathbb{E}_{s_t, a_t \sim \rho_\pi} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right], \tag{1}$$

where ρ_π is the state-action marginals of the trajectory distribution induced by policy π. Thus, the optimal policy is $\pi^* = \arg\max_\pi J(\pi)$.

In reinforcement learning, the state-action value function $Q : S \times A \to \mathbb{R}$ is defined as the expected cumulative discounted reward for selecting action a in state s, then following a policy π afterwards: $Q^\pi(s, a) =$

$\mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)|s_0=s,a_0=a\right]$. Similarly, the state value function V denotes value under a certain state s, i.e., $V^\pi(s)=\mathbb{E}_\pi\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)|s_0=s\right]$.

Soft Actor-Critic

Soft Actor-Critic (SAC) (Haarnoja et al. 2018b) is an off-policy reinforcement learning algorithm that optimizes a stochastic policy that maximizes the maximum entropy objective:

$$J_{\text{Ent}}(\pi)=\mathbb{E}_{s_t,a_t\sim\rho_\pi}\left[\sum_{t=0}^{T}\gamma^t\big(r(s_t,a_t)+\alpha\mathcal{H}(\pi(\cdot|s_t))\big)\right] \quad (2)$$

where the temperature parameter α determines the relative importance of the reward versus the policy entropy term at state s_t, $\mathcal{H}(\pi(\cdot|s_t))=-\mathbb{E}_{a_t\sim\pi}\log\pi(\cdot|s_t)$.

SAC uses *soft policy iteration* which alternates between policy evaluation and policy improvement within the maximum entropy framework. The policy evaluation step involves computing the values of policy π through repeatedly applying the modified Bellman backup operator T^π as:

$$\mathcal{T}^\pi Q(s_t,a_t)=r(s_t,a_t)+\gamma\mathbb{E}_{s_{t+1}\sim P}[V(s_{t+1})], \quad (3)$$
$$\text{where } V(s_t)=\mathbb{E}_{a_t\sim\pi}\big[Q(s_t,a_t)-\alpha\log(\pi(a_t|s_t))\big],$$

where $Q(s_t,a_t)$ and $V(s_t)$ here denote the soft variants of value functions. The policy improvement step then involves updating the policy towards the exponential of the soft Q-function, with the overall policy improvement step given by:

$$\pi_{\text{new}}=\arg\min_{\pi\in\Pi}D_{\text{KL}}\left(\pi(\cdot|s_t)\|\frac{\exp\left(\frac{1}{\alpha}Q^{\pi_{\text{old}}}(s_t,\cdot)\right)}{Z^{\pi_{\text{old}}}(s_t)}\right), \quad (4)$$

where Π denotes the policy space and the partition function $Z^{\pi_{\text{old}}}(s_t)$ is intractable but does not contribute to the gradient with respect to the new policy thus can be ignored.

With continuous states, the soft Q-function $Q_\theta(s_t,a_t)$ is approximated with parameters θ via minimizing the soft Bellman residual according to (3). The policy $\pi_\phi(a_t|s_t)$ that parameterized with parameters ϕ is learned by minimizing the expected KL-divergence (4). In the original paper, the authors propose the practical algorithm in continuous action setting by applying the reparameterization trick to Gaussian policy π and utilize an additional V-network to stable the training. Two separate Q-networks are adopted and the minimum of them is used as Q estimates to reduce the overestimation issue (v. Hasselt 2010). The temperature parameter α can also be learned to automatically adapt through introducing a target entropy hyperparameter and optimizing the dual problem (Haarnoja et al. 2018c).

The discrete action version of SAC is derived in (Christodoulou 2019), where the policy network outputs a multinomial over finite actions rather than a Gaussian distribution so that V-function can be estimated directly and no long need Monte-Carlo estimates. In this paper, we focus on discrete action setting and consider the discrete SAC as policy core by default. For continuous action case, our approach can also be applied with a few modification which is beyond the scope of this paper and we leave it for future work.

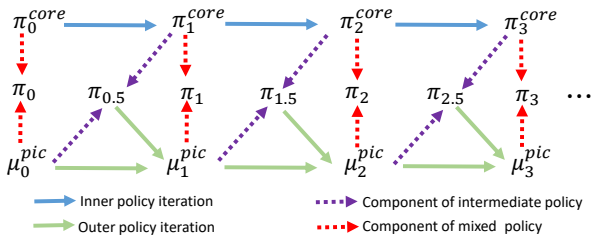


Figure 2: Sketch map of Nested Policy Iteration, π_t and π_t^{core} denote the mixed policy and policy core at t respectively, and $\pi_{t.5}$ denotes the intermediate status of π_t when policy core π_t^{core} has been updated in the inner policy iteration yet μ^{pic} has not.

Approaches

In this section, we first introduce the Policy Inertia Controller (PIC) framework, then we propose Nested Policy Iteration (NPI) to train PIC and the policy core in a general way. Finally, we propose a practical algorithm, Nested Soft Actor-Critic with Policy Inertia Controller (PIC-NSAC), that combines PIC framework and SAC algorithm.

Policy Inertia Controller Framework

Before introducing Policy Inertia Controller (PIC), we first introduce a practical metric that measures the degree of action oscillation of a policy π formally. We define action oscillation ratio $\xi(\pi)$ below:

$$\xi(\pi)=\mathbb{E}_{\tau\sim\rho_\pi^\tau}\left[\frac{1}{T}\sum_{t=1}^{T}\left(1-\mathbb{I}_{\{a_{t-1}\}}(a_t)\right)\right], \quad (5)$$

where ρ_π^τ is the distribution of state-action trajectory induced by policy π and $\mathbb{I}_A(x)$ denotes the indicator function with set A. Intuitively, $\xi(\pi)$ indicates how smooth the actions selected by policy π are when acting in the environment. The lower of $\xi(\pi)$ means the smoother of π. A high $\xi(\pi)$ means that the policy tends to select different actions within consecutive timesteps, i.e., more severe action oscillation.

One straightforward way to address action oscillation is to introduce reward shaping of adding action inconsistency penalty or use similar regularization according to Equation (5). However, the drawbacks of such mechanisms are apparent: they alter the original learning objective and the hyperarameters involved need to be tuned for different tasks. Moreover, such approaches have no guarantee on how the smoothness of policy learned will be. To this end, we propose Policy Inertia Controller (PIC) that regulates a DRL policy distribution directly as follows:

$$\begin{aligned}\pi(\cdot|s_t,a_{t-1})=&\,\mu^{\text{pic}}(s_t,a_{t-1})\delta(a_{t-1})\\&+\left(1-\mu^{\text{pic}}(s_t,a_{t-1})\right)\pi^{\text{core}}(\cdot|s_t),\end{aligned} \quad (6)$$

where $\delta(a_{t-1}):A\to\mathbb{R}^{|A|}_{\in[0,1]}$ denotes a discrete Dirac function that puts all probability mass on the action executed at timestep $t-1$, and π^{core} denotes the policy

Algorithm 1 Nested Policy Iteration (NPI) for PIC-augmented Policy

Input: Policy evaluation and policy improvement processes $\mathcal{E}_{\text{in}}$, $\mathcal{I}_{\text{in}}$ for mixed policy π^{core}, and $\mathcal{E}_{\text{out}}$, $\mathcal{I}_{\text{out}}$ for mixed policy π (Equation (6))

```
Initialize policy core π^core and its corresponding mixed policy π
for Outer policy iteration number t_out do
    for Inner policy iteration number t_in do
        Evaluate the values of π^core until convergence with E_in
        Improve π^core according to I_in
    end for
    Evaluate the values of π until convergence with E_out
    Improve π (i.e., update μ^pic) according to I_out while keep π^core fixed
end for
```

core that is trained as usual. The policy inertia controller $\mu^{\text{pic}}(s_t, a_{t-1}) : S \times A \to \mathbb{R}_{\in[0,1]}$ outputs a scalar weight and the final policy $\pi(\cdot|s_t, a_{t-1})$ is a linear combination of the Dirac and the policy core. We also call the PIC-augmented policy $\pi(\cdot|s_t, a_{t-1})$ as *mixed policy* in the following of this paper for distinction to policy core. In another view, $\mu^{\text{pic}}(s_t, a_{t-1})$ can be viewed as a 1-dimensional continuous policy that regulates the inertia of policy core (i.e., the persistence of last action) depending on current state and the last action.

Next, we show that the structure of the mixed policy has the appealing property to ensure the existence of a family of smoother policies with equivalent or better performance than the policy core (Theorem 0.1) as below:

Theorem 0.1 *Given any policy core $\pi^{\text{core}}(\cdot|s)$, there exists some μ^{pic} such that $\xi(\pi) \leq \xi(\pi^{\text{core}})$ and $J(\pi) \geq J(\pi^{\text{core}})$, where π is the corresponding mixed policy counterpart (Equation 6).*

Detailed proof is provided in Supplementary Material A.1. Theorem 0.1 implies that we can obtain a better policy in terms of both action oscillation rate and expected return through optimizing $\mu^{\text{pic}}(s_t, a_{t-1})$.

The mixed policy π defined in Equation (6) is a nested policy in which the policy core π^{core} is nested. In next section we introduce a general algorithm to train the nested policies, i.e., π^{core} and μ^{pic}, together.

Nested Policy Iteration for PIC-augmented Policy

In this section, we propose Nested Policy Iteration (NPI), a general training algorithm for the family of mixed policy defined in Equation (6). As in Algorithm 1, NPI consists of an outer policy iteration and an inner policy iteration which is nested in the former one. The policy core π^{core} is trained as usual according to inner policy iteration. The outer policy iteration is in the scope of the mixed policy π yet only the PIC module μ^{pic} is updated during outer policy improvement. The sketch map of NPI is shown in Figure 2. In the following, we show that our proposed NPI can achieve monotonically non-decreasing updates for the mixed policy π.

First, we show that an improvement for policy core π^{core} during the inner policy iteration can induce an improvement for the mixed policy π as well (Lemma 0.1). We use $Q^{\text{core}}(s_t, a_t)$, $Q(s_t, a_{t-1}, a_t)$ to denote the Q-functions of $\pi^{\text{core}}(\cdot|s_t)$ and $\pi(\cdot|s_t, a_{t-1})$ respectively, and also adopt subscripts *old* and *new* to indicate Q-functions and policies before and after one policy improvement iteration. Specially, we further use subscript *mid* to denote the intermediate status of the mixed policy π when π^{core} has been updated in the inner iteration yet μ^{pic} has not, i.e., $\pi_{\text{mid}}(\cdot|s_t, a_{t-1}) = \mu^{\text{pic}}_{\text{old}}\delta(a_{t-1}) + \pi^{\text{core}}_{\text{new}}(\cdot|s_t)$. Now we formalize above result in the following lemma.

Lemma 0.1 *(Intermediate Policy Improvement). Given an inner policy iteration with policy improvement (i.e., $\pi^{\text{core}}_{\text{old}} \to \pi^{\text{core}}_{\text{new}}$ or $\pi_{\text{old}} \to \pi_{\text{mid}}$) that ensures $Q^{\text{core}}_{\text{new}}(s_t, a_t) \geq Q^{\text{core}}_{\text{old}}(s_t, a_t)$ for all (s_t, a_t), and assume $\mu^{\text{pic}}_{\text{old}}(s_t, a_{t-1})$ satisfies the following inequality,*

$$\mu^{\text{pic}}_{\text{old}}(s_t, a_{t-1}) \leq \frac{\min_{(s_t,a_t)}(Q^{\text{core}}_{\text{new}}(s_t, a_t) - Q^{\text{core}}_{\text{old}}(s_t, a_t))}{N \cdot C_0(\sum_t^T t\gamma^t)}$$

for all (s_t, a_t, a_{t-1}), where $N \geq 4$ and C_0 is the upper bound of both $A^{\pi^{\text{core}}_{\text{new}}}$ and $A^{\pi_{\text{old}}}$, then we have

$$\begin{aligned} & Q_{\text{mid}}(s_t, a_{t-1}, a_t) - Q_{\text{old}}(s_t, a_{t-1}, a_t) \\ \geq \quad & (1 - \frac{4}{N}) \min_{(s_t,a_t)}(Q^{\text{core}}_{\text{new}}(s_t, a_t) - Q^{\text{core}}_{\text{old}}(s_t, a_t)) \end{aligned}$$

for all (s_t, a_{t-1}, a_t) tuples.

Detailed proof is provided in Supplementary Material A.2.

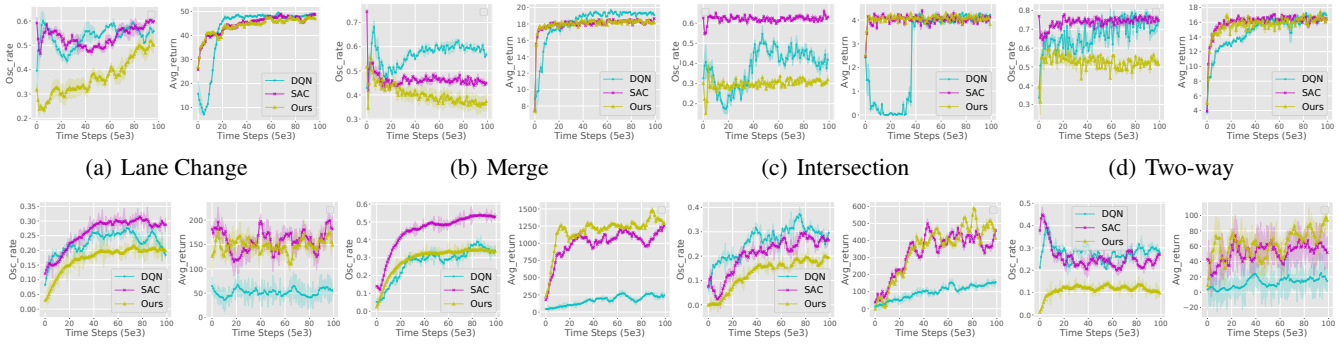
Remark 0.1 *Lemma 0.1 implies that the improvement of policy core by inner policy iteration can also bring an improvement to the mixed policy given that $\mu^{\text{pic}}_{\text{old}}$ is appropriately small. Besides, the upper bound of tolerated $\mu^{\text{pic}}_{\text{old}}$ to guarantee such improvement and the increment from Q_{old} to Q_{mid} are both linearly dependent on the increment of policy core, which means that bigger increment in the inner iteration can tolerate bigger policy inertia $\mu^{\text{pic}}_{\text{old}}$, and the increment for the intermediate mixed policy is proportional to the increment of policy core.*

Next, the full NPI algorithm alternates between the inner policy iteration and the outer policy iteration steps, and it provably leads to a nested policy improvement (Lemma 0.2) and then monotonically non-decreasing updates for the mixed policy π during overall NPI process (Theorem 0.2).

Lemma 0.2 *(Nested Policy Improvement). Based on Lemma 0.1, given an outer policy iteration with policy improvement (i.e., $\mu^{\text{pic}}_{\text{old}} \to \mu^{\text{pic}}_{\text{new}}$ or $\pi_{\text{mid}} \to \pi_{new}$) that ensures $Q_{\text{new}}(s_t, a_{t-1}, a_t) \geq Q_{\text{mid}}(s_t, a_{t-1}, a_t)$ for all $(s_t, a_{t-1}, a_t) \in S \times A \times A$, then we have $Q_{\text{new}}(s_t, a_{t-1}, a_t) \geq Q_{\text{old}}(s_t, a_{t-1}, a_t)$.*

Proof for Lemma 0.2 can be easily obtained by chaining the inequalities between Q_{new}, Q_{mid} and Q_{old}, which can be viewed as a two-step improvement obtained by inner and outer policy iteration respectively.

Theorem 0.2 *(Nested Policy Iteration). By repeatedly applying Lemma 0.2, the mixed policy π achieves monotonically non-decreasing updates.*



(a) Lane Change (b) Merge (c) Intersection (d) Two-way

(e) SpaceInvaders (f) MsPacman (g) Qbert (h) JamesBond

Figure 3: Training curves of algorithms on *(a)-(d)* four *Highway* tasks and *(e)-(h)* four Atari games w.r.t. action oscillation ratio (left) and returns (right). Our approach (red) consistently achieves lower action oscillation rate while retaining comparable performance across all tasks. The horizontal axis denotes time step. Results are means and one stds over 5 random seeds.

Proof for Theorem 0.2 is a straightforward extension of Lemma 0.2 in an iterative fashion under mild conditions.

According to *Generalized Policy Iteration* (GPI) (Sutton and Barto 2018), almost any RL algorithm can be interpreted in a policy iteration fashion. Therefore, NPI can be viewed as a special case of GPI that combines any two RL algorithms for μ^{pic} and π^{core} respectively, since no assumption is made on the choice of both the outer and the inner policy iteration. To make above theoretical results hold, the inner and the outer policy iteration should have the same learning objective, e.g., both common RL objective (Equation (1)) or maximum entropy objective (Equation (2)).

Remark 0.2 *The outer policy iteration of NPI algorithm is in the scope of the mixed policy π that consists of μ^{pic} and π^{core}. Since the PIC module $\mu^{\text{pic}}(s_t, a_{t-1})$ can also be viewed as a continuous policy, one may conduct the outer policy iteration (i.e., an RL algorithm) in the scope of μ^{pic} solely (instead of the mixed policy π). However, such approach can be flawed in two aspects: first, the update signal can be very weak and stochastic since μ^{pic} only indirectly influences the action selected; moreover, the time-variant updates of the policy core are not considered explicitly during the evaluation process, thus resulting in a non-stationary environment for the learning of μ^{pic}.*

Nested Soft Actor-Critic

Based on the general NPI algorithm presented in previous section, we further derive a practical implementation with function approximation for continuous state space domains. To be specific, we propose Nested Soft Actor-Critic with Policy Inertia Controller (PIC-NSAC) to train the PIC module μ^{pic} and the policy core π^{core} simultaneously.

For the inner policy iteration, we resort to SAC algorithm because it is a representative of maximum entropy RL approaches that may typically suffer from action oscillation issue as we mentioned before. We parameterize the policy core $\pi^{\text{core}}_{\phi}(\cdot|s_t)$ with parameter ϕ and update parameters ϕ and θ by SAC algorithm as usual. As to the outer policy iteration, we also use SAC algorithm in the scope of the mixed policy to optimize the same objective as the inner one. We approximate PIC module as $\mu^{\text{pic}}_{\varphi}(s_t, a_{t-1})$ with parameter φ, thus obtain parameterized mixed policy $\pi_{\phi,\varphi}$. The value networks of $\pi_{\phi,\varphi}$ is approximated during soft policy evaluation (Equation (3)) and only parameter φ is updated during soft policy improvement (Equation (4)). All data used above comes from a replay buffer of past experiences collected using the mixed policy $\pi_{\phi,\varphi}$.

The overall algorithm (Algorithm 2) and complete formulations are provided in Supplementary Material B.

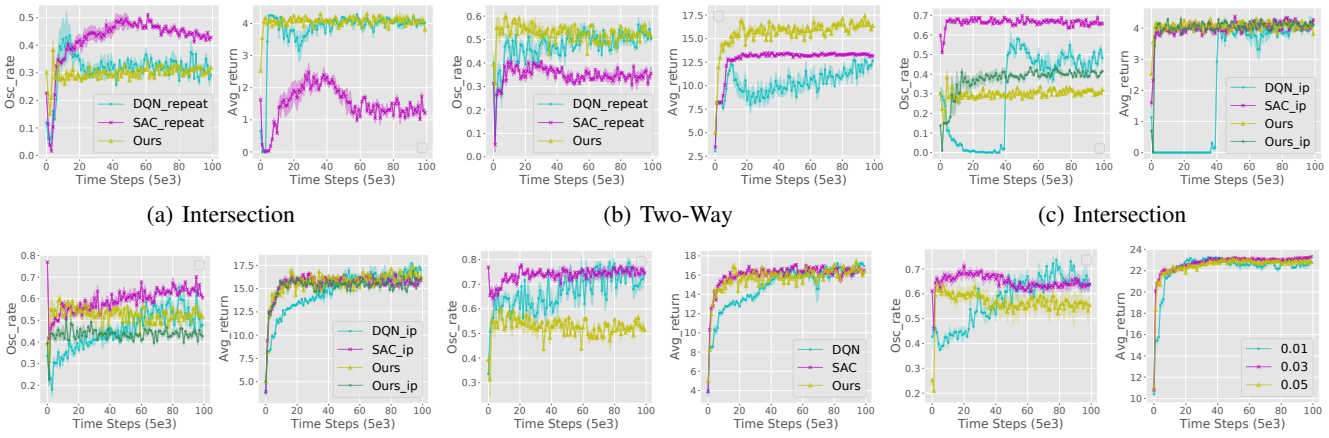
Experiments

This section presents the experimental results of our approach. We first provide the setups and the benchmark approaches in Section **Setups**, and then followed by evaluation results and an analysis study in Section **Results** and Section **Analysis Study**. Finally, we analyze the learned regularization of smoothness for further insights of PIC frameworks.

Setups

Environments and Benchmark Approaches. We use the *Highway* simulator[1] which includes a collection of autonomous driving scenarios, as well as several Atari games in OpenAI-Gym in our experiments. Highway simulator have been used in previous works (Leurent and Maillard 2019; Leurent and Mercat 2019; Li et al. 2019; Carrara et al. 2019) and four typical tasks, i.e., *Lane Change*, *Merge*, *Intersection* and *Two-Way* are picked to conduct experiments. The state of these tasks are mainly about locomotion and kinematics of vehicles. The action space are discrete, consisting of vehicle controls, e.g., left/right lane change, accelerate/decelerate. Detailed configuration of the tasks are provided in Supplementary Material C. For OpenAI Atari, we use *MsPacman-v4*, *SpaceInvaders-v4*, *Qbert-v4* and *JamesBond-v4* with pixel-input states. We compare against benchmark approaches algorithms, including DQN (Mnih et al. 2015), discrete SAC (Christodoulou 2019) and

[1]Highway environments are originally provided at https://github.com/eleurent/highway-env.



(a) Intersection (b) Two-Way (c) Intersection

(d) Two-Way (e) Complex Scenario (f) Simple Scenario

Figure 4: Comparisons results for *(a)-(b)* Action Repetition, *(c)-(d)* Reward Shaping, and *(e)-(f)* Complex v.s. Simple scenarios.

their variants with dynamic action repetition and reward shaping tricks (see Section). All experimental details are provided Supplementary material C.1.

Training and Evaluation. We train five different instances of each algorithm with different random seeds, with each performing 20 evaluation rollouts with some other seed every 5000 environment steps. For each task we report undiscounted return and the action oscillation ratio (Equation (5)) which is calculated from the evaluation trajectories. Concretely, for each episode i, we record the count of action switch within consecutive steps, denoted by c_i, and the total steps of the episode, denoted by n_i, then oscillation ratio is computed as $(\sum_{i=1}^{20} c_i/n_i)/20$. The solid curves corresponds to the mean and the shaded region to half a standard deviation over the five trials.

Results

Evaluations. We first compare NSAC (ours) with DQN and SAC across all 4 Highway tasks and 4 Atari games. The results are shown in Figure 3. We see that our approach achieves substantial reduction with respect to the action oscillation rate than benchmark approaches, especially when compared with SAC, while retaining comparable performance across all tasks. We credit the results to the smoothness property (Theorem 0.1) of mixed policy and the effectiveness of NPI (Theorem 0.2).

Comparison with Action Repetition. Further, we absorb the core idea of action repetition works (Durugkar et al. 2016; Lakshminarayanan, Sharma, and Ravindran 2016; Sharma, Lakshminarayanan, and Ravindran 2017) into another two variants of DQN and SAC, that learn both actions and action repetitions from extended action space. Concretely, we set the repetition set as $Re = \{1, 2, 4, 8\}$, which means that the action is repeated for $1, 2, 4, 8$ times, then the augmented action space A' is the Cartesian product $A \times Re$. *DQN-repeat* baseline and *SAC-repeat* baseline mean that DQN and SAC are trained on the augmented action space A', respectively. We present representative comparison results in *Intersection* and *Two-Way* as shown in Figure 4(a) -4(b). The results show that the action repetition approaches can achieve certain reduction in action oscillation yet sacrificing performance when comparing with our approach within the same environment steps. This is because action repetition hampers sample efficiency due to temporal abstraction as we discuss before.

Comparison with Reward Shaping. Moreover, we also consider the setting where the reward is shaped with action inconsistent penalty, to some degree, this can be viewed to be equivalent to inject a regularizer based on $negative$ action oscillation ration defined in Equation 5. *DQN-ip*, *SAC-ip* and *Ours-ip* are DQN, SAC, NSAC algorithms trained on the environments with action inconsistency penalty -0.05 within consecutive steps yet evaluated without it. The results in Figure 4(c)-4(d) show such reward shaping is effective in reducing oscillation in *Two-Way* for all DQN, SAC and our approach, while cause a counterproductive result in *Intersection*. We conjecture it is because action inconsistent penalty violates the original reward structure (sparse reward in *Intersection*) then the learned policies tend to fall into the unexpected local minimum, revealing the poor scalability of such reward shaping treatment. Additionally, reward shaping can be exhaustive and even impossible in complex problems.

Analysis Study

In this section, we conduct several analysis studies to further examine our approach from three aspects: the performance in complex scenario against simple scenario, the influence of an extra lower bound on the output of PIC module μ^{pic} and the effect of temperature parameter α of the mixed policy π.

Simple v.s. Complex Scenarios. To compare how the complexity of the environments affects the performance, we conduct experiments on both the complex scenarios and simple scenarios on *Two-way* task, where the vehicles number in complex scenarios doubles that in simple scenarios. Figure 4(e)-4(f) indicates that the oscillation reduction is more significant in complex cases with a large margin. This is as

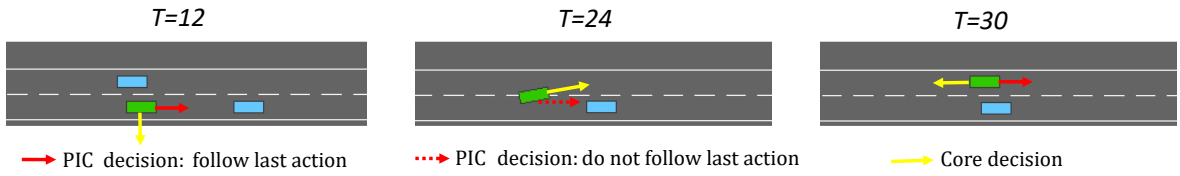


Figure 5: Visualization of several consecutive frames in *Two-Way*. Green denotes the vehicle controlled by the mixed policy learned with NSAC and blue denotes other vehicles to overtake. Red and yellow arrows illustrates the decisions of the PIC module and policy core. The solid arrow denote a high value or probability (≥ 0.5) and the dashed is for a low one. We see that the PIC module gives good regulation when improper oscillations happen ($t = 12$ and $t = 14$) while impose few regulation to follow the previous action for necessary change ($t = 24$).

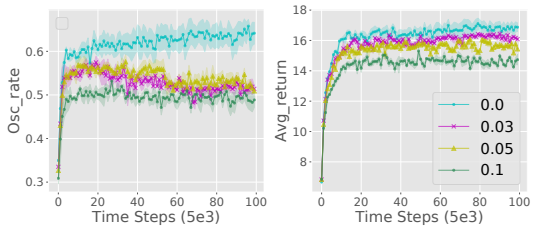


Figure 6: Training curves of our algorithm on the complex scenario of *Two-Way* with different lower bound values μ_0 of the policy inertia controller μ^{pic}.

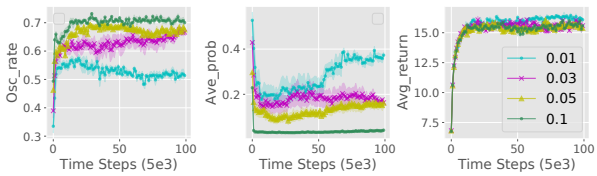


Figure 7: Training curves of our algorithm on *Two-Way* with different temperature parameter α of the mixed policy π. The results are with respect to action oscillation ratio (left), average inertia controller's outputs (middle) and returns (right). A smaller α induces better regularization (middle), lower action oscillation rate (left) as well as a higher performance (right).

expected that the policy learned is likely to be more bumpy since the solution policy space become more complex, and thus there exists a larger space for our approach to reduce the oscillation in actions.

Lower bound of Policy Inertia Controller. We also consider to impose an extra lower bound μ_0 on the PIC module μ^{pic} to further encourage smoothness of learned policies. We find in Figure 6 that a smaller μ_0 performs better regarding both oscillation rate and average return, while a large μ_0 induces too much regulation which causes substantial oscillation reduction but performance degradation as well.

Temperature Parameter of the Mixed Policy. We additionally examine the influence of the temperature parameter α of the mixed policy π. The results in Figure 7 show that a smaller α induces better regularization of μ^{pic} (which is computed as the average μ^{pic} in an episode), lower action oscillation rate as well as a higher performance. A relatively large α hampers the regularization of μ^{pic} since it encourages the stochasticity of the mixed policy.

A Close Look at Learned PIC Regularization

To better understand how the regulation on policy core π^{core} is given by a learned PIC module μ^{pic}, we visualize the execution of the vehicle (green) in *Two-Way* controlled by the mixed policy learned with NSAC, as in Figure 5. We find an oscillation occurs at timestep $t = 12$ when π^{core} tend to pick the right lane change action, and μ^{pic} outputs a high value to keep the vehicle forward. At timestep $t = 24$, when π^{core} chooses the left lane change action, μ^{pic} outputs a low value that does not regulate π^{core} to keep forward any more. At timestep $t = 30$, another improper oscillation happens when the decelerate action is encouraged by π^{core}, μ^{pic} regulates it to keep accelerating instead. This shows the PIC module has learned a good strategy to remedy improper oscillations ($t = 12$ and $t = 14$) by strongly regulate policy core to follow the previous action, while to impose few regulation when necessary change ($t = 24$) is offered by policy core.

Conclusion

In this paper, we propose a generic framework Policy Inertia Controller (PIC) to address the action oscillation issue of DRL algorithms through directly regulating the policy distribution. Moreover, we propose Nested Policy Iteration to train the PIC-augmented policies with monotonically non-decreasing updates in a general way. Our empirical results in a range of autonomous driving tasks and several Atari games show that our derived Nested Soft Actor-Critic algorithm achieves substantial action oscillation reduction without sacrificing policy performance at the same time, which is of much significance to real-world scenarios. The future work is to apply and develop our approaches in practical applications like real-world autonomous driving cars, and to investigate the extension for continuous policies.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214, U1813204), Special Program of Artificial Intelligence and Special Program of Artificial Intelligence of Tian-

jin Municipal Science and Technology Commission (No.: 56917ZXRGGX00150), Tianjin Natural Science Fund (No.: 19JCYBJC16300), Research on Data Platform Technology Based on Automotive Electronic Identification System. We would like to thank Haitham Ammar and Xueshuang Xiang for their valuable feedback on the paper and insightful discussions.

References

Carrara, N.; Leurent, E.; Laroche, R.; Urvoy, T.; Maillard, O.; and Pietquin, O. 2019. Budgeted Reinforcement Learning in Continuous State Space. In *NeurIPS*, 9295–9305.

Christodoulou, P. 2019. Soft Actor-Critic for Discrete Action Settings. *CoRR* abs/1910.07207.

Durugkar, I. P.; Rosenbaum, C.; Dernbach, S.; and Mahadevan, S. 2016. Deep Reinforcement Learning With Macro-Actions. *CoRR* abs/1606.04615.

Haarnoja, T.; Ha, S.; Zhou, A.; Tan, J.; Tucker, G.; and Levine, S. 2018a. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103* .

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018b. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*, 1856–1865.

Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; and Levine, S. 2018c. Soft Actor-Critic Algorithms and Applications. *CoRR* abs/1812.05905.

Hafner, D.; Lillicrap, T. P.; Ba, J.; and Norouzi, M. 2020. Dream to Control: Learning Behaviors by Latent Imagination. In *ICLR*.

Kendall, A.; Hawke, J.; Janz, D.; Mazur, P.; Reda, D.; Allen, J.-M.; Lam, V.-D.; Bewley, A.; and Shah, A. 2019. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, 8248–8254. IEEE.

Korenkevych, D.; Mahmood, A. R.; Vasan, G.; and Bergstra, J. 2019. Autoregressive policies for continuous control deep reinforcement learning. *arXiv preprint arXiv:1903.11524* .

Lakshminarayanan, A. S.; Sharma, S.; and Ravindran, B. 2016. Dynamic Frame skip Deep Q Network. *CoRR* abs/1605.05365.

Leurent, E.; and Maillard, O. 2019. Practical Open-Loop Optimistic Planning. In *ECML PKDD*, 69–85.

Leurent, E.; and Mercat, J. 2019. Social Attention for Autonomous Decision-Making in Dense Traffic. *CoRR* abs/1911.12250.

Li, M.; Wu, L.; Wang, J.; and Bou-Ammar, H. 2019. Multi-View Reinforcement Learning. In *NeurIPS*, 1418–1429.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. In *ICLR*.

Metelli, A. M.; Mazzolini, F.; Bisi, L.; Sabbioni, L.; and Restelli, M. 2020. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning*, 6862–6873. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533.

Popova, M.; Shvets, M.; Oliva, J.; and Isayev, O. 2019. MolecularRNN: Generating realistic molecular graphs with optimized properties. *CoRR* abs/1905.13372.

Schreck, J. S.; Coley, C. W.; and Bishop, K. J. 2019. Learning retrosynthetic planning through simulated experience. *ACS central science* 5(6): 970–981.

Sharma, S.; Lakshminarayanan, A. S.; and Ravindran, B. 2017. Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning. In *ICLR*.

Shen, Q.; Li, Y.; Jiang, H.; Wang, Z.; and Zhao, T. 2020. Deep Reinforcement Learning with Smooth Policy. *CoRR* abs/2003.09534.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587): 484–489.

Smith, L.; Dhawan, N.; Zhang, M.; Abbeel, P.; and Levine, S. 2019. AVID: Learning Multi-Stage Tasks via Pixel-Level Translation of Human Videos. *CoRR* abs/1912.04443.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

v. Hasselt, H. 2010. Double Q-learning. In Lafferty, J. D.; Williams, C. K. I.; Shawe-Taylor, J.; Zemel, R. S.; and Culotta, A., eds., *NeurIPS*, 2613–2621.

You, J.; Liu, B.; Ying, Z.; Pande, V. S.; and Leskovec, J. 2018. Graph Convolutional Policy Network for Goal-Directed Molecular Graph Generation. In *NeurIPS 2018*, 6412–6422.