LR-RaNN: Lipschitz Regularized Randomized Neural Networks for System Identification

Chunyang Liao
Department of Mathematics
University of California, Los Angeles
Los Angeles, CA 90095
liaochunyang@math.ucla.edu

Abstract

Approximating the governing equations from data is of great importance in studying the dynamical systems. In this paper, we propose randomized neural networks (RaNN) to investigate the problem of approximating the governing equations of the system of ordinary differential equations. In contrast with other neural networks based methods, training randomized neural network solves a least-squares problem, which significant reduces the computational complexity. Moreover, we introduce a regularization term to the loss function, which improves the generalization ability. We provide an estimation of Lipschitz constant for our proposed model and analyze its generalization error. Our empirical experiments on synthetic datasets demonstrate that our proposed method achieves good generalization performance and enjoys easy implementation.

1 Introduction

Dynamical system is widely used to analyze and understand how systems evolve with time, which are ubiquitous in various scientific areas such as physics [14], biology [9], chemistry [25], and economics [6]. Traditionally, dynamical systems are derived from first principles, which can guarantee conservation laws and are easy to interpret. However, physics-based approaches required strong modeling assumptions, and domain knowledge. For many real-world systems of interest, the dynamical systems are either unknown or can only be evaluated to high accuracy at significant computational expense. Recently data-driven methods for dynamical systems discovery have gained great interests [7, 17, 1]. System identification refers to the general process of building mathematical models and approximating dynamical systems from measured input-output data. In particular, data-driven system identification uses machine learning techniques to derive models from input-output data, bypassing prior domain and system knowledge.

There have been several recent methods using sparse regression [24, 2, 23], Gaussian process [18, 22, 5], and neural networks [15, 10, 16, 28, 27] for data-driven system identification. With a user-determined dictionary of candidate terms, sparse regression approaches select the most important terms using a sparsity-promoting regression, such as Least Absolute Shrinkage and Selection Operator (LASSO). Gaussian process regression is based on imposing a Gaussian prior on the unknown coefficients of the differential equation and statistically conditioning them on the observed data. The unknown coefficients are inferred via maximum likelihood estimation. Neural Network approaches use a neural network to learn the relationship between the input states and output states of a system. The neural network is trained on a set of input-output pairs, which typically results in solving a non-convex optimization problem due to the structure of neural networks. The theoretical guarantee relies on the result that neural networks are universal approximators. In [16], the authors introduced a Lipschitz regularization term to the loss function, which improved the generalization

Proceedings of the 1st Conference on Topology, Algebra, and Geometry in Data Science(TAG-DS 2025).

and recovery of governing equation. In [27], the authors proposed a system identification method using Lipschitz neural networks directly.

Despite remarkable empirical achievements, the training of neural networks requires significant computational resources. Its training involves solving a non-convex optimization problem. Due to the non-convexity nature of the optimization problem, training neural networks suffers from saddle points and local minima resulting in poor convergence behavior and numerical approximation accuracy. In addition, stochastic gradient descent (SGD) algorithm and its variants are typically used to solve the non-convex optimization problems, which iterates large datasets multiple times to update the high-dimensional space of trainable parameters. The gradient is usually approximated by numerical algorithms such as auto-differentiation, which increases the computational burden.

Implementing randomized Neural Networks (RaNNs) has became to an effective strategy for reducing computational time and minimizing optimization errors [11, 12, 3, 4]. Unlike the vanilla fully connected neural networks, the weights linked hidden layers in RaNNs are randomly sampled from certain distributions and the fixed. Only the parameters connecting the last hidden layer and the output layer are trainable parameters, which can be trained by solving a least-squares problem.

In this work, we investigate the problem of approximating the governing equations using the randomized neural networks. Specifically, our contributions are summarized as follows:

1. We approximate the governing equations using the randomized neural networks. Specifically, we introduce a Lipschitz regularization term to the loss function, which improves the generalization performance. Compared with the state-of-the-art Lipschitz regularized neural network [16] and Lipschitz neural network [27], our proposed method is easy to implement and to estimate the Lipschitz constant.
2. We provide theoretical guarantees on the generalization performance of our proposed randomized neural network-based method from a deterministic perspective. Specifically, we provide a worst-case error analysis over a compact domain and show how the number of hidden neurons, the regularization parameter, and the Lipschitz constant affect the generalization error.
3. We empirically verify the performance by conducting experiments on three synthetic datasets. The experimental results demonstrate that our proposed method is easy to implement, has comparable or even better generalization performance compared with state-of-the-art benchmarks, and significantly reduces the computational resources.

2 Preliminaries, Notations, and Problem Setup

In this paper, we use $x(t) \in \mathbb{R}^d$ to denote the state vector of a d-dimensional dynamical system at time t, $\dot{x}(t)$ denote the first order time derivative of $x(t)$. Let $f : \mathbb{R}^{1+d} \to \mathbb{R}^d$ be a vector-valued function. The system of ODE governed by f is defined as

$$\dot{x}(t) = f(t, x(t)). \tag{1}$$

A function f is said to be L-Lipschitz if $\|f(x)-f(y)\| \leq L\|x-y\|$ for all x, y. The goal of this paper is to approximate the unknown governing equation f from trajectory data using randomized neural networks. In practice, we are able to observe state vectors $x(t)$ at discrete time steps for differential initial conditions. Specifically, given uniformly-spaced time steps $t_1, \ldots, t_M$ and initial conditions $x_1(0), \ldots, x_K(0) \in \mathbb{R}^d$, we define $x_i(t_j) \in \mathbb{R}^d$ be an observation of state vector $x(t)$ at time t_j with initial condition $x_i(0)$ for all $i \in [K]$ and $j \in [M]$. Then we define the input data $\mathbf{x}_h \in \mathbb{R}^{1+d}$ and the targets $\mathbf{y}_h \in \mathbb{R}^d$ that are used to train randomized neural networks. For $h = j + (i-1)M$, we define $\mathbf{x}_h$ and $\mathbf{y}_h$ as follows:

$$\mathbf{x}_h = \begin{bmatrix} t_j, & - & x_i(t_j)^\top & - \end{bmatrix}^\top \in \mathbb{R}^{1+d}, \qquad \mathbf{y}_h = \dot{x_i}(t_j) \in \mathbb{R}^d. \tag{2}$$

In this paper, we consider the shallow (one-hidden layer) randomized neural network $\hat{f} : \mathbb{R}^d \to \mathbb{R}$, which takes the following form

$$\hat{f}(x) = \sum_{k=1}^{N} c_k \phi(\langle \boldsymbol{\omega}_k, x \rangle + b_k). \tag{3}$$

Parameter N denotes the number of neurons at the single hidden layer. Weight vectors $\boldsymbol{\omega}_k \in \mathbb{R}^d$ and bias term $b_k \in \mathbb{R}$ are sampled from certain distributions and then fixed. Function $\phi : \mathbb{R} \to \mathbb{R}$ is a non-linear activation function. Throughout this paper, we assume that the activation function ϕ is L-Lipschitz. Then we can derive that the shallow randomized neural network taking the form (3) is also Lipschitz continuous with Lipschitz constant $L \sum_{k=1}^{N} |c_k| \|\boldsymbol{\omega}_k\|$. Coefficients $c_k \in \mathbb{R}$ are trainable parameters. Training the randomized neural networks is equivalent to find best coefficients c_k, which can be trained by solving a least-squares problem.

There are various types of randomized neural networks such as random vector functional link (RVFL) networks [13], extreme learning machine (ELM) [8], and random features [20]. While all of them share the same structures and are widely used in practice, there are several major differences in terms of the motivations and the distributions of random weights. Random features were originally proposed to approximate large-scale kernel machines, and hence the distributions of random weights depend on the kernel function, see [20] for a list of kernels and their corresponding distributions. RVFL networks and ELMs were proposed to reduce the computational costs when training neural networks and the random weights were usually sampled from uniform distributions.

3 Theoretical Analysis

We aim to construct a randomized neural network to approximate the governing equation f from inputs-outputs pair $(\mathbf{x}_h, \mathbf{y}_h)$ defined in (2). We first notice that the target $\mathbf{y}_h \in \mathbb{R}^d$ has d-components and we may assume that each component $\mathbf{y}_h^j$ for all $j \in [d]$ can be approximated by a randomized neural network (3) separately. For ease of notation, in the rest of paper we omit the index j and use $y_h \in \mathbb{R}$ instead. We consider the following loss function

$$L(\mathbf{c}) = \frac{1}{KM} \sum_{h=1}^{KM} \left(y_h - \hat{f}(\mathbf{x}_h) \right)^2 + \lambda \sum_{k=1}^{N} |c_k|^2 \|\boldsymbol{\omega}_k\|^2. \tag{4}$$

The first term is the mean-squared error on the input-output pairs $(\mathbf{x}_h, y_h)$. The second term corresponds to the Lipschitz constant of $\hat{f}$. We slightly modify the Lipschitz constant by taking square of $c_k \|\boldsymbol{\omega}_k\|$. Unlike the ridge regularization, our regularization term include weights on the coefficients, which is indeed a weighted ridge regularization. Let $\mathbf{A} \in \mathbb{R}^{KM \times N}$ be the random matrix defined entry-wise by $\mathbf{A}_{j,k} = \phi(\boldsymbol{\omega}_k, \mathbf{x}_j)$ for all $j \in [KM]$ and $k \in [N]$ and $\mathbf{W} \in \mathbb{R}^{N \times N}$ be a diagonal matrix with entries $\mathbf{W}_{j,j} = \|\boldsymbol{\omega}_j\|$. The solution $\hat{\mathbf{c}} \in \mathbb{R}^N$ has closed form

$$\hat{\mathbf{c}} = (\mathbf{A}^\top \mathbf{A} + \lambda KM \mathbf{W}^2)^{-1} \mathbf{A}^\top \mathbf{y}, \tag{5}$$

where $\mathbf{y} = [y_1, \ldots, y_{KM}] \in \mathbb{R}^{KM}$.

3.1 Estimation on the Lipschitz Constant

We first estimate the Lipschitz constant of the trained randomized neural network, which quantifies the worst-case robustness against the small, adversarial perturbations of the input. In the context of system identification, the input $\mathbf{x}_h$ contains the observed state vector $x_i(t_j)$ with initial condition $x_i(0)$ at time t_j, which is usually polluted by observational noise. Therefore, controlling the Lipschitz constant is crucial to improve the model robustness.

Proposition 1. Assume that the observation vector $\mathbf{y}$ has unit norm, i.e. $\|\mathbf{y}\|_2 = 1$. The Lipschitz constant of the trained randomized neural network (3) with coefficients (5) is upper bounded by

$$\frac{\sqrt{N} \max_k \|\boldsymbol{\omega}_k\|}{\sqrt{\lambda KM} \min_k \|\boldsymbol{\omega}_k\|}.$$

Remark 2. Our theory indicates that the Lipschitz constant decreases as we increase the regularization parameter λ, which emphasizes the importance and necessity of regularization term. Our empirical results also verify the decay of Lipschitz constant, see Table 3.

Remark 3. The ratio $\frac{\max_{k \in N} \|\boldsymbol{\omega}_k\|}{\min_{k \in N} \|\boldsymbol{\omega}_k\|}$ determines the Lipschitz constant. In the asymptotic setting where we set $d \to \infty$, we notice that the ratio decreases showing that our model is robust in the high-dimensional setting. In addition, we numerically verify that the ratio does not diverge as $N \to \infty$, which shows that our model is also robust in the over-parametrization regime. We numerically verify the asymptotic behaviors of the ratio in the Appendix.

3.2 Generalization Error

In this section, we provide a generalization error analysis for the trained regularized shallow randomized neural network. We first define the following function space

$$\mathcal{F}(\rho) := \left\{ f(\mathbf{x}) = \int_{\mathbb{R}^d} \alpha(\boldsymbol{\omega}, b)\phi(\langle \boldsymbol{\omega}, \mathbf{x}\rangle + b) d\rho(\boldsymbol{\omega}, b) : \|f\|_\rho^2 = \mathbb{E}_{\boldsymbol{\omega},b}[\alpha(\boldsymbol{\omega}, b)^2] < \infty \right\}, \quad (6)$$

where $\rho(\boldsymbol{\omega}, b)$ is a joint probability distribution of $(\boldsymbol{\omega}, b) \in \mathbb{R}^d \times \mathbb{R}$. Notice that the completion of $\mathcal{F}(\rho)$ is a Hilbert space equipped with norm $\|f\|_\rho$. Indeed, the function space $\mathcal{F}(\rho)$ is a reproducing kernel Hilbert space. We refer readers to Proposition 4.1 in [21] for a rigorous proof.

In [16], the authors assumed that the test samples are independent and identically distributed from the true data distribution and applied Hoeffding's inequality to give a bound on the difference between the actual MSE and the empirical one. In the following theorem, we evaluate the generalization performance from a deterministic perspective.

Theorem 4. Let f be the true right-hand side belonging to function space $\mathcal{F}(\rho)$ and $\hat{f}$ be the trained shallow randomized neural network taking the form (3). Let D be a compact domain. Then for any $\delta \in (0, 1)$, it holds with probability at least $1 - \delta$ that

$$\sup_{x\in D} |f(x) - \hat{f}(x)| \leq C \left(\frac{1}{\sqrt{N}} + \frac{1}{\sqrt{\lambda} \min_k \|\boldsymbol{\omega}_k\|} + \frac{\max_k \|\boldsymbol{\omega}_k\|}{\min_k \|\boldsymbol{\omega}_k\|} \right),$$

where $C = 12\|f\|_\rho \log(2/\delta)$.

We consider a general compact domain D in the theorem. In the system identification problems, compact domain $D \in \mathbb{R}^{d+1}$ contains the trajectory data $(t_j, x_i(t_j))$.

Sketch of Proof. We split the error into two terms, i.e. $|f(x) - \hat{f}(x)| \leq |f(x) - f^*(x)| + |f^*(x) - \hat{f}(x)|$, where $f^*(x)$ is the "best" randomized neural network for approximating f. We bound each term separately. The first term is the approximation error. We apply the concentration inequalities in high-dimensional probability to obtain an upper bound $\frac{C}{\sqrt{N}}$. The second term is the estimation error, which measures the difference between the "best" randomized neural network and the trained randomized neural network. Adding the upper bound for each term together leads to the desired result. □

4 Numerical Experiments

In this section, we present several numerical experiments on the recovery of ordinary differential equations. We consider three examples: one dimensional autonomous ODE, one dimensional non-autonomous ODE, and Van der Pol Oscillator. We adopt the test mean-squared error (MSE), generalization gap, and the recovery error on full domain (non-trajectory data) as metrics to test the generalization performance. Let D_{test} be a set of test data with size $|D_{test}|$, the test MSE is defined as

$$\text{Test_MSE}(\hat{f}) = \frac{1}{|D_{test}|} \sum_{(X_h, Y_h)\in D_{test}} \left(Y_h - \hat{f}(X_h) \right)^2.$$

The generalization gap refers to the difference of training MSE and test MSE, which measures model generalization property, i.e

$$\text{Generalization Gap}(\hat{f}) = \text{Train_MSE}(\hat{f}) - \text{Test_MSE}(\hat{f}).$$

The recovery error on full domain compares the approximation by our networks with the true governing equation of the ODE system on the domain of interest. Specifically, let $\{(t_i, x_i)\}_{i\in[M]}$ be a set of points in the domain, the recovery error is defined as

$$\frac{1}{M} \sum_{i=1}^{M} \frac{|f(t_i, x_i) - \hat{f}(t_i, x_i)|}{\max_i f(t_i, x_i) - \min_i f(t_i, x_i)}. \quad (7)$$

We compare our proposed regularized randomized neural networks with Lipschitz regularized neural network and Lipschitz neural network. We perform the empirical experiments in the noiseless and noisy regimes. All experiments are performed in a MacBook with 2.3GHz 8-core Intel Core i9 processor.

4.1 Data Descriptions

In this section, we describe how we generate the trajectory data and non-trajectory data for each example. We generate the synthetic datasets using the function odeint from the scipy package in Python.

Autonomous ODE. We consider the following 1D Autonomous ODE:

$$\dot{x}(t) = \cos(5x) + x^3 - x,$$

where the right-hand side only depends on the space variable x. We compute the approximated solutions of ODE for time steps t in the interval $[0, 1]$ with $\Delta t = 0.04$ and for $K = 500$ initial conditions that are uniformly sampled from interval $[-0.7, 0.9]$. Our setting results in 12500 data points. We use the uniform grid over domain $[0, 1] \times [-0.7, 0.9]$ of resolution 100×100. to compute the recovery error on full domain.

Non-Autonomous ODE. Our second example is a 1D non-autonomous ODE, where the right-hand side depends on both space variable x and time t, i.e.

$$\dot{x}(t) = e^{-x}\log(t) - t^2.$$

We generate $K = 200$ trajectories whose initial conditions are uniformly sampled in the interval $[0.5, 5]$ and the solution for equispaced time steps t in the interval [0.1,2] with $\Delta t = 0.02$. Here we have 19000 data points. We use uniform grids over domain $[0.1, 2] \times [0.5, 5]$ of resolution 100×100 to compute the recovery error on full domain.

Van der Pol Oscillator. The Van der Pol oscillator is

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = \mu(1 - x_1^2)x_2 - x_1, \tag{8}$$

where $\mu > 0$. In this example, we set up $\mu = 0.02$. We generate $K = 400$ trajectories of 5 seconds [1]. The initial conditions are sampled from domain $[-4, 4] \times [-4, 4]$ and we approximate the solutions of ODE system for time steps t in interval $[0.15, 5.14]$ with $\Delta t = 0.01$. Our problem setting leads to 200000 data points. In this example, we generate uniform grids over domain $[0.15, 5.14] \times [-4, 4] \times [-4, 4]$ of resolution $20 \times 20 \times 20$ as non-trajectory data.

To generate noisy data, we follow the procedures in [16] and [27]. Specifically, for Autonomous ODE and Non-autonomous ODE, we compute mean range M_k across trajectories as

$$M_k = \frac{1}{K}\sum_{i=1}^{K}\left(\max_{j\in[M]} x_i^k(t_j) - \min_{j\in[M]} x_i^k(t_j)\right)$$

Then, the 1% noisy regime is given by

$$\hat{x}_i^k(t_j) = x_i^k(t_j) + n_{ij}M_k,$$

where n_{ij} follows a normal distribution $\mathcal{N}(0, 0.01)$ with mean 0 and variance 0.01. We produce 5% noisy regime in a similar way. For Van der Pol Oscillator, we produce 1% noisy version by

$$\hat{x}_1 = x_1 + e_1 \text{ and } \hat{x}_2 = x_2 + e_2,$$

where e_1 and e_2 are independent and sampled from a normal distribution $\mathcal{N}(0, 0.01)$ with mean 0 and variance 0.01. In a similar way we add 5% noise to the data.

In all examples, we use 80% of trajectory data for training and the remaining 20% samples for testing. We visualize the training and test samples for all examples in the Appendix.

4.2 Comparison

In this section, we compare our proposed method with two neural network-based methods. In [16], the authors used neural networks and added a Lipschitz regularization term in the loss function. In [27], the authors directly used Lipschitz neural networks, which are a class of neural networks with a prescribed upper bound of the Lipschitz constant [26]. In all experiments, we use a shallow randomized neural network with $N = 50$ neurons, and hence our model has only 50 trainable parameters.

[1] The data samples are available from here.

For neural networks with Lipschitz regularization term (LRNN), we consider neural networks with 8 hidden layers of widths $[10, 30, 30, 30, 30, 30, 30, 30]$ with ReLU activation function, which results in 5041 trainable parameters. For Lipschitz neural networks (LNN), we consider neural networks with 8 hidden layers of width 64 on each layer with ReLU activation function. The total number of trainable parameters are 54346 [2]. The training of our proposed regularized randomized neural networks is done by solving a regularized least-squares problem and we use numpy.linalg.solve method. The training of neural networks uses Adam with learning rate 0.01 and 15 epochs.

	Metric	Model	No Noise	1% Noise	5% Noise
Autonomous	Test MSE	our method	$\mathbf{7.33 \times 10^{-6}}$	$\mathbf{9.35 \times 10^{-3}}$	4.35×10^{-2}
		LRNN	2.05×10^{-3}	9.66×10^{-3}	3.63×10^{-2}
		LNN	9.65×10^{-5}	1.03×10^{-2}	$\mathbf{3.30 \times 10^{-2}}$
	Recovery Error on Full Domain	our method	$\mathbf{1.09}\%$	$\mathbf{3.88}\%$	$\mathbf{6.64}\%$
		LRNN	2.67%	4.21%	7.58%
		LNN	1.19%	4.62%	7.21%
Non-autonomous	Test MSE	our method	$\mathbf{3.95 \times 10^{-5}}$	$\mathbf{7.07 \times 10^{-3}}$	$\mathbf{2.09 \times 10^{-2}}$
		LRNN	7.95×10^{-4}	7.42×10^{-3}	2.53×10^{-2}
		LNN	9.22×10^{-3}	9.58×10^{-3}	2.38×10^{-2}
	Recovery Error on Full Domain	our method	0.67%	$\mathbf{0.86}\%$	$\mathbf{0.82}\%$
		LRNN	$\mathbf{0.64}\%$	1.07%	1.82%
		LNN	2.03%	1.28%	1.30%

Table 1: Summary of Autonomous and Non-autonomous ODEs: we report test MSEs and recovery errors on full domain of our proposed method, Lipschitz-regularized neural networks, and Lipschitz neural networks.

Metric	Model	No Noise	1% Noise	5% Noise
Test MSE	our method	2.69×10^{-3}	2.31×10^{-2}	9.76×10^{-2}
	LRNN	$\mathbf{2.38 \times 10^{-3}}$	$\mathbf{2.15 \times 10^{-2}}$	$\mathbf{5.77 \times 10^{-2}}$
	LNN	5.37×10^{-3}	4.15×10^{-2}	7.32×10^{-2}
Recovery Error on Full Domain (First Component)	our method	$\mathbf{0.21}\%$	$\mathbf{0.72}\%$	1.93%
	LRNN	1.19%	0.89%	$\mathbf{1.81}\%$
	LNN	1.95%	2.58%	2.37%
Recovery Error on Full Domain (Second Component)	our method	$\mathbf{0.17}\%$	$\mathbf{0.57}\%$	1.41%
	LRNN	0.90%	1.51%	$\mathbf{1.17}\%$
	LNN	1.03%	1.84%	1.71%

Table 2: Summary of Van der Pol Oscillator: we report test MSEs and recovery errors on full domain of our proposed method, Lipschitz-regularized neural networks, and Lipschitz neural networks.

In Tables 1 and 2, we report test MSE and recovery error on full domain for each model. Our numerical results suggest that our proposed method achieves similar or even beats Lipschitz regularized Neural Network and Lipschitz neural network over all examples. In particular, our proposed method is robust in the noisy regime. Across all experiments, we use $N = 50$ neurons for the randomized neural networks, which is much less than vanilla neural networks. Moreover, we solve a regularized least-squares problem to train the parameters, which is easy to solve and has convergence guarantee. In summary, our model has good generalization property and can significantly reduce the computational complexity.

4.3 Hyper-parameter Selection

In this section, we discuss the choices of hyper-parameters of our proposed model. In particular, the hyper-parameters are the number of hidden neurons N, the variance of Gaussian random weights σ^2, and the regularization parameter λ.

In the first study, we fix the number of hidden neurons $N = 50$ and the variance of Gaussian random weights $\sigma^2 = 1$. We aim to verify the effect of varying regularization parameter. Each

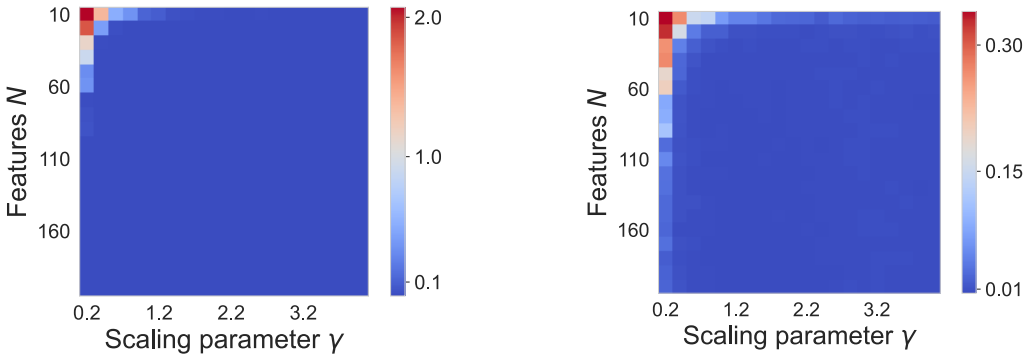
[2] We follow the network structures implemented in [16, 27]. The implementation of neural network with Lipschitz regularization can be found here. The implementation of Lipschitz neural networks can be found here.

number in Table 3 is the average of 50 runs over the random sampling of Gaussian random weights. We observe that the regularization term is helpful on the control of Lipschitz constant and results in better generalization performance. Our empirical results also verify that the Lipschitz constant decreases as we increase the regularization parameter λ. Numerical results of Non-autonomous ODE and Van der Pol Oscillator are presented in the Appendix E.2.

Regularization Parameter	Test Error	Generalization Gap	Recovery Error on Full Domain	Lipschitz Constant
0	8.98×10^{-4}	$\mathbf{-2.89 \times 10^{-5}}$	8.36%	20877813.9
0.0001	$\mathbf{4.91 \times 10^{-5}}$	-1.24×10^{-5}	**1.37%**	310.1
0.001	1.32×10^{-4}	-1.44×10^{-5}	1.68%	206.9
0.01	4.43×10^{-4}	-2.60×10^{-5}	2.64%	**106.6**

Table 3: Numerical results of Autonomous ODE: we report regularization parameters, test errors, generalization gaps, recovery errors on full domain, and Lipschitz constant of randomized neural networks in the noiseless regime.

In the second study, we fix the regularization parameter and demonstrate how the number of hidden neurons N and the variance of Gaussian random weights σ^2 affect the generalization results. In Figure 1, we show the heatmap of test errors and recovery errors of various number of features N and scaling parameter γ (variance of Gaussian weights $\sigma^2 = 1/\gamma^2$). Our results suggest that the generalization properties (test and recovery errors) are robust to the change of N and γ.

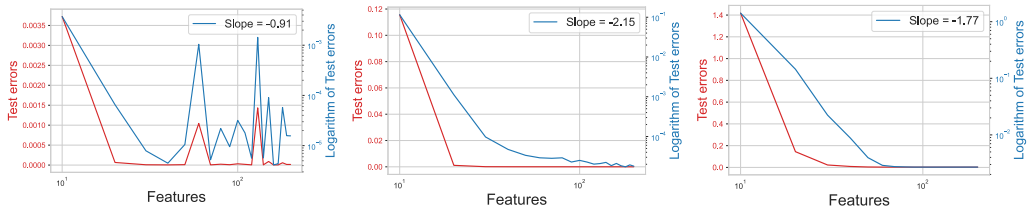


(a) Test error

(b) Recovery error on full domain

Figure 1: 1D Non-Autonomous ODE: we report test error and recovery error on full domain for various number of features N and scaling parameter γ (variance of Gaussian weights $\sigma^2 = 1/\gamma^2$).

In Figure 2, we also numerically verify the decay rate of the test errors as we increase the number of hidden neurons, which numerically verifies the upper bound in Theorem 4. We depict the test errors and logarithm of test errors versus the number of hidden neurons N.



(a) Autonomous ODE

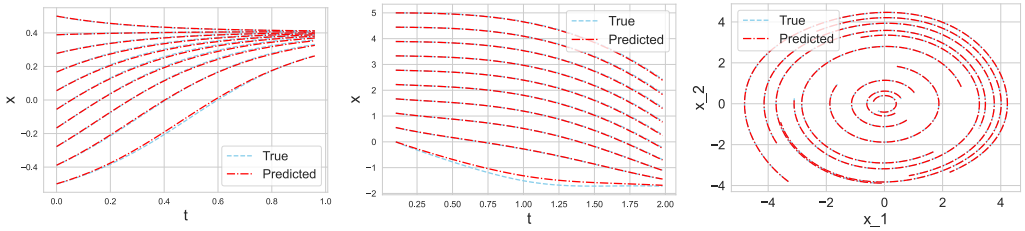
(b) Non-autonomous ODE

(c) Van der Pol Oscillator

Figure 2: Test MSE and Logarithm Test MSE as a function of the number of hidden neurons. Reported Slopes in the legends denote empirical decay rates.

4.4 Solution Estimation

We use the trained regularized randomized neural network as the right-hand side and solve for the ODE system. We numerically estimated the solution using the function odeint from the scipy package in Python. For the randomized neural network, we use $N = 50$ neurons at the single hidden layer, sample random weights from the normal distribution with mean 0 and variance $\sigma^2 = 1$, and set regularization parameter $\lambda = 10^{-5}$. In Figure 3, we show the true solution (blue line, solve using the true right-hand side f) and the predicted solution (red line, solve using the approximated right-hand side $\hat{f}$). We observe that the predicted solutions are accurate, even at the end of time interval, for all the selected initial conditions across all examples. In the Appendix E.3, we depict the predicted solutions for all examples in the noisy regimes. Our results indicate that our proposed model can provide accurate solution estimation in the noisy regimes.



(a) Autonomous ODE (b) Non-autonomous ODE (c) Van der Pol Oscillator

Figure 3: True and predicted solution of three ODE benchmarks.

5 Conclusion

In this paper, we study the problem of approximating the governing equation of system of ODEs by using the regularized randomized neural networks and introducing a Lipschitz regularization term to the loss function. We estimate the Lipschitz constant and derive a generalization error bound, which show the robustness and generalization ability of our proposed model. Our empirical studies show that our proposed model achieves similar or even better generalization performance compared with the state-of-the-art neural network-based methods. Moreover, our model has less trainable parameters and the training of our model can be done by solving a regularized least-squares problem, which significantly reduce the computational time and complexity. Due to the competitive generalization performance and advantages in computation, we believe that our proposed regularized randomized neural network-based method provides a good benchmark for system identification problem.

One of the limitations of our theoretical results is that our generalization error bound is in the worst-case scenario. The results could be generalized to the L^2 case. One can also derive an error bound in terms of the number of trajectory data and the number of time steps. We will leave those questions for future exploration. On the computational side, we only consider synthetic data because it is easy to test the generalization performance. We would like to apply our method to real-world data in the future.

References

[1] Jason J Bramburger. *Data-driven methods for dynamic systems*. SIAM, 2024.

[2] 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.

[3] Gianluca Fabiani. Random projection neural networks of best approximation: Convergence theory and practical applications. *SIAM Journal on Mathematics of Data Science*, 7(2):385–409, 2025.

[4] Gianluca Fabiani, Ioannis G Kevrekidis, Constantinos Siettos, and Athanasios N Yannacopoulos. Randonets: Shallow networks with random projections for learning linear and nonlinear operators. *Journal of Computational Physics*, 520:113433, 2025.

[5] Jinchao Feng, Charles Kulick, Yunxiang Ren, and Sui Tang. Learning particle swarming models from data with gaussian processes. *Mathematics of Computation*, 93(349):2391–2437, 2024.

[6] Giancarlo Gandolfo. *Economic dynamics: Methods and models*, volume 16. Elsevier, 1971.

[7] Amin Ghadami and Bogdan I Epureanu. Data-driven prediction in dynamical systems: recent developments. *Philosophical Transactions of the Royal Society A*, 380(2229):20210213, 2022.

[8] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489–501, 2006. Neural Networks.

[9] Trachette Jackson and Ami Radunskaya. *Applications of dynamical systems in biology and medicine*, volume 158. Springer, 2015.

[10] John G Kuschewski, Stefen Hui, and Stanislaw H Zak. Application of feedforward neural networks to dynamical system identification and control. *IEEE transactions on control systems technology*, 1(1):37–49, 1993.

[11] Chunyang Liao. Solving partial differential equations with random feature models. *Communications in Nonlinear Science and Numerical Simulation*, 152:109343, 2026.

[12] Chunyang Liao, Deanna Needell, and Hayden Schaeffer. Cauchy random features for operator learning in sobolev space. *arXiv: 2503.00300*, 2025.

[13] A.K. Malik, Ruobin Gao, M.A. Ganaie, M. Tanveer, and Ponnuthurai Nagaratnam Suganthan. Random vector functional link network: Recent developments, applications, and future directions. *Applied Soft Computing*, 143:110377, 2023.

[14] Paul Melby, Nicholas Weber, and Alfred Hübler. Dynamics of self-adjusting systems with noise. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 15(3), 2005.

[15] Kumpati S Narendra and Kannan Parthasarathy. Neural networks and dynamical systems. *International Journal of Approximate Reasoning*, 6(2):109–131, 1992.

[16] Elisa Negrini, Giovanna Citti, and Luca Capogna. System identification through lipschitz regularized deep neural networks. *Journal of Computational Physics*, 444:110549, 2021.

[17] Joshua S North, Christopher K Wikle, and Erin M Schliep. A review of data-driven discovery for dynamic systems. *International Statistical Review*, 91(3):464–492, 2023.

[18] Gianluigi Pillonetto, Francesco Dinuzzo, Tianshi Chen, Giuseppe De Nicolao, and Lennart Ljung. Kernel methods in system identification, machine learning and function estimation: A survey. *Automatica*, 50(3):657–682, 2014.

[19] I. F. Pinelis and A. I. Sakhanenko. Remarks on inequalities for large deviation probabilities. *Theory of Probability & Its Applications*, 30(1):143–148, 1986.

[20] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[21] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 555–561, 2008.

[22] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.

[23] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2197):20160446, 2017.

[24] Hayden Schaeffer, Russel Caflisch, Cory D. Hauck, and Stanley Osher. Sparse dynamics for partial differential equations. *Proceedings of the National Academy of Sciences*, 110(17):6634–6639, 2013.

[25] Joseph John Thomson. *Applications of dynamics to physics and chemistry*. Macmillan, 1888.

[26] Ruigang Wang and Ian Manchester. Direct parameterization of Lipschitz-bounded deep networks. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 36093–36110. PMLR, 23–29 Jul 2023.

[27] Shiqing Wei, Prashanth Krishnamurthy, and Farshad Khorrami. Data-efficient system identification via lipschitz neural networks. *arXiv:2410.21234*, 2024.

[28] Rose Yu and Rui Wang. Learning dynamical systems from data: An introduction to physics-guided deep learning. *Proceedings of the National Academy of Sciences*, 121(27):e2311808121, 2024.