Addressing Action Oscillations through Learning Policy Inertia

Chen Chen\textsuperscript{1*,} Hongyao Tang\textsuperscript{2,1*}, Jianye Hao\textsuperscript{1,2†}, Wulong Liu\textsuperscript{1}, Zhaopeng Meng\textsuperscript{2}

\textsuperscript{1}Noah’s Ark Lab, Huawei  \\
\textsuperscript{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.

1 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 2011; Raffin and Sokolkov 2019; 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...
In the figure, we can see 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.

Figure 1: Examples of action oscillation of DRL policies. (a) 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.
where, then following a policy \( \pi \) 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 \( V^\pi \) denotes value under a certain state \( s \), i.e.,
\[
V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{T} \gamma^t r(s_t, a_t) | s_0 = s \right].
\]

2.2 Soft Actor-Critic

Soft Actor-Critic (SAC) \cite{Haarnoja2018} is an off-policy reinforcement learning algorithm that optimizes a stochastic policy that maximizes the maximum entropy objective:
\[
J_{\text{Ent}}(\pi) = \mathbb{E}_{s_t, a_t \sim \rho^\pi} \left[ \sum_{t=0}^{T} \gamma^t (r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))) \right] \tag{2}
\]
where the temperature parameter \( \alpha \) determines the relative importance of the reward versus the policy entropy term at state \( s_t, \mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{s_t, a_t \sim \rho} \log \pi(a_t|s_t). \)

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 \( \pi \) through repeatedly applying the modified Bellman backup operator \( T^\pi \) as:
\[
T^\pi Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \rho} [V(s_{t+1})], \tag{3}
\]
where \( V(s_t) = \mathbb{E}_{s_t, a_t \sim \rho} [Q(s_t, a_t) - \alpha \log(\pi(a_t|s_t))] \), 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:
\[
\pi_{\text{new}} = \arg \min_{\pi \in \Pi} D_{\text{KL}} \left( \pi(\cdot|s_t) \| \exp \left( \frac{1}{\alpha} Q^{\pi_{\text{old}}}(s_t, \cdot) \right) \right). \tag{4}
\]
where \( \Pi \) 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 \( Q^\pi(s_t, a_t) \) is approximated with parameters \( \theta \) via minimizing the soft Bellman residual according to \cite{Haarnoja2018}. The policy \( \pi_\theta(a_t|s_t) \) that parameterized with \( \theta \) 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 \( \pi \) 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 \cite{Hasselt2010}. The temperature parameter \( \alpha \) can also be learned to automatically adapt through introducing a target entropy hyperparameter and optimizing the dual problem \cite{Haarnoja2018}.

The discrete action version of SAC is derived in \cite{Christodoulou2019}, 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.

3 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.

3.1 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 \( \pi \) formally. We define action oscillation ratio \( \xi(\pi) \) below:
\[
\xi(\pi) = \mathbb{E}_{s \sim \rho^\pi} \left[ \frac{1}{T} \sum_{t=1}^{T} (1 - \mathbb{I}_{A(\pi_t)}(a_t)) \right], \tag{5}
\]
where \( \rho^\pi \) is the distribution of state-action trajectory induced by policy \( \pi \) and \( \mathbb{I}_A(x) \) denotes the indicator function with set \( A \). Intuitively, \( \xi(\pi) \) indicates how smooth the actions selected by policy \( \pi \) are when acting in the environment. The lower of \( \xi(\pi) \) means the smoother of \( \pi \). 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 hyperparameters 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:
\[
\pi_\theta(\cdot|s_t, a_{t-1}) = \mu_{\text{pic}}(s_t, a_{t-1}) \delta(a_{t-1}) + (1 - \mu_{\text{pic}}(s_t, a_{t-1})) \pi_{\text{core}}(\cdot|s_t), \tag{6}
\]
where \( \delta(a_{t-1}) : A \rightarrow \mathbb{R}^{|A|} \) denotes a discrete Dirac function that puts all probability mass on the action executed at timestep \( t-1 \), and \( \pi_{\text{core}} \) denotes the policy...
Algorithm 1 Nested Policy Iteration (NPI) for PIC-augmented Policy

Input: Policy evaluation and policy improvement processes \( E_{in}, I_{in} \) for mixed policy \( \pi_{mixed} \), and \( E_{out}, I_{out} \) for mixed policy \( \pi \) (Equation 6).

1: Initialize policy core \( \pi_{core} \) and its corresponding mixed policy \( \pi \)
2: for Outer policy iteration number \( t_{out} \) do
3: for Inner policy iteration number \( t_{in} \) do
4: Evaluate the values of \( \pi_{core} \) until convergence with \( E_{in} \)
5: Improve \( \pi_{core} \) according to \( I_{in} \)
6: end for
7: Evaluate the values of \( \pi \) until convergence with \( E_{out} \)
8: Improve \( \pi \) (i.e., update \( \mu_{pic} \)) according to \( I_{out} \) while keep \( \pi_{core} \) fixed
9: end for

core that is trained as usual. The policy inertia controller \( \mu_{pic}(s_1, a_{t-1}) : S \times A \rightarrow \mathbb{R} \) outputs a scalar weight and the final policy \( \pi(s_1, a_{t-1}) \) is a linear combination of the Dirac and the policy core. We also call the PIC-augmented policy \( \pi(s_1, a_{t-1}) \) as mixed policy in the following of this paper for distinction to policy core. In another view, \( \mu_{pic}(s_1, a_{t-1}) \) 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 3.1) as below:

Theorem 3.1 Given any policy core \( \pi_{core}(s) \), there exists some \( \mu_{pic} \) such that \( \xi(\pi) \leq \xi(\pi_{core}) \) and \( J(\pi) \geq J(\pi_{core}) \), where \( \pi \) is the corresponding mixed policy counterpart (Equation 6).

Detailed proof is provided in Supplementary Material A.1. Theorem 3.1 implies that we can obtain a better policy in terms of both action oscillation rate and expected return through optimizing \( \mu_{pic}(s_1, a_{t-1}) \).

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

3.2 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 \( \pi_{core} \) is trained as usual according to inner policy iteration. The outer policy iteration is in the scope of the mixed policy \( \pi \) yet only the PIC module \( \mu_{pic} \) 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 \( \pi \).

First, we show that an improvement for policy core \( \pi_{core} \) during the inner policy iteration can induce an improvement for the mixed policy \( \pi \) as well (Lemma 3.1). We use \( Q_{core}(s_t, a_t), Q_{new}(s_t, a_{t-1}, a_t) \) to denote the \( Q \)-functions of \( \pi_{core} \) and \( \pi \) 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 \( \pi \) when \( \pi_{core} \) has been updated in the inner iteration yet \( \mu_{pic} \) has not, i.e., \( \pi_{mid}(s_t, a_{t-1}) = \mu_{pic}(s_t, a_{t-1}) + \pi_{core}(s_t) \). Now we formalize above result in the following lemma.

Lemma 3.1 (Intermediate Policy Improvement). Given an inner policy iteration with policy improvement (i.e., \( \pi_{core} \rightarrow \pi_{core}^{old} \) or \( \pi_{core} \rightarrow \pi_{core}^{mid} \)) that ensures \( Q_{core}^{old}(s_t, a_t) \geq Q_{core}^{old}(s_t, a_t) \) for all \( (s_t, a_t) \), and assume \( \mu_{pic}^{old}(s_t, a_{t-1}) \) satisfies the following inequality,

\[
\mu_{pic}^{old}(s_t, a_{t-1}) \leq \min_{(s_t, a_t)} \left( Q_{core}^{old}(s_t, a_t) - Q_{core}^{old}(s_t, a_t) \right)
\]

for all \( (s_t, a_t, a_{t-1}) \), where \( N \geq 4 \) and \( C_0 \) is the upper bound of both \( A_{new}^{core} \) and \( A_{mid}^{core} \), then we have

\[
Q_{mid}(s_t, a_{t-1}, a_t) - Q_{old}(s_t, a_{t-1}, a_t) \\ \geq (1 - \frac{4}{N}) \min_{(s_t, a_t)} \left( Q_{new}^{old}(s_t, a_t) - Q_{old}^{core}(s_t, a_t) \right)
\]

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

Detailed proof is provided in Supplementary Material A.2.

Remark 3.1 Lemma 3.1 implies that the improvement of policy core by inner policy iteration can also bring an improvement to the mixed policy given that \( \mu_{pic}^{old} \) is appropriately small. Besides, the upper bound of tolerated \( \mu_{pic}^{old} \) to guarantee such improvement and the increment from \( Q_{old}^{old} \) to \( Q_{mid}^{old} \) are both linearly dependent on the increment of policy core, which means that bigger increment in the inner iteration can tolerate bigger policy inertia \( \mu_{pic}^{old} \) 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 3.2) and then monotonically non-decreasing updates for the mixed policy \( \pi \) during overall NPI process (Theorem 3.2).

Lemma 3.2 (Nested Policy Improvement). Based on Lemma 3.1 given an outer policy iteration with policy improvement (i.e., \( \mu_{pic}^{old} \rightarrow \mu_{pic}^{new} \) or \( \pi_{mid} \rightarrow \pi_{new} \)) that ensures \( Q_{new}^{old}(s_t, a_{t-1}, a_t) \leq Q_{mid}^{old}(s_t, a_{t-1}, a_t) \) for all \( (s_t, a_{t-1}, a_t) \) \( \in S \times A \times A \), then we have \( Q_{new}(s_t, a_{t-1}, a_t) \geq Q_{old}(s_t, a_{t-1}, a_t) \).

Proof for Lemma 3.2 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 3.2 (Nested Policy Iteration). By repeatedly applying Lemma 3.2 in an iterative fashion under mild conditions.

Proof for Theorem 3.2 is a straightforward extension of Lemma 3.2 in an iterative fashion under mild conditions. According to Generalized Policy Iteration (GPI) (Sutton and Barto 1988), 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 $\pi^{\text{bic}}$ and $\pi^{\text{core}}$ 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 3.2 The outer policy iteration of NPI algorithm is in the scope of the mixed policy $\pi$ that consists of $\mu^{\text{bic}}$ and $\pi^{\text{core}}$. Since the PIC module $\mu^{\text{bic}}(s_t, a_{t-1})$ can also be viewed as a continuous policy, one may conduct the outer policy iteration (i.e., an RL algorithm) in the scope of $\mu^{\text{bic}}$ solely (instead of the mixed policy $\pi$). However, such approach can be flawed in two aspects: first, the update signal can be very weak and stochastic since $\mu^{\text{bic}}$ 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 $\mu^{\text{pic}}$.

3.3 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 $\mu^{\text{pic}}$ and the policy core $\pi^{\text{core}}$ 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 $\pi^{\text{core}}(\cdot|s_t)$ with parameter $\phi$ and update parameters $\phi$ and $\theta$ 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 $\mu^{\text{pic}}(s_t, a_{t-1})$ with parameter $\varphi$, thus obtain parameterized mixed policy $\pi^{\text{phi}}$. The value networks of $\pi^{\text{phi}}$ is approximated during soft policy evaluation (Equation (3)) and only parameter $\varphi$ 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 $\pi^{\text{phi}}$.

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

4 Experiments

This section presents the experimental results of our approach. We first provide the setups and the benchmark approaches in Section 4.1, and then followed by evaluation results and an analysis study in Section 4.2 and Section 4.3. Finally, we analyze the learned regularization of smoothness for further insights of PIC frameworks in Section 4.4.

4.1 Setups

Environments and Benchmark Approaches. We use the Highway simulator 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 highway environments are originally provided at https://github.com/leurent/highway-env.
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 their variants with dynamic action repetition and reward shaping tricks (see Section 4.2). 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 \(\frac{\sum_{i=1}^{20} c_i/n_i}{20}\). The solid curves correspond to the mean and the shaded region to half a standard deviation over the five trials.

**4.2 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 5. 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 3.1) of mixed policy and the effectiveness of NPI (Theorem 3.2).

**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.

**4.3 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 \(\mu^{\text{PIC}}\) and the effect of temperature parameter \(\alpha\) of the mixed policy \(\pi\).

**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(c)-4(f) indicates that the oscillation reduction is more significant in complex cases with a large margin. This is as 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 $\mu_0$ on the PIC module $\mu_{\text{pic}}$ to further encourage smoothness of learned policies. We find in Figure 6 that a smaller $\mu_0$ performs better regarding both oscillation rate and average return, while a large $\mu_0$ 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 $\alpha$ of the mixed policy $\pi$. The results in Figure 7 show that a smaller $\alpha$ induces better regularization of $\mu_{\text{pic}}$ (which is computed as the average $\mu_{\text{pic}}$ in an episode), lower action oscillation rate as well as a higher performance. A relatively large $\alpha$ hampers the regularization of $\mu_{\text{pic}}$ since it encourages the stochasticity of the mixed policy.

### 4.4 A Close Look at Learned PIC Regularization

To better understand how the regulation on policy core $\pi_{\text{core}}$ is given by a learned PIC module $\mu_{\text{pic}}$, 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 $\pi_{\text{core}}$ tends to pick the right lane change action, and $\mu_{\text{pic}}$ outputs a high value to keep the vehicle forward. At timestep $t = 24$, when $\pi_{\text{core}}$ chooses the left lane change action, $\mu_{\text{pic}}$ outputs a low value that does not regulate $\pi_{\text{core}}$ to keep forward any more. At timestep $t = 30$, another improper oscillation happens when the deceleration action is encouraged by $\pi_{\text{core}}$, $\mu_{\text{pic}}$ 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.

### 5 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 Tianjin Municipal Science and Technology Commission (No.: 56917ZXRGBGX00150), Tianjin Natural Science Fund (No.: 19JYBJC16300), Research on Data Platform Technology Based on Electronic Identification System.

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

Christodoulopou, 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.; Mazzolani, F.; Bisi, L.; Sabbioni, L.; and Restelli, M. 2020. Control Frequency Adaptation via Action Persistence in Batch Reinforcement Learning. arXiv preprint arXiv:2002.06836.

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.

Raffin, A.; and Sokolov, R. 2019. Learning to Drive Smoothly in Minutes. https://github.com/araffin/learning-to-drive-in-5-minutes.

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

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M. I.; and Moritz, P. 2015. Trust Region Policy Optimization. In ICML, volume 37, 1889–1897.

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. 1988. Reinforcement Learning: An Introduction. IEEE Transactions on Neural Networks 16: 285–286.

Sutton, R. S.; and Barto, A. G. 2011. Reinforcement learning: An introduction.

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.
A.1 Proof of Theorem 3.1

Proof: We define an auxiliary reward function \( r_o(s_t, a_{t-1}, a_t) = r_o(a_t, a_{t-1}) = \mathbb{I}_{a_{t-1}}(a_t) \) and a default action \( a_{-1} \) as the previous action of \( a_0 \) specially to make the formulation work. We also define \( V^{\pi}_o(s) \) as the value function of the state \( s \) under the reward function \( r_o(s_t, a_{t-1}, a_t) \) for given \( \pi \). Therefore, to prove \( \xi(\pi) \leq \xi(\pi^{\text{core}}) \) is equivalent to prove \( \mathbb{E}_{s_0 \sim \rho_0} \left[ \frac{1}{2} V^{\pi}_o(s_0) \right] \geq \mathbb{E}_{s_0 \sim \rho_0} \left[ \frac{1}{2} V^{\pi^{\text{core}}}_o(s_0) \right] \), thus to prove \( V^{\pi}_o(s_0) \geq V^{\pi^{\text{core}}}_o(s_0) \) for all possible initial states \( s_0 \) given that \( \gamma = 1 \).

For \( s_0 \) and arbitrary \( \mu(s_0, a_{-1}) \geq 0 \) with its corresponding \( \pi \), we have

\[
\mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} Q^{\pi^{\text{core}}}_o(s_0, a_{-1}, a_0) \\
= \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} \left[ r_o(s_0, a_{-1}, a_0) + E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) \right] \\
= \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} r_o(s_0, a_{-1}, a_0) + \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) \\
= \left( 1 - \mu(s_0, a_{-1}) \right) \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} r_o(s_0, a_{-1}, a_0) + \mu(s_0, a_{-1}) r_o(s_0, a_{-1}) \\
+ \left( 1 - \mu(s_0, a_{-1}) \right) \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) + \mu(s_0, a_{-1}) \mathbb{E}_{s_1 \sim p(\cdot|s_0, a_{-1})} V^{\pi^{\text{core}}}_o(s_1) \\
= A + B + \mu(s_0, a_{-1})(1 - A + C - B) = V^{\pi^{\text{core}}}_o(s_0) + \mu(s_0, a_{-1})(1 - A + C - B)
\]

where the last equality comes from the definition of \( r_o \) and \( 1 - A \geq 0 \) always holds.

We also have the similar relation on the original reward \( r(s_t, a_t) \) as follows:

\[
\mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} Q^{\pi^{\text{core}}}_o(s_0, a_0) \\
= \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} \left[ r(s_0, a_0) + E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) \right] \\
= \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} r(s_0, a_0) + \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) \\
= \left( 1 - \mu(s_0, a_{-1}) \right) \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} r(s_0, a_0) + \mu(s_0, a_{-1}) r(s_0, a_{-1}) \\
+ \left( 1 - \mu(s_0, a_{-1}) \right) \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) + \mu(s_0, a_{-1}) \mathbb{E}_{s_1 \sim p(\cdot|s_0, a_{-1})} V^{\pi^{\text{core}}}_o(s_1) \\
= A' + B' + \mu(s_0, a_{-1})(r(s_0, a_{-1}) - A' + C' - B') = V^{\pi^{\text{core}}}_o(s_0) + \mu(s_0, a_{-1})(r(s_0, a_{-1}) - A' + C' - B')
\]

Hence, we can adopt the following rule to pick \( \mu(s_0, a_{-1}) \):

\[
\mu(s_0, a_{-1}) \begin{cases} > 0, \text{ if } (1 - A) + (C - B) > 0 \text{ and } r(a_0, a_{-1}) - A' + C' - B' > 0 \\ = 0, \text{ else.} \end{cases}
\]

By this way, we can ensure that \( \square \) is not less than \( V^{\pi^{\text{core}}}_o(s_0) \) and \( \square \) is not less than \( V^{\pi^{\text{core}}}_o(s_0) \) as well. And this implies that

\[
V^{\pi^{\text{core}}}_o(s_0) \leq \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} Q^{\pi^{\text{core}}}_o(s_0, a_{-1}, a_0)
\]

and

\[
V^{\pi^{\text{core}}}_o(s_0) \leq \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} Q^{\pi^{\text{core}}}_o(s_0, a_0).
\]

Repeatedly expand \( V^{\pi^{\text{core}}}_o(s_0) \) on the RHS by applying the inequality \( \square \) and follow the rule of choosing \( \mu(s_t, a_{t-1}) \) at each state, we have

\[
V^{\pi^{\text{core}}}_o(s_0) \leq \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} Q^{\pi^{\text{core}}}_o(s_0, a_{-1}, a_0) \\
= \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} \left[ r_o(s_0, a_{-1}, a_0) + E_{s_1 \sim p(\cdot|s_0, a_0)} V^{\pi^{\text{core}}}_o(s_1) \right] \\
\leq \mathbb{E}_{a_0 \sim \pi(\cdot|s_0)} \left[ r_o(s_0, a_{-1}, a_0) + E_{s_1 \sim p(\cdot|s_0, a_{-1}, a_0)} \left( E_{a_1 \sim \pi(\cdot|s_1)} r(s_1, a_0, a_1) + E_{a_1 \sim \pi(\cdot|s_1)} E_{s_2 \sim p(\cdot|s_1, a_1)} V^{\pi^{\text{core}}}_o(s_2) \right) \right] \\
\leq \cdots \\
\leq V^{\pi}_o(s_0).
\]
Similarly, we can repeatedly expand $V^\pi_{core}$ on the RHS by applying the inequality \[[11]\] and obtain
\[ V^\pi_{core}(s_0) \leq V^\pi(s_0). \] \[[13]\]

**A.2 Proof of Lemma 3.1**

Two lemmas are presented firstly.

**Lemma A.1** \cite{Schulmanetal2015} For arbitrary $\theta$ and $\theta'$, the following equality holds:
\[ J(\theta') - J(\theta) = E_{\tau \sim p_{\theta'}(\tau)} \left[ \mathbb{E}^{A^\theta}(s_t, a_t) \right]. \] \[[14]\]

The following lemma can be derived from \cite{Schulmanetal2015}.

**Lemma A.2** Assume $\pi(s_t)$ is an arbitrary distribution, then if $|\pi'(a_t|s_t) - \pi(a_t|s_t)| < \epsilon$ for all $s_t, a_t$, we have $\pi(s_t)$ is close enough to $\pi(s_t)$, and the following results hold:
\[ |p_\pi(s_t) - p_{\pi}(s_t)| \leq 2\epsilon t, \]
\[ E_{p_{\pi}(s_t)}[f(s_t)] \geq E_{p_{\pi}(s_t)}[f(s_t)] - 2\epsilon t \max |f(s_t)|, \] where $p_\pi$ denotes the state marginal of the trajectory distribution induced by policy $\pi$ (in a specific MDP).

**Proof of Lemma 3.1** Firstly, we rewrite $Q_{\pi_{mid}}(s_t, a_{t-1}, a_t) - Q_{\pi_{old}}(s_t, a_{t-1}, a_t)$ into three parts as follows:
\[ Q_{\pi_{mid}}(s_t, a_{t-1}, a_t) = (Q_{\pi_{mid}}(s_t, a_{t-1}, a_t) - Q_{\pi_{core}}(s_t, a_{t-1}, a_t)) + (Q_{\pi_{core}}(s_t, a_{t-1}, a_t) - Q_{\pi_{old}}(s_t, a_{t-1}, a_t)). \] \[[17]\]

We expand the first term in \[[17]\] as follows:
\[ Q_{\pi_{mid}}(s_t, a_{t-1}, a_t) = Q_{\pi_{new}}(s_t, a_{t-1}, a_t) - Q_{\pi_{old}}(s_t, a_{t-1}, a_t) \]
\[ = E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[V^\pi_{mid}(s_{t+1}, a_t)] - E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[V^\pi_{new}(s_{t+1}, a_t)] \]
\[ = E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[\gamma^t A^\pi_{new}(s_t, a_{t-1}, a_t)]. \] \[[18]\]

where the last equality comes from Lemma A.1. Since the difference of action probability between $\pi_{mid}(s_t, a_{t-1})$ and $\pi_{core}(s_t, a_{t-1})$ is bounded by $\mu_{\pi_{mid}}(s_t, a_{t-1}) - \mu_{\pi_{core}}(s_t, a_{t-1})$ is also uniformly upper bounded by $\epsilon := N\min_{t} \sum_t \mathcal{C}_0(t^\gamma)$ from the conditions in Lemma 3.1, then directly applying \[[16]\] in Lemma A.2 we obtain
\[ E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[\gamma^t A^\pi_{new}(s_t, a_{t-1}, a_t)] - \frac{2\epsilon C_0}{N} \gamma^t \]
\[ = -\frac{2\epsilon}{N} \min_{s_t, a_t} (Q^\pi_{new}(s_t, a_t) - Q^\pi_{old}(s_t, a_t)). \] \[[19]\]

where $C_0$ is the upper bound of both $A^\pi_{new}$ and $A^\pi_{old}$, the equality holds because of the fact that $\forall \pi, E_{\tau \sim \pi}[\sum_t \gamma^t A^\pi(s_t, a_t)] = 0$.

For the third term we can similarly derive that
\[ Q^\pi_{old}(s_t, a_{t-1}, a_t) = E_{s_t \sim \pi_{old}}[\sum_t \gamma^t A^\pi_{old}(s_t, a_t)]. \]
\[ \geq -\frac{2\epsilon}{N} \min_{s_t, a_t} (Q^\pi_{new}(s_t, a_t) - Q^\pi_{old}(s_t, a_t)). \] \[[20]\]

Combining \[[18]\] and \[[20]\] with \[[17]\], we obtain
\[ Q_{\pi_{mid}}(s_t, a_{t-1}, a_t) - Q_{\pi_{old}}(s_t, a_{t-1}, a_t) \]
\[ \geq (1 - \frac{4\epsilon}{N}) \min_{s_t, a_t} (Q^\pi_{new}(s_t, a_t) - Q^\pi_{old}(s_t, a_t)), \] \[[21]\]

which yields our proof.
B Overall Algorithm of Nested Soft Actor-Critic (NSAC)

We propose Nested Soft Actor-Critic (NSAC) to train the PIC-augmented policy (i.e., the mixed policy \( \pi \)), consisting of the PIC module \( \mu_{\text{pic}} \) and the policy core \( \pi_{\text{core}} \) simultaneously. NSAC is a practical implementation of Nested Policy Iteration (NPI) we proposed in Section 3.2. For both the inner policy iteration and the outer policy iteration, we use SAC algorithm.

For the inner policy iteration (i.e., to train the policy core \( \pi_{\text{core}}(s_t, a_t) \)), we parameterize the policy core and its \( Q \)-function as \( \pi_{\phi}(s_t, a_t) \) and \( Q_{\theta}(s_t, a_t) \) with parameters \( \phi \) and \( \theta \) respectively. We also set the temperature parameter for training \( \pi_{\text{core}}(s_t, a_t) \) as a hyperparameter \( \alpha_{\text{core}} \). Then \( Q_{\theta}(s_t, a_t) \) is trained to minimize the soft Bellman residual according to Equation (3):

\[
L_{\text{core}}(\theta) = E_{(s_t, a_t) \sim D} \left[ \frac{1}{2} (Q_{\theta}(s_t, a_t) - (r(s_t, a_t) + \gamma E_{s_{t+1} \sim P} [V_{\theta}(s_{t+1})]))^2 \right],
\]

where \( D \) is a replay buffer of past experiences collected using the mixed policy \( \pi \), and \( V_{\theta}(s_{t+1}) \) is estimated using a target network with exponentially moving parameters \( \theta \) for the online network \( Q_{\theta}(s_t, a_t) \). Since the action space is discrete, we can calculate \( V_{\theta}(s_{t+1}) \) directly as:

\[
V_{\theta}(s_{t+1}) = \pi_{\phi}(s_{t+1})^T [Q_{\theta}(s_{t+1}) - \alpha_{\text{core}} \log (\pi_{\phi}(s_{t+1}))]\]

For the improvement of \( \pi_{\text{core}} \), we minimize the expected KL-divergence (Equation (4)) after multiplying by the temperature parameter \( \alpha_{\text{core}} \) and ignoring the partition function, again utilizing the property of discrete policy, we have

\[
L_{\text{core}}(\phi) = E_{s_t \sim D} [\pi_{\phi}(s_t) \alpha_{\text{core}} \log (\pi_{\phi}(s_t)) - Q_{\theta}(s_t)],
\]

The training of \( \pi_{\text{core}} \) and \( Q_{\theta} \) described above is the usual form of discrete SAC (Christodoulou 2019). We recommend the readers who are not familiar with discrete SAC to refer to the original paper for more details.

For the outer policy iteration (i.e., to train the PIC module \( \mu_{\text{pic}}(s_t, a_{t-1}) \)), we parameterize \( \mu_{\text{pic}}(s_t, a_{t-1}) \) with parameters \( \varphi \), thus the parameterized mixed policy is denoted as \( \pi_{\varphi}(s_t, a_{t-1}) \). We also parameterize the \( Q \)-function of \( \pi_{\varphi} \) as \( Q_{\varphi}(s_t, a_{t-1}, a_{t}) \) with parameters \( \varphi \). To train PIC module \( \mu_{\text{pic}}(s_t, a_{t-1}) \), we use SAC algorithm with the temperature parameter \( \alpha \) in the scope of the mixed policy \( \pi_{\varphi} \). Similar to Equation (22), we optimize \( \varphi \) by conducting soft policy evaluation for the mixed policy \( \pi \), i.e., minimizing the soft Bellman residual of \( Q_{\varphi} \) as:

\[
L_{\text{mix}}(\varphi) = E_{(s_{t-1}, a_{t-1}, a_t) \sim D} \left[ \frac{1}{2} (Q_{\varphi}(s_{t-1}, a_{t-1}, a_t) - r(s_{t-1}, a_{t-1}) + \gamma E_{s_{t+1} \sim P} [V_{\varphi}(s_{t+1}, a_{t-1}, a_t))]^2 \right],
\]

and \( V_{\varphi} \) is calculated as:

\[
V_{\varphi}(s_{t+1}, a_t) = \pi(s_{t+1}, a_t) \alpha_{\text{core}} Q_{\varphi}(s_{t+1}, a_t) \log (\pi(s_{t+1}, a_t)]
\]

\[
= \left( \mu_{\psi}(s_{t+1}, a_t) \alpha_{\text{core}} Q_{\varphi}(s_{t+1}, a_t) \right)^T \\
\cdot \pi_{\phi}(s_{t+1}, a_t) \alpha_{\text{core}} Q_{\varphi}(s_{t+1}, a_t) \log (\pi_{\phi}(s_{t+1}, a_t)]
\]

where \( \varphi \) is the target network parameters and note the calculation of expectation and entropy is based on the mixed policy \( \pi(s_{t+1}, a_t) \).

To update the PIC module parameters \( \varphi \), a policy improvement step of the mixed policy \( \pi_{\varphi} \) is conducted while the parameters \( \phi \) (i.e., the policy core \( \pi_{\text{core}}(s_t, a_t) \)) is kept fixed, yielding the following objective:

\[
L_{\text{pic}}(\psi) = E_{s_{t-1} \sim D} \left[ \alpha_{\text{core}} \log (\pi(s_{t-1}, a_{t-1})) - Q_{\varphi}(s_{t-1}, a_{t-1}) \right]
\]

\[
= E_{s_{t-1} \sim D} \left[ (\mu_{\psi}(s_{t-1}, a_{t-1}) \alpha_{\text{core}} Q_{\varphi}(s_{t-1}, a_{t-1}))^T \\
\alpha_{\text{core}} \log (\pi_{\phi}(s_{t-1}, a_{t-1})) + (1 - \pi_{\phi}(s_{t-1}, a_{t-1})) \pi_{\phi}(s_{t-1}) - Q_{\varphi}(s_{t-1}, a_{t-1}) \right]
\]

Based on above formulations, Algorithm 2 gives the overall algorithm of NSAC.

C Experimental Details

C.1 Environment Setup

We conduct our experiments on Highway autonomous driving simulators, which is provided at [https://github.com/eleurent/highway-env](https://github.com/eleurent/highway-env). A few modifications are made to make the rendering of the environments faster. For the detailed configurations for the selected four environments, please see Section C.2. Besides, we also adopt several representative tasks in OpenAI Atari games, i.e., MsPacman, SpaceInvaders, Qbert, JamesBond. More details can be found at [https://gym.openai.com/envs/#atari](https://gym.openai.com/envs/#atari).
The critic in all algorithms.

Table 4 shows the common hyperparameters of algorithms used in our experiments. No regularization is used for the actor and the critic in all algorithms.

### C.2 Configurations of Highway Tasks

We follow the built-in configurations in Highway as originated presented at [https://github.com/eleurent/highway-env](https://github.com/eleurent/highway-env) and add some extra configurations in Table [1].

### C.3 Implementation Details

Our codes are implemented with Python 3.6 and Tensorflow. For discrete Soft Actor-Critic (SAC) algorithm, we use a modified version of the implementation at [https://github.com/ku2482/sac-discrete.pytorch](https://github.com/ku2482/sac-discrete.pytorch) to suit our environments, and based on which we implement our algorithm Nested Soft Actor-Critic (NSAC). For Deep Q-Network (DQN), we use a standard implementation of the vanilla version, i.e., with only experience replay and target network. For the action repetition approach, 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 × Re. DQN-repeat baseline and SAC-repeat baseline mean that DQN and SAC are trained on the augmented action space A', respectively. The hyperparameter settings for DQN-repeat baseline and SAC-repeat baseline are the same as DQN and SAC. Our codes will be released on Github soon.

**Network Structures.** The network structures of DQN/SAC/NSAC used for Highway autonomous driving environments and OpenAI Atari games are shown in Table [2] and Table [3] respectively. We use a two-layer fully-connected neural network in Highway and an additional convolutional neural network is adopted in Atari games for pixel inputs.

**Training Details.** For all approaches, we update the parameters every 2 transition samples are collected in Highway autonomous driving environments, and every 4 transition samples for Atari games. For DQN and its variants, ε-greedy is adopted for exploration. We gradually schedule ε from 1 to 0.1 through decaying it with a decrease of 5 · 10⁻⁶ for each update for both Highway tasks and Atari games. For NSAC, the policy core ρcore (inner policy iteration) and the PIC module ρpic (outer policy iteration) is updated at a 1 : 1 frequency. Moreover, we do not use the code-level tricks, such as state normalization, reward normalization, gradient clip and etc.

### C.4 Hyperparameters

Table [4] shows the common hyperparameters of algorithms used in our experiments. No regularization is used for the actor and the critic in all algorithms.

---

**Algorithm 2 Nested Soft Actor-Critic algorithm (NSAC)**

**Input:** Learning rates λθ, λ ϑ, λϕ for corresponding parameters, target network update rates σcore, σ and train intervals mcore, m

1. Initialize ρϕ, Qθ1, Qθ2, Qθ1, Qθ2
2. Initialize µϕ
3. Initialize Qθ1, Qθ2, Qθ1, Qθ2
4. Initialize D ← ∅
5. Set θ₁ ← θ₁, θ₂ ← θ₂, 〈θ₁, 〈θ₂)
6. for each environment step t from 0 to T do
7. a₁ ← π(s₁, a₁−1) # Sample action from the mixed policy π. Specially, a₁−1 is set to be a default/null action
8. s₁+1 ← P(s₁, a₁) # Sample transition from the environment
9. D ← D ∪ {s₁, a₁−1, a₁, r(s₁, a₁), s₁+1} # Store the transition in the replay buffer
10. if step t mod mcore = 0 then
11. θ₁ ← θ₁ − λθ∇θLcore(θ₁) for i, 1, 2 # Update the parameters of Qcore
12. ϕ ← ϕ − λϕ∇ϕLcore(ϕ) # Update the parameters of πcore
13. ϑ₁ ← σcoreϑ₁ + (1 − σcore)ϑ₁ for i, 1, 2 # Update the parameters of target Qcore
14. end if
15. if step t mod m = 0 then
16. ϑ₁ ← ϑ₁ − λϕ∇ϕLmix(ϑ₁) for i, 1, 2 # Update the parameters of Q
17. ϕ ← ϕ − λϕ∇ϕLmix(ϕ) # Update the parameters of µϕ(s₁, a₁−1)
18. ϑ₁ ← σϑ₁ + (1 − σ)ϑ₁ for i, 1, 2 # Update the parameters of target Q
19. end if
20. end for
21. Output θ₁, θ₂, ϕ, ϑ₁, ϑ₂, ψ # Output the optimized parameters
Table 1: Detail configurations of four Highway tasks. \([a, b]\) denotes a Uniform distribution on the range from integer \(a\) to integer \(b\).

| Configuration Items                              | Lane Change | Merge | Intersection | Two-Way |
|--------------------------------------------------|-------------|-------|--------------|---------|
| Vehicles Count for Training                      | [20, 50]    | [6, 12]| 10           | 10      |
| Vehicles Count for Testing                       | 45          | [6, 12]| 10           | 10      |
| Vehicles Count in Ego Vehicle’s Scope            | 10          | 10    | 5            | 10      |
| COLLISION REWARD                                 | -1          | -1    | 0            | 0       |
| RIGHT LANE REWARD                                | 0           | 0.1   | 0            | 0       |
| HIGH VELOCITY REWARD                             | 0.4         | 0.2   | 0            | 0.8     |
| LANE CHANGE REWARD                               | -0.1        | -0.05 | 0            | 0       |
| MERGING VELOCITY REWARD                          | 0           | -0.5  | 0            | 0       |
| ARRIVED REWARD                                   | 0           | 0     | 5            | 0       |
| LEFT LANE CONSTRAINT                             | 0           | 0     | 0            | 1       |
| LEFT LANE REWARD                                 | 0           | 0     | 0            | 0.2     |
| Duration                                         | 70s         | 25s   | 25s          | 25s     |

Table 2: Network structures of approaches in Highway autonomous driving environments.

| Layer                              | Actor Network \((\pi^{\text{core}}(s_t))\) | Critic or Q Network \((Q^{\text{core}}(s_t, a_t) \text{ and } Q(s_t, a_{t-1}, a_t))\) | PIC Network \((\mu^{\text{pic}}(s_t, a_{t-1}))\) |
|-----------------------------------|-------------------------------------------|-------------------------------------------------|-------------------------------------------------|
| Fully-connected                   | (state dim, 64) ReLU                      | (state dim, 64) or (state dim + action dim, 64) ReLU | (state dim + action dim, 64) ReLU              |
| Activation                        | (64, 64) ReLU                            | (64, 64) ReLU                                  | (64, 64) ReLU                                  |
| Fully-connected                   | (64, action dim) ReLU softmax             | (64, action dim) None                          | (64, 1) tanh                                   |

Table 3: Network structures of approaches in OpenAI Atari games.

| Network                           | Layer (Name)          | Structure                                                                 |
|-----------------------------------|-----------------------|--------------------------------------------------------------------------|
| Convolutional Network             | Convolution           | 32 channels, 8x8 kernel, 4 stride, 0 padding ReLU                         |
| Activation                        | Convolution           | 64 channels, 4x4 kernel, 2 stride, 0 padding ReLU                         |
| Activation                        | Convolution           | 64 channels, 3x3 kernel, 1 stride, 0 padding ReLU                         |
| Flatten                            | Flatten               | (into size [batch size, 7*7*64])                                       |
| Fully-connected Network            | Fully-connected       | \((7*7*64, 512)\) for \(\pi^{\text{core}}\) and \(Q^{\text{core}}\); \((7*7*64 + \text{action dim}, 512)\) for \(\mu^{\text{pic}}\) and \(Q\) ReLU |
| Activation                        | Fully-connected       | \((512, \text{action dim})\) for \(\pi^{\text{core}}\), \(Q^{\text{core}}\) and \(Q\); \((512, 1)\) for \(\mu^{\text{pic}}\) |
| Fully-connected                   | Activation            | softmax for \(\pi^{\text{core}}\), None for \(Q^{\text{core}}\) \(Q\); tanh for \(\mu^{\text{pic}}\) |
Table 4: Hyperparameter setting of algorithms adopted.

| Hyperparameter                                           | Highway | Atari |
|-----------------------------------------------------------|---------|-------|
| Learning Rate                                            | $3 \cdot 10^{-4}$ | $3 \cdot 10^{-4}$ |
| Discount Factor                                           | 0.99    | 0.99  |
| Batch Size                                                | 64      | 64    |
| Buffer Size                                               | $2 \cdot 10^5$ | $10^6$ |
| Optimizer                                                 | Adam    | Adam  |
| Policy Inertia Controller Learning Rate (for NSAC)        | $3 \cdot 10^{-4}$ | $3 \cdot 10^{-3}$ |
| Soft Target Replacement Rate (for SAC/NSAC)               | 0.002   | 0.005 |
| Hard Target Replacement Rate (for DQN)                    | $10^4$  | $10^4$ |
| Temperature Parameter (for SAC and $\pi_{\text{core}}$ in NSAC) | 0.1     | 0.1   |
| Temperature Parameter (for mixed policy $\pi$ in NSAC)    | 0.01    | 0.0001 |