Dissipative Neural Networks for Modeling Distributed Nonlinear Systems

Jona Schulz, Carlos Santos, Till Seefeldt

Abstract— **Ensuring dissipativity in distributed systems is crucial for achieving stability and robust performance. This paper addresses the challenge of enforcing dissipativity in a distributed system, specifically focusing on non-linear dynamics learned by a neural network. Our approach integrates the concept of dissipativity directly into the learning process to ensure that the resulting model accurately reflects the dissipative properties of the original system. This is achieved by incorporating the dissipativity inequality into the neural network's cost function, thereby guiding the learning algorithm to adhere to the dissipative nature of the system. We demonstrate that our method not only captures the complex dynamics of the system but also maintains the dissipative behavior. This work provides a further basis for the use of machine learning in the area of distributed systems control.**

I. INTRODUCTION

Distributed systems, which are governed by various dynamic models, are common in engineering problems. When used to model dynamics, such as leader-follower multiagent systems [1] or traffic flow [2], these systems demonstrate a wide array of non-linear, complex behaviors.

Dissipativity is a desirable characteristic of nonlinear dynamical systems, establishing important links to stability, reachability, and controllability, which are essential concepts in control theory [3]. It is an input-output property, which is based on the notion of energy storage within a system when deviating from an equilibrium. Returning to the equilibrium state minimizes the internal storage, if no additional energy is introduced to the system [4]. A system is considered dissipative if the change of energy stored in the system over time is less than or equal to the energy supplied to the system [4, 5].

Among the methods of identifying dynamic models, neural network learning based on input-output data has recently garnered attention [6, 7, 8]. Especially for non-linear systems, which have traditionally been modeled with linear approximations [9], neural networks provide a promising alternative. However, ensuring guarantees of the learned system behavior, such as stability, robustness, and adherence to physical constraints, is essential for their practical deployment [3, 6, 8]. Without these guarantees, neural networks may produce models that, while accurate during training, could behave unpredictably or violate essential system properties when applied in real-world scenarios. Among the essential properties a learned system must maintain, is the dissipative property of a system. Past work has investigated the enforcement of dissipativity of neural dynamical models [3, 8]. In the following, this paper will attempt to provide a method for encouraging dissipativity of distributed nonlinear system models based on neural networks.

We will obtain a storage function from the dynamics of a distributed system, proving its dissipativity. The dynamics of the system will be learned by a neural network with a novel training method for encouraging dissipative properties. The resulting systems' dissipativity will be compared and contrasted against a baseline model without dissipativity constraint.

II. MATHEMATICAL PRELIMINARIES

A. Neural Networks

A feed-forward, fully-connected neural network is a mapping $f_{NN} : \mathbb{R}^n \to \mathbb{R}^m$ consisting of a sequence of layers L_i, $i \in \{1, \dots, l\}$ given by

$$L_i : z_i = \phi(v_i) \tag{1}$$

where $v_i = W_i z_{i-1} + b_i$. $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function that is applied element-wise to the vector v_i. $z_0 = x$ is the input to the neural network, v_l is its output. Such a neural network mapping will be referred to as $f_{NN}(x; \theta)$ in the following, where θ is the parameter vector containing the concatenation of all flattened weight matrices W_i and biases b_i, $i \in \{1, \dots, l\}$.

B. Dissipativity

Dissipativity theory in control generalizes the notion of energy exchange within a system and provides a framework to study the stability of interconnections.

Formally, the general open system described in equation (2) is said to be dissipative with respect to a scalar-valued supply rate, $s(u, y)$, if there exists a non-negative storage function, $V(x) : \mathbb{R}^n \to \mathbb{R}$, such that inequality (3) is fulfilled at all times.

$$\dot{x} = f(x, u), \quad y = f(x, u) \tag{2}$$

$$s(u(t), y(t)) - \dot{V}(x) \geq 0 \tag{3}$$

The supply rate, $s(u, y)$, describes the energy supplied to the system as a function of its inputs u and outputs y while the storage function $V(x)$ represents the energy stored within the system. The dissipativity inequality (3) states that at any time point, the variation in energy stored in the system, should be smaller than or equal to the energy supplied to it.

Common in dissipativity theory are quadratic supply rates, as they can be expressed in the form of equation (4), with Q and R symmetric.

$$s(u,y) = \begin{pmatrix} u \\ y \end{pmatrix}^T \begin{pmatrix} Q & S \\ S^T & R \end{pmatrix} \begin{pmatrix} u \\ y \end{pmatrix} \tag{4}$$

Examples of QSR dissipativity are passive systems where the supply rate is given by

$$s(u,y) = u^T y \tag{5}$$

and systems with finite L_2 gain where the supply rate is given by

$$s(u,y) = \gamma^2 u^T u - y^T y \tag{6}$$

C. SOS Programming

Sum of Squares (SOS) programming is an optimization technique that leverages concepts from semidefinite programming (SDP) to construct polynomials as sums of squares of simpler polynomials [10].

Specifically, a SOS polynomial is one of the form presented in equation (7).

$$p(x) = \sum_i q_i(x)^2 \tag{7}$$

Such decompositions allow to prove non-negativity for polynomic expressions.

Further, SOS decomposition can be solved trough SDP programming when constrained to theorem (II.1) [10].

Theorem II.1. *A polynomial $p(x)$ is SOS if and only if there exists a positive semidefinite matrix Q, also refered to as Gram matrix, such that equation (8) is fulfilled.*

$$p(x) = z^T Q z \tag{8}$$

,where z is the vector of monomials in x_i variables.

III. Problem Formulation

In the following we outline how neural networks can be used to model nonlinear distributed dynamical systems.

A. Distributed Neural Dynamical Models

In this paper we consider distributed dynamical systems where each node i, $i \in \{1,\dots,N\}$ has the same dynamics given by

$$\begin{aligned} \dot{x}_i =& f(x_i) + u_i \\ y_i =& h(x_i) \end{aligned} \tag{9}$$

where $x_i \in \mathbb{R}^d$ is the nodal state and $u_i \in \mathbb{R}^d$ is the input to node i. Given an adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, the inputs u_i are given by

$$u_i = \sum_{j=1}^{N} \mathbf{A}_{ij} g(x_i, x_j) \tag{10}$$

where $g : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$. The vector $\mathbf{x} = (x_1^T, \dots, x_N^T)^T$ contains the states of all nodes of the system.

As proposed in [11], such distributed dynamical systems can be modeled using neural networks $f_{NN}(x_i;\theta_f)$ and $g_{NN}(x_i,x_j;\theta_g)$ which model the nodal and inter-nodal coupling dynamics respectively. The combined neural network model given by

$$\mathbf{F}_{NN}(\mathbf{x};\Theta)_i = f_{NN}(x_i;\theta_f) + \sum_{j=1}^{N} \mathbf{A}_{ij} g_{NN}(x_i,x_j;\theta_g) \tag{11}$$

where $i \in \{1,\dots,N\}$ and Θ is the parameter vector containing θ_f and θ_g [11].

The evolution of $\mathbf{x}$ over time is given by

$$\mathbf{x}(t_1) = \mathbf{x}(t_0) + \int_{t_0}^{t_1} \mathbf{F}(\mathbf{x};\Theta) dt \tag{12}$$

which can be solved using an ODE solver [11].

The neural network model is trained on a dataset of trajectories from a particular distributed dynamical system. where each sample contains the states $\mathbf{x}$ for a forecast window of N_f time steps at intervals Δ. The dataset is given by

$$\mathcal{D} = \{\mathbf{X}^{(k)}(t), \mathbf{X}^{(k)}(t+\Delta), \dots, \mathbf{X}^{(k)}(t+N_f\Delta)\} \tag{13}$$

where $\mathbf{X}^{(k)} = \{\mathbf{x}_1, \dots, \mathbf{x}_B\}$ denotes the k-th batch with batch size B [11].

The objective to be minimized during training is given by the mean squared error between ground-truth and predicted trajectories. By using a differentiable ODE solver like a Runge-Kutta method, backpropagation of errors through the ODE solver and the neural network model can be performed [11].

B. Ensuring Dissipativity

Even though neural networks can act as universal functions and correctly capture system dynamics, unconstrained training does not ensure the inheritance of underlying fundamental properties like dissipativity, as showcased in [8, 3].

To this end, we consider the training loss formulation depicted in equation (14) that combines the standard mean squared error loss with a dissipativity-encouraging loss.

$$\mathcal{L} = \mathcal{L}_{MSE} + \lambda \mathcal{L}_{Dissipativity} \tag{14}$$

In training the neural network model described above with this additional dissipativity objective, we aim at obtaining neural dynamical models which respect the dissipativity property of the nodes of the underlying distributed dynamical system.

IV. Results

In the following we propose a novel method for training dissipative neural network models for distributed dynamical systems with the use of a new dissipativity loss function. We will first outline how storage functions for polynomial systems can be obtained using SOS programming and then proceed to outline how a dissipativity loss function for neural network training is constructed from the system storage function and the dissipativity inequality.

A. Finding Storage Functions

Given a dynamical system (9), finding a storage function is necessary for validating the dissipativity of the system for a given supply rate. If the system dynamics are described by a polynomial in x and u, a storage function can be found using sum-of-squares (SOS) programming [10]. SOS programming is used to find polynomial coefficients such that the resulting polynomial can be expressed as a sum of squared polynomials ensuring non-negativity for all inputs. Choosing a polynomial degree d, the problem of finding a storage function reduces to finding polynomial coefficients such that the storage function $V(x)$ and $s(u,y) - \dot{V}(x)$ are sum of squares and therefore non-negative. For systems with finite L_2 gain this results in the following optimization problem:

$$\begin{aligned} &\min_{\gamma,\alpha} \gamma \\ \text{s.t. } &P^d(x;\alpha) \text{ is SOS} \\ &s(u,y) - \left(\frac{dP^d(x;\alpha)}{dx}\right)^T f(x,u) \text{ is SOS} \\ &\gamma \geq 0 \end{aligned} \tag{15}$$

where $P^d(x;\alpha) =: V(x)$ is a polynomial of degree d in x with the vector of coefficients α and $s(u,y)$ is the supply rate for finite L_2 gain defined in equation (6). [12] offers an implementation for solving such an SOS optimization problem.

B. Dissipativity Loss Function

Using the storage function obtained for the true system dynamics, we implemented a loss function that penalizes violations of the dissipativity inequality by the neural network dynamics. Specifically, for a given input state $\mathbf{x}(t)$ the dissipativity loss for the i-th node is given by

$$\mathcal{L}^{i,t}_{\text{diss}} = \max\left\{0, \left(\frac{dV(x_i(t))}{dx}\right)^T \mathbf{F}_{NN}(\mathbf{x}(t);\Theta)_i\right\} \tag{16}$$

For a batch of predicted trajectories, the dissipativity loss is computed as the average of $\mathcal{L}^{i,t}_{\text{diss}}$ over all nodes $i \in \{1,\ldots,N\}$, time points $t \in \{t_0, t_0+\Delta, \ldots, t_0+N_f\Delta\}$ and batch items.

V. Numerical Examples and Simulations

The methodology for training neural networks with dissipativity constraint outlined in the previous section was applied to two different distributed non-linear dynamical systems. For each system, a baseline model with no dissipativity loss was compared to models with varying dissipativity loss weights λ. The mean squared error (MSE) between ground-truth and predicted trajectories as well as the dissipativity loss outlined in (16) serve not only as loss functions during neural network training but also as suitable metrics to quantify the performance of the models at accurately modeling the dynamics and dissipative properties of the underlying system.

A. Dampened Nonlinear Oscillators with Linear Coupling

The first dynamical system considered consists of 11 nonlinear dampened oscillators interconnected through linear coupling. The system dynamics for the i-th node, $i \in \{1,\ldots,N\}$, are defined as

$$\begin{aligned} \dot{x}_{i1} &= x_{i2} \\ \dot{x}_{i2} &= -\alpha x_{i1}^3 - kx_{i2} + \beta u_i \\ u_i &= \sum_{j=1}^{N} \mathbf{A}_{ij}(x_{j2} - x_{i2}) \\ y_i &= x_{i2} \end{aligned} \tag{17}$$

where $\alpha = 0.1$, $\beta = 0.01$ and $k = 0.01$ were chosen for simulations. The interconnection of nodes is give by the adjacency matrix illustrated in figure 1. The state evolution, given a random initial condition, is illustrated for the first node in figure 2. A training dataset of 5,000 samples and a test dataset of 10,000 samples in the format given in (13) were generated using a 5-th order Runge-Kutta ODE solver. Initial conditions for x_1 and x_2 were sampled from a uniform distribution on the range $[-1,1]$.

Solving the SOS optimization problem (15) each node of the system was found to be dissipative with finite L_2 gain of 1 and the 4-th order polynomial storage function

$$V(x_1,x_2) = 5x_1^4 + 100x_2^2 + 20 \tag{18}$$

A proof that this storage function satisfies the dissipativity inequality is provided in section VII.

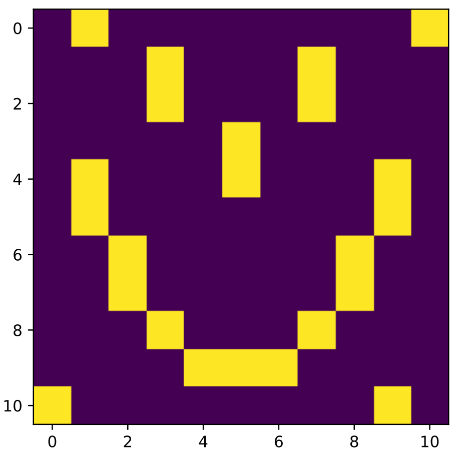

Fig. 1: Adjacency matrix associated with the connected dampened oscillators system.

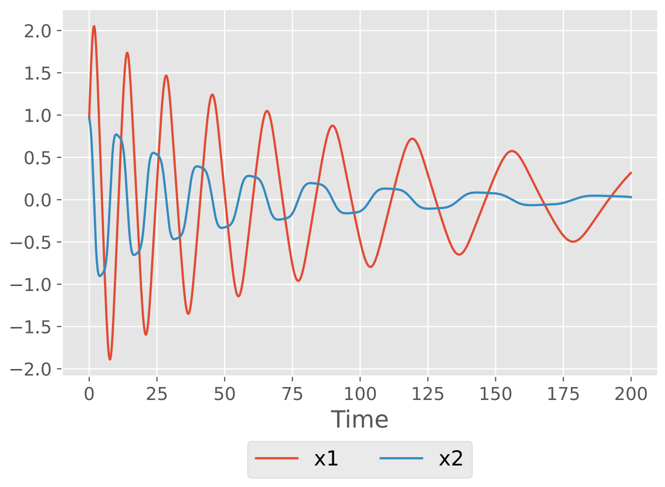

Fig. 2: The evolution of a single node within the 11-node network of dampened nonlinear oscillators.

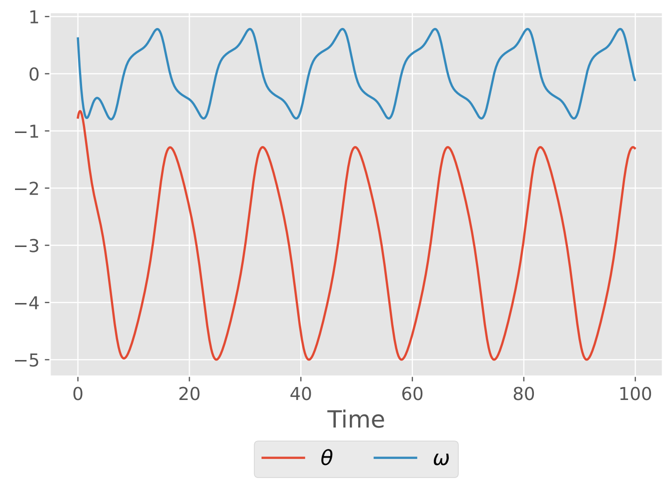

Fig. 3: The evolution of a single node within the 3-node network of dampened pendulums.

B. Dampened Pendulums with Sinusoidal Coupling

The second distributed dynamical system we examined is a set of three dampened pendulums with sinusoidal coupling. The system dynamics for the i-th node, $i \in \{1, \dots, N\}$, are governed by

$$\begin{aligned} \dot{\theta}_i &= \omega_i \\ \dot{\omega}_i &= -\frac{k}{m}\sin(\theta_i) - \frac{d}{m}\omega_i + \frac{1}{m}u_i \\ u_i &= \sum_{j=1}^{N} \mathbf{A}_{ij}\sin(\theta_j - \theta_i) \end{aligned} \tag{19}$$

where $k = 1$, $d = 1$ and $m = 1$ were chosen for simulations. The following adjacency matrix was used for interconnecting the three pendulums:

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{20}$$

Figure 3 shows the evolution of θ and ω for the first node of the network.

A training dataset of 50,000 samples and a test set of 10,000 samples were created for neural network training and evaluation. Initial conditions for θ and ω were sampled from uniform distributions on the ranges $[-\pi, \pi]$ and $[-1, 1]$ respectively.

Since these system dynamics are not polynomial, a storage function cannot be obtained using SOS programming. Instead, the storage function

$$V(\theta, \omega) = \frac{m}{2}\omega^2 + k(1 - \cos(\theta)) \tag{21}$$

built from well-known physics energy functions for kinetic and potential energy, fulfills the dissipativity inequality with respect to the supply rate ωu (passivity). A proof for this is provided in section VII.

C. Learned Models and their dissipative properties

The neural dynamical model $\mathbf{F}_{NN}$ is composed of two feed-forward networks: f_{NN}, consisting of two hidden layers of width 50 with LeakyReLU activation function, and g_{NN} with a single hidden layer of width 4 and the same activation function. An Adam optimizer with base learning rate 0.02 was used for joint training of f_{NN} and g_{NN}. Models were trained for 20 epochs.

For both systems, three different training runs were considered: a naive unregularized approach without dissipativity objective (λ_0 = 0) and two dissipativity-encouraging experiments (λ_1 = 0.001, λ_2 = 0.01).

For the dampened nonlinear oscillator network with linear coupling depicted in section V-A prediction versus ground truth comparisons for an example trajectory are illustrated in figure 4. For a numerical comparison, the test average mean squared errors are shown in table I. Results indicate that the mean squared error for λ_1 is the lowest, however comparable to λ_0, as opposed to λ_2 that shows a significant increase.

The learned models are also investigated for their dissipative properties. Figure 4 shows the supply rate subtracted by the derivative of the storage function, $s(u, y) - \dot{V}(x)$, over a time frame for the first node of the network. As stated by the dissipativity inequality (3), when this function is negative, the system is no longer dissipative. The percentage of time during which the inequality is violated as well as the average dissipativity loss as specified in section IV-B are reported on the test dataset in table I. The neural networks trained with λ_2 and λ_1 show a considerable improvement on dissipativity percentage and loss.

λ	Mean squared Error	Percentage of time not dissipative	Dissipativity Loss
0	6.706×10^{-6}	12.4%	0.039
0.001	$\mathbf{6.666\times10^{-6}}$	1.7%	6.419×10^{-4}
0.01	9.986×10^{-6}	**0.5%**	$\mathbf{1.403\times10^{-4}}$

TABLE I: Mean squared error, percentage of time not dissipative and dissipativity loss compared across different weights λ for the dissipativity loss. Models were trained on the dampened nonlinear oscillator network. The best result is in **bold** for the respective metric.

For the dampened sinusoidally coupled pendulums described in section V-B, similar results are obtained. Figure 5 portrays prediction versus ground truth trajectory comparisons for all three models, as well as the time evolution of the dissipativity inequality. Further, table II contains the MSE, time percentage and average dissipativity losses on the test dataset for these models. Results show that even though models trained with λ_0 and λ_1 are comparable, the model trained with a more aggressive dissipativity contraint, λ_2, shows an improvement not only on dissipativity, but also on system approximation as a whole, as pointed out by the MSE reduction.

λ	Mean squared Error	Percentage of time not dissipative	Dissipativity Loss
0	5.776×10^{-5}	2.0%	4.871×10^{-5}
0.001	4.883×10^{-5}	2.1%	1.9×10^{-5}
0.01	$\mathbf{2.73\times10^{-5}}$	**1.6%**	$\mathbf{0.841\times10^{-5}}$

TABLE II: Mean squared error, percentage of time not dissipative and dissipativity loss compared across different weights λ for the dissipativity loss. Models were trained on the dampened pendulum network. The best result is in **bold** for the respective metric.

VI. Discussion

Recent advancements in AI present significant opportunities in data-driven system identification. In this study, we have explored the impact of enforcing a dissipativity constraint on the learning of distributed dynamical systems using neural networks.

Our experiments portray how even if neural networks are suitable candidates for system dynamics approximation from data, these do not necessarily inherit underlying system properties like dissipativity. Our findings demonstrate that incorporating a dissipativity penalty does not significantly reduce the accuracy of trajectories predicted by a neural dynamical model in a mean squared error sense and, in some cases, even slightly improves it. In turn the adherence of the neural network model to dissipativity properties inherent to the underlying dynamical system may be greatly improved with our method. Such adherence to dissipativity can be a desirable property in data-driven system modeling, for example when stability of the modeled distributed system is important.

Future work should extend this methodology in data-inferred storage function paradigms, allowing for end-to-end dissipativity-enforcing with no prior knowledge of the dynamics. Moreover, the development of more sophisticated neural architectures that inherently incorporate such constraints could further improve the modeling capabilities and expand the applicability of neural networks in this field.

VII. Proofs

In this section we include proofs of dissipativity for the storage functions obtained for both systems.

A. Storage Function for Dampened Nonlinear Oscillators

As stated in section V-A, the storage function found through SOS programming for this system is depicted in equation (18) with $\alpha = 0.1$, $\beta = 0.01$ and $k = 0.01$. For this to be a valid storage function it must satisfy being non-negative, clear from the function definition

$$V(x) = V(x_1, x_2) = 5x_1^4 + 100x_2^2 + 20 > 0 \tag{22}$$

and the dissipativity inequality (3). The proof for L_2 dissipativity with gain $\gamma = 1$ is shown next.

$$\begin{aligned} s(u,y) - \dot{V}(x) &= \gamma^2 u^2 - x_2^2 - (\nabla_x V(x))^T \dot{x} \\ &= u^2 - 2ux_2 + x_2^2 \\ &= (u - x_2)^2 \\ &\geq 0 \end{aligned} \tag{23}$$

B. Storage Function for Dampened Pendulums

For the second system, presented in section V-B, the proposed storage function is shown in equation (21) with $k = 1$, $d = 1$ and $m = 1$. For it to be valid, it must be non-negative

$$V(\theta, \omega) = \frac{1}{2}\omega^2 + (1 - \cos(\theta)) \geq 0 \tag{24}$$

and satisfy the dissipativity inequality (3) for a passive supply rate. The proof for passivity of this system is shown next.

$$\begin{aligned} s(u,y) - \dot{V}(x) &= uy - (\nabla_x V(x))^T \dot{x} \\ &= u\omega - (u\omega - \omega^2) \\ &= \omega^2 \\ &\geq 0 \end{aligned} \tag{25}$$

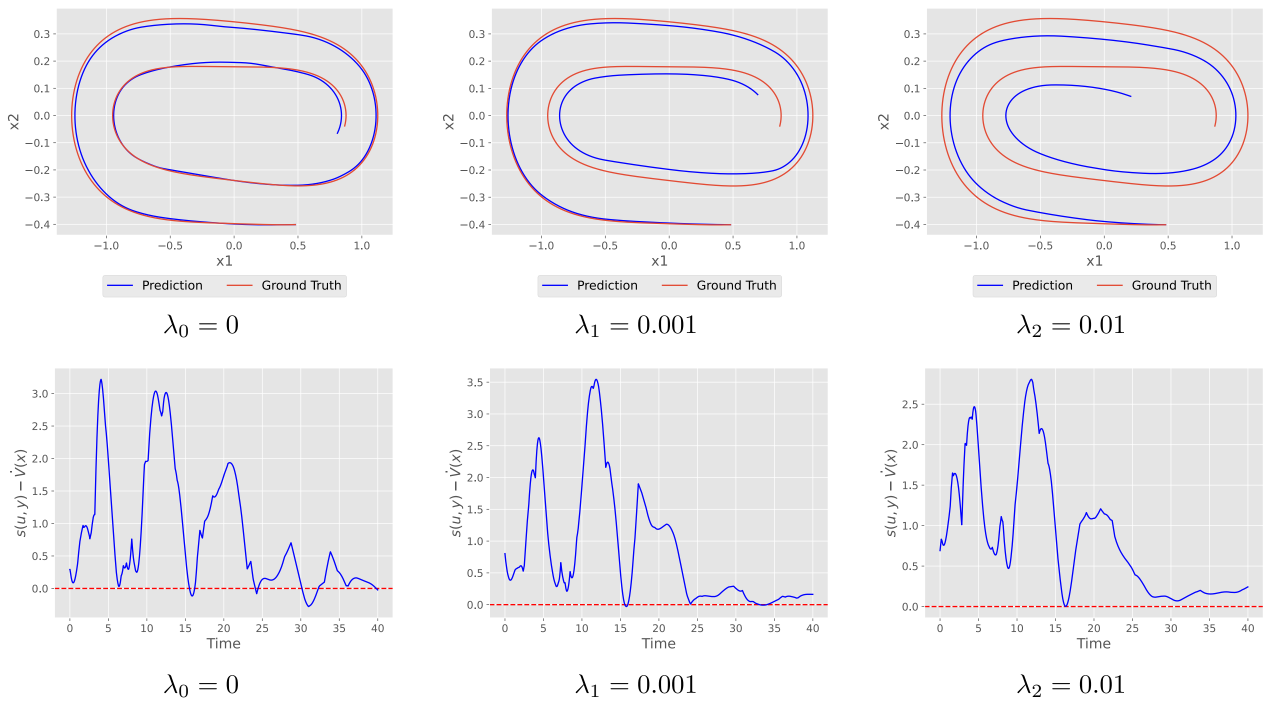

Fig. 4: Above: Comparison of the trajectories of a single node for models trained on the dampened oscillator system with linear coupling. Models trained with $\lambda_0 = 0$, $\lambda_1 = 0.001$ and $\lambda_2 = 0.01$ are each compared the the ground truth system. Below: Dissipativity condition $s(u, y) - \dot{V}(x)$ as a function of time for the trajectories above.

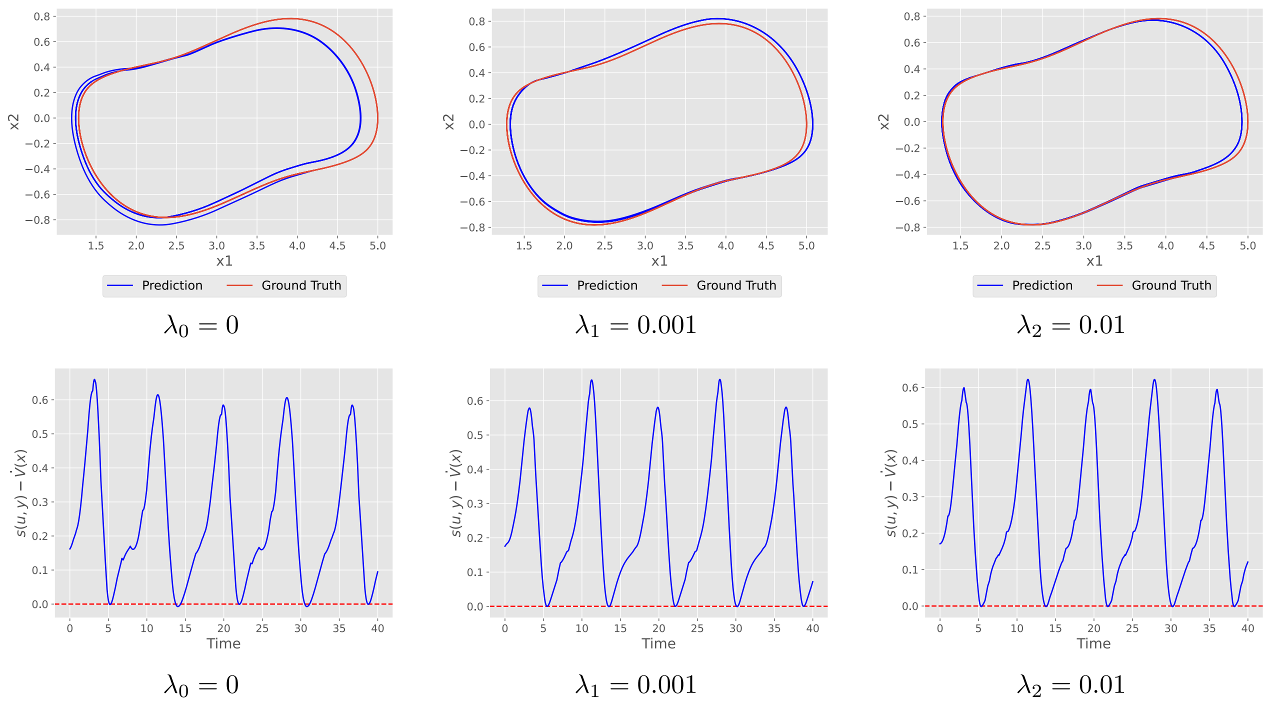

Fig. 5: Above: Comparison of the trajectories of a single node for models trained on the dampened pendulum system with sinusoidal coupling. Models trained with $\lambda_0 = 0$, $\lambda_1 = 0.001$ and $\lambda_2 = 0.01$ are each compared the the ground truth system. Below: Dissipativity condition $s(u, y) - \dot{V}(x)$ as a function of time for the trajectories above.

References

[1] Jian Liu et al. "Distributed event-triggered fixed-time consensus for leader-follower multiagent systems with nonlinear dynamics and uncertain disturbances". In: *International Journal of Robust and Nonlinear Control* 28 (11 July 2018), pp. 3543–3559. ISSN: 10991239. DOI: 10.1002/rnc.4098.

[2] Kang Sun, Xiangmo Zhao, and Xia Wu. "A cooperative lane change model for connected and autonomous vehicles on two lanes highway by considering the traffic efficiency on both lanes". eng. In: *Transportation research interdisciplinary perspectives* 9 (2021), p. 100310. ISSN: 2590-1982.

[3] Jan Drgona et al. "Dissipative Deep Neural Dynamical Systems". In: *IEEE Open Journal of Control Systems* 1 (June 2022), pp. 100–112. DOI: 10.1109/ojcsys.2022.3186838.

[4] David J Hill and Peter J Moylan. *Disspative Dynamical Systems: Basic Input-output and State Properties*. 1980.

[5] Jan C. Willems. "Dissipative Dynamical Systems". In: *European Journal of Control* 13.2 (2007), pp. 134–151. ISSN: 0947-3580. DOI: https://doi.org/10.3166/ejc.13.134-151. URL: https://www.sciencedirect.com/science/article/pii/S0947358007708166.

[6] Truong X. Nghiem et al. *Physics-Informed Machine Learning for Modeling and Control of Dynamical Systems*. 2023. arXiv: 2306.13867 [eess.SY]. URL: https://arxiv.org/abs/2306.13867.

[7] Christian Møldrup Legaard et al. *Constructing Neural Network-Based Models for Simulating Dynamical Systems*. 2022. arXiv: 2111.01495 [cs.LG]. URL: https://arxiv.org/abs/2111.01495.

[8] Yuezhu Xu and S. Sivaranjani. "Learning Dissipative Neural Dynamical Systems". In: (Sept. 2023). DOI: 10.1109/LCSYS.2023.3337851. URL: http://arxiv.org/abs/2309.16032%20http://dx.doi.org/10.1109/LCSYS.2023.3337851.

[9] S. Sivaranjani, Etika Agarwal, and Vijay Gupta. "Data-Driven Identification of Dissipative Linear Models for Nonlinear Systems". In: *IEEE Transactions on Automatic Control* 67.9 (2022), pp. 4978–4985. DOI: 10.1109/TAC.2022.3180810.

[10] Pablo A. Parrilo and Rekha R. Thomas. *Sum of squares : theory and applications*. Providence, Rhode Island: American Mathematical Society, 2019.

[11] James Koch et al. "Structural Inference of Networked Dynamical Systems with Universal Differential Equations". In: (July 2022). DOI: 10.1063/5.0109093. URL: http://arxiv.org/abs/2207.04962%20http://dx.doi.org/10.1063/5.0109093.

[12] Chenyang Yuan. *SumOfSquares.py*. URL: https://github.com/yuanchenyang/SumOfSquares.py.