OPTIMAL CONTROL NEURAL NETWORKS FOR DATA-DRIVEN DISCOVERY OF GRADIENT FLOWS

Anonymous authors
Paper under double-blind review

ABSTRACT

This work aims to discover nonlinear dynamical systems given a set of time series data on solution trajectories. To tackle this problem, we propose Optimal Control Neural Networks (OCN) to learn the unknown vector field. The OCN consists of a neural network representation of the system, coupled with an optimal control formulation. Specifically, we formulate the parameter learning problem as a data-driven optimal control problem. This allows for the use of existing optimal control tools. We derive generalization error bounds for both the solution and the vector field, and the bounds are shown to depend on both the training error and the time gaps between neighboring data. We also provide several numerical examples to demonstrate the viability of OCN, as well as its generalization ability.

1 INTRODUCTION

A central challenge in many diverse areas of science and engineering is to discover physical laws. This work is concerned with learning dynamical systems arising from real-world applications, but where a complete mathematical description is unavailable. In such scenarios, we rely on extracting insight from data. In particular, machine learning techniques have emerged to ascertain the properties of these systems, given an abundance of data.

Data-driven discovery of dynamical systems. There is a long and fruitful history of modeling dynamics from data. Earlier efforts for system discovery include a large set of methods (See 1.1 below); one of them is the symbolic regression Bongard & Lipson (2007); Schmidt & Lipson (2009), which finds nonlinear dynamic equations. This strategy balances the complexity of the model with predictive power, however, it is often expensive and requires extra care in dealing with candidate models. More recently, sparsity has been used to determine the governing dynamical system Proctor et al. (2014); Brunton et al. (2016); Champion et al. (2019a;b), where the authors deploy certain sparsity promoting strategies to obtain parsimonious models. The challenge with this strategy lies in choosing a suitable sparsifying function basis. Instead of discovering exact systems, increasing efforts have been made to seek accurate numerical approximations to dynamical systems; see e.g. Raissi et al. (2018); Rudy et al. (2019); Qin et al. (2019); Lu et al. (2019) for works using the neural network representation. Our work in this paper falls into this category.

Deep neural networks (DNN). DNNs have seen tremendous successes in many different disciplines, particularly in supervised learning. Their structure with numerous consecutive layers of artificial neurons allows DNNs to express complex input-output relationships. Efforts have been devoted to the use of DNNs for various aspects of scientific computing, including solving and learning systems involving ODEs and PDEs. Recently, the interpretation of residual networks by He et al. He et al. (2016) as approximate ODE solvers in E (2017) spurred research on the use of ODEs in deep learning Chen et al. (2018); Gholami et al. (2019); Quaglino et al. (2019). Neural ODEs Chen et al. (2018) as neural network models generalize standard layer-to-layer propagation to continuous depth models. Along this line of research, work Liu & Markowich (2020) uses the PDE model to represent a continuum limit of neural networks in both depth and width.

Optimal control neural networks. Recently, there has been a growing interest in understanding deep learning methods through the lens of dynamical systems and optimal control Li et al. (2018); Li & Hao (2018); Zhang et al. (2019); Benning et al. (2019). An appealing feature of this approach is that the compositional structure is explicitly taken into account in the time evolution of the dynamical systems, from which novel algorithms and network structures can be designed using optimal control techniques. In particular, mathematical concepts from optimal control theory are naturally translatable to dynamic neural networks, and provide interesting possibilities including computing loss gradient by the adjoint method and natural incorporation of regularization and/or prior knowledge into the loss function. The current work directly takes advantage of these concepts.

In this paper, we build upon recent efforts that discover dynamical systems using deep neural networks (DNN) Raissi et al. (2018); Qin et al. (2019) and the optimal control approach for learning system parameters Liu & Tian (2021) for COVID-19. We seek to gain new insight into the dynamic discovery problem with optimal control networks (OCN for short). In this framework, we use DNN as a global representation to approximate the unknown vector field in the underlying dynamical system, and define a loss function to fit the observational data, this way the resulting optimization becomes an optimal control problem. The study on the learning problem can then be conducted by optimal control techniques. Here we select gradient flows $\dot{x} = -\nabla f(x)$ as an example, while the methodology can be applied to more general dynamical systems $\dot{x} = F(x, t)$.

In summary, our contributions are:

1. We propose and study a method for discovering unknown dynamical systems from observed data, using neural network approximations and an optimal control formulation.
2. We establish generalization error bounds for both the solution and the vector field, which show that the bounds depend only on the training error and the time interval between data in the time series.
3. We demonstrate OCN's performance on linear and nonlinear gradient flows by learning their dynamical behavior around different types of nodes. We further illustrate OCN's performance on general dynamic systems, including the damped pendulum system and Lorenz system.

1.1 Related Works

There are techniques that address various aspects of the dynamical system discovery problem, including methods to discover governing equations from time-series data Crutchfield & McNamara (1987), equation-free modeling Kevrekidis et al. (2003), empirical dynamic modeling Sugihara et al. (2012); Ye et al. (2015), modeling emergent behavior Roberts (2014), nonlinear Laplacian spectral analysis Giannakis & Majda (2012), artificial neural networks González-García et al. (1998), Koopman analysis Williams et al. (2015); Brunton et al. (2017) and automated inference of dynamics Daniels & Nemenman (2015); Schmidt et al. (2011).

Our work is aligned with those of Raissi et al. (2018); Qin et al. (2019) but with a different strategy. Work in Qin et al. (2019) first discretizes the dynamical system based on a local integral form, then uses a neural network as an approximation to the local flow map between two neighboring data points. In contrast, we incorporate a global network representation into the optimal control formulation. Such global approximation using neural network representation is also considered in Raissi et al. (2018), while the parameter learning method therein is also built for a discretized dynamical system in the form of multi-step time-stepping schemes. Importantly, we are able to obtain generalization error bounds that allow users to judiciously reason about the accuracy and convergence of such methods.

This work is complementary to efforts that incorporate ODE solver technology into training neural networks. This includes numerical methods to accelerate learning neural ODEs like the checkpoint methods Gholami et al. (2019), symplectic adjoint method Matsubara et al. (2021), interpolation instead of adjoint method Daulbaev et al. (2020), and the proximal implicit solvers Baker et al. (2022).

The rest of the paper is arranged as follows: problem setup and our methods are introduced in Section 2 with detailed mathematical formulations. Section 3 presents a theoretical analysis of the errors. Computational results are presented in Section 4. Finally, some concluding remarks and discussions are given in Section 5. Implementation details and technical proofs are given in the appendix.

2 Methods

Here we provide an overview of our method. We first present the problem setup based on a set of time series data in order to learn the unknown vector field. Afterward, we argue why we can use neural networks to realize the needed approximation. Then we explain the learning phase of the neural network, which seeks to solve an optimal control problem. And finally, we explain the training stage, where we are able to produce gradients in parameter space in order to update network parameters.

2.1 Problem Setup

Many applications are modeled by gradient flows Ambrosio et al. (2005). We consider gradient flow systems of form

$$\dot{x}(t) = -\nabla f(x(t)), \quad x(0) = x_0 \tag{2.1}$$

on $[0, T]$, where $x \in \mathbb{R}^d$ is the state variable. In this paper, we assume the form of $f : \mathbb{R}^d \to \mathbb{R}$ is unknown. We aim to create an accurate model for learning or recovering f using data sampled from the solution trajectories, and generating solutions over the entire time interval.

Numerically, in order to produce trajectories of the dynamical system when f is known one can use various integrators, such as forward Euler,

$$x(t_{i+1}) = x(t_i) - \Delta t \nabla f(x(t_i)), \tag{2.2}$$

where the time domain is divided into equal step-sizes so that $0 = t_0 < ... < t_n = T$. Other high-accuracy schemes with better convergence properties, e.g. 4th order Runge-Kutta can also be used. Here we assume that data is collected as solution states on a uniform lattice of time points $\{t_i\}_{i=0}^n$.

2.2 Neural Network Approximator

Our approach is to first model $f(x)$ using a deep neural network. Though $f(x(t))$ is an operator, taking $x(t)$ as the input, we can still apply network approximations since input functions are actually represented discreetly.

A fully connected feedforward neural network $G(\cdot, \theta) : \mathbb{R}^{N_1} \to \mathbb{R}^{N_m}$ can be seen as a composition of a sequence of linear functions and nonlinear functions:

$$G(\cdot, \theta) = \sigma_{m-1} \circ h_{m-1} \circ \cdots \circ \sigma_1 \circ h_1.$$

Here $h_j : \mathbb{R}^{N_j} \to \mathbb{R}^{N_{j+1}}$ are linear functions: $h_j(x) = W_j x + b_j$, where $W_j \in \mathbb{R}^{N_j \times N_{j+1}}$ are matrices, also called weights, $b_j \in \mathbb{R}^{N_{j+1}}$ are biases. $\sigma_j : \mathbb{R} \to \mathbb{R}$ are nonlinear activation functions applied component-wisely to the j-th layer. $\theta \in \mathbb{R}^N$ denotes the parameter set containing all the parameters $W_1, b_1, ..., W_{m-1}, b_{m-1}$ in the network, where $N = \sum_{j=1}^{m-1}(N_j+1)N_{j+1}$. In this work, we take $N_1 = d, N_m = 1$ so that

$$f(\cdot) \sim G(\cdot, \theta) : \mathbb{R}^d \to \mathbb{R}.$$

The universal approximation capacity of full connected feedforward neural networks Hornik et al. (1989); Barron (1993) ensure that this is a reasonable choice. Additionally, to ensure that ∇f is Lipschitz continuous in x, we take $\sigma_j(x) = \tanh(x)$ so that $\nabla_y G(y, \theta)$ is also Lipschitz continuous.

2.3 Neural gradient flow and Loss function

With no access to function values $f(x_i)$ for $i \in [n] := \{1, 2, ..., n\}$, the usual supervised learning of a function is not directly applicable. The way we learn θ in the approximator $G(\cdot, \theta)$ is to solve the parameterized ODE system

$$\dot{y}(t) = -\nabla_y G(y(t), \theta), \quad y(0) = x_0, \tag{2.3}$$

so that its solution at t_i is close to x_i for $i \in [n]$. To this end, we simply take the loss function

$$J(\theta) = \sum_{i=1}^{n} \|y(t_i) - x_i\|^2, \tag{2.4}$$

where the dependence of J on θ is through $y(t)$. Solution trajectory of (2.3) when an optimal parameter θ is obtained should be close to the solution trajectory of

$$\dot{x}(t) = -\nabla f(x(t)), \quad x(0) = x_0. \tag{2.5}$$

Note that the dataset $\{x_i\}_{i=1}^n$ used to train the model is time-independent, hence θ can be a time-independent parameter. This point is important for our choice of solvers of (2.3).

2.4 Optimal control formulation

Now our problem is reduced to learning θ by minimizing the loss function (2.4) subjected to the neural gradient flow (2.3). From the perspective of control, we only need to find an optimal parameter θ^* for dynamic system (2.3) such that the loss function (2.4) is minimized. This motivates us to formulate it as an optimal control problem:

$$\begin{aligned} \min_{\theta \in \mathcal{A}} \quad & J(\theta) = \sum_{i=1}^{n} L_i(y(t_i)), \\ \text{s.t.} \quad & \dot{y}(t) = -\nabla_y G(y(t), \theta) \quad t \in (0, T], \quad y(0) = x_0, \end{aligned} \tag{2.6}$$

where $\mathcal{A} \subset \mathbb{R}^N$ is the control set, $t_0 = 0$, $t_n = T$ and

$$L_i(y(t_i)) := \|y(t_i) - x_i\|^2, \quad 1 \le i \le n. \tag{2.7}$$

Here L_i is a local loss that measures the error between the solution to neural gradient flow (2.3) and the observed data at t_i. When $n = 1$, this reduces to the usual optimal control. We solve this optimal control problem by iteration with gradient-based methods to update θ. For instance, given θ_k, the vanilla gradient descent (GD) computes θ_{k+1} by

$$\theta_{k+1} = \theta_k - \eta \nabla J(\theta_k), \tag{2.8}$$

where η is a step size. One of the main tasks here is to compute the gradient $\nabla J(\theta)$. This can be obtained via backpropagation through ODE solvers, which gives a discrete approximation to the dynamical system. Another approach to computing the gradient is to use the adjoint method, which is summarized in Theorem 1.

2.5 Compute the gradient

The following result allows computation of the gradient $\nabla J(\theta)$.

Theorem 1 *If $(y(t), \theta)$, $0 \le t \le T$ is the state control trajectory starting from $x(0)$, then there exists a co-state trajectory $p(t)$ satisfying*

$$\begin{aligned} &\dot{y}(t) = -\nabla_y G(y(t), \theta), \quad y(0) = x_0, \\ &\dot{p}(t) = \left(\nabla_y^2 G(y(t), \theta)\right)^\top p(t), \quad t_{i-1} \le t < t_i, \quad i = n, ..., 1, \\ &p(T) = \nabla_y L_n(y(T)),\ p(t_i^-) = p(t_i^+) + \nabla_y L_i(y(t_i)), \quad i = n-1, ..., 1. \end{aligned} \tag{2.9}$$

Moreover, the gradient of J can be evaluated by

$$\nabla J = -\int_0^T \left(\nabla_\theta \nabla_y G(y(t), \theta)\right)^\top p(t) dt. \tag{2.10}$$

This allows us to compute ∇J at each iteration, say when $\theta = \theta_k$, in three steps:

Step 1. Solve the forward problem to obtain state $y_k(t) := y(t; \theta_k)$,

Step 2. Solve the piece-wise backward problem to obtain co-state p_k,

Step 3. Evaluate the gradient of J by (2.10), which gives the needed $\nabla J(\theta_k)$.

In our experiments, we use both approaches, backpropagation or the adjoint method, to compute the gradient, depending on which method is easier to compute for a specific example. The computational procedure for the adjoint method used in our experiments follows the framework introduced in Chen et al. (2018). For the sake of completeness, we summarize the details in Appendix A.1.

In practice, some real-world systems are not in the form of gradient flows, our algorithm is readily extended to encompass these situations, allowing for the discovery of general ODE systems

$$\dot{x}(t) = F(x(t)), \tag{2.11}$$

where $F : \mathbb{R}^d \to \mathbb{R}^d$ is unknown. In such case, Theorem 1 needs to be modified by replacing $-\nabla_y G(y(t), \theta)$ by $G(y(t), \theta)$, where $G(\cdot, \theta) : \mathbb{R}^d \to \mathbb{R}^d$ is a neural network approximator of F. We also conducted some numerical tests on this type of problem; see Section 4.3. Finally, we should point out that *a priori* knowledge of properties of G (say, smoothness) can be incorporated to improve the performance of OCN.

2.6 Data generation

In this work, we assume the training data are collected from one or multiple trajectories of the dynamical system with randomly chosen initial points. To simulate this process, we generate the training data in our numerical experiments in the following way:

- We first generate m initial points from the uniform distribution over a domain, in which we would like to learn the dynamical behavior of the solutions. Denote $y^{(j)}$ as the solution to the neural gradient flow in (2.6) starting with the j-th initial point, the loss function in (2.6) becomes
$$J(\theta) = \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n} L_i(y^{(j)}(t_i)).$$
- Starting with each initial point, we generate $\{x_i\}_{i=1}^n$ over time interval $[0,T]$ with $\Delta t = t_{i+1} - t_i$ for $i = 1, ..., n-1$ by solving the true dynamical system using a high accuracy ODE solver. For simplicity of notation, we assume the time interval $[0,T]$, number of data points n, and the distance between two neighboring data points Δt are the same for all trajectories.

3 Error Analysis

In this section, we present theoretical results on the convergence behavior and the generalization error for OCN.

Let $f(x)$ be such that $\nabla f(x)$ is Lipschitz continuous, and $x(t)$ be the unique solution to the ODE, and $y_k(t) := y(t;\theta_k)$ be the solution corresponding to the network parameter θ_k at the $k-$th iteration of the optimization. These are functions evaluated at any point $t \in [0,T] = [t_0, t_n]$. We want to bound the generalization error
$$e_k(t) = \|x(t) - y_k(t)\|.$$
We shall show that the generalization error is bounded by respectively the optimization error $J(\theta_K)$ and time step $O(\Delta t)$ with $\Delta t = \max_{0\le i\le n-1}|t_{i+1} - t_i|$.

To quantify the errors and also control their propagation in time we make the following assumptions:

Assumption 1. $f \in C^1(\mathbb{R}^d)$ and ∇f is Lipschitz continuous with constant L_f:
$$\|\nabla f(x) - \nabla f(z)\| \le L_f\|x - z\|, \quad \forall x, z \in \mathbb{R}^d.$$
Assumption 1 is a sufficient condition for the existence and uniqueness of the solution to (2.5). This is also used to control the truncation error in the discrete ODE (2.2).

Assumption 2. $G \in C^1(\mathbb{R}^d \times \mathbb{R}^N)$ and there exist constant L_{G_y} such that for any $\theta \in \mathcal{A}$,
$$\|\nabla_y G(y,\theta) - \nabla_y G(z,\theta)\| \le L_{G_y}\|y - z\|, \quad \forall y, z \in \mathbb{R}^d.$$
Assumption 2 plays a similar role for the neural gradient flows as in Assumption 1 for the target ODE system. Here Assumption 2 can be ensured by the choice of activation functions in the construction of deep neural networks.

The main result is stated as follows.

Theorem 2 *Let Assumption 1 and 2 hold respectively on the structure of f and network prediction G. Suppose that $\theta_k \in \mathcal{A}$, where θ_k is generated by gradient descent method (2.8) with gradient be computed using Theorem 1, and $\mathcal{A}$ is bounded. If $\Delta t = \max_{0\le i\le n-1}|t_{i+1} - t_i| \le \frac{1}{2L_{G_y}}$, then*
$$\max_{t\in[0,T]}\|x(t) - y_k(t)\| \le C_1(\sqrt{J(\theta_k)} + (\Delta t)^2). \tag{3.1}$$
In addition,
$$\max_i\|\nabla f(x_i) - \nabla_y G(x_i,\theta_k)\| \le C_2\left(\frac{\sqrt{J(\theta_k)}}{\Delta t} + \Delta t\right), \tag{3.2}$$
where $J(\theta_k)$ is the training loss defined by (2.4), C_1, C_2 are constants depend on the data, control set $\mathcal{A}$, and structural parameters for f and G in Assumptions 1 and 2.

Due to space constraints, a detailed proof is relegated to Appendix A.3.

In an asymptotic manner, we have $\lim_{k\to\infty} J(\theta_k) = J(\theta^*)$, which is zero or rather small, then the generalization error in (3.1) will ultimately be dominated by $(\Delta t)^2$, which is determined by how dense the data is collected over time. We note that the bound in (3.2) is not optimal.

4 Experimental results

In this section, we demonstrate the proposed method on several canonical systems. For all experiments, we use feed-forward neural networks with tanh activation function. The detailed structure of the neural network applied for each problem is provided. All the weights are initialized randomly from Gaussian distributions, and all the biases are initialized to zero. During training, each trajectory is divided into several mini-batches, and all batches of data are trained simultaneously. After the neural network is well trained, we generate $\{y(t_i)\}_{i=1}^n$ from the learned dynamics $\dot{y} = -\nabla_y G(y, \cdot)$ (or $\dot{y} = G(y, \cdot)$) and compare it against the training data $\{x_i\}_{i=1}^n$. We further investigate the performance of the trained neural network by applying it to the observed data $\{x_i\}_{i=0}^n$. Specifically, we compute $G(x_i, \cdot)$ and compare it against $f(x_i)$.

For experiments on the gradient flow problem, we also test the performance of the trained neural network by applying it to test data, which are some initial points generated randomly over the same domain and do not appear in the training data. Just like the comparison of the training result, we also compare the trajectories given by neural gradient flow and that given by the true dynamics.

For each experiment, we provide the true dynamical system, which is used to generate the training data and verify the performance of the trained neural network, but in no way facilitates the neural network approximation of form $G(y, \theta)$.

4.1 Linear gradient flow

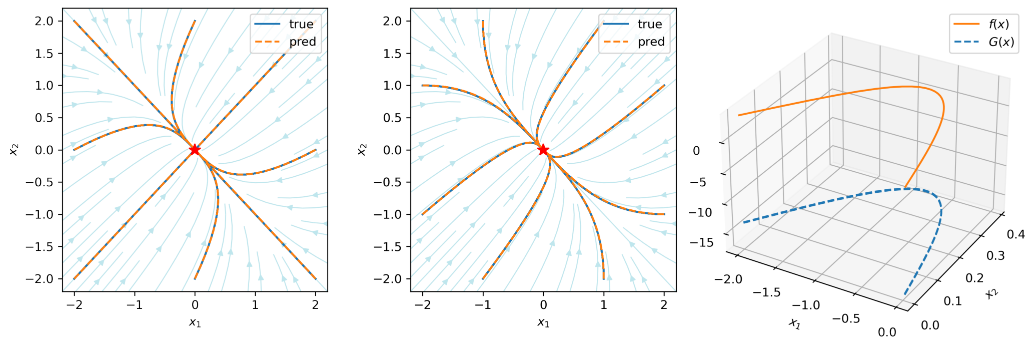
For this example, the training data is collected on solution trajectories to

$$\dot{x}_1 = -2x_1 - x_2,$$
$$\dot{x}_2 = -x_1 - 2x_2,$$

This is of form $\dot{x} = -\nabla f(x)$ where

$$f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2. \tag{4.1}$$

This system has critical point $(0, 0)$ as a stable node. All solution trajectories tend to $(0, 0)$ as $t \to \infty$. We want to extract f from the training data, which is sampled from 8 trajectories on domain $[-2, 2] \times [-2, 2]$ with time interval $[0, 5]$ and time step $\Delta t = 0.05$. The neural network G used to approximate f in (4.1) has 2 hidden layer of 50 neurons.



(a) Training result (b) Test result (c) Function

Figure 1: Results of the linear gradient flow. For (a) and (b), the star represents the minimizer of f in 4.1. For (c), two curves for $G(x)$ correspond to two independent runs of our method with different initial parameters θ_0 for $G(x, \theta)$.

The training and test results are presented in Figure 1 (a) and (b), respectively. We observe that all trajectories generated by the trained neural gradient flow (2.3) fit the data generated by the true dynamical system accurately.

Figure 1 (c) is a comparison between the true governing function $f(x)$ and the trained neural network $G(x, \cdot)$, where x represents the training data set $\{x_i\}$. There is a distance between them because the original problem (2.1) is uniquely determined by $f + c$ for any constant c. For $G(x, \cdot)$ that satisfies (2.1), $G(x, \cdot) + c$ also satisfies (2.1) for any constant c.

4.2 Nonlinear gradient flow

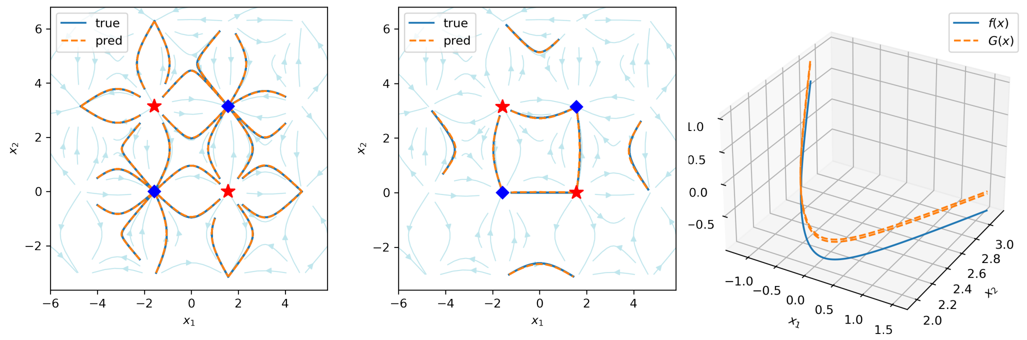
For this example, the training data is collected on solution trajectories to

$$\begin{aligned} \dot{x}_1 &= -\cos(x_1)\cos(x_2), \\ \dot{x}_2 &= \sin(x_1)\sin(x_2), \end{aligned} \tag{4.2}$$

This is of form $\dot{x} = -\nabla f(x)$ where

$$f(x_1, x_2) = \sin(x_1)\cos(x_2). \tag{4.3}$$

This system has three types of nodes – stable nodes, unstable nodes, and saddle points – spread over the domain in a staggered pattern. The training data consists of 24 trajectories sampled from domain $[-6, 6] \times [-4, 6]$ with time interval $[0, 8]$ and $\Delta t = 0.05$. The neural network G used to approximate f in (4.3) has 2 hidden layers of 200 neurons.



(a) Training result (b) Test result (c) Function

Figure 2: Results of the nonlinear gradient flow. The stars represent unstable nodes, the squares represent stable nodes.

The training results are presented in Figure 2 (a). We observe that for trajectories around different types of nodes, either diverging from sources or converging to sinks, the trained neural gradient flow (2.3) fits the training data accurately.

The performance of the trained neural network on test data is shown in Figure 2 (b). The test data is composed of 8 initial points, among which 4 initial points (in the center of the figure) correspond to trajectories that have a similar pattern to that of the training data; another 4 initial points correspond to trajectories whose dynamic behavior is different from that of the training data. For both types of initial points, the trained neural gradient flow recovers their corresponding trajectories accurately.

4.3 Damped pendulum

To show that our method can be applied to general ODE systems, we consider the pendulum problem, which has the form of $\dot{x}(t) = f(x(t))$. Specifically,

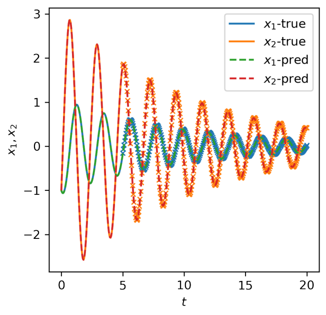
$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= -0.2x_2 - 8.91\sin(x_1). \end{aligned}$$

Here x_1 is the angular displacement, and x_2 is angular velocity. This is a damped system that obeys a dissipation law:

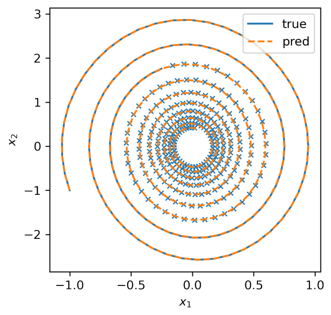
$$\frac{d}{dt}\left(\frac{x_2^2}{2} + 8.91(1 - \cos(x_1))\right) = -0.2x_2^2 \leq 0.$$

The critical point $(0, 0)$ is a stable spiral. The training data is collected from 1 trajectory starting from $[-1, -1]$ within time interval $[0, 5]$ and time step $\Delta t = 0.05$. The neural network G used to approximate f has 1 hidden layer of 100 neurons.

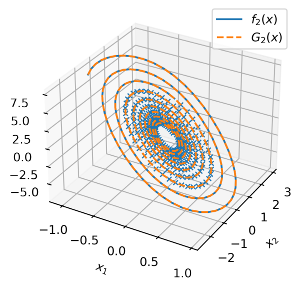
After finishing training, we generate a trajectory over $[0, 20]$ to examine the relatively long-term predictive behavior of the neural gradient flow. The results are presented in Figure 3. We observe accurate fitting between the true trajectory and the trajectory generated by the neural gradient flow, even on a time horizon that is much larger than what was used during training.



(a) Trajectory



(b) Phase portrait



(c) Function

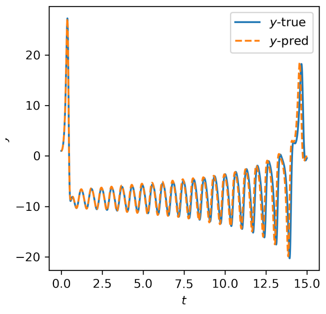
Figure 3: Results of the nonlinear ODE system.

4.4 Lorenz system

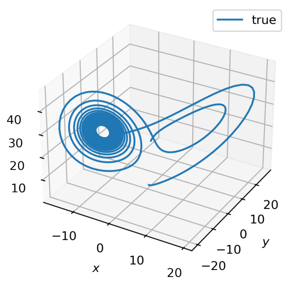
We also demonstrate our method on the 3D Lorenz system:

$$\begin{aligned}\dot{x} &= \sigma(y - x),\\ \dot{y} &= x(\rho - z) - y,\\ \dot{z} &= xy - \beta z.\end{aligned} \tag{4.4}$$

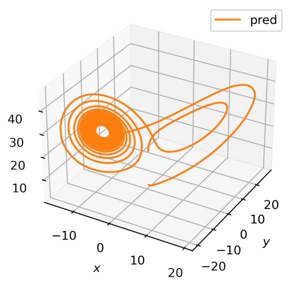
The dynamics is very rich for different parameters. The well-known Lorenz attractor shows up for $(\sigma, \rho, \beta) = (10, 28, 8/3)$. The training data is collected from 1 trajectory starting from $[0, 1, 1.05]$, and the time interval is $[0, 15]$ with $\Delta t = 0.01$. The neural network G used to approximate f has 4 hidden layers of 500 neurons. The training results presented in Figure 4 demonstrate that our method does well in capturing the chaotic dynamics.



(a) Trajectory - y



(b) True dynamics



(c) Learned dynamics

Figure 4

5 Discussion

In summary, we have demonstrated a powerful approach to discovering gradient flows from data without assumptions on the form of the governing equations. This builds on prior work in data-driven discovery of ODEs using machine learning techniques, but with innovations related to a global network representation of the vector field and an optimal control formulation, which allow our algorithm to scale to more complex problems.

We demonstrate this method on a number of example systems, ranging from linear gradient flow, nonlinear gradient flow, and the damped pendulum, to the Lorenz system exhibiting chaos. As shown in these examples, our method has the ability to predict a particular trajectory, and capture complex dynamics. There are many dynamical systems to which this method may be applied, where there are ample data with the absence of governing equations. We believe our method is an important step toward the long-held goal of intelligent, unassisted discovery of dynamical systems.

The general form of the loss function allows for incorporating further knowledge or regularization, so to make the methods more accurate and robust. We have derived generalization error bounds for both the solution and the vector field. Specifically, we prove that the generalization error depends on both the optimization error and the sparsity level of the time series data. We achieve this by carefully studying the error equation and obtaining a priori bounds.

We see several avenues for work, both theoretical and computational. For example, is it possible to improve the error bounds for $\|\nabla f - \nabla G\|$? WHat if we assume more structure on the dynamics? How can we improve the computational efficiency of solving the coupled control system? Can we deploy this to learn the dynamics of truly large-scale problems?

We now briefly discuss possible extensions of our method. For systems with time dependence, such as $\dot{x} = F(x,t)$, for which we consider the augmented system

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

For systems with physical parameters, $\dot{x} = F(x,\mu)$, then μ can be appended to the dynamics in the following way

$$\dot{x} = F(x,u), \quad \dot{u} = 0.$$

It is then possible to use neural networks to represent $F(x,u)$ or $F(x,\mu)$.

Finally, we illustrate how gradeint flows in the form of PDEs can be reduced by the method of lines as a finite dimensional gradient flow so that our method can be applied. We take the Allen-Cahn equation

$$\partial_t \phi(x,t) = \epsilon^2 \Delta\phi(x,t) - F'(\phi(x,t)),$$

with energy

$$E(\phi) := \int_\Omega (F(\phi) + \frac{\epsilon^2}{2}|\nabla\phi|^2)dx.$$

an an example. We restrict to the one-dimensional periodic torus. Applying the finite difference method to the Allen-Cahn equation over the grid points $\{x_i\}$ with uniform mesh $x_{i+1} - x_i = h$, we have

$$\frac{d}{dt}\phi_i = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{h^2} - F'(\phi_i), \quad \phi_{N+1} = \phi_1, \quad \phi_0 = \phi_N.$$

Here the corresponding discrete energy is

$$E_h(\phi) = \sum_{i=1}^{N} F(\phi_i) + \frac{\epsilon^2}{2h^2}(\phi_{i+1} - \phi_i)^2.$$

One can verify that

$$\frac{d}{dt}\Phi = -\nabla_\Phi E_h(\Phi),$$

where Φ represents $[\phi_1, ..., \phi_N]$. This is a gradient flow in $\mathbb{R}^N$.

REFERENCES

Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2005.

Justin Baker, Hedi Xia, Yiwei Wang, Elena Cherkaev, Akil Narayan, Long Chen, Jack Xin, Andrea L Bertozzi, Stanley J Osher, and Bao Wang. Proximal implicit ode solvers for accelerating learning neural odes. *arXiv preprint arXiv:2204.08621*, 2022.

Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

Martin Benning, Elena Celledoni, Matthias J Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: Models and numerical methods. *arXiv preprint arXiv:1904.05657*, 2019.

Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

Steven L Brunton, Bingni W Brunton, Joshua L Proctor, Eurika Kaiser, and J Nathan Kutz. Chaos as an intermittently forced linear system. *Nature communications*, 8(1):1–9, 2017.

Kathleen Champion, Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116 (45):22445–22451, 2019a.

Kathleen P Champion, Steven L Brunton, and J Nathan Kutz. Discovery of nonlinear multiscale systems: Sampling strategies and embeddings. *SIAM Journal on Applied Dynamical Systems*, 18 (1):312–333, 2019b.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

James P Crutchfield and BS McNamara. Equations of motion from a data series. *Complex systems*, 1:417–452, 1987.

Bryan C Daniels and Ilya Nemenman. Automated adaptive inference of phenomenological dynamical models. *Nature communications*, 6(1):1–8, 2015.

Talgat Daulbaev, Alexandr Katrutsa, Larisa Markeeva, Julia Gusak, Andrzej Cichocki, and Ivan Oseledets. Interpolation technique to speed up gradients propagation in neural odes. *Advances in Neural Information Processing Systems*, 33:16689–16700, 2020.

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

Amir Gholami, Kurt Keutzer, and George Biros. ANODE: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019.

Dimitrios Giannakis and Andrew J Majda. Nonlinear laplacian spectral analysis for time series with intermittency and low-frequency variability. *Proceedings of the National Academy of Sciences*, 109(7):2222–2227, 2012.

Raul González-García, Ramiro Rico-Martìnez, and Ioannis G Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & chemical engineering*, 22:S965–S968, 1998.

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.

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

Ioannis G Kevrekidis, C William Gear, James M Hyman, Panagiotis G Kevrekidis, Olof Runborg, Constantinos Theodoropoulos, et al. Equation-free, coarse-grained multiscale computation: enabling microscopic simulators to perform system-level analysis. *Commun. Math. Sci*, 1(4):715–762, 2003.

Qianxiao Li and Shuji Hao. An optimal control approach to deep learning and applications to discrete-weight neural networks. In *International Conference on Machine Learning*, pp. 2985–2994. PMLR, 2018.

Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *Journal of Machine Learning Research*, 18(165):1–29, 2018.

Hailiang Liu and Peter Markowich. Selection dynamics for deep neural networks. *Journal of Differential Equations*, 269(12):11540–11574, 2020.

Hailiang Liu and Xuping Tian. Data-driven optimal control of a SEIR model for COVID-19. *Communications on Pure and Applied Analysis*, 2021.

Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepOnet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

Takashi Matsubara, Yuto Miyatake, and Takaharu Yaguchi. Symplectic adjoint method for exact gradient of neural ode with minimal memory. *Advances in Neural Information Processing Systems*, 34:20772–20784, 2021.

Joshua L Proctor, Steven L Brunton, Bingni W Brunton, and JN Kutz. Exploiting sparsity and equation-free architectures in complex systems. *The European Physical Journal Special Topics*, 223(13):2665–2684, 2014.

Tong Qin, Kailiang Wu, and Dongbin Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. SNODE: Spectral discretization of neural odes for system identification. *arXiv preprint arXiv:1906.07038*, 2019.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

Anthony John Roberts. *Model emergent dynamics in complex systems*, volume 20. SIAM, 2014.

Samuel Rudy, Alessandro Alla, Steven L Brunton, and J Nathan Kutz. Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, 18(2): 643–660, 2019.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

Michael D Schmidt, Ravishankar R Vallabhajosyula, Jerry W Jenkins, Jonathan E Hood, Abhishek S Soni, John P Wikswo, and Hod Lipson. Automated refinement and inference of analytical models for metabolic networks. *Physical biology*, 8(5):055011, 2011.

George Sugihara, Robert May, Hao Ye, Chih-hao Hsieh, Ethan Deyle, Michael Fogarty, and Stephan Munch. Detecting causality in complex ecosystems. *science*, 338(6106):496–500, 2012.

Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Hao Ye, Richard J Beamish, Sarah M Glaser, Sue CH Grant, Chih-hao Hsieh, Laura J Richards, Jon T Schnute, and George Sugihara. Equation-free mechanistic ecosystem forecasting using empirical dynamic modeling. *Proceedings of the National Academy of Sciences*, 112(13):E1569–E1576, 2015.

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *Advances in Neural Information Processing Systems*, 32, 2019.

A Appendix

A.1 Numerical computation of the optimal control system

From (2.10), we see that computing ∇J requires the value of $y(t)$ and $p(t)$ along $t \in [0,T]$. The value of $y(t)$ can be obtained by solving the forward problem (A.1), which can be done by calling an ODE solver. The tricky part is to solve the backward problem in (2.9) for $p(t)$, since it requires the value of $y(t)$ along the trajectory of $p(t)$. This difficulty is resolved by recomputing $y(t)$ backward in time with $p(t)$ using $y(T)$ computed in Step 1 as the initial data. For Step 3, we note that $\nabla J = \xi(0)$ with ξ defined by

$$\xi(t) = -\int_t^T \left(\nabla_\theta \nabla_y G(y(s,\theta))\right)^\top p(s)ds.$$

Hence ∇J can be obtained by solving the following equation

$$\dot{\xi}(t) = \left(\nabla_\theta \nabla_y G(y(t,\theta))\right)^\top p(t)$$

backward in time with initial condition $\xi(T) = \mathbf{0}$. Numerically, we concatenate y, p, ξ into a single vector U, such that all the backward computation can be conducted at the same time by solving the following augmented system:

$$\dot{U}(t) = F(U(t),\theta),$$

where

$$U = \begin{bmatrix} y \\ p \\ \xi \end{bmatrix}, \quad F(U(t),\theta) = \begin{bmatrix} -\nabla_y G(y(t),\theta) \\ \left(\nabla_y^2 G(y(t),\theta)\right)^\top p(t) \\ \left(\nabla_\theta \nabla_y G(y(t),\theta)\right)^\top p(t) \end{bmatrix}.$$

Algorithm 1

```
Require: $\{x_i\}_{i=0}^n$: data, $G(\cdot,\theta)$: parameterized neural network, $\theta_0$: initial guess, $\eta$: step size for
    the optimizer, $K$: total number of iterations.
1: for $k = 0$ to $K-1$ do
2:     Solve the forward problem:
           $y(T) = \text{ODEsolver}\begin{pmatrix} \dot{y}(t) = -\nabla_y G(y(t),\theta_k), \\ y(0) = x_0 \end{pmatrix}$
3:     Solve the augmented backward system:
           $[y(0), p(0), \nabla J(\theta_k)] = \text{ODEsolver}\begin{pmatrix} \dot{U}(t) = F(U(t),\theta_k), \\ U(T) = [y(T), \nabla_y L_n(y(T)), \mathbf{0}], \\ \nabla_y L_i(y(t_i)), \quad i = 1,...,n-1 \end{pmatrix}$
4:     Update the parameter: $\theta_{k+1} \leftarrow \text{Optimizer}(\theta_k, \nabla J(\theta_k), \eta)$
5: return $\theta_K$
```

Remark 3 *The augmented backward system is solved piece-wisely as the initial condition for the co-state p needs to be adjusted between each intervals as indicated in (2.9).*

Algorithm 1 is a general framework for solving the optimal control problem (2.9). To extend it or make it more concrete, one may apply different ODE solvers such as the Runge-Kutta method to solve the forward and backward problems depending on the desired level of accuracy.

A.2 Proof of Theorem 1

The computation of the gradient of J can be realized in the following recipe when $y = y(t;\theta)$ has been found to solve the following forward problem:

$$\dot{y}(t) = -\nabla_y G(y(t),\theta), \quad y(0) = x_0. \tag{A.1}$$

(i) Build an augmented functional (associated Lagrangian) $\mathcal{L}$, a functional of independent variables $\tilde{y}, p, \theta$ defined by

$$\mathcal{L}(\tilde{y},p,\theta) = \sum_{i=1}^n L_i(\tilde{y}(t_i)) - \int_0^T (\dot{\tilde{y}}(t) + \nabla_{\tilde{y}} G(\tilde{y}(t),\theta))^\top p(t)dt,$$

where p is the Lagrange multiplier, and can be chosen freely. Taking $\tilde{y} = y$, we have

$$\mathcal{L}(y, p, \theta) = \sum_{i=1}^{n} L_i(y(t_i)) = J(\theta). \tag{A.2}$$

In order to evaluate $\nabla_\theta J$, we proceed to calculate the first variation of $\mathcal{L}(\tilde{y}, p, \theta)$ at (y, θ), defined by

$$\delta\mathcal{L}(y, p, \theta) := \lim_{\tau \to 0} \frac{\mathcal{L}(y + \tau\delta y, p, \theta + \tau\delta\theta) - \mathcal{L}(y, p, \theta)}{\tau},$$

from which we will see why p should be chosen as in (2.9).

(ii) Defining the adjoint-state equations for p. By formal calculations, we obtain

$$\begin{aligned}
&\delta\mathcal{L}(y, p, \theta) \\
&= \delta \sum_{i=1}^{n} \Big(L_i(y(t_i)) - \int_{t_{i-1}}^{t_i} \Big(\dot{y}(t) + \nabla_y G(y(t), \theta)\Big)^\top p(t)dt \Big) \\
&= \sum_{i=1}^{n} \Big(\delta y(t_i)^\top \nabla_y L_i(y(t_i)) - \int_{t_{i-1}}^{t_i} \Big(\delta\dot{y}(t) + \delta\nabla_y G(y(t), \theta)\Big)^\top p(t)dt \Big) \\
&= \sum_{i=1}^{n} \Big(\delta y(t_i)^\top \nabla_y L_i(y(t_i)) - \delta y(t_i)^\top p(t_i^-) + \delta y(t_{i-1})^\top p(t_{i-1}^+) \\
&\qquad + \int_{t_{i-1}}^{t_i} (\delta y)^\top \dot{p}(t) - \Big(\nabla_y^2 G(y(t), \theta)\delta y + \nabla_\theta \nabla_y G(y(t), \theta)\delta\theta\Big)^\top p(t)dt \Big) \\
&= \delta y(T)^\top \Big(\nabla_y L_n(y(T)) - p(T)\Big) + \delta y(0)^\top p(0) + \sum_{i=1}^{n-1} \delta y(t_i)^\top \Big(\nabla_y L_i(y(t_i)) - p(t_i^-) + p(t_i^+)\Big) \\
&\qquad + \sum_{i=1}^{n} \Big(\int_{t_{i-1}}^{t_i} (\delta y)^\top \Big(\dot{p}(t) - \big(\nabla_y^2 G(y(t), \theta)\big)^\top p(t)\Big) - (\delta\theta)^\top \Big(\big(\nabla_\theta \nabla_y G(y(t), \theta)\big)^\top p(t)\Big) dt \Big),
\end{aligned}$$

where we have used integration by parts, and regrouping of terms. Since $y(0) = x_0$ is fixed, $\delta y(0) = 0$; if p is taken to satisfy (2.9), then

$$\delta\mathcal{L}(y, p, \theta) = -(\delta\theta)^\top \int_0^T \big(\nabla_\theta \nabla_y G(y(t), \theta)\big)^\top p(t)dt.$$

(iii) Computation of the gradient of J. Recall (A.2), the first variation of $J(\theta)$ is actually $\nabla J \cdot \delta\theta$, we thus conclude

$$\nabla J = -\int_0^T \big(\nabla_\theta \nabla_y G(y(t), \theta)\big)^\top p(t)dt,$$

as asserted in (2.10).

A.3 Proof of Theorem 2

It suffices to prove that the stated result holds for any $t \in [0, T]$. Without loss of generality, we assume $t \in I_i := (t_i, t_{i+1}]$ for some $i \in \{0, 1, ..., n-1\}$. Using the notation

$$e_k(t) := \|y_k(t) - x(t)\|,$$

and (2.3), (2.5), we get

$$\frac{d}{dt} e_k^2 = 2(y_k - x) \cdot \frac{d}{dt}(y_k - x) \leq 2e_k \|\nabla f(x) - \nabla_y G(y_k, \theta_k)\|,$$

which is estimated by the Cauchy-Schwarz inequality. This further implies

$$\begin{aligned}
\dot{e}_k &\leq \|\nabla f(x) - \nabla_y G(y_k, \theta_k)\| \\
&\leq \|\nabla f(x) - \nabla f(y_k)\| + \|\nabla f(y_k) - \nabla_y G(y_k, \theta_k)\| \\
&\leq L_f e_k + R(y_k).
\end{aligned} \tag{A.3}$$

Here we used the assumption that ∇f is L_f Lipschitz continuous and the notation

$$R(y_k(t)) := \|\nabla f(y_k(t)) - \nabla_y G(y_k(t), \theta_k)\|.$$

Rewriting (A.3) against an integrating factor $e^{-L_f t}$ we obtain

$$\frac{d}{dt}(e^{-L_f t} e_k(t)) \leq e^{-L_f t} R(y_k(t)).$$

Integration of this over (t_i, t) gives

$$\begin{aligned} e_k(t) &\leq e^{L_f(t-t_i)} e_k(t_i) + \int_{t_i}^{t} e^{L_f(t-s)} R(y_k(s)) ds \\ &\leq e^{L_f \Delta t}\Big(e_k(t_i) + \Delta t \max_{t \in I_i} R(y_k(t))\Big), \end{aligned} \tag{A.4}$$

where $|t_{i+1} - t_i| \leq \max_i |t_{i+1} - t_i| =: \Delta t$ is used.

We now proceed to bound the right hand side (RHS) of (A.4). First notice that

$$e_k(t_i) = \sqrt{\|y_k(t_i) - x_i\|^2} \leq \sqrt{J(\theta_k)}. \tag{A.5}$$

For $R(y_k(t))$, we use triangle inequality to obtain

$$\begin{aligned} R(y_k(t)) \leq\ & \|\nabla f(y_k(t)) - \nabla f(y_k(t_i))\| \\ & + \|\nabla_y G(y_k(t_i), \theta_k) - \nabla_y G(y_k(t), \theta_k)\| \\ & + \|\nabla f(y_k(t_i)) - \nabla_y G(y_k(t_i), \theta_k)\|, \end{aligned}$$

which implies

$$\max_{t \in I_i} R(y_k(t)) \leq D_1 + D_2 + D_3, \tag{A.6}$$

where

$$\begin{aligned} D_1 &= \max_{t \in I_i} \|\nabla f(y_k(t)) - \nabla f(y_k(t_i))\|, \\ D_2 &= \max_{t \in I_i} \|\nabla_y G(y_k(t_i), \theta_k) - \nabla_y G(y_k(t), \theta_k)\|, \\ D_3 &= \|\nabla f(y_k(t_i)) - \nabla_y G(y_k(t_i), \theta_k)\|. \end{aligned}$$

We further derive bounds on D_1, D_2, D_3. The derivation of bounds on D_1 and D_2 are similar. The idea is to use L_f Lipschitz continuity of ∇f and L_{G_y}, respectively with respect to y to get

$$\begin{aligned} D_1 &\leq L_f \max_{t \in I_i} \|y_k(t) - y_k(t_i)\|, \\ D_2 &\leq L_{G_y} \max_{t \in I_i} \|y_k(t) - y_k(t_i)\|, \end{aligned}$$

then show the following bound

$$\max_{t \in I_i} \|y_k(t) - y_k(t_i)\| \leq \frac{\Delta t}{1 - \Delta t L_{G_y}} \Big(\|\nabla_y G(x_i, \theta_k)\| + L_{G_y}\sqrt{J(\theta_k)}\Big). \tag{A.7}$$

Hence for $\Delta t \leq \frac{1}{2L_{G_y}}$, we have

$$D_1 + D_2 \leq C_0 \Delta t. \tag{A.8}$$

where

$$C_0 = 2\Big(\|\nabla_y G(x_i, \theta_k)\| + L_{G_y}\sqrt{J(\theta_k)}\Big)(L_f + L_{G_y}).$$

For the derivation of (A.7), we start with

$$\max_{t \in I_i} \|y_k(t) - y_k(t_i)\| = \max_{t \in I_i} \|\int_{t_i}^{t} \nabla_y G(y_k(s), \theta_k) ds\| \leq \Delta t \max_{t \in I_i} \|\nabla_y G(y_k(t), \theta_k)\|. \tag{A.9}$$

Using the L_{G_y} Lipschitz continuity of $\nabla_y G$ with respect to y, we have

$$\begin{aligned} \|\nabla_y G(y_k(t), \theta_k) - \nabla_y G(x_i, \theta_k)\| &\leq L_{G_y} \|y_k(t) - x_i\| \\ &\leq L_{G_y}(\|y_k(t) - y_k(t_i)\| + \|y_k(t_i) - x_i\|), \end{aligned}$$

which together with (A.5) lead to

$$\max_{t\in I_i}\|\nabla_y G(y_k(t),\theta_k)\| \le \|\nabla_y G(x_i,\theta_k)\| + L_{G_y}\sqrt{J(\theta_k)} + L_{G_y}\max_{t\in I_i}\|y_k(t)-y_k(t_i)\|. \quad \text{(A.10)}$$

Connecting (A.9) and (A.10), we obtain (A.7).

For the bound on D_3, we use triangle inequality to get

$$\begin{aligned} D_3 \le\ & \|\nabla f(y_k(t_i)) - \nabla f(x_i)\| + \|\nabla f(x_i) + \frac{x_{i+1}-x_i}{\Delta t}\| \\ & + \|-\frac{y_k(t_{i+1})-y_k(t_i)}{\Delta t} - \nabla_y G(y_k(t_i),\theta_k)\| + \|\frac{y_k(t_{i+1})-y_k(t_i)}{\Delta t} - \frac{x_{i+1}-x_i}{\Delta t}\|. \end{aligned} \quad \text{(A.11)}$$

The first term on the RHS of (A.11) can be bounded by

$$\|\nabla f(y_k(t_i)) - \nabla f(x_i)\| \le L_f e_k(t_i) \le L_f\sqrt{J(\theta_k)}, \quad \text{(A.12)}$$

using the L_f Lipschitz continuous of ∇f and (A.5).

For the second and third term on the RHS of (A.11), note that Assumption 1 and 2 also imply

$$\begin{aligned} x(t_{i+1}) &\le x(t_i) - \Delta t\nabla f(x(t_i)) + \frac{L_f}{2}(\Delta t)^2, \\ y(t_{i+1}) &\le y(t_i) - \Delta t\nabla_y G(y_k(t_i),\theta_k) + \frac{L_{G_y}}{2}(\Delta t)^2. \end{aligned}$$

Since $x(t_i) = x_i$, we have

$$\begin{aligned} &\|\nabla f(x_i) + \frac{x_{i+1}-x_i}{\Delta t}\| \le \frac{L_f}{2}\Delta t, \\ &\|-\frac{y_k(t_{i+1})-y_k(t_i)}{\Delta t} - \nabla_y G(y_k(t_i),\theta_k)\| \le \frac{L_{G_y}}{2}\Delta t. \end{aligned} \quad \text{(A.13)}$$

For the last term on the RHS of (A.11), we use triangle inequality and (A.5) to get

$$\begin{aligned} &\|\frac{y_k(t_{i+1})-y_k(t_i)}{\Delta t} - \frac{x_{i+1}-x_i}{\Delta t}\| \\ \le\ & \frac{1}{\Delta t}\Big(\|y_k(t_{i+1})-x_{i+1}\| + \|y_k(t_i)-x_i\|\Big) \le \frac{2\sqrt{J(\theta_k)}}{\Delta t}. \end{aligned} \quad \text{(A.14)}$$

Substituting (A.12), (A.13), (A.14) into (A.11), we obtain the following bound on D_3

$$D_3 \le L_f\sqrt{J(\theta_k)} + \frac{L_f + L_{G_y}}{2}\Delta t + \frac{2\sqrt{J(\theta_k)}}{\Delta t}. \quad \text{(A.15)}$$

With bounds on D_1, D_2 in (A.8) and D_3 in (A.15), (A.6) becomes

$$\max_{t\in I_i} R(y_k(t)) \le C_0\Delta t + (L_f + \frac{2}{\Delta t})\sqrt{J(\theta_k)}.$$

This together with (A.5), (A.4) and $\Delta t \le \frac{1}{2L_{G_y}}$ leads to

$$\begin{aligned} e_k(t) &\le e^{L_f\Delta t}\Big(\sqrt{J(\theta_k)} + C_0(\Delta t)^2 + (L_f\Delta t + 2)\sqrt{J(\theta_k)}\Big), \\ &\le C_1\Big(\sqrt{J(\theta_k)} + (\Delta t)^2\Big), \end{aligned}$$

where

$$C_1 = e^{\frac{L_f}{2L_{G_y}}}\max\Big\{C_0, 3 + \frac{L_f}{2L_{G_y}}\Big\},$$

which further implies (3.1).

The method used to derive (3.2) is similar as that used for D_3. For any $i \in \{1, ..., n\}$,

$$\begin{aligned} &\|\nabla f(x_i) - \nabla_y G(x_i, \theta_k)\| \\ \le\ &\|\nabla f(x_i) + \frac{x_{i+1} - x_i}{\Delta t}\| + \| - \frac{y_k(t_{i+1}) - y_k(t_i)}{\Delta t} - \nabla_y G(y_k(t_i), \theta_k)\| \\ &\|\nabla G(y_k(t_i)) - \nabla G(x_i, \theta_k)\| + \|\frac{y_k(t_{i+1}) - y_k(t_i)}{\Delta t} - \frac{x_{i+1} - x_i}{\Delta t}\|. \end{aligned} \tag{A.16}$$

Using (A.13), (A.14) and

$$\|\nabla G(y_k(t_i)) - \nabla G(x_i, \theta_k)\| \le L_{G_y} e_k(t_i) \le L_{G_y} \sqrt{(\theta_k)},$$

we obtain

$$\begin{aligned} \|\nabla f(x_i) - \nabla_y G(x_i, \theta_k)\| &\le L_{G_y} \sqrt{J(\theta_k)} + \frac{L_f + L_{G_y}}{2} \Delta t + \frac{2\sqrt{J(\theta_k)}}{\Delta t}, \\ &\le \frac{5}{2} \frac{\sqrt{J(\theta_k)}}{\Delta t} + \frac{L_f + L_{G_y}}{2} \Delta t, \\ &\le C_2 \left(\frac{\sqrt{J(\theta_k)}}{\Delta t} + \Delta t \right) \end{aligned}$$

where

$$C_2 = \max \left\{ \frac{5}{2}, \frac{L_f + L_{G_y}}{2} \right\}.$$

This further implies (3.2).