# LipsNet: A Smooth and Robust Neural Network with Adaptive Lipschitz Constant for High Accuracy Optimal Control

**Xujie Song** [1] **Jingliang Duan** [2] **Wenxuan Wang** [1] **Shengbo Eben Li** [1] **Chen Chen** [1] **Bo Cheng** [1] **Bo Zhang** [3] **Junqing Wei** [3] **Xiaoming Simon Wang** [3]

## Abstract

Deep reinforcement learning (RL) is a powerful approach for solving optimal control problems. However, RL-trained policies often suffer from the action fluctuation problem, where the consecutive actions significantly differ despite only slight state variations. This problem results in mechanical components' wear and tear and poses safety hazards. The action fluctuation is caused by the high Lipschitz constant of actor networks. To address this problem, we propose a neural network named LipsNet. We propose the Multi-dimensional Gradient Normalization (MGN) method, to constrain the Lipschitz constant of networks with multi-dimensional input and output. Benefiting from MGN, LipsNet achieves Lipschitz continuity, allowing smooth actions while preserving control performance by adjusting Lipschitz constant. LipsNet addresses the action fluctuation problem at network level rather than algorithm level, which can serve as actor networks in most RL algorithms, making it more flexible and user-friendly than previous works. Experiments demonstrate that LipsNet has good landscape smoothness and noise robustness, resulting in significantly smoother action compared to the Multilayer Perceptron.

## 1. Introduction

Recently, deep reinforcement learning (RL) has emerged as an effective method for solving optimal control problems in the physical world (Guan et al., 2021). As neural networks can fit complex nonlinear functions conveniently (Hornik et al., 1989; Kidger & Lyons, 2020), neural networks have become RL's classical control policy container. However, the policy trained by RL severely suffers from the action fluctuation problem, which means the consecutive actions vibrate greatly despite only slight difference in states. This problem is often overlooked in simulation and training, but has important consequences in real-world applications. Fluctuating actions do not satisfy control requirements, resulting in the wear and tear on mechanical components and safety hazards. This problem is prevalent in a variety of scenarios, such as the drone control (Mysore et al., 2021; Shi et al., 2019), robot arm control (Yu et al., 2021) and autonomous driving (Chen et al., 2021; Wasala et al., 2020).

In order to make RL more applicable to the real-world usage, researchers are attempting to address the action fluctuation problem. Siddharth Mysore et al. (2021) found that filters would not be directly used to smoothen the actions, because it changes the dynamic response and breaks the Markov assumption, which results in anomalous behaviors. Therefore, the problem must be addressed during training. Current methods can be divided into four categories, which are action penalty methods (Mysore et al., 2021; Kobayashi, 2022), adversarial disturbance methods (Shen et al., 2020; Zhao et al., 2022), hierarchical network methods (Chen et al., 2021; Yu et al., 2021) and network enhancement methods (Takase et al., 2020; 2022).

In action penalty methods, penalty terms indicating action smoothness are added to the actor loss of RL. For example, the Conditioning for Action Policy Smoothness (CAPS) method (Mysore et al., 2021) introduces two penalty terms in actor loss. The first term indicates the similarity between adjacent actions in successive time steps. The second term indicates the similarity between actions under close states. In the Locally Lipschitz Continuous Constraint (L2C2) method (Kobayashi, 2022), the penalty terms include the similarity between actions under close states and the similarity between critic values under close states. The distance between the actual state and the sampled close state in L2C2 is bounded by a multiple of the distance between the actual state and the subsequent state in the next

[1]School of Vehicle and Mobility, Tsinghua University, Beijing, China [2]School of Mechanical Engineering, University of Science and Technology Beijing, Beijing, China [3]Didi Chuxing, Beijing, China. Correspondence to: Shengbo Eben Li <lishbo@tsinghua.edu.cn>.

*Proceedings of the $40^{th}$ International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. 

time step. However, the hyperparameter tuning is tough in action penalty methods because an excessive penalty will harm performance, otherwise ineffective for action smoothing. Moreover, the action penalty methods complicate RL algorithms due to the close state sampling.

In adversarial disturbance methods, the actor network is trained under adversarial states with disturbance noise. For example, the Smooth Regularized Reinforcement Learning ($SR^2L$) method (Shen et al., 2020) uses projected gradient ascent to find the optimal disturbance noise. The noise maximizes the distance between actions under actual states and adversarial states. Then, the actor network is trained by minimizing both the actor loss and the action distance. However, adversarial disturbance methods complicate RL algorithms because of the additional adversarial policy.

In hierarchical network methods, there are two stages in actor network. For example, the Policy Inertia Controller (PIC) method (Chen et al., 2021) uses one network to output the action distribution and uses the other network to output the policy inertia scalar. In Temporally Abstract Actor-critic (TAAC) method (Yu et al., 2021), one network outputs the deterministic action at the current time step and the other network makes binary choices on whether to use the current action or the last action. However, hierarchical network methods also complicate RL algorithms. For example, a nested policy iteration is needed in PIC to train networks, and a compare-through operator is needed in TAAC to achieve policy evaluation.

In network enhancement methods, the Lipschitz constant of actor network is constrained by Spectral Normalization (SN). SN is proposed from SN-GAN (Miyato et al., 2018), and applied in RL recently (Gogianu et al., 2021; Bjorck et al., 2021; Mehta et al., 2022). For instance, Ryoichi Takase (2020; 2022) applied SN on each layer of MLP, denoted as MLP-SN in our paper, to make the actor network globally Lipschitz continuous. The methods in network enhancement category achieve action smoothing by only modifying neural networks without modifying RL algorithms. Modifying at network level is preferred to algorithm level, because it leaves RL algorithm straightforward and an effective neural network can be used generally in various RL algorithms. However, applying SN on all layers usually leads to severe performance loss, while leaving some layers normal lack a theoretical guarantee of Lipschitz continuity.

Theoretically, the action penalty methods, adversarial disturbance methods and network enhancement methods all aim to reduce the Lipschitz constant of actor network. It is because action fluctuation is affected by the landscape smoothness and noise robustness of actor network, and Lipschitz constant can characterize network's smoothness and robustness, as Section 2.1 discussed. The smaller the Lipschitz constant of actor network, the more robust and smoother the actor network becomes. Therefore, we tend to reduce the Lipschitz constant of actor network under the premise of good enough control performance.

This paper proposes a novel neural network structure, named LipsNet, for action smoothing in RL that has three superiorities compared to previous works. Firstly, it smoothens action by modifying at neural network level rather than algorithm level, not complicating RL algorithms. LipsNet can be used in most existing RL algorithms, such as TRPO (Schulman et al., 2015), DSAC (Duan et al., 2021) and TD3 (Fujimoto et al., 2018). Secondly, LipsNet constrains Lipschitz constant in network-wise, rather than in layer-wise like SN. It is because LipsNet benefits from the Multi-dimensional Gradient Normalization (MGN) we proposed. As illustrated in (Wu et al., 2021), network-wise constraints do not limit layer capacities, which is better than layer-wise constraints. Thirdly, LipsNet does not suffer severe performance loss, because it can be not only globally Lipschitz continuous but also locally Lipschitz continuous, and it has a mechanism for automatically adjustmenting the Lipschitz constant.

Our contributions are four-fold. **(1)** We propose a network-wise Lipschitz constraint method, named Multi-dimensional Gradient Normalization (MGN), which is suitable for neural networks with multi-dimensional input and output. Proof of networks' Lipschitz continuity is provided. **(2)** A novel network structure named LipsNet is proposed for smoothing actions in RL. LipsNet can be either globally or locally Lipschitz continuous, with the ability to adjust its Lipschitz constant automatically. It can be used in most existing RL algorithms without any modification. **(3)** Extensive experiments show that LipsNet has better landscape smoothness and noise robustness than Multilayer Perceptron (MLP). It can smoothen the action without noticeable performance loss. **(4)** We package LipsNet into a Pytorch module and release the code[1] to facilitate the implementation and future research.

## 2. Preliminaries

### 2.1. Lipschitz Continuity

**Definition 2.1** (Global Lipschitz Continuity)**.** Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, if there exists a constant $K > 0$ satisfies

$$\|f(x_1) - f(x_2)\| \leq K\|x_1 - x_2\|, \ \forall x_1, x_2 \in \mathbb{R}^n, \quad (1)$$

then $f$ is a globally $K$-Lipschitz continuous function over $\mathbb{R}^n$. The smallest $K$ is called the global Lipschitz constant.

**Definition 2.2** (Local Lipschitz Continuity)**.** Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, if there exists a constant $K > 0$ satisfies

$$\|f(x_1) - f(x_2)\| \leq K\|x_1 - x_2\|, \ \forall x_1, x_2 \in \mathcal{X},$$

[1] https://github.com/jerry99s/LipsNet

then $f$ is a locally $K$-Lipschitz continuous function over $\mathcal{X}$ where $\mathcal{X} \subset \mathbb{R}^n$. The smallest $K$ is called the local Lipschitz constant over $\mathcal{X}$.

Lipschitz constant $K$ could characterize the landscape smoothness level of a function. Suppose $x_1 = x$ and $x_2 = x+\Delta x$, if $\Delta x \to 0$, then $\frac{\|f(x)-f(x+\Delta x)\|}{\|x-(x+\Delta x)\|} \to \|f'(x)\|$. The Lipschitz constraint means the gradient norm $\|f'(x)\|$ tends to be smaller than $K$. Therefore, $K$ characterizes functions' landscape smoothness.

Lipschitz constant $K$ could characterize the noise robustness level of a function. Suppose $x_1 = x$ and $x_2 = x + \sigma$, if $\sigma$ is the observation noise, then the constraint $\frac{\|f(x)-f(x+\sigma)\|}{\|x-(x+\sigma)\|} \leq K$ means that the ratio of the disturbed output difference to the noised input difference is bounded by $K$. Therefore, $K$ characterizes functions' noise robustness.

### 2.2. Gradient Normalization

In GN-GAN (Wu et al., 2021) and GraN-GAN(Bhaskara et al., 2022), they proposed Gradient Normalization (GN) to constrain the Lipschitz constant of a neural network $f : \mathbb{R}^n \to \mathbb{R}$. GN achieves Lipschitz constraint at network level. GN is superior to its former method, SN, who achieves Lipschitz constraint at layer level.

As GN described, let $f : \mathbb{R}^n \to \mathbb{R}$ be a continuously differential neural network with piecewise linear activations. Then $f_{\mathrm{GN}}$ is a 1-Lipschitz continuous function:

$$f_{\mathrm{GN}}(x) = \frac{f(x)}{\|\nabla_x f(x)\| + \epsilon}, \tag{2}$$

where $\|\nabla_x f(x)\|$ is the 2-norm of gradient vector, $\epsilon$ is a small positive constant.

## 3. Method

### 3.1. Action Fluctuation Ratio

To measure the action fluctuation of RL in quantitation, the action fluctuation ratio needs to be defined. The action fluctuation ratio for discrete action space is already defined in (Chen et al., 2021). To take continuous action space into account, we define the action fluctuation ratio of policy $\pi$ as

$$\xi(\pi) = \mathbb{E}_{\tau\sim\rho_\pi}\left[\frac{1}{T}\sum_{t=1}^{T}||a_t - a_{t-1}||\right], \tag{3}$$

where $\rho_\pi$ is the distribution of state-action trajectory induced by policy $\pi$, and $T$ is the termination time of a trajectory. $||a_t - a_{t-1}||$ means the 2-norm of the difference vector between $a_t$ and $a_{t-1}$, where $a_t$ and $a_{t-1}$ represent the action in the $t$-th time step and the action in the last time step. We use the action fluctuation ratio as an indicator to measure the fluctuation. The smaller $\xi(\pi)$ is, the smoother action policy $\pi$ has.

### 3.2. Multi-dimensional Gradient Normalization

GN is a method to constrain the Lipschitz constant of a neural network $f : \mathbb{R}^n \to \mathbb{R}$, as we describe in Section 2.2. But the neural network $f$ can only have one-dimensional output because GN is used on the discriminator of GAN. However, the actor network in RL has a multi-dimensional output representing multi-dimensional action. So, we propose Multi-dimensional Gradient Normalization (MGN) and prove that MGN can constrain the Lipschitz constant of neural networks with multi-dimensional intput and output.

**Theorem 3.1** (Multi-dimensional Gradient Normalization)**.** *Suppose $f : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differential neural network with piecewise linear activations. Then $f_{\mathrm{MGN}}$ is a globally $K$-Lipschitz continuous neural network:*

$$f_{\mathrm{MGN}}(x) = K \cdot \frac{f(x)}{\|\nabla_x f(x)\| + \epsilon}, \tag{4}$$

*where $K$ is a positive constant, $\|\nabla_x f(x)\|$ is the 2-norm of Jacobian matrix and $\epsilon$ is a small positive constant.*

*Proof.* See Appendix A in the supplementary material. □

Noteworthily, the notation $\nabla_x f(x)$ in MGN is a Jacobian matrix, while $\nabla_x f(x)$ in GN is a gradient vector. The notation $\|\cdot\|$ in MGN is the matrix norm, while $\|\cdot\|$ in GN is the vector norm. Also, the division in equation (4) operates between a vector and a scalar, while the division in equation (2) operates between two scalars.

### 3.3. Automatic Adjustment for Lipschitz Constant

Based on MGN, the Lipschitz constant of actor network can be constrained. However, manually tuning the Lipschitz constant $K$ is disgusting because a small $K$ will harm performance but a large $K$ will not smoothen actions. Additionally, in general situations, no exact information about the environment's Lipschitz property can be acquired before starting training. Therefore, we let Lipschitz constant $K$ as a learnable parameter, and use an L2 regularization term for a lower $K$. The weight $\lambda$ is introduced in loss function to balance the original term and the regularization term. This Lipschitz adjustment mechanism benefits from MGN, because MGN represents network's Lipschitz constant just using one scalar $K$, unlike SN using the product of all layers' Lipschitz constant.

**LipsNet-G.** Then we propose LipsNet, the neural network with MGN and Lipschitz adjustment. Because the network is globally Lipschitz continuous, we name it LipsNet-G. The parameters updating process of LipsNet-G is shown in

Algorithm 1, where network $f(x)$ is an MLP with piecewise linear activations. The learning rate of $f(x)$ and $K$ are $\eta_f$ and $\eta_k$, respectively.

**Algorithm 1** Update Parameters in LipsNet-G

**Input:** input vector $x$, loss function $\mathcal{L}$, parameter $\theta$ in network $f(x)$, Lipschitz parameter $K$.

$f_{\text{MGN}}(x) = K\frac{f(x)}{\|\nabla_x f(x)\|+\epsilon}$ ▷ forward propagation

$\mathcal{L}' = \mathcal{L} + \lambda K^2$

$\theta \leftarrow \theta - \eta_f \nabla_\theta \mathcal{L}'$ ▷ backward propagation

$K \leftarrow K - \eta_k \nabla_K \mathcal{L}'$ ▷ Lipschitz adjustment

**LipsNet-L.** Global Lipschitz continuity around the whole domain may harm the performance severely. Different input $x$ could have different Lipschitz constants around their neighborhoods, called locally Lipschitz continuous. We propose LipsNet-L, which constrains the network in local Lipschitz continuity by introducing an extra network $K(x)$.

The network $K(x)$ is an MLP followed by `Softplus` activation, with parameter $\phi$. The learning rate of $f(x)$ and $K(x)$ are $\eta_f$ and $\eta_k$, respectively. The network $K(x)$ should be updated slower than the network $f(x)$, i.e., $\eta_k < \eta_f$. Because $K(x)$ means the local Lipschitz value around $x$, it should be relatively more stable for correctly updating $f(x)$. The parameters updating process of LipsNet-L is shown in Algorithm 2.

**Algorithm 2** Update Parameters in LipsNet-L

**Input:** input vector $x$, loss function $\mathcal{L}$, parameter $\theta$ in network $f(x)$, parameter $\phi$ in network $K(x)$.

$f_{\text{MGN}}(x) = K(x)\frac{f(x)}{\|\nabla_x f(x)\|+\epsilon}$ ▷ forward propagation

$\mathcal{L}' = \mathcal{L} + \lambda K(x)^2$

$\theta \leftarrow \theta - \eta_f \nabla_\theta \mathcal{L}'$ ▷ backward propagation

$\phi \leftarrow \phi - \eta_k \nabla_\phi \mathcal{L}'$ ▷ Lipschitz adjustment

When LipsNet-G and LipsNet-L are used as actor network in RL, the loss function $\mathcal{L}$ becomes the actor loss of policy improvement. It would not modify the RL algorithms because the backward of regularization term can be completed automatically after forward propagation and the Lipschitz adjustment process can be completed automatically before the next forward propagation. Therefore, these two operations can be packaged inside the network module.

### 3.4. Overall Architecture of LipsNet

The overall architecture of LipsNet-L is shown in Figure 1. LipsNet-L contains two MLP networks. The first MLP represents $f(x)$. It produces two variables $f(x)$ and $\nabla_x f(x)$. The other MLP with `Softplus` represents $K(x)$. It outputs one scalar $K(x)$ where $K(x) > 0$.

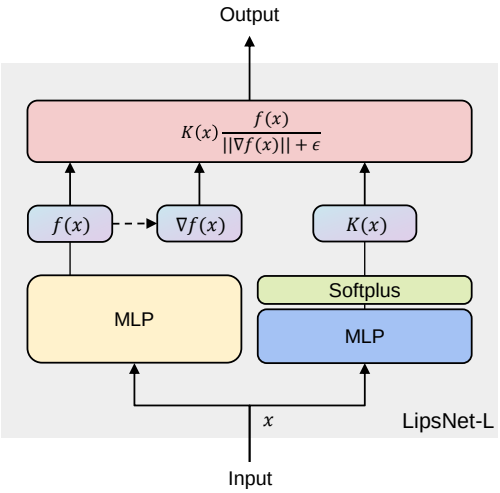



*Figure 1.* **Structure of LipsNet-L**. The input is $x$. The MLP in yellow represents $f(x)$. The MLP in blue with `Softplus` represents $K(x)$. The dashed line means derivative operation. The equation in pink represents MGN.

When the activations in $f(x)$ are all piecewise linear, we can prove that LipsNet is Lipschitz continuous by Theorem 3.1. Piecewise linear activations include ReLU, Leaky ReLU(Maas et al., 2013), PReLU(He et al., 2015), Maxout (Goodfellow et al., 2013), PWLU(Zhou et al., 2021) et al. Furthermore, we found that normal activations, such as `tanh`, are also suitable in $f(x)$ despite lacking a theoretical guarantee of Lipschitz continuity and could achieve action smoothing aim. We provide an experiment to show the finding. More details can be found in Appendix B. For $K(x)$, there is theoretically no limitation for its activation's type and no limitation for $K(x)$ to be an MLP structure. In the backward propagation, the gradients go through both $f(x)$ and $\nabla f(x)$ for updating the network $f(x)$.

For RL tasks, the environment usually has an action bound. In this situation, `tanh` can be added after LipsNet then followed by scaling up. The network activated by `tanh` before scaling up is still $K$-Lipschitz, as Theorem 3.2 illustrated.

**Theorem 3.2.** *Suppose $g(x) : \mathbb{R}^n \to \mathbb{R}^m$ is a neural network composed of LipsNet followed by `tanh`:*

$$g(x) = \tanh(f_{\text{MGN}}(x)),$$

*where `tanh` operates in element-wise and $f_{\text{MGN}}(x)$ is $K$-Lipschitz continuous. Then $g(x)$ is still $K$-Lipschitz continuous.*

*Proof.* See Appendix C in the supplementary material. □

### 3.5. Simple Implementation for Practitioners

We package LipsNet as a module in PyTorch (Paszke et al., 2019). As shown below, practitioners can use LipsNet just like an MLP network. The code is available in [2].

```
net = LipsNet()
out = net(input)
...
loss.backward()
```

## 4. Experiments

### 4.1. Double Integrator

In this environment, a particle lies on an axis without resistance. We can change its acceleration at every time step by applying a force parallel to the axis. The system dynamic is

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} a,$$

where $x_1$ is the position of particle, $x_2$ is the velocity of particle, $a$ is the acceleration produced by the applied force.

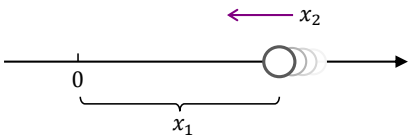



*Figure 3.* **Double integrator environment**. A particle lies on an axis. $x_1$ and $x_2$ are the position and velocity, respectively.

The reward in each time step is

$$r = -2x_1^2 - x_2^2 - a^2.$$

The reward urges the particle to be stable at the origin with zero velocity. The observation noise in each dimension is distributed in $U(-0.2, 0.2)$.

We use the model-based RL method, Infinite-time Approximate Dynamic Programming (INFADP) (Li, 2023), to train in this environment. INFADP with MLP is chosen as the baseline. We compare the action fluctuation between baseline and INFADP with LipsNet. The results are shown in Figure 2. In Figure 2(a), we simulate 30 times starting from the same initial state. The solid line is the mean of actions. The shadow is the standard deviation of actions, which implies action fluctuation amplitude. Obviously, the action fluctuation amplitude of LipsNet is smaller than MLP. Figure 2(c) shows the output of $K(x)$ in LipsNet-L. $K(x)$ is large around state $(1, 1)$ and $(-1, -1)$ , because the particle has positive position with positive speed in $(1, 1)$ and negative position with negative speed in $(-1, -1)$. Around these states, the particle is departing from the steady state $(0, 0)$. Therefore, the local Lipschitz constant could be large. More details and hyperparameters are shown in Appendix E. Comparison between LipsNet and previous works is shown in Appendix J.

[2] https://github.com/jerry99s/LipsNet

### 4.2. Vehicle Trajectory Tracking

Vehicle trajectory tracking is an important task in autonomous driving (Guan et al., 2022; Mu et al., 2020). In this environment, the bicycle model of lateral vehicle dynamic (Ge et al., 2021; Rajamani, 2011), a widely used model, is employed to simulate the vehicle motion. The vehicle aims to track a given reference trajectory, meanwhile, satisfies certain constraint. The longitudinal speed keeps in $5\ \mathrm{m/s}$ and the only available control input is the steering angle. The vehicle model is depicted in Figure 4. The states and action are listed in Table 1.

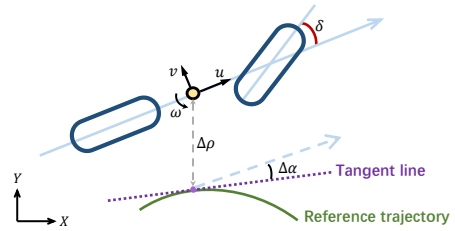



*Figure 4.* **Vehicle dynamic model**.

*Table 1.* **Variables in vehicle trajectory tracking environment**.

| Variable | | Description | Unit |
|---|---|---|---|
| State | $\Delta\rho$ | trajectory offset | m |
| | $\Delta\alpha$ | heading angle error | rad |
| | $u$ | longitudinal speed | m/s |
| | $v$ | lateral speed | m/s |
| | $\omega$ | yaw rate | rad/s$^2$ |
| Action | $\delta$ | front wheel steering angle | rad |

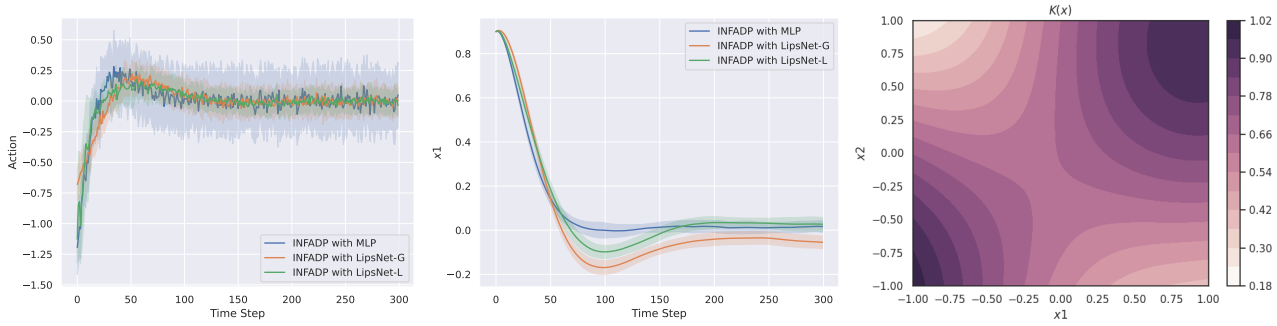
We provide two kinds of reference trajectories, i.e., sine curve and double-line curve. Those two scenarios are shown in Figure 5. The sine curve scenario simulates the road with consecutive curves. The double-line scenario urges the vehicle to bypass obstacles.

The reward signal is

$$r = -0.01\left(4 \cdot \Delta\rho^2 + 2 \cdot \Delta\alpha^2 + v^2 + \omega^2 + \delta^2\right).$$

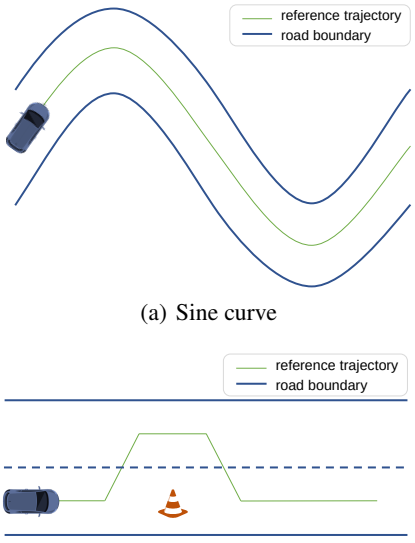
The constraint requires

$$|\Delta\rho| < 0.1\ .$$




(a) Action fluctuation amplitude (b) Particle position (c) Landscape of $K(x)$

*Figure 2.* **Results in double integrator environment**. The solid line and shadow are the mean and standard deviation, respectively. (a) The action fluctuation amplitude of LipsNet is smaller than MLP. (b) Particle positions all converge to around 0. (c) The network $K(x)$ in LipsNet-L represents the local Lipschitz constant around $x$.




(a) Sine curve

(b) Double-line curve

*Figure 5.* **Reference trajectory type**.

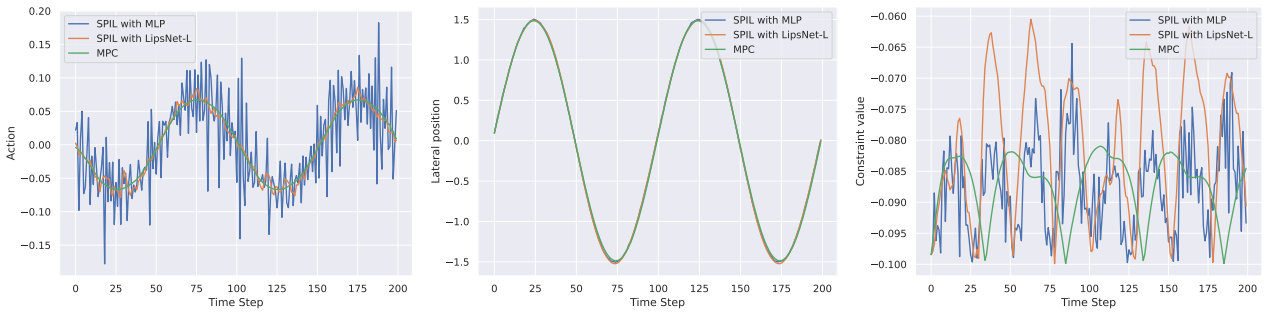
Constrained RL tasks should be solved by safe RL algorithms (Garcıa & Fernández, 2015; Gu et al., 2022; Yu et al., 2022; Ma et al., 2022). To solve the vehicle trajectory tracking task, we use the Separated Proportional-integral Lagrangian (SPIL) algorithm (Peng et al., 2021). Hyperparameters are listed in Appendix F. The Model Predictive Control (MPC) (Camacho & Alba, 2013) method is also used for comparison.

We compare the results of SPIL with LipsNet-L, SPIL with MLP and MPC. This section will only involve LipsNet-L excluding LipsNet-G for simplification, and more details including both LipsNet-L and LipsNet-G can be found in Appendix F. The network weights at 25000 training iterations are used for both LipsNet-L and MLP to ensure a fair comparison. The prediction step in MPC is 10. The observation noise in each dimension for SPIL is distributed in $U(-0.002, 0.002)$. There is no observation noise for MPC to obtain a nearly optimal action trajectory. Figure 6 and Figure 7 show the results when tracking sine and double-line curves, respectively. In both scenarios, the action fluctuation amplitude of LipsNet-L is smaller than MLP's, as shown in Figure 6(a) and Figure 7(a). LipsNet-L also exhibits good tracking performance and constraint satisfaction, as shown in Figure 6(b,c) and Figure 7(b,c).

To further evaluate LipsNet, we set different observation noises and compare the action fluctuation ratio. The results are summarized in Table 2. In all noise levels, LipsNet-L has smaller action fluctuation ratios than MLP. For example, when the trajectory is sine and the noise level is $1 \cdot 10^{-2}$, meaning the observation noise in each dimension is distributed in $U(-10^{-2}, 10^{-2})$, the action fluctuation ratio of LipsNet-L is only $9.8\%$ of MLP's. Moreover, we plot action fluctuation ratio as a function of noise to analyze the increasing trend of fluctuation. As shown in Figure 8, when the observation noise increases, the action fluctuation ratio of MLP grows much faster than that of LipsNet-L.

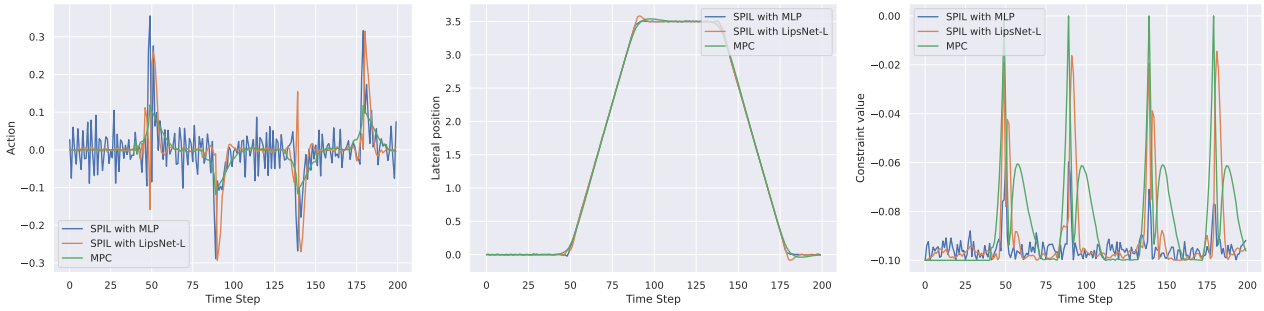
*Table 2.* **Action fluctuation ratio** comparison on vehicle trajectory tracking. The observation noise in each dimension is distributed in $U(-\sigma, \sigma)$.

| Env | | Method | |
|---|---|---|---|
| Trajectory | Noise $\sigma$ | SPIL (MLP) | SPIL (LipsNet-L) |
| sine curve | $1 \cdot 10^{-4}$ | 0.005 $\pm$ 0.0002 | **0.004** $\pm$ 0.0001 |
| | $1 \cdot 10^{-3}$ | 0.027 $\pm$ 0.0016 | **0.005** $\pm$ 0.0001 |
| | $1 \cdot 10^{-2}$ | 0.224 $\pm$ 0.0115 | **0.022** $\pm$ 0.0009 |
| double-line | $1 \cdot 10^{-4}$ | 0.024 $\pm$ 0.0003 | **0.019** $\pm$ 0.0001 |
| | $1 \cdot 10^{-3}$ | 0.043 $\pm$ 0.0018 | **0.020** $\pm$ 0.0001 |
| | $1 \cdot 10^{-2}$ | 0.227 $\pm$ 0.0198 | **0.034** $\pm$ 0.0013 |




(a) Action curve (b) Lateral position (c) Constraint value

*Figure 6.* **Results on vehicle trajectory tracking environment**. The reference trajectory is a **sine curve**. (a) The action fluctuation amplitude of LipsNet-L is smaller than MLP. (b) The tracking performances are good for both LipsNet-L and MLP. (c) A positive constraint value implies the severity of constraint violation. There is no constraint violation during the simulation.




(a) Action curve (b) Lateral position (c) Constraint value

*Figure 7.* **Results on vehicle trajectory tracking environment**. The reference trajectory is a **double-line curve**. (a) The action fluctuation amplitude of LipsNet-L is smaller than MLP. (b) The tracking performances are good for both LipsNet-L and MLP. (c) A positive constraint value implies the severity of constraint violation. There is no constraint violation during the simulation.

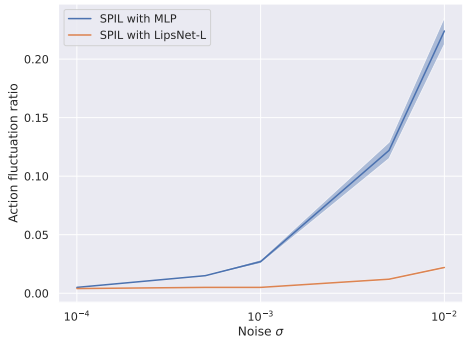
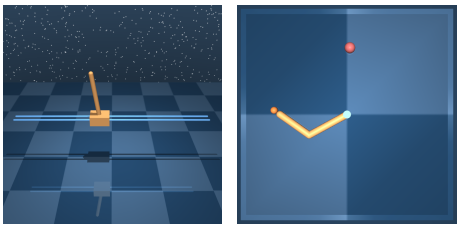



*Figure 8.* **Increasing trend of action fluctuation ratio**. The reference trajectory is a sine curve. X-axis is the radius of uniformly distributed noise applied in each observation dimension. Y-axis is the mean of action fluctuation ratio and its standard deviation.

### 4.3. DeepMind Control Suit

The DeepMind Control Suite (DMControl) (Tassa et al., 2018) contains a set of well-designed continuous control tasks with standardized structures and interpretable rewards. Currently, it is one of the most recognized benchmark environments for RL and continuous control (Mu et al., 2022). This paper selects four environments in DMControl, i.e., Cartpole, Reacher, Cheetah and Walker. The environments are shown in Figure 9, and more details can be found in Appendix G.

We use the model-free RL algorithm, Twin Delayed Deep Deterministic policy gradient (TD3) (Fujimoto et al., 2018), to train on DMControl. The hyperparameters of TD3 are the same in all environments, while only the weight $\lambda$ varies. The hyperparameters are listed in Appendix H. Then we make a comparison between TD3 with LipsNet and TD3 with MLP. When there is no observation noise, the total average return and action fluctuation ratio are listed in Table 3 and 4, respectively. The data shows the mean and standard deviation simulating 5 rounds using the best policies.



(a) Cartpole (b) Reacher

(c) Cheetah (d) Walker

*Figure 9.* **DeepMind Control Suit Benchmarks**.

*Table 3.* **Total average return** on DMControl without observation noise.

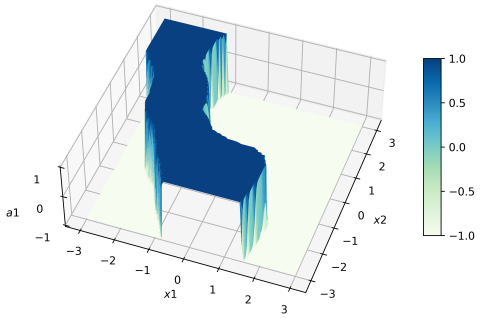
| Method | Cartpole | Reacher | Cheetah | Walker |
|---|---|---|---|---|
| TD3 (MLP) | 805 $\pm$ 0.8 | 981 $\pm$ 10 | 816 $\pm$ 30 | 926 $\pm$ 12 |
| TD3 (LipsNet-L) | **831** $\pm$ 0.9 | **983** $\pm$ 10 | **822** $\pm$ 4 | 945 $\pm$ 13 |
| TD3 (LipsNet-G) | 691 $\pm$ 1.0 | 979 $\pm$ 11 | 702 $\pm$ 10 | **956** $\pm$ 20 |

*Table 4.* **Action fluctuation ratio** on DMControl without observation noise. Note: DMControl environments have 1000 time steps. In Cartpole and Reacher, we only consider the first 500 time steps, otherwise the action fluctuation ratio will be 0.00 for LipsNet-L.

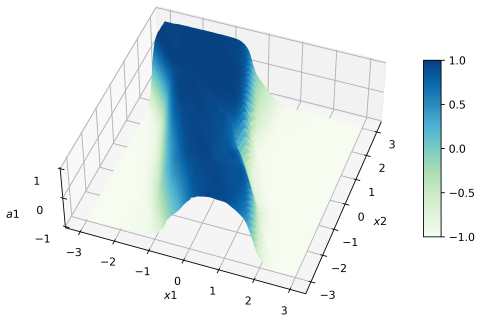
| Method | Cartpole | Reacher | Cheetah | Walker |
|---|---|---|---|---|
| TD3 (MLP) | 0.04 $\pm$ 0.00 | 2.07 $\pm$ 0.60 | 1.08 $\pm$ 0.02 | 1.89 $\pm$ 0.02 |
| TD3 (LipsNet-L) | **0.01** $\pm$ 0.00 | **0.01** $\pm$ 0.00 | 0.94 $\pm$ 0.01 | **0.93** $\pm$ 0.01 |
| TD3 (LipsNet-G) | 0.08 $\pm$ 0.00 | 0.13 $\pm$ 0.24 | **0.92** $\pm$ 0.01 | 1.25 $\pm$ 0.02 |

As Table 3 shown, LipsNet-G may severely damage the control performance, but LipsNet-L has almost the same performance as MLP. It is because LipsNet-L benefits from the local Lipschitz continuity. As Table 4 shown, LipsNet-L has the lowest action fluctuation ratio in most environments. This superiority is because the landscape smoothness of LipsNet-L is better than that of MLP. We visualize the neural network landscape in Figure 10.

When there are observation noises, the total average return and action fluctuation ratio are listed in Table 5 and Table 6, respectively. The histograms based on Table 5 and Table 6 are shown in Figure 11. The noise level in each environment is illustrated in Appendix H.




(a) Landscape of MLP




(b) Landscape of LipsNet-L

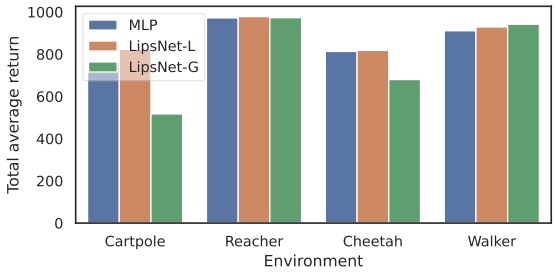
*Figure 10.* **Neural network landscape**. Let $x_1$, $x_2$ be the first two observations in Reacher environment. Let $a_1$ be the first component of action. We vary $x_1$, $x_2$ and fix other observations. $a_1$ is calculated by actor network based on observation $[x_1, x_2, 0.1, 0.1, 0, 0]^\top$. The figures show that the landscape smoothness of LipsNet-L is much better than that of MLP.

*Table 5.* **Total average return** on DMControl with observation noise.

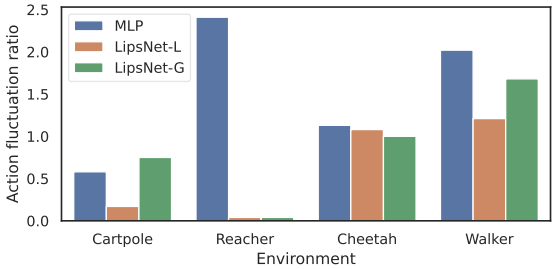
| Method | Cartpole | Reacher | Cheetah | Walker |
|---|---|---|---|---|
| TD3 (MLP) | 763 $\pm$ 9 | 972 $\pm$ 25 | 813 $\pm$ 29 | 911 $\pm$ 26 |
| TD3 (LipsNet-L) | **823** $\pm$ 6 | **978** $\pm$ 17 | **818** $\pm$ 11 | 929 $\pm$ 11 |
| TD3 (LipsNet-G) | 517 $\pm$ 41 | 973 $\pm$ 18 | 680 $\pm$ 7 | **942** $\pm$ 15 |

*Table 6.* **Action fluctuation ratio** on DMControl with observation noise.

| Method | Cartpole | Reacher | Cheetah | Walker |
|---|---|---|---|---|
| TD3 (MLP) | 0.58 $\pm$ 0.03 | 2.41 $\pm$ 0.28 | 1.13 $\pm$ 0.02 | 2.02 $\pm$ 0.03 |
| TD3 (LipsNet-L) | **0.17** $\pm$ 0.01 | **0.04** $\pm$ 0.03 | 1.08 $\pm$ 0.01 | **1.21** $\pm$ 0.01 |
| TD3 (LipsNet-G) | 0.75 $\pm$ 0.09 | 0.04 $\pm$ 0.00 | **1.00** $\pm$ 0.01 | 1.68 $\pm$ 0.01 |




(a) Total average return




(b) Action fluctuation ratio

*Figure 11.* **Effect of LipsNet**. The figures visualize the mean values in Table 5 and Table 6.

As illustrated in Table 6, the action fluctuation ratio of LipsNet-L in Reacher environment is only 1.6% of MLP's. As Figure 11 shown, LipsNet-L gains significant action fluctuation ratio decreases without noticeable performance loss in most environments. Those results imply that the noise robustness of LipsNet-L is much better than MLP.

### 4.4. Gym Humanoid

To illustrate the effectiveness of LipsNet on high-dimensional tasks, we implement it on the Gym humanoid environment (Tassa et al., 2012). The environment has 376 observations and 17 actions, powered by OpenAI Gym (Brockman et al., 2016) and MuJoCo (Todorov et al., 2012). A 3D bipedal robot is designed in the environment to simulate a human, as shown in Figure 12. The goal of the environment is to walk forward as fast as possible without falling over. We compare the performance when choosing MLP or LipsNet-L as TD3's actor network. Observation noise is not added in this environment. The results are listed in Table 7.

*Table 7.* **Results on Gym humanoid environment**.

| Method | Total average return | Action fluctuation ratio |
|---|---|---|
| TD3 (MLP) | 4968 $\pm$ 46 | 0.83 $\pm$ 0.02 |
| TD3 (LipsNet-L) | **5266** $\pm$ 15 | **0.66** $\pm$ 0.01 |

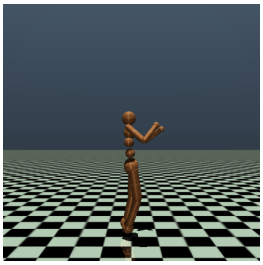


*Figure 12.* **Gym humanoid environment**.

The result implies that LipsNet-L achieves 20% reduction of action fluctuation ratio with the same level of return compared to MLP. The result successfully demonstrates the effectiveness of LipsNet on high-dimensional tasks.

## 5. Discussion

**Computational Efficiency.** An extra analysis of computational efficiency is provided in Appendix D, including the forward and backward time comparisons. Overall, despite LipsNet having a slightly longer computation time than MLP, it is still suitable in real-time applications.

**Relationship with Robust RL.** The relationship between LipsNet and robust RL is described in detail in Appendix K. Overall, although Lipschitz continuity is a particular form of robustness, the aims of LipsNet and robust RL differ greatly. LipsNet can smooth the action even in the environments without noise, which is distinct from the robust RL settings, thanks to the landscape smoothness of LipsNet.

## 6. Conclusion

In this paper, we propose a novel network structure LipsNet to address action fluctuation problem in RL control tasks. LipsNet is either globally or locally Lipschitz continuous, which benefits from MGN method we proposed. LipsNet can automatically adjust Lipschitz constant to obtain smooth action with satisfactory performance. It can be used as actor network in most existing RL algorithms. Various experiments show that LipsNet has excellent landscape smoothness and noise robustness, resulting in smooth action in RL control tasks. We hope that our work could inspire future research in control network field, and could contribute to the real-world RL applications.

## Acknowledgements

This study is supported by National Key R&D Program of China with 2022YFB2502901, Tsinghua University Initiative Scientific Research Program, and Tsinghua University-Didi Joint Research Center for Future Mobility.

## References


Bhaskara, V. S., Aumentado-Armstrong, T., Jepson, A. D., and Levinshtein, A. Gran-gan: Piecewise gradient normalization for generative adversarial networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 3821–3830, 2022.

Bjorck, N., Gomes, C. P., and Weinberger, K. Q. Towards deeper deep reinforcement learning with spectral normalization. *Advances in Neural Information Processing Systems*, 34:8242–8255, 2021.

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

Camacho, E. F. and Alba, C. B. *Model predictive control*. Springer science & business media, 2013.

Chen, C., Tang, H., Hao, J., Liu, W., and Meng, Z. Addressing action oscillations through learning policy inertia. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7020–7027, 2021.

Duan, J., Guan, Y., Li, S. E., Ren, Y., Sun, Q., and Cheng, B. Distributional soft actor-critic: Off-policy reinforcement learning for addressing value estimation errors. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

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

Ge, Q., Sun, Q., Li, S. E., Zheng, S., Wu, W., and Chen, X. Numerically stable dynamic bicycle model for discrete-time control. In *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pp. 128–134. IEEE, 2021.

Gogianu, F., Berariu, T., Rosca, M. C., Clopath, C., Busoniu, L., and Pascanu, R. Spectral normalisation for deep reinforcement learning: an optimisation perspective. In *International Conference on Machine Learning*, pp. 3734–3744. PMLR, 2021.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. In *International Conference on Machine Learning*, pp. 1319–1327. PMLR, 2013.

Gu, S., Yang, L., Du, Y., Chen, G., Walter, F., Wang, J., Yang, Y., and Knoll, A. A review of safe reinforcement learning: Methods, theory and applications. *arXiv preprint arXiv:2205.10330*, 2022.

Guan, Y., Li, S. E., Duan, J., Li, J., Ren, Y., Sun, Q., and Cheng, B. Direct and indirect reinforcement learning. *International Journal of Intelligent Systems*, 36(8):4439–4467, 2021.

Guan, Y., Ren, Y., Sun, Q., Li, S. E., Ma, H., Duan, J., Dai, Y., and Cheng, B. Integrated decision and control: toward interpretable and computationally efficient driving intelligence. *IEEE Transactions on Cybernetics*, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Kidger, P. and Lyons, T. Universal approximation with deep narrow networks. In *Conference on Learning Theory*, pp. 2306–2327. PMLR, 2020.

Kim, H., Park, J., Lee, C., and Kim, J.-J. Improving accuracy of binary neural networks using unbalanced activation distribution. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7862–7871, 2021.

Kobayashi, T. L2c2: Locally lipschitz continuous constraint towards stable and smooth reinforcement learning. *arXiv preprint arXiv:2202.07152*, 2022.

Li, S. E. *Reinforcement learning for sequential decision and optimal control*. Springer Verlag, Singapore, 2023.

Ma, H., Liu, C., Li, S. E., Zheng, S., and Chen, J. Joint synthesis of safety certificate and safe control policy using constrained reinforcement learning. In *Learning for Dynamics and Control Conference*, pp. 97–109. PMLR, 2022.

Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, pp. 3. Atlanta, Georgia, USA, 2013.

Mehta, K., Mahajan, A., and Kumar, P. Effects of spectral normalization in multi-agent reinforcement learning. *arXiv preprint arXiv:2212.05331*, 2022.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Moos, J., Hansel, K., Abdulsamad, H., Stark, S., Clever, D., and Peters, J. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1):276–315, 2022.

Mu, Y., Peng, B., Gu, Z., Li, S. E., Liu, C., Nie, B., Zheng, J., and Zhang, B. Mixed reinforcement learning for efficient policy optimization in stochastic environments. In *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, pp. 1212–1219. IEEE, 2020.

Mu, Y. M., Chen, S., Ding, M., Chen, J., Chen, R., and Luo, P. Ctrlformer: Learning transferable state representation for visual control via transformer. In *International Conference on Machine Learning*, pp. 16043–16061. PMLR, 2022.

Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K. Regularizing action policies for smooth control with reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1810–1816. IEEE, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

Peng, B., Mu, Y., Duan, J., Guan, Y., Li, S. E., and Chen, J. Separated proportional-integral lagrangian for chance constrained reinforcement learning. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pp. 193–199. IEEE, 2021.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pp. 2817–2826. PMLR, 2017.

Rajamani, R. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897. PMLR, 2015.

Shen, Q., Li, Y., Jiang, H., Wang, Z., and Zhao, T. Deep reinforcement learning with robust and smooth policy. In *International Conference on Machine Learning*, pp. 8707–8718. PMLR, 2020.

Shi, G., Shi, X., O'Connell, M., Yu, R., Azizzadenesheli, K., Anandkumar, A., Yue, Y., and Chung, S.-J. Neural lander: Stable drone landing control using learned dynamics. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 9784–9790. IEEE, 2019.

Takase, R., Yoshikawa, N., Mariyama, T., and Tsuchiya, T. Stability-certified reinforcement learning via spectral normalization. *arXiv preprint arXiv:2012.13744*, 2020.

Takase, R., Yoshikawa, N., Mariyama, T., and Tsuchiya, T. Stability-certified reinforcement learning control via spectral normalization. *Machine Learning with Applications*, 10:100409, 2022.

Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4906–4913. IEEE, 2012.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tessler, C., Efroni, Y., and Mannor, S. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pp. 6215–6224. PMLR, 2019.

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. doi: 10.1109/IROS.2012.6386109.

Wasala, A., Byrne, D., Miesbauer, P., O'Hanlon, J., Heraty, P., and Barry, P. Trajectory based lateral control: A reinforcement learning case study. *Engineering Applications of Artificial Intelligence*, 94:103799, 2020.

Wu, Y.-L., Shuai, H.-H., Tam, Z.-R., and Chiu, H.-Y. Gradient normalization for generative adversarial networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6373–6382, 2021.

Yu, D., Ma, H., Li, S. E., and Chen, J. Reachability constrained reinforcement learning. In *International Conference on Machine Learning*, pp. 25636–25655. PMLR, 2022.

Yu, H., Xu, W., and Zhang, H. Taac: Temporally abstract actor-critic for continuous control. *Advances in Neural Information Processing Systems*, 34:29021–29033, 2021.

Zhang, H., Chen, H., Xiao, C., Li, B., Liu, M., Boning, D., and Hsieh, C.-J. Robust deep reinforcement learning

against adversarial perturbations on state observations. *Advances in Neural Information Processing Systems*, 33: 21024–21037, 2020.

Zhao, Z., Zuo, S., Zhao, T., and Zhao, Y. Adversarially regularized policy learning guided by trajectory optimization. In *Learning for Dynamics and Control Conference*, pp. 844–857. PMLR, 2022.

Zhou, Y., Zhu, Z., and Zhong, Z. Learning specialized activation functions with the piecewise linear unit. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12095–12104, 2021.

# Appendix

## A. Theoretical Results

**Lemma 3.1** *Suppose $f : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differential function. The K-Lipschitz continuity constraint (1) is equivalent to*

$$\|\nabla_x f(x)\| \leq K, \ \forall x.$$

*Proof.* Firstly, we prove the sufficient condition.
($\Rightarrow$) From the definition of Lipschitz continuity constraint (1), we know

$$\|f(x_1) - f(x_2)\| \leq K\|x_1 - x_2\|, \ \forall x_1, x_2 \in \mathbb{R}^n.$$

Let $g(t) = f(x + t \cdot v)$ where $t \in \mathbb{R}$ and $v \in \mathbb{R}^n$, then $g'(t) = \nabla f(x + t \cdot v)v$. From the Newton-Leibniz formula, we know

$$g(\eta) - g(0) = \int_0^\eta g'(t)dt,$$

which means

$$f(x + \eta \cdot v) - f(x) = \int_0^\eta \nabla f(x + t \cdot v)v \cdot dt.$$

Take the 2-norm on both sides, get

$$\begin{aligned}\left\| \int_0^\eta \nabla f(x + t \cdot v)dt \cdot v \right\| &= \|f(x + \eta \cdot v) - f(x)\| \\ &\leq \eta K \|v\|.\end{aligned}$$

Divide $\eta$ on both sides and let $\eta \to 0^+$, get

$$\|\nabla f(x) \cdot v\| \leq K \|v\|.$$

From the definition of matrix norm, we know

$$\|\nabla f(x)\| = \max_{v \neq 0} \frac{\|\nabla f(x) \cdot v\|}{\|v\|} \leq K.$$

Then, we prove the necessary condition.
($\Leftarrow$) From the description of Lemma 3.1, suppose we know

$$\|\nabla_x f(x)\| \leq K.$$

Let $g(t) = f(x + t(y - x))$ where $t \in \mathbb{R}$ and $y \in \mathbb{R}^n$, then $g'(t) = \nabla f(x + t(y - x)) \cdot (y - x)$. From the Newton-Leibniz formula, we know

$$\begin{aligned}f(y) - f(x) &= \int_0^1 \nabla f(x + t(y - x)) \cdot (y - x)dt \\ &= \left( \int_0^1 \nabla f(x + t(y - x))dt \right) (y - x).\end{aligned}$$

Take the 2-norm on both sides, get

$$\begin{aligned}\|f(y) - f(x)\| &= \left\| \left( \int_0^1 \nabla f(x + t(y - x))dt \right) (y - x) \right\| \\ &\leq \left\| \int_0^1 \nabla f(x + t(y - x))dt \right\| \|y - x\| \\ &\leq \left( \int_0^1 \|\nabla f(x + t(y - x))\| \, dt \right) \|y - x\| \\ &\leq \left( \int_0^1 K \cdot dt \right) \|y - x\| \\ &= K \|y - x\|.\end{aligned}$$

Now, the sufficient condition and necessary condition are both proved. □

Noteworthily, Wu et al. (2021) have given a lemma similar to Lemma 3.1 where $f$ is a function mapping from $\mathbb{R}^n$ to $\mathbb{R}$. In this paper, we expand it to Lemma 3.1 where $f$ is a function mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$.

**Theorem 3.1** (Multi-dimensional Gradient Normalization) *Suppose $f : \mathbb{R}^n \to \mathbb{R}^m$ is a continuously differential neural network with piecewise linear activations. Then $f_{\text{MGN}}$ is a globally $K$-Lipschitz continuous neural network:*

$$f_{\text{MGN}}(x) = K \cdot \frac{f(x)}{\|\nabla_x f(x)\| + \epsilon},$$

*where $K$ is a positive constant, $\|\nabla_x f(x)\|$ is the 2-norm of Jacobian matrix and $\epsilon$ is a small positive constant.*

*Proof.* By definition, the 2-norm of Jacobian matrix of $f_{\text{MGN}}(x)$ according to $x$ is

$$\begin{aligned}\|\nabla_x f_{\text{MGN}}(x)\| &= K \cdot \left\| \nabla \frac{f}{\|\nabla f\| + \epsilon} \right\| \\ &= K \cdot \left\| \frac{\nabla f(\|\nabla f\| + \epsilon) - f\left(\nabla(\|\nabla f\| + \epsilon)\right)^\top}{\left(\|\nabla f\| + \epsilon\right)^2} \right\|.\end{aligned} \tag{5}$$

Because there are only piecewise linear activation functions in $f$, the Jacobian matrix $\nabla f$ is a constant matrix. Then, the norm of it, $\|\nabla f\|$, is a constant value. So, $\nabla(\|\nabla f\| + \epsilon)$ is a zero vector.

Finally, the equation (5) can be simplified to

$$\|\nabla_x f_{\text{MGN}}(x)\| = K \cdot \left\| \frac{\nabla f}{\|\nabla f\| + \epsilon} \right\| \leq K.$$

By Lemma 3.1, we complete the proof of Theorem 3.1. □

## B. Limitation of Activation Function

In Theorem 3.1, we assume that the activations in network $f(x)$ should be piecewise linear. This assumption makes $\nabla\|\nabla f\|$ become a zero vector, which enables the proof of Theorem 3.1 in Appendix A. Piecewise linear activations include ReLU, Leaky ReLU(Maas et al., 2013), PReLU(He et al., 2015), Maxout (Goodfellow et al., 2013), Hard Tanh (Kim et al., 2021), PWLU(Zhou et al., 2021) et al. However, we experimentally found that LipsNet can also reduce action fluctuation when $f(x)$ has non-piecewise-linear activations, such as `tanh`.

To illustrate the behavior when using non-piecewise-linear activations, we compare MLP with ReLU activation, LipsNet-L with ReLU activation and LipsNet-L with `tanh` activation on the double integrator environment. The environment setting and hyperparameters are consistent with those outlined in Section 4.1. Figure 13 draws the action fluctuation amplitudes. The result demonstrates that LipsNet-L with `tanh`-activated $f(x)$ performs similarly to LipsNet-L with ReLU-activated $f(x)$. They both have lower action fluctuation amplitude than MLP.

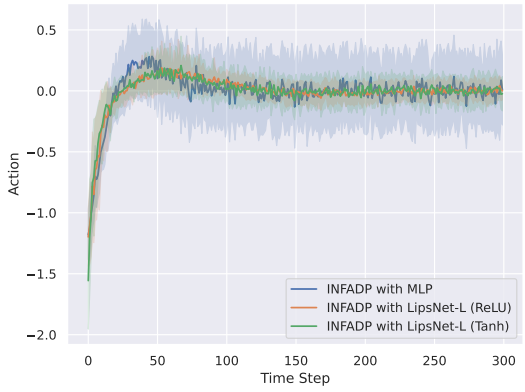



*Figure 13.* Action fluctuation amplitude on double integrator environment. Parentheses show the activation functions used in network $f(x)$.

## C. LipsNet Followed by `tanh`

For RL tasks, the environment usually has an action bound. In this situation, `tanh` can be added after LipsNet then followed by scaling up. The network activated by `tanh` before scaling up is still $K$-Lipschitz continuous because `tanh` is a 1-Lipschitz continuous function. The proof is shown in the following Theorem.

**Theorem 3.2** *Suppose $g(x) : \mathbb{R}^n \to \mathbb{R}^m$ is a neural network composed of LipsNet followed by `tanh`:*

$$g(x) = \tanh(f_{\text{MGN}}(x)),$$

*where `tanh` operates in element-wise and $f_{\text{MGN}}(x)$ is $K$-Lipschitz continuous. Then $g(x)$ is still $K$-Lipschitz continuous.*

*Proof.* By definition, the 2-norm of Jacobian matrix of $g(x)$ according to $x$ is

$$\begin{aligned}\|\nabla_x\, g(x)\| &= \left\|\nabla_{f_{\text{MGN}}(x)}\, g(x) \cdot \nabla_x\, f_{\text{MGN}}(x)\right\| \\ &\le \left\|\nabla_{f_{\text{MGN}}(x)}\, g(x)\right\| \left\|\nabla_x\, f_{\text{MGN}}(x)\right\|.\end{aligned}$$

Because `tanh` operates in element-wise and $0 < \nabla \tanh \le 1$, $\nabla_{f_{\text{MGN}}(x)}\, g(x)$ is a diagonal matrix whose elements are all positive and smaller than 1. It implies that the norm $\left\|\nabla_{f_{\text{MGN}}(x)}\, g(x)\right\| \le 1$. Therefore,

$$\begin{aligned}\|\nabla_x\, g(x)\| &\le 1 \cdot \|\nabla_x\, f_{\text{MGN}}(x)\| \\ &\le K.\end{aligned}$$

By Lemma 3.1, we complete the proof of Theorem 3.2. □

For example, the action bound in a single-action environment is $[-5, 5]$. The output of LipsNet followed by `tanh` lies in $[-1, 1]$. Then LipsNet followed by `tanh` is still $K$-Lipschitz continuous. The output can be scaled up from $[-1, 1]$ to $[-5, 5]$, by multiplying 5. The whole network, composed of LipsNet, `tanh` and scaling up, will be $5K$-Lipschitz continuous.

## D. Computational Efficiency Analysis

We provide an extra analysis of computational efficiency, including the forward and backward time comparisons. The running platform is AMD Ryzen Threadripper 3960X 24-Core Processor. The number of power iterations in MLP-SN is set as 1, whose time usage is included in the backward propagation stage. The results are summarized in Table 8.

*Table 8.* Computation time comparison.

| Settings | | Neural network | | |
|---|---|---|---|---|
| Propagation | Batch size | MLP | MLP-SN | LipsNet-L |
| forward | 1 | 0.10 ms | 0.11 ms | 0.75 ms |
| | 100 | 0.11 ms | 0.12 ms | 1.41 ms |
| backward | 1 | 0.17 ms | 0.76 ms | 0.45 ms |
| | 100 | 0.28 ms | 0.89 ms | 0.73 ms |

It implies that the computation time of LipsNet is slightly higher than MLP's. The computation bottleneck in LipsNet is the calculation of the Jacobian matrix $\nabla f$ and its backward propagation. Fortunately, the 1-batch forward time of LipsNet is a small value within 1 ms, which is still suitable in real-time applications.

## E. Implementation Details on Double Integrator Environment

Double integrator is a classical control task with linear dynamic and quadratic cost. The environment used in this paper is a particle-moving environment, described in Section 4.1. We use INFADP, a model-based RL algorithm, to train in this environment. The network $K(x)$ needs a relatively large output at the beginning of training for sufficient explorations. Therefore, we add the expected initial Lipschitz constant $K_{\text{init}}$ to the bias of the last linear layer before `Softplus` in network $K(x)$. The hyperparameters of INFADP are listed in Table 9.

*Table 9.* Hyperparameters of INFADP.

| Parameter | Setting |
|---|---|
| Replay buffer capacity | 100000 |
| Buffer warm-up size | 1000 |
| Batch size | 64 |
| Discount $\gamma$ | 0.99 |
| Target network soft-update rate $\tau$ | 0.2 |
| Initial random interaction steps | 0 |
| Interaction steps per iteration | 8 |
| Network update times per iteration | 1 |
| Prediction step | 1 |
| Action bound | $[-5, 5]$ |
| Exploration noise std. deviation | 0 |
| Hidden layers in $f(x)$ | $[64, 64]$ |
| Activations in $f(x)$ | ReLU |
| Hidden layers in $K(x)$ | $[32]$ |
| Activations in $K(x)$ | Tanh |
| Initial Lipschitz constant $K_{\text{init}}$ | 1 |
| Hidden layers in critic network | $[64, 64]$ |
| Activations in critic network | ReLU |
| Optimizer | Adam |
| Actor learning rate $\eta_f$ | $3 \cdot 10^{-5}$ |
| Actor learning rate $\eta_k$ | $1 \cdot 10^{-5}$ |
| Critic learning rate | $8 \cdot 10^{-5}$ |
| Weight $\lambda$ | $1 \cdot 10^{-1}$ |
| Small constant $\epsilon$ | $1 \cdot 10^{-4}$ |

## F. Implementation Details on Vehicle Trajectory Tracking Environment

To train on the vehicle trajectory tracking environment, which is a constrained optimal control problem, we use the Separated Proportional-integral Lagrangian algorithm (SPIL) (Peng et al., 2021). We use the same hyperparameters for sine and double-line scenarios. The hyperparameters of SPIL are listed in Table 10.

*Table 10.* Hyperparameters of SPIL.

| Parameter | Setting |
|---|---|
| Replay buffer capacity | 100000 |
| Buffer warm-up size | 1000 |
| Batch size | 64 |
| Discount $\gamma$ | 0.99 |
| Target network soft-update rate $\tau$ | 0.005 |
| Initial random interaction steps | 0 |
| Interaction steps per iteration | 8 |
| Network update times per iteration | 1 |
| Prediction step | 10 |
| Action bound | $[-0.4, 0.4]$ |
| Exploration noise std. deviation | 0.2 |
| Hidden layers in $f(x)$ | $[64, 64]$ |
| Activations in $f(x)$ | ReLU |
| Hidden layers in $K(x)$ | $[32]$ |
| Activations in $K(x)$ | Tanh |
| Initial Lipschitz constant $K_{\text{init}}$ | 5 |
| Hidden layers in critic network | $[64, 64]$ |
| Activations in critic network | ReLU |
| Optimizer | Adam |
| Actor learning rate $\eta_f$ | $3 \cdot 10^{-4}$ |
| Actor learning rate $\eta_k$ | $1 \cdot 10^{-4}$ |
| Critic learning rate | $3 \cdot 10^{-4}$ |
| Weight $\lambda$ | $1 \cdot 10^{-3}$ |
| Small constant $\epsilon$ | $1 \cdot 10^{-4}$ |

We set different observation noises and compare the results of SPIL with LipsNet-L, SPIL with LipsNet-G, SPIL with MLP, and MPC. Table 11 summarizes the total average return. Table 12 summarizes the action fluctuation ratio. Figure 14 shows the variation trends of total average return and action fluctuation ratio as the noise level increases. As shown in Figure 14(a), the total average return of MLP decreases much faster than that of LipsNet-L and LipsNet-G. As shown in Figure 14(b), the action fluctuation ratio of MLP increases much faster than that of LipsNet-L and LipsNet-G. These results indicate that LipsNet has superior action smoothness and noise robustness compared to MLP.

## G. DeepMind Control Suit Benchmark

The DeepMind Control Suite (DMControl) (Tassa et al., 2018) contains a set of well-designed continuous control tasks. These tasks have standardized structures, interpretable and normalized rewards. The tasks in DMControl are written in Python and powered by MuJoCo physic engine (Todorov et al., 2012). Currently, it is one of the most recognized benchmark environments for RL and continuous control.

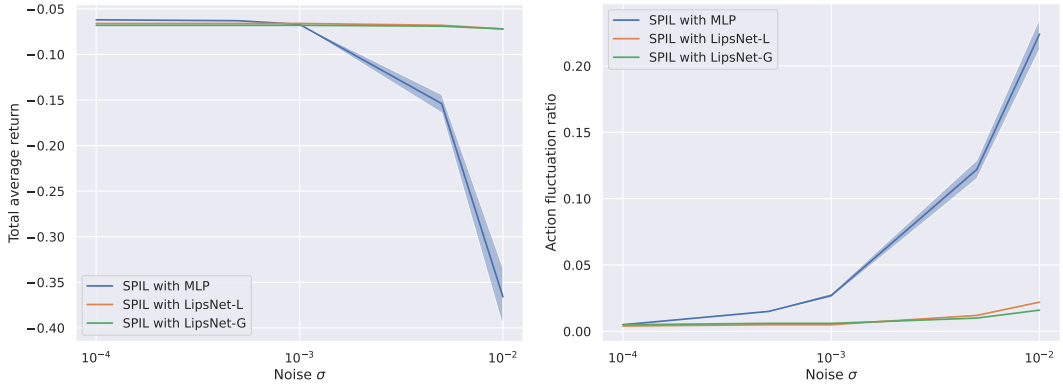
The domain in DMControl refers to a physical model, while a task refers to an instance of that model with a particular MDP structure. For example, the difference between the `swingup` and `balance` tasks of the cart-

*Table 11.* Total average return comparison on vehicle trajectory tracking environment. The observation noise in each dimension is distributed in $U(-\sigma, \sigma)$.

| Env | | Method | | |
|---|---|---|---|---|
| Trajectory | Noise $\sigma$ | SPIL (MLP) | SPIL (LipsNet-L) | SPIL (LipsNet-G) |
| sine curve | $1 \cdot 10^{-4}$ | **-0.062** ± 0.0001 | -0.066 ± 0.0001 | -0.068 ± 0.0001 |
| | $5 \cdot 10^{-4}$ | **-0.063** ± 0.0001 | -0.066 ± 0.0001 | -0.068 ± 0.0001 |
| | $1 \cdot 10^{-3}$ | -0.067 ± 0.0005 | **-0.066** ± 0.0001 | -0.068 ± 0.0001 |
| | $5 \cdot 10^{-3}$ | -0.154 ± 0.0101 | **-0.068** ± 0.0003 | -0.069 ± 0.0002 |
| | $1 \cdot 10^{-2}$ | -0.366 ± 0.0322 | **-0.072** ± 0.0009 | -0.072 ± 0.0006 |
| double-line | $1 \cdot 10^{-4}$ | **-0.088** ± 0.0002 | -0.113 ± 0.0001 | -0.115 ± 0.0001 |
| | $5 \cdot 10^{-4}$ | **-0.089** ± 0.0005 | -0.113 ± 0.0003 | -0.115 ± 0.0001 |
| | $1 \cdot 10^{-3}$ | **-0.092** ± 0.0020 | -0.113 ± 0.0005 | -0.115 ± 0.0004 |
| | $5 \cdot 10^{-3}$ | -0.180 ± 0.0138 | **-0.113** ± 0.0026 | -0.116 ± 0.0014 |
| | $1 \cdot 10^{-2}$ | -0.351 ± 0.0187 | **-0.117** ± 0.0063 | -0.119 ± 0.0027 |

*Table 12.* Action fluctuation ratio comparison on vehicle trajectory tracking environment. The observation noise in each dimension is distributed in $U(-\sigma, \sigma)$.

| Env | | Method | | |
|---|---|---|---|---|
| Trajectory | Noise $\sigma$ | SPIL (MLP) | SPIL (LipsNet-L) | SPIL (LipsNet-G) |
| sine curve | $1 \cdot 10^{-4}$ | 0.005 ± 0.0002 | **0.004** ± 0.0001 | 0.005 ± 0.0001 |
| | $5 \cdot 10^{-4}$ | 0.015 ± 0.0006 | **0.005** ± 0.0001 | 0.006 ± 0.0001 |
| | $1 \cdot 10^{-3}$ | 0.027 ± 0.0016 | **0.005** ± 0.0001 | 0.006 ± 0.0001 |
| | $5 \cdot 10^{-3}$ | 0.122 ± 0.0070 | 0.012 ± 0.0007 | **0.010** ± 0.0003 |
| | $1 \cdot 10^{-2}$ | 0.224 ± 0.0115 | 0.022 ± 0.0009 | **0.016** ± 0.0001 |
| double-line | $1 \cdot 10^{-4}$ | 0.024 ± 0.0003 | 0.019 ± 0.0001 | **0.016** ± 0.0001 |
| | $5 \cdot 10^{-4}$ | 0.032 ± 0.0008 | 0.019 ± 0.0001 | **0.016** ± 0.0001 |
| | $1 \cdot 10^{-3}$ | 0.043 ± 0.0018 | 0.020 ± 0.0001 | **0.017** ± 0.0001 |
| | $5 \cdot 10^{-3}$ | 0.137 ± 0.0094 | 0.026 ± 0.0005 | **0.021** ± 0.0004 |
| | $1 \cdot 10^{-2}$ | 0.227 ± 0.0198 | 0.034 ± 0.0013 | **0.026** ± 0.0009 |




(a) Decreasing trend of total average return

(b) Increasing trend of action fluctuation ratio

*Figure 14.* Trend of total average return and action fluctuation ratio when noise level changes in vehicle trajectory tracking environment. The reference trajectory is a sine curve. The data in subfigures (a) and (b) are from Table 11 and 12, respectively. X-axis is the radius of uniformly distributed noise applied in each observation dimension. Y-axis is the mean of total average return or action fluctuation ratio. Shadow means the standard deviation.

pole domain is whether the pole is initialized pointing downwards or upwards, respectively. We list the detailed descriptions of the domains used in this paper below, names are followed by three integers specifying the dimensions of the state, action and observation spaces, i.e., $\left(\dim(\mathcal{S}), \dim(\mathcal{A}), \dim(\mathcal{O})\right)$.

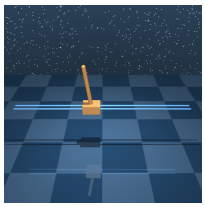


*Figure 15.* **Cartpole(4, 1, 5):** This domain includes a cart and a pole, connected by an un-actuated joint. The domain contains four tasks. In our experiment, the `swingup` task is used. In this task, the pole starts pointing down and we need to apply force on the cart, therefore keeping the pole upright upward.

*Figure 16.* **Reacher(4, 2, 7):** This domain includes two linked poles and a sphere with a randomly initialized location. One endpoint of the linked poles is fixed while the other is free. The domain contains two tasks. In our experiment, the `easy` task is used. In this task, we need to apply force and make the free endpoint keep inside the target sphere.

*Figure 17.* **Cheetah(18, 6, 17):** This domain includes a planar biped. Only one task is involved in this domain, which is the `run` task. In this task, we need to control the planar biped to stand up and run with a forward velocity.

*Figure 18.* **Walker(18, 6, 24):** This domain includes a planar walker. The domain contains three tasks. In our experiment, the `walk` task in this domain is used. In this task, we need to control the walker to keep an upright torso, required torso height and forward velocity.

## H. Implementation Details on DMControl

We use the model-free reinforcement learning algorithm, Twin Delayed Deep Deterministic policy gradient (TD3), to train on DMControl. The hyperparameters of TD3 are the same in all environments, while only the weight $\lambda$ in loss varies. The network $K(x)$ needs a relatively large output at the beginning of training for sufficient exploration. Therefore, we add the expected initial Lipschitz constant $K_{\text{init}}$ to the bias of the last linear layer before `Softplus` in $K(x)$ network. The hyperparameters of TD3 are listed in Table 13. The weights $\lambda$ are listed in Table 14.

*Table 13.* Hyperparameters of TD3.

| Parameter | Setting |
|---|---|
| Replay buffer capacity | 1000000 |
| Buffer warm-up size | 1000 |
| Batch size | 100 |
| Discount $\gamma$ | 0.99 |
| Target network soft-update rate $\tau$ | 0.005 |
| Target noise | 0.2 |
| Target noise limit | 0.5 |
| Exploration noise std. deviation | 0.1 |
| Policy delay times | 2 |
| Initial random interaction steps | 25000 |
| Interaction steps per iteration | 50 |
| Network update times per iteration | 50 |
| Hidden layers in $f(x)$ | $[64, 64]$ |
| Activations in $f(x)$ | ReLU |
| Hidden layers in $K(x)$ | $[32]$ |
| Activations in $K(x)$ | Tanh |
| Initial Lipschitz constant $K_{\text{init}}$ | 50 |
| Hidden layers in critic network | $[64, 64]$ |
| Activations in critic network | ReLU |
| Optimizer | Adam |
| Actor learning rate $\eta_f$ | $1 \cdot 10^{-3}$ |
| Actor learning rate $\eta_k$ | $1 \cdot 10^{-5}$ |
| Critic learning rate | $1 \cdot 10^{-3}$ |
| Small constant $\epsilon$ | $1 \cdot 10^{-4}$ |

*Table 14.* Weight $\lambda$ on DMControl.

| Env | Weight $\lambda$ |
|---|---|
| Cartpole | $10^{-3}$ |
| Reacher | $10^{-5}$ |
| Cheetah | $10^{-7}$ |
| Walker | $10^{-5}$ |

To evaluate the noise robustness of LipsNet, we set observation noises in DMControl environments. The observation noises used in Section 4.3 are listed in Table 15.

*Table 15.* Observation noise in DMControl environments. The observation noise in each dimension is distributed in $U(-\sigma, \sigma)$.

| Env | Noise $\sigma$ |
|---|---|
| Cartpole | $[0.1, 0.1, 0.1, 0.2, 0.2]$ |
| Reacher | [0.001 repeats 7 times] |
| Cheetah | $[0.01, 0.01, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.5, 0.05, 0.1, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]$ |
| Walker | [0.25 repeats 24 times] |

## I. Comparison to Reward Penalty

Punishing the difference between consecutive actions in the reward is an effective way to smooth the actions in some environments. However, such an approach breaks the Markov property, which affects the performance, albeit to a minor extent in certain environments. Moreover, we found that adding reward penalty in a sparse reward environment increases action fluctuation rather than smoothing it, which is consistent with the finding by Chen et al. (2021).

Cartpole in DMControl is a sparse reward environment. The reward is 1 when the pole is within 30° of the vertical and 0 otherwise. We implement TD3 in this environment, punishing the difference between consecutive actions in the reward. Specifically, the new reward is $r = r_{\text{origin}} + \alpha \|a_{t+1} - a_t\|$, where $r_{\text{origin}}$ is the original sparse reward, $\alpha$ is the penalty coefficient and $a_{t+1}$ is the output of actor network under $s_{t+1}$. The experiment results are summarized in Table 16. The results imply that simply adding reward penalty in the sparse reward environment increases the action fluctuation ratio. Superiorly, LipsNet can smooth actions even in the sparse reward environment.

## J. Comparison to Previous Works

One of the main superiorities of LipsNet compare to previous works is that LipsNet smoothens actions by modifying at neural network level rather than algorithm level. In this way, RL algorithms will not be complicated, which makes LipsNet applicable to most existing RL algorithms. Among the previous action smoothing methods, only the network enhancement methods modify at neural network level and do not complicate RL algorithms, as illustrated in Section 1. Therefore, we only compare LipsNet with the network enhancement methods. Actually, LipsNet also belongs to the category of network enhancement methods. Ryoichi Takase's method (2020; 2022), marked as MLP-SN in our paper because it applies SN on MLP, is the only one belonging to the network enhancement methods category previously. MLP-SN needs a series of predefined hyperparameters whose product is the Lipschitz constant of actor network. Because only simple tasks' Lipschitz constant can be roughly estimated ahead, we only make comparisons on double integrator environment and DMControl-reacher environment. On more complex tasks, it is hard to finetune the Lipschitz constant of MLP-SN, which reflects the superiority of LipsNet since LipsNet can automatically adjust its Lipschitz constant.

For training on the double integrator environment, the Lipschitz constant of MLP-SN is set as 1, an appropriate value among our tests. The comparison results are listed in Table 17.

*Table 17.* Comparison on double integrator environment. The environment setting is the same as Section 4.1.

| Method | Total average return | Action fluctuation ratio |
|---|---|---|
| INFADP (MLP) | -78.4 $\pm$ 1.8 | 0.34 $\pm$ 0.02 |
| INFADP (MLP-SN) | -65.9 $\pm$ 1.7 | 0.13 $\pm$ 0.01 |
| INFADP (LipsNet-G) | -59.8 $\pm$ 1.1 | 0.17 $\pm$ 0.01 |
| INFADP (LipsNet-L) | **-58.7** $\pm$ 1.4 | **0.12** $\pm$ 0.01 |

As shown in Table 17, MLP-SN and LipsNet-G show similar behavior since they are both globally Lipschitz continuous neural networks. LipsNet-L has the highest return with the lowest action fluctuation ratio. This superiority benefits from the local Lipschitz continuity of LipsNet-L.

We set different observation noises for further evaluation on the double integrator environment, then compare the total average return and the action fluctuation ratio. The results are listed in Table 19 and Table 20. We draw the decreasing trend of total average return and the increasing trend of action fluctuation ratio. As shown in Figure 19, when the observation noise increases, the total average return of MLP decreases rapidly and the action fluctuation ratio of MLP grows rapidly. MLP-SN can smoothen the action, but its total average return is lower than that of LipsNet-L. Overall speaking, LipsNet-L has the lowest action fluctuation ratio with relatively good control performance.

For training on the DMControl-reacher environment with MLP-SN, we use a 3-layer MLP with SN and manually tune each layer's spectral norm. The global Lipschitz constant of MLP-SN is the product of all layers' spectral norms. The results are listed in Table 18.

*Table 18.* Performance of MLP-SN on DMControl-reacher.

| Layers spectral norm | Total average return | Action fluctuation ratio |
|---|---|---|
| $[5.0, 5.0, 5.0]$ | 760 $\pm$ 381 | 0.01 $\pm$ 0.00 |
| $[5.5, 5.5, 5.5]$ | 831 $\pm$ 102 | 0.01 $\pm$ 0.00 |
| $[5.8, 5.8, 5.8]$ | 954 $\pm$ 10 | 0.08 $\pm$ 0.05 |
| $[6.0, 6.0, 6.0]$ | 967 $\pm$ 28 | 0.13 $\pm$ 0.08 |

As shown in Table 18, the performances of MLP-SN in all settings are poor than that of LipsNet-L (listed in Table 3 and 4), i.e., LipsNet-L has a higher return with a lower action fluctuation ratio. It is because LipsNet constrains the whole network to be Lipschitz continuous, but SN produces layer-wise constraints reducing the expression ability. We do not conduct comparisons to MLP-SN across all environments used in this paper, since it requires manually tuning the spectral norm hyperparameters for each layer, yielding huge combinations of hyperparameters.

Moreover, LipsNet is not limited to specific environments and RL algorithms. Instead, it can be applied to most exist-

*Table 16.* Comparison to reward penalty.

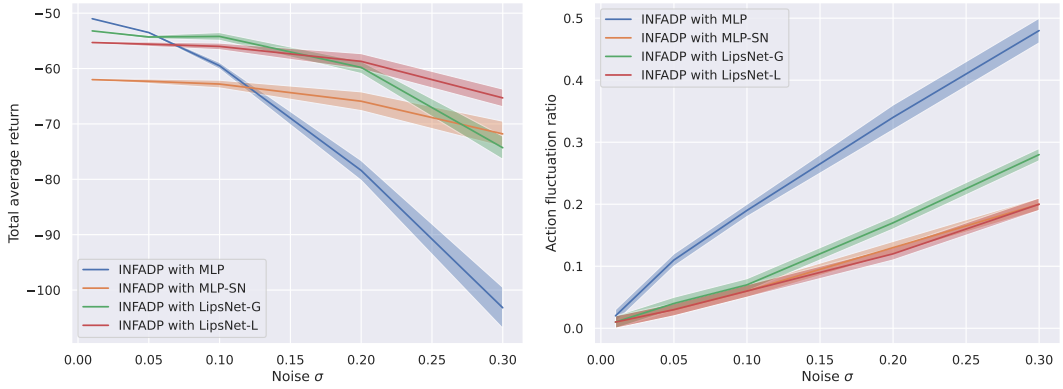
| Method | Penalty coefficient $\alpha$ | Total average return | Action fluctuation ratio |
|---|---|---|---|
| TD3 (MLP, reward penalty) | 0.01 | 825 $\pm$ 0.5 | 0.27 $\pm$ 0.01 |
| TD3 (MLP, reward penalty) | 0.1 | 819 $\pm$ 0.8 | 0.21 $\pm$ 0.01 |
| TD3 (MLP, reward penalty) | 1 | 13 $\pm$ 0.5 | 0.02 $\pm$ 0.00 |
| TD3 (MLP) | | 805 $\pm$ 0.8 | 0.04 $\pm$ 0.00 |
| TD3 (LipsNet-G) | | 691 $\pm$ 1.0 | 0.08 $\pm$ 0.00 |
| TD3 (LipsNet-L) | | 831 $\pm$ 0.9 | 0.01 $\pm$ 0.00 |

*Table 19.* Total average return comparison on double integrator environment. The observation noise in each dimension is distributed in $U(-\sigma, \sigma)$.

| Noise $\sigma$ | Method | | | |
|---|---|---|---|---|
| | INFADP (MLP) | INFADP (MLP-SN) | INFADP (LipsNet-G) | INFADP (LipsNet-L) |
| $1 \cdot 10^{-2}$ | **-51.0** $\pm$ 0.1 | -62.0 $\pm$ 0.1 | -53.2 $\pm$ 0.1 | -55.3 $\pm$ 0.1 |
| $5 \cdot 10^{-2}$ | **-53.5** $\pm$ 0.2 | -62.3 $\pm$ 0.4 | -54.3 $\pm$ 0.3 | -55.6 $\pm$ 0.4 |
| $1 \cdot 10^{-1}$ | -59.5 $\pm$ 0.6 | -62.8 $\pm$ 0.7 | **-54.2** $\pm$ 0.7 | -56.0 $\pm$ 0.6 |
| $2 \cdot 10^{-1}$ | -78.4 $\pm$ 1.8 | -65.9 $\pm$ 1.7 | -59.8 $\pm$ 1.1 | **-58.7** $\pm$ 1.4 |
| $3 \cdot 10^{-1}$ | -103.2 $\pm$ 3.7 | -71.8 $\pm$ 2.3 | -74.3 $\pm$ 2.1 | **-65.3** $\pm$ 1.6 |

*Table 20.* Action fluctuation ratio comparison on double integrator environment. The observation noise in each dimension is distributed in $U(-\sigma, \sigma)$.

| Noise $\sigma$ | Method | | | |
|---|---|---|---|---|
| | INFADP (MLP) | INFADP (MLP-SN) | INFADP (LipsNet-G) | INFADP (LipsNet-L) |
| $1 \cdot 10^{-2}$ | 0.02 $\pm$ 0.01 | 0.01 $\pm$ 0.01 | 0.01 $\pm$ 0.01 | **0.01** $\pm$ 0.01 |
| $5 \cdot 10^{-2}$ | 0.11 $\pm$ 0.01 | 0.03 $\pm$ 0.01 | 0.04 $\pm$ 0.01 | **0.03** $\pm$ 0.01 |
| $1 \cdot 10^{-1}$ | 0.19 $\pm$ 0.01 | 0.06 $\pm$ 0.01 | 0.07 $\pm$ 0.01 | **0.06** $\pm$ 0.01 |
| $2 \cdot 10^{-1}$ | 0.34 $\pm$ 0.02 | 0.13 $\pm$ 0.01 | 0.17 $\pm$ 0.01 | **0.12** $\pm$ 0.01 |
| $3 \cdot 10^{-1}$ | 0.48 $\pm$ 0.02 | 0.20 $\pm$ 0.01 | 0.28 $\pm$ 0.01 | **0.20** $\pm$ 0.01 |




(a) Decreasing trend of total average return

(b) Increasing trend of action fluctuation ratio

*Figure 19.* Trend of total average return and action fluctuation ratio when noise level changes in double integrator environment. The data in subfigures (a) and (b) are from Table 19 and 20, respectively. X-axis is the radius of uniformly distributed noise applied in each observation dimension. Y-axis is the mean of total average return or action fluctuation ratio. Shadow means the standard deviation.

ing actor-critic RL algorithms and various tasks, making it a more versatile action smoothing method. This is because LipsNet modifies the neural networks rather than the RL algorithms, and it does not require a rough estimate of the network's Lipschitz constant ahead or manual tuning of the Lipschitz constant. As a result, LipsNet's ability to automatically adjust Lipschitz constant and its general applicability to various tasks and RL algorithms make it a superior choice for action smoothing compared to previous works.

## K. Relationship with Robust RL

The primary connection between LipsNet and robust RL is that the Lipschitz continuity is a particular form of robustness. As described in Section 2.1, the Lipschitz constant could characterize the noise robustness level of a function. However, the primary difference is that robust RL aims to prevent catastrophic failures under uncertainties, perturbations or structural changes (Moos et al., 2022), while LipsNet focuses on smoothing the action trajectory during model deployment. The direct evidence of the primary difference is that LipsNet can smooth the action even in the environments without noise, as shown in Table 4 and Table 7. This superior is due to the landscape smoothness, as shown in Figure 10, rather than the noise robustness. Furthermore, LipsNet is a network-level design, but most robust RL methods are algorithm-level designs. In other words, robust RL achieves its aim through specific algorithms, such as SA-MDP (Zhang et al., 2020) for observation perturbation scenarios, NR-MDP (Tessler et al., 2019) for action perturbation scenarios, and RARL (Pinto et al., 2017) for parameter perturbation scenarios. In contrast, LipsNet achieves its aim by designing a particular network structure, which can be used as actor networks in many RL algorithms rather than one specific algorithm.