Robust Neural Network Approach to System Identification in the High-Noise Regime

Elisa Negrini[1(✉)], Giovanna Citti[2], and Luca Capogna[3]

[1] Department of Mathematics, University of California Los Angeles, Los Angeles, USA
enegrini@ucla.edu
[2] Department of Mathematics, University of Bologna, Bologna, Italy
giovanna.citti@unibo.it
[3] Department of Mathematical Sciences, Smith College, Northampton, USA
lcapogna@smith.edu

Abstract. We present a new algorithm for learning unknown governing equations from trajectory data, using a family of neural networks. Given samples of solutions $x(t)$ to an unknown dynamical system $\dot{x}(t) = f(t, x(t))$, we approximate the function f using a family of neural networks. We express the equation in integral form and use Euler method to predict the solution at every successive time step using at each iteration a different neural network as a prior for f. This procedure yields M-1 time-independent networks, where M is the number of time steps at which $x(t)$ is observed. Finally, we obtain a single function $f(t, x(t))$ by neural network interpolation. Unlike our earlier work, where we numerically computed the derivatives of data, and used them as target in a Lipschitz regularized neural network to approximate f, our new method avoids numerical differentiations, which are unstable in presence of noise. We test the new algorithm on multiple examples in a high-noise setting. We empirically show that generalization and recovery of the governing equation improve by adding a Lipschitz regularization term in our loss function and that this method improves our previous one especially in the high-noise regime, when numerical differentiation provides low quality target data. Finally, we compare our results with other state of the art methods for system identification.

Keywords: Deep Learning · System Identification · Network Regularization

1 Introduction

System identification refers to the problem of building mathematical models and approximating governing equations using only observed data from the system. Governing laws and equations have traditionally been derived from expert

M. Sellmann and K. Tierney (Eds.): LION 2023, LNCS 14286, pp. 165–178, 2023.
https://doi.org/10.1007/978-3-031-44505-7_12

knowledge and first principles, however in recent years the large amount of data available resulted in a growing interest in data-driven models and approaches for automated dynamical systems discovery. The applications of system identification include any system where the inputs and outputs can be measured, such as industrial processes, control systems, economic data and financial systems, biology and the life sciences, medicine, social systems, and many more (see [3] for more examples of applications).

In this work we train a family of neural networks to learn from noisy data a nonlinear and potentially multi-variate mapping f, right-hand-side of the differential equation:

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

The trained network can then be used to predict the future system states.

In general, two main approaches can be used to approximate the function f with a neural network. The first approach aims at approximating the function f directly, like we did in our previous paper [11], which we refer to as *splines method*. In this work, inspired by the work of Oberman and Calder in [12], we use a Lipschitz regularized neural network to approximate the RHS of the ODE (1), directly from observations of the state vector $x(t)$. The target data for the network is made of discrete approximations of the velocity vector $\dot{x}(t)$, which act as a prior for f. One limitation of this approach is that, in order to obtain accurate approximations of the function f, one needs to obtain reliable target data, approximations of the velocity vector, from the observations of $x(t)$. This proved to be hard when a large amount of noise was present in the data. The second approach aims at approximating the function f implicitly by expressing the differential equation (1) in integral form and enforcing that the network that approximates f satisfies an appropriate update rule. This is the approach used in [18], which we refer to as *multistep method*, where the authors train the approximating network to satisfy a linear multistep method. An advantage of this approach over the previous one is that the target data used to train the multistep network is composed only of observations of the state vector $x(t)$. However, noise in the observations of $x(t)$ can still have a strong impact on the quality of the network approximation of f.

In this work we build on the second approach and introduce a new idea to overcome the limitations of the methods mentioned above. Similarly to the multistep method, we express the differential equation in integral form and train the network that approximates f to satisfy Euler update rule (with minimal modifications one can use multistep methods as well). This implicit approach overcomes the limitations of the splines method, whose results were strongly dependent on the quality of the velocity vector approximations used as target data. Differently than the multistep method, our proposed approach is based on a Lipschitz regularized family of neural networks and it is able to overcome the sensitivity to noise. Later on we compare these methods and other methods for system identification with our proposed approach and show that in the high-noise setting our method produces more accurate results thanks to the use of Lipschitz regularization and multiple networks.

The rest of the paper is organized as follows: Sect. 2 outlines relevant efforts in the field of system identification. In Sect. 3 we describe in detail our proposed method. Section 4 describes and discusses the experimental results. Finally conclusion and future research directions are outlined in Sect. 5

2 Related Works

In recent years many methods have been proposed for data-driven discovery of nonlinear differential equations. Commonly used approaches are sparse regression, Gaussian processes, applied Koopmanism and dictionary based approaches, among which neural networks. Sparse regression approaches are based on a user-determined library of candidate terms from which the most important ones are selected using sparse regression [4,20–22]. These methods provide interpretable results, but they are sensitive to noise and require the user to choose an "appropriate" sets of basis functions a priori. In contrast, since neural networks are universal approximators our method allows to accurately recover very general and complex RHS functions even when no information on the target function is available. Identification using Gaussian processes places a Gaussian prior on the unknown coefficients of the differential equation and infers them via maximum likelihood estimation [16,17,19]. The Koopman approach is based on the idea that non linear system identification in the state space is equivalent to linear identification of the Koopman operator in the infinite-dimensional space of observables. Since the Koopman operator is infinite-dimensional, in practice one computes a projection of the Koopman operator onto a finite-dimensional subspace of the observables. This approximation may result inaccurate in presence of noise and has proven challenging in practical applications [5,9,10]. In contrast our proposed method is able to overcome the sensitivity to noise thanks to the use of Lipschitz regularization and multiple networks. Since neural networks are universal approximators, they are a natural choice for nonlinear system identification: depending on the architecture and on the properties of the loss function, they can be used as sparse regression models, they can act as priors on unknown coefficients or completely determine an unknown differential operator [2,6,7,11,13–15,18]. Our method is part of this category, but adds to the existing literature thanks to the use of multiple networks and Lipschitz Regularization. Moreover, since our proposed method is based on weak notion of solution using integration it can be used to reconstruct both smooth and non-smooth RHS functions. This is especially an advantage over models that rely on the notion of classical solution like the splines method [11] and make it an extremely valuable approach when working with real-world data.

3 Proposed Method

In this section we describe the architecture used in the experiments.

Our goal is to approximate a vector-valued RHS $f(t, x)$ of a system of differential equations $\dot{x}(t) = f(t, x)$, directly from discrete noisy observations of the

state vector $x(t) \in \mathbb{R}^d$. We propose to do so using a neural network architecture composed of two blocks: the *target data generator* and the *interpolation network*. See Algorithm 1 for the full architecture algorithm.

The Target Data Generator: The target data generator is a family of neural networks whose goal is to produce reliable velocity vector approximations which will be used as target data for the interpolation network. The data is selected as follows: given time instants $t_1, \dots, t_M$ and K trajectories, define

$$x_i(t_j) \in \mathbb{R}^d, \quad i = 1, \dots, K, \quad j = 1, \dots, M$$

to be an observation of the state vector $x(t)$ at time t_j for trajectory i. For each time instant t_j, $j = 1, \dots, M-1$ we train a feed forward neural network $N_j(x(t_j))$ to approximate the velocity vector $\dot{x}(t)$ at time instant t_j. Indicating by θ^j the network parameters, the loss function L_j used for training forces each N_j to satisfy Euler update rule and it is defined as:

$$L_j(\theta^j) = \frac{1}{K} \sum_{i=1}^{K} \| \Delta t \; N_j(x_i(t_j), \theta^j) + x_i(t_j) - x_i(t_{j+1}) \|_2^2 \quad j = 1 \dots, M-1$$

Once the networks N_j are trained, they collectively provide a discrete approximation of the velocity vector on the full time domain, which we indicate by $\widetilde{\dot{x}(t)}$.

The Interpolation Network: The interpolation network N_{int} is a Lipschitz regularized neural network (as defined in [11]) which takes as input a time t and an observation of the state vector $x(t)$ and uses as target data the approximation of the velocity vector $\widetilde{\dot{x}(t)}$ given by the target data generator (this acts as a prior for the unknown function $f(t, x)$). Once trained the interpolation network N_{int} provides an approximation of the RHS function f on its domain, that is $N_{int}(t, x) \approx f(t, x)$. The loss function $L(\theta_{int})$ minimized to train the interpolation network contains two terms. The first one is the Mean Squared Error (MSE) between the network output and the target data: this forces the network predictions to be close to the observed data. The second term is a a Lipschitz regularization term which forces the Lipschitz constant of the network N_{int} to be small (the Lipschitz constant is a proxy for the size of the network's gradient):

$$L(\theta_{int}) = MSE\left(\widetilde{\dot{x}(t)}, N_{int}(t, x(t); \theta_{int})\right) + \alpha \mathrm{Lip}(N_{int}).$$

Here $\alpha > 0$ is a regularization parameter and $\mathrm{Lip}(N_{int})$ is the Lipschitz constant of the network N_{int}. Inspired by the work of Jin et al. in [8] the Lipschitz constant of network is computed as:

$$\mathrm{Lip}(N_{int}) \leq \|W_{int}^1\|_2 \|W_{int}^2\|_2 \dots \|W_{int}^L\|_2. \tag{2}$$

Since this is an explicit constraint on the network weights, the computational cost for this approximation is low. This is in contrast with the approximation we used in our previous paper [11], (Section 3.1) based on Rademacher's theorem, which requires to compute the gradient of the network at each iteration.

Algorithm 1. Full Architecture Algorithm

Require: $t = (t_j),\ x(t) = (x_i(t_j))\,,\ i = 1, \ldots, K,\ j = 1, \ldots, M$

Train Target data generator

for $j = 1 : M - 1$ **do**
$\Theta_j \leftarrow \text{argmin}_{\Theta_j}\, MSE(\Delta t\, N_j(x(t_j); \theta^j) + x(t_j),\, x(t_{j+1}))$
end for

Obtain $\widetilde{\dot{x}(t)} := (N_j(x(t_j))\,,\ j = 1, \ldots, M - 1$

Train Interpolation Network

$\Theta_{int} \leftarrow \text{argmin}_{\Theta_{int}} \left(MSE\left(\widetilde{\dot{x}(t)}, N_{int}(t, x(t); \theta_{int})\right) + \alpha \text{Lip}(N_{int}) \right)$

Obtain $N_{int}(t, x(t); \theta_{int}) \approx f(t, x(t))$

4 Experimental Results and Discussion

In this section we propose numerical examples of our method and comparisons with other methods for system identification. In the examples we use synthetic data with noise amount up to 10%. In this paper we only show one dimensional examples, but we explicitly notice that, since our method is applied component-wise, it can be used for data of any dimension. Three different metrics are used to evaluate the performance of the our method:

1. *Mean Squared Error (MSE)* on test data which measures the distance of the model prediction from the test data. We also report the *generalization gap* (difference between test and training error) obtained with and without Lipschitz regularization in the interpolation network. The smaller the generalization gap the better the network generalizes to unseen data (for a more precise description see [1]).
2. Since we use synthetic data, we have access to the true RHS function $f(t, x)$. This allows to compute the relative MSE between the true $f(t, x)$ and the approximation given by our architecture on arbitrary couples (t, x) in the domain of f. We call this error *recovery error*.
3. Since our method produces a function $N_{int}(t, x)$, it can be used as RHS of a differential equation $\dot{x} = N_{int}(t, x)$. We then solve this differential equation in Python and compute the relative MSE between the solution obtained when using as RHS the network approximation $N_{int}(t, x)$ and when using the true function $f(t, x)$. We call this *error in the solution*.

4.1 Smooth Right-Hand Side

The first example we propose is the recovery of the ODE

$$\dot{x} = xe^t + \sin(x)^2 - x \tag{3}$$

We generate solutions in Python for time steps t in the interval [0,0.8] with $\Delta t = 0.04$ and for 500 initial conditions uniformly sampled in the interval [-3,3]. The hyperparameters for our model are selected in each example by cross validation: the interpolation network N_{int} has $L = 8$ layers, each layer has 20 neurons, while each network N_j of the target data generator has $L_j = 3$ layers with 10 neurons each. The target data generator is made of 20 networks. In Table 1, we report the training MSE, testing MSE, Generalization Gap and estimated Lipschitz constant when 5% and 10% of noise is present in the data. Since our goal here is to compare the performance on test data of the networks with and without regularization, we select the number of epochs during training so as to achieve the same training MSE across all the regularization parameters choices and compare the corresponding Testing errors and Generalization Gaps. We report here only the results obtained for the non-regularized case and for the best regularized one when 5% and 10% of noise is present in the data. We can see from the tables that Lipschitz regularization improves the generalization gap by one order of magnitude for all amounts of noise, that a larger regularization parameter is needed when more noise is present in the data and that, as expected, adding Lipschitz regularization results in a smaller estimated Lipschitz constant. This confirms the findings from our previous paper that Lipschitz regularization improves generalization and avoids overfitting, especially in presence of noise.

Table 1. Test error and Generalization Gap comparison for 5% and 10% noise.

$\dot{x} = xe^t + \sin(x)^2 - x$, 5% *Noise*				
Regularization Parameter	*Training MSE*	*Testing MSE*	*Generalization Gap*	*Estimated Lipschitz Constant*
0	0.618%	0.652%	0.034%	7.09
0.004	**0.618%**	**0.619%**	**0.001%**	**6.33**
$\dot{x} = xe^t + \sin(x)^2 - x$, 10% *Noise*				
Regularization Parameter	*Training MSE*	*Testing MSE*	*Generalization Gap*	*Estimated Lipschitz Constant*
0	2.01%	2.32%	0.310%	7.72
0.015	**2.01%**	**2.03%**	**0.030%**	**6.38**

In Table 2 we report the error in the recovery for the RHS function $f(t, x) = xe^t + \sin(x)^2 - x$ and the error in the solution of the ODE when using the interpolation network as RHS. We can see that for all amounts of noise in the data, both the reconstruction error and the error in the solution are small, respectively they are less than 0.7% and 0.04%. For larger amounts of noise, the method still works, but provides less accurate approximations. For example with 20% noise the recovery error is approximately 3%. Making the method more robust in higher noise regimes will be object of a future work.

Table 2. Left: Relative MSE in the recovery of the RHS for up to 10% of noise. **Right:** Relative MSE in the solution of the ODE for up to 10% of noise

Relative MSE in the recovery of the RHS of $\dot{x} = xe^t + \sin(x)^2 - x$	
0% Noise	0.100%
5% Noise	0.144%
10% Noise	0.663%

Relative MSE in the solution of $\dot{x} = xe^t + \sin(x)^2 - x$	
0% Noise	0.016%
5% Noise	0.025%
10% Noise	0.038%

The left panel of Fig. 1 shows the true and reconstructed RHS and recovery error on the domain on which the original data was sampled for 5% of noise in the data. In the error plot a darker color represents a smaller error. We can see that the largest error is attained at the right boundary of the domain: by design of our architecture the target data generator only generates target data up to the second-last time step. As a consequence the interpolation network has only access to observations up to the second-last time step and so it is forced to predict the value of the RHS function at the last time step by extrapolation. It is then reasonable that the largest recovery error is attained at the right boundary of the domain. In the right panel of Fig. 1 we report the true solution (red line) and the solution predicted when using the interpolation network as RHS (dashed black line) for multiple initial condition and for 5% noise in the data. We notice that the prediction is accurate for all the initial conditions selected, but that it gets worse towards the end of the time interval because of the inaccurate approximation of the RHS at the right boundary of the time interval.

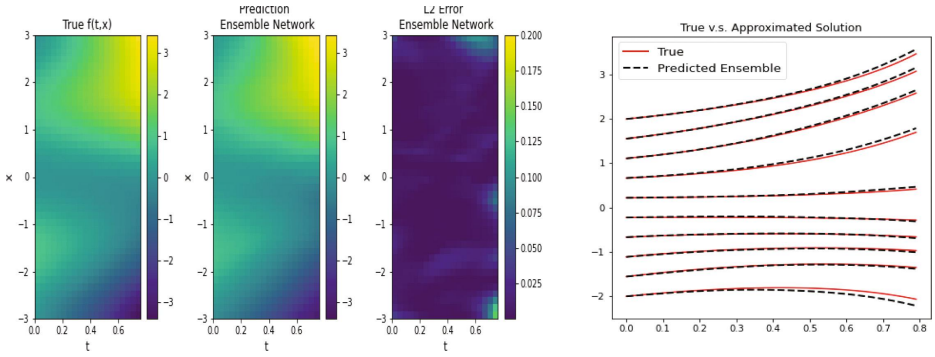

Fig. 1. Left: True RHS, Predicted RHS and recovery error for 5% noise in the data. **Right:** True and Predicted solution for 5% noise in the data

Finally, since the *test error*, the *error in the recovery* and the *error in the solution* are all measured using MSE, it makes sense to compare such homogeneous measurements. The first thing to notice is that the testing errors are larger than the recovery errors. This shows the ability of our network to avoid overfitting and produce reliable approximations of the true RHS even when large

amounts of noise are present in the data. In fact, the Test MSE is computed by comparing the value predicted by the network with the value of the corresponding *noisy* observation, while the recovery error is computed by comparing the value predicted by the network with the value of the *true* function f. The disparity between the test error and the recovery error then shows that the interpolation network provides results that successfully avoid fitting the noise in the data. The second thing to notice is the disparity between the recovery error and the error in the solution: the error in the solution is on average smaller than the recovery error. This is due to the data sampling: when recovering the RHS we reconstruct the function on the full domain, while the original data was only sampled on discrete trajectories; for this reason large errors are attained in the parts of the domain where no training data was available. On the other hand the error in the solution is computed on trajectories which were originally part of the training set, so it is reasonable to expect a smaller error in this case.

4.2 Non-smooth Right-Hand Side

We propose is the recovery of

$$\dot{x} = \text{sign}(t - 0.1) \tag{4}$$

and compare the results given by our proposed method and the splines method. Both methods aim at learning a Lipschitz approximation of the right-hand side function. The spline method is based on the notion of classical solution and it is doomed to fail in such a non-smooth setting. In contrast, our proposed method is based on weak notion of solution using integration and is able to accurately reconstruct even non-smooth functions. We generate data for time steps t in the interval [0,0.2] with $\Delta t = 0.02$ and for 500 initial conditions uniformly sampled in the interval $[-0.1, 0.1]$ for noise amounts up to 2%. We only use up to 2% of noise since, the splines model can only provide reliable target data for small noise amounts. The hyperparameters for the models in this example are as follows: each network N_j has $L_j = 3$ layers with 10 neurons each, the interpolation network and the network used in the splines method both have $L = 4$ layers, each layer has 30 neurons. The target data generator is made of 10 networks. As seen in Table 3, because of the low quality target data (approximation of the velocity vector from noisy observations of the positions) obtained by the splines method, this approach fails at reconstructing the non-smooth RHS, while our proposed method is able to produce an accurate reconstruction even in this case. The superior performance of our method over the spline method for this example can also be seen from Fig. 2. From left to right we represent the true, reconstructed RHS and the error in the reconstruction for the spline based method (top row) and for our method (bottom row) when 1% of noise is present in the data. We can see from the figure that the spline method in this case is not even able to find the general form of the RHS function correctly because of the bad quality of the target data. On the contrary, our proposed method, being completely data driven and based on a weak notion of solution, is able to reconstruct RHS functions like $\text{sign}(t - 0.1)$ that are non-smooth in t.

Table 3. Relative MSE in the recovery of the RHS for up to 2% of noise for our method and the splines method.

Relative MSE in the recovery of the RHS of $\dot{x} = \text{sign}(t - 0.1)$		
	Ours	*Splines*
1% Noise	0.002%	12.5%
2% Noise	0.004%	12.9%

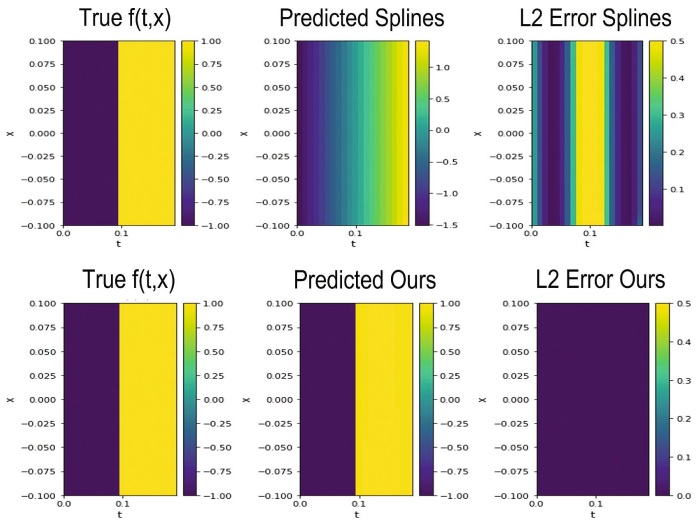

Fig. 2. Top row: Spline method. **Bottom row:** Our proposed method. From left to right: True RHS, Reconstructed RHS and Error in the reconstruction when 1% of noise is present in the data.

4.3 Comparison with Other Methods

We compare our method with the methods proposed in [18] and in [4]. For completeness we also provide a comparison with the splines method [11]. The method proposed in [18], (*multistep method*), is similar to ours: the authors place a neural network prior on the RHS function f, express the differential equation in integral form and use a multistep method to predict the solution at each successive time steps. In contrast with our method they do not use a family of networks and Lispchitz Regularization. The method proposed in [4], (*SINDy*), is based on a sparsity-promoting technique: sparse regression is used to determine, from a dictionary of basis functions, the terms in the dynamic governing equations which most accurately represent the data. Finally, we compare with the splines method described in Sect. 1. We report here the relative error obtained by the different methods in the approximation of the true f as well as the computational time for each method.

We generated the data by computing approximated solutions of

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

for time steps t in the interval [0,1] with $\Delta t = 0.04$ and for 500 initial conditions uniformly sampled in the interval $[-0.7, 0.9]$. The interpolation network has $L = 8$ layers, each layer has 30 neurons, while each network N_j has $L_j = 3$ layers with 20 neurons each. The target data generator is made of 25 networks. We compare the results obtained by our proposed method and the spline, multistep methods, a polynomial regression with degree 20 and SINDy. The dictionary of functions used for SINDy constraints polynomials up to degree 10 as well as other elementary functions: $\cos(x), \sin(x), \sin(3x)$, $\cos(3x)$, e^x, $\ln(x), \tan(x)$. In Table 4 we report the relative MSE in the recovery of the RHS function $f = cos(3x) + x^3 - x$ for up to 10% of noise. We notice that when no noise is present in the data, so that overfitting is not a concern, SINDy outperforms all the other methods. However, when noise is present in the data our method gives the best results. For example, when 5% noise is present in the data our method obtains an error of 0.096% which is smaller than the errors obtained by all the other methods by one order of magnitude or more. This shows that our proposed method is able to overcome the sensitivity to noise. In terms of computational time we can see that polynomial regression and SINDy are the fastest at performing the reconstruction with computational time lower than 1 s. This is expected since they have approximately 100 times less parameters than the neural network methods. The neural network methods have higher computational cost and our proposed method, while giving the most accurate results for noisy data, is the slowest. This is because it requires training of multiple networks, while the splines and multistep methods only require training one network. Note, however, that our method, while being slower than the other methods we compare with, provides the most accurate result in under 2 min.

Table 4. Relative MSE and computational time comparison in the recovery of the RHS for up to 10% of noise for our method, the splines and multistep methods, polynomial regression with degree 20, SINDy with custom library.

Relative MSE and computational time comparison for $\dot{x} = cos(3x) + x^3 - x$					
	Ours	*Splines*	*Multistep*	*Polynomial Regression degree 20*	*SINDy custom library*
0% Noise	0.0505%	0.214%	0.116%	6.3e-05%	**5.7e-05%**
5% Noise	**0.0957%**	0.585%	1.20%	3.33%	0.619%
10% Noise	**0.520%**	1.90%	3.51%	17.0%	3.36%
Time (s)	119.5	34.6	26.1	0.60	**0.54**

In Fig. 3 we report the true (red line) and recovered RHS function (blue line) when 5% of noise is present in the data. This figure confirms the findings shown in the previous table: our method is able to reconstruct the true RHS most accurately showing that our method is robust to noise. From the table above we notice that, for noisy data, the worst accuracy was always attained by the polynomial regression. In this case, even if a 20° polynomial has 100 times less parameters than our neural network, increasing the degree of the polynomial

increased the error in the recovery. From this figure we can clearly see why that happens: the polynomial regression with degree 20 is already overfitting the noisy data and the largest errors are attained at the boundaries of the domain where the polynomial is highly oscillatory. The other three methods are able to provide approximations that capture the general form of the true RHS function, but only our method is able to provide an accurate approximation even at the boundary of the domain.

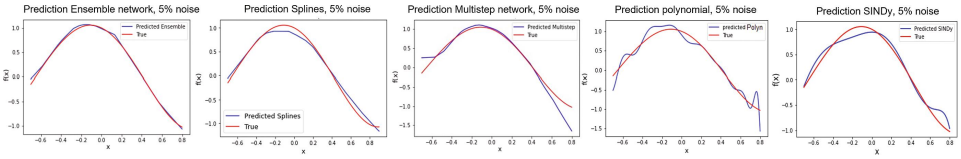

Fig. 3. From left to right, true and recovered RHS for 5% noise in the data obtained by our method, splines method, Multistep Method, Polynomial Regression with degree 20, SINDy with custom library.

4.4 Improving Interpretability Using SINDy

In this section we show how we can improve the interpretability of our method by combining it with SINDy. The strategy is as follows:

1. Given noisy $x(t)$ we use our neural network architecture to find a network N_{int} which approximates the unknown function f.
2. We solve the differential equation $\dot{x}(t) = N_{int}(t, x)$ for multiple initial conditions and obtain new solutions $\bar{x}(t)$. These solutions are a denoised version of the original observations since they were produced using the regularizing neural network architecture.
3. The denoised data $\bar{x}(t)$ is then given to SINDy to produce an interpretable and sparse representation of N_{int}.

We show the results of this strategy for the example proposed in Sect. 4.3. Recall that our goal is to approximate the equation

$$\dot{x} = cos(3x) + x^3 - x \tag{6}$$

In Sect. 4.3 we showed that our neural network architecture is able to reconstruct correctly the RHS and that, when noise it's present in the data, the recovered RHS function is more accurate than the one obtained by SINDy. In this section, we use the network N_{int} found in Sect. 4.3 when 5% of noise is present in the data and use it to produce denoised solutions $\bar{x}(t)$ as explained above. We then use SINDy with the same custom library of functions as in Sect. 4.3 to produce an interpretable and sparse approximation of the original f. When using this technique we obtain the following RHS approximation:

$$\dot{x}(t) \approx 0.898x^3 - 1.055\sin(x) + 0.996\cos(3x) \tag{7}$$

while applying SINDy directly to noisy data gave:

$$\dot{x}(t) \approx -9.431x^3 + 28.141x^5 + 2.730x^6 + -18.665x^7 - 4.902x^9 - 7.477x^{10} + 0.945\cos(3x) \quad (8)$$

We see that Eq. (7) is very close to the true Eq. (6). The main difference is that instead of the term "$-x$" SINDy found "$-1.055\sin(x)$". This is reasonable since for small values of x, like in this example, $x \approx \sin(x)$. On the contrary, Eq. (8) obtained by applying SINDy directly to the noisy data results in an approximated f containing high order terms: this is caused by the noise in the data. As a consequence the MSE in the reconstruction improves from 0.619% to 0.0096%. This can also be seen in the figure below (Fig. 4):

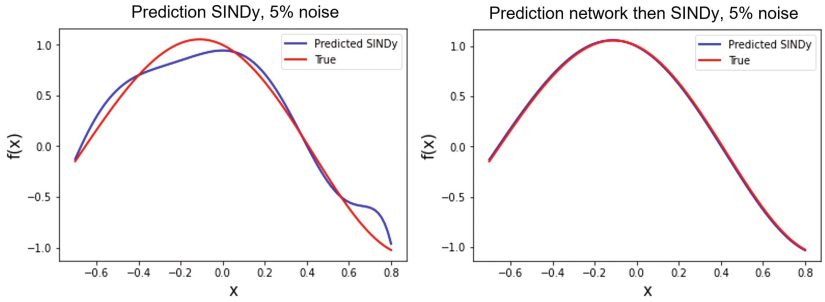

Fig. 4. Left: SINDy reconstruction from 5% noisy data. Right: SINDy reconstruction from denoised network data.

5 Conclusion

In this paper we use a Lipschitz regularized family of neural networks to learn governing equations from data. There are two main differences between our method and other neural network system identification methods in the literature. First, we add a Lipschitz regularization term to our loss function to force the Lipschitz constant of the network to be small. This regularization results in a smoother approximating function and better generalization properties when compared with non-regularized models, especially in presence of noise. Second, we use a family of neural networks instead of a single neural network for the reconstruction. We show that this makes out model robust to noise and able to provide better reconstruction than other state of the art methods for system identification. To our knowledge this is the first time that Lipschitz regularization is added to a family of networks to overcome the sensitivity to noise in a system identification problem. More in detail, our numerical examples, which are representative of a larger testing activity with several different types of right-hand sides $f(x,t)$, show multiple strengths of our method: when noise is present in the data, the Lipschitz regularization improves the generalization gap by one order of magnitude or more. Our architecture is robust to noise and is able to avoid overfitting even when large amounts of noise are present in the data (up

to 10%). This robustness to noise is especially an advantage over methods that do not use a family of networks such as [18]. The method is completely data-driven and it is based on weak notion of solution using integration. For this reason, it can be used to reconstruct even non-smooth RHS functions. This is especially an advantage over models that rely on the notion of classical solution like the Splines Method [11]. Since neural networks are universal approximators, we do not need any prior knowledge on the ODE system, in contrast with sparse regression approaches in which a library of candidate functions has to be defined. As shown in Sect. 4.3, direct comparison with polynomial regression and SINDy shows that our model is a better fit when learning from noisy data, even if it comes at the cost of increased computational time. Since our method is applied component-wise, it can be used to identify systems of any dimension, which makes it a valuable approach for high-dimensional real-world problems. As shown in Sect. 4.4, combining our method with SINDy produces a more accurate, intepretable and sparse reconstruction than using SINDy on the original noisy data, thanks to the denoising properties of our architecture.

Future research directions include applying our methods to real world data and extending our methods to the reconstruction of Partial Differential Equations (PDEs). More in detail, first we would like to use our method to reconstruct an equation that approximately describes the evolution in time of COVID-19 infected people. Another interesting application would be to reconstruct the Hodgkin-Huxley model from data. This is a ODE system that describes how action potentials in neurons are initiated and propagated. Second, we would like to generalize our models to the recovery of partial differential equations. Specifically, consider the parabolic PDE $u_t = f(t, x, u, \nabla u, D^2 u)$; given a finite number of observations of $u(t, x)$ the goal is to reconstruct the function f.

Acknowledgements. E. N. is supported by Simons Postdoctoral program at IPAM and DMS 1925919. L. C. is partially supported by NSF DMS 1955992 and Simons Collaboration Grant for Mathematicians 585688.

G. C. is partially supported by the EU Horizon 2020 project GHAIA, MCSA RISE project GA No 777822.

Results in this paper were obtained in part using a high-performance computing system acquired through NSF MRI grant DMS-1337943 to WPI.

References

1. Abu-Mostafa, Y.S., Magdon-Ismail, M., Lin, H.T.: Learning from Data, vol. 4. AMLBook, New York (2012)
2. Berg, J., Nyström, K.: Data-driven discovery of PDEs in complex datasets. J. Comput. Phys. **384**, 239–252 (2019)
3. Billings, S.A.: Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains. Wiley, Hoboken (2013)
4. Brunton, S.L., Proctor, J.L., Kutz, J.N.: Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc. Natl. Acad. Sci. **113**(15), 3932–3937 (2016)

5. Budišić, M., Mohr, R., Mezić, I.: Applied Koopmanism. Chaos: Interdisc. J. Nonlinear Sci. **22**(4), 047510 (2012)
6. Champion, K., Lusch, B., Kutz, J.N., Brunton, S.L.: Data-driven discovery of coordinates and governing equations. Proc. Natl. Acad. Sci. **116**(45), 22445–22451 (2019)
7. Chen, R.T., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: Advances in Neural Information Processing Systems, vol. 31 (2018)
8. Jin, P., Lu, L., Tang, Y., Karniadakis, G.E.: Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. Neural Netw. **130**, 85–99 (2020)
9. Lusch, B., Kutz, J.N., Brunton, S.L.: Deep learning for universal linear embeddings of nonlinear dynamics. Nat. Commun. **9**(1), 1–10 (2018)
10. Nathan Kutz, J., Proctor, J.L., Brunton, S.L.: Applied Koopman theory for partial differential equations and data-driven modeling of spatio-temporal systems. Complexity **2018** (2018)
11. Negrini, E., Citti, G., Capogna, L.: System identification through Lipschitz regularized deep neural networks. J. Comput. Phys. **444**, 110549 (2021). https://doi.org/10.1016/j.jcp.2021.110549. https://www.sciencedirect.com/science/article/pii/S0021999121004447
12. Oberman, A.M., Calder, J.: Lipschitz regularized deep neural networks converge and generalize. arXiv preprint arXiv:1808.09540 (2018)
13. Ogunmolu, O., Gu, X., Jiang, S., Gans, N.: Nonlinear systems identification using deep dynamic neural networks. arXiv preprint arXiv:1610.01439 (2016)
14. Qin, T., Wu, K., Xiu, D.: Data driven governing equations approximation using deep neural networks. J. Comput. Phys. **395**, 620–635 (2019)
15. Raissi, M.: Deep hidden physics models: deep learning of nonlinear partial differential equations. J. Mach. Learn. Res. **19**(1), 932–955 (2018)
16. Raissi, M., Karniadakis, G.E.: Hidden physics models: machine learning of nonlinear partial differential equations. J. Comput. Phys. **357**, 125–141 (2018)
17. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Machine learning of linear differential equations using Gaussian processes. J. Comput. Phys. **348**, 683–693 (2017)
18. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Multistep neural networks for data-driven discovery of nonlinear dynamical systems. arXiv preprint arXiv:1801.01236 (2018)
19. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. SIAM J. Sci. Comput. **40**(1), A172–A198 (2018)
20. Rudy, S.H., Brunton, S.L., Proctor, J.L., Kutz, J.N.: Data-driven discovery of partial differential equations. Sci. Adv. **3**(4), e1602614 (2017)
21. Schaeffer, H.: Learning partial differential equations via data discovery and sparse optimization. Proc. R. Soc. A: Math. Phys. Eng. Sci. **473**(2197), 20160446 (2017)
22. Schaeffer, H., Caflisch, R., Hauck, C.D., Osher, S.: Sparse dynamics for partial differential equations. Proc. Natl. Acad. Sci. **110**(17), 6634–6639 (2013)