Deep Reinforcement Learning with Robust and Smooth Policy

Qianli Shen [*1] **Yan Li** [*2] **Haoming Jiang** [2] **Zhaoran Wang** [3] **Tuo Zhao** [2]

Abstract

Deep reinforcement learning (RL) has achieved great empirical successes in various domains. However, the large search space of neural networks requires a large amount of data, which makes the current RL algorithms not sample efficient. Motivated by the fact that many environments with continuous state space have smooth transitions, we propose to learn a smooth policy that behaves smoothly with respect to states. We develop a new framework — **S**mooth **R**egularized **R**einforcement **L**earning (**SR**^{2}**L**), where the policy is trained with smoothness-inducing regularization. Such regularization effectively constrains the search space, and enforces smoothness in the learned policy. Moreover, our proposed framework can also improve the robustness of policy against measurement error in the state space, and can be naturally extended to distributionally robust setting. We apply the proposed framework to both on-policy (TRPO) and off-policy algorithm (DDPG). Through extensive experiments, we demonstrate that our method achieves improved sample efficiency and robustness.

1. Introduction

Deep reinforcement learning has enjoyed great empirical successes in various domains, including robotics, personalized recommendations, bidding, advertising and games (Levine et al., 2018; Zheng et al., 2018; Zhao et al., 2018; Silver et al., 2017; Jin et al., 2018). At the backbone of its success is the superior approximation power of deep neural networks, which parameterize complex policy, value or state-action value functions, etc. However, the high complexity of deep neural networks makes the search space of the learning algorithm prohibitively large, thus often requires a significant amount of training data, and suffers from numerous training difficulties such as overfitting and training instability (Thrun & Schwartz, 1993; Boyan & Moore, 1995; Zhang et al., 2016).

Reducing the size of the search space while maintaining the network's performance requires special treatment. While one can simply switch to a network of smaller size, numerous empirical evidences have shown that small network often leads to performance degradation and training difficulties. It is widely believed that training a sufficiently large network (also known as over-parameterization) with suitable regularization (e.g., dropout (Srivastava et al., 2014), orthogonality parameter constraints (Huang et al., 2018)) is the most effective way to adaptively constrain the search space, while maintaining the performance of a large network.

For reinforcement learning problems, entropy regularization is one commonly adopted regularization, which is believed to help facilitate exploration in the learning process. Yet in the presence of high uncertainty in the environment and large noise, such regularization might yield poor performances. More recently, Pinto et al. (2017) propose robust adversarial reinforcement learning (RARL) that aims to perform well under uncertainties by training the agent to be robust against the adversarially perturbed environment. However, in addition to the marginal performance gain, the requirement of learning the additional adversarial policy makes the update of RARL computationally expensive and less sample efficient than traditional learning algorithms. Cheng et al. (2019) on the other hand propose a control regularization that enforces the behavior of the deep policy to be similar to a policy prior, yet designing a good prior often requires a significant amount of domain knowledge.

Different from previous works, we propose a new training framework – **S**moothness **R**egularized **R**einforcement **L**earning (**SR**^{2}**L**) for training reinforcement algorithms. Through promoting smoothness, we effectively reduce the size of the search space when learning the policy network and achieve state-of-the-art sample efficiency. Our goal of promoting smoothness in the policy is motivated by the fact that natural environments with continuous state space often have smooth transitions from state to state, which favors a smooth policy – similar states leading to similar actions. As a concrete example, for MuJoCo environment (Todorov

[*]Equal contribution [1] Peking University, Beijing, China. [2] Georgia Institute of Technology, Atlanta, USA. [3]Northwestern University, Evanston, USA.. Correspondence to: Tuo Zhao <tourzhao@gatech.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

et al., 2012), which is a system powered by physical laws, the optimal policy can be described by a set of differential equations with certain smoothness properties.

Promoting smoothness is particularly important for deep RL, since deep neural networks can be extremely non-smooth, due to their high complexity. It is observed that small changes in neural networks' input would result in significant changes in its output (Goodfellow et al., 2014; Kurakin et al., 2016). To train a smooth neural network, we need to employ many hacks in the training process. In supervised learning setting with i.i.d. data, these hacks include but not limited to batch normalization (Ioffe & Szegedy, 2015), layer normalization (Ba et al., 2016), orthogonal regularization (Huang et al., 2018). However, most of existing hacks do not work well in RL setting, where the training data has complex dependencies. As one significant consequence, current reinforcement learning algorithms often lead to undesirable non-smooth policy.

Our proposed **SR^{2}L** training framework uses a smoothness-inducing regularization to encourage the output of the policy (decision) to not change much when injecting small perturbation to the input of the policy (observed state). The framework is motivated by local shift sensitivity in robust statistics literature (Hampel, 1974), which can also be considered as a measure of the local Lipschitz constant of the policy. We highlight that **SR^{2}L** is highly flexible and can be readily adopted into various reinforcement learning algorithms. As concrete examples, we apply **SR^{2}L** to the TRPO algorithm (Schulman et al., 2015), which is an *on-policy* method, and the regularizer directly penalizes non-smoothness of the policy. In addition, we also apply **SR^{2}L** to DDPG algorithm (Lillicrap et al., 2015), which is an *off-policy* method, and the regularizer penalizes non-smoothness of either the policy or the state-action value function (also known as the Q-function), which can be used to induce a smooth policy.

Moreover, we remark that besides promoting the smoothness of policy network, our proposed regularizer can also help improve the robustness of policy against random, or even adversarial measurement error in the state space.

Our proposed smoothness-inducing regularizer is related to several existing works (Miyato et al., 2018; Zhang et al., 2019; Hendrycks et al., 2019; Xie et al., 2019; Jiang et al., 2019). These works consider similar regularization techniques, but target at other applications with different motivations, e.g., semi-supervised learning, unsupervised domain adaptation and harnessing adversarial examples.

The rest of the paper is organized as follows: Section 2 introduces the related background; Section 3 introduces our proposed smooth regularized reinforcement learning (**SR^{2}L**) in detail; Section 4 discusses how our proposed method help improve the robustness and a natural extension to distributionally robust settings; Section 5 presents numerical experiments on various MuJoCo environments to demonstrate the superior performance of **SR^{2}L**; Section 6 draws a brief conclusion.

2. Background

We consider a Markov Decision Process $(\mathcal{S}, \mathcal{A}, \mathbb{P}, r, p_0, \gamma)$, in which an agent interacts with an environment in discrete time steps. We let $\mathcal{S} \subseteq \mathbb{R}^S$ denote the continuous state space, $\mathcal{A} \subseteq \mathbb{R}^A$ denote the action space, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denote the transition kernel, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denote the reward function, p_0 denote the initial distribution and γ denote the discount factor. An agent's behavior is defined by a policy, either stochastic or deterministic. A stochastic policy π maps a state to a probability distribution over the action space $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. A deterministic policy μ maps a state directly to an action $\mu : \mathcal{S} \to \mathcal{A}$. At each time step, the agent observes its state $s_t \in \mathcal{S}$, takes action $a_t \sim \pi(s_t)$, and receives reward $r_t = r(s_t, a_t)$. The agent then transits into the next state s_{t+1} following the transition kernel $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$. The goal of the agent is to find a policy that maximize the expected discounted reward:

$$\max_{\pi} V(\pi) = \mathbb{E}_{s_0, a_0, \ldots}\left[\sum_{t \geq 0} \gamma^t r(s_t, a_t)\right],$$

with $s_0 \sim p_0, a_t \sim \pi(s_t), s_{t+1} \sim \mathbb{P}(s_{t+1}|s_t, a_t)$. One way to solve the above problem is the classical policy gradient algorithms, which estimate the gradient of the expected reward through trajectory samples, and update the parameters of the policy by following the estimated gradient. The policy gradient algorithms suffers from high variance of estimated gradient, which often leads to aggressive updates and unstable training. To address this issue, numerous variants have been proposed. Below we briefly review two popular ones used in practice.

2.1. Trust Region Policy Optimization (TRPO)

TRPO iteratively improves a parameterized policy π_θ by solving a trust region type optimization problem. Before we describe the algorithm in detail, we need several definitions in place. The value function $V^\pi(s)$ and the state-action value function $Q^\pi(s, a)$ are defined by:

$$V^\pi(s) = \mathbb{E}_{s_0 = s, a_0, \ldots}\left[\sum_{t \geq 0} \gamma^t r(s_t, a_t)\right],$$

$$Q^\pi(s, a) = \mathbb{E}_{s_1, a_1, \ldots}\left[\sum_{t \geq 0} \gamma^t r(s_t, a_t)|s_0 = s, a_0 = a\right],$$

with $a_t \sim \pi(s_t), s_{t+1} \sim \mathbb{P}(s_{t+1}|s_t, a_t)$. The advantage function $A^\pi(s, a)$ and the discounted state visitation distri-

bution (unnormalized) $\rho^\pi(s)$ are defined by:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s), \rho^\pi(s) = \sum_{i\geq 0} \gamma^i \mathbb{P}(s_i = s).$$

At the k-th iteration of TRPO, the policy is updated by:

$$\theta_{k+1} = \operatorname*{argmax}_\theta \mathbb{E}_{\substack{s\sim\rho^{\pi_{\theta_k}}, \\ a\sim\pi_{\theta_k}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right],$$
$$\text{subject to } \mathbb{E}_{s\sim\rho^{\pi_{\theta_k}}} \left[\mathcal{D}_{\mathrm{KL}}(\pi_{\theta_k}(\cdot|s)\|\pi_\theta(\cdot|s)\right] \leq \delta, \quad (1)$$

where δ is a tuning parameter for controlling the size of the trust region, and $\mathcal{D}_{\mathrm{KL}}(P||Q) = \int_{\mathcal{X}} \log\left(\frac{dP}{dQ}\right) dQ$ denotes the Kullback-Leibler divergence between two distributions P, Q over support $\mathcal{X}$. For each update, the algorithm: (i) Samples trajectories using the current policy π_{θ_k}; (ii) Approximates $A^{\pi_{\theta_k}}$ for each state-action pair by taking the discounted sum of future rewards along the trajectory; (iii) Replaces the expectation in (1) and $A^{\pi_{\theta_k}}$ by sample approximation, then solves (1) with conjugate gradient algorithm.

2.2. Deep Deterministic Policy Gradient (DDPG)

DDPG uses the actor-critic architecture, where the agent learns a parameterized state-action value function Q_ϕ (also known as the critic) to update the parameterized deterministic policy μ_θ (also known as the actor).

DDPG uses a replay buffer, which is also used in in Deep Q-Network (Mnih et al., 2013). The replay buffer is a finite sized cache. Transitions are sampled from the environment according to the policy and the tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer. When the replay buffer is full, the oldest samples are discarded. At each step, μ_θ and Q_ϕ are updated by sampling a mini-batch from the buffer.

Update of state-action value function. The update of the state-action value function network Q_ϕ depends on the deterministic Bellman equation:

$$Q_\phi(s,a) = \mathbb{E}_{s'\sim\mathbb{P}(\cdot|s,a)} \left[r(s,a) + \gamma Q_\phi(s', \pi_\theta(s'))\right]. \quad (2)$$

The expectation depends only on the environment. This means that unlike TRPO, DDPG is an off-policy method, which can use transitions generated from a different stochastic behavior policy denoted as β (see Lillicrap et al. (2015) for detail). At the t-th iteration, we update the Q_ϕ by minimizing the associated mean squared Bellman error of transitions $\{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}_{i\in B}$ sampled from the replay buffer. Specifically, let $Q_{\phi'_t}$ and $\mu_{\theta'_t}$ be a pair of target networks, we set $y_t^i = r_t^i + \gamma Q_{\phi'_t}(s_{t+1}^i, \mu_{\theta'_t}(s_{t+1}^i))$, and then update the critic network:

$$\phi_{t+1} = \operatorname*{argmin}_\phi \sum_{i\in B} \left(y_t^i - Q_\phi(s_t^i, a_t^i)\right)^2.$$

After both critic and actor networks are updated, we update the target networks by slowly tracking the critic and actor networks: $\phi'_{t+1} = \tau\phi_{t+1} + (1-\tau)\phi'_t, \theta'_{t+1} = \theta\phi_{t+1} + (1-\tau)\theta'_t$ with, $\tau \ll 1$.

Update of policy. The policy network μ_θ is updated by maximizing the value function using policy gradient:

$$\max_\theta \mathbb{E}_{s\sim\rho^\beta} \left[Q_\phi(s,a)\big|_{a=\pi_\theta(s)}\right]. \quad (3)$$

Similar to updating the critic, we use the minibatch sampled from the replay buffer to compute approximated gradient of θ and perform the update:

$$\theta_{t+1} = \theta_t + \frac{\eta_t}{|B|} \sum_{i\in B} \nabla_a Q_{\phi_{t+1}}(s,a)\Big|_{\substack{s=s_i, \\ a=\mu_{\theta_t}(s_i)}} \nabla_\theta \mu_{\theta_t}(s_i).$$

3. Method

We present the smoothness-inducing regularizer in its general form and describe its intuition in detail. We also apply the proposed regularizer to popular reinforcement learning algorithms to demonstrate its great adaptability.

3.1. Learning Policy with SR^{2}L

We first focus on directly learning smooth policy π_θ with the proposed regularizer. We assume that the state space is continuous, i.e., $\mathcal{S} \subseteq \mathbb{R}^S$.

For a fixed state $s \in \mathcal{S}$ and a policy π_θ, **SR^{2}L** encourages the output $\pi_\theta(s)$ and $\pi_\theta(\widetilde{s})$ to be similar, where state $\widetilde{s}$ is obtained by injecting a small perturbation to state s. We assume the perturbation set $\mathbb{B}_d(s,\epsilon) = \{\widetilde{s} : d(s,\widetilde{s}) \leq \epsilon\}$ is an ϵ-radius ball measured in metric $d(\cdot,\cdot)$, which is often chosen to be the ℓ_p distance: $d(s,\widetilde{s}) = \|s - \widetilde{s}\|_p$. To measure the discrepancy between the outputs of a policy, we adopt a suitable metric function denoted by $\mathcal{D}$. The non-smoothness of policy π_θ at state s is defined in an adversarial manner:

$$\max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} \mathcal{D}(\pi_\theta(s), \pi_\theta(\widetilde{s})).$$

To obtain a smooth policy π_θ, we encourage smoothness at each state of the entire trajectory. We achieve this by taking expectation with respect to the state visitation distribution ρ^π induced by the policy, and our smoothness-inducing regularizer is defined by:

$$\mathcal{R}_{\mathrm{s}}^\pi(\theta) = \mathbb{E}_{s\sim\rho^{\pi_\theta}} \max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} \mathcal{D}(\pi_\theta(s), \pi_\theta(\widetilde{s})). \quad (4)$$

For a stochastic policy π_θ, we set the metric $\mathcal{D}(\cdot,\cdot)$ to be the Jeffrey's divergence, and the regularizer takes the form

$$\mathcal{R}_{\mathrm{s}}^\pi(\theta) = \mathbb{E}_{s\sim\rho^\pi} \max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} \mathcal{D}_{\mathrm{J}}(\pi(\cdot|s) \,||\, \pi(\cdot|\widetilde{s})), \quad (5)$$

where the Jeffrey's divergence for two distributions P, Q is defined by:

$$\mathcal{D}_{\mathrm{J}}(P||Q) = \frac{1}{2}\mathcal{D}_{\mathrm{KL}}(P||Q) + \frac{1}{2}\mathcal{D}_{\mathrm{KL}}(Q||P). \quad (6)$$

For a deterministic policy μ_θ, we set the metric $\mathcal{D}(\cdot,\cdot)$ to be the squared ℓ_2 norm of the difference:

$$\mathcal{R}_{\mathrm{s}}^{\mu}(\theta) = \mathop{\mathbb{E}}_{s\sim\rho^{\mu}} \max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} \|\mu(s)-\mu(\widetilde{s})\|_2^2. \tag{7}$$

The smoothness-inducing adversarial regularizer is essentially measuring the local Lipschitz continuity of policy π_θ under the metric $\mathcal{D}$. More precisely, we encourage the output (decision) of π_θ to not change much if we inject a small perturbation bounded in metric $d(\cdot,\cdot)$ to the state s (See Figure 1). Therefore, by adding the regularizer (4) to the policy update, we can encourage π_θ to be smooth within the neighborhoods of all states on all possible trajectories regarding to the sampling policy. Such a smoothness-inducing property is particularly helpful to prevent overfitting, improve sample efficiency and overall training stability .

TRPO with SR^{2}L (TRPO-SR). We now apply the proposed smoothness inducing regularizer to TRPO algorithm, which is itself an on-policy algorithm. Since TRPO uses a stochastic policy, we use the Jeffrey's divergence to penalize the discrepancy between decisions for the regular state and the adversarially perturbed state, as suggested in (5).

Specifically, TRPO with smoothness-inducing regularizer updates the policy by solving the following subproblem at the k-th iteration:

$$\begin{aligned}\theta_{k+1} = \mathop{\mathrm{argmin}}_{\theta} & -\mathbb{E}_{\substack{s\sim\rho^{\pi_{\theta_k}},\\ a\sim\pi_{\theta_k}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a)\right] \\ & + \lambda_{\mathrm{s}}\mathbb{E}_{s\sim\rho^{\pi_{\theta_k}}} \max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} \mathcal{D}_{\mathrm{J}}(\pi_\theta(\cdot|s)\,||\,\pi_\theta(\cdot|\widetilde{s})), \\ \text{s.t.}\quad & \mathbb{E}_{s\sim\rho^{\pi_{\theta_k}}}\left[\mathcal{D}_{\mathrm{KL}}(\pi_{\theta_k}(\cdot|s)\|\pi_\theta(\cdot|s)\right] \le \delta. \end{aligned} \tag{8}$$

3.2. Learning Q-function with Smoothness-inducing Regularization

The proposed smoothness-induced regularizer can be also used to learn a smooth Q-function, which can be further used to generate a smooth policy.

We measure the non-smoothness of a Q-function at state-action pair (s,a) by the squared difference of the state-action value between the normal state and the adversarially perturbed state: $\max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)}(Q_\phi(s,a)-Q_\phi(\widetilde{s},a))^2$. To enforce smoothness at every state-action pair, we take expectation with respect to the entire trajectory, and the smoothness-inducing regularizer takes the form

$$\mathcal{R}_s^Q(\phi) = \mathbb{E}_{\substack{s\sim\rho^{\beta},\\ a\sim\beta}} \max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} (Q_\phi(s,a)-Q_\phi(\widetilde{s},a))^2.$$

where β denotes the behavior policy for sampling in off-policy training setting.

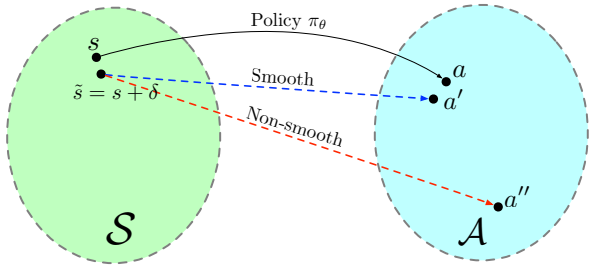


Figure 1. Smoothness of policy π_θ at state s. If policy π_θ is smooth at state s, then perturbed state $\widetilde{s}$ leads to action a' similar to the original action a. If the policy π_θ is non-smooth at state s, then the perturbed state $\widetilde{s}$ leads to drastically different action a''.

DDPG with SR^{2}L. We now apply the proposed smoothness-inducing regularizer to DDPG algorithm, which is itself an off-policy algorithm. Since DDPG uses two networks: the actor network and the critic network, we propose two variants of DDPG, where the regularizer is applied to the actor or the critic network.

• **Regularizing the Actor Network** (DDPG-SR-A). We can directly penalize the non-smoothness of the actor network to promote a smooth policy in DDPG. Since DDPG uses a deterministic policy μ_θ, when updating the actor network, we penalize the squared difference as suggested in (7) and minimize the following objective:

$$\mathop{\mathbb{E}}_{s\sim\rho^{\beta}}\left[-Q_\phi(s,a)\big|_{a=\mu_\theta(s)} + \lambda_{\mathrm{s}} \max_{\widetilde{s}\sim\mathbb{B}_d(s,\epsilon)} \|\mu_\theta(s)-\mu_\theta(\widetilde{s})\|_2^2\right].$$

The policy gradient can be written as:

$$\begin{aligned}\mathop{\mathbb{E}}_{s\sim\rho^{\beta}}\Big[& -\nabla_a Q_\phi(s,a)\big|_{a=\mu_\theta(s)}\nabla_\theta\mu_\theta(s) \\ & + \lambda_{\mathrm{s}}\nabla_\theta\|\mu_\theta(s)-\mu_\theta(\widetilde{s})\|_2^2\Big],\end{aligned}$$

with $\widetilde{s} = \mathop{\mathrm{argmax}}_{\widetilde{s}\sim\mathbb{B}_d(s,\epsilon)} \|\mu_\theta(s)-\mu_\theta(\widetilde{s})\|_2^2$ for $s\sim\rho^{\beta}$.

• **Regularizing the Critic Network** (DDPG-SR-C). Since DDPG simultaneously learns a Q-function (critic network) to update the policy (actor network), inducing smoothness in the critic network could also help us to generate a smooth policy. By incorporating the proposed regularizer for penalizing Q-function, we obtain the following update for inducing a smooth Q-function in DDPG:

$$\begin{aligned}\phi_{t+1} = \mathop{\mathrm{argmin}}_{\phi} & \sum_{i\in B}\left(y_t^i - Q_\phi(s_t^i,a_t^i)\right)^2 \\ & + \lambda_{\mathrm{s}}\sum_{i\in B}\max_{\widetilde{s}_t^i\sim\mathbb{B}_d(s_t^i,\epsilon)} (Q_\phi(s_t^i,a_t^i)-Q_\phi(\widetilde{s}_t^i,a_t^i))^2,\end{aligned}$$

$$\text{with}\quad y_t^i = r_t^i + \gamma Q_{\phi_t'}(s_{t+1}^i, \mu_{\theta_t'}(s_{t+1}^i)), \forall i\in B,$$

where B is the mini-batch sampled from the replay buffer.

3.3. Solving the Min-max Problem

Adding the smoothness-inducing regularizer in the policy/Q-function update often involves solving a min-max problem. Though the inner max problem is not concave, simple

stochastic gradient algorithm has been shown to be able to solve it efficiently in practice. Below we describe how to perform the update of TRPO-SR, including solving the corresponding min-max problem. The details are summarized in Algorithm 1. We leave the detailed description of DDPG-SR-A and DDPG-SR-C in the appendix.

Algorithm 1 Trust Region Policy Optimization with Smoothness-inducing Regularization.

Input: step sizes η_δ, η_θ, number of iterations D for inner optimization, number of iterations K for policy updates, perturbation strength ϵ, regularization coefficient λ_{s}.
Initialize: randomly initialize the policy network π_{θ_0}.
for $k = 1, \ldots, K-1$ **do**
  Sample trajectory $\mathcal{S}_k = \{(s_k^t, a_k^t)\}_{t=1}^T$ from current policy π_{θ_k}.
  Estimate advantage function $\widehat{A}^{\pi_{\theta_k}}(s,a)$ using sample approximation.
  for $s_k^t \in \mathcal{S}_k$ **do**
    Randomly initialize δ_0^t.
    for $\ell = 0, \ldots, D-1$ **do**
      $\delta_{\ell+1}^t = \delta_\ell^t + \eta_\delta \nabla_\delta \mathcal{D}_{\mathrm{JS}}(\pi_{\theta_k}(\cdot|s_k^t) || \pi_{\theta_k}(\cdot|s_k^t + \delta_\ell^t))$.
      $\delta_{\ell+1}^t = \Pi_{\mathbb{B}_d(0,\epsilon)}\left(\delta_{\ell+1}^t\right)$.
    end for
    $\widetilde{s}_k^t = s_k^t + \delta_D^t$.
  end for

$$\theta_{k+1} = \theta_k + \eta_\theta \sum_{(s_k^t, a_k^t) \in \mathcal{S}_k} \frac{\widehat{A}^{\pi_{\theta_k}}(s_k^t, a_k^t)}{\pi_{\theta_k}(a_k^t|s_k^t)} \nabla_\theta \pi_{\theta_k}(a_k^t|s_k^t) - \eta_\theta \lambda_{\mathrm{s}} \sum_{s_k^t \in \mathcal{S}_k} \gamma^t \nabla_\theta \mathcal{D}_{\mathrm{J}}(\pi_{\theta^k}(\cdot|s_k^t) || \pi_{\theta^k}(\cdot|\widetilde{s}_k^t)).$$

end for

4. Connection to (Distributionally) Robust Reinforcement Learning

Besides promoting the smoothness of the learnt policy, our proposed regularizer also enjoys another advantage – improving *robustness against measurement error* in the state space. Specifically, we consider a noisy reinforcement learning enviroment, where the agent can only observe inexact state information. Taking the robot motion planning as an example, the robot gets its locations and velocity from the equipped sensors, which often encouter systematic or stochastic measurement error.

To address this issue, researchers usually resort to Partially observable Markov decision process (POMDP, Monahan (1982)) to model the inexact state information as a conditional distribution that depends on the exact state. However, POMDP can only handle i.i.d. stochastic measurement errors, and require the prior knowledge of the measurement error distribution. In contrast, our regularizer can handle more complex non i.i.d. error, or even adversarial measurement error. Our proposed regularizer encourages the policy to make similar actions for any pair of states that are close to each other, which implies that actions for the observed state and true state should be close. This inductive bias is well suited for RL with smooth environments, whose optimal policy does not drastically change its decision when adding a small perturbation to the state.

Our regularizer can be naturally extended to *distributionally robust* settings, where the observed state comes from a state-visitation distribution ρ' that is close to the state-visitation distribution ρ of the true state. Specifically, the regularizer takes the form

$$\mathcal{R}_{\mathrm{s}}^\pi(\theta) = \max_{\mathcal{F}(\rho,\rho') \le \epsilon} \mathbb{E}_{s\sim\rho, s'\sim\rho'} \mathcal{D}\left(\pi_\theta(s), \pi_\theta(s')\right),$$

where $\mathcal{F}(\cdot,\cdot)$ denotes some discrepancy measure between a pair of visitation probability distributions (e.g., Wasserstein distance, f-divergence). For more details on solving distributionally robust optimization via duality, please refer to Gao & Kleywegt (2016).

5. Experiment

We apply the proposed **SR^{2}L** training framework to two popular reinforcement learning algorithms: TRPO and DDPG. Both of these algorithms have become the standard routine to solve large-scale control tasks, and the building blocks of many state-of-the-art reinforcement learning algorithms. For TRPO, we directly learns a smooth policy; For DDPG, we promote the smoothness either in the actor (policy) or the critic (Q-function).

5.1. Implementation

Our implementation of **SR^{2}L** training framework is based on the open source toolkit garage (garage contributors, 2019). We test our algorithms on OpenAI gym (Brockman et al., 2016) control environments with the MuJoCo (Todorov et al., 2012) physics simulator. For all tasks, we use a network of 2 hidden layers, each containing 64 neurons, to parameterize the policy and the Q-function. For fair comparison, except for the hyper-parameters related to the smooth regularizer, we keep all the hyper-parameters the same as in the original implementation of garage. We use the grid search to select the hyper-parameters (perturbation strength ϵ, regularization coefficient λ_{s}) of the smoothness-inducing regularizer. We set the search range to be $\epsilon \in [10^{-5}, 10^{-1}]$, $\lambda_{\mathrm{s}} \in [10^{-2}, 10^2]$. To solve the inner maximization problem, we run 10 steps of projected gradient ascent, with step size set as 0.2ϵ. For each algorithm and each environment, we train 10 policies with different initialization for 500 iterations (1K environment steps for each iteration). Below we briefly describe the environments used to evaluate our algorithms (See also Figure 2).

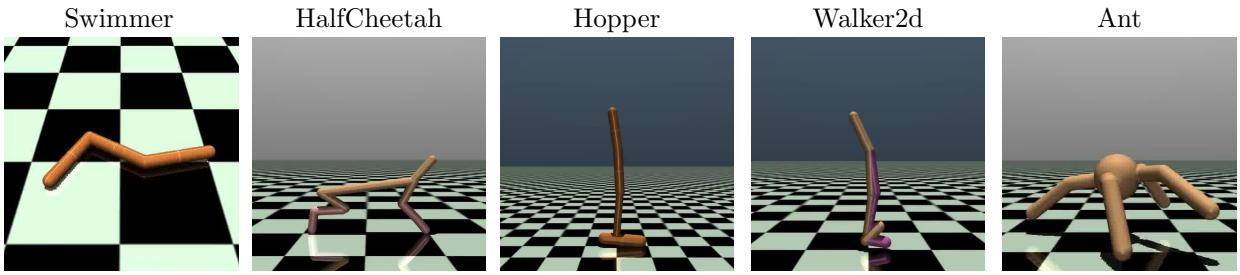


Figure 2. OpenAI Gym Mujoco Benchmarks.

Swimmer. The swimmer is a planar robot of a single torso with 3 links and 2 actuated joints in a viscous container. The 8-dimensional state space includes positions and velocities of sliders, angles and angular velocity of hinges. The 2-dimensional action space includes torque of each actuated joints.

HalfCheetah. The half-cheetah is a planar biped robot with 7 rigid links, including two legs and a torso, along with 6 actuated joints. The 17-dimensional state space includes positions and velocities of sliders, angles and angular velocity of hinges. The 6-dimensional action space includes torque of each actuated joints.

Walker2D. The walker is a planar biped robot consisting of 7 links, corresponding to two legs and a torso, along with 6 actuated joints. The 17-dimensional state space includes positions and velocities of sliders and angles and angular velocity of hinges. The 6-dimensional action space includes torque of each actuated joints.

Hopper. The hopper is a planar monopod robot with 4 rigid links, corresponding to the torso, upper leg, lower leg, and foot, along with 3 actuated joints. The 11-dimensional state space includes positions and velocities of sliders, angles and angular velocity of hinges. The 3-dimensional action space includes torque of each actuated joints.

Ant. The ant is a planar biped robot consisting of 9 links, corresponding to 4 legs and a torso, along with 8 actuated joints. The 111-dimensional state space includes positions and velocities of sliders, angles and angular velocity of hinges. The 8-dimensional action space includes torque of each actuated joints.

5.2. Evaluating the Learned Policies

TRPO with SR^{2}L (TRPO-SR). We use Gaussian policy in our implementation. Specifically, for a given state s, the action follows a Gaussian distribution $a \sim N(\pi_\theta(s), \sigma^2 I_A)$, where σ is also a learnable parameter. Then the smoothness-inducing regularizer (5) takes the form: $\mathcal{R}_{\mathrm{s}}^{\pi}(\theta) = \mathbb{E}_{s\sim\rho^\pi} \max_{\widetilde{s}\in\mathbb{B}_d(s,\epsilon)} \|\pi_\theta(s) - \pi_\theta(\widetilde{s})\|_2^2/\sigma^2$.

Figure 3 shows the mean and variance of the cumulative reward (over 10 policies) for policies trained by TRPO-SR and TRPO for Swimmer, HalfCheetah, Hopper and Ant. For all the four tasks, TRPO-SR learns a better policy in terms of the mean cumulative reward. In addition, TRPO-SR enjoys a smaller variance of the cumulative reward with respect to different initializations. These two observations confirm that our smoothness-inducing regularization improves sample efficiency as well as the training stability.

We further show that the advantage of our proposed **SR^{2}L** training framework goes beyond improving the mean cumulative reward. To show this, we run the algorithm with 10 different initializations, sort the cumulative rewards of learned policies and plot the percentiles in Figure 4. For all four tasks, TRPO-SR uniformly outperforms the baseline TRPO. For Swimmer and HalfCheetah tasks, TRPO-SR significantly improves the worst case performance compared to TRPO, and have similar best case performance. For Walker and Ant tasks, TRPO-SR significantly improves the best case performance compared to TRPO. Our empirical results show strong evidences that the proposed **SR^{2}L** not only improves the average reward, but also makes the training process significantly more robust to failure case

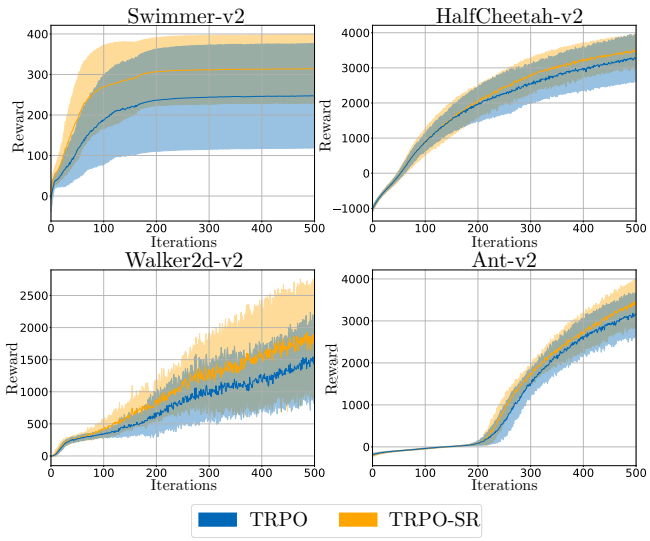


Figure 3. Learning curves (mean±standard deviation) for TRPO-SR (orange) trained policies versus the TRPO (blue) baseline when tested in clean environment. For all the tasks, TRPO-SR achieves a better mean reward than TRPO, with a reduction of variance in initial stage.

compared to the baseline method.

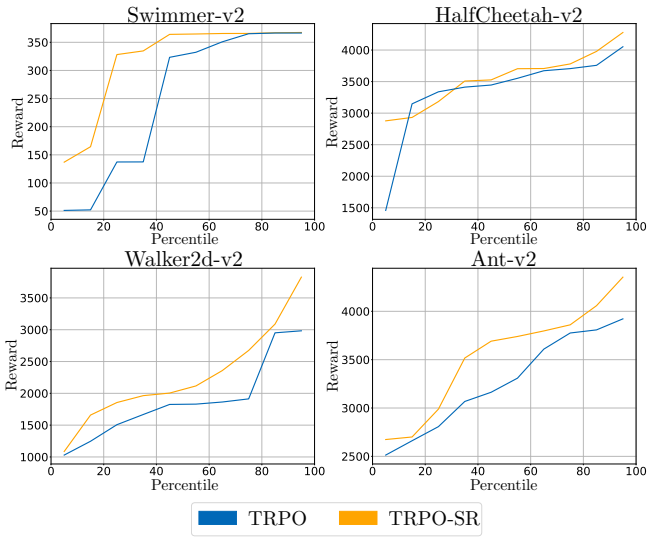


Figure 4. Percentile plots for TRPO-SR (orange) trained policies versus the TRPO (blue) baseline when tested in clean environment. Algorithms are run on ten different initializations and sorted to show the percentiles of cumulative reward. For all the tasks, TRPO-SR achieves better or competitive best performance, and better worst case performance, compared with TRPO baseline.

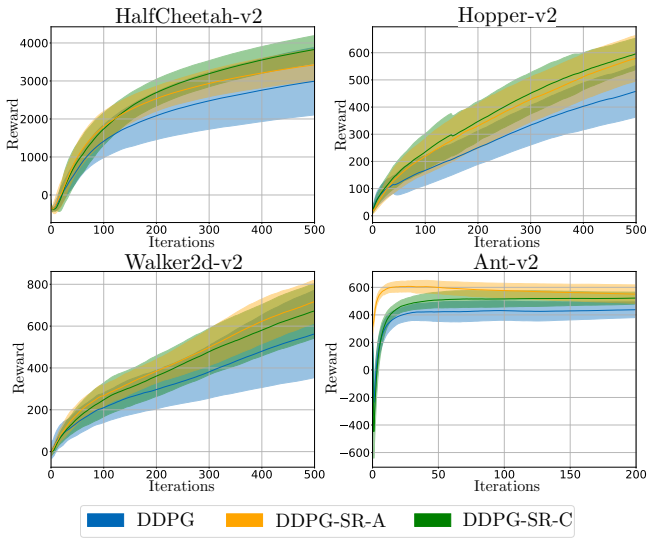


Figure 5. Learning curves (mean±standard deviation) for DDPG-SR-A (orange) and DDPG-SR-C (green) trained policies versus the DDPG (blue) baseline when tested in clean environment. For all the tasks, DDPG-SR-A(C) trained policies achieve better mean reward compared to DDPG.

DDPG with SR^{2}L. We repeat the same evaluations for applying the proposed **SR^{2}L** framework to DDPG (DDPG-SR-A and DDPG-SR-C). Figure 5 shows the mean and variance of the cumulative reward for policies trained by DDPG-SR-A and DDPG-SR-C in HalfCheetah, Hopper and

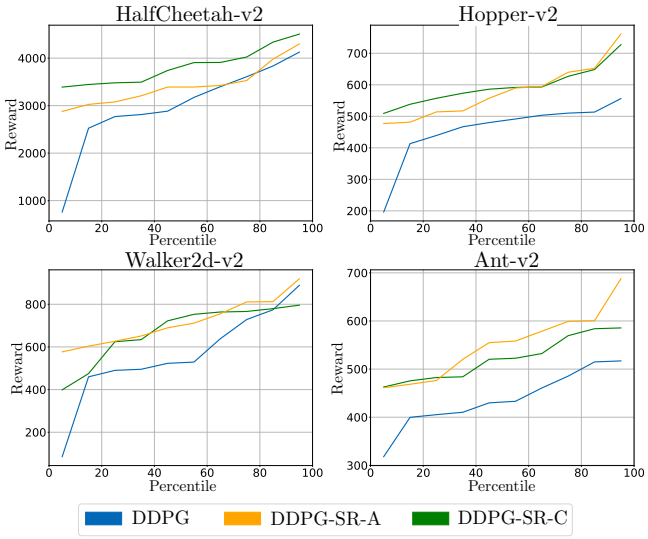


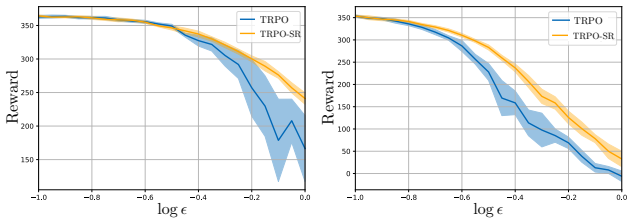
Figure 6. Percentile plots for DDPG-SR-A (orange) and DDPG-SR-C (green) trained policies versus the DDPG (blue) baseline when tested in clean environment. Algorithms are run with 10 different initializations and sorted to show the percentiles of cumulative reward. For all the tasks, DDPG-SR-A(C) achieves better or competitive best performance, and significantly better worst case performance, compared with DDPG baseline.

Walker2D and Ant environments. For all the four tasks, DDPG-SR learns a better policy in terms of mean reward. For task Ant, DDPG-SR-A shows superior training stability, which is the only algorithm without drastic decay in the initial training stage. In addition, DDPG-SR-C shows competitive performance compared to DDPG-SR-A, significantly outperforms DDPG-SR-A and DDPG for task HalfCheetah. This shows that instead of directly learning a smooth policy, we can turn to learn a smooth Q-function and obtain similar performance benefits.

Figure 6 plots percentiles of cumulative reward of learned policies using DDPG and DDPG-SR. Similar to TRPO-SR, both DDPG-SR-A and DDPG-SR-C uniformly outperform the baseline DDPG for all the reward percentiles. DDPG-SR is able to significantly improve the the worst case performance, while maintaining competitive best case performance compared to DDPG.

5.3. Robustness against Measurement Error

We demonstrate that by inducing smoothness into the policy using **SR^{2}L** training framework , the trained policy is able to achieve robustness against both stochastic and adversarial measurement error, a classical setting considered in partially observable Markov decision process (POMDP) (Monahan, 1982). To show this, we evaluate the robustness of the proposed **SR^{2}L** training framework in the Swimmer and HalfCheetah environments. We evaluate the trained policy with two types of disturbances in the test environ-

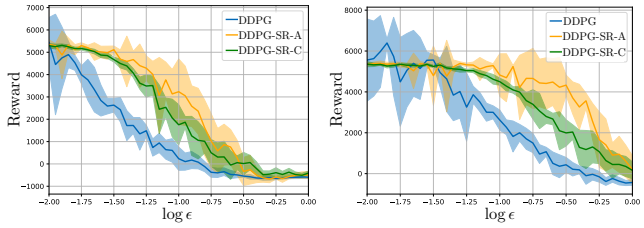


(a) Swimmer - Random Disturbed Rollout (b) Swimmer - Adversarially Disturbed Rollout

Figure 7. Plot of cumulative reward (mean±standard deviation with multiple rollouts) of TRPO-SR (orange) trained policies versus the TRPO (blue) baseline when tested in disturbed environment. The random disturbance is uniformly sampled from set $\mathbb{B}_d(0,\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$. The adversarial disturbance belongs to the set $\mathbb{B}_d(0,\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$. TRPO-SR trained policies achieve a slower decline in performance than TRPO as we increase disturbance strength, and a significant reduction of variance under large disturbance strength.

ment: for a given state s, we add it with either (i) random disturbance which are sampled uniformly from $\mathbb{B}_d(0,\epsilon)$, or (ii) adversarial disturbance, which are generated by solving: $\widetilde{\delta} = \text{argmax}_{\delta \in \mathbb{B}_d(0,\epsilon)} \mathcal{D}(\pi_\theta(s), \pi_\theta(s+\delta))$ using 10 steps of projected gradient ascent. For all evaluations, we use disturbance set $\mathbb{B}_d(0,\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$. For each policy and disturbed environment, we do 10 stochastic rollouts to evaluate the policy and plot the cumulative reward of policy.

To evaluate the robustness of TRPO with **SR^{2}L**, we run both baseline TRPO and TRPO-SR in the Swimmer environment. Figure 7 plots the cumulative reward against the disturbance strength (ϵ). We see that for both random and adversarial disturbance, increasing the strength of the disturbance decreases the cumulative reward of the learned policies. On the other hand, we see that TRPO-SR clearly achieves improved robustness against perturbations, as its reward declines much slower than the baseline TRPO.



(a) HalfCheetah - Adversarially Disturbed Rollout (b) HalfCheetah - Random Disturbed Rollout

Figure 8. Plot of cumulative reward (mean±standard deviation with multiple rollouts) of DDPG-SR-A (orange) and DDPG-SR-C (green) trained policies versus the DDPG (blue) baseline when tested in disturbed environment. The random disturbance is uniformly sampled from set $\mathbb{B}_d(0,\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$. The adversarial disturbance belongs to the set $\mathbb{B}_d(0,\epsilon) = \{\delta : \|\delta\|_\infty \leq \epsilon\}$. As disturbance strength ϵ increases, DDPG-SR-A and DDPG-SR-C trained policies has slower decline in performance than DDPG. To evaluate the robustness of DDPG with **SR^{2}L**, we run baseline DDPG, DDPG-SR-A and DDPG-SR-C in the HalfCheetah environment. Figure 8 plots the cumulative reward against the disturbance strength (ϵ). We see that incorporating the proposed smoothness-inducing regularizer into either the actor or the critic network improves the robustness of the learned policy against state disturbances.

5.4. Sensibility with Hyperparameters

The proposed smoothness-inducing regularizer involves setting two hyper-parameters, the coefficient of the regularizer λ_s and the disturbance strength ϵ. We vary different choices of (ϵ, λ_s) and plot the heatmap of cumulative reward for each configuration in Figure 9. In principle, λ_s and ϵ both control the strength of the regularization in similar way, as large λ_s and ϵ both increase the strength of the regularization, advocating for more smoothness; small λ_s and ϵ both decrease the strength of the regularization, favoring for cumulative reward over smoothness. We observe similar behavior in Figure 9: a relatively small λ_s and large ϵ shows similar performance with a relatively large λ_s and small ϵ.

6. Conclusion

We develop a novel regularization based training framework **SR^{2}L** to learn a smooth policy in reinforcement learning. The proposed regularizer encourages the learned policy to produce similar decisions for similar states. It can be applied to either induce smoothness in the policy directly, or induce smoothness in the Q-function, thus enjoys great applicability. We demonstrate the effectiveness of **SR^{2}L** by applying it to two popular reinforcement learning algorithms, including TRPO and DDPG. Our empirical results show that **SR^{2}L** improves sample efficiency and training stability of current algorithms. In addition, the induced smoothness in the learned policy also improves robustness against both random and adversarial perturbations to the state.

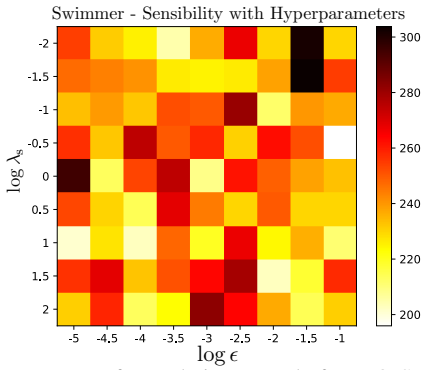


Figure 9. Heatmap of cumulative reward of TRPO-SR trained policies with different ϵ and λ_s. Each policy is trained for 2M environment steps.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Boyan, J. A. and Moore, A. W. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pp. 369–376, 1995.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Cheng, R., Verma, A., Orosz, G., Chaudhuri, S., Yue, Y., and Burdick, J. W. Conrol regularization for reduced variance reinforcement learning. *arXiv preprint arXiv:1905.05380*, 2019.

Gao, R. and Kleywegt, A. J. Distributionally robust stochastic optimization with wasserstein distance. *arXiv preprint arXiv:1604.02199*, 2016.

garage contributors, T. Garage: A toolkit for reproducible reinforcement learning research. https://github.com/rlworkgroup/garage, 2019.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Hampel, F. R. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974.

Hendrycks, D., Mazeika, M., Kadavath, S., and Song, D. Using self-supervised learning can improve model robustness and uncertainty. *arXiv preprint arXiv:1906.12340*, 2019.

Huang, L., Liu, X., Lang, B., Yu, A. W., Wang, Y., and Li, B. Orthogonal weight normalization: Solution to optimization over multiple dependent stiefel manifolds in deep neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Zhao, T. Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization. *arXiv preprint arXiv:1911.03437*, 2019.

Jin, J., Song, C., Li, H., Gai, K., Wang, J., and Zhang, W. Real-time bidding with multi-agent reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pp. 2193–2201, 2018.

Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., and Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37(4-5):421–436, 2018.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Miyato, T., Maeda, S.-i., Ishii, S., and Koyama, M. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Monahan, G. E. State of the art?a survey of partially observable markov decision processes: theory, models, and algorithms. *Management science*, 28(1):1–16, 1982.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817–2826. JMLR. org, 2017.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897, 2015.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Thrun, S. and Schwartz, A. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.

Xie, Q., Dai, Z., Hovy, E., Luong, M.-T., and Le, Q. V. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

Zhao, J., Qiu, G., Guan, Z., Zhao, W., and He, X. Deep reinforcement learning for sponsored search real-time bidding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1021–1030, 2018.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N. J., Xie, X., and Li, Z. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pp. 167–176, 2018.

A. Appendix

We present two variants of DDPG with the proposed smoothness-inducing regularizer. The first algorithm, DDPG-SR-A, directly learns a smooth policy with a regularizer that measures the non-smoothness in the actor network (policy). The second variant, DDPG-SR-C, learns a smooth Q-function with a regularizer that measure the non-smoothness in the critic network (Q-function). We present the details of DDPG-SR-A and DDPG-SR-C in Algorithm 2 and Algorithm 3, respectively.

Algorithm 2 DDPG with smoothness-inducing regularization on the actor network (DDPG-SR-A).

Input: step size for target networks $\alpha \in (0,1)$, coefficient of regularizer λ_{s}, perturbation strength ϵ, number of iterations to solve inner optimization problem D, number of training steps T, number of training episodes M, step size for inner maximization η_δ, step size for updating actor/critic network η.

Initialize: randomly initialize the critic network $Q_\phi(s,a)$ and the actor network $\mu_\theta(s)$, initialize target networks $Q_{\phi'}(s,a)$ and $\mu_{\theta'}(s)$ with $\phi' = \phi$ and $\theta' = \theta$, initialize replay buffer $\mathcal{R}$.

for episode = 1 ..., M **do**

Initialize a random process ϵ for action exploration.

Observe initial state s_1.

for $t = 1 \ldots T$ **do**

Select action $a_t = \mu_\theta(s_t) + \epsilon_t$ where ϵ_t is the exploration noise.

Take action a_t, receive reward r_t and observe the new state s_{t+1}.

Store transition (s_t, a_t, r_t, s_{t+1}) into the replay buffer $\mathcal{R}$.

Sample mini-batch B of transitions $\{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}_{i\in B}$ from the replay buffer $\mathcal{R}$.

Set $y_t^i = r_t^i + \gamma Q_{\phi'}(s_{t+1}^i, \mu_{\theta'}(s_{t+1}^i))$ for $i \in B$.

Update the critic network: $\phi \leftarrow \mathrm{argmin}_{\widetilde{\phi}} \sum_{i\in B}(y_t^i - Q_{\widetilde{\phi}}(s_t^i, a_t^i))^2$.

for $s_t^i \in B$ **do**

Randomly initialize δ_i.

for $\ell = 1 \ldots D$ **do**

$\delta_i \leftarrow \delta_i + \eta_\delta \nabla_\delta \big\|\mu_\theta(s_t^i) - \mu_\theta(s_t^i + \delta_i)\big\|_2^2$.

$\delta_i \leftarrow \Pi_{\mathbb{B}_d(0,\epsilon)}(\delta_i)$.

end for

Set $\widehat{s}_t^i = s_t^i + \delta_i$.

end for

Update the actor network:

$$\theta \leftarrow \theta + \frac{\eta}{|B|}\sum_{i\in B}\left(\nabla_a Q_\phi(s,a)\big|_{s=s_t^i, a=u_\theta(s_t^i)} \nabla_\theta \mu_\theta(s)\big|_{s=s_t^i} - \lambda_s \nabla_\theta \big\|\mu_\theta(s_t^i) - \mu_\theta(\widehat{s}_t^i)\big\|_2^2\right).$$

Update the target networks:

$$\theta' \leftarrow \alpha\theta + (1-\alpha)\theta',$$
$$\phi' \leftarrow \alpha\phi + (1-\alpha)\phi'.$$

end for

end for

Algorithm 3 DDPG with smoothness-inducing regularization on the critic network (DDPG-SR-C).

Input: step size for target networks $\alpha \in (0,1)$, coefficient of regularizer λ_{s}, perturbation strength ϵ, number of iterations to solve inner optimization problem D, number of training steps T, number of training episodes M, step size for inner maximization η_δ, step size for updating actor/critic network η.

Initialize: randomly initialize the critic network $Q_\phi(s,a)$ and the actor network $\mu_\theta(s)$, initialize target networks $Q_{\phi'}(s,a)$ and $\mu_{\theta'}(s)$ with $\phi' = \phi$ and $\theta' = \theta$, initialize replay buffer $\mathcal{R}$.

for episode = $1 \ldots$, M **do**

Initialize a random process ϵ for action exploration.

Observe initial state s_1.

for $t = 1 \ldots T$ **do**

Select action $a_t = \mu_\theta(s_t) + \epsilon_t$ where ϵ_t is the exploration noise.

Take action a_t, receive reward r_t and observe the new state s_{t+1}.

Store transition (s_t, a_t, r_t, s_{t+1}) into replay buffer $\mathcal{R}$.

Sample mini-batch B of transitions $\{(s_t^i, a_t^i, r_t^i, s_{t+1}^i)\}_{i\in B}$ from the replay buffer $\mathcal{R}$.

Set $y_t^i = r_t^i + \gamma Q_{\phi'}(s_{t+1}^i, \mu_{\theta'}(s_{t+1}^i))$ for $i \in B$.

for $s_t^i \in B$ **do**

Randomly initialize δ_i.

for $\ell = 1 \ldots D$ **do**

$\delta_i \leftarrow \delta_i + \eta_\delta \nabla_\delta (Q_\phi(s_t^i, a_t^i) - Q_\phi(s_t^i + \delta, a_t^i))^2$.

$\delta_i \leftarrow \Pi_{\mathbb{B}_d(0,\epsilon)}(\delta_i)$.

end for

Set $\widehat{s}_t^i = s_t^i + \delta_i$.

end for

Update the critic network:

$$\phi \leftarrow \operatorname*{argmin}_{\widetilde{\phi}} \sum_{i\in B} (y_t^i - Q_{\widetilde{\phi}}(s_t^i, a_t^i))^2 + \lambda_{\mathrm{s}} \sum_{i\in B} (Q_\phi(s_t^i, a_t^i) - Q_\phi(\widehat{s}_t^i, a_t^i))^2.$$

Update the actor network:

$$\theta \leftarrow \theta + \frac{\eta}{|B|} \sum_{i\in B} \nabla_a Q_\phi(s,a)\big|_{s=s_t^i, a=u_\theta(s_t^i)} \nabla_\theta \mu_\theta(s)\big|_{s=s_t^i}.$$

Update the target networks:

$$\theta' \leftarrow \alpha\theta + (1-\alpha)\theta',$$
$$\phi' \leftarrow \alpha\phi + (1-\alpha)\phi'.$$

end for

end for