TOWARDS ROBUST NEURAL NETWORKS VIA CLOSE-LOOP CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite their success in massive engineering applications, deep neural networks are vulnerable to various perturbations due to their black-box nature. Recent study has shown that a deep neural network can misclassify the data even if the input data is perturbed by an imperceptible amount. In this paper, we address the robustness issue of neural networks by a novel close-loop control method from the perspective of dynamic systems. Instead of modifying the parameters in a fixed neural network architecture, a close-loop control process is added to generate control signals adaptively for the perturbed or corrupted data. We connect the robustness of neural networks with optimal control using the geometrical information of underlying data to design the control objective. The detailed analysis shows how the embedding manifolds of state trajectory affect error estimation of the proposed method. Our approach can simultaneously maintain the performance on clean data and improve the robustness against many types of data perturbations. It can also further improve the performance of robustly trained neural networks against different perturbations. To the best of our knowledge, this is the first work that improves the robustness of neural networks with close-loop control.

1 INTRODUCTION

Due to the increasing data and computing power, deep neural networks have achieved state-of the art performance in many applications such as computer vision, natural language processing and recommendation systems. However, many deep neural networks are vulnerable to various malicious perturbations due to their black-box nature: a small (even imperceptible) perturbation of input data may lead to completely wrong predictions (Szegedy et al., 2013; Nguyen et al., 2015). This has been a major concern in some safety-critical applications such as autonomous driving (Grigorescu et al., 2020) and medical image analysis (Lundervold & Lundervold, 2019). Various perturbations have been reported, including the ℓ_p norm based attack (Madry et al., 2017; Moosavi-Dezfooli et al., 2016; Carlini & Wagner, 2017), semantic perturbation (Engstrom et al., 2017) etc. On the other side, some algorithms to improve the robustness against those perturbations have shown great success (Madry et al., 2017). However, most robustly trained models are tailored for certain types of perturbations, and they do not work well for other types of perturbations. Khoury & Hadfield-Menell (2018) showed the non-existence of optimal decision boundary for any ℓ_p-norm perturbation.

Recent works (E, 2017; Haber & Ruthotto, 2017) have shown the connection between dynamical systems and neural networks. This dynamic system perspective provides some interesting theoretical insights about the robustness issue. Given a set of data $\mathbf{x}_0 \in \mathbb{R}^d$ and its labels $\mathbf{y} \in \mathbb{R}^l$ with a joint distribution $\mathcal{D}$, training a neural network can be considered as following

$$\min_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\mathbf{x}_0,\mathbf{y})\sim\mathcal{D}} [\Phi(\mathbf{x}_T, \mathbf{y})], \quad \text{s.t. } \mathbf{x}_{t+1} = f(\mathbf{x}_t, \boldsymbol{\theta}_t),$$

where $\boldsymbol{\theta}$ are the unknown parameters to train, and f, Φ represent the forward propagation rule and loss function (e.g. cross-entropy) respectively. The dynamical system perspective interprets the vulnerability of neural networks as a system instability issue, which addresses the state trajectory variation under small perturbations applied on initial conditions. The optimal control theory focuses on developing a control model to adjust the system state trajectory in a optimal manner. The first work that links and extends the classical back-propagation algorithm using optimal control theory was presented in Li et al. (2017), where the direct relationship between the Pontryagin's Maximum

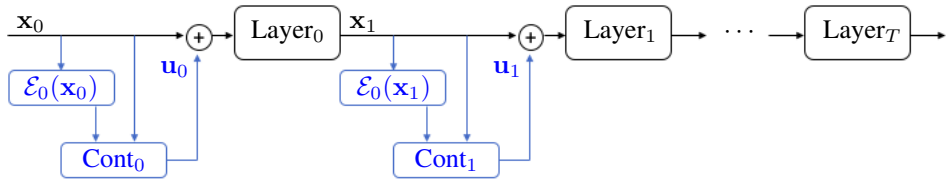


Figure 1: The structures of feed-forward neural network (black) and the proposed method (blue).

Principle (Kirk, 1970) and the gradient based network training was established. Ye et al. (2019) used control theory to adjust the hyperparameters in the adversarial training algorithm. Han et al. (2018) established the mathematical basis of the optimal control viewpoint of deep learning. These existing works on algorithm development are *open-loop control methods* since they commonly treat the network weights $\boldsymbol{\theta}$ as control parameters and keep them fixed once the training is done. The fixed control parameters $\boldsymbol{\theta}$ operate optimally for data sampled from the data distribution $\mathcal{D}$. However, various perturbation methods cause different data distribution deviating from the true distribution $\mathcal{D}$ (Song et al., 2017) and cause poor performance with the fixed open-loop control parameters.

1.1 Paper Contributions

To address the limitation of using *open-loop control methods*, we propose the **C**lose- **L**oop **C**ontrol **N**eural **N**etwork (**CLC-NN**), the first close-loop control method to improve the robustness of neural networks. As shown in Fig. 1, our method adds additional blocks to a given T-layer neural network: embedding functions $\mathcal{E}_t$, which induce running losses in all layers that measure the discrepancies between true features and observed features under input perturbation, then control processes generate control variables $\mathbf{u}_t$ to minimize the total running loss under various data perturbations. The original neural network can be designed by either standard training or robust training. In the latter case, our CLC-NN framework can achieve extra robustness against different perturbations. The forward propagation rule is thus modified with an extra control parameter $\mathbf{u}_t \in \mathbb{R}^{d'}$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \boldsymbol{\theta}_t, \mathbf{u}_t).$$

The technical contributions of this paper are summarized below:

- The proposed method relies on the well accepted assumption that the data and hidden state manifolds are low dimensional compared to the ambient dimension (Fefferman et al., 2016). We study the geometrical information of the data and hidden layers to define the objective function for control. Given a trained T-layer neural network, a set of embedding functions $\mathcal{E}_t$ are trained off-line by minimizing the reconstruction loss $\|\mathcal{E}(\mathbf{x}_t) - \mathbf{x}_t\|$ over some clean data from $\mathcal{D}$ only. The embedding functions support defining a running loss required in our control method.
- We define the control problem by dynamic programming and implement the online iterative solver based on the Pontryagin's Maximum Principle to avoid the curse of dimensionality. The proposed close-loop control formulation does not require prior information of the perturbation.
- We provide a theoretical error bound of the controlled system for the simplified case with linear activation functions and linear embedding. This error bound reveals how the close-loop control improves neural network robustness in the simplest setting.

2 Related Works

Many techniques have been reported to improve the robustness of neural networks, such as data augmentation (Shorten & Khoshgoftaar, 2019), gradient masking (Liu et al., 2018), etc. We review adversarial training and reactive defense which are most relevant to this work.

Adversarial Training. Adversarial training is (possibly) the most popular robust training method, and it solves a min-max robust optimization problem to minimize the worse-case loss with perturbed data. Adversarial training effectively regularizes the network's local Lipschitz constants of the loss surface around the data manifold (Liu et al., 2018). Zhang et al. (2019) formulated the robustness training using the Pontryagon's Maximum Principle, such *open-loop control methods* result in a

set of fixed parameters that operates optimally on the considered perturbation. Liu et al. (2020a;b) considered a close-loop formulation from the differential dynamic programming perspective, this algorithm is categorized as a *open-loop control method* because it utilizes the state feedback information to boost the training convergence and results in a set of fixed controls for any unseen data. On the contrary, the proposed CLC-NN formulation adaptively targets on the inputs with different control parameters and is capable of distinguishing clean data by generating no control.

Reactive Defense. A reactive defense method tries to reject or pre-process the input data that may cause mis-classifications. Metzen et al. (2017) rejected perturbed data by using adversarial detectors that are trained with adversarial data to detect abnormal data during forward propagation. Song et al. (2017) estimated the input data distribution $\mathcal{D}$ with a generative model (Oord et al., 2016) to detect data that does not belong to $\mathcal{D}$, it applies a greedy method to search the local neighbour of input data for a more statistically plausible counterpart. This purification process has shown improved accuracy with adversarial data contaminated by various types of perturbations. Purification can be considered as a one-step method to solve the optimal control problem that has the objective function defined over the initial condition only. On the contrary, the proposed CLC-NN solves the control problem by the dynamic programming principle and its objective function is defined over the entire state trajectory, which guarantees the optimality for the resulted controls.

3 The Close-loop Control Framework for Neural Networks

Let us now present our close-loop optimal control formulation to address the robustness issue of deep learning. Consider a neural network consisting of model parameters $\boldsymbol{\theta}$ equipped with external control policy $\boldsymbol{\pi}$, where $\boldsymbol{\pi} \in \boldsymbol{\Pi}$ is a collection of functions $\mathbb{R}^d \to \mathbb{R}^{d'}$ acting on the state and outputting the control signal. The feed-forward propagation in a T-layer neural network can be represented as

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \boldsymbol{\theta}_t, \boldsymbol{\pi}_t(\mathbf{x}_t)), \quad t = 0, \cdots, T-1. \tag{1}$$

Given a trained network, we solve the following optimization problem

$$\min_{\overline{\boldsymbol{\pi}}} \mathbb{E}_{(\mathbf{x}_0,\mathbf{y})\sim\mathcal{D}} \left[J(\mathbf{x}_0, \mathbf{y}, \overline{\boldsymbol{\pi}})\right] := \mathbb{E}_{(\mathbf{x}_0,\mathbf{y})\sim\mathcal{D}} \left[\Phi(\mathbf{x}_T, \mathbf{y}) + \sum_{s=0}^{T-1} \mathcal{L}(\mathbf{x}_s, \boldsymbol{\pi}_s(\mathbf{x}_s))\right], \quad s.t. \text{ Eq. (1)}, \tag{2}$$

where $\overline{\boldsymbol{\pi}}$ collects the control policies $\boldsymbol{\pi}_0, \cdots, \boldsymbol{\pi}_{T-1}$ for all layers. Note that (2) differs from the open-loop control used in standard training. An open-loop control that treats the network parameters as control variables seeks for a set of *fixed* parameters $\boldsymbol{\theta}$ to match the output with true label $\mathbf{y}$ by minimizing the terminal loss Φ, and the running loss $\mathcal{L}$ defines a regularization for $\boldsymbol{\theta}$. However, the terminal and running losses play different roles when our goal is to improve the robustness of a neural network by generating some adaptive controls for different inputs.

Challenge of Close-loop Control for Neural Networks. Optimal control has been well studied in the control community for trajectory optimization, where one defines the running loss as the error between the actual state $\mathbf{x}_t$ and a reference state $\mathbf{x}_{t,\text{ref}}$ over time interval $[0, T]$. The resulting control policy adjusts $\mathbf{x}_t$ and make it approach $\mathbf{x}_{t,\text{ref}}$. In this paper, we apply the idea of trajectory optimization to improve the robustness of a neural network via adjusting the undesired state of $\mathbf{x}_t$. However, the formulation is more challenging in neural networks: we do not have a "reference" state during the inference process, therefore it is unclear how to define the running loss $\mathcal{L}$.

In the following, we investigate manifold embedding of the state trajectory to precisely define the loss functions Φ and $\mathcal{L}$ of Eq. (2) required for the control objective function of a neural network.

3.1 Manifold Learning for State Trajectories

State Manifold. Our controller design is based on the "manifold hypothesis": real-world high dimensional data can often be embedded in a lower dimensional manifold $\mathcal{M}$ (Fefferman et al., 2016). Indeed, neural networks extract the embedded features from $\mathcal{M}$. To fool a well-trained neural network, the perturbed data often stays away from the data manifold $\mathcal{M}$ (Khoury & Hadfield-Menell, 2018). We consider the data space Z ($\mathbf{x} \in Z, \forall \mathbf{x} \sim \mathcal{D}$) as: $Z = Z_{\|} \bigoplus Z_{\perp}$, where $Z_{\|}$ contains the embedded manifold $\mathcal{M}$ and $Z_{\perp}$ is the orthogonal complement of $Z_{\|}$. During forward propagation, the state manifold embedded in $Z_{\|}$ varies at different layers due to both the nonlinear activation

function f and state dimensionality variation. Therefore, we denote $Z^t = Z^t_{\|} \bigoplus Z^t_{\perp}$ as the state space decomposition at layer t and $\mathcal{M}_t \in Z^t_{\|}$. Once an input data sample is perturbed, the main effects of causing misclassifications are in $Z_{\perp}$. Therefore, it is important to measure how far the possibly perturbed state $\mathbf{x}_t$ deviates from the state manifold $\mathcal{M}_t$.

Embedding Function. Given an embedding function $\mathcal{E}_t$ that encodes $\mathbf{x}_t$ onto the lower-dimensional manifold $\mathcal{M}_t$ and decodes the result back to the full state space Z_t, the reconstruction loss $\|\mathcal{E}_t(\mathbf{x}_t) - \mathbf{x}_t\|$ measures the deviation of the possibly perturbed state $\mathbf{x}_t$ from the manifold $\mathcal{M}_t$. The reconstruction loss is nonzero as long as $\mathbf{x}_t$ has components in $Z^t_{\perp}$. The embedding functions are constructed offline by minimizing the total reconstruction losses over a clean training data set.

- **Linear Case:** $\mathcal{E}_t(\cdot)$ can be considered as $\mathbf{V}^r_t(\mathbf{V}^r_t)^T$ where $\mathbf{V}^r_t$ is a matrix whose columns are an orthonormal basis for $Z^t_{\|}$. Specifically one can first perform a principle component analysis over a collection of hidden states at layer t, then $\mathbf{V}^r_t$ can be obtained as the first r columns of the resulting eigenvectors.
- **Nonlinear Case:** we choose a convolutional auto-encoder (detailed in Appendix B) to obtain a representative manifold embedding function $\mathcal{E}_t$ due to its ease of implementation. Based on the assumption that most perturbations are in the $Z_{\perp}$ subspace, the embedding are effective to detect the perturbations as long as the target manifold is of a low dimension. Alternative manifold learning methods such as Izenman (2012) may also be employed.

3.2 Formulation for the Close-Loop Control of Neural Networks

Control Objectives. The above embedding function allows us to define a **running loss** $\mathcal{L}$:

$$\mathcal{L}(\mathbf{x}_t, \boldsymbol{\pi}_t(\mathbf{x}_t), \mathcal{E}_t(\cdot)) = \|\mathcal{E}_t(\mathbf{x}_t) - \mathbf{x}_t\|_2^2 + (\boldsymbol{\pi}_t(\mathbf{x}_t))^T \mathbf{R} \boldsymbol{\pi}_t(\mathbf{x}_t). \tag{3}$$

Here the matrix $\mathbf{R}$ defines a regularization term promoting controls of small magnitudes. In practical implementations, using a diagonal matrix $\mathbf{R}$ with small elements often helps to improve the performance. Now we are ready to design the control objective function of CLC-NN. Different from a standard open-loop control, this work sets the terminal loss Φ as zero because no true label is given during inference. Consequently, the **close-loop control** formulation in Eq. (2) becomes

$$\min_{\overline{\boldsymbol{\pi}}} \mathbb{E}_{(\mathbf{x}_0, \mathbf{y}) \sim \mathcal{D}} \left[J(\mathbf{x}_0, \mathbf{y}, \overline{\boldsymbol{\pi}}) \right] := \min_{\overline{\boldsymbol{\pi}}} \mathbb{E}_{(\mathbf{x}_0, \mathbf{y}) \sim \mathcal{D}} \sum_{t=0}^{T-1} \left[\mathcal{L}(\mathbf{x}_t, \boldsymbol{\pi}_t(\mathbf{x}_t), \mathcal{E}_t(\cdot)) \right], \quad s.t.\ Eq.\ (1). \tag{4}$$

Assume that the input data is perturbed by a bounded and small amount, i.e.,

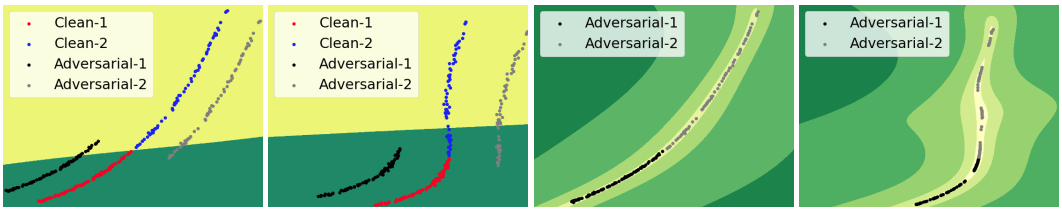
$$\mathbf{x}_{\epsilon,0} = \mathbf{x}_0 + \epsilon \cdot \mathbf{z},$$

where $\mathbf{z}$ can be either random or adversarial. The proposed CLC-NN adjusts the perturbed state trajectory $\mathbf{x}_{\epsilon,t}$ such that it stays at a minimum distance from the desired manifold $\mathcal{M}_t$ while promoting small magnitudes of controls.

Intuition. We use a toy example to intuitively show how the proposed CLC-NN controls the state trajectory of an unseen data sample. We create a synthetic binary classification data set with 1500 samples. We train a residual neural network with one hidden layer of dimension 2, and adopt the fast gradient sign method (Goodfellow et al., 2014) to generate adversarial data. Fig. 2 (a) and (b) show the states of clean data (red and blue) and of perturbed data (black and gray) at $t = 0$ and $t = 1$, respectively. The proposed CLC-NN adjusts the state trajectory to reduce the reconstruction loss as shown in Fig. 2 (c) and (d), where lighter background color represents lower reconstruction loss. By comparing Fig. 2 (a) with (c), and Fig. 2 (b) with (d), we see that the perturbed states in Fig. 2 (a) and (b) deviate from the desired state manifold (light green region) and has a high reconstruction loss. Running 1000 iterations of Alg. 1 adjusts the perturbed states and improves the classification accuracy from 86% to 100%.

4 Implementation via the Pontryagin's Maximum Principle

Dynamic Programming for Close-Loop Control (4). The control problem in Eq. (4) can be solved by the dynamical programming principle (Bellman, 1952). For simplicity we consider one



(a) $\mathbf{x}_0, \mathbf{x}_{\epsilon,0}$ (b) $\mathbf{x}_1, \mathbf{x}_{\epsilon,1}$ (c) Controlled $\overline{\mathbf{x}}_{\epsilon,0}$ (d) Controlled $\overline{\mathbf{x}}_{\epsilon,1}$

Figure 2: (a) and (b) show the states of clean data (red and blue) and of perturbed data (black and gray) at the initial and hidden layers respectively. The yellow and green backgrounds represent the two classes predicted by the network. Some of the perturbed data are mis-classified. (c) and (d) show the adjusted states after the proposed close-loop control. The background lightness from light to dark represent the increasing reconstruction loss $\|\mathcal{E}_t(\mathbf{x}_t) - \mathbf{x}_t\|_2^2$.

input data sample, and define a value function $V : \mathcal{T} \times \mathbb{R}^d \to \mathbb{R}$ (where $\mathcal{T} := \{0, 1, \ldots, T-1\}$). Here $V(t, \mathbf{x})$ represents the optimal cost-to-go function of Eq. (4) incurred from time t at state $\mathbf{x}$. One can show that $V(t, \mathbf{x})$ satisfies the dynamic programming principle

$$V(t, \mathbf{x}) = \inf_{\boldsymbol{\pi} \in \boldsymbol{\Pi}} \left[V(t+1, \mathbf{x} + f(\mathbf{x}, \boldsymbol{\theta}_t, \boldsymbol{\pi}(\mathbf{x}))) + \mathcal{L}(\mathbf{x}, \boldsymbol{\pi}(\mathbf{x}), \mathcal{E}_t(\cdot)) \right]. \tag{5}$$

Eq. (5) gives a necessary and sufficient condition for optimality of Eq. (4), and it is often solved backward in time by discretizing the entire state space. The state dimension of a modern neural network is at the order of thousands or even higher, therefore, discretizing the state space and directly solving Eq. (5) is intractable for real-world applications due to the curse of dimensionality.

Solving (5) via the Pontryagin's Maximum Principle. To overcome the computational challenge, the Pontryagin's Maximum Principle (Kirk, 1970) converts the intractable dynamical programming into two ordinary differential equations and a maximization condition. Instead of computing the control policy $\boldsymbol{\pi}$ of Eq. (5), the Pontryagin's Maximum Principle provides a necessary condition for the optimality with a set of control parameters $[\mathbf{u}_0^*, \cdots, \mathbf{u}_T^*]$. The mean-field Pontryagin's Maximum Principle can be considered when the initial condition is a batch of i.i.d. samples drawn from $\mathcal{D}$. Specifically, we trade the intractable computational complexity with processing time for solving the Hamilton equations and its maximization condition for every newly observed data. To begin with, we define the Hamiltonian $H : \mathcal{T} \times \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^l \times \mathbb{R}^m \to \mathbb{R}$ as

$$H(t, \mathbf{x}_t, \mathbf{p}_{t+1}, \boldsymbol{\theta}_t, \mathbf{u}_t) := \mathbf{p}_{t+1}^T \cdot f(\mathbf{x}_t, \boldsymbol{\theta}_t, \mathbf{u}_t) - \mathcal{L}(\mathbf{x}_t, \mathbf{u}_t, \mathcal{E}_t(\cdot)). \tag{6}$$

Let $\mathbf{x}^*$ denote the corresponding optimally controlled state trajectory. There exists a co-state process $\mathbf{p}^* : [0, T] \to \mathbb{R}^d$ such that the Hamilton's equations

$$\mathbf{x}_{t+1}^* = \nabla_p H(t, \mathbf{x}_t^*, \mathbf{p}_t^*, \boldsymbol{\theta}_t, \mathbf{u}_t^*), \qquad (\mathbf{x}_0^*, \mathbf{y}) \sim \mathcal{D}, \tag{7}$$

$$\mathbf{p}_t^* = \nabla_x H(t, \mathbf{x}_t^*, \mathbf{p}_{t+1}^*, \boldsymbol{\theta}_t, \mathbf{u}_t^*), \qquad \mathbf{p}_T^* = \mathbf{0}, \tag{8}$$

are satisfied. The terminal co-state $\mathbf{p}_T = \mathbf{0}$, since we do not consider the terminal loss $\Phi(\mathbf{x}_T, \mathbf{y})$. Moreover, we have the Hamiltonian maximization condition

$$H(t, \mathbf{x}_t^*, \mathbf{p}_t^*, \boldsymbol{\theta}_t, \mathbf{u}_t^*) \geq H(t, \mathbf{x}_t^*, \mathbf{p}_t^*, \boldsymbol{\theta}_t, \mathbf{u}_t), \forall \mathbf{u} \in \mathbb{R}^{d'} \text{ and } \forall t \in \mathcal{T}. \tag{9}$$

Instead of solving Eq. (5) for the optimal control policy $\boldsymbol{\pi}^*(\mathbf{x}_t)$, for a given initial condition, the Pontryagin's Maximum Principle seeks for a open-loop optimal solution such that the global optimum of Eq. (5) is satisfied. The limitation of using the maximum principle is that the control parameters $\mathbf{u}_t^*$ need to be solved for every unseen data to achieve the optimal solution.

Algorithm Flow. The numerical implementation of CLC-NN is summarized in Alg. 1. Given a trained network (either from standard or adversarial training) and a set of embedding functions, the controls are initialized as $\mathbf{u}_t = \mathbf{0}, \forall t \in \mathcal{T}$, because adding random initialization weakens the robustness performance in general, and clean trajectory often does not result in any running loss for the gradient update on the control parameters. In every iteration, a given input $\mathbf{x}_0$ is propagated forwardly with Eq. (7) to obtain all the intermediate hidden states $\mathbf{x}_t$ for all t and accumulates cost J. Eq. (8) backward propagates the co-state $\mathbf{p}_t$ and Eq. (9) maximizes the tth Hamiltonian with current $\mathbf{x}_t$ and $\mathbf{p}_t$ to compute the optimal control parameters $\mathbf{u}_t^*$.

Algorithm 1: CLC-NN with the Pontryagin's Maximum Principle.

Input : Possibly perturbed data $\mathbf{x}_\epsilon$, a trained neural network, embedding functions $[\mathcal{E}_1, \cdots, \mathcal{E}_{T-1}]$, maxItr (maximum number of iterations).

Output: A set of optimal control parameters $\mathbf{u}_0^*, \cdots, \mathbf{u}_{T-1}^*$.

for $k = 0$ *to* maxItr **do**
  $J_k = 0$,
  for $t = 0$ *to* $T-1$ **do**
    $\mathbf{x}_{t+1,k} = f(\mathbf{x}_{t,k}, \boldsymbol{\theta}_t, \mathbf{u}_{t,k})$, where $\mathbf{x}_{0,k} = \mathbf{x}_\epsilon$, ▷ Forward propagation Eq. (7),
    $J_k = J_k + \mathcal{L}(\mathbf{x}_{t,k}, \mathbf{u}_{t,k}, \mathcal{E}_t(\mathbf{x}_{t,k}))$, ▷ Objective function Eq. (4),
  end for
  for $t = T$ *to* 1 **do**
    $\mathbf{p}_{t,k} = \mathbf{p}_{t+1}^T \cdot \nabla_{\mathbf{x}_t} f(\mathbf{x}_{t,k}, \boldsymbol{\theta}_t, \mathbf{u}_{t,k}) - \nabla_{\mathbf{x}_t} \mathcal{L}(\mathbf{x}_{t,k}, \mathbf{u}_{t,k}, \mathcal{E}_t(\mathbf{x}_{t,k}))$,
    where $\mathbf{p}_{T,k} = \mathbf{0}$, ▷ Backward propagation Eq. (8)
  end for
  for $t = 0$ *to* $T-1$ **do**
    $\mathbf{u}_{t,k+1} = \mathbf{u}_{t,k} + \big(\mathbf{p}_{t+1,k}^T \cdot \nabla_{\mathbf{u}_t} f(\mathbf{x}_{t,k}, \boldsymbol{\theta}_t, \mathbf{u}_{t,k}) - \nabla_{\mathbf{u}_t} \mathcal{L}(\mathbf{x}_{t,k}, \mathbf{u}_{t,k}, \mathcal{E}_t(\mathbf{x}_{t,k}))\big)$,
    ▷ Maximization of Hamiltonian Eq. (9) based on Eq. (6) and gradient ascent.
  end for
end for

5 Error Analysis for Simplified Linear Cases

For the ease of analysis, we consider a simplified neural network with linear activation functions:

$$\mathbf{x}_{t+1} = \boldsymbol{\theta}_t(\mathbf{x}_t + \mathbf{u}_t),$$

and reveal why our proposed method can improve robustness in the simplest setting. Given a perturbed data sample $\overline{\mathbf{x}}_{\epsilon,0}$, we denote its perturbation-free counterpart as $\mathbf{x}_0$. We use a linear orthogonal projection as the embedding function such that $\mathcal{E}_t = \mathbf{V}_t^r(\mathbf{V}_t^r)^T$, where $\mathbf{V}_t^r$ is the first r columns of the eigenvectors computed by the Principle Component Analysis on a collection of states $\mathbf{x}_t$. The proposed CLC-NN minimizes $\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2$ by reducing the components of $\mathbf{x}_{\epsilon,t}$ that lie in the the orthogonal complement of $Z_\parallel^t$. By considering the input perturbation as a random vector, the following theorem provides an error estimation between $\overline{\mathbf{x}}_{\epsilon,t}$ and $\mathbf{x}_t$.

Theorem 1. *For $t \geq 1$, we have the error estimation*

$$\mathbb{E}\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 \leq \epsilon^2 \|\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_0\|_2^2 \left(\alpha^{2t} + \frac{r}{d}\left(2\alpha^t(1-\alpha^t) + \beta_t(1-\alpha^t)^2\right)\right), \tag{10}$$

where $\beta_t = \prod_{s=0}^{t-1} \kappa(\boldsymbol{\theta}_s) = \prod_{s=0}^{t-1} \|\boldsymbol{\theta}_s\|_2 \|\boldsymbol{\theta}_s^{-1}\|_2$, and $\alpha = \frac{c}{1+c}$, c represents the control regularization. In particular, the equality

$$\mathbb{E}\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 = \epsilon^2 \left(\alpha^{2t} + \frac{r}{d}(1-\alpha^{2t})\right) \tag{11}$$

holds if all $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t - 1$ are orthogonal.

The detailed derivation is presented in Appendix A. Let us summarize the insights from Theorem 1.

- The above error estimation decreases as the control regularization c goes to 0. It is not the sharpest possible as it relies on a greedily optimal control at each layer. The globally optimal control defined by the Ricatti equation may achieve a lower loss when $c \neq 0$.
- When the dimension of embedding subspace r decreases, our control becomes more effective in reducing $\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2$. This means that the control approach works the best when the data is constrained on a low dimensional manifold, which is consistent with the manifold hypothesis.
- The obtained upper bound is tight: the estimated upper bound becomes the actual error if all the forward propagation layers are orthogonal matrices.

Table 1: Experimental results on some networks from standard training.

Dataset	ϵ	Accuracy: Baseline / CLC-NN + Linear Embedding / CLC-NN + Nonlinear Embedding				
		Type of input perturbations				
		None	Rand	FGSM	PGD	CW
CIFAR-10	2	92 / 89 / 89	91 / 89 / 89	27 / 54 / 52	0 / 44 / 38	10 / 79 / 80
	4		89 / 86 / 90	19 / 39 / 34	0 / 35 / 34	0 / 79 / 80
	8		79 / 77 / 77	15 / 28 / 27	0 / 25 / 21	0 / 79 / 81
CIFAR-100	2	67 / 68 / 68	65 / 65 / 66	9 / 31 / 24	0 / 23 / 18	4 / 45 / 48
	4		59 / 61 / 60	5 / 20 / 20	0 / 10 / 15	1 / 43 / 48
	8		42 / 47 / 43	4 / 13 / 10	0 / 0 / 7	0 / 43 / 46

6 Numerical Experiments

We test our proposed CLC-NN framework under various input data perturbations. Here we briefly summarize our experimental settings, and we refer readers to Appendix B for the details.

- **Original Networks without Close-Loop Control.** We choose residual neural networks (He et al., 2016) with ReLU activation functions as our target for close-loop control. In order to show that CLC-NN can improve the robustness in various settings, we consider networks from both standard and adversarial trainings. We consider multiple adversarial training methods: fast gradient sign method (**FGSM**) (Goodfellow et al., 2014), project gradient descent (**PGD**) (Madry et al., 2017), and the free adversarial training (**Free**) (Shafahi et al., 2019).
- **Input Perturbations.** In order to test the performance of our CLC-NN framework, we perturb the input data within a radius of ϵ with $\epsilon = 2, 4$ and 8 respectively. We consider various perturbations, including non-adversarial perturbations with random uniform distributions (**Rand**), as well as some adversarial attacks such as **FGSM**, **PGD** and the **CW** methods (Carlini & Wagner, 2017).
- **CLC-NN Implementations.** We consider both linear and nonlinear embedding in our close-loop control. Specifically, we employ a principal component analysis with a 1% truncation error for linear embedding, and convolutional auto-encoders for nonlinear embedding. We use Adam (Kingma & Ba, 2014) to maximize the Hamiltonian function (9) and adjust the learning rate of Adam based on the input data perturbations.

Result Summary: Table 1 and Table 2 show the results for both CIFAR-10 and CIFAR-100 datasets on some neural networks from both standard training and adversarial training respectively.

- **CLC-NN significantly improves the robustness of neural networks from standard training.** Table 1 shows that the baseline network trained on a clean data set becomes completely vulnerable (with almost 0% accuracy) under PGD and CW attacks. Our CLC-NN improves its accuracy to nearly 40% and 80% under PGD and CW attacks respectively. The accuracy under FGSM attacks has almost been doubled by our CLC-NN method. The accuracy on clean data is slightly decreased because the lower-dimensional embedding functions cannot exactly capture $Z_{\|}$ or $\mathcal{M}$.
- **CLC-NN further improves the robustness of adversarially trained networks.** Table 2 shows that while an adversarially trained network is inherently robust against certain types of perturbations, CLC-NN strengthens its robustness significantly against various perturbations. For instance, CLC-NN improves the accuracy of an FGSM trained network under PGD and CW attacks by a maximum of 14% and 35%, respectively.
- **The robustness improvement of adversarially trained networks is less significant.** This is expected because the trajectory of perturbed data lies on the embedding subspace $Z_{\|}$ if that data sample has been used in adversarial training. However, our experiments show that applying CLC-NN to adversarially trained networks can achieve the best performance under most attacks.

Comparison with PixelDefend (Song et al., 2017). Compared with PixelDefend (Song et al., 2017), our method achieves similar performance on CIFAR-10 with slightly different experimental setting. Specifically, PixelDefend improved the robustness of a normally trained 62-layer ResNet from 0% to 78% against CW attack, and from 0% to 46% against the Basic Iterative Method (Kurakin et al., 2016) (denoted as PGD in Madry et al. (2017)). Our proposed CLC-NN improves the robustness of a 20-layer ResNet from 0% to 81% and from 0% to 44% against PGD and CW attacks, respectively. No result was reported for CIFAR-100 in Song et al. (2017).

Table 2: Experimental results on robustly trained networks.

Dataset	Training	ϵ	Accuracy: Baseline Model / CLC-NN + Linear Embedding				
			Type of input perturbations				
			None	Rand	FGSM	PGD	CW
CIFAR-10	FGSM	2	87 / 87	87 / 90	74 / 79	73 / 79	75 / 80
		4		87 / 90	60 / 65	54 / 61	60 / 75
		8		86 / 90	37 / 43	19 / 33	35 / 70
	PGD	2	82 / 82	82 / 82	74 / 75	73 / 75	73 / 77
		4		82 / 82	65 / 70	62 / 66	64 / 74
		8		82 / 83	48 / 53	39 / 47	46 / 69
	Free	2	85 / 84	85 / 84	75 / 75	74 / 75	75 / 76
		4		85 / 83	64 / 65	59 / 64	64 / 63
		8		85 / 81	43 / 47	29 / 44	42 / 65
CIFAR-100	FGSM	2	59 / 59	59 / 59	43 / 44	41 / 43	43 / 46
		4		59 / 59	30 / 32	26 / 30	30 / 42
		8		58 / 58	14 / 19	7 / 18	15 / 38
	PGD	2	53 / 54	53 / 53	43 / 43	42 / 43	42 / 45
		4		53 / 53	34 / 35	33 / 35	33 / 44
		8		53 / 53	21 / 23	17 / 22	21 / 41
	Free	2	58 / 59	58 / 59	45 / 46	44 / 45	44 / 56
		4		58 / 59	34 / 34	30 / 35	33 / 52
		8		58 / 58	18 / 22	11 / 19	19 / 49

Table 3: The accuracy of CLC-NN subtracted by that of reactive defense, ϵ_{attack} = 2 / 4 / 8.

Method	Type of input perturbations				
	None	Rand	FGSM	PGD	CW
CIFAR-10	-3	-2 / -3 / +2	+27 / +20 / +13	+43 / +35 / +25	+66 / +76 / +77
CIFAR-100	1	0 / +1 / +3	+22 / 0 / +9	+44 / +30 / +11	+37 / +30 / +16

Comparison with Reactive Defense When one considers data pre-processing as reactive defense, it can be understood as only applying a control at the initial condition of a dynamical system. Specifically, reactive defense equipped with linear embedding admits the following dynamics:

$$\overline{\mathbf{x}}_{t+1} = f(\overline{\mathbf{x}}_t, \boldsymbol{\theta}_t), \quad s.t.\ \overline{\mathbf{x}}_0 = \mathbf{x}_0 + \mathbf{V}_0^r(\mathbf{V}_0^r)^T\mathbf{x}_{\epsilon,0} - \mathbf{x}_{\epsilon,0}. \tag{12}$$

By contrast, CLC-NN controls all hidden states and results in a decreasing error as the number of layers T increases (c.f. Theorem 1). To quantitatively compare CLC-NN with reactive defense, we implement them with the same linear embedding functions and against all perturbations. In Table 3, CLC-NN outperforms reactive defense in almost all cases except that their performances are case-dependent on clean or randomly perturbed data.

7 Conclusion

We have proposed a close-loop control formulation to improve the robustness of neural networks. We have studied the embedding of state trajectory during forward propagation to define the optimal control objective function. The numerical experiments have shown that our method can improve the robustness of a trained neural network against various perturbations. We have provided an error estimation for the proposed method in the linear case. Our current implementation uses the Pontryagin's Maximum Principle and an online iterative algorithm to overcome the intractability of solving a dynamical programming. This online process adds extra inference time. In the future, we plan to show the theoretical analysis for the nonlinear embedding case.

REFERENCES

Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pp. 39–57. IEEE, 2017.

E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

Logan Engstrom, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 1(2):3, 2017.

Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

Jiequn Han, Qianxiao Li, et al. A mean-field optimal control formulation of deep learning. *arXiv preprint arXiv:1807.01083*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Alan Julian Izenman. Introduction to manifold learning. *Wiley Interdisciplinary Reviews: Computational Statistics*, 4(5):439–446, 2012.

Marc Khoury and Dylan Hadfield-Menell. On the geometry of adversarial examples. *arXiv preprint arXiv:1811.00525*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Donald E Kirk. *Optimal control theory: an introduction*. Springer, 1970.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.

Guan-Horng Liu, Tianrong Chen, and Evangelos A Theodorou. Differential dynamic programming neural optimizer. *arXiv preprint arXiv:2002.08809*, 2020a.

Guan-Horng Liu, Tianrong Chen, and Evangelos A Theodorou. A differential game theoretic neural optimizer for training residual networks. *arXiv preprint arXiv:2007.08880*, 2020b.

Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018.

Alexander Selvikvåg Lundervold and Arvid Lundervold. An overview of deep learning in medical imaging focusing on mri. *Zeitschrift für Medizinische Physik*, 29(2):102–127, 2019.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2574–2582, 2016.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pp. 3358–3369, 2019.

Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Nanyang Ye, Qianxiao Li, and Zhanxing Zhu. Amata: An annealing mechanism for adversarial training acceleration. 2019.

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In *Advances in Neural Information Processing Systems*, pp. 227–238, 2019.

A APPENDIX A ERROR ESTIMATION FOR THE PROPOSED CLC-NN

Preliminaries We define the performance index at time t as

$$J(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2}\|\mathbf{Q}_t(\mathbf{x}_t + \mathbf{u}_t)\|_2^2 + \frac{c}{2}\|\mathbf{u}_t\|_2^2, \tag{13}$$

where $\mathbf{Q}_t = \mathbf{I} - \mathbf{V}_t^r(\mathbf{V}_t^r)^T$, $\mathbf{V}_t^r$ is the linear projection matrix at time t with only its first r principle components corresponding to the largest r eigenvalues. The optimal feedback control is defined as

$$\mathbf{u}_t^*(\mathbf{x}_t) = \arg\min_{\mathbf{u}_t} J(\mathbf{x}_t, \mathbf{u}_t),$$

due to the linear system and quadratic performance index, the optimal feedback control admits an analytic solution by taking the gradient of performance index (Eq. (13)) and setting it to 0.

$$\begin{aligned}\nabla_{\mathbf{u}} J(\mathbf{x}_t, \mathbf{u}_t) &= \nabla_{\mathbf{u}}\left(\frac{1}{2}\|\mathbf{Q}_t(\mathbf{x}_t + \mathbf{u}_t)\|_2^2 + \frac{c}{2}\|\mathbf{u}_t\|_2^2\right), \\ &= \mathbf{u}_t^T\mathbf{Q}_t^T\mathbf{Q}_t\mathbf{x}_t + \mathbf{u}_t^T\mathbf{Q}_t^T\mathbf{Q}_t\mathbf{u}_t + c\cdot\mathbf{u}_t,\end{aligned}$$

which leads to the analytic solution of $\mathbf{u}_t^*(\mathbf{x}_t)$ as

$$\mathbf{u}_t^*(\mathbf{x}_t) = -(c\cdot\mathbf{I} + \mathbf{Q}_t^T\mathbf{Q}_t)^{-1}\mathbf{Q}_t^T\mathbf{Q}_t\mathbf{x}_t. \tag{14}$$

The above analytic control solution $\mathbf{u}_t^*$ optimizes the performance index instantly at time step t, the error measured by Eq. (13) for the dynamical programming solution $\overline{\mathbf{x}}_{\epsilon,t}$ must be smaller or equal to the state trajectory equipped with $\mathbf{u}_t^*$ define by Eq. (14), which gives a guaranteed upper bound for the error estimation of the dynamic programming solution.

We define the feedback gain matrix $\mathbf{K}_t = (c \cdot \mathbf{I} + \mathbf{Q}_t^T \mathbf{Q}_t)^{-1} \mathbf{Q}_t^T \mathbf{Q}_t$. Thus, the on-step optimal feedback control can be represented as $\mathbf{u}_t^* = -\mathbf{K}_t \mathbf{x}_t$.

The difference between the controlled system applied with perturbation at initial condition and the uncontrolled system without perturbation is shown

$$\begin{aligned} \overline{\mathbf{x}}_{\epsilon,t+1} - \mathbf{x}_{t+1} &= \boldsymbol{\theta}_t(\overline{\mathbf{x}}_{\epsilon,t} + \mathbf{u}_t - \mathbf{x}_t), \\ &= \boldsymbol{\theta}_t(\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{K}_t \overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t). \end{aligned} \tag{15}$$

The control objective is to minimize the state components that span the orthogonal complement of the data manifold ($\mathbf{I} - \mathbf{V}_t^r (\mathbf{V}_t^r)^T$), when the input data to feedback control only stays in the state manifold, such that $\|(\mathbf{I} - \mathbf{V}_t^r (\mathbf{V}_t^r)^T)\mathbf{x}_t\|_2^2 = 0$, the feedback control $\mathbf{K}_t \mathbf{x}_t = \mathbf{0}$. The state difference of Eq. (15) can be further shown by adding a $\mathbf{0}$ term of $(\boldsymbol{\theta}_t \mathbf{K}_t \mathbf{x}_t)$

$$\begin{aligned} \overline{\mathbf{x}}_{\epsilon,t+1} - \mathbf{x}_{t+1} &= \boldsymbol{\theta}_t(\mathbf{I} - \mathbf{K}_t)\overline{\mathbf{x}}_{\epsilon,t} - \boldsymbol{\theta}_t \mathbf{x}_t + \boldsymbol{\theta}_t \mathbf{K}_t \mathbf{x}_t, \\ &= \boldsymbol{\theta}_t(\mathbf{I} - \mathbf{K}_t)(\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t). \end{aligned} \tag{16}$$

In the following, we show a transformation on the control dynamic term $(\mathbf{I} - \mathbf{K}_t)$ based on its definition.

Lemma 1. *For $t \geq 0$, we have*

$$\mathbf{I} - \mathbf{K}_t = \alpha \cdot \mathbf{I} + (1 - \alpha) \cdot \mathbf{P}_t,$$

where $\mathbf{P}_t := \mathbf{V}_t^r (\mathbf{V}_t^r)^T$, which is the orthogonal projection onto $Z_{\|}^t$, and $\alpha := \frac{c}{1+c}$ such that $\alpha \in [0, 1]$.

Proof. Recall that $\mathbf{K}_t = (c \cdot \mathbf{I} + \mathbf{Q}_t^T \mathbf{Q}_t)^{-1} \mathbf{Q}_t^T \mathbf{Q}_t$, and $\mathbf{Q}_t = \mathbf{I} - \mathbf{V}_t^r (\mathbf{V}_t^r)^T$, $\mathbf{Q}_t$ can be diagonalized as following

$$\mathbf{Q}_t = \mathbf{V}_t \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \mathbf{V}_t^T,$$

where the first r diagonal elements have common value of 0 and the last $(d - r)$ diagonal elements have common value of 1. Furthermore, the feedback gain matrix $\mathbf{K}_t$ can be diagonalized as

$$\mathbf{K}_t = \mathbf{V}_t \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & \cdots & \frac{1}{1+c} & 0 \\ 0 & 0 & \cdots & 0 & \frac{1}{1+c} \end{bmatrix} \mathbf{V}_t^T,$$

where the last $(d - r)$ diagonal elements have common value of $\frac{1}{1+c}$. The control term $(\mathbf{I} - \mathbf{K}_t)$ thus can be represented as

$$\mathbf{I} - \mathbf{K}_t = \mathbf{V}_t \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ 0 & 0 & \cdots & \frac{c}{1+c} & 0 \\ 0 & 0 & \cdots & 0 & \frac{1}{1+c} \end{bmatrix} \mathbf{V}_t^T,$$

where the first r diagonal elements have common value of 1 and the last $(d - r)$ diagonal elements have common value of $\frac{c}{1+c}$. By denoting the projection of first r columns as $\mathbf{V}_t^r$ and last $(d - r)$

columns as $\hat{\mathbf{V}}_t^r$, it can be further shown as

$$\begin{aligned}\mathbf{I} - \mathbf{K}_t &= \mathbf{V}_t^r(\mathbf{V}_t^r)^T + \frac{c}{1+c}\big(\hat{\mathbf{V}}_t^r(\hat{\mathbf{V}}_t^r)^T\big),\\ &= \mathbf{P}_t + \alpha\big(\mathbf{I} - \mathbf{P}_t\big),\\ &= \alpha \cdot \mathbf{I} + (1-\alpha)\cdot \mathbf{P}_t.\end{aligned}$$

□

Oblique Projections. Let $\mathbf{P}$ be a linear operator on $\mathbb{R}^d$,

- We say that $\mathbf{P}$ is an projection if $\mathbf{P}^2 = \mathbf{P}$.
- $\mathbf{P}$ is an orthogonal projection if $\mathbf{P} = \mathbf{P}^T = \mathbf{P}^2$.
- If $\mathbf{P}^2 = \mathbf{P}$ but $\mathbf{P} \neq \mathbf{P}^T$, it is called an oblique projection.

Proposition 2. *For a projection* $\mathbf{P}$*,*

1. *If* $\mathbf{P}$ *is an orthogonal projection, then* $\|\mathbf{P}\|_2 = 1$.
2. *If* $\mathbf{P}$ *is an oblique projection, then* $\|\mathbf{P}\|_2 > 1$.
3. *If* $\mathbf{P}$*,* $\mathbf{Q}$ *are two projections such that* $range(\mathbf{P}) = range(\mathbf{Q})$*, then* $\mathbf{PQ} = \mathbf{Q}$ *and* $\mathbf{QP} = \mathbf{P}$.
4. *If* $\mathbf{P}$ *is a projection, then* $rank(\mathbf{P}) = Tr(\mathbf{P})$*. Furthermore, if* $\mathbf{P}$ *is an orthogonal projection, then* $rank(\mathbf{P}) = \|\mathbf{P}\|_F^2 = Tr(\mathbf{P}\mathbf{P}^{\mathbf{T}})$.

Define for $t \geq 0$

$$\begin{cases}\mathbf{P}_t^0 := \mathbf{P}_t,\\ \mathbf{P}_t^{(s+1)} := \boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s\boldsymbol{\theta}_{t-s-1}, \quad s = 0, 1, \ldots, t-1,\end{cases}$$

Lemma 3. *Let* $\mathbf{P}_t^s$ *be defined as above for* $0 \leq s \leq t$*. Then*

1. $\mathbf{P}_t^s$ *is a projection.*
2. $\mathbf{P}_t^s$ *is a projection onto* $Z_{\|}^{t-s}$*, i.e.* $range(\mathbf{P}_t^s) = Z_{\|}^{t-s}$.
3. $$\|\mathbf{P}_t^s\|_F^2 \leq \kappa(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\ldots\boldsymbol{\theta}_{t-s})^2 \cdot r,$$
   where $\kappa(\mathbf{A})$ *is the condition number of* $\mathbf{A}$*, i.e.* $\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \cdot \|\mathbf{A}^{-1}\|_2$*, and* $r = rank(Z_{\|}^0) = rank(Z_{\|}^1) = \ldots = rank(Z_{\|}^t)$.

Proof.

1. We prove it by induction on s for each t. For $s = 0$, $\mathbf{P}_t^0 = \mathbf{P}_t$, which is a projection by its definition. Suppose it is true for s such that $\mathbf{P}_t^s = \mathbf{P}_t^s\mathbf{P}_t^s$, then for $(s+1)$,
$$\begin{aligned}(\mathbf{P}_t^{s+1})^2 &= \big(\boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s\boldsymbol{\theta}_{t-s-1}\big)^2,\\ &= \boldsymbol{\theta}_{t-s-1}^{-1}\big(\mathbf{P}_t^s\big)^2\boldsymbol{\theta}_{t-s-1},\\ &= \boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s\boldsymbol{\theta}_{t-s-1},\\ &= \mathbf{P}_t^{s+1}.\end{aligned}$$

2. We prove it by induction on s for each t. For $s = 0$, $\mathbf{P}_t^0 = \mathbf{P}_t$, which is the orthogonal projection onto $Z_{\|}^t$. Suppose that it is true for s such that $\mathbf{P}_t^s$ is a projection onto $Z_{\|}^{t-s}$, then for $(s+1)$, $\mathbf{P}_t^{s+1} = \boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s\boldsymbol{\theta}_{t-s-1}$, which implies
$$\begin{aligned}range(\mathbf{P}_t^{s+1}) &= range(\boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s),\\ &= \{\boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{x} : \mathbf{x} \in Z_{\|}^{t-s}\},\\ &= Z_{\|}^{t-s-1}.\end{aligned}$$

3. We use the inequalities $\|\mathbf{AB}\|_F \leq \|\mathbf{A}\|_2\|\mathbf{B}\|_F$, and $\|\mathbf{AB}\|_F \leq \|\mathbf{A}\|_F\|\mathbf{B}\|_2$. By the definition of $\mathbf{P}_t^s$,
$$\mathbf{P}_t^s = \big(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_{t-s}\big)^{-1}\mathbf{P}_t^0\big(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_{t-s}\big),$$
we have the following
$$\begin{aligned}\|\mathbf{P}_t^s\|_F^2 &\leq \|\big(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_{t-s}\big)^{-1}\|_2^2 \cdot \|\big(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_{t-s}\big)\|_2^2 \cdot \|\mathbf{P}_t^0\|_F^2, \\ &\leq \kappa(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_{t-s})^2 \cdot r, && \text{Lemma 2(4).}\end{aligned}$$

□

The following Lemma uses the concept of oblique projection to show a recursive relationship to project any t^{th} state space of Eq. (16) back to the input data space.

Lemma 4. *Define for* $0 \leq s \leq t$,
$$\mathbf{G}_t^s := \alpha \cdot \mathbf{I} + (1-\alpha)\mathbf{P}_t^s.$$
Then, Eq. (16) can be written as
$$\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t = (\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1}\mathbf{G}_{t-2}^{t-2}\cdots\mathbf{G}_0^0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0), \quad t \geq 1.$$

Proof. We prove it by induction on t. For $t = 1$, by the definition of $\mathbf{G}_t^s$ and transformation from Lemma 1,
$$\begin{aligned}\overline{\mathbf{x}}_{\epsilon,1} - \mathbf{x}_1 &= \boldsymbol{\theta}_0(\mathbf{I} - \mathbf{K}_0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0), && \text{Eq. (16),} \\ &= \boldsymbol{\theta}_0(\alpha \cdot \mathbf{I} + (1-\alpha)\cdot\mathbf{P}_0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0), && \text{Lemma 1,} \\ &= \boldsymbol{\theta}_0\mathbf{G}_0^0(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0).\end{aligned}$$
Suppose that it is true for $(\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t)$, by using Eq. (16) and Lemma 1, we have
$$\begin{aligned}\overline{\mathbf{x}}_{\epsilon,t+1} - \mathbf{x}_{t+1} &= \boldsymbol{\theta}_t(\mathbf{I} - \mathbf{K}_t)(\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t), && \text{Eq. (16),} \\ &= \boldsymbol{\theta}_t(\alpha \cdot \mathbf{I} - (1-\alpha)\cdot\mathbf{P}_t)(\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t), && \text{Lemma 1,} \\ &= \boldsymbol{\theta}_t\mathbf{G}_t^0(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1}\mathbf{G}_{t-2}^{t-2}\cdots\mathbf{G}_0^0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0). && (17)\end{aligned}$$
Recall the definitions of $\mathbf{P}_t^{(s+1)} := \boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s\boldsymbol{\theta}_{t-s-1}$, and $\mathbf{G}_t^s := \alpha \cdot \mathbf{I} + (1-\alpha)\mathbf{P}_t^s$, we have the following
$$\begin{aligned}\mathbf{G}_t^{s+1} &= \alpha \cdot \mathbf{I} + (1-\alpha)\cdot\mathbf{P}_t^{(s+1)}, \\ &= \alpha \cdot \mathbf{I} + (1-\alpha)\cdot\boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{P}_t^s\boldsymbol{\theta}_{t-s-1}, \\ &= \boldsymbol{\theta}_{t-s-1}^{-1}\big(\alpha \cdot \mathbf{I} + (1-\alpha)\cdot\mathbf{P}_t^s\big)\boldsymbol{\theta}_{t-s-1}, \\ &= \boldsymbol{\theta}_{t-s-1}^{-1}\mathbf{G}_t^s\boldsymbol{\theta}_{t-s-1},\end{aligned}$$
which results in the equality for the oblique projections. Furthermore,
$$\boldsymbol{\theta}_{t-s-1}\mathbf{G}_t^{(s+1)} = \mathbf{G}_t^s\boldsymbol{\theta}_{t-s-1}.$$
Applying the above to Eq. (17) results in
$$\begin{aligned}\overline{\mathbf{x}}_{\epsilon,t+1} - \mathbf{x}_{t+1} &= \boldsymbol{\theta}_t\mathbf{G}_t^0(\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1}\mathbf{G}_{t-2}^{t-2}\cdots\mathbf{G}_0^0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0), \\ &= (\boldsymbol{\theta}_t\boldsymbol{\theta}_{t-1})\mathbf{G}_t^1(\boldsymbol{\theta}_{t-2}\boldsymbol{\theta}_{t-3}\cdots\boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1}\mathbf{G}_{t-2}^{t-2}\cdots\mathbf{G}_0^0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0), \\ &= (\boldsymbol{\theta}_t\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2})\mathbf{G}_t^2(\boldsymbol{\theta}_{t-3}\boldsymbol{\theta}_{t-4}\cdots\boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1}\mathbf{G}_{t-2}^{t-2}\cdots\mathbf{G}_0^0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0), \\ &= (\boldsymbol{\theta}_t\boldsymbol{\theta}_{t-1}\cdots\boldsymbol{\theta}_0)(\mathbf{G}_t^t\mathbf{G}_{t-1}^{t-1}\cdots\mathbf{G}_0^0)(\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0).\end{aligned}$$

□

Lemma 5. *Let*
$$\mathbf{F}_t := \mathbf{G}_{t-1}^{(t-1)}\mathbf{G}_{t-2}^{(t-2)}\cdots\mathbf{G}_0^0, \quad t \geq 1.$$
Then,
$$\mathbf{F}_t = \alpha^t \cdot \mathbf{I} + (1-\alpha)\sum_{s=0}^{t-1}\alpha^s\mathbf{P}_s^s.$$

Proof. We prove it by induction on t. Recall the definition of $\mathbf{G}_t^s := \alpha \cdot \mathbf{I} + (1-\alpha) \cdot \mathbf{P}_t^s$. When $t = 1$,

$$\mathbf{F}_1 = \mathbf{G}_0^0 = \alpha \cdot \mathbf{I} + (1-\alpha) \cdot \mathbf{P}_0^0.$$

Suppose that it is true for t such that

$$\mathbf{F}_t = \mathbf{G}_{t-1}^{(t-1)} \mathbf{G}_{t-2}^{(t-2)} \cdots \mathbf{G}_0^0 = \alpha^t \cdot \mathbf{I} + (1-\alpha) \sum_{s=0}^{t-1} \alpha^s \mathbf{P}_s^s,$$

for $(t+1)$,

$$\begin{aligned}
\mathbf{F}_{t+1} &= \mathbf{G}_t^t F(t), \\
&= (\alpha \cdot \mathbf{I} + (1-\alpha) \cdot \mathbf{P}_t^t) \mathbf{F}_t, \\
&= (\alpha \cdot \mathbf{I} + (1-\alpha) \cdot \mathbf{P}_t^t)(\alpha^t \cdot \mathbf{I} + (1-\alpha) \sum_{s=0}^{t-1} \alpha^s \mathbf{P}_s^s), \\
&= \alpha^{t+1} \cdot \mathbf{I} + \alpha^t (1-\alpha) \mathbf{P}_t^t + (1-\alpha)^2 \sum_{s=0}^{t-1} \alpha^s \cdot \mathbf{P}_t^t \mathbf{P}_s^s + \alpha(1-\alpha) \sum_{s=0}^{t-1} \alpha^s \cdot \mathbf{P}_s^s.
\end{aligned}$$

Recall Lemma 3, $range(\mathbf{P}_t^t) = range(\mathbf{P}_s^s) = Z_{\|}^0$. According to Proposition 2 (3), $\mathbf{P}_t^t \mathbf{P}_s^s = \mathbf{P}_s^s$. Hence,

$$\begin{aligned}
\mathbf{F}_{t+1} &= \alpha^{t+1} \cdot \mathbf{I} + \alpha^t (1-\alpha) \cdot \mathbf{P}_t^t + (1-\alpha) \sum_{s=0}^{t-1} \alpha^s \cdot \mathbf{P}_s^s, \\
&= \alpha^{t+1} \cdot \mathbf{I} + (1-\alpha) \sum_{s=0}^{t} \alpha^s \cdot \mathbf{P}_s^s.
\end{aligned}$$

□

Theorem 1. *For $t \geq 1$, we have the error estimation*

$$\mathbb{E}\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 \leq \epsilon^2 \|\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2} \cdots \boldsymbol{\theta}_0\|_2^2 \left(\alpha^{2t} + \frac{r}{d}\Big(2\alpha^t(1-\alpha^t) + \beta_t(1-\alpha^t)^2\Big)\right),$$

where $\beta_t = \prod_{s=0}^{t-1} \kappa(\boldsymbol{\theta}_s) = \prod_{s=0}^{t-1} \|\boldsymbol{\theta}_s\|_2 \|\boldsymbol{\theta}_s^{-1}\|_2$, and $\alpha = \frac{c}{1+c}$, c represents the control regularization. In particular, the equality

$$\mathbb{E}\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 = \epsilon^2 \left(\alpha^{2t} + \frac{r}{d}(1-\alpha^{2t}) \right)$$

Proof. Using the fact that $\overline{\mathbf{x}}_{\epsilon,0} - \mathbf{x}_0 = \mathbf{z}$, and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \frac{\epsilon^2}{d}\mathbf{I})$, and Lemma 4,

$$\begin{aligned}
\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 &= Tr\left[(\boldsymbol{\theta}_{t-1} \cdots \boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1} \cdots \mathbf{G}_0^0) \mathbf{z}\mathbf{z}^T (\mathbf{G}_{t-1}^{t-1} \cdots \mathbf{G}_0^0)^T (\boldsymbol{\theta}_{t-1} \cdots \boldsymbol{\theta}_0)^T \right], \\
&= Tr\left[\mathbf{z}\mathbf{z}^T (\mathbf{G}_{t-1}^{t-1} \cdots \mathbf{G}_0^0)^T (\boldsymbol{\theta}_{t-1} \cdots \boldsymbol{\theta}_0)^T (\boldsymbol{\theta}_{t-1} \cdots \boldsymbol{\theta}_0)(\mathbf{G}_{t-1}^{t-1} \cdots \mathbf{G}_0^0) \right], \\
&\leq \frac{\epsilon^2}{d} \|\boldsymbol{\theta}_{t-1} \cdots \boldsymbol{\theta}_0\|_2^2 \cdot \|\mathbf{G}_{t-1}^{t-1} \cdots \mathbf{G}_0^0\|_F^2,
\end{aligned}$$

By using Lemma 5,

$$\begin{aligned}
&\|\mathbf{G}_{t-1}^{t-1} \cdots \mathbf{G}_0^0\|_F^2 \\
&= \|\mathbf{F}_t\|_F^2, \\
&= Tr[\mathbf{F}_t \mathbf{F}_t^T], \\
&= Tr\left[\big(\alpha^t \cdot \mathbf{I} + (1-\alpha) \sum_{s=0}^{t-1} \alpha^s \mathbf{P}_s^s\big) \big(\alpha^t \cdot \mathbf{I} + (1-\alpha) \sum_{s=0}^{t-1} \alpha^s \mathbf{P}_s^s\big)^T \right], \\
&= \alpha^{2t} Tr(I) + \alpha^t (1-\alpha) \sum_{s=0}^{t-1} \alpha^s Tr\left[\mathbf{P}_s^s + (\mathbf{P}_s^s)^T \right] + (1-\alpha)^2 \sum_{s=0}^{t-1} \sum_{q=0}^{t-1} \alpha^s \alpha^q Tr\left[\mathbf{P}_s^s (\mathbf{P}_q^q)^T \right].
\end{aligned}$$

Notice that $Tr(\mathbf{I}) = d$. Recall Proposition 2 (4) and Lemma 3 (2), $Tr\big[\mathbf{P}_s^s\big] = Tr\big[(\mathbf{P}_s^s)^T\big] = rank(\mathbf{P}_s^s) = r$. By the Cauchy–Schwarz inequality and Lemma 3 (3),

$$\begin{aligned} Tr\left[\mathbf{P}_s^s(\mathbf{P}_q^q)^T\right] &\leq \|\mathbf{P}_s^s\|_F \cdot \|\mathbf{P}_q^q\|_F, \\ &\leq \kappa(\boldsymbol{\theta}_{s-1}\cdots\boldsymbol{\theta}_{s-s})\sqrt{r}\kappa(\boldsymbol{\theta}_{q-1}\cdots\boldsymbol{\theta}_{q-q})\sqrt{r}, \\ &\leq \left(\prod_{k=0}^{t-1}\kappa(\boldsymbol{\theta}_k)^2\right)\cdot r. \end{aligned}$$

Hence, we have

$$\begin{aligned} \|\mathbf{G}_{t-1}^{t-1}\cdots\mathbf{G}_0^0\|_F^2 &\leq \alpha^{2t}d + 2\alpha^t(1-\alpha)r\sum_{s=0}^{t-1}\alpha^s + \left(\prod_{k=0}^{t-1}\kappa(\boldsymbol{\theta}_k)^2\right)\cdot r(1-\alpha)^2\sum_{s=0}^{t-1}\sum_{q=0}^{t-1}\alpha^s\alpha^q, \\ &= \alpha^{2t}d + 2\alpha^t(1-\alpha^t)r + (1-\alpha^t)^2\left(\prod_{k=0}^{t-1}\kappa(\boldsymbol{\theta}_k)^2\right)r. \end{aligned}$$

Hence,

$$\mathbb{E}\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 \leq \epsilon^2\|\boldsymbol{\theta}_{t-1}\boldsymbol{\theta}_{t-2}\cdots\boldsymbol{\theta}_0\|_2^2\left(\alpha^{2t} + \frac{r}{d}\Big(2\alpha^t(1-\alpha^t) + \beta_t(1-\alpha^t)^2\Big)\right),$$

where $\beta_t = \prod_{s=0}^{t-1}\kappa(\boldsymbol{\theta}_s)^2 = \prod_{s=0}^{t-1}\|\boldsymbol{\theta}_s\|_2^2\|\boldsymbol{\theta}_s^{-1}\|_2^2$, and $\alpha = \frac{c}{1+c}$. Specifically, the inequality becomes equality if $\boldsymbol{\theta}_0, \cdots, \boldsymbol{\theta}_{t-1}$ are orthogonal matrices, in which case $\kappa(\boldsymbol{\theta}_t) = 1$, and $\beta_t = 1$. Thus,

$$\begin{aligned} \mathbb{E}\|\overline{\mathbf{x}}_{\epsilon,t} - \mathbf{x}_t\|_2^2 &= \epsilon^2\left(\alpha^{2t} + \frac{r}{d}\Big(2\alpha^t(1-\alpha^t) + (1-\alpha^t)^2\Big)\right), \\ &= \epsilon^2\left(\alpha^{2t} + \frac{r}{d}(1-\alpha^{2t})\right). \end{aligned}$$

□

B Appendix B Details of Experimental Setting

B.1 Network Configurations

Since the proposed CLC-NN optimizes the entire state trajectory, it is important to have a relatively smooth state trajectory, in which case, when the reconstruction loss $\|\mathcal{E}_t(\mathbf{x}_t) - \mathbf{x}_t\|_2^2$ at layer t is small, the reconstruction losses at its adjacent layers should be small. For this reason, we use residual neural network (He et al., 2016) as network candidate to retain smoother dynamics. The configuration of the residual neural network used for both CIFAR-10 and CIFAR-100 is shown in Tab. 4.

Based on the configuration of residual neural network shown in Tab. 4, we construct 4 embedding functions applied at input space, outputs of initial layer, residual block 1 and residual block 2. The output of residual block 3 is embedded with a linear orthogonal projection. We randomly select 5000 clean training data to collect state trajectories at all 5 locations.

- For the linear orthogonal projections: we apply the Principle Component Analysis on each of the state collections. We retain the first r columns of the resulted basis, such that $r = \arg\min\{i : \frac{\lambda_1+\ldots+\lambda_i}{\lambda_1\ldots+\lambda_d} \geq 1-\delta\}$, where $\delta = 0.1$.
- For the nonlinear embedding: we train 4 convolutional auto-encoders for the input space, outputs of the initial layer and residual blocks $1, 2$. All of the embedding functions are trained individually. We adopt shallow convolutional auto-encoder structure to gain fast inference speed, in which case, CLC-NN equipped with linear embedding often outperform the nonlinear embedding as shown in Tab. 1. The configuration of all 4 convolutional auto-encoders are shown in Tab. 5.

Table 4: ResNet for Both CIFAR-10 and CIFAR-100

Structure	Configuration
Initial Layer	Conv2d (input channel = 3, output channel = 16, kernel size = 3×3), BatchNorm2d(channel = 16), Relu()
Residual Block 1	{Conv2d (input channel = 16, output channel = 16, kernel size = 3×3), BatchNorm2d(channel = 16), Relu(), Shortcut()} $\times 6$
Residual Block 2	Conv2d (input channel = 16, output channel = 32, kernel size = 3×3), BatchNorm2d(channel = 16), Relu(), Shortcut(), {Conv2d (input channel = 32, output channel = 32, kernel size = 3×3), BatchNorm2d(channel = 32), Relu(), Shortcut()} $\times 5$
Residual Block 3	Conv2d (input channel = 32, output channel = 64, kernel size = 3×3), BatchNorm2d(channel = 64), Relu(), Shortcut(), {Conv2d (input channel = 64, output channel = 64, kernel size = 3×3), BatchNorm2d(channel = 64), Relu(), Shortcut()} $\times 5$
Final Layer	Fully Connected $(64, 10)$

Table 5: Convolutional Auto-Encoders

Structure	Configuration
Encoder	Conv2d (input channel = c_1, output channel = c_2, kernel size = 4×4, stride = 2×2, padding = 1×1), ELU(alpha=1), BatchNorm2d(channel = c_2), Conv2d (input channel = c_2, output channel = c_3, kernel size = 4×4, stride = 2×2, padding = 1×1), ELU(alpha=1).
Decoder	ConvTranspose2d (input channel = c_3, output channel = c_2, kernel size = 4×4, stride = 2×2, padding = 1×1), ELU(alpha=1), ConvTranspose2d (input channel = c_2, output channel = c_1, kernel size = 4×4, stride = 2×2, padding = 1×1),

Auto-encoder Index	0	1	2	3
Channel Dimensions $[c_1, c_2, c_3]$	$[3, 18, 36]$	$[16, 36, 72]$	$[16, 36, 72]$	$[32, 128, 256]$

B.2 Perturbations and Defensive Training

In this section, we show details about the perturbations and robust networks that have been considered in this work. For the adversarial training objective function,

$$\min_{\boldsymbol{\theta} \in \Theta} \max_{\mathbf{x}_{\epsilon,0} = \Delta(\mathbf{x}_0, \epsilon)} \mathop{\mathbb{E}}_{(\mathbf{x}_0, \mathbf{y}) \sim \mathcal{D}} [(1 - \lambda) \cdot \Phi_i(\mathbf{x}_{\epsilon,T}, \mathbf{y}, \boldsymbol{\theta}) + \lambda \cdot \Phi_i(\mathbf{x}_T, \mathbf{y}, \boldsymbol{\theta})],$$

where $\Delta(\mathbf{x}_0, \epsilon)$ generates a perturbed data from given input $\mathbf{x}_0$ within the range of ϵ. λ balances between standard accuracy and robustness. We choose $\lambda = 0.5$ in all adversarial training.

For robust networks, we consider both perturbation agnostic and non-agnostic methods. For the perturbation agnostic adversarial training algorithms equipped $\Delta(\mathbf{x}_0, \epsilon)$, the resulted network that is the most robust against the $\Delta(\mathbf{x}_0, \epsilon)$ perturbation. On the contrary, perturbation non-agnostic robust training methods are often robust against many types of perturbations.

- Adversarial training with the fast gradient sign method (FGSM) (Goodfellow et al., 2014) considers perturbed data as follows.

$$\mathbf{x}_{\epsilon,0} = \mathbf{x}_0 + sign(\nabla_{\mathbf{x}_0} Phi(\mathbf{x}_T, \mathbf{y})), \quad (\mathbf{x}_0, y) \sim \mathcal{D},$$

  where $sign(\cdot)$ outputs the sign of the input. In which case, FGSM considers the worse case within the range of ϵ along the gradient $\nabla_{\mathbf{x}_0} Phi(\mathbf{x}_T, \mathbf{y})$ increasing direction. Due to the worse case consideration, it does not scale well for deep networks, for this reason, we adversarially train the network with FGSM with $\epsilon = 4$, which is half of the maximum perturbation considered in this paper.
- Adversarial training with the project gradient descent (PGD) (Madry et al., 2017) generates adversarial data by iteratively run FGSM with small step size, which results in stronger perturbations compared with FGSM within the same range ϵ. We use 7-step of $\epsilon = 2$ to generate adversarial data for robust training.
- The free adversarial training (Free) (Shafahi et al., 2019) does not utilize any perturbation information $\Delta(\mathbf{x}_0, \epsilon)$. It optimizes an noise vector through each data according to the direction of increasing gradient $\nabla_{\mathbf{x}_0} \Phi(\mathbf{x}_T, \mathbf{y})$. It have been observed that by incorporating such noise with input data, the resulted network achieves robustness against various perturbations (Shafahi et al., 2019).

For Perturbations, we consider the maximum range of $\epsilon = 2, 4, 8$ to test the performance the network robustness against both strong and weak perturbations. For this work, we test network robustness with uniformly distributed random noise, FGSM (Goodfellow et al., 2014), 20-step of PGD (Madry et al., 2017) and the CW attack (Carlini & Wagner, 2017).

B.3 Online Optimization

Optimization Methods. we use Adam (Kingma & Ba, 2014) to maximize the Hamiltonian Eq. (9) with the maximum number of iterations from Alg. 1 to be 20. Our proposed close-loop control formulation improves robustness of a given neural network based on the input and state embedding only. However, due to the optimization based solution of Alg. 1, under maximum of 10 iterations, we choose Adam (Kingma & Ba, 2014) to maximize the Hamiltonian Eq. (9) with adjusted learning rate against different perturbations, in which case, the numerical implementation of the proposed method is no longer perturbation agnostic.