

# Control Theoretic Approach to Fine-Tuning and Transfer Learning

Erkan Bayram<sup>1</sup>, Shenyu Liu<sup>2</sup>, Mohamed-Ali Belabbas<sup>1</sup>, and Tamer Başar<sup>1</sup>

<sup>1</sup> Coordinated Science Laboratory, University of Illinois at Urbana-Champaign  
Urbana, IL 61801

(ebayram2,belabbas,basar1)@illinois.edu, \*

<sup>2</sup> The School of Automation, Beijing Institute of Technology, Beijing 100089, China  
shenyuliu@bit.edu.cn

**Abstract.** Given a training set in the form of a paired  $(\mathcal{X}, \mathcal{Y})$ , we say that the control system  $\dot{x} = f(x, u)$  has learned the paired set via the control  $u^*$  if the system steers each point of  $\mathcal{X}$  to its corresponding target in  $\mathcal{Y}$ . If the training set is expanded, most existing methods for finding a new control  $u^*$  require starting from scratch, resulting in a quadratic increase in complexity with the number of points. To overcome this limitation, we introduce the concept of *tuning without forgetting*. We develop an *iterative algorithm* to tune the control  $u^*$  when the training set expands, whereby points already in the paired set are still matched, and new training samples are learned. At each update of our method, the control  $u^*$  is projected onto the kernel of the end-point mapping generated by the controlled dynamics at the learned samples. It ensures keeping the end-points for the previously learned samples constant while iteratively learning additional samples.

**Keywords:** Control of ensemble of points, geometric control, fine-tuning, control for learning

## 1 Introduction

In recent years, neural networks and other data-driven models have grown significantly in size and capability, often requiring extensive computation for training. To address the challenge of retraining models for additional tasks or new data, methods like *transfer learning* and *fine-tuning* have become popular. These methods allow models to be adapted for new tasks or updated with new examples without complete retraining.

In this paper, we show that control theory provides a natural framework, as well as efficient algorithms, to perform these tasks. Mathematically, fine-tuning and transfer learning are often treated similarly, as these methods are blind to the nature of the examples they are fed [1]. This also applies to the algorithms we propose here, but we focus on fine-tuning for the sake of exposition.

---

\* Research of UIUC authors was supported in part by the ARO Grant W911NF-24-1-0085, NSF-CCF 2106358, ARO W911NF-24-1-0105 and AFOSR FA9550-20-1-0333.

The models we consider are controlled differential equations of the type

$$\dot{x} = f(x, u) \quad (1)$$

(see Section 2 for their use in learning and the proper definitions). For a given paired training set  $(\mathcal{X}, \mathcal{Y})$  with finite cardinality  $q$ , the learning task is to find a control  $u^*$  so that the flow  $\varphi_t(u^*, \cdot)$  generated by (1) satisfies

$$R(\varphi_T(u^*, x^i)) = y^i, \text{ for all } (x^i, y^i) \in (\mathcal{X}, \mathcal{Y})$$

for some finite  $T \geq 0$  and a given fixed readout map  $R$ . This task is also known as the control of an ensemble of points [2]. Thus, we interchangeably refer to the training sample set  $\mathcal{X}$  as the input ensemble and the training label set  $\mathcal{Y}$  as the output ensemble in line with the usage in the control theory literature.

In this work, our goal is to adapt a control function  $u^*$ , trained on the original set, to an expanded training set. This aligns with the main goal of fine-tuning, where parameters  $u^*$  are adapted by further training on an expanded dataset.

One of the main issues addressed by fine-tuning is to avoid a loss of performance on previous tasks while specializing the model to a new dataset. This problem is also known as *catastrophic forgetting* in the continual learning (incremental learning) literature.

Common approaches involve learning a sequence of tasks one by one and aiming to match the performance as if they were observed simultaneously [3]. However, in continual learning, access to previously observed data points is typically not available. When such access exists, it is usually discussed in the literature on robust fine-tuning. Similar approaches to our work can be found in optimization-based continual learning methods that incorporate memory [4].

In the control theory literature, finding a control  $u$  that steers each point in  $\mathcal{X}$  with the cardinality  $q$  to the associated labels in  $\mathcal{Y}$  is done by concatenating all points in  $\mathcal{X}$  in a state of higher dimension, and replicate the controlled dynamics  $q$  times [2]. Then, this approach requires solving a controlled ordinary differential equation (CODE) for the  $q$ -tuple system. We call this method the *q-folded method*, details of which will be discussed later in the paper. This approach has two practical shortcomings:

- **Tuning Ability:** If the training set is augmented with additional samples, it is necessary to apply the  $q$ -folded method on the new set from scratch.
- **Scalability:** The  $q$ -folded method has a complexity of  $\mathcal{O}(n^2 q^2 N)$  per iteration with  $N$  time-step discretization. However, our method has a complexity of  $\mathcal{O}(n^2 q N)$  per iteration with  $N$  time-step discretization. Therefore, as the size of the training set,  $q$ , grows, the complexity of the  $q$ -folded method increases quadratically, leading to scalability issues while the complexity of our method increases linearly.

We introduce here a novel fine-tuning method. It is based on the hereby introduced concept of *tuning without forgetting*  $\mathcal{X}$ . The method consists of an iterative algorithm in which each update on the control function  $u$ , denoted by  $\delta u$ , is selected as a projection of the gradient of the given cost functional onto the

kernel of end-point mapping generated by the controlled dynamics. It ensures keeping the end-points for the previously steered points constant up to the first order while simultaneously steering the new points of the training set to their associated labels.

Our work thus contributes to the tuning ability and scalability of the methods using controlled dynamical systems for supervised learning and addresses the challenges stemming from training set updates. We will compare our proposed method to an existing fine-tuning method in a computational example.

Furthermore, as our method iteratively learns from samples  $(\mathcal{X}, \mathcal{Y})$ , we do not need to have a control system of dimension scaling with  $q$ . This is particularly important since, as we mentioned earlier, this scaling causes computational issues, but it can additionally run into memory capacity issues as well. These features make our approach a preferred alternative to the existing  $q$ -folded method in the control of an ensemble of points.

**Related Works:** A popular idea is to restrict the distance between the updated parameters and the original parameters [1] to avoid a loss of performance on the previous task while specializing the model to a new dataset. The works [5,6] propose to use Euclidean norm and the matrix absolute row sum matrix norm (MARS) regularization to restrict the distance of tuned parameters from the reference parameters. The work [7] proposes a trainable projected gradient method (TPGM) for fine-tuning to automatically learn the distance constraints for each layer.

Other optimization-based approaches in the continual learning literature include [4], which proposes the so-called “gradient episodic memory (GEM)”, that performs an update in weight in the direction of gradient of loss function if it has negative inner product with the gradient of loss for the previous samples. Furthermore, the works [8] proposes Orthogonal Gradient Descent (OGD) that stores the gradients for the initially observed points and rectifies the current gradient to be orthogonal to them. The study [9] introduces an algorithm called Adam-NSCL, which sequentially optimizes network parameters in the null space of previous tasks. This null space is approximated by applying singular value decomposition to the covariance matrix of all input features of previous tasks for each linear layer.

The dynamical system approach, which we adopt here, has been used in the past in the learning literature. For example, the studies [10,11,12] consider a continuous flow and introduce neural ordinary differential equations (NODE). The works presented in [13,14] focus on the universal approximation capabilities of deep residual neural networks from a geometric control viewpoint. Our work falls within the scope of these two pieces of literature.

## 2 Preliminaries

In this section, we introduce the problem and provide the preliminaries for the iterative algorithm. Consider the paired sets  $(\mathcal{X}, \mathcal{Y}) = \{(x^i, y^i)\}_{i=1}^q$ , where the  $x^i \in \mathbb{R}^n$ ’s are called initial points so that they are in a connected Riemannian

manifold  $\mathcal{M}$ . and the associated  $y^i \in \mathbb{R}^{n_o}$ 's are target points (labels). We assume that the elements of the input ensemble (training sample set)  $\mathcal{X}$  are pairwise distinct, i.e.,  $x^i \neq x^j$  for  $i \neq j$  and Let  $\mathcal{I}$  be an index set labelling the entries of the ensembles. We take the system:

$$\dot{x}(t) = f(u(t)x(t)), u \in L_\infty([0, T], \mathbb{R}^{\bar{n} \times \bar{n}}), \quad (2)$$

where  $x(t) \in \mathbb{R}^{\bar{n}}$  is the state vector at time  $t$  and  $f(\cdot)$  is a smooth vector field on  $\mathbb{R}^{\bar{n}}$ . Let  $E : x \in \mathbb{R}^n \mapsto \bar{x} \in \mathbb{R}^{\bar{n}}$  be an injective function with  $\bar{n} \geq n$ , called the uplift function. Let  $R : \mathbb{R}^{\bar{n}} \rightarrow \mathbb{R}^{n_o}$  be a given function, called the *readout map* so that the Jacobian of  $R$  is of full row rank. Note that both functions  $E(\cdot)$  and  $R(\cdot)$  are independent of the control  $u$ .

We note that we have adopted a model that is continuous in time for theoretical analysis. However, controlled dynamical systems can be viewed in the context of neural networks with residual connections. When using a model with residual connections, such as a neural network with infinitely many layers, it can be represented by a controlled dynamical system in the form of (2) [10,15]. For example, if we select  $f(\cdot) = \tanh(\cdot)$ , the function is applied elementwise to the product  $u(t)x(t)$ , resulting in  $\dot{x}(t) = \tanh(u(t)x(t))$ . If our numerical approach uses a discretization of 10 time steps, it is equivalent to a 10-layer deep residual network with a  $\tanh(\cdot)$  activation function. However, the depth of the network affects the discretization error. We next define the memorization property for a dynamical system.

**Definition 1 (Memorization Property).** *Assume that a paired set  $(\mathcal{X}, \mathcal{Y})$ , a fixed readout map  $R$  and an up-lift function  $E$  are given. The control  $u$  is said to have memorized the ensemble  $(\mathcal{X}, \mathcal{Y})$  for the model  $\dot{x}(t) = f(x(t), u(t))$  if the following holds for a finite  $T \geq 0$ :*

$$R(\varphi_T(u, E(x^i))) = y^i, \forall x^i \in \mathcal{X},$$

where the flow  $\varphi_t(u, \cdot)$  is generated by the system at the control function  $u$ .

We suppress the subscript  $T$  in the notation at  $t = T$  for simplicity. We call  $R(\varphi(u, E(\cdot)))$  the *end-point mapping*. that is, we have the following end-point mapping for a given  $u$  at  $x^i$ :

$$x^i \in \mathbb{R}^n \xrightarrow{E} \bar{x}^i \in \mathbb{R}^{\bar{n}} \xrightarrow{\varphi^T(u, \cdot)} \bar{y}^i \in \mathbb{R}^{\bar{n}} \xrightarrow{R} y^i \in \mathbb{R}^{n_o}$$

In other words, the dynamical system has memorized the ensemble if the end-point mapping maps each  $x^i \in \mathbb{R}^n$  to the corresponding  $y^i \in \mathbb{R}^{n_o}$ . The learning problem turns into a “multi-motion planning” problem (in which we look for a control function  $u$  that steers all  $\bar{x}^i$  in  $\mathcal{X}$  to the corresponding  $\bar{y}^i$ ). One can also see that, from the deep learning perspective, the memorization property is equivalent to the universal interpolation property of neural networks [14].

For convenience, we assume that  $E$  is the identity function, meaning  $\bar{n} = n$ , but our result holds for any injective up-lift function. Thus, we interchangeably use  $x^i$  and  $E(x^i)$  for any  $x^i \in \mathcal{X}$ . If  $n = n_o$  and  $R$  is the identity map, we call the

problem the control of an ensemble of points with *fixed end-points*. Otherwise, we call the problem the control of an ensemble of points with *partially constrained end-points*. In this work, we consider the latter. For the sake of simplicity, we let the map  $R$  be the orthogonal projection of a point in  $\mathbb{R}^n$  onto the subspace  $\mathbb{R}^{n_o}$ ; precisely,  $R : x \in \mathbb{R}^n \mapsto Cx \in \mathbb{R}^{n_o}$  where  $C = [0_{n_o \times n - n_o} \ I_{n_o \times n_o}] \in \mathbb{R}^{n_o \times n}$ . We define per-sample cost functional  $\mathcal{J}^i(u)$  for a given point  $x^i$  as follows:

$$\mathcal{J}^i(u) = \frac{1}{2} \|C\varphi(u, x^i) - y^i\|^2$$

Then, the cost-functional for the control of an ensemble of points with partially constrained end-points including regularization is defined as follows:

$$\mathcal{J}(u, \mathcal{X}) := \sum_{i=1}^q \|C\varphi(u, x^i) - y^i\|^2 + \lambda \int_0^T \|u(\tau)\|^2 d\tau \quad (3)$$

where  $\lambda$  is some regularization coefficient. If we had a single initial point, the minimization of the functional (3) subject to the dynamic (2) would be the Bolza optimal control problem with a fixed time [16]. However, we have an ensemble of points  $\mathcal{X}$ , which might be a finite set of points or a continuum.

*q-Folded Method:* The authors in [15] discuss the following method, known as the *q-folded method*, to find the control function  $u$ . This turns the problem into a Bolza problem with the following steps: Copy all the points in  $\mathcal{X}$  into an  $q$ -tuple, denoted by  $X_0$ .

Similarly, stack all the output points into a vector of dimension  $n_o q$ , denoted by  $Y$ , and copy  $n$ -dimensional dynamics in (2)  $q$ -times, creating an  $nq$ -dimensional vectors  $F(u(t), X(t))$ . Then, they find the flow  $\vec{\varphi}_t(u, \cdot) : \mathbb{R}^{nq} \times \mathbb{R} \rightarrow \mathbb{R}^{nq}$ , by solving the minimization problem associated with the following functional for a finite time  $T$ :

$$\mathcal{J}(u, X_0) := \|\Lambda(C)\vec{\varphi}_T(u, X_0) - Y\|^2 + \int_0^T \|u(\tau)\|^2 d\tau$$

where  $\Lambda(C) := \text{diag}(C, \dots, C) \in \mathbb{R}^{n_o q \times nq}$  subject to  $\dot{X}(t) = F(u(t), X(t))$ ,  $X(0) = X_0$ . The stacked structure of  $\dot{X}(t)$  and the definition of  $\Lambda(C)$  ensure that the flow  $\vec{\varphi}_t(u, \cdot)$  is constructed by concatenating  $q$  instances of the flow  $\varphi_t(u, \cdot)$ .

One can easily see that  $q$ -folded method requires solving a boundary value problem on  $qn$  dimension. Solving a boundary value problem by using the shooting method requires solving backward-forward initial value problem for CODE [17] at each iteration. An explicit solver of an initial value problem for CODE on  $\mathbb{R}^n$  has the complexity of  $\mathcal{O}(n^2 N)$  per iteration with  $N$  time-step discretization. Consequently, the  $q$ -folded method has complexity of  $\mathcal{O}(n^2 q^2 N)$  per iteration. Therefore, as the size of the training set,  $q$ , grows, the complexity of the  $q$ -folded method increases quadratically.

### 3 Main Results

*Controllability on Partially Constrained Ensembles:* We first state a simple condition for the existence of a control function  $u$  that memorizes a given ensemble

$(\mathcal{X}, \mathcal{Y})$ . To be more precise, we introduce the following subspaces of our control space:

$$U(x^i, y^i) := \{u \in L_\infty([0, T], \mathbb{R}^{n \times n}) \mid R(\varphi(u, x^i)) = y^i\}$$

Then, we need to show that there exists a control function  $u \in \bigcap_{i=1}^q U(x^i, y^i)$  to prove that the model has memorization property.

Let  $g_1$  and  $g_2$  be differentiable vector fields in  $\mathcal{M} \subseteq \mathbb{R}^n$ . We call the Lie bracket of  $g_1$  and  $g_2$  [18], denoted by  $[g_1, g_2](x)$ , the vector field:

$$[g_1, g_2](x) := \frac{\partial g_2(x)}{\partial x} g_1(x) - \frac{\partial g_1(x)}{\partial x} g_2(x), \text{ for } x \in \mathcal{M}$$

**Definition 2 (Control distributions).** Let  $\dot{x} = f(x, u)$ ,  $u \in L_\infty([0, T], \mathbb{R}^{n \times n})$ , we associate it with the sets of vector fields defined recursively

$$\mathcal{F}^k = \mathcal{F}^{k-1} \cup \{[g_i, g_j](x) \mid g_i, g_j \in \mathcal{F}^{k-1}\}$$

for  $k \geq 1$  with the set of control vectors  $\mathcal{F}^0 = \{f(x, u) \mid u \in L_\infty([0, T], \mathbb{R}^{n \times n})\}$ . The corresponding distributions at point  $x \in \mathbb{R}^n$  are then

$$\mathcal{D}_x^k(\mathcal{F}) = \text{span} \{g(x) \mid g \in \mathcal{F}^k\}$$

One can see that the distributions are nested  $\mathcal{D}_x^k(\mathcal{F}) \subseteq \mathcal{D}_x^{k+1}(\mathcal{F})$ . The control distribution  $\mathcal{D}_x^k(\mathcal{F})$  is said to be involutive if  $\mathcal{D}_x^k(\mathcal{F}) = \mathcal{D}_x^{k+1}(\mathcal{F})$  [18]. We call  $\mathcal{F}$  *bracket-generating* if  $\mathcal{D}_x^\infty(\mathcal{F})$  spans  $T_x E(\mathcal{M})$  for all  $x \in E(\mathcal{M})$ .

We denote  $q$  times Cartesian product of  $E(\mathcal{M})$  by  $E(\mathcal{M})^q := E(\mathcal{M}) \times \cdots \times E(\mathcal{M})$ . A finite ensemble of points is a  $q$ -tuple  $X = [E(x^1)^\top, \dots, E(x^q)^\top]^\top \in E(\mathcal{M})^q \subseteq \mathbb{R}^{nq}$ . We define the set  $\Delta^q := \{[E(x^1)^\top, \dots, E(x^q)^\top]^\top \in E(\mathcal{M})^q \mid E(x^i) = E(x^j) \text{ for } i \neq j\}$ . Let  $E(\mathcal{M})^{(q)} := E(\mathcal{M})^q \setminus \Delta^q$  be the complement of  $\Delta^q$  on  $E(\mathcal{M})^q$ . Please note that if  $\dim E(\mathcal{M}) > 1$ , then  $E(\mathcal{M})^{(q)}$  is an open connected subset and submanifold of  $E(\mathcal{M})^q$  [15].

The set of control vectors of the  $q$ -folded system, denoted by  $\tilde{\mathcal{F}}^0$ , is then

$$\tilde{\mathcal{F}}^0 = \{[f(x^1, u)^\top, f(x^2, u)^\top, \dots, f(x^q, u)^\top]^\top \in \mathbb{R}^{nq} \mid u \in L_\infty([0, T], \mathbb{R}^{n \times n})\}.$$

We note that the control function  $u$  still belongs to  $L_\infty([0, T], \mathbb{R}^{n \times n})$ , and not  $L_\infty([0, T], \mathbb{R}^{nq \times nq})$ . Therefore, having  $\mathcal{D}_X^\infty(\tilde{\mathcal{F}}) = T_X E(\mathcal{M})^{(q)} (= \bigotimes_{i=1}^q T_{x^i} E(\mathcal{M}))$  for all  $X \in E(\mathcal{M})^{(q)}$  is stronger requirement than having  $\mathcal{D}_x^\infty(\mathcal{F}) = T_x E(\mathcal{M})$  for all  $x \in E(\mathcal{M})$ .

**Lemma 1.** Assume that the ensemble  $\mathcal{X}$  consists of finite pairwise distinct points and  $n > n_o$ . For the readout map  $R(x) = Cx$ , if the set of control vector fields of  $q$ -folded system is bracket-generating in  $E(\mathcal{M})^{(q)} (= E(\mathcal{M})^q \setminus \Delta^q)$ , then there exists a control function  $u$  and a finite time  $T \geq 0$  such that the system (2) memorizes the ensemble  $(\mathcal{X}, \mathcal{Y})$  by the control function  $u$ .

*Proof.* First consider that  $i \neq j \Leftrightarrow y^i \neq y^j$ . In this case, the result trivially follows from [2, Prop 6.1]. Now assume that it is not the case; then, since the cardinality of  $\mathcal{Y}$  is finite, we can always construct a set  $\mathcal{Y}' \subset \mathbb{R}^n$ , with elements  $\tilde{y}^i$  having the property that  $i \neq j \Leftrightarrow \tilde{y}^i \neq \tilde{y}^j$  and  $R(\tilde{y}^i) = y^i$  for all  $i$ . We can then appeal to [2, Prop 6.1] to conclude.  $\square$

*Tuning without forgetting:* We describe here the core of our approach to developing numerical methods. Let  $\mathcal{X}^j = \{x^i \in \mathcal{X} | i = 1, 2, \dots, j\}$  be a subset of the ensemble  $\mathcal{X}$  (called *batch* or *sub-ensemble*). Let  $\mathcal{Y}^j$  be the corresponding batch of labels. We denote by a superscript the value of the control at a given iteration, i.e.,  $u^k$  is the control function at the  $k$ th iteration of the algorithm. Assume that  $u^k$  has memorized the ensemble  $(\mathcal{X}^j, \mathcal{Y}^j)$  for the model (2). Expand the ensemble by adding the point  $x^{j+1}$  with its corresponding label  $y^{j+1}$ . Clearly, it does not necessarily hold that  $C\varphi(u^k, x^{j+1}) = y^{j+1}$ . We propose an iterative method to find a control  $u^*$  such that  $C\varphi(u^*, x^i) = y^i$  for all  $x^i \in \mathcal{X}^{j+1} (= \mathcal{X}^j \cup \{x^{j+1}\})$ .

**Definition 3.** Consider an ensemble  $(\mathcal{X}, \mathcal{Y})$ . Assume that the control  $u^k$  has memorized the sub-ensemble  $(\mathcal{X}^j, \mathcal{Y}^j)$  for (2) for some  $j < |\mathcal{I}|$ . If the update  $\delta u^k$  satisfies the following:

1.  $\mathcal{J}^{j+1}(u^k + \delta u^k) \leq \mathcal{J}^{j+1}(u^k)$
2.  $R(\varphi(u^k + \delta u^k, x^i)) = y^i + o(\delta u^k), \forall x^i \in \mathcal{X}^j$

then the control function  $u^{k+1} := u^k + \delta u^k$  has been tuned for  $\mathcal{X}^{j+1}$  without forgetting  $\mathcal{X}^j$ .

Paraphrasing, the definition says that we need to select an update  $\delta u^k$  satisfying the following two conditions: (i) it decreases the per-sample cost functional for the additional point  $x^{j+1}$ , equivalently, the control system with the updated control  $u^{k+1}$  steers the point  $x^{j+1}$  to a point whose projection onto the output subspace  $\mathbb{R}^{n_o}$  gets closer to the label  $y^{j+1}$ , and (ii) the points in  $\mathcal{X}^j$  are mapped to points whose projection onto the output subspace is within  $o(\delta u^k)$  of their corresponding labels. To be more precise, in *tuning without forgetting*, we aim to minimize the per-sample cost for the new point  $x^{j+1}$ , denoted by  $\mathcal{J}^{j+1}(u)$ , with  $u \in \bigcap_{i=1}^j U(x^i, y^i)$ .

*A projected gradient descent method:* Consider the flow  $\varphi_t(u, \cdot)$ . It yields the trajectory  $t \mapsto \varphi_t(u, x^i)$  of (2) with control  $u$  and initialized at  $x^i$  at  $t = 0$ . The first-order variation of the trajectory  $\varphi_t(u, x^i)$  in  $\delta u$  is defined as  $\delta \varphi_t(u, x^i) := \varphi_t(u + \delta u, x^i) - \varphi_t(u, x^i)$ . Under Condition 2 in Definition 3, it should hold that  $C\delta \varphi_T(u, x^i) = 0$  for all  $x^i \in \mathcal{X}^j$  up to first order in  $\delta u$ .

It is well known that  $\delta \varphi_t(u, x^i)$  obeys the linear time-varying equation, which is simply the linearization of the control system (2) about the trajectory  $\varphi_t(u, x^i)$ . Thus, we define the following property:

**Definition 4 (Linearized Controllability Property).** We say the system  $\dot{x}(t) = f(x(t), u(t))$  has the Linearized Controllability Property (LCP) at  $x^i$  for all  $u \in L_\infty([0, T], \mathbb{R}^{n \times n})$  if the linear time varying system:

$$\dot{z}(t) = \left( \frac{\partial f(x, u)}{\partial x} \Big|_{(x=\varphi_t(u, x^i), u)} \right) z(t) + \left( \frac{\partial f(x, u)}{\partial u} \Big|_{(x=\varphi_t(u, x^i), u)} \right) v(t), \quad (4)$$

$v(t) \in L_\infty([0, T], \mathbb{R}^{n \times n})$  is controllable.

Denote by  $\Phi_{(u, x^i)}(t, \tau)$  the state transition matrix of (4).

**Lemma 2.** *Suppose that a given control function  $u$  has memorized the pair of points  $(x^i, y^i)$  for the model (2). Then*

$$\delta\varphi_t(u, x^i) = \int_0^t \Phi_{(u, x^i)}(t, \tau) \frac{\partial f(x, u)}{\partial u} \Big|_{(x=\varphi_\tau(u, x^i), u)} \delta u(\tau) d\tau.$$

up to first order in  $\delta u(t)$ .

*Proof.* For a small variation of the control function  $u(\tau)$ , denoted by  $\delta u(\tau)$ , we have the following:

$$\dot{x}(t) + \delta\dot{x}(t) = f(x(t) + \delta x(t), u(t) + \delta u(t)) \quad (5)$$

Taking the *first* order Taylor expansion of (5) around the trajectory  $x(t) = \varphi_t(u, x^i)$  and subtracting  $\varphi_t(u, x^i)$ , we get

$$\frac{d}{dt} \delta\varphi_t(u, x^i) = \frac{\partial f(x, u)}{\partial x} \delta\varphi_t(u, x^i) + \frac{\partial f(x, u)}{\partial u} \delta u \quad (6)$$

up to first order in  $\delta u$ . To emphasize the linearity of the system above in the control update  $\delta u$ , we can introduce the notation  $z(t) = \delta\varphi_t(u, x^i)$ . Then, from the introduced notation, one can easily see that (6) matches with (4). Using the variation of constants formula [16], we have the following:

$$z(t) = \Phi_{(u, x^i)}(t, 0)z(0) + \int_0^t \Phi_{(u, x^i)}(t, \tau) \frac{\partial f(x, u)}{\partial u} \Big|_{(x=\varphi_\tau(u, x^i), u)} \delta u(\tau) d\tau$$

We have  $z(0) = 0$  since we have  $\varphi_0(u + \delta u, x^i) = x^i$  and  $\varphi_0(u, x^i) = x^i$ . This completes the proof.  $\square$

Based on Lemma 2, we define an affine operator from the space of bounded functions over the time interval  $[0, T]$ ,  $\delta u(t) \in L_\infty([0, T], \mathbb{R}^{n \times n})$ , to  $\mathbb{R}^{n_o}$ , mapping a control variation to the resulting variation in the end-point of the trajectory see through the readout map  $R$ . This operator is defined as

$$\mathcal{L}_{(u, x^i)}(\delta u) := R \left( \int_0^T \Phi_{(u, x^i)}(T, \tau) \frac{\partial f(x(\tau), u(\tau))}{\partial u} \delta u(\tau) d\tau \right)$$

for  $x(\tau) = \varphi_\tau(u, x^i)$ . Next, we let

$$\mathcal{K}(u, x^i) := \text{span}\{\delta u \in L_\infty([0, T], \mathbb{R}^{n \times n}) \mid \mathcal{L}_{(u, x^i)}(\delta u) = 0\}$$

be the kernel of the operator  $\mathcal{L}_{(u, x^i)}(\cdot)$ . Then, we define the intersection of the kernel  $\mathcal{K}(u, x^i)$  for all  $x^i \in \mathcal{X}^j$  as follows:

$$\mathcal{K}(u, \mathcal{X}^j) := \text{span}\{\delta u \in L_\infty([0, T], \mathbb{R}^{n \times n}) \mid \delta u \in \bigcap_{x^i \in \mathcal{X}^j} \mathcal{K}(u, x^i)\}$$

We define the gradient of the given per-sample cost functional for the sample  $x^i$  at control  $u$ , as the first order variation of  $\mathcal{J}^i(u)$  in  $\delta u$ , precisely,  $\nabla_u \mathcal{J}^i(u) := \mathcal{J}^i(u + \delta u) - \mathcal{J}^i(u)$ . Then, one can see that we have the following:

$$\nabla_{u(t)} \mathcal{J}^i(u) := \delta\varphi_t^\top(u, x^i) C^\top (C\varphi(u, x^i) - y^i)$$



Note that it is a function of time  $t$  via the first-order variation  $\delta\varphi_t(u, x^i)$ . We define the projection of  $\nabla_{u(t)}\mathcal{J}^{j+1}(u)$  on a given subspace of functions  $\mathcal{K}(u, \mathcal{X}^j)$  as the solution of the following optimization problem:

$$\text{proj}_{\mathcal{K}(u, \mathcal{X}^j)} \nabla_{u(t)}\mathcal{J}^{j+1}(u) := \arg \min_{d(t) \in \mathcal{K}(u, \mathcal{X}^j)} \int_0^T \|d(\tau) - \nabla_{u(\tau)}\mathcal{J}^{j+1}(u)\|^2 d\tau$$

Now, we can state the main result:

**Theorem 1.** *Consider model (2) and suppose that the control vector fields of  $q$ -folded system (2) are bracket-generating in  $E(\mathcal{M})^{(q)}$ , and the control function  $u^k$  has memorized the ensemble  $(\mathcal{X}^j, \mathcal{Y}^j)$ . Assume the space of controls that memorize  $(\mathcal{X}, \mathcal{Y})$  is connected Banach submanifold of  $L_\infty([0, T], \mathbb{R}^{n \times n})$ . If  $\delta u^k$  is selected as  $\text{proj}_{\mathcal{K}(u, \mathcal{X}^j)} \nabla_{u(t)}\mathcal{J}^{j+1}(u)$ , then the control function  $u^{k+1}(:= u^k + \delta u^k)$  for  $\mathcal{X}^{j+1}$  has been tuned without forgetting  $\mathcal{X}^j$  up to the first order.*

We note that Theorem 1 assumes that the space of controls that memorize  $(\mathcal{X}, \mathcal{Y})$  is a connected Banach submanifold of  $L_\infty([0, T], \mathbb{R}^{n \times n})$ . This assumption guarantees that gradient descent to minimize  $\mathcal{J}^{j+1}(u)$  subject to  $u \in \bigcap_{i=1}^j U(x^i, y^i)$  is a well-founded approach. However, by using LCP, we have the following Theorem to relax this assumption.

**Theorem 2.** *Assume the system  $\dot{x}(t) = f(x(t), u(t))$  on a manifold  $E(\mathcal{M})$  has the LCP for each  $x^i$ , and that the set of control vector fields of  $q$ -folded system is bracket-generating in  $E(\mathcal{M})^{(q)}$ . Assume that  $E(\mathcal{M})$  is connected and that the fundamental group  $\pi_1(E(\mathcal{M})) = 0$ . Then, the space of controls that memorize  $(\mathcal{X}, \mathcal{Y})$  is connected Banach submanifold of  $L_\infty([0, T], \mathbb{R}^{n \times n})$  of finite-codimension.*

Due to our focus on algorithms and because of space limitations, we do not include the proof of Theorem 2 here. We will provide it an extended version. Now, we can present the proof of Theorem 1.

*Proof (Theorem 1).* From Lemma 1, we know that there exists a control  $u^{k+1}$  such that

$$C\varphi(u^{k+1}, \mathcal{X}^{j+1}) = \mathcal{Y}^{j+1}.$$

From definition of  $\mathcal{L}_{(u^k, x^i)}(\cdot)$  and Lemma 2, we have:

$$C\varphi(u^k + \delta u^k, x^i) = C\varphi(u^k, x^i) + \mathcal{L}_{(u^k, x^i)}(\delta u^k), \forall x^i \in \mathcal{X}^j$$

up to first order in  $\delta u^k$ . Then, we project  $\nabla_u\mathcal{J}^{j+1}(u^k)$  onto  $\mathcal{K}(u^k, \mathcal{X}^j)$ , that is, we select  $\delta u^k$  as  $\delta u^k = \text{proj}_{\mathcal{K}(u^k, \mathcal{X}^j)} \nabla_u\mathcal{J}^{j+1}(u^k)$ . The selection satisfies Condition 1 in Definition. 3. By the construction of  $\mathcal{K}(u^k, \mathcal{X}^j)$  and from the memorization assumption, we have that  $\mathcal{L}_{(u^k, x^i)}(\delta u^k) = 0, \forall x^i \in \mathcal{X}^j$  and  $C\varphi(u^k, x^i) = y^i, \forall x^i \in \mathcal{X}^j$ , respectively. Then, we have:

$$C(\varphi(u^k + \delta u^k, x^i)) = y^i + o(\delta u^k), \forall x^i \in \mathcal{X}^j$$

It matches the Condition 2 in Definition 3, and this completes the proof.  $\square$

We emphasize that we *only* consider the intersection of the kernel  $\mathcal{K}(u^k, x^i)$  for all the points  $x^i \in \mathcal{X}^j$  because we need to keep the first-order variation of the end-point mapping  $C(\varphi(u^k, x^i))$  zero for all points  $x^i$  in  $\mathcal{X}^j$  while we allow variation on the end-point mapping for the rest of the points.

## 4 Numerical Method for Tuning without Forgetting

Building on the results of the previous section, we describe in this section a numerical algorithm to *tune* a control function *without forgetting*  $\mathcal{X}^j$  up to the first order. The algorithm comprises three main phases, on which we elaborate below. Assume that a control  $u^0$  has memorized  $(\mathcal{X}^j, \mathcal{Y}^j)$  for a given control affine system. Consider an expansion of the ensemble,  $\{x^i\}_{i=j+1}^q$ . For the iteration  $k = 1$ , we let  $u^1 = u^0$ . We assume that  $T = 1$  without loss of generality. For the sake of exposition, we also rely on an explicit Euler method to solve the ODEs involved; using different solvers would entail a trivial modification. We let  $N$  be the number of discretization points.

For the sake of notation simplicity, we assume that the control system is affine in the control:  $\dot{x}(t) = \sum_{d=1}^p u_d(t) f_d(x(t))$ ,  $u_d(t) \in \mathbb{R}$ ,  $u \in L_\infty([0, T], \mathbb{R}^p)$  for some smooth functions  $f_d(\cdot) \in \text{Vect}(E(\mathcal{M}))$ ,  $d = 1, \dots, p$ . However, one can easily apply the algorithm by vectorizing the control function in the matrix form (2).

We let  $u^k \in \mathbb{R}^{pN}$  be the discretization of a time-varying control function and denote the corresponding discretized trajectory by  $\varphi_{[1:N]}(u^k, x^i) \in \mathbb{R}^{n \times N}$ . We let  $\delta u^k \in \mathbb{R}^{pN}$  be the discretization of a time-varying control variation.

*Approximation of  $\mathcal{L}_{(u, x^i)}(\cdot)$ :* We first provide a method to compute a numerical approximation of  $\mathcal{L}_{(u^1, x^i)}(\cdot)$  for all  $x^i \in \mathcal{X}^j$ . This is Algorithm 1 below, which computes the numerical approximation of  $d_u \varphi(u^k, x^i)$ . The algorithm iteratively

---

### Algorithm 1 Approximation of $\mathcal{L}_{(u, x^i)}(\cdot)$

---

```

1: Input:  $u, x^i$ 
2:  $z \leftarrow \varphi_{[1:N]}(u, x^i)$ ,  $\Phi_{N+1} \leftarrow I_{n \times n}$ 
3: for  $\ell = N$  to 1 do
4:    $F[\ell] \leftarrow f(z[\ell], u[\ell])$ 
5:    $\Phi_\ell \leftarrow \Phi_{\ell+1} (I + \frac{\partial F[\ell]}{\partial x})$ 
6: end for
7: return  $C[\Phi_1 F[1], \Phi_2 F[2], \dots, \Phi_N F[N]]$ 

```

---

computes the state transition matrix  $\Phi_{(u^k, x^i)}(T, \tau)$  of the system in (4) for a given initial point  $x^i$  and a control function  $u^k$ . Then,  $\Phi_{(u^k, x^i)}(T, \tau)$  is multiplied with the matrix of vector fields in (2) and the product is stored in a matrix, which is the numerical approximation of  $d_u \varphi(u^k, x^i)$ . Let  $L_i \in \mathbb{R}^{n_o \times pN}$  be the output of Algorithm 1 for a given  $u^k$  and an initial point  $x^i$ . We have  $\mathcal{L}_{(u^k, x^i)}(\delta u^k) \approx L_i \delta u^k$ . Then, the right kernel of the matrix  $L_i$  is the numerical approximation of  $\mathcal{K}(u^k, x^i)$ .

*Phase I:* We now implement the statement of Theorem 1. We state the Phase I in Algorithm 2. We first compute  $L_i$  for all  $x^i \in \mathcal{X}^j$  by using Algorithm 1 to construct  $\mathcal{K}(u^k, \mathcal{X}^j)$  for a given  $j$ . Then, we column-wise concatenate them and place the concatenated matrix into the first block of  $L \in \mathbb{R}^{n_o q \times pN}$  and fill the remaining entries with zeros. One can easily see that the right kernel of the matrix  $L$ , denoted by  $\mathcal{N}(L)$ , is a numerical approximation of  $\mathcal{K}(u^1, \mathcal{X}^j)$  for a given  $j$ . Following Theorem 1, we project  $\nabla_u \mathcal{J}^{j+1}(u^k) \in \mathbb{R}^{pN}$  onto  $\mathcal{N}(L)$ , which is a linear operation. We have the following for the update control  $u^{k+1}$ :

$$u^{k+1} = u^k - \alpha^k \text{proj}_{\mathcal{N}(L)} \nabla_u \mathcal{J}^{j+1}(u^k) \quad (7)$$

where  $\alpha^k \in \mathbb{R}^+$  is a step size. We repeat the iteration in (7), and we compute the matrix  $L_i$  for  $i = 1, \dots, j$  for the control  $u^k$  to replace corresponding blocks in the matrix  $L$  until the per-sample loss converges. Once it converges, we pick the next point  $x^{j+2}$  and compute the matrix  $L_i$  for  $i = 1, \dots, j+1$  to replace corresponding blocks in the matrix  $L$ . We then continue picking new points.

---

**Algorithm 2** Phase I: Kernel Projected Gradient Descent

---

```

1:  $L_i \leftarrow \text{Algorithm1}(u^0, x^i), \forall i \in \mathcal{I}^j$ 
2:  $u \leftarrow u^0$ 
3:  $L \leftarrow [L_1; L_2; \dots; L_j; 0; \dots; 0]$ 
4: for  $i = j+1$  to  $q$  do
5:   repeat
6:      $u \leftarrow u - \alpha \text{proj}_{\mathcal{N}(L)} \nabla_u \mathcal{J}^i(u)$ 
7:      $L_\ell \leftarrow \text{Algorithm1}(u, x^\ell)$ , for  $\ell = 1, \dots, i-1$ 
8:     Update the corresponding block  $L_\ell$  in  $L, \forall \ell \in \mathcal{I}^{i-1}$ 
9:   until convergence
10: end for
11: return  $L, u^k$ 
```

---

*Remark 1.* If we consider  $\mathcal{X}^0$  as an empty set and  $u^0 = \vec{0}$ , it results in  $L$  to be a zero matrix. Thus, one can use the proposed algorithm to design a control function  $u$  that sends given initial states to desired final ones. This is an alternative to the  $q$ -folded method.

In the next two phases, we focus on minimizing the cost-functional  $\mathcal{J}(u, \mathcal{X})$  in (3), which comprises two sub-cost functionals: the sum of per-sample costs and the  $L^2$  norm of the control  $u$ . First, we minimize the  $L^2$  norm of the control  $u$ , and then we minimize the sum of per-sample costs. We call the consecutive execution of Phase II and III *refinement rounds*. One can employ multiple refinement rounds to reduce the cost.

*Phase II* In this phase, we minimize the  $L^2$  norm of the control  $u$ . We project the gradient of the  $L^2$  norm of the control function onto the subspace of functions  $\mathcal{K}(u^k, \mathcal{X})$  at each iteration. We state Phase II in Algorithm 3.

*Phase III* In this phase, we aim to refine the control  $u$  to steer all the points closer to their associated end-points. Let  $\mathcal{P}$  be the number of iterations per sample in the algorithm. For a given point  $x^i$ , we first call Algorithm 1 to compute  $L_\ell$  for all  $\ell \in \mathcal{I} \setminus \{i\}$ . Notably, we do not update the kernel for  $x^i$ . Then, we update the matrix  $L$  accordingly and compute the projected gradient. Then, we pick the next point  $x^{i+1}$ . We repeat these steps until the last point in the training set. Then, we again pick  $x^1$  and repeat the steps above  $\mathcal{P}$  times.

---

**Algorithm 3** Phase II: Regularization

---

```

1:  $L, u \leftarrow \text{Algorithm2}()$ 
2: for  $i = 1$  to  $q$  do
3:   repeat
4:      $L_\ell \leftarrow \text{Algorithm1}(u, x^\ell), \forall x^\ell \in \mathcal{X}$ 
5:     Update the corresponding block  $L_\ell$  in  $L, \forall \ell \in \mathcal{I}$ 
6:      $u \leftarrow u - \alpha \text{proj}_{\mathcal{N}(L)}(u)$ 
7:   until convergence
8: end for
9: return  $L, u$ 

```

---



---

**Algorithm 4** Phase III: Refinement

---

```

1:  $L, u \leftarrow \text{Algorithm3}()$ 
2: for  $m = 1$  to  $\mathcal{P}$  do
3:   for  $i = 1$  to  $q$  do
4:      $L_\ell \leftarrow \text{Algorithm1}(u, x^\ell), \forall x^\ell \in \mathcal{X} \setminus \{x^i\}$ 
5:     Update the corresponding block  $L_\ell$  in  $L, \forall \ell \in \mathcal{I} \setminus \{i\}$ 
6:      $u \leftarrow u - \alpha \text{proj}_{\mathcal{N}(L)} \nabla_u \mathcal{J}^i(u)$ 
7:   end for
8: end for
9: return  $u$ 

```

---

In the *tuning without forgetting* method, we solve a boundary value problem on  $\mathbb{R}^n$  for each sample point. Thus our method has a complexity of  $\mathcal{O}(qn^2N)$  per iteration in total. The  $q$ -folded method has complexity of  $\mathcal{O}(q^2n^2N)$  per iteration.

## 5 A Computational Example

In this section, we provide computational examples. We consider the model

$$\dot{x}(t) = W_2(t) \tanh(W_1(t)x(t) + b_1(t)) + b_2(t) \quad (8)$$

where  $W_1(t), W_2(t) \in L_\infty([0, T], \mathbb{R}^{\bar{n} \times \bar{n}})$  and  $b_1(t), b_2(t) \in L_\infty([0, T], \mathbb{R}^{\bar{n}})$ . We discretize trajectories with  $N = 10$ . For a given  $x^i \in \mathcal{X} \in \mathbb{R}^2$ , we consider the

following end-point mapping.

$$x^i \in \mathbb{R}^2 \xrightarrow{E} [x^i{}^\top, 0_{1 \times 6}]^\top \in \mathbb{R}^8 \xrightarrow{\varphi_T(u, \cdot)} \bar{y}^i \in \mathbb{R}^8 \xrightarrow{C=[0, \dots, 0, 1]} y^i = \begin{cases} -1 & , \|x^i\|_2^2 \leq 1 \\ 1 & , \|x^i\|_2^2 > 1 \end{cases}$$

In words, we consider a unit radius ball in  $\mathbb{R}^2$  centered at the origin. If a point is inside the ball, we aim to steer it to the hyperplane  $x_8 = 1$ , otherwise, to the hyperplane  $x_8 = -1$  (a similar example is provided in [11]). One can see that the set of control vector fields for the (overparametrized)  $q$ -folded control system (8) is bracket-generating in  $E(\mathcal{M})^{(q)}$ . We define the average error for a control  $u$  on the ensemble  $\mathcal{X}$  as

$$\mathcal{E}(u, \mathcal{X}) := \frac{1}{|\mathcal{X}|} \sum_{x^i \in \mathcal{X}} \|C\varphi(u, x^i) - y^i\|.$$

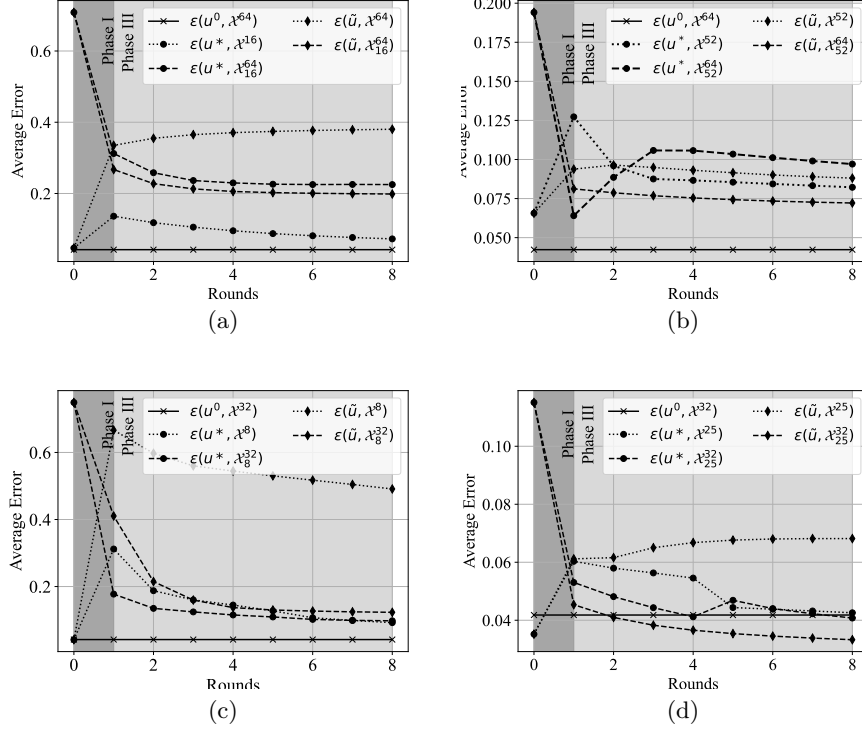
We have a set  $\mathcal{X}$  of cardinality 64 whose elements are indexed by  $\mathcal{I}$ . First, we have that an initial training set (sub-ensemble)  $\mathcal{X}^j$  is given for some  $j$ . We apply the  $q$ -folded method in Section 2 to learn these samples and denote the control function that  $q$ -method gives by  $u^0$ . Then, we expand the training set  $\mathcal{X}^j$  to  $\mathcal{X}^i$  by adding samples in the order provided by  $\mathcal{I}$ . For convenience, we denote the difference of the subensembles  $\mathcal{X}_j^i := \{x^\ell \in \mathcal{X} : j < \ell \leq i\} (= \mathcal{X}^i \setminus \mathcal{X}^j)$ .

To adapt the control function  $u^0$  to new samples while remembering previous ones, we use the *tuning without forgetting*  $\mathcal{X}^j$  method with multiple refinement rounds to obtain the control function  $u^*$ . We also compare our results with the *Penalty Method*, which encourages the tuned control function  $\tilde{u}$  to stay close to  $u^0$  using a penalty term in the cost functional [1].

$$\mathcal{J}(\tilde{u}, \mathcal{X}) := \sum_{i=1}^M \|C\varphi(\tilde{u}, x^i) - y^i\|^2 + \lambda \int_0^T \|\tilde{u}(\tau) - u^0(\tau)\|^2 d\tau$$

Figure 1 depicts the average error on different sets as a function of number of rounds for both algorithms. First, we observe that, in Fig 1, the average error on the new points  $\mathcal{E}(u^0, \mathcal{X}_j^i)$  is remarkably higher compared to  $\mathcal{E}(u^0, \mathcal{X}^j)$  for any given training set at round 0 (where  $u^* = u^0$  and  $\tilde{u} = u^0$ ). As expected, it shows that the control  $u^0$  has *not* learned the additional samples  $\mathcal{X}_j^i$  for the given model.

For the learning plasticity [3], we measure the difference in the performance of model on a task between its joint training performance  $\mathcal{E}(u^0, \mathcal{X}^i)$  and fine-tuning performance  $\mathcal{E}(\cdot, \mathcal{X}_j^i)$ . We observe that both  $\mathcal{E}(u^*, \mathcal{X}_j^i)$  and  $\mathcal{E}(\tilde{u}, \mathcal{X}_j^i)$  for  $(i, j) = (64, 52), (32, 8), (32, 25)$  are close to  $\mathcal{E}(u^0, \mathcal{X}^i)$ . It shows that the control  $u^*$  and  $\tilde{u}$  has learned the additional samples satisfactorily. For the memory stability [3], we compare the error on the previously learned points  $\mathcal{E}(\cdot, \mathcal{X}^j)$  as the tuning continues. We observe that  $\mathcal{E}(u^*, \mathcal{X}^j)$  for  $j = 8, 16, 25$  is close to  $\mathcal{E}(u^0, \mathcal{X}^j)$ . It shows that the control  $u^*$  keeps the performance on the previously learned points  $\mathcal{X}^j$  nearly constants. Comparing our method to the Penalty Method, we observe that  $\mathcal{E}(\tilde{u}, \mathcal{X}^j)$  is higher than  $\mathcal{E}(u^0, \mathcal{X}^j)$  for any given  $j$ . It shows that the control function  $\tilde{u}$  has learned the additional points but has forgotten the previously learned points. Thus, the penalty method does not directly address the issue of catastrophic forgetting.



**Fig. 1.** (a) and (b) average error as a function of number of rounds for  $|\mathcal{X}| = 64$  for  $j = 16$  and  $j = 52$ , respectively. (c) and (d) average error as a function of number of rounds for  $|\mathcal{X}| = 32$  for  $j = 8$  and  $j = 25$ , respectively. The dark gray region is Phase I region and the light gray region is Phase III region (each round is followed by Phase II). Average error on the given set for the control functions  $u^*$ ,  $\tilde{u}$ , and  $u^0$  are marked by  $\bullet$ ,  $\diamond$ , and  $\times$ , respectively.

## 6 Summary and Outlook

In this work, we have considered a controlled dynamical system to learn a task, focusing on the impact of training set expansion. We have addressed the issue of model performance loss on the original task while adapting the model to a new task-specific dataset, known as catastrophic forgetting. We have introduced a novel fine-tuning method, *tuning without forgetting* up to first-order. Our work contributes to the scalability of control methods, offering a novel approach to adaptively handle training set expansions. In our numerical results, we have observed that the proposed algorithm effectively handles changes in ensemble cardinalities, preserving previously learned points and adopting new points.

The present work can be extended in several directions. Our method has the disadvantage of storing  $\mathcal{X}^j$  to compute the set  $\mathcal{K}(u, \mathcal{X}^j)$ . To address this, one can keep track of the variation in the kernels to approximate the new kernel. Also, we will provide the proof for Theorem 2 which relaxes some of the assumptions for Theorem 1 in an extended version.

## References

1. H. Gouk, T. M. Hospedales, and M. Pontil, “Distance-based regularisation of deep networks for fine-tuning,” in *ICLR*, ICLR, 2021.
2. A. Agrachev and A. Sarychev, “Control in the spaces of ensembles of points,” *SIAM Journal on Control and Optimization*, vol. 58, no. 3, pp. 1579–1596, 2020.
3. L. Wang, X. Zhang, H. Su, and J. Zhu, “A comprehensive survey of continual learning: Theory, method and application,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
4. D. Lopez-Paz and M. Ranzato, “Gradient episodic memory for continual learning,” *Advances in neural information processing systems*, vol. 30, 2017.
5. L. Xuhong, Y. Grandvalet, and F. Davoine, “Explicit inductive bias for transfer learning with convolutional networks,” in *International Conference on Machine Learning*, pp. 2825–2834, PMLR, 2018.
6. X. Li, H. Xiong, H. Wang, Y. Rao, L. Liu, and J. Huan, “Delta: Deep learning transfer using feature map with attention for convolutional networks,” in *International Conference on Learning Representations*, 2018.
7. J. Tian, Z. He, X. Dai, C.-Y. Ma, Y.-C. Liu, and Z. Kira, “Trainable projected gradient method for robust fine-tuning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7836–7845, 2023.
8. M. Farajtabar, N. Azizan, A. Mott, and A. Li, “Orthogonal gradient descent for continual learning,” in *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773, PMLR, 2020.
9. S. Wang, X. Li, J. Sun, and Z. Xu, “Training networks in null space of feature covariance for continual learning,” in *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 184–193, 2021.
10. R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, “Neural ordinary differential equations,” *Advances in Neural Information Processing Systems*, vol. 31, 2018.
11. E. Dupont, A. Doucet, and Y. W. Teh, “Augmented neural ODEs,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
12. Y. Lu, A. Zhong, Q. Li, and B. Dong, “Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations,” in *International Conference on Machine Learning*, pp. 3276–3285, PMLR, 2018.
13. C. Cuchiero, M. Larsson, and J. Teichmann, “Deep neural networks, generic universal interpolation, and controlled odes,” *SIAM Journal on Mathematics of Data Science*, vol. 2, no. 3, pp. 901–919, 2020.
14. P. Tabuada and B. Ghahserifard, “Universal approximation power of deep residual neural networks through the lens of control,” *IEEE Transactions on Automatic Control*, vol. 68, pp. 2715–2728, 2023.
15. A. Agrachev and A. Sarychev, “Control on the manifolds of mappings with a view to the deep learning,” *Journal of Dynamical and Control Systems*, vol. 28, no. 4, pp. 989–1008, 2022.
16. D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton university press, 2011.
17. H. B. Keller, *Numerical Solution of Two Point Boundary Value Problems*. SIAM, 1976.
18. R. Brockett, “The early days of geometric nonlinear control,” *Automatica*, vol. 50, no. 9, pp. 2203–2224, 2014.