Phase space approach to solving higher order differential equations with artificial neural networks

Floriano Tori [*] and Vincent Ginis [†]
Data Lab/Applied Physics, Vrije Universiteit Brussel, Elsene, Belgium

(Received 11 May 2022; accepted 1 October 2022; published 8 November 2022)

The ability to solve differential equations represents a key step in the modeling and understanding of complex systems. There exist several analytical and numerical methods for solving differential equations, each with their own advantages and limitations. Physics-informed neural networks (PINNs) offer an alternative perspective. Although PINNs deliver promising results, many stones remain unturned about this method. In this paper, we introduce a method that improves the efficiency of PINNs in solving differential equations. Our method is related to the formulation of the problem: Instead of training a network to solve an nth order differential equation, we propose transforming the problem into the equivalent system of n first-order equations in phase space. The target of the network is to solve all equations of the system simultaneously, effectively introducing a multitask optimization problem. We compare both approaches empirically on various problems, ranging from second-order differential equations with constant coefficients to higher-order and nonlinear problems. We also show that our approach is suited for solving partial differential equations. Our results show that the system approach performs equal or better in most experiments performed. We analyze the learning process for the few runs that did not perform well and show that the problem stems from conflicting gradients during training, effectively obstructing multitask learning. The result of this paper is a straightforward heuristic that can be incorporated into any subsequent research that builds on PINNs solving differential equations. Moreover, it also shows how to make PINNs even more efficient by implementing techniques from multitask learning literature.

DOI: 10.1103/PhysRevResearch.4.043090

I. INTRODUCTION

Through artificial neural networks (ANNs), deep learning methods have demonstrated their potential in many scientific fields, from image recognition [1] to chemistry [2], high-energy particle physics [3], and even genomics [4]. The power of these techniques to solve such problems comes from their exceptional capabilities to process large amounts of data and find patterns within. However, purely data-driven approaches are susceptible to limitations. One of these is the lack of guarantee that the models generated will make predictions that respect physical laws which the underlying data follow. In addition, training the networks to obtain accurate predictions requires a large amount of training data, which is not something that can always be assumed in many scientific disciplines. Finally, a purely data-driven approach offers the least transparency and interpretability, two essential features when using machine learning algorithms for scientific insight [5].

One method to solve these problems is to endow networks with prior knowledge with the task they are solving, resulting in informed machine learning [6]. A successful technique for this is physics-informed neural networks (PINNs) [7–9], which incorporates the prior knowledge through a differential equation in the loss term of the network, describing the natural laws which the data follow. This term incentivizes the output of the network to satisfy the physical constraints and acts as a regularization term. Many applications, from solving cosmological phase transitions [10] to predicting plasma flow [9] or even computing the temperature distribution above an espresso cup [11], have shown this to be a viable technique. In addition to enhancing learning, PINNs can be used purely as differential equation solvers because neural networks are universal function approximators [12]. This idea has been around for some time [7,13] but has received more attention due to recent breakthroughs in deep learning.

Solving (partial) differential equations is an essential element in science to understand complex systems, but often, one cannot solve these problems analytically. Therefore, numerical techniques, such as Runge-Kutta, have been developed and optimized over the years to obtain (approximate) solutions. One important difference neural networks have with respect to classical numerical methods is that their result is a continuous and differentiable function. This allows us to use networks to solve differential equations, without predetermining where we should sample the solution, which makes them more flexible tools. This advantage, however, is reduced if the solution contains a discontinuity, as this is a hindrance the network then has to overcome. Although using neural networks to solve differential equations is very promising, this technique is still in its infancy, and much focus in

[*]Floriano.Tori@vub.be
[†]Also at School of Engineering and Applied Sciences, Harvard University; ginis@seas.harvard.edu

recent efforts has been toward understanding the convergence and generalization errors of PINNs [14,15]. In recent work [16–18], it was demonstrated that PINNs are hindered by two limitations. The first one is due to the appearance of stiff gradient flow dynamics as the complexity of the problem is increased, e.g., by adding higher-frequency components or multiscale features [16]. The consequence of this is that unbalanced gradients are backpropagated during learning. Further investigating these problems, the authors of Ref. [18] used neural tangent kernel theory [19,20] to show that PINNs can suffer from a large difference in convergence rate for different terms in the loss function. The second limitation of PINNs originates from spectral bias [21,22]. This phenomenon limits the capabilities of fully connected networks and renders them unable to learn solutions with high-frequency components [23–25]. In Ref. [16], the authors showed that this problem was present in PINNs, and in further work [18], it was demonstrated that this spectral bias corresponds to first learning the target function along the eigendirection of the neural tangent kernel with larger eigenvalues. In addition to obtaining a theoretical understanding, authors have also focused on extensions to circumvent these shortcomings. Since PINNs often struggle with problems defined on large, multiscale domains, it is possible to enhance traditional PINNs with domain decomposition to improve learning [26,27]. Another approach to circumvent the spectral bias comes from phase deep neural networks (PhaseDNN) [28] or multiscale deep neural networks (MscaleDNN) [29–31], which transform the learning of high frequency into a problem of learning low frequencies. Other general extensions include, for example, Bayesian PINNs [32], which aim to include uncertainties in the output of the network, an essential element when one wishes to use these methods on equations with no known analytical solution. Other authors have also focused on reducing the computational requirements. As the highest costs come from training a network from scratch for each differential equation with a specific set of initial and boundary conditions, methods such as DeepONets [33–35] or physics-informed neural operator [36] learn operators, allowing them to be used in an even broader class of problems.

An essential aspect of solving a differential equation is satisfying the initial and boundary conditions. Two methods have been developed in the field of PINNs to achieve this. A first approach consists of imposing the requirements through a term in the loss function. This technique means that the learning procedure will prioritize solutions that satisfy the conditions as close as possible. Although straightforward to implement, this approach has a significant drawback: The network does not have to adhere to the requirements strictly, as the term in the loss only incentivizes it to find solutions that satisfy those conditions. In cases where the initial and boundary conditions must be strictly adhered to, it is therefore not possible to use this method. In this paper, we focus on the second approach used in the original work by Lagaris *et al.* [7]. In this case, we encompass the network in a trial function and train the network such that the trial function, not the network itself, satisfies the differential equation. We, therefore, obtain a result that perfectly adheres to the initial conditions and is an approximate solution to the differential equation. The trial function consists of a sum of two parts, where one part is a function $A(t)$ that satisfies the initial condition of the problem, and a second function $\mathcal{C}(t,\mathcal{N})$ suppresses the output of the network at the values of t specified by the initial conditions. In this way, the trial function satisfies, by construction, the initial conditions. The choice of these two functions is not unique, and the structure of the trial function is therefore also an element that can be analyzed in the learning process [37] and has been generalized to problems that are defined on irregular domains [38].

In this paper, we take a step sideways by investigating the problem formulation and its influence on the learning capacity of the network. There exists a bijective relationship between the solution of an ordinary differential equation (ODE) of order n and the solution of a system of n first-order differential equations. Given the same constraints, in the sense of architecture complexity and learning algorithm, we compare the performance of the network trained to solve the higher-order differential equation with the performance of the network trained on the equivalent problem formulated as a system of equations. The structure of this paper is as follows: Sec. II covers both approaches used and the network architectures. Section III describes our results, comparing the performance of both methods for a variety of ODEs. Here, we also show the validity of our method in the case of partial differential equations (PDEs). Finally Sec. IV is devoted to analyzing the results.

II. METHODS

Consider a nth-order ODE:

$$x^{(n)} = F[t, x, x', \ldots, x^{(n-1)}], \quad t \in [a, b], \tag{1}$$

with n initial conditions, which has a unique solution on the interval $[a, b]$ [39]. To solve this differential equation through hard constrained PINNs, we consider a neural network with one input and output node $\mathcal{N}(t,\theta)$ and define the trial function:

$$\hat{x}(t) = A(t) + \mathcal{C}[t, \mathcal{N}(t,\theta)]. \tag{2}$$

The function $\mathcal{C}$ constrains the output of the network and ensures that, at the values of t for which an initial condition is defined, the output of the neural network is suppressed. Specifically, this means $\mathcal{C}(t,\mathcal{N})$ and its derivatives, up to the order of the highest derivative involved in the initial condition, must be zero at the values of t involved in those initial conditions. By choosing a specific form for $A(t)$, we can then guarantee that, at those values of t, the trial function $\hat{x}(t)$ automatically satisfies the initial conditions. To clarify with an example, we consider a second-order equation with the initial condition $x(0) = 1$ and $\dot{x}(0) = 2$. In this case, we could choose

$$\mathcal{C}(t,\mathcal{N}) = t^2\mathcal{N}(t,\theta) \quad \text{and} \quad A(t) = 1 + 2t. \tag{3}$$

We note as well that the choice for these functions is not unique, as the factor multiplying the network in the function $\mathcal{C}(t,\mathcal{N})$ has, in this case, as its only requirements that the function itself and its first derivative must be 0 when $t = 0$. The network is subsequently trained by minimizing the loss

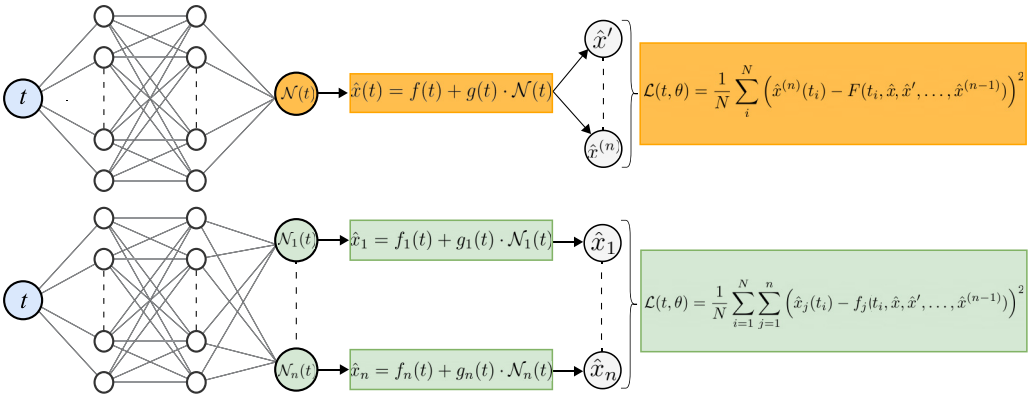

FIG. 1. Above: Architecture and loss used to solve the higher-order formulation of the problem. Below: Architecture and loss used for the system formulation.

function:

$$\mathcal{L}(t,\theta) = \frac{1}{N}\sum_{i}^{N}\{\hat{x}^{(n)}(t_i) - F[t_i, \hat{x}, \hat{x}', \ldots, \hat{x}^{(n-1)}]\}^2, \quad (4)$$

with $t_i \in [a, b]$. As can be seen, it is the trial function, namely, Eq. (2), which enters in the loss function. The choice of $\mathcal{C}(t, \mathcal{N})$ and $A(t)$ therefore play an important role in shaping the loss landscape, which influences the learning capacity of the network.

The preceding paragraph summarizes the traditional approach to solve higher-order differential equations with PINNs. We now introduce our shortcut in phase space. Given the differential equation in Eq. (1), it is always possible to transform it into a system of n first-order ODEs $\mathbf{x}' = \mathbf{f}(t, \mathbf{x})$ by defining $\mathbf{f} = (f_1, f_2, \ldots, f_n)$ as

$$\begin{aligned} f_1[t, x, x', \ldots, x^{(n-1)}] &= x' \\ f_2[t, x, x', \ldots, x^{(n-1)}] &= x'' \\ &\vdots \\ f_n[t, x, x', \ldots, x^{(n-1)}] &= F[t, x, x', \ldots, x^{(n-1)}]. \end{aligned}$$

Interestingly, the approach outlined above can still be used, modulo a few modifications. The neural network now needs n output nodes, denoted as $\mathcal{N}_i(t, \theta)$, and the n trial functions are then constructed as

$$\begin{aligned} \hat{x}_1(t) &= A_1(t) + \mathcal{C}_1[t, \mathcal{N}_1(t, \theta)] \\ \hat{x}_2(t) &= A_2(t) + \mathcal{C}_2[t, \mathcal{N}_2(t, \theta)] \\ &\vdots \\ \hat{x}_n(t) &= A_n(t) + \mathcal{C}_n[t, \mathcal{N}_n(t, \theta)]. \end{aligned}$$

Coupling back to our previous example, one possible choice for the trial functions could be

$$\begin{aligned} \hat{x}_1(t) &= 1 + t\mathcal{N}_1(t, \theta) \\ \hat{x}_2(t) &= 2 + t\mathcal{N}_2(t, \theta). \end{aligned}$$

Since both trial functions only need to encompass one condition, we can choose the constraining functions to be linear: $\mathcal{C}_i(t, \mathcal{N}_i) = t\mathcal{N}_i(t, \theta)$.

The neural network $\mathcal{N}(t, \theta)$ then trains by optimizing the loss function:

$$\mathcal{L}(t,\theta) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n}\{\hat{x}_j(t_i) - f_j[t_i, \hat{x}, \hat{x}', \ldots, \hat{x}^{(n-1)}]\}^2, \quad (5)$$

i.e., the network is instructed to simultaneously optimize each equation of the system. A visual overview of the difference in the two methods is shown in Fig. 1.

For both the higher-order and system approaches, we need to calculate the derivative of the network with respect to its input nodes to compute the loss function. Performing these computations is made possible through algorithms implementing automatic differentiation that efficiently compute these derivatives [40].

III. RESULTS

This section presents the results of the various experiments performed to establish the performance difference between both methods discussed in the previous section. Both methods were evaluated on various differential equations, considering a range of parameters. The first test consists of second-order linear systems with constant coefficients. Gradually, the complexity of the problem is increased by incorporating nonconstant coefficients and nonlinear terms. We also evaluated the methods on two third-order problems. All systems used in this analysis were analytically solvable, allowing us to compare the solutions of the methods to the true solution. The number of training points, namely, 35 equidistant points over the interval, and the training algorithm are equal for all problems.

The performance is quantified by computing, after training, the relative L_2 error evaluated in validation points with respect to the analytical solution. We highlight that we did not fine-tune the hyperparameters to obtain the lowest possible validation. The goal was to compare both approaches given the same constraints. In most cases, we chose an architecture with two hidden layers composed of 20 nodes and trained it during 6000 epochs. For some problems, a more extensive architecture of three hidden layers with 60 nodes each was required and subsequently trained on 12 000 epochs. Finally, hyperparameters such as the optimizer (ADAM [41]), learning rate (0.001), and the activation function (SWISH [42]) were maintained equal for all problems.

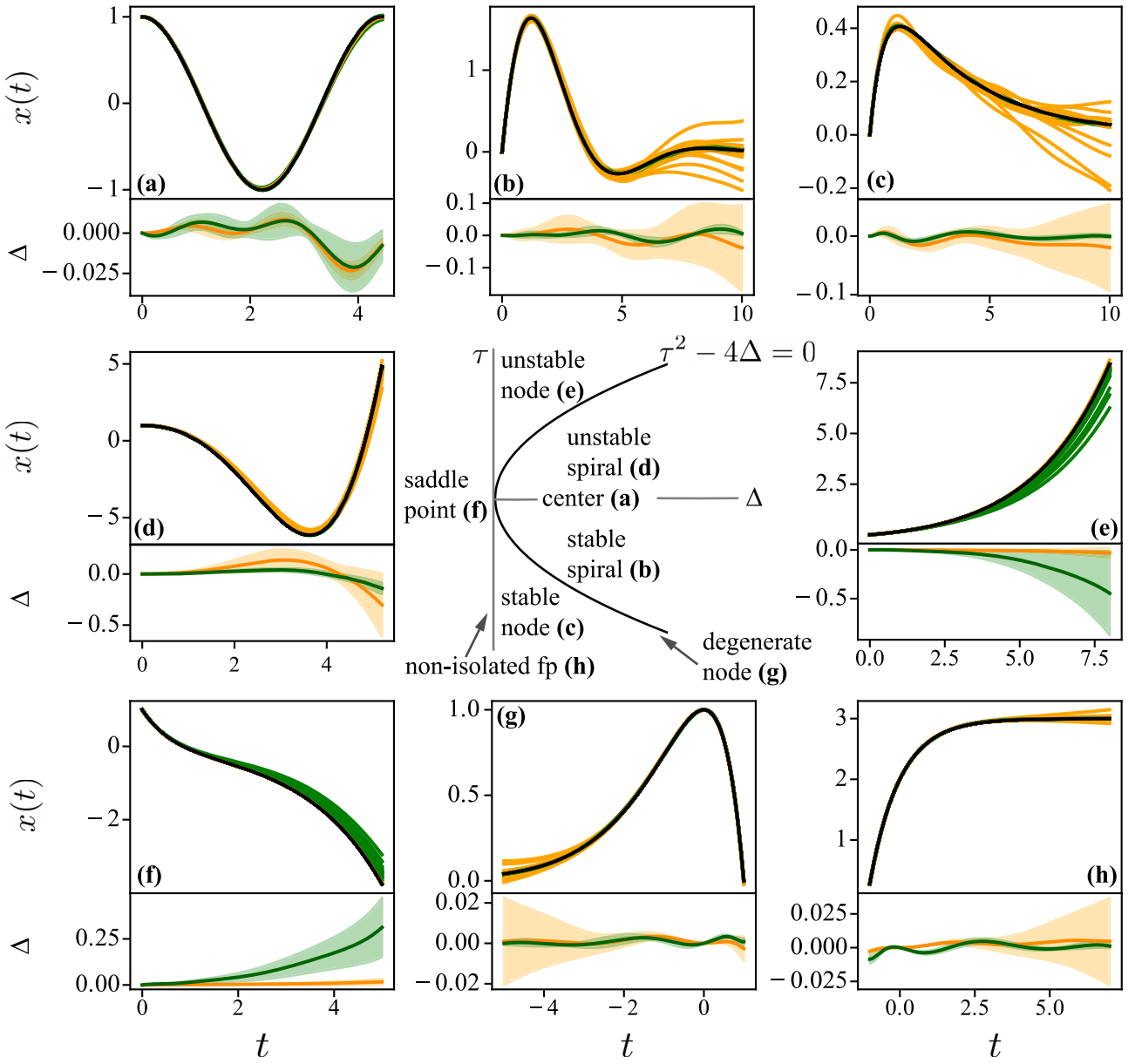

FIG. 2. The 30 solutions of the neural networks, after training for 6000 or 12 000 epochs to solve the higher-order equation (orange) or the system of equations (green), together with the analytical solution are displayed. The lower panel shows the average error at each point (line), while the shaded region indicates one standard deviation. All learning constraints were equal for both approaches, and the initial parameters for each run were randomly initialized. (a) Center: $\ddot{x} + 2x = 0$. (b) Stable spiral: $\ddot{x} + \dot{x} + x = 0$. (c) Stable node: $\ddot{x} + 2\dot{x} + \frac{1}{2}x = 0$. (d) Unstable spiral: $\ddot{x} - \dot{x} + x = 0$. (e) Unstable node: $\ddot{x} - \frac{1}{2}\dot{x} + \frac{1}{20}x = 0$. (f) Saddle point: $\ddot{x} + \dot{x} - x = 0$. (g) Degenerate node: $\ddot{x} - 2\dot{x} + x = 0$. (h) Nonisolated fixed points: $\ddot{x} + \dot{x} = 0$. The central panel shows the classification of constant coefficients, based on the trace (τ) and determinant (Δ), of the matrix $\mathbf{A}$ in Eq. (6).

A. Linear ODEs with constant coefficients

It is widely known that a system of n first-order differential equations is said to be linear if it can be written as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \tag{6}$$

where $\mathbf{A}$ is a $n \times n$ matrix. When the coefficients of $\mathbf{A}$ are constant, it is possible to categorize the phase space of linear systems by the trace (τ) and determinant (Δ) of the matrix $\mathbf{A}$. The fixed points of the system, which are the positions in phase space where $\dot{\mathbf{x}} = 0$, can then be classified into nine distinct types. This categorization can be found in Ref. [43] and is presented here in the center of Fig. 2.

This paper exploits the correspondence between higher-order differential equations and systems of equations. We start from the second-order homogeneous equation $a\ddot{x} + b\dot{x} + cx = 0$, which corresponds to a linear system with a matrix:

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ \tilde{c} & \tilde{b} \end{pmatrix} \quad \text{with} \quad \tilde{c} = -\frac{c}{a}, \quad \tilde{b} = -\frac{b}{a}. \tag{7}$$

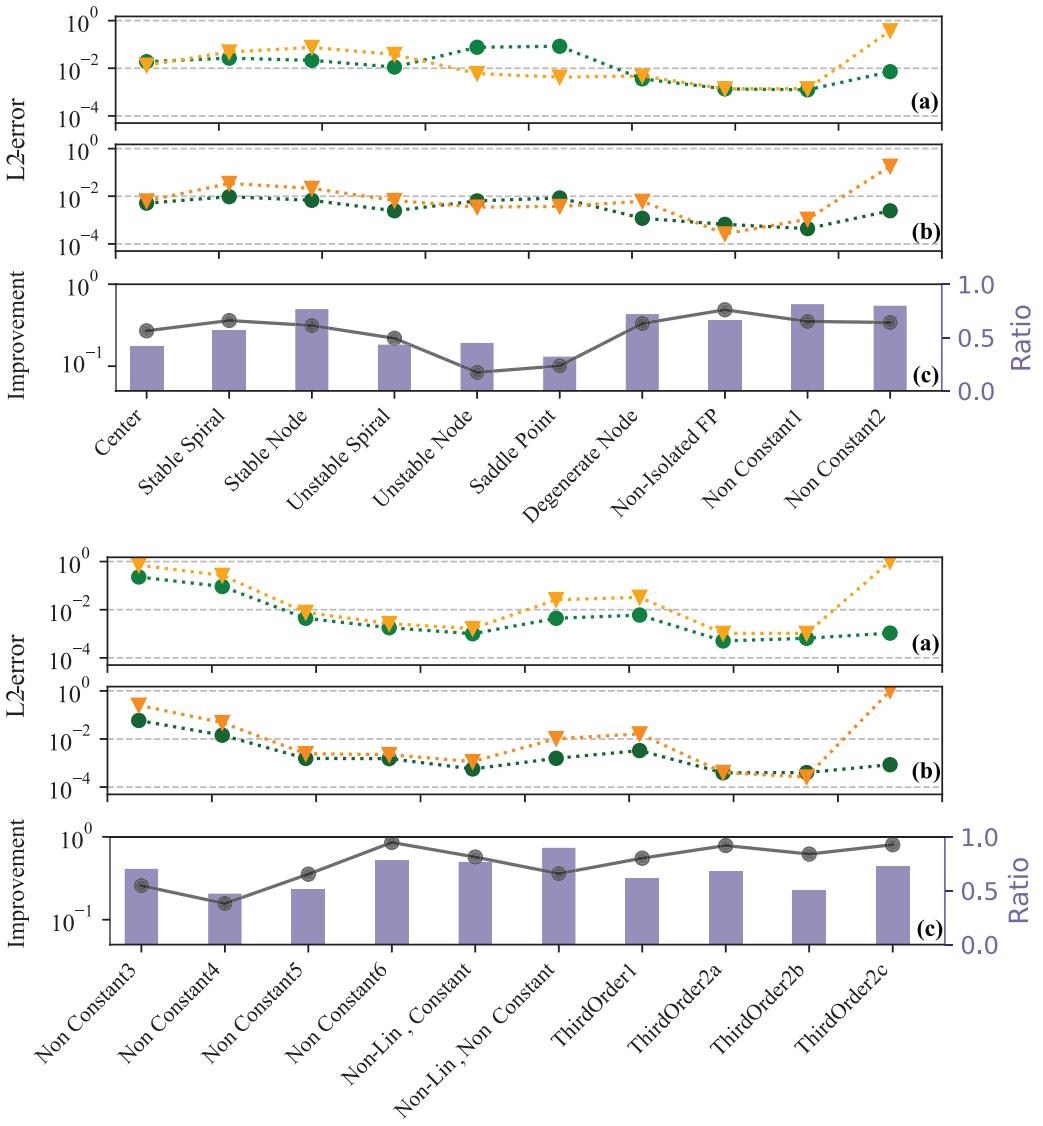

FIG. 3. (a) L_2 error for short training times in log scale. (b) L_2 error for long training times in log scale. (c) Nonconflict ratio R (blue) overlayed with the relative improvement (black) between short and long training times.

Even though the form of the matrix **A** is restricted, it is still general enough to explore the full range of different linear systems.

To assess the performance of the approaches against each other, we solve a linear system with each type of phase space. For each problem, we performed 30 random initializations of the networks, and Fig. 2 displays the final results, after training, of those random initializations for both methods. The lower panel of each figure shows the mean error and one standard deviation at each point. The range on which the differential equations were solved depended on the specific problem, as the value of the analytical solution was kept between predefined bounds. We expect the choice not to influence the results, as both methods received the same problem. The results of all runs can be found in Fig. 2, while Fig. 3 displays the average L_2-error values reached. The results obtained for the differential equation show that the system approach outperformed the higher-order formulation in 6 of the 8 cases while having similar training times. We performed a second run for all eight equations with a training time of 18 000 epochs to examine the dependence on the epoch hyperparameters. Figure 3 shows the resulting L_2 errors. The results indicate that the system approach, in these cases, performs better on shorter training times but can also outperform its counterpart when trained longer. For these longer training times, the system approach also closed the gap for the two equations it struggled with on shorter training times, reaching an L_2 error which was on the same order of magnitude as the higher-order approach. Both behaviors mentioned above are visible in Fig. 4.

B. Linear ODEs with nonconstant coefficients

Linear (inhomogeneous) differential equations with nonconstant coefficients were the second class of equations used to benchmark the two approaches. We performed the

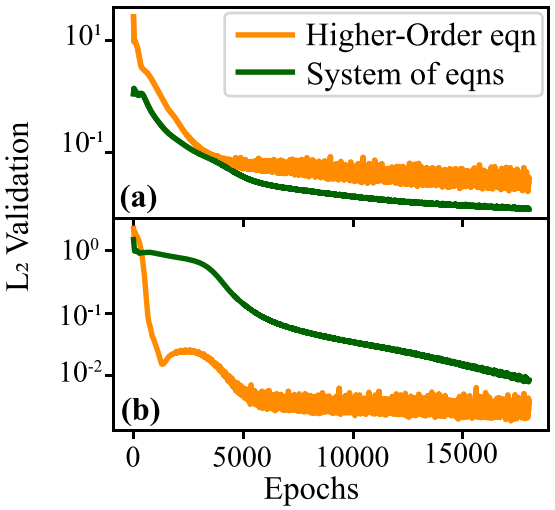

FIG. 4. Examples of the evolution of the L_2 validation for two cases from the equations in Sec. III A. (a) Evolution for the stable spiral. (b) Evolution for the saddle point.

analysis for six second-order differential equations, giving the results in Fig. 5. The choice of differential equations had the goal of including the largest variety in analytical solutions. For equation in Fig. 5(d), an L_2 error of order smaller than $\mathcal{O}(10^{-1})$ was not obtained by either approach in an initial run with the original architecture. To remedy this, we changed the architecture to three hidden layers with each 60 while training for 12 000 epochs to ensure enough training time for this deeper and wider network. We note here that we only did this because both methods did not seem to learn with the original architecture. The system approaches performed better for all six differential equations, even obtaining an L_2 error one order of magnitude smaller for equations in Figs. 5(b), 5(c), and 5(d). The difference in performance favored the system approach even when increasing the training time to 18 000 [or 24 000 for equation in Fig. 5(d)] epochs. We show these results on the second panel of Fig. 3(b).

C. Nonlinear systems and higher orders

We performed the final part of the analysis of both algorithms with nonlinear equations and differential equations of higher order. For each category, we selected two problems. Since the differential equations are more complex, we chose the architecture with three hidden layers of 60 nodes (trained during 12 000 epochs). To obtain additional variety in the higher-order equations, we also ran the second third-order equation with three different combinations of initial conditions. We chose to train during 24 000 epochs for the longer training runs. Figure 6 displays all the solutions obtained by both approaches.

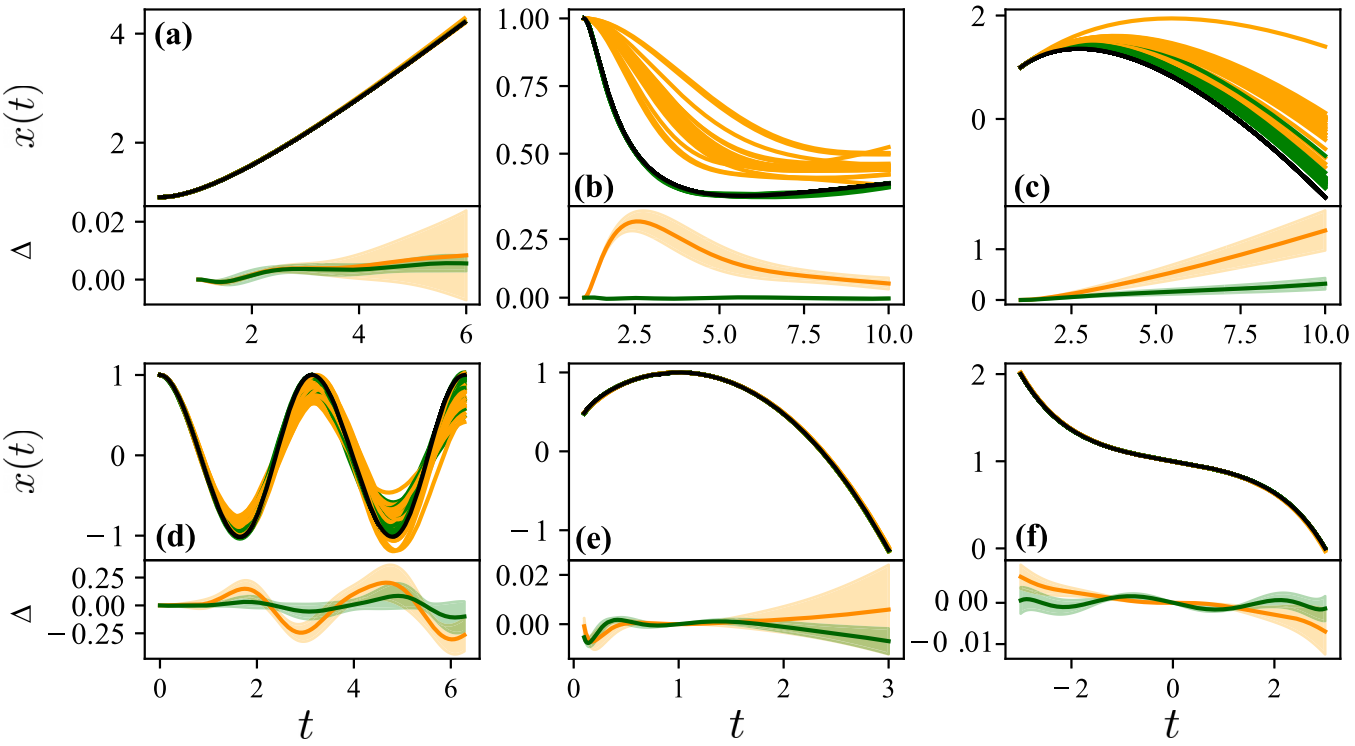

FIG. 5. The 30 solutions of the neural networks, after training for 6000 (or 12 000) epochs to solve the (orange) or the system of equations (green), together with the analytical solution are displayed. The lower panel shows the average error at each point (line), while the shaded region indicates one standard deviation. All learning constraints were equal for both approaches, and the initial parameters for each run were randomly initialized. (a) $t\ddot{x} + \dot{x} = 1$. (b) $t^2\ddot{x} + 5t\dot{x} + 4x = \ln(t)$. (c) $t^2\ddot{x} - t\dot{x} + x = 0$. (d) $\ddot{x} + 4x = \cos(2t)\sin(2t)$. (e) $\ddot{x} - \tanh(t)x = 0$. (f) $t^2\ddot{x} - 2t\dot{x} + x = 0$.

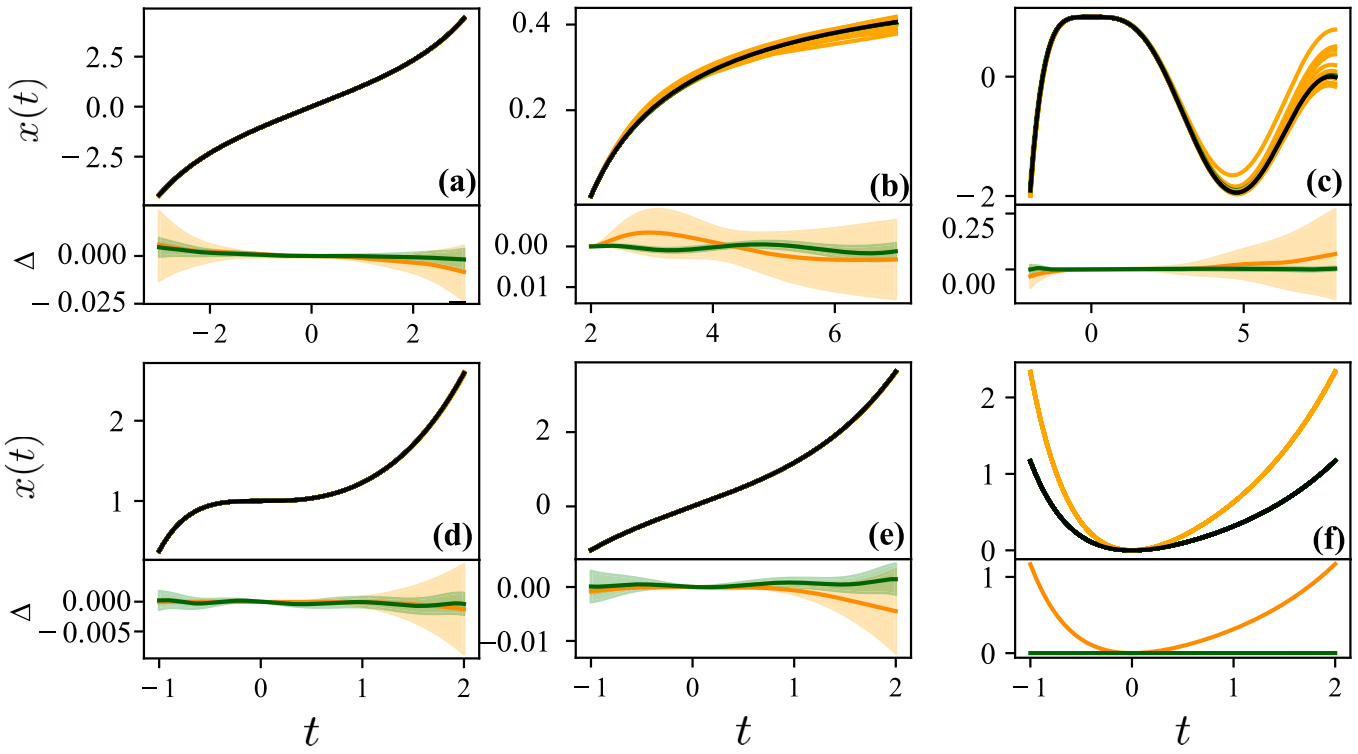

FIG. 6. The 30 solutions of the neural networks, after training for 6000 (or 12 000) epochs to solve the (orange) or the system of equations (green), together with the analytical solution are displayed. The lower panel shows the average error at each point (line), while the shaded region indicates one standard deviation. All learning constraints were equal for both approaches, and the initial parameters for each run were randomly initialized. (a) $\ddot{x} - \frac{1}{5}x\dot{x} = 0$. (b) $\ddot{x} + 2t\dot{x}^2 = 0$. (c) $\dddot{x} + 2\ddot{x} + \dot{x} = -2\sin(t)$. (d)–(f) $\dddot{x} + 2\ddot{x} - \dot{x} - 2x = 0$ (with three different initial conditions).

D. PDEs

The final benchmark of our method was performed with a PDE, namely, the two-dimensional diffusion equation:

$$\frac{\partial y}{\partial t} = \frac{\partial^2 y}{\partial x^2} - e^{-t}\sin(\pi x)(1-\pi^2). \tag{8}$$

We solved this equation for $x \in [-1, 1]$, $t \in [0, 2]$ and with the Dirichlet boundary conditions $y(t, -1) = y(t, 1) = 0$. To transform Eq. (8) into a system of equations, we define $v \doteq \partial_t y$ and $w \doteq \partial_x y$ and obtain

$$\partial_t y = v$$
$$\partial_x y = w$$
$$\partial_x w = v + e^{-t}\sin(\pi x)(1-\pi^2).$$

The training point for the networks was a 200×200 grid. For this problem, we used an architecture composed of four hidden layers with 200 hidden nodes each and trained it during 40 000 epochs.

Our results show that the system approach can outperform the higher-order approach also for PDEs. Figure 7 displays the results obtained from the networks together with the analytical solution, the evolution of the L_2 validation, and the absolute difference between the results of the networks and the analytical solution. We can see that the L_2 validation for the system is on average an order of magnitude lower than the higher-order approach. We further note that the oscillations in the validation were also visible for the higher-order approach in other runs and not specific to the system approach. Although specific conclusions about the performance of the system approach vs the higher-order approach on problems of PDEs require another analysis, our results show that the system approach can successfully be applied to such problems.

IV. DISCUSSION

The results presented seem to show a difference in performance when training a neural network to solve ODEs or PDEs, depending on how one formulates the problem. Our method for analyzing this was by tackling a variety of differential equations.

We found a slight variation in performance for the linear systems with constant coefficients but no discernible pattern dictating which method would perform better based on the type of phase space considered. Equations on which the system approach outperformed the higher-order one still favor the system approach even with longer training times. The linear problems with nonconstant coefficients seemed to benefit the most from using the system approach, as we found that the system method outperformed the higher-order approach each time for those cases. This behavior continued even when training for longer epochs. This behavior was also visible in the higher-order equations. On the other hand, both approaches displayed similar results when solving nonlinear equations. For all problems, both approaches also displayed similar training times.

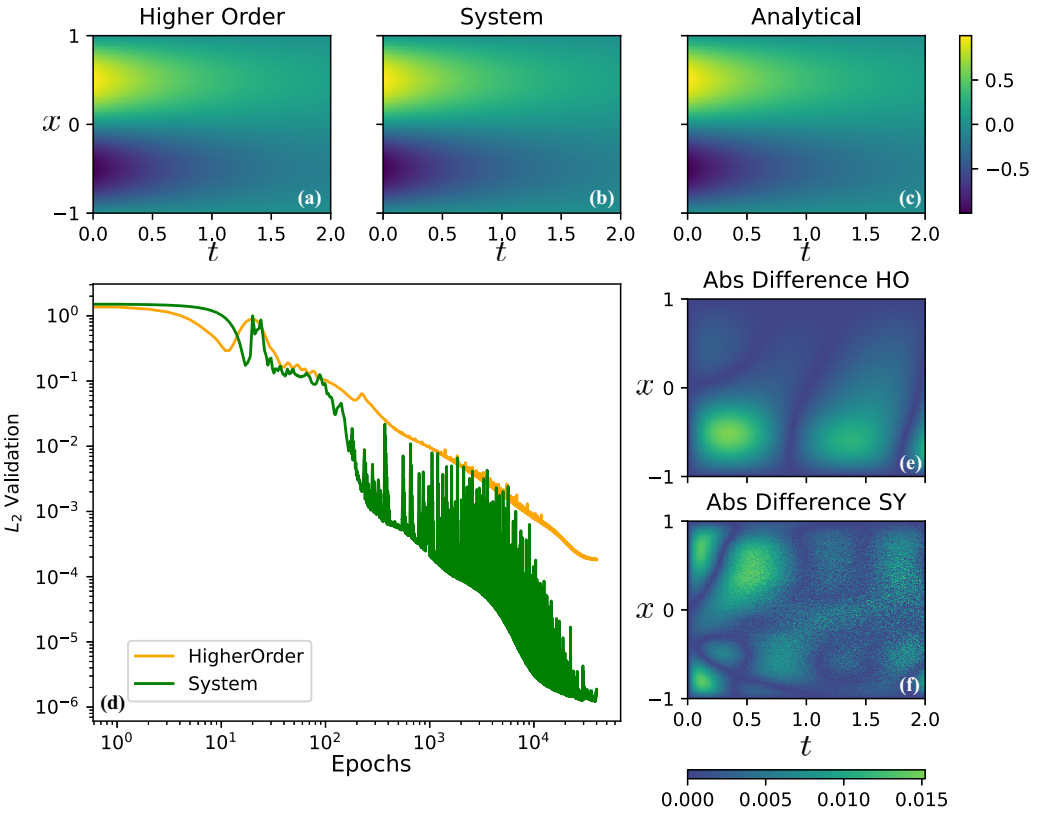

FIG. 7. Solving the equivalent system instead of the higher-order differential equation can also work for partial differential equations (PDEs). The data presented originate from one iteration. (a)–(c) Higher-order, system, and analytical results. (d) L_2 validation evolution for the system (green) and the higher-order (orange) approaches. (e) and (f) Absolute difference between the analytical solution and the results of the networks.

As mentioned in Sec. II, training the network to solve the system requires optimizing multiple tasks at once. We can therefore try to understand the performance of this approach through the lens of multitask learning, especially for those equations where it performed worse than the higher-order method. As mentioned in Ref. [44], problems of negative transfer, where two tasks have conflicting gradients during learning, are the main challenges when training a network on more than one task. We can infer the measure of conflict between two gradients $\mathbf{g}_1$ and $\mathbf{g}_2$ by computing the angle between the two and the similarity in their magnitude [45]:

$$\Phi(\mathbf{g}_1, \mathbf{g}_2) = \frac{2||\mathbf{g}_1||_2||\mathbf{g}_2||_2}{||\mathbf{g}_1||_2^2 + ||\mathbf{g}_2||_2^2}. \tag{9}$$

If during a learning epoch we have a combination of both $\Phi(\mathbf{g}_1, \mathbf{g}_2) \approx 1$ and a large angle between the gradients, then the resulting gradient will be close to zero, resulting in a lack of improvement. To generalize this occurrence of conflict to a network that is learning more than two tasks, we consider the nonconflict ratio R, which we define as

$$R \doteq \frac{\left|\left|\sum_i^n \mathbf{g_i}\right|\right|}{\sum_i^n ||\mathbf{g_i}||}. \tag{10}$$

The closer R is to one, the fewer conflicts were present when summing the gradients of the individual tasks. The mean value of R for the differential equations is displayed in Fig. 3 in the lower panes, together with the relative improvement of the network between short and long training times. The value of this relative improvement shows by what factor the validation at longer training times was reduced. The smaller the value, the more the network improved between the two training times. The correlation between both quantities can indicate that a conflict in gradients is one of the reasons for the cases where the system approach is slower to learn. Such a phenomenon means that gradient modification techniques developed for multitask learning, such as in Refs. [45,46] could further improve the method. These approaches alter the individual gradients to remove the conflicting elements before applying them to the weights, ensuring optimal learning.

Another approach to optimize learning in the multitask context is hard parameter sharing [47]. Here, a neural network is constructed with first a number of hidden layers, shared among the tasks, followed by hidden layers which are task specific. This architecture is shown in the upper panel of Fig. 8. We performed an analysis to see if this method could also improve the results we obtained for the systems approach. For this, we solved the differential equations which were solved previously on the small architecture (with two hidden layers of 20 nodes).

We chose an architecture with a total maximum of four hidden layers and analyzed the performance when replacing one or more common layers by task-specific layers. The common layers had 40 nodes each, while the task-specific layers

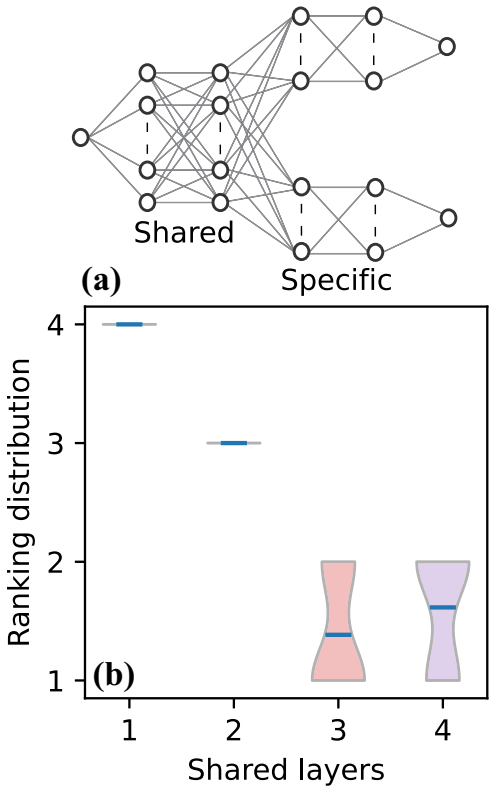

FIG. 8. (a) Architecture with shared and task-specific layers. (b) Distribution of the obtained rankings and the averaged obtained rank (blue line) of the architectures with varying amounts of shared layers when solving all equations with constant coefficients and five equations with nonconstant coefficients. Lower is better.

contained 25 nodes. For each equation, we performed 30 random initializations and subsequently compared the mean L_2 error between the varying architectures and ranked the performance of the architectures from best (rank 1) to worst (rank 4). To finalize the analysis, we averaged the obtained rank for each architecture over all the equations and illustrated the results in Fig. 8. The results obtained in our analysis show that an architecture with one shared layer outperforms the architecture with no shared layers for some problems, indicating that these types of architecture can indeed have a positive effect on learning for the system approach. The question still stands as to why our technique generally works better. Our simulations provide a quantitative argument that the performance increase in solving the system of equations is related to how the errors are handled during training. For example, the standard higher-order setting has only one node to approximate both the function and all its derivatives. This causes errors in the approximation of the function to influence the errors in its derivatives. In contrast, the system of equations approach introduced in this paper optimizes all errors individually, as they appear independently in the loss function, making it easier to change the error of one output without affecting the others.

Another important aspect to consider when choosing one of these methods is the setup of the trial function. Although the functions in the trial function are only there to ensure it satisfies the initial and boundary conditions, these functions impact the learning capacity of the network, as the entire trial function appears in the loss term. In the higher-order approach, these two functions need to encompass all initial conditions, making them often higher-order polynomials. In contrast, the trial functions for the system approach only encompass the conditions imposed on a particular order of the derivative. The simplest choice, in that case, is often a linear function. This is important for two reasons. First, it simplifies the construction of the trial function, making this method more straightforward to use. Second, this allows us to fine-tune the constraining function based on the problem. In Ref. [48], a constraining function $\mathcal{C}(t,\mathcal{N}) = (1-e^{-t})\mathcal{N}(t,\theta)$ is used, as it tends to 1 when t takes on large values. The authors show empirically that this choice of function drastically improves the capacity of the network to learn. Choosing such a function is only possible when the constraints originating from the initial conditions allow it. As the trial functions of the system approach are subject to fewer constraints, it is easier to optimize the choice of $\mathcal{C}(t,\mathcal{N})$.

V. CONCLUSIONS

Neural networks are universal function approximators, which means that, given sufficient hidden nodes, any function can be modeled up to an error ε. This property allows us to use neural networks to learn solutions to differential equations. In this paper, we investigated the influence of the problem formulation on the learning capacity of the network. Our experiments show that, for most problems we ran, neural networks indeed perform better when presented with a differential equation formulated as a system of equations. Our analysis covered a total of 20 ODEs with various characteristics. Additionally, we showed that our methods can also work for PDEs. In this paper, we also looked at the role of multitask learning elements in the system approach. Analyzing our results through this lens indicated the presence of conflicting gradients during training for some of the problems that we tackled. Therefore, future efforts could focus on implementing gradient modification algorithms to ensure that learning occurs optimally in all cases.

Our results also show that future work studying the trial function itself could significantly improve the use of neural networks as differential equation solvers. As the system approach imposes fewer constraints on its trial functions, it is easier to implement functions that benefit learning. Combined, these efforts will also lead to a more comprehensive understanding of PINNs methods, ensuring greater reliability when solving future tasks.

ACKNOWLEDGMENTS

Work at VUB was partially supported by the Research Foundation Flanders under Grants No. G032822N and No. G0K9322N and by the research council of the VUB. This article was published with the support of the University Foundation of Belgium.

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems*, Vol. 25, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., Red Hook, 2012).

[2] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, Highly accurate protein structure prediction with AlphaFold, Nature (London) **596**, 583 (2021).

[3] D. Bourilkov, Machine and deep learning applications in particle physics, Int. J. Mod. Phys. A **34**, 1930019 (2019).

[4] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning, Nat. Biotechnol. **33**, 831 (2015).

[5] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke, Explainable machine learning for scientific insights and discoveries, IEEE Access **8**, 42200 (2020).

[6] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, M. Walczak, J. Pfrommer, A. Pick *et al.*, Informed machine learning—a taxonomy and survey of integrating prior knowledge into learning systems, in *IEEE Transactions on Knowledge and Data Engineering* (IEEE, 2021).

[7] I. E. Lagaris, A. Likas, and D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. **9**, 987 (1997).

[8] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Physics-informed machine learning, Nat. Rev. Phys. **3**, 422 (2021).

[9] A. Mathews, M. Francisquez, J. W. Hughes, D. R. Hatch, B. Zhu, and B. N. Rogers, Uncovering turbulent plasma dynamics via deep learning from partial observations, Phys. Rev. E **104**, 025205 (2021).

[10] M. L. Piscopo, M. Spannowsky, and P. Waite, Solving differential equations with neural networks: applications to the calculation of cosmological phase transitions, Phys. Rev. D **100**, 016002 (2019).

[11] S. Cai, Z. Wang, F. Fuest, Y. J. Jeon, C. Gray, and G. E. Karniadakis, Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks, J. Fluid Mech. **915**, 1 (2021).

[12] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, Neural Networks **2**, 359 (1989).

[13] D. Psichogios and L. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE J. **38**, 1499 (1992).

[14] S. Mishra and R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating PDEs, IMA J. Numer. Anal. **42**, 981 (2022).

[15] Y. Shin, J. Darbon, and G. E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, Commun. Comput. Phys. **28**, 2042 (2020).

[16] S. Wang, X. Yu, and P. Perdikaris, When and why PINNs fail to train: a neural tangent kernel perspective, J. Comput. Phys. **449**, 110768 (2022).

[17] S. Wang, Y. Teng, and P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. **43**, A3055 (2021).

[18] S. Wang, H. Wang, and P. Perdikaris, On the eigenvector bias of Fourier feature networks: from regression to solving multi-scale PDEs with physics-informed neural networks, Comput. Methods Appl. Mech. Eng. **384**, 113938 (2021).

[19] A. Jacot, F. Gabriel, and C. Hongler, Neural tangent kernel: convergence and generalization in neural networks, arXiv:1806.07572 (2018).

[20] G. Yang, Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation, arXiv:1902.04760 (2019).

[21] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, On the spectral bias of neural networks, in *Proceedings of the 36th International Conference on Machine Learning, 9–15 June 2019, Long Beach, California, USA*, Proceedings of Machine Learning Research, Vol. 97, edited by K. Chaudhuri and R. Salakhutdinov (PMLR, 2019), pp. 5301–5310.

[22] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, and Q. Gu, Towards understanding the spectral bias of deep learning, in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence Montreal, 19–27 August 2021*, edited by Z.-H. Zhou (IJCAI, 2021), pp. 2205–2211, main Track.

[23] O. Fuks and H. A. Tchelepi, Limitations of physics informed machine learning for nonlinear two-phase transport in porous media, J. Mach. Learn. Model. Comput. **1**, 19 (2020).

[24] M. Raissi, Deep hidden physics models: deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. **19**, 1 (2018).

[25] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. **394**, 56 (2019).

[26] A. D. Jagtap and G. Em Karniadakis, Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. **28**, 2002 (2020).

[27] B. Moseley, A. Markham, and T. Nissen-Meyer, Finite basis physics-informed neural networks (FBPINNs): A scalable domain decomposition approach for solving differential equations, arXiv:2107.07871 (2021).

[28] W. Cai, X. Li, and L. Liu, A phase shift deep neural network for high frequency approximation and wave problems, SIAM J. Sci. Comput. **42**, A3285 (2020).

[29] B. Wang, Multi-scale deep neural network (MscaleDNN) methods for oscillatory Stokes flows in complex domains, Commun. Comput. Phys. **28**, 2139 (2020).

[30] Z. Liu, W. Cai, and Z.-Q. J. Xu, Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains, Commun. Comput. Phys. **28**, 1970 (2020).

[31] X.-A. Li, A multi-scale DNN algorithm for nonlinear elliptic equations with multiple scales, Commun. Comput. Phys. **28**, 1886 (2020).

[32] L. Yang, X. Meng, and G. E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, J. Comput. Phys. **425**, 109913 (2021).

[33] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. **3**, 218 (2021).

[34] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data, Comput. Methods Appl. Mech. Eng. **393**, 114778 (2022).

[35] S. Wang, H. Wang, and P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. **7**, eabi8605 (2021).

[36] Z. Li, H. Zheng, N. Kovachki, D. Jin, H. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar, Physics-informed neural operator for learning partial differential equations, arXiv:2111.03794 (2021).

[37] C. Leake and D. Mortari, Deep theory of functional connections: A new method for estimating the solutions of partial differential equations, Mach. Learn. Knowl. Extr. **2**, 37 (2020).

[38] J. Berg and K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing **317**, 28 (2018).

[39] L. Perko, *Differential Equations and Dynamical Systems*, 3rd ed. (Springer, New York, 2002), pp. 74–77.

[40] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, Automatic differentiation in machine learning: a survey, J. Mach. Learn. Res. **18**, 1 (2018).

[41] D. P. Kingma and J. Ba, ADAM: a method for stochastic optimization, arXiv:1412.6980 (2014).

[42] P. Ramachandran, B. Zoph, and Q. V. Le, Searching for activation functions, arXiv:1710.05941 (2017).

[43] S. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering* (CRC Press, Boca Raton, 2018), p. 138.

[44] M. Crawshaw, Multi-task learning with deep neural networks: a survey, arXiv:2009.09796 (2020).

[45] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, Gradient surgery for multi-task learning, in *Advances in Neural Information Processing Systems 33*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Curran Associates, Inc., Red Hook, 2020), pp. 5824–5836.

[46] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, Conflict-averse gradient descent for multi-task learning, in *Advances in Neural Information Processing Systems 34*, edited by M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Curran Associates, Inc., Red Hook, 2021), pp. 18878–18890.

[47] R. Caruana, Multitask learning: a knowledge-based source of inductive bias, in *Proceedings of the Tenth International Conference on Machine Learning* (Morgan Kaufmann Publishers Inc., San Francisco, 1993), pp. 41–48.

[48] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas, Hamiltonian neural networks for solving equations of motion, Phys. Rev. E **105**, 065305 (2022).