Generalizing the SINDy approach with nested neural networks

Camilla Fiorini, Clément Flint, Louis Fostier, Emmanuel Franck, Reyhaneh Hashemi, Victor Michel-Dansac, Wassim Tenachi

▶ **To cite this version:**

Camilla Fiorini, Clément Flint, Louis Fostier, Emmanuel Franck, Reyhaneh Hashemi, et al.. Generalizing the SINDy approach with nested neural networks. 2024. hal-04557263v2

HAL Id: hal-04557263
https://hal.science/hal-04557263v2

Preprint submitted on 26 Jan 2025

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ESAIM: PROCEEDINGS AND SURVEYS, Vol. ?, 2025, 1-10
Editors: Will be set by the publisher

GENERALIZING THE SINDy APPROACH WITH NESTED NEURAL NETWORKS

Camilla Fiorini[1], Clément Flint[2], Louis Fostier[3, 4], Emmanuel Franck[2], Reyhaneh Hashemi[5], Victor Michel-Dansac[2] and Wassim Tenachi[6]

Abstract. Symbolic Regression (SR) is a widely studied field of research that aims to infer symbolic expressions from data. A popular approach for SR is the Sparse Identification of Nonlinear Dynamical Systems (SINDy) framework, which uses sparse regression to identify governing equations from data. This study introduces an enhanced method, Nested SINDy, that aims to increase the expressivity of the SINDy approach thanks to a nested structure. Indeed, traditional symbolic regression and system identification methods often fail with complex systems that cannot be easily described analytically. Nested SINDy builds on the SINDy framework by introducing additional layers before and after the core SINDy layer. This allows the method to identify symbolic representations for a wider range of systems, including those with compositions and products of functions. We demonstrate the ability of the Nested SINDy approach to accurately find symbolic expressions for simple systems, such as basic trigonometric functions, and sparse (false but accurate) analytical representations for more complex systems. Our results highlight Nested SINDy's potential as a tool for symbolic regression, surpassing the traditional SINDy approach in terms of expressivity. However, we also note the challenges in the optimization process for Nested SINDy and suggest future research directions, including the designing of a more robust methodology for the optimization process. This study proves that Nested SINDy can effectively discover symbolic representations of dynamical systems from data, offering new opportunities for understanding complex systems through data-driven methods.

[1] Conservatoire National des Arts et Métiers, Laboratoire M2N, 75003, Paris
[2] Université de Strasbourg, CNRS, Inria, IRMA, F-67000 Strasbourg, France
[3] PRC, INRAE, CNRS, Université de Tours, Nouzilly, France
[4] Université Paris-Saclay, Inria, Inria Saclay-Île-de-France, Palaiseau, France
[5] Aix–Marseille University, Aix-en-Provence, France
[6] Université de Strasbourg, CNRS, Observatoire astronomique de Strasbourg, UMR 7550, F-67000 Strasbourg, France

© EDP Sciences, SMAI 2025

Résumé. La régression symbolique est un domaine de recherche bien établi qui cherche à déduire des expressions symboliques directement à partir de données. L'une des approches les plus reconnues dans ce domaine est celle de l'Identification Parcimonieuse de Systèmes Dynamiques non Linéaires (Sparse Identification of Nonlinear Dynamical Systems, SINDy), qui recourt à la régression parcimonieuse pour extraire les expressions des équations sous-tendant les données observées. Dans cette étude, nous proposons une méthode avancée, dénommée "Nested SINDy", qui vise à améliorer l'expressivité de la méthode SINDy grâce à une structure multi-couches. En effet, les limites des méthodes traditionnelles de régression symbolique et d'identification de systèmes concernent principalement les sytèmes complexes, dont la description analytique n'est pas aisée. En s'appuyant sur la méthode SINDy, Nested SINDy introduit des couches supplémentaires avant et après la couche principale de SINDy. Ceci lui permet d'inférer des représentations symboliques pour une plus grande gamme de systèmes, incluant ceux décrits par des équations formées de compositions et de produits de fonctions. Nous démontrons la capacité de Nested SINDy à identifier avec précision des expressions symboliques pour des équations simples, tels que des fonctions trigonométriques basiques, et à fournir des représentations analytiques creuses (fausses, mais précises) pour des systèmes plus complexes. Nos résultats mettent en lumière le potentiel de Nested SINDy comme outil puissant pour la régression symbolique, surpassant les capacités expressives de la méthode SINDy traditionnelle. Toutefois, nous soulignons également les défis rencontrés dans le processus d'optimisation de Nested SINDy et proposons des pistes pour des recherches futures, notamment en vue de développer une méthodologie d'optimisation plus robuste. Cette étude établit que Nested SINDy est capable de découvrir efficacement des représentations symboliques des systèmes dynamiques à partir de données, ouvrant ainsi de nouvelles voies pour la compréhension des systèmes complexes via des méthodes basées sur les données.

1. Introduction

Symbolic regression (SR) consists in the inference of a free-form symbolic analytical function $f : \mathbb{R}^{n_1} \longrightarrow \mathbb{R}^{n_2}$ that fits $\mathbf{y} = f(\mathbf{x})$ given data $(\mathbf{x}, \mathbf{y})$. It is distinct from regular numerical optimization procedures in that it consists in a search in the space of functional forms themselves by optimizing the arrangement of mathematical symbols (e.g., $+$, $-$, $\times$, $/$, sin, cos, exp, log, …).

The rationale for employing SR can be broadly categorized into the following three core objectives.

(1) SR can be used to produce models in the form of compact analytical expressions that are interpretable and intelligible. This objective is particularly vital in natural sciences, such as physics [61], where the capacity to explain phenomena is equally valuable as predictive prowess. This is typically probed by assessing the capability of a system to recover the exact symbolic functional form from its associated data. However, one should note that many SR approaches excelling in this metric are often bested in fit accuracy when exact symbolic recovery is unsuccessful [21]. In other contexts where the compactness and inherent intelligibility of expressions may not be as critical, significantly longer but more robustly accurate expressions ($> 10^3$ mathematical symbols) are desirable as SR still offers key advantages in such scenarios.
(2) SR demonstrates the advantage of producing models that frequently exhibit superior generalization properties when compared to neural networks. [16, 17, 42, 55, 60]
(3) Another noteworthy advantage is the ability to create models that demand significantly fewer computational resources than extensive numerical models like neural networks. This efficiency becomes especially relevant in *multi-query* scenarios such as control loops [17, 55], optimization or uncertainty quantification, where models must be executed frequently, and thus computational efficiency is crucial.

SR has traditionally been approached through genetic programming, where a population of candidate mathematical expressions undergoes iterative refinement using operations inspired by natural evolution, such as natural selection, crossover, and mutation. This approach includes well-known tools like Eureqa software [43, 44], as

well as more recent developments [20, 46, 58, 59]. Additionally, SR has been explored using a diverse array of probabilistic methods [3, 7, 15, 29, 52]. For recent SR reviews, refer to [1, 21, 30].

The rise of neural networks and auto-differentiation[1] has spurred significant efforts to incorporate these techniques into SR, challenging the dominance of Eureqa-like approaches [21, 31, 33]. Numerous methods for integrating neural networks into SR have been developed, ranging from advanced problem simplification schemes [53, 54] to end-to-end supervised symbolic regression approaches in which neural networks are trained in a supervised manner to map datasets to their corresponding symbolic functions [4, 5, 9, 10, 14, 16, 22, 27, 28, 34, 57]. Unsupervised approaches also exist, where recurrent neural networks are trained through trial-and-error using reinforcement learning to generate analytical expressions that fit a given dataset [11, 13, 23, 24, 26, 36, 39, 50, 62]. Furthermore, it should be noted that it has been a major focus of the SR community to facilitate the incorporation of prior knowledge to constrain the search for functional forms by leveraging domain-specific knowledge [2, 6, 12, 18, 35, 40, 47–49] and that SINDy-like frameworks as the one proposed here can accommodate such prior knowledge, as demonstrated in works like [41].

Supervised approaches offer rapid inference but lack a self-correction mechanism. If the generated expression is suboptimal, there are little means of correction. In contrast, unsupervised approaches enable iterative correction based on fit quality. However, they often rely on reinforcement learning frameworks to approximate gradients because direct optimization using auto-differentiation is infeasible due to the discrete nature of the problem, which involves discrete symbolic choices.

However, other unsupervised methods include neuro-symbolic approaches, wherein mathematical symbols are integrated into neural network frameworks. The goal being to sparsely fit the neural network to enable interpretability, generalization or even recover a compact mathematical expression. Prominent examples include SINDy [8], which is central to this study, and others such as [19, 32, 37, 38, 42, 45, 56].

SINDy-like approaches are the only type of unsupervised techniques capable of directly utilizing gradients from data to iteratively refine function expressions as they effectively render the discrete symbolic optimization problem continuous. Moreover, SINDy-like frameworks possess the advantage of being well-suited for exact symbolic recovery by enabling the creation of concise, intelligible analytical expressions through the promotion of sparse symbolic representations while yielding highly accurate and general expressions when exact symbolic recovery is unsuccessful or impossible. However, a limitation of the current SINDy framework is its inability to handle nested symbolic functions, which often results in suboptimal performances, especially in more complex problems as evidenced by comparative benchmarks (see e.g., [47]). This is the primary motivation for our study, where we introduce a Nested SINDy approach.

The paper is organized as follows. First, we introduce the traditional SINDy approach in Section 2. Then, we present our Nested SINDy approach in Section 3, with two distinct architectures: the PR model in Section 3.2 and the PRP model in Section 3.3, the latter being more expressive, but also more challenging to train, than the former. The training procedure is explained in detail in Section 4, and two main applications are tackled in Section 5: function identification in Section 5.1 and ODE discovery in Section 5.2. Finally, we conclude in Section 6.

2. The SINDy paradigm

In this section, we present the traditional SINDy approach, as introduced in [8]. The principle of the method is detailed in Section 2.1, while the mathematical framework and notation are set in Section 2.2.

2.1. Principle of the SINDy method

The Sparse Identification of Nonlinear Dynamical Systems (SINDy) approach extends previous work in SR, introducing innovations in sparse regression. The SINDy approach seeks to deduce the governing equations of a nonlinear dynamical system directly from observational data, doing so in a concise and sparse way. It is based

[1] Leveraging the capabilities of deep learning libraries to meticulously track gradients associated with a set of parameters in relation to a numerical process, regardless of its intricacy.

on the essential assumption that these governing equations can be succinctly expressed by only a few significant terms, resulting in a sparse representation within the space of potential functions [8].

More specifically, the SINDy approach involves approximating a target function through a linear combination of (potentially nonlinear) basis functions, contained in a so-called library or dictionary $\mathcal{F}$. For instance, $\mathcal{F}$ might include constant, polynomial, or trigonometric functions:

$$\mathcal{F} = \begin{Bmatrix} x \mapsto 1, & x \mapsto x, & x \mapsto x^2, & x \mapsto x^4, & \dots \\ x \mapsto \sin(x), & x \mapsto \cos(x), & x \mapsto \sin(2x), & x \mapsto \cos(\pi x), & \dots \end{Bmatrix}. \tag{2.1}$$

In order to achieve expressiveness, it is necessary to chose a large number of basis functions. However, with that many basis functions, there is a risk of losing interpretability. To address this issue, a sparsity constraint is imposed on the coefficients of the linear combination.

The main advantages of the SINDy method are the following:

- The use of underlying convex optimization algorithms ensures the method's applicability to large-scale problems [8].
- The resulting nonlinear model identification inherently balances model complexity (i.e., sparsity of the function to be learned) with accuracy, leading to strong generalization ability.
- SINDy automatically identifies the relevant terms in the dynamical system without making prior assumptions about the system's form, through the use of gradient descent.

The main limitations of the method include:

- The necessity to carefully select the appropriate library $\mathcal{F}$ based on the available data: for instance, compositions or multiplications of simpler functions have to be included in the dictionary to correctly represent more complex target functions.
- Training (to determine which functions to keep in the dictionary) is more sensitive to initialization than with other approaches.

The goal of the following section is to set the mathematical framework and the main notation associated to the SINDy method, to be used throughout this paper.

2.2. Mathematical framework and notation

For the sake of simplicity, we present the method in the case where the target function is from $\mathbb{R}$ to $\mathbb{R}$, but the approach can be extended to functions from $\mathbb{R}^n$ to $\mathbb{R}^m$.

Given the data $(x_i, y_i)_{i=1,\dots,N}$, we aim to find a function f such that $f(x_i) \approx y_i$ (which is nothing but a regression problem) using the SINDy approach.

Let $\mathcal{F} = \{f_1, \dots, f_l\}$ be the aforementioned dictionary of basis functions. For instance, it could be the one given by (2.1). We denote by $L(\mathcal{F}) = \text{Span}(\mathcal{F})$ the set of linear combinations of these basis functions, defined by

$$f \in L(\mathcal{F}) \iff \exists \theta \in \mathbb{R}^l \text{ such that } f = \sum_{i=1}^{l} \theta_i f_i. \tag{2.2}$$

The regression problem can then be formulated as the following least squares problem:

$$\min_{f \in L(\mathcal{F})} \|Y - f(X)\|_2^2,$$

where $X = (x_1, \dots, x_N)^T$ and $Y = (y_1, \dots, y_N)^T$. This problem can itself be reformulated in matrix form, using the definition of the vector space $L(\mathcal{F})$:

$$\min_{\theta \in \mathbb{R}^l} \|Y - \mathbb{F}(X)\theta\|_2^2 \tag{2.3}$$

where $[\mathbb{F}(X)]_{i,j} = (f_j(x_i)) \in \mathcal{M}_{N,l}(\mathbb{R})$.

Numerous algorithms exist to solve this problem while promoting sparsity. Without being exhaustive, notable methods include the standard STLSQ (sequentially thresholded least squares) and the LARS (least-angle regression) methods. Another approach involves adding a regularization term on the coefficients of the linear combination to favor sparsity. The most popular is Lasso regularization, but others exist (SR3, SCAD, MCP, ...). In this work, we focus on the Lasso approach. Introducing a Lasso regularization term to promote sparsity, the optimization problem for the SINDy approach becomes, instead of (2.3):

$$\min_{\theta \in \mathbb{R}^l} \|Y - \mathbb{F}(X)\theta\|_2^2 + \lambda \|\theta\|_1, \tag{2.4}$$

where $\lambda > 0$ is a hyperparameter, to be manually set when using the method. The values of λ will be reported when using the method in the following sections. Specialized optimization algorithms, such as ADMM (alternating direction method of multipliers), are effective in solving regression problems with regularization.

3. The Nested SINDy approach

As mentioned in Section 2.1, one of the main limitations of the SINDy approach is the choice of the nonlinear basis functions populating the dictionary $\mathcal{F}$. For instance, if the unknown function happens to be a composition or a multiplication of simple functions, we cannot find the correct expression unless this specific composition or multiplication is in $\mathcal{F}$. In this paper, we aim at relaxing this constraint by introducing a way of composing simple functions, without having to manually add these compositions to the dictionary.

In the same spirit as the approach investigated in [32, 42], we will enlarge the set $L(\mathcal{F})$. We will proceed by analogy with a standard approach in machine learning, which involves considering models with multi-layer neural networks rather than a single broad layer. Here, instead of considering a single layer of nonlinear functions, we explore an augmented architecture, consisting of several such layers. We will refer to this approach as Nested SINDy.

The cost to bear is the heightened complexity of the optimization landscape. Indeed, the optimization problem, used to be the linear least squares problem given by (2.4). Now, it becomes a nonlinear problem, since the matrix $\mathbb{F}(X)$ is replaced with a composition of nonlinear functions. The new optimization problem (still with Lasso regularization) is formulated as follows:

$$\min_{\theta} \frac{1}{2} \|y - \mathcal{N}(x, \theta)\|_2^2 + \lambda \|\theta\|_1,$$

where $\mathcal{N}$ denotes our nested model, parameterized by θ. Consequently, the resolution of this nonlinear optimization problem may be significantly more challenging and algorithms that work well in the linear case do not necessarily have the same performance in the nonlinear case.

The method then primarily depends on the choice of the architecture $\mathcal{N}$. The goal is to introduce new layers that achieve favorable trade-offs between expressivity and optimization complexity. In this work, we propose two architectures, which are described in the following sections: the PR model (in Section 3.2) and the PRP model (in Section 3.3). Both of are based on the polynomial layer, described in Section 3.1. From now on, we call the basic SINDy layer, given by a projection (2.2) onto $L(\mathcal{F})$, the radial layer.

3.1. Polynomial layers

To improve the expressivity of the usual SINDy approach, we introduce an additional layer, to be applied after the usual SINDy expression. This layer constructs a variety of monomials from the input variable x, represented as follows:

$$f_{\text{poly}}(x) = \sum_{i=0}^{d} \omega_i x^i,$$

where d represents the maximum allowed polynomial degree and ω_i are the weights. Note that ω_0 is the constant part of the layer, which corresponds to the bias in traditional neural networks. For inputs with multiple variables, the layer extends to a multivariate polynomial, facilitating complex combinations of the variables. For example, with two variables x and y, we obtain

$$f_{\text{poly}}(x, y) = \sum_{i=0}^{d} \sum_{j=0}^{d} \omega_{i,j} x^i y^j,$$

with $\omega_{0,0}$ acting as the constant term of the polynomial, effectively substituting the bias. Another choice is to limit the sum over $i + j$ to a maximum value d to reduce the number of terms in the polynomial, and thus obtain bivariate polynomials up to degree d. This strategy is applied in Section 3.3; with two variables x and y, such a polynomial layer reads:

$$f_{\text{poly}}(x, y) = \sum_{0 \le i+j \le d} \omega_{i,j} x^i y^j. \tag{3.1}$$

3.2. **The PR Model**

The PR (Polynomial-Radial) model augments the basic SINDy framework by introducing a polynomial layer that operates before the usual SINDy radial layer. As an example, for one input variable, if $\mathcal{F} = \{\sin, \cos\}$ and $d = 2$, then the PR model can learn all functions with expression

$$\lambda \cos(a_1 + b_1 x + c_1 x^2) + \mu \sin(a_2 + b_2 x + c_2 x^2).$$

This is way more expressive than standard SINDy, where functions such as $x \mapsto \cos(2x)$ or $x \mapsto \sin(1 + x^2)$ would have to be manually added to the dictionary.

This PR layer can also be seen as a pure polynomial layer combined with a linear layer. We, hence, consider the PR model to have four layers: a polynomial layer from Section 3.1, a (fully-connected) linear layer, a radial layer, and a final linear layer. The last two layers are identical to the standard SINDy model, while the first two layers are new additions. Figure 1 illustrates this structure.

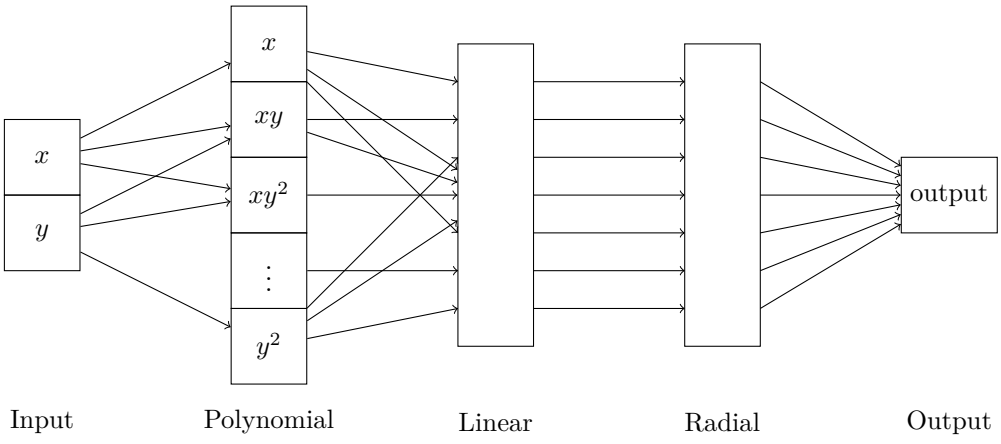


FIGURE 1. Structure of the PR model for $d = 2$ and 2 input variables.

The full expression of the model is:

$$f_{\theta,\text{PR}}(x) = \sum_{j=1}^{l} c_j f_j \left(\sum_{i=0}^{d} \omega_{i,j} x^i \right) + \beta, \tag{3.2}$$

where θ includes all the trainable parameters of the model. The functions f_j are derived from the specified function set $\mathcal{F}$, while $\omega_{i,j}$ are the weights of the polynomial layer, and c_j and β are the weights and bias of the final linear layer, respectively. The parameters $\omega_{i,j}$ now depend on j since they are in the j-th radial layer. It is important to note that the radial and polynomial layers are not associated with any adjustable parameters.

The main advantage of the PR model is its enhanced expressivity, which facilitates the creation of linear combinations both before and after the radial layer. The polynomial layer allows the model to identify more complex functions and to integrate various inputs effectively in the case of multivariate data. Compared with the traditional SINDy approach, the model benefits from a reduced need for an extensive dictionary because it is capable of discovering linear combinations of the functions contained within the dictionary. Observations from our experiments suggest that the training process of the model is capable of converging to correct solutions even for non-trivial problems. This will be highlighted in Section 5.

3.3. **The PRP Model**

The PRP (Polynomial-Radial-Polynomial) model enhances the PR model by introducing an additional polynomial layer following the radial layer. This structure significantly improves the ability of the model to represent complex interactions within datasets. For example, the PRP model can express the function $f(x) = \arctan(x)\sin(x)$, assuming arctangent and sine are in the dictionary, while the PR model cannot, as it is not a linear combination of the functions contained within the dictionary. In the PRP model, the outputs of the radial layer are first processed through a fully-connected linear layer. This linear layer serves as an intermediate stage, transforming the outputs of the radial layer into a new set of variables. These variables are then fed into a subsequent polynomial layer, which allows for the formation of various monomial combinations of the outputs of the linear layers, such as squaring or multiplying them.

The mathematical expression of the PRP model is given as:

$$f_{\theta,\mathrm{PRP}}(x) = \sum_{1\leq|i|\leq d} \omega^{\mathrm{PR}}_{i_1,i_2,\ldots,i_l} f_{1,\mathrm{PR}}(x)^{i_1} f_{2,\mathrm{PR}}(x)^{i_2} \ldots f_{k,\mathrm{PR}}(x)^{i_k} + \beta', \tag{3.3}$$

where $|i| = i_1 + i_2 + \ldots + i_k$ is the length of the multi-index $(i_1, \ldots, i_k)$, θ includes all the trainable parameters of the model, $\omega^{\mathrm{PR}}_{i_1,i_2,\ldots,i_k}$ are the weights of the final linear layer, β' is the bias of the final linear layer, k is the size of the output chosen for the intermediate linear layer (set to $k = 2$ in our experiments), and $f_{1,\mathrm{PR}}(x)$, $f_{2,\mathrm{PR}}(x)$, …, $f_{k,\mathrm{PR}}(x)$ are given by the following equation:

$$f_{k',\mathrm{PR}}(x) = \sum_{j=1}^{l} c_{j,k'} f_j \left(\sum_{i=0}^{d} \omega_{i,j} x^i \right) + \beta_{k'}, \quad \text{for all } 1 \leq k' \leq k.$$

Figure 2 shows a graphical representation of this model.

The addition of a second polynomial layer in the PRP model markedly increases the expressivity of the model. It enables the model to capture more intricate relationships in the data, particularly beneficial for complex datasets where simpler models may fall short. Therefore, the PRP model is especially good at handling datasets with intricate variable interactions.

In summary, the PRP model, with its dual polynomial layers, offers a sophisticated extension of the SINDy approach. It provides a powerful framework for modeling complex systems, capable of capturing higher-order interactions and nonlinear relationships inherent in the data.

3.4. **Downsides of the Nested SINDy approach**

The Nested SINDy approach complexifies the optimization landscape, which destabilizes the training process. We can reasonably assume that the addition of linear layers creates local minima because of their composition with the nonlinear layers. Moreover, it adds a substantial amount of trainable parameters. This is because the number of parameters in the linear layers is quadratic, whereas those in the SINDy approach grow linearly with

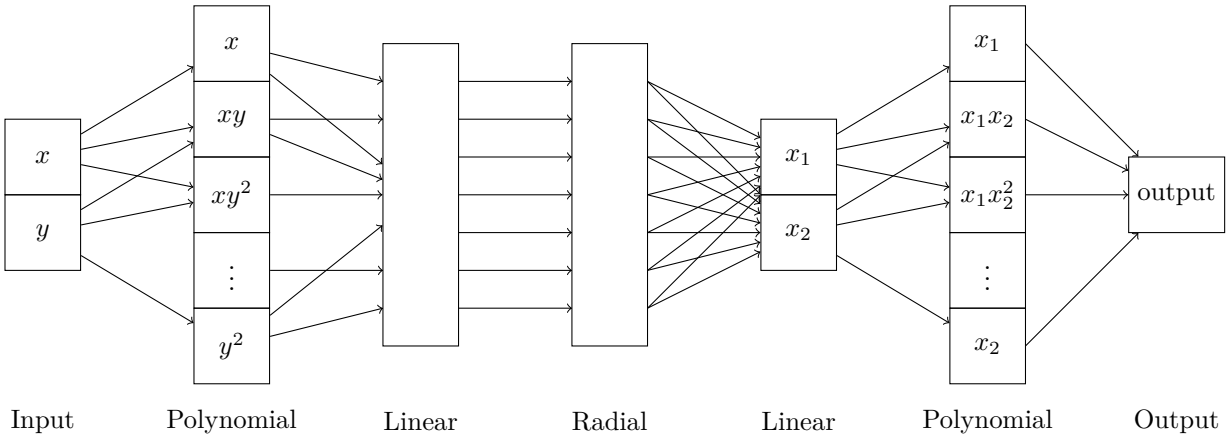


FIGURE 2. Structure of the PRP model for $k = 2$, $d = 2$, and 2 input dimensions.

the number of functions in the dictionary. This deviates from the original SINDy approach, which is oriented towards function discovery, and assumes that the dictionary can become arbitrarily large. In the Nested SINDy approach, the number of functions in the dictionary should remain relatively small to avoid an explosion in the learning time. The next section addresses some training-related issues.

4. Training the nested SINDy model

We experimented and combined various strategies to best overcome challenges associated with nonlinear optimization. These strategies are given below, where we mention how we actually used them for training. Values of the hyperparameters introduced in this section will be given in Section 5.

4.1. Adding a regularization term to enforce sparsity

In our framework, the Lasso regularization term is added to the loss function to enforce sparsity in the model. We have tested different strategies to adapt the Lasso coefficient during training:

- A constant Lasso parameter throughout the training, which is the standard approach.
- A varying Lasso coefficient, initially set to zero for the early epochs, and then taken oscillating around some constant value, depending on the epoch. The intuition of this idea is that "shaking" the learning landscape helps to get out of local minima, in a direction that is still relevant to one of the two objectives (sparsity versus mean squared error).
- We also tried selecting neuron-dependent Lasso coefficients to promote specific functions or layers over others in the radial layer.

The approach that gave the most consistent results across the tested cases was to change the weight of the Lasso coefficient throughout the learning process. In the experiments, the Lasso coefficient is given by

$$\lambda(\text{epoch}) = \lambda_0(1 + 0.5\sin(\text{epoch}/10)), \tag{4.1}$$

where λ_0 is the initial Lasso coefficient, and where the sine function is used to make the coefficient oscillate between $0.5\lambda_0$ and $1.5\lambda_0$.

4.2. Pruning to enforce sparsity

To enhance model sparsity, a complementary approach to Lasso regularization is to prune the neural network during training, reducing the number of parameters. In order to prune the neural network, we remove a parameter θ_i from the set of parameters $(\theta_i)_i$ if the following two conditions are met:

- The mean squared error (MSE) is below a predefined threshold value $\text{MSE}_{\text{prune}}$, and
- $|\theta|$ is below a threshold value $\varepsilon_{\text{prune}}$ for a given number of epochs n_{prune}.

4.3. **Choosing the optimization algorithm**

We tested training our network with fairly standard optimization algorithms implemented in PyTorch but not specifically tailored to our problem: Adam, stochastic gradient descent, Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm. For instance, LBFGS [63] is uncommon in neural network training, as it was designed for optimizing constants in equations. However, it works well in our case, possibly because our optimization problem is close to a classical regression problem. Moreover, it would be interesting to implement a more specific optimization algorithm that takes into account the form of our objective function (mean squared error plus regularization term), such as an ADMM algorithm coupled with a standard PyTorch optimizer. Another promising avenue for SINDy-like approaches is the *basin-hopping* algorithm which combines LBFGS with global search techniques in order to avoid local minima as proposed in [45].

4.4. **Adding noise to the gradient of the loss function**

During a training step, we can add random noise to the gradient just before updating weights. This can be done at each training step, or only when the loss function does not vary sufficiently, e.g. when stuck in a local minimum.

The noise amplitude has to be well-tuned. It depends on the learning rate ℓ of the optimizer, the values of the parameters θ, and on the actual mean squared error L_{MSE}:

$$\nabla_\theta L \leftarrow \nabla_\theta L + \varepsilon(\ell, L_{\text{MSE}}, \theta). \tag{4.2}$$

with

$$\varepsilon \sim \mathcal{N}\left(0, \alpha \times \ell \times \min(L_{\text{MSE}}, 1) \times |\theta|\right), \text{ and } \alpha \text{ a parameter to tune.}$$

By making the learning process less deterministic, we hope to more easily escape local minima. In the same spirit, we could also directly add noise to the parameters when the loss function is stuck in a local minimum.

4.5. **Initializing the network parameters**

The training process is very sensitive to the weight initialization, given the presence of multiple local minima in the objective function. Furthermore, concerning the exploration of Ordinary Differential Equations (ODEs), for certain parameter values, the solution to the ODE may be ill-defined (or only locally defined), emphasizing the importance of careful weight initialization for our model. This is indeed one of the main limitations of the model and, consequently, an area for possible improvement.

5. Applications

In this work, we consider two main applications: function discovery in Section 5.1, and Ordinary Differential Equation (ODE) discovery in Section 5.2. Function discovery consists in recovering the expression of a function from a dataset, while ODE discovery aims at finding the differential equation that governs the evolution of a system from a dataset. Therefore, function discovery can be seen as a stepping stone towards ODE discovery.

5.1. **Function discovery**

We first tackle function discovery. To that end, we present four test cases, of increasing complexity.

5.1.1. *Case 1: trigonometric function involving composition using the PR block*

In Section 2.1, we recalled that the standard SINDy method encounters difficulties when the target is a function defined as the composition of several simpler functions, unless that specific composite function is included in the dictionary. We wish to demonstrate, in such cases, the capability of the proposed PR nested SINDy method (3.2) described in Section 3.2. To that end, the function $f : x \mapsto \cos(x^2)$ is considered over the interval $[0, 3]$. With a dataset comprising 10^4 data points and utilizing a single input dimension, the PR-nested SINDy model attempts to replicate this function.

The architecture of the model is as follows. The first layer is polynomial (P), involving monomials up to the third degree (i.e. x, x^2, and x^3). This is followed by a linear layer, which transforms the three features into seven, with a bias term included. Subsequently, the radial layer (R) applies a series of predefined functions on a node-wise basis, belonging to the following dictionary:

$$\mathcal{F} = \left\{ x \mapsto x, x \mapsto x^2, \arctan, \sin, \cos, \exp, x \mapsto \log(|x| + 10^{-5}), x \mapsto \frac{1}{1+x^2} \right\}.$$

The last linear layer maps these seven transformed features down to a single output.

The hyperparameters and training settings are as follows:

- Optimizer: the Adam optimization algorithm is used, with a learning rate of 10^{-3}.
- Weight initialization: the P and R layers do not use any weight initialization. The linear layer, however, applies a normal distribution initialization, where the weights are drawn from a Gaussian distribution with a specified mean (μ) set to 0 and a standard deviation (σ) set to 1.5.
- Batch: the training is performed on 10 000 uniformly distributed values, with a batch size of $n_{\text{batch}} = 1\,000$, resulting in 10 iterations per epoch.
- Sparsity: Lasso regularization is applied at a rate of 10^{-3}, and the model weights undergo pruning every 30 epochs with a threshold set to 0.05.

The function uses a `patience` counter to keep track of how many consecutive epochs have passed without significant improvement in the loss function. This is calculated by comparing the relative difference in loss between the current and previous epochs to a predefined threshold (here, 10^{-2}). If the relative difference is lower than this threshold for 50 consecutive epochs (`patience` = 50), the training stops. In this case, the training stopped after 300 epochs out of a maximum of 1 000.

The learning process of the PR model with the specified parameters successfully approximated the target function with the expression $x \mapsto -\sin(x^2 - 1.57)$ that captures the underlying behaviour of the target function within the considered domain, see Figure 3. Indeed, at the end of the training, the absolute and relative MSEs are respectively given by

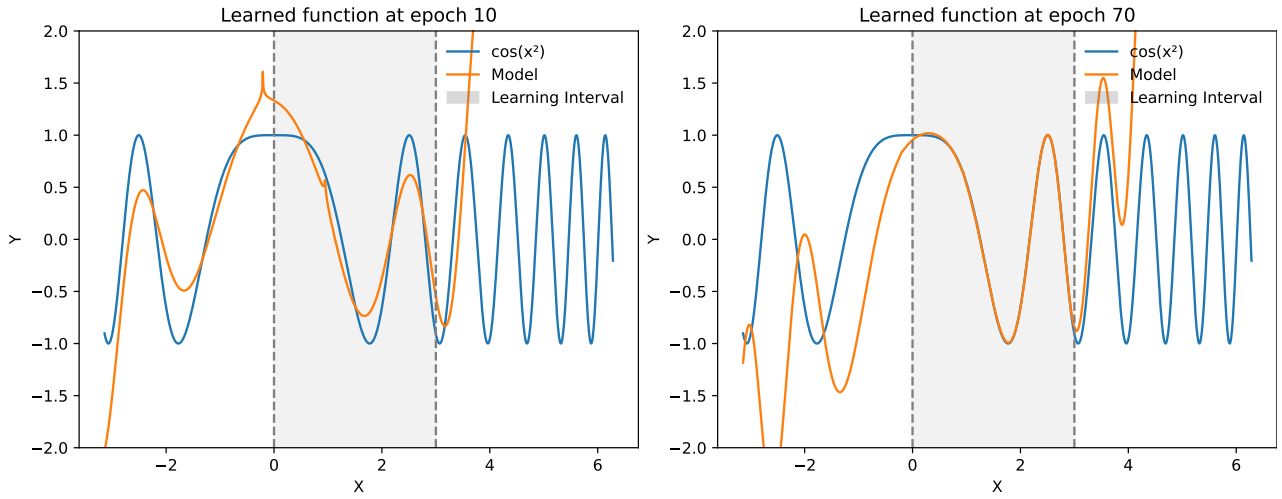
$$\int_0^3 \left(-\sin(x^2 - 1.57) + \cos(x^2)\right)^2 \mathrm{d}x = 8.31 \times 10^{-7} \quad \text{and} \quad \frac{\int_0^3 \left(\sin(x^2 - 1.57) + \cos(x^2)\right)^2 \mathrm{d}x}{\int_0^3 cos^2(x^2)\mathrm{d}x} = 4.92 \times 10^{-7}.$$

We clearly see that the behaviour of the function is captured up to (single-precision) machine error. The sparsity of the solution is significant, with only 3 nonzero coefficients out of the original 29 parameters, underscoring the effectiveness of the model in identifying the essential structure of the function within the given domain.

5.1.2. *Case 2: trigonometric function involving composition using the PRP block*

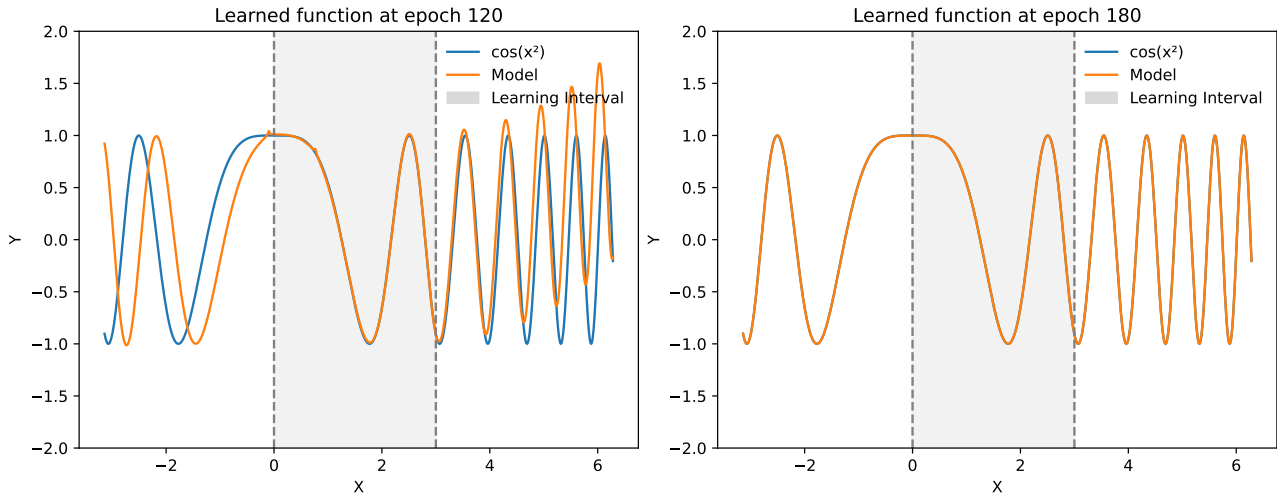
To further illustrate the capabilities of the introduced PRP-Nested SINDy method (3.3) described in Section 3.3, we examine the same function as in Section 5.1.1, namely $f : x \mapsto \cos(x^2)$, considered over the interval $[0, 3]$. This time, employing a dataset comprising 1 000 data points, the PRP-Nested SINDy model is tasked with replicating this function.

The model's architecture begins with a polynomial (P) layer that computes monomials up to the second degree (i.e. with monomials x and x^2). A subsequent linear layer then transforms these two polynomial



(A) Epoch 10: early model predictions vs. target function, illustrating the initial learning phase.

(B) Epoch 70: model predictions show improved alignment with the target function as learning progresses.



(C) Epoch 120: further refined predictions, with the model beginning to capture the function's periodicity.

(D) Epoch 180: the model has approximated the target function.

FIGURE 3. Case 1 from Section 5.1.1: evolution of the learned function $x \mapsto \cos(x^2)$ over successive training epochs. Each subfigure represents the PR model's predictions (in blue) against the target function (in orange) at epochs 10, 70, 120, and 180, showcasing the model's progressive learning and convergence towards approximation of the target function. We observe that the first epochs learn on the interval $[0, 3]$, while subsequent epochs are able to generalize.

features into nine outputs, incorporating a bias term in the process. The radial (R) layer follows, applying a suite of predefined functions to each node, belonging to the following dictionary:

$$\mathcal{F} = \left\{ x \mapsto x, x \mapsto x^2, \arctan, \sin, \cos, \exp, x \mapsto \sqrt{x}, x \mapsto e^{-x^2}, x \mapsto \log(1 + e^x) \right\}.$$

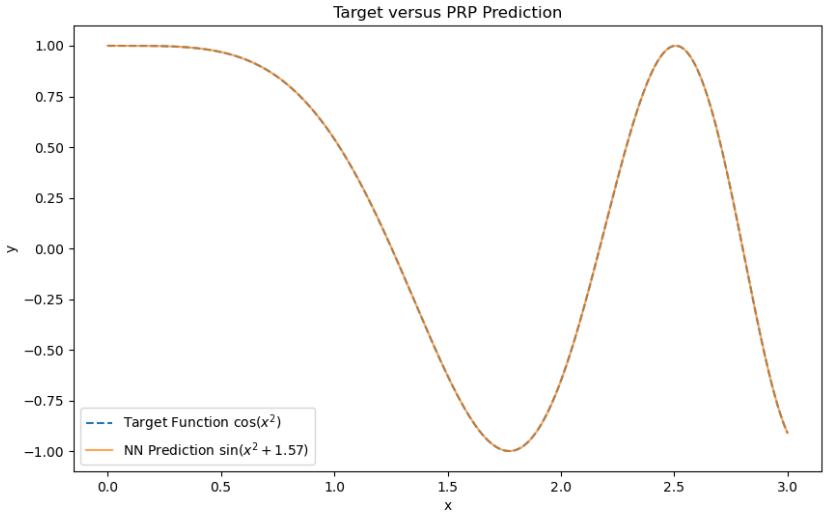


FIGURE 4. Case 2 from Section 5.1.2: comparison of the results from the PRP-Nested SINDy, which yielded the function $x \mapsto \sin(x^2 + 1.57)$, with the target function $x \mapsto \cos(x^2)$, over the interval $[0, 3]$.

The architecture then introduces an intermediate linear layer with output size set to $k = 2$, a second polynomial (P) layer still with $d = 2$, i.e. with monomials x, y, xy, x^2, y^2, and a final linear layer, thus completing the structure of the model.

The hyperparameters match those of Section 5.1.1, except that the Adam learning rate is set to 10^{-4}. We note that the loss function stagnates after around 300 epochs.

The PRP model's learning process successfully approximated the expression $x \mapsto \sin(x^2 + 1.57)$. This expression effectively captures the underlying behaviour within the specified domain (Figure 4). After training, the absolute and relative MSEs are respectively given by

$$\int_0^3 \left(\cos(x^2) - \sin(x^2 + 1.57)\right)^2 \mathrm{d}x = 8.31 \times 10^{-7} \quad \text{and} \quad \frac{\int_0^3 \left(\cos(x^2) - \sin(x^2 + 1.57)\right)^2 \mathrm{d}x}{\int_0^3 \cos^2(x^2)\mathrm{d}x} = 4.92 \times 10^{-7}.$$

Just like before, the behaviour of the function is captured up to (single-precision) machine error. The resulting model exhibits notable sparsity, with only 3 nonzero coefficients from the initial 31 parameters.

5.1.3. *Case 3: Trigonometric Function Multiplication using the PRP Block*

The next test case consists in learning the two-dimensional function $(x, y) \mapsto 2\sin(x)\cos(y)$ over the space domain $[-2, 2]^2$, using PRP blocks. This example is particularly interesting as it highlights the model's ability to learn complex functions involving the multiplication of simple functions, whose product is not included in the dictionary. This is a clear advantage over the naive SINDy approach. Indeed, when two-dimensional input variables are considered, the SINDy function dictionary needs to be much larger (e.g. including all possible products of the functions in the dictionary).

Initially, a Polynomial (P) layer processes the input, generating monomials up to the second degree (i.e. x, y, xy, x^2, y^2). Following this, the first linear layer transforms these polynomial features from 5 inputs to 9 outputs, including a bias term. Next comes the Radial (R) layer, which applies a variety of nonlinear transformations to

each node, belonging to the following dictionary:

$$\mathcal{F} = \left\{ \begin{array}{ccccc} x \mapsto \sqrt{|x| + 10^{-5}}, & x \mapsto x, & \sin, & \cos, & \tanh, \\ \exp, & x \mapsto \dfrac{1}{1+x^2}, & x \mapsto \log(|x| + 10^{-5}), & x \mapsto \exp\left(\dfrac{1}{1+x^2}\right), & x \mapsto \log(1+e^x) \end{array} \right\}.$$

After the R layer, a second P layer is applied, also containing monomials up to degree 2. This additional polynomial processing adds depth to the feature extraction. The architecture then continues with another linear layer, which consolidates the outputs of the preceding layers. This final layer maps the 5-dimensional output from the second P layer to a single output dimension, providing the final model prediction.

The training parameters for learning the function $(x, y) \mapsto 2\sin(x)\cos(y)$ are specifically configured as follows:

- Optimizer: the Adam optimization algorithm is employed with a base learning rate of 10^{-4}, and is run for 1 000 epochs.
- Weight initialization: in the uniform weight initialization for the linear layers, the standard deviation is set to 0.5, and the mean is maintained at 0.
- Batch: the training is performed on 10 000 uniformly distributed values, with a batch size of $n_{\text{batch}} = 1\,000$, resulting in 10 iterations per epoch.
- Sparsity: the Lasso regularization rate is set to 0.1, and the pruning of model weights occurs every 50 epochs with an initial pruning at 10 epochs, using a pruning threshold of 0.01.

Utilizing these parameters, the PRP blocks successfully learn the following function:

$$\begin{aligned}(x, y) \mapsto &- 0.77\left(1 - 0.612\sin\left(0.032x^2 - 0.162xy + 0.998x + 0.035y^2 - y - 0.14\right)\right)^2 \\ &+ 2.01\left(\cos\left(0.039xy - 0.499x - 0.497y + 0.809\right)\right)^2\end{aligned} \tag{5.1}$$

with sparsity accounted for by 14 nonzero coefficients out of the original 32 parameters. A comparison of this function with the target function is illustrated in Figure 5. The MSE between the target function and its approximation is 5.69×10^{-2}.

5.1.4. *Case 4: Perimeter of an ellipse*

Calculating the perimeter of an ellipse is a well-studied topic for which the solution cannot be expressed in terms of elementary functions. Let us first define an ellipse as the set of points (x, y) such that:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1,$$

which can be described by the parametric equations:

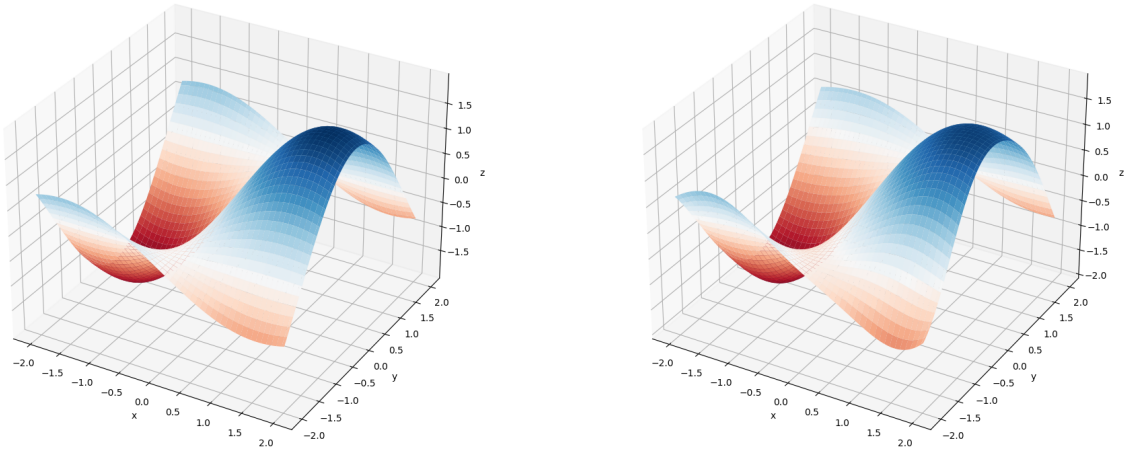
$$\begin{cases} x = a\cos(\alpha), \\ y = b\sin(\alpha), \end{cases}$$

with $\alpha \in [0, 2\pi)$. The perimeter of an ellipse can then be expressed as:

$$P(a, b) = 4\int_0^{\frac{\pi}{2}} \sqrt{a^2\cos^2(\alpha) + b^2\sin^2(\alpha)}\, d\alpha, \tag{5.2}$$

because the differential arc length of our parametric equation is $ds = -\sqrt{((-a\sin(\alpha))^2 + (b\cos(\alpha))^2)}d\alpha$ and the four quadrants have the same length. Several approximations are known to approach the perimeter of an ellipse, such as the one proposed by Ramanujan:

$$P(a, b) \approx \pi(3(a + b) - \sqrt{(3a + b)(a + 3b)}). \tag{5.3}$$



(A) True model ($2\sin(x)\cos(y)$)

(B) Predicted model

FIGURE 5. Case 3: comparison of the true and predicted models. A visual representation of the target function $(x, y) \mapsto 2\sin(x)\cos(y)$ and its learned approximation by the PRP model, given by (5.1), over the specified domain.

Since any rescaled ellipse remains an ellipse, we assume without loss of generality that $b = 1$ for the rest of this experiment. To assess the relevance of our Nested SINDy technique, we will compare its performance against the two reference solutions: Ramanujan's approximation, and a linear interpolation between the two endpoints of the interval $[1, 30]$. Indeed, using a linear interpolation makes sense for large values of a, since P is equivalent to $4a$ when $a \to \infty$, as can be seen by inspecting (5.2).

To find formulas that approximate the perimeter of an ellipse, we train two PRP models. The first one is trained on the interval $[1, 5]$ and the second one on the interval $[1, 25]$. We run 50 learning sessions for each model and keep the model with the shortest associated formula. The only difference between the learning sessions is the seed used for initializing the random number generator. A learning session consists of two trainings, each with its own parameters:

- Optimizer: the Adam optimization algorithm is used for both trainings. The first training is run for 100 epochs with a base learning rate of 10^{-1}, while the second one includes 1 500 epochs and a learning rate of 10^{-3}.
- Weight initialization: the initial weights are uniformly sampled with mean 0 and standard deviation 0.5.
- Batch: the training is performed on 1 000 uniformly distributed values, using a single batch.
- Sparsity: the Lasso regularization rate is set to 10^{-1} in the first training and 10^{-2} in the second one. The pruning occurs after 30 epochs below the threshold of 5×10^{-2} in both trainings.

Overall, the first training aims to move faster near a solution, while the second one aims to converge to a precise solution.

In its best session, the first model found a basic quadratic polynomial:

$$P_{\text{quadratic}}(a) = 0.061(a + 0.544)^2 + 3.28a + 2.72. \tag{5.4}$$

However, other relatively sparse solutions were found, such as a model with 9 nonzero parameters

$$P_1(a) = 1.65a + 0.553(0.485a + 0.135\log(0.817|a^2|) + 1)^2 + 0.459\log(0.817|a^2|) + 3.4,$$

or one with 15 nonzero parameters

$$\begin{aligned} P_2(a) = {} & 0.535a + 0.966(0.394a + 0.721\arctan(0.278a^2 + 0.393) + 1 \\ & + 0.111\exp(-0.063a^4))^2 + 0.978\arctan(0.278a^2 + 0.393) + 1.36 + 0.15\exp(-0.063a^4), \end{aligned}$$

thus demonstrating the ability of the model to find a variety of approximations. The MSE obtained at the end of the training are 2.26×10^{-3}, 2.69×10^{-3}, and 3.30×10^{-3} for the quadratic, P_1, and P_2 models, respectively.

The second model found a more complex expression which is too long to be displayed here. In this second model, 25 out of the 64 parameters are nonzero, corresponding to approximately 40% sparsity. The MSE obtained at the end of the training is 1.97. This metric, as the previous ones, is biased because the training is stopped at an arbitrary step and there is no final tuning step. Hence, we perform a final tuning step, by writing the final expression of the model and performing a gradient descent to only minimize the MSE. The later mentioned results include this final tuning step and show that the effective MSE can be significantly reduced.

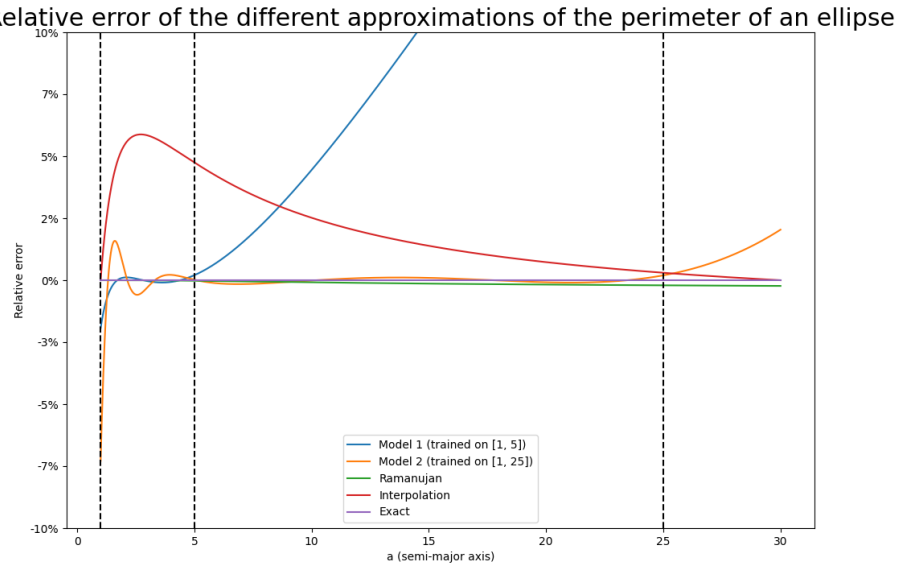


FIGURE 6. Case 4: relative error of the different approximations of the ellipse perimenter on the intervals $[1, 5]$, $[1, 25]$ and $[1, 30]$.

We display the relative error of each approximation on different intervals in Figure 6. Table 1 reports the MSE of each approximation on the intervals $[1, 5]$, $[1, 25]$ and $[1, 30]$. *Model 1* and *model 2* correspond to our two models (with *model 1* corresponding to (5.4)), *Ramanujan* corresponds to the approximation proposed by Ramanujan and *Interpolation* corresponds to the linear interpolation between the two endpoints of the interval $[1, 30]$. The two models perform well on the interval they were trained on. The first model performs better than the linear interpolation on $[1, 5]$, but worse than the Ramanujan approximation. The second model outperforms both the linear interpolation and the Ramanujan approximation on $[1, 25]$, but starts diverging on $[1, 30]$.

TABLE 1. Case 4: mean squared error of the different approximation of the ellipse perimeter on the intervals $[1, 5]$, $[1, 25]$ and $[1, 30]$. The closest approximations are denoted in bold.

	$[1, 5]$	$[1, 25]$	$[1, 30]$
Model 1	7.24×10^{-4}	1.05×10^{2}	2.61×10^{2}
Model 2	7.43×10^{-3}	$\mathbf{3.66 \times 10^{-3}}$	2.73×10^{-1}
Ramanujan	$\mathbf{2.78 \times 10^{-6}}$	9.84×10^{-3}	$\mathbf{1.82 \times 10^{-2}}$
Interpolation	4.70×10^{-2}	4.75×10^{-1}	5.46×10^{-1}

Overall, this experiment provides an insight into the ability of the Nested SINDy approach to discover an approximation on a classical problem without knowledge other than the data points. It demonstrates that this method can be used to discover an approximation of a complex function with little effort. The main downside of this approach is the lack of guarantee to find a sparse solution. The initial choice of the model's coefficients appears to have a significant impact on the final solution.

5.2. **ODE discovery**

In this section, we extend the proposed approach to try to discover an unknown autonomous ODE

$$x'(t) = f(x(t)), \tag{5.5}$$

based on time data associated to K discrete trajectories in time. The full dataset (see Figure 7a) is given by

$$\mathcal{X} = \bigcup_{i=1}^{K} \mathcal{X}^k,$$

where we have defined the k^{th} trajectory as follows:

$$\mathcal{X}^k = \left\{ x_1^{(k)}, \dots, x_N^{(k)} \right\},$$

and the subscripts indicate the time steps. The goal is to provide an approximation f_θ of f in (5.5).

To approximate the dynamical system, we will adapt the method developed by the authors of [25]. They suggest a coupling between SINDy and Neural ODE approaches. Among other things, this method appears to be robust even when the dataset is collected at large or irregular time steps, or contains additive noise.

The idea is, first, to minimize

$$L(\theta) = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \left\| x_{\theta,i}^{(k)} - x_i^{(k)} \right\|_2 + \lambda \left\| \theta \right\|_1$$

where $(x_{\theta,i}^{(k)})_{i \geq 0}$ is the solution of the following Cauchy problem:

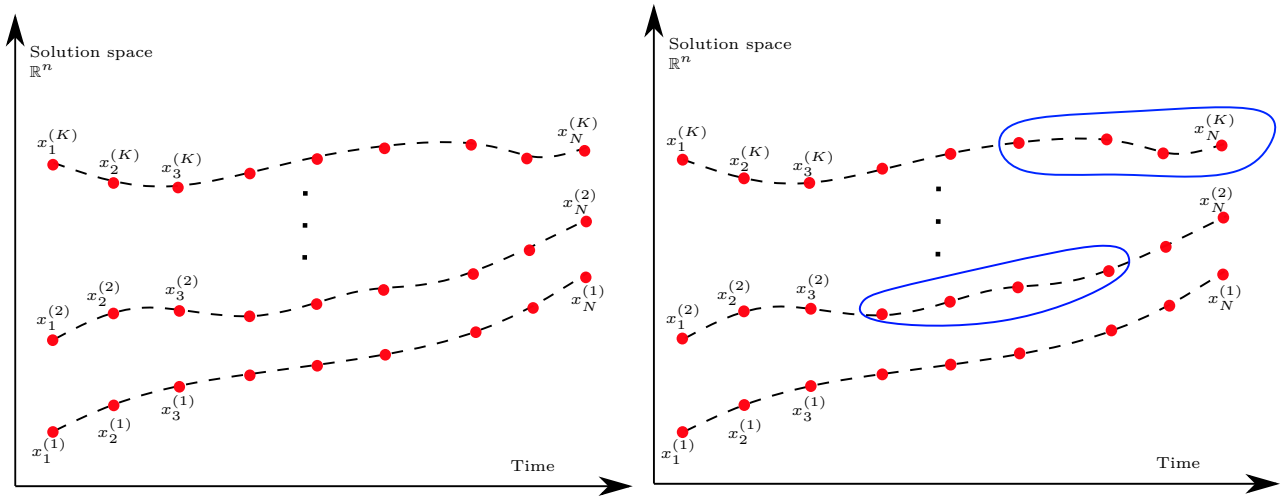
$$\begin{cases} x'(t) = f_\theta(x(t)), \\ x(0) = x_0^{(k)}, \end{cases} \tag{5.6}$$

where f_θ is given by the Nested SINDy network. In practice, the ODE (5.6) is solved numerically using an ODE solver (a fifth-order Runge-Kutta method in our application). Hence, $(x_{\theta,i}^{(k)})_{i \geq 0}$ is an *approximate* solution of (5.6). To train the model, the strategies described in Section 4 are used. One must be careful about how

the dataset is used to train the model. To train efficiently the Nested SINDy network, we divide the dataset in batches at each training step. To do that, we randomly sample n_{batch} trajectories from the dataset $\mathcal{X}$. Then, we randomly sample initial points from the selected n_{batch} trajectories, in such a way that the reduced dataset consists of n_{batch} trajectories of fixed lengths l_{batch}, as shown in Figure 7b. The model parameters are then updated by minimizing the following reduced loss function:

$$L_{\text{red}}(\theta) = \frac{1}{n_{\text{batch}}} \sum_{k=1}^{n_{\text{batch}}} \sum_{i=i_0^{(\sigma(k))}}^{i_0^{(\sigma(k))}+l_{\text{batch}}-1} \left\| x_{\theta,i}^{(\sigma(k))} - x_i^{(\sigma(k))} \right\|_2 + \lambda \left\| \theta \right\|_1$$

where $\mathcal{X}^{\sigma(1)}, \ldots, \mathcal{X}^{\sigma(k)}, \ldots, \mathcal{X}^{\sigma(n_{\text{batch}})}$ are the sampled trajectories, with the associated sampled initial points $i_0^{(\sigma(1))}, \ldots, i_0^{(\sigma(k))}, \ldots, i_0^{(\sigma(n_{\text{batch}}))}$.



(A) Illustration of the full data set for ODE discovery.

(B) Reduced dataset for one training step, with $n_{\text{batch}} = 2$ and $l_{\text{batch}} = 4$ (batches are highlighted in blue).

FIGURE 7. Illustration of data set and batches for the training of the Neural Nested SINDy model for ODE discovery.

Through the applications, we will attempt to demonstrate the added value of the Nested SINDy algorithm compared to the classical one, considering cases where the function f is a composition or product of elementary functions. It should be noted that we will only consider scalar ODEs, but it is entirely possible to explore systems of ODEs with the same method.

5.2.1. *Case 1: A trigonometric function involving composition using the PR block*

Firstly, we try to recover the following ODE:

$$x'(t) = \sin(x^2) \tag{5.7}$$

The function $f : x \mapsto \sin(x^2)$ is a composite function, that could be learned by a PR block. The dataset consists of $K = 100$ trajectories with $N = 500$ constant time steps for each trajectory on the time interval $[0, 1]$. The initial condition of each trajectory is uniformly sampled between -3 and 3.

Our network consists of a single PR block, where the polynomial layer contains monomials up to degree 2, and where the radial layer contains the following functions:

$$\mathcal{F} = \{\ x \mapsto x,\ x \mapsto x^2,\ x \mapsto x^3,\ x \mapsto \sin(x),\ x \mapsto \log(|x|),\ x \mapsto \sqrt{|x|}\ \}.$$

The hyperparameters and training settings are as follows:

- Optimizer: we use the Adam optimizer, with learning rate set to 0.01 and multiplied by 0.999 every 10 epochs. We add noise to the gradient when the loss do not decrease since 100 epochs, as explained in (4.2) with $\alpha = 16$.
- Weight initialization: all the weights are initialized to 0.2.
- Batch: we set $n_{\text{batch}} = 30$ and $l_{\text{batch}} = 5$.
- Sparsity: the Lasso parameter is oscillating around $\lambda_0 = 10^{-4}$ (see (4.1) for details), and pruning parameters are set to $\text{MSE}_{\text{prune}} = 0.01$, $\varepsilon_{\text{prune}} = 0.01$, and $n_{\text{prune}} = 2$.

We train the model for 100 epochs. The results for 10 and 20 epochs are displayed in Figure 8, where we observe that the model has not yet found a satisfactory approximation. The model retrieves the right formula after 100 epochs, as depicted on Figure 9 and in Table 2.

This example showcases the PR model's ability to accurately learn the velocity of an ODE from trajectories. To further validate this ability, future research directions include conducting tests by increasing the dictionary size and using reduced data (fewer trajectories), sparser data (lower sampling frequency in the data), and noisy data.

TABLE 2. Case 1 from Section 5.2.1: evolution of the training of Neural Nested SINDy to recover equation (5.7). The table contains the formulas of f_θ for epochs greater than 50, the mean squared error between f_θ and f, and the number of pruned parameters in the PRP block, at epochs 10, 20, 50, 100.

Epoch	f_θ	MSE on $[-3, 3]$	MSE on $[-5, 5]$	pruned parameters
10	expression too long to display	3.67×10^{-1}	4.16×10^{-1}	12
20	expression too long to display	2.91×10^{-1}	4.16×10^{-1}	14
50	$-0.04\log(\lvert 0.14x^2 + 0.49\rvert) + 0.99\sin(x^2 - 0.02)$	2.29×10^{-4}	7.72×10^{-4}	19
100	$1.00\sin(1.00x^2)$	3.54×10^{-5}	3.00×10^{-4}	23

5.2.2. *Case 2: The Gompertz model*

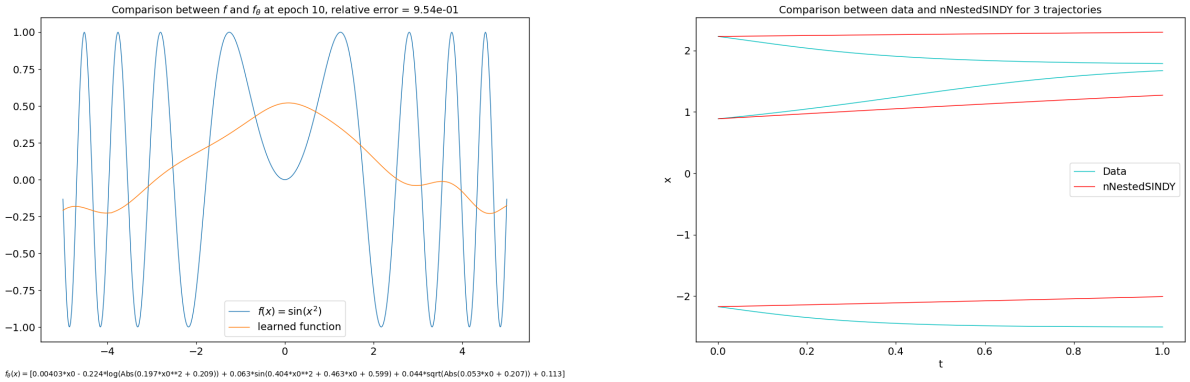
The Gompertz model was initially designed to describe human mortality, but is now applied in various fields, especially in biology, see for instance the review paper [51]. We introduce this model in the context of tumor growth. The evolution of the tumor volume $x(t)$ is governed by the following ODE:

$$x'(t) = rx(t)\log\left(\frac{k}{x(t)}\right) = rx(t)\log(k) - rx(t)\log(x(t)), \tag{5.8}$$

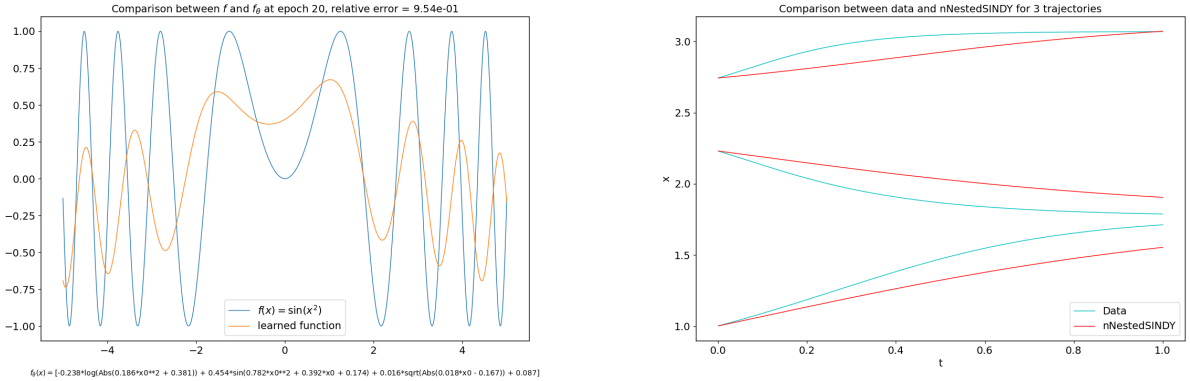
where k is the maximum size that can be reached by the tumor, and r is a constant linked to the cells' proliferative capacity. In the following, we will set $k = 1$ and $r = 2$. By setting $k = 1$, we slightly simplify the formula of the growth speed, which becomes:

$$f(x) = -2x\log(x)\,. \tag{5.9}$$

The exact formula of the growth speed function f can only be recovered with a PRP block, to reconstruct the multiplication between the two basis functions $x \mapsto x$ and log.



(A) After 10 epochs: $f_\theta(x) = 0.004x - 0.224\log(|0.197x^2 + 0.209|) + 0.063\sin(0.404x^2 + 0.463x + 0.599) + 0.044\sqrt{|0.053x + 0.207|} + 0.113$.



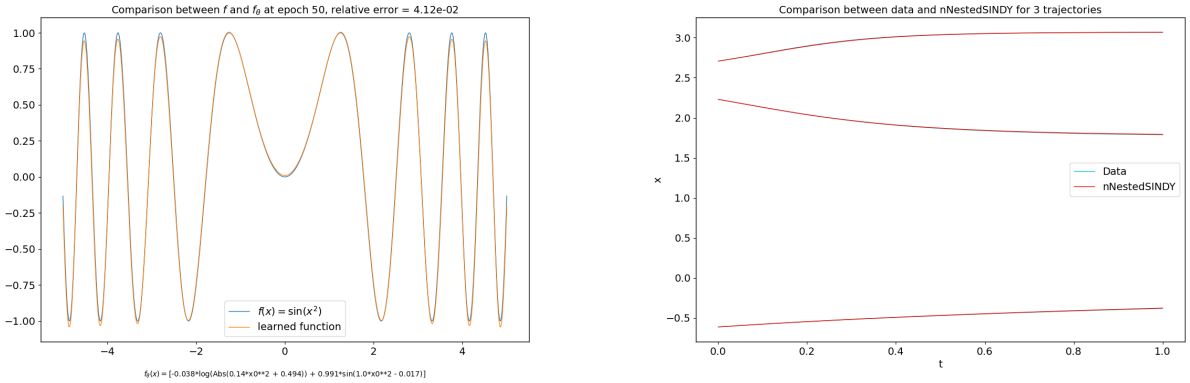
(B) After 20 epochs: $f_\theta(x) = -0.238\log(|0.186x^2+0.381|)+0.454\sin(0.782x^2+0.392x+0.174)+0.016\sqrt{|0.018x - 0.167|}+0.087$.

FIGURE 8. Case 1 from Section 5.2.1: training of Nested SINDy to recover equation (5.7). In the left figures, we compare f (blue curve) and f_θ (orange curve), while in the right figures, we compare the trajectories of dynamic systems (5.6) (red curves) and (5.7) (blue curves) for three different initial conditions. After 20 epochs, the model is still far from the true formula.

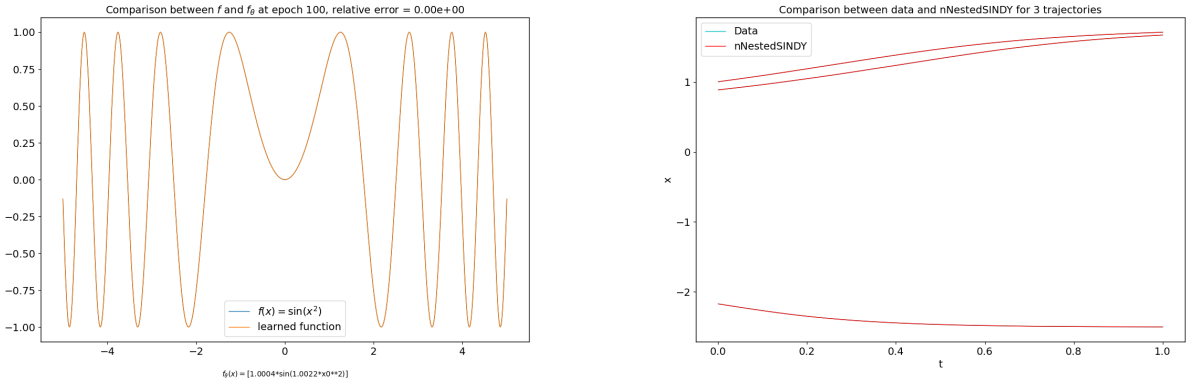
To begin, we attempt to learn the function using the smallest possible PRP block that can still recover the exact formula (see Figure 10). Nevertheless, this block can generate a large number of functions.

As for the previous application case, the dataset consists of $K = 100$ trajectories with $N = 500$ constant time steps for each trajectory. Trajectories are observed over the time interval $[0, 2]$. The initial conditions of each trajectory are uniformly sampled between 0 and 3. For the training, we use the same hyperparameters as in the previous case. The only difference concerns weight initialization, where all the weights randomly initialized according to a normal distribution centered at 0.15 with a standard deviation of 0.05. This introduces randomness when running multiple optimizations.

We train the model for 2000 epochs. Results are summarized in Figure 11 and Table 3. After 1900 epochs, the model approximates up to an error of 10^{-2} the right formula. During the training, the MSE between f and f_θ is not always decreasing, but parameters are (slowly) pruned, and finally the model manages to approximate the right formula.



(A) After 50 epochs: $f_\theta(x) = -0.038\log(|0.14x^2 + 0.494|) + 0.991\sin(x^2 - 0.017)$.



(B) After 100 epochs: $f_\theta(x) = 1.0004\sin(1.0022x^2)$.

FIGURE 9. Case 1 from Section 5.2.1: training of Neural Nested SINDy to recover equation (5.7). In the left figures, we compare f (blue curve) and f_θ (orange curve), while in the right figures, we compare the trajectories of dynamic systems (5.6) (red curves) and (5.7) (blue curves) for three different initial conditions. We observe that the model approximates well the right formula after 100 epochs.

As expected (see Section 5.1.3), PRP models take more effort to retrieve the expected function. In this case we succeed to approximate the true formula of growth speed, but we considered a compact architecture with few dictionary functions, and 20 times more epochs than in the case of the PR model used in the previous case. As a consequence, a major way of improving the PRP models lies in improving the optimization process.

6. CONCLUSION

In this study, we explored the capabilities and limitations of the Nested SINDy approach in discovering symbolic representations of dynamical systems from data. Our investigation covered a spectrum of cases, including the identification of "simple" functions, as well as the discovery of ordinary differential equations (ODEs) from trajectory data. The versatility of the Nested SINDy framework was demonstrated through various examples, showcasing its ability to approximate complex functions and discover the ODEs underlying data-generating processes.

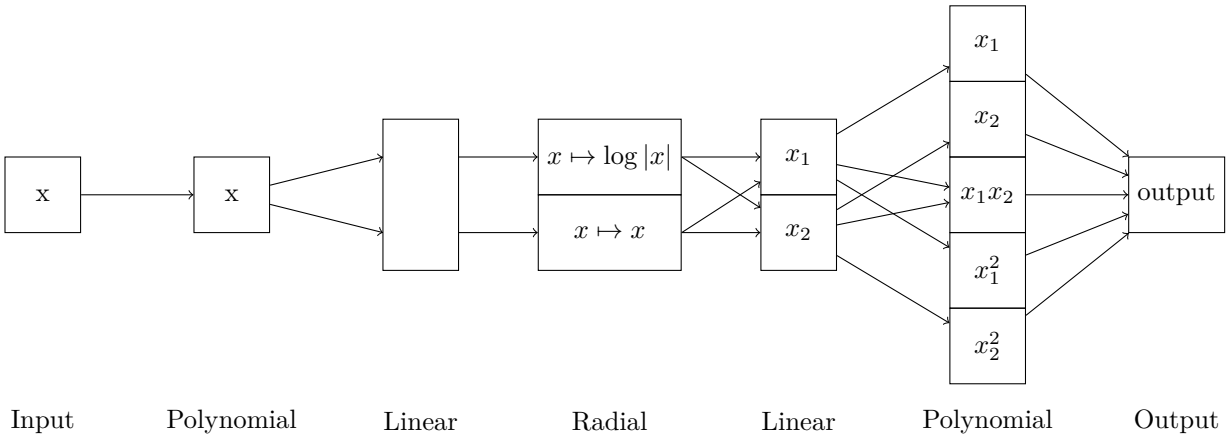


FIGURE 10. Structure of the PRP model used to learn the Gompertz model equation, in case 2.

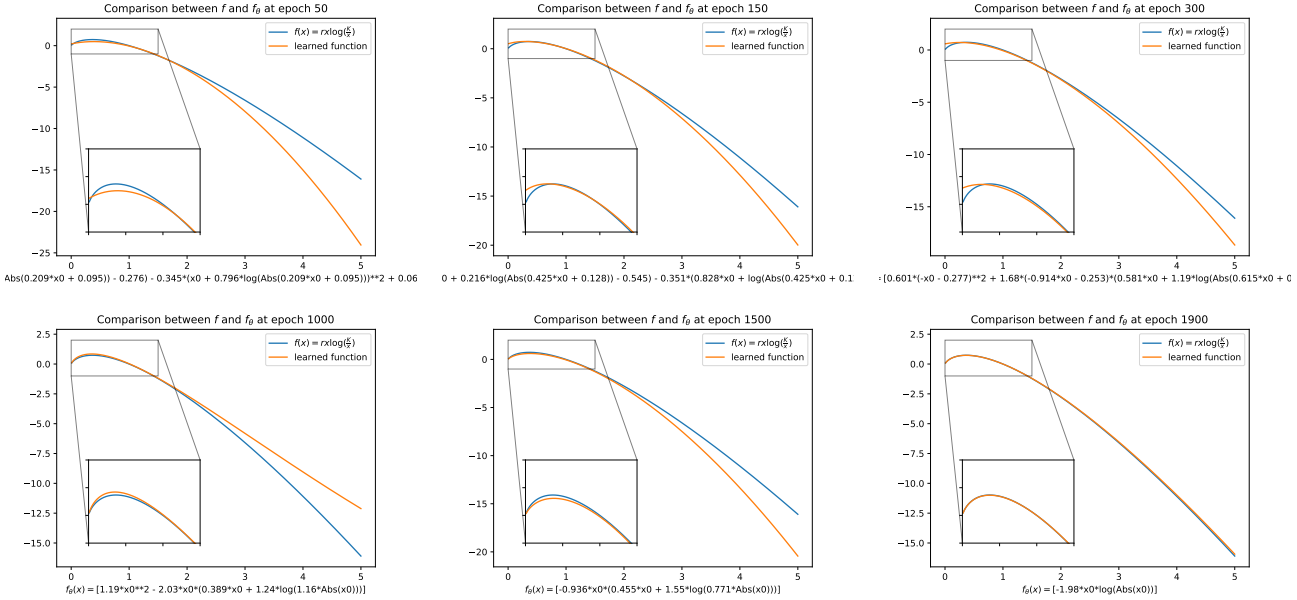
TABLE 3. Case 2: evolution of the training of Neural Nested SINDy to recover equation (5.8). The table contains the formulas of f_θ for epochs greater than 1 000, the mean squared error between f_θ and f, and the number of pruned parameters in the PRP block, at epochs 50, 150, 300, 1000, 1500, 1900.

Epoch	f_θ	MSE on $[0,3]$	MSE on $[0,5]$	pruned parameters
50	expression too long to display	1.50×10^{-1}	3.83×10^{-1}	4
150	expression too long to display	5.18×10^{-2}	1.78×10^{-1}	5
300	expression too long to display	5.31×10^{-2}	1.22×10^{-1}	7
1000	$1.19x^2 - 2.03x(0.39x + 1.24\log(1.16\|x\|))$	9.17×10^{-2}	1.97×10^{-1}	9
1500	$-0.94x(0.46x + 1.55\log(0.77\|x\|))$	2.17×10^{-1}	3.83×10^{-1}	10
1900	$-1.98x\log(\|x\|))$	1×10^{-2}	1×10^{-2}	11

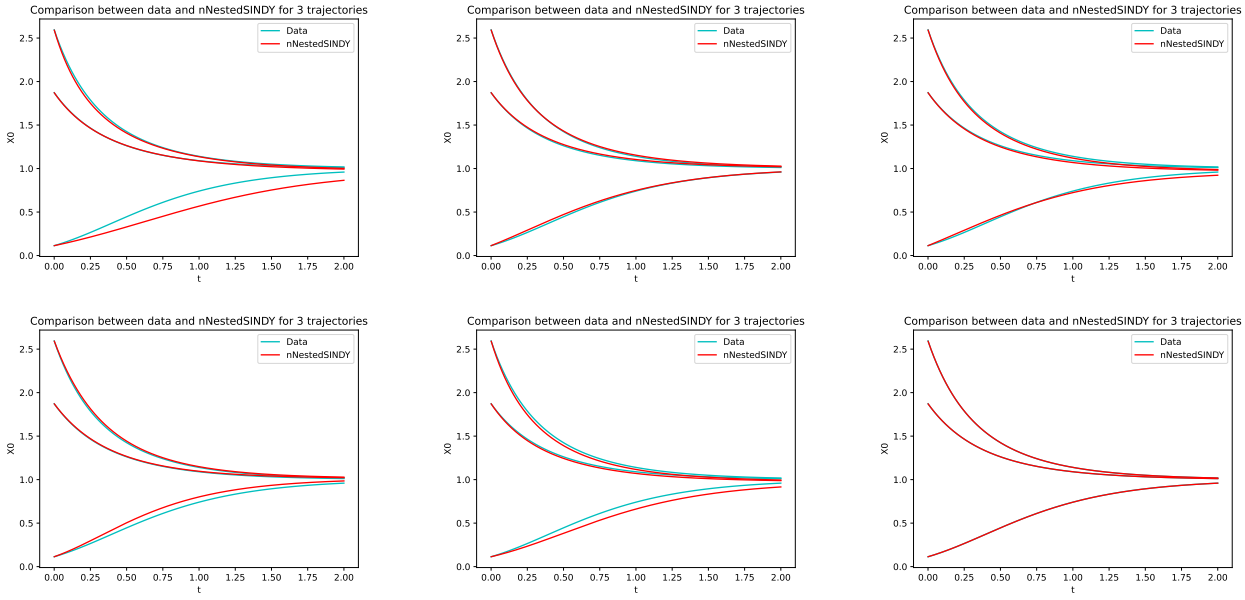
The Nested SINDy approach extends the original SINDy methodology by incorporating nested structures and neural network architectures, allowing for the identification of complex symbolic expressions that are not directly accessible to traditional methods. This capability is necessary in cases involving compositions and multiplications of functions, where the Nested SINDy method accurately identifies symbolic representations in simple cases, or finds sparse symbolic representations in more complex cases.

Our results confirm that Nested SINDy can recover accurate symbolic expressions for a range of problems, from simple trigonometric functions to the more complex Gompertz model of tumor growth. However, the complexity of the optimization landscape associated with the Nested SINDy approach highlights the need for careful selection of hyperparameters and initialization strategies to ensure approximation of meaningful solutions.

Future work could focus on several areas to enhance the Nested SINDy framework. First, exploring alternative optimization algorithms specifically designed for the unique challenges of nested symbolic regression could improve the efficiency and reliability of the method. Additionally, incorporating mechanisms for automatic selection of the dictionary of basis functions based on preliminary data analysis might streamline the model development process and improve the model's adaptability to different types of dynamical systems. Furthermore, one could envision integrating a supervised learning component, as in [45], in which a model is pre-trained to



(A) We compare f given by (5.9) (blue curve) and the learned function f_θ (orange curve), at epochs 50, 150, 300, 1000, 1500, 1900 (from left to right and top to bottom)



(B) We compare trajectories of dynamic systems (5.6) (red curves) and (5.8) (blue curves) for three different initial conditions, at epochs 50, 150, 300, 1000, 1500, 1900 (from left to right and top to bottom)

FIGURE 11. Case 2: training of Neural Nested SINDy to recover equation (5.8). The model converges after 1900 epochs.

map the relationship between datasets and SINDy-like sparse patterns encoding analytical expressions. This effectively enables the automatic formulation of high-quality initial solutions that can then be refined on a case-by-case basis. Finally, extending the Nested SINDy approach to handle partial differential equations (PDEs) could open new avenues for discovering the underlying physics of spatially extended systems from data.

In summary, this study underscores the potential of the Nested SINDy approach as a powerful tool for symbolic regression and dynamical system discovery, while also pointing out avenues for further refinement and expansion of the methodology.

References

[1] D. Angelis, F. Sofos, and T. E. Karakasidis. Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives. *Arch. Comput. Method. E.*, 30(6):3845–3865, 2023.

[2] D. Bartlett, H. Desmond, and P. Ferreira. Priors for Symbolic Regression. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, GECCO '23 Companion, pages 2402–2411, New York, NY, USA, 2023. Association for Computing Machinery.

[3] D. J. Bartlett, H. Desmond, and P. G. Ferreira. Exhaustive Symbolic Regression. *IEEE Transactions on Evolutionary Computation*, 2023.

[4] L. Biggio, T. Bendinelli, A. Lucchi, and G. Parascandolo. A seq2seq approach to symbolic regression. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020.

[5] L. Biggio, T. Bendinelli, A. Neitz, A. Lucchi, and G. Parascandolo. Neural Symbolic Regression that scales. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 936–945. PMLR, 2021.

[6] J. Brence, S. Džeroski, and L. Todorovski. Dimensionally-consistent equation discovery through probabilistic attribute grammars. *Inform. Sciences*, 632:742–756, 2023.

[7] J. Brence, L. Todorovski, and S. Džeroski. Probabilistic grammars for equation discovery. *Knowl.-Based Syst.*, 224:107077, 2021.

[8] S. L. Brunton, J. L. Proctor, and J. N. 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.

[9] T. Chen, P. Xu, and H. Zheng. Bootstrapping OTS-Funcimg Pre-training Model (Botfip) – A Comprehensive Symbolic Regression Framework, 2024.

[10] S. d'Ascoli, S. Bengio, J. Susskind, and E. Abbé. Boolformer: Symbolic Regression of Logic Functions with Transformers. *arXiv e-prints*, page arXiv:2309.12207, 2023.

[11] J. G. Faris, C. F. Hayes, A. R. Goncalves, K. G. Sprenger, D. Faissol, B. K. Petersen, M. Landajuela, and F. Leno da Silva. Pareto Front Training For Multi-Objective Symbolic Optimization. In *The Sixteenth Workshop on Adaptive and Learning Agents*, 2024.

[12] R. Guimerà, I. Reichardt, A. Aguilar-Mogas, F. A. Massucci, M. Miranda, J. Pallarès, and M. Sales-Pardo. A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Sci. Adv.*, 6(5), 2020.

[13] Y. He, B. Sheng, and Z. Li. Channel Modeling Based on Transformer Symbolic Regression for Inter-Satellite Terahertz Communication. *Applied Sciences*, 14(7), 2024.

[14] S. Holt, Z. Qian, and M. van der Schaar. Deep Generative Symbolic Regression. In *The Eleventh International Conference on Learning Representations*, 2023.

[15] Y. Jin, W. Fu, J. Kang, J. Guo, and J. Guo. Bayesian symbolic regression. *arXiv preprint arXiv:1910.08892*, 2019.

[16] P.-A. Kamienny, S. d'Ascoli, G. Lample, and F. Charton. End-to-end Symbolic Regression with Transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[17] P.-A. Kamienny and S. Lamprier. Symbolic-Model-Based Reinforcement Learning. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.

[18] J. T/ Kim, M. Landajuela, and B. K. Petersen. Distilling wikipedia mathematical knowledge into neural network models. In *1st Mathematical Reasoning in General Artificial Intelligence, International Conference on Learning Representations (ICLR)*, 2021.

[19] S. Kim, P. Y. Lu, S. Mukherjee, M. Gilbert, L. Jing, V. Čeperić, and M. Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE Trans. Neural Netw. Learn. Syst.*, 32(9):4166–4177, 2020.

[20] M. Kommenda, B. Burlacu, G. Kronberger, and M. Affenzeller. Parameter identification for symbolic regression using nonlinear least squares. *Genetic Programming and Evolvable Machines*, 21(3):471–501, 2020.

[21] W. La Cava, P. Orzechowski, B. Burlacu, F. de Franca, M. Virgolin, Y. Jin, M. Kommenda, and J. Moore. Contemporary Symbolic Regression Methods and their Relative Performance. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021.

[22] F. Lalande, Y. Matsubara, N. Chiba, T. Taniai, R. Igarashi, and Y. Ushiku. A Transformer Model for Symbolic Regression towards Scientific Discovery. *arXiv e-prints*, page arXiv:2312.04070, 2023.

[23] M. Landajuela, C. S. Lee, J. Yang, R. Glatt, C. P. Santiago, I. Aravena, T. Mundhenk, G. Mulcahy, and B. K. Petersen. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35:33985–33998, 2022.
[24] M. Landajuela, B. K. Petersen, S. K. Kim, C. P. Santiago, R. Glatt, T. N. Mundhenk, J. F. Pettit, and D. M. Faissol. Improving exploration in policy gradient search: Application to symbolic optimization. In *1st Mathematical Reasoning in General Artificial Intelligence, International Conference on Learning Representations (ICLR)*, 2021.
[25] K. Lee, N. Trask, and P. Stinis. Structure-preserving Sparse Identification of Nonlinear Dynamics for Data-driven Modeling. In Bin Dong, Qianxiao Li, Lei Wang, and Zhi-Qin John Xu, editors, *Proceedings of Mathematical and Scientific Machine Learning*, volume 190 of *Proceedings of Machine Learning Research*, pages 65–80. PMLR, 15–17 Aug 2022.
[26] S. Li, I. Marinescu, and S. Musslick. GFN-SR: Symbolic Regression with Generative Flow Networks. In *NeurIPS 2023 AI for Science Workshop*, 2023.
[27] Y. Li, W. Li, L. Yu, M. Wu, J. Liu, W. Li, M. Hao, S. Wei, and Y. Deng. Discovering Mathematical Formulas from Data via GPT-guided Monte Carlo Tree Search, 2024.
[28] Y. Li, J. Liu, W. Li, L. Yu, M. Wu, W. Li, M. Hao, S. Wei, and Y. Deng. MMSR: Symbolic Regression is a Multimodal Task, 2024.
[29] C. Luo, C. Chen, and Z. Jiang. Divide and Conquer: A Quick Scheme for Symbolic Regression. *Int. J. Comput. Methods*, 19(08):2142002, 2022.
[30] N. Makke and S. Chawla. Interpretable scientific discovery with symbolic regression: a review. *Artif. Intell. Rev.*, 57(1), 2024.
[31] I. Marinescu, Y. Strittmatter, C. Williams, and S. Musslick. Expression Sampler as a Dynamic Benchmark for Symbolic Regression. In *NeurIPS 2023 AI for Science Workshop*, 2023.
[32] G. Martius and C. H. Lampert. Extrapolation and learning equations. *CoRR*, abs/1610.02995, 2016.
[33] Y. Matsubara, N. Chiba, R. Igarashi, and Y. Ushiku. SRSD: Rethinking Datasets of Symbolic Regression for Scientific Discovery. In *NeurIPS 2022 AI for Science: Progress and Promises*, 2022.
[34] K. Meidani, P. Shojaee, C. K. Reddy, and A. B. Farimani. SNIP: Bridging Mathematical Symbolic and Numeric Realms with Unified Pre-training. In *The Twelfth International Conference on Learning Representations*, 2024.
[35] D. Melching, F. Paysan, T. Strohmann, and E. Breitbarth. A universal crack tip correction algorithm discovered by physical deep symbolic regression, 2024.
[36] Y. Michishita. Alpha Zero for Physics: Application of Symbolic Regression with Alpha Zero to find the analytical methods in physics, 2024.
[37] R. Ouyang, S. Curtarolo, E. Ahmetcik, M. Scheffler, and L. M. Ghiringhelli. Sisso: A compressed-sensing method for identifying the best low-dimensional descriptor in an immensity of offered candidates. *Phys. Rev. Mater.*, 2:083802, 2018.
[38] M. Panju and A. Ghodsi. A Neuro-Symbolic Method for Solving Differential and Functional Equations. *arXiv preprint arXiv:2011.02415*, 2020.
[39] B. K. Petersen, M. Landajuela Larma, T. N. Mundhenk, C. Prata Santiago, S. K. Kim, and J. Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021.
[40] B. K. Petersen, C. Santiago, and M. Landajuela. Incorporating domain knowledge into neural-guided search via in situ priors and constraints. In *8th ICML Workshop on Automated Machine Learning (AutoML)*, 2021.
[41] T. A. R. Purcell, M. Scheffler, and L. M. Ghiringhelli. Recent advances in the SISSO method and their implementation in the SISSO++ code. *J. Chem. Phys.*, 159(11), 2023.
[42] S. Sahoo, C. Lampert, and G. Martius. Learning equations for extrapolation and control. In *International Conference on Machine Learning*, pages 4442–4450. PMLR, 2018.
[43] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.
[44] M. Schmidt and H. Lipson. *Age-Fitness Pareto Optimization*, pages 129–146. Springer New York, New York, NY, 2011.
[45] P. Scholl, K. Bieker, H. Hauger, and G. Kutyniok. ParFam – Symbolic Regression Based on Continuous Global Optimization. *arXiv e-prints*, page arXiv:2310.05537, 2023.
[46] T. Stephens. GPLearn. https://github.com/trevorstephens/gplearn, 2015.
[47] W. Tenachi, R. Ibata, and F. I. Diakogiannis. Deep Symbolic Regression for Physics Guided by Units Constraints: Toward the Automated Discovery of Physical Laws. *ApJ*, 959(2):99, 2023.
[48] W. Tenachi, R. Ibata, and F. I. Diakogiannis. Physical Symbolic Optimization. *arXiv e-prints*, page arXiv:2312.03612, 2023.
[49] W. Tenachi, R. Ibata, T. L. François, and F. I. Diakogiannis. Class Symbolic Regression: Gotta Fit 'Em All. *arXiv e-prints*, page arXiv:2312.01816, 2023.
[50] Y. Tian, W. Zhou, H. Dong, D. S. Kammer, and Olga Fink. Sym-Q: Adaptive Symbolic Regression via Sequential Decision-Making, 2024.
[51] K. M. C. Tjørve and E. Tjørve. The use of Gompertz models in growth analyses, and new Gompertz-model approach: An addition to the Unified-Richards family. *PLoS One*, 12(6):e0178691, 2017.
[52] T. Tohme, D. Liu, and K. Youcef-Toumi. GSR: A Generalized Symbolic Regression Approach. *Transactions on Machine Learning Research*, 2023.
[53] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark. AI Feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. *Adv. Neur. In.*, 33:4860–4871, 2020.

[54] S.-M. Udrescu and M. Tegmark. AI Feynman: A physics-inspired method for symbolic regression. *Sci. Adv.*, 6(16), 2020.
[55] M. Usama and I.-Y. Lee. Data-Driven Non-Linear Current Controller Based on Deep Symbolic Regression for SPMSM. *Sensors-*, 22(21):8240, 2022.
[56] C. M. C. O. Valle and S. Haddadin. SyReNets: Symbolic Residual Neural Networks. *arXiv preprint arXiv:2105.14396*, 2021.
[57] M. Vastl, J. Kulhánek, J. Kubalík, E. Derner, and R. Babuška. Symformer: End-to-end symbolic regression using transformer-based architecture. *IEEE Access*, 12, 2024.
[58] M. Virgolin, T. Alderliesten, and P. A. N. Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pages 1084–1092, 2019.
[59] M. Virgolin, T. Alderliesten, C. Witteveen, and P. A. N. Bosman. Improving Model-Based Genetic Programming for Symbolic Regression of Small Expressions. *Evol. Comput.*, 29(2):211–237, 2021.
[60] C. Wilstrup and J. Kasak. Symbolic regression outperforms other models for small data sets. *arXiv preprint arXiv:2103.15147*, 2021.
[61] T. Wu and M. Tegmark. Toward an artificial intelligence physicist for unsupervised learning. *Phys. Rev. E*, 100(3):033311, 2019.
[62] W. Zheng, S. P. Sharan, Z. Fan, K. Wang, Y. Xi, and Z. Wang. Symbolic Visual Reinforcement Learning: A Scalable Framework with Object-Level Abstraction and Differentiable Expression Search. *arXiv preprint arXiv:2212.14849*, 2022.
[63] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.