One-Shot Transfer Learning of Physics-Informed Neural Networks

Shaan Desai [1 2] **Marios Mattheakis** [2] **Hayden Joy** [2] **Pavlos Protopapas** [2] **Stephen Roberts** [1]

Abstract

Solving differential equations efficiently and accurately sits at the heart of progress in many areas of scientific research, from classical dynamical systems to quantum mechanics. There is a surge of interest in using Physics-Informed Neural Networks (PINNs) to tackle such problems as they provide numerous benefits over traditional numerical approaches. Despite their potential benefits for solving differential equations, transfer learning has been under explored. In this study, we present a general framework for transfer learning PINNs that results in one-shot inference for linear systems of both ordinary and partial differential equations. This means that highly accurate solutions to many unknown differential equations can be obtained instantaneously without retraining an entire network. We demonstrate the efficacy of the proposed deep learning approach by solving several real-world problems, such as first- and second-order linear ordinary equations, the Poisson equation, and the time-dependent Schrödinger complex-value partial differential equation.

1. Introduction

Differential equations are used to study a broad array of phenomena, from infection models in biology (Kaxiras et al., 2020) to chaotic motion in physics (Choudhary et al., 2019). As such, our ability to efficiently and accurately solve these equations under various conditions remains a critical challenge in the scientific community. While traditional approaches such as Runge-Kutta (Dormand et al., 1987) and Finite Element Methods are well studied and provide solutions of high fidelity, recently, Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019; Karniadakis et al., 2021) have attracted significant attention as an alternative framework for solving differential equations. PINNs are Neural Networks (NNs) capable of leveraging data and known physical constraints to identify solutions to differential equations. They have numerous advantages over traditional approaches such as being able to easily incorporate data, eliminating the need for a numerical integrator, being able to generate continuous and differentiable solutions, improving accuracy in high dimensions, and maintaining memory efficiency (Karniadakis et al., 2021). However, one of the limitations of using PINNs is the computational expense associated with training networks for different but closely linked tasks. To address this, we explore how transfer learning - the method of training a network on a task and transferring it to new tasks, can be be used to overcome this bottleneck.

Specifically, we show that a PINN, pre-trained on a family of differential equations, can be effectively re-used to solve new differential equations. By freezing the hidden layers of a pre-trained PINN, we demonstrate that solving new differential equations of the same family reduces to optimizing/fine-tuning a linear layer. Furthermore, we specifically show that in the special case of linear systems of differential equations, this optimization is equivalent to solving the normal equations for a latent space of learnt functions. This implies that the optimal linear weights needed to satisfy a new differential equation can be computed in one-shot with the computational cost of a matrix inversion. This therefore entirely eliminates the need for further training/fine-tuning, dramatically reducing the training overhead by orders of magnitude while maintaining high fidelity solutions. We investigate the efficiency of this approach by solving several ordinary differential equations (ODEs) as well as partial differential equations (PDEs) of practical interest. For many systems, we are able to identify highly accurate solutions to unseen differential equations in a fraction of the time needed to train the equations from scratch.

2. Background

The general form of an explicit n^{th} order ODE can be written as:

$$F(t, \psi, \psi^{(1)}, ..., \psi^{(n-1)}) = \psi^{(n)}, \tag{1}$$

where $\psi^{(i)} = \frac{d^i \psi}{dt^i}$ is the ith derivative of the solution $\psi(t)$ with respect to the independent time variable t. Non-homogeneous linear ODEs, a subclass of the general form

[1]Machine Learning Research Group, University of Oxford, Oxford, United Kingdom [2]School of Engineering and Applied Science, Harvard University, Cambridge, MA, U.S.A.. Correspondence to: Shaan Desai <shaandesai@live.com>.

2nd AI4Science Workshop at the 39th International Conference on Machine Learning (ICML), 2022. Copyright 2022 by the author(s).

of Eqn. 1, can be represented as follows:

$$\hat{D}_n\psi = f(t); \quad \hat{D}_n\psi = \sum_{i=0}^{n} a_i(t)\psi^{(i)}, \tag{2}$$

where n denotes the order of the ODE, $f(t)$ is considered a forcing (or control) term that influences the homogeneity of the solution, and $a_i(t)$ is a time dependent coefficient for each derivative.

Traditionally, when an ODE is known *apriori*, the differential equation can be solved using integrators (such as Runge-Kutta) for given initial conditions (ICs). Recently, it has been shown that neural networks can be used to efficiently determine accurate solutions to such problems. One such approach uses PINNs. PINNs use a neural network, with weights parametrized by θ, to transform an input t to output solutions $\psi_\theta(t)$. Then, by leveraging backpropagation and autograd (Maclaurin et al.), exact derivatives of the network output can be computed with respect to the input $\frac{\partial\psi}{\partial t}$. Therefore, given ICs $u_{\text{ic}} = [\psi_0, \psi_0^{(1)}, .., \psi_0^{(n-1)}]^T$, and known differential operator $\hat{D}_n$ and force $f(t)$, the loss function of a PINN is defined as:

$$\mathcal{L} = (\hat{D}_n\psi_\theta(t) - f(t))^2 + (\bar{D}_0\psi_\theta(t) - \psi_{\text{ic}})^2, \tag{3}$$

where $\bar{D}_0\psi = [\psi(0), \psi^{(1)}(0), ..., \psi^{(n-1)}(0)]^T$. The first term enforces the differential equation and the second enforces the initial conditions.

Indeed, such a loss can be enforced for other architectures such as NeuralODE (Chen et al., 2018) and Reservoir Computing (Mattheakis et al., 2021) which exploit recurrent neural networks. However, such networks are not as easily adaptable to PDEs as PINNs are. Many well known PDEs such as the diffusion, wave as well as Schrödinger equation, can be modeled using PINNs. With PDEs, additional variables are used as inputs $[t, x, y, ...]$, so the output is a function of these inputs $\psi(t, x, y, ..)$ subject to certain ICs and boundary conditions (BCs). As such, a loss function similar to Eqn. 3 can be defined for PDEs.

The benefits of using a NN architecture to solve differential equations (both ODEs and PDEs) over traditional methods (such as Runge-Kutta or Finite Elements) include rapid inference, elimination of the curse of dimensionality (Han et al., 2018), no accumulation of errors as would be found with integrators, continuously differentiable solutions, and low memory cost (Karniadakis et al., 2021). While these benefits have been extensively explored across a range of applications (Sirignano & Spiliopoulos, 2018; Mattheakis et al., 2022; Zhai & Hu, 2021; Wang et al., 2021; McClenny & Braga-Neto, 2020; de Wolff et al., 2021), a limited study exists on how transfer learning techniques can be used (Guo et al., 2021; Wang et al., 2021; Mattheakis et al., 2021). Transfer learning was first developed to accelerate network optimization in computer vision where large datasets are costly to re-train on for specific tasks. The main idea behind the approach is to train a neural network on a large corpus of data and then to freeze the network and re-use some of the layers for new, unseen tasks. This was shown to dramatically reduce training time while maintaining high fidelity solutions in vision. With this in mind, we decided to apply transfer learning to PINNs to significantly accelerate the training on new equations. In doing so, we identify a novel one-shot transfer learning framework for systems of linear differential equations that significantly speeds up inference on unseen linear systems.

3. Related Work

Constraining neural networks to learn solutions to differential equations was first introduced by Lagaris et al (1998). The authors showed that partial derivatives of a neural network output with respect to its inputs can be analytically computed when the architecture of the network is known. Therefore, given the solution and its derivatives, it is possible to simultaneously enforce the underlying differential equation as well as the ICs and BCs. Indeed this approach forms the basis of PINNs (Raissi et al., 2019; Karniadakis et al., 2021) where the analytic derivatives from Lagaris et al (1998) are replaced with backpropagation, i.e. replacing the need for an analytic derivation of the partial derivatives. PINNs have since been extensively used across many applications including non-linear structures (Zhang et al., 2020), fluid flow on large domains (Wang et al., 2021), moving boundaries (Wang & Perdikaris, 2021), inferring micro bubble dynamics (Zhai & Hu, 2021), cardiac activation mapping (Sahli Costabal et al., 2020), ocean modelling (de Wolff et al., 2021), bundle solvers (Flamant et al., 2020) and stochastic and high-dimensional PDEs (Karniadakis et al., 2021; Yang et al., 2018; Sirignano & Spiliopoulos, 2018).

Recently, this technique has been extensively used to learn underlying dynamics from data - a concept first proposed by Howse et.al (1996). For example, numerous works show that energy conserving trajectories can be effectively learnt from data by enforcing known energy constraints such as Hamiltonians (Greydanus et al., 2019; Sanchez-Gonzalez et al., 2019), Lagrangians (Cranmer et al., 2020) and variational integrators (Saemundsson et al., 2020; Desai et al., 2021b) into networks. Extensions in this direction have pushed the envelope to also learn non-conservative/irreversible systems (Yin et al., 2021; Desai et al., 2021a; Zhong et al., 2020; Lee et al., 2021) and contact dynamics from sparse, noisy data (Hochlehnert et al., 2021).

Other alternative methods to solve equations such as the fourier neural operator (Li et al., 2020) have also emerged.

These networks highlight how operator regression coupled with fourier transforms can help us identify the underlying dynamics of physical systems from data. To increase the pace of innovation across these methods, several software packages have been developed that use neural networks and the backpropagation technique of PINNs to approximate solutions of differential equations such as NeuroDiffEq (Chen et al., 2020), DeepXDE (Lu et al., 2021), and SimNet (Hennigh et al., 2020).

In spite of these developments, transfer learning remains under explored. Wang et.al (2021) show transfer learning methods can be used to stitch solutions together to resolve a large domain. Yet further work by Mattheakis et.al (2021) illustrates how a reservoir of weights can be transferred to new ICs. Here, we push these further and identify a general model-agnostic method to do one-shot inference for systems of linear ordinary and partial differential equations.

4. Method

4.1. ODEs

We define a neural network such that the approximate network solution $\psi(t)$ at time points t is: $\psi(t) = H(t)_{\theta_H} W_{\theta_W} + B_{\theta_B}$. In other words, the neural network, parametrized by $\theta = [\theta_H, \theta_W, \theta_B]$, transforms the inputs $t \in \mathbb{R}^{t\times 1}$ into a high dimensional, non-linear latent space $H \in \mathbb{R}^{t\times h}$ through a composition of non-linear activations and hidden layers. Then, a linear combination of the latent space is taken, akin to reservoir computing (Jaeger & Haas, 2004), to obtain the solution $\psi(t)$.

To train the network, we design the final weights layer to consist of multiple outputs, i.e. $W_{\theta_W} \in \mathbb{R}^{h\times q}$. This is done so that multiple (q) solutions, $\psi(t) \in \mathbb{R}^{t\times q}$, can be estimated and simultaneously trained to satisfy equations that have different linear operators D_n defined by different coefficients $a_i(t)$, as well as different initial conditions ψ_{ic}, and forces f. Bundle training allows us to (1) integrate the training into a single network and (2) to encourage the hidden states $H(t)$ to be versatile across equations.

At inference, the weights for the hidden layers are frozen and H is computed at specific time points $\hat{t}$. The solution is therefore $\psi(\hat{t}) = H(\hat{t})W_{\text{out}}$ where W_{out} is trainable. For a new set of ICs ψ'_{ic}, source f', and differential operator $\hat{D}'_n$ the loss of the linear ODE (Eqn. 3), becomes:

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\text{diffeq}} + \mathcal{L}_{\text{IC}} \\ &= \left(\hat{D}'_n H W_{\text{out}} - f'(t)\right)^2 + \left(\bar{D}_0 H W_{\text{out}} - \psi'_{\text{ic}}\right)^2 \end{aligned} \quad (4)$$

Since Eqn. 4 is convex, the fine-tuning of W_{out} can be computed analytically. In other words, to minimize $\mathcal{L}$ we need to solve the equation $\partial\mathcal{L}/\partial W_{\text{out}} = 0$. The derivative of the first term of Eqn. 4 is:

$$\frac{\partial \mathcal{L}_{\text{diffeq}}}{\partial W_{\text{out}}} = 2\left(\hat{D}'_n H\right)^T \left(\hat{D}'_n H W_{\text{out}} - f'(t)\right). \quad (5)$$

Taking the same approach for the second term of Eqn. 4 that enforces ICs, we obtain:

$$\frac{\partial \mathcal{L}_{\text{ICs}}}{\partial W_{\text{out}}} = 2(\bar{D}_0 H)^T (\bar{D}_0 H W_{\text{out}} - \psi'_{\text{ic}}). \quad (6)$$

We let $\hat{D}'_n H = \hat{D}_H$ and $\bar{D}_0 H = \bar{D}_H$ to simplify the notation. Adding the loss terms together and setting them to zero yields the optimal output weights:

$$W_{\text{out}} = \left(\hat{D}_H^T \hat{D}_H + \bar{D}_H^T \bar{D}_H\right)^{-1} \left(D_H^T f'(t) + \bar{D}_H^T \psi'_{\text{ic}}\right). \quad (7)$$

Therefore, given any fixed hidden states $H(\hat{t})$ at fixed time-points $\hat{t}$, one can analytically compute a W_{out} for any linear differential equation that minimizes 4. Broadly, we can think of H as being a collection of non-orthogonal basis functions that can be linearly combined to determine the output function.

Note that one special outcome of this formalism is that the matrix inversion at inference is independent of the ICs ψ'_{ic} and force f', which means for any new ICs or f', W_{out} can be computed with a simple matrix multiplication if the inverse term in Eqn. 7 is pre-computed. The benefits of this approach are multi-fold, given H we achieve fast inference (order of seconds for 1000s of differential equations), eliminate the need for gradient-based optimization as no further training is required, and maintain high accuracy if H is well-trained. Indeed this approach relies on determining an inverse matrix. If $D_H^T D_H$ has a large condition number, the matrix will have many eigenvalues close to zero - indicating ill-conditioning. Experimentally, we circumvent this issue by using regularisation or QR decomposition (see Appendix).

We have shown that an analytic W_{out} can be determined for linear non-homogeneous ODEs. However, the proposed network design can still be used, as we show later, for efficient transfer learning of non-linear ODEs.

4.2. PDEs

An important outcome of the formalism for ODEs is a natural extension to linear PDEs. Many PDEs that appear in real-world problems are linear, including the diffusion equation, Laplace equation, the wave equation as well as the time-dependent Schrödinger PDE. Considering one spatial dimension x and one time dimension t, a general linear second order differential equation takes the form:

$$\left(D^t + D^x + D^{xt} + V(t,x)\right)\psi(x,t) = f(x,t), \quad (8)$$

Differential Equation	# Training Bundles	# Test Bundles	Test Time (s)	Test Accuracy (MSE)
First-order linear ODEs	10	1000	7.4×10^{-3}	$1.35 \pm 1.65 \times 10^{-10}$
Second-order linear ODEs	10	1000	3.4×10^{-3}	$2.84 \pm 1.87 \times 10^{-9}$
Coupled linear oscillators	10	100	4.7×10^{-2}	$2.29 \pm 4.74 \times 10^{-12}$
Nonlinear oscillator	5	30	5.2	$1.47 \pm 3.88 \times 10^{-4}$
Poisson	4	100	33.2	$3.60 \pm 8.84 \times 10^{-5}$
Schrödinger	3	400	19.4	$5.02 \pm 8.92 \times 10^{-5}$

Table 1. Summary results of our method on all the systems investigated. Training on a few bundles is sufficient to rapidly and accurately scale to many unseen conditions. Note that the nonlinear oscillator is optimized using gradient descent whereas the other methods are all optimized using analytic W_{out}. For reference, training a PINN requires several thousand iterations to obtain accurate solutions, where a single iteration costs 0.07s. All times are reported for a CPU.

where we denote a second order time operator $D^t\psi = \sum_{i=1}^{2} a_i(t,x)\psi_t^{(i)}$, the spatial second-order operator $D^x\psi = \sum_{i=1}^{2} b_i(t,x)\psi_x^{(i)}$, and a mixed space-time operator, $D^{xt}\psi = D^{tx}\psi = c(x,t)\psi_{xt}$. The coefficients a, b, c and commonly called source and potential f, V functions, respectively, are continuous functions of x, t, where the lower indices indicate partial derivatives according to the notation: $\psi_\nu^{(i)} = \frac{\partial^{(i)}\psi}{\partial\nu^{(i)}}$ and $\psi_{\nu\nu'} = \frac{\partial^2\psi}{\partial\nu\nu'}$. The structure of Eqn. 8 can generalize to higher orders and for more variables.

The last part to complete the derivation is to enforce the BCs and ICs in the loss function. For the purpose of the derivation, we use Dirichlet BCs. Thus,

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\text{diffeq}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BCs}} \\ &= \left(\hat{D}\psi - f(t,x)\right)^2 + \left(\psi(0,x) - g(x)\right)^2 \\ &\quad + \sum_{\mu=L,R} \left(\psi(t,\mu) - B_\mu(t)\right)^2, \end{aligned} \tag{9}$$

where $\hat{D} = (D^t + D^x + D^{xt}) + V(t,x)$, $B_L(t)$ and $B_R(t)$ are the left and right boundary conditions, and $g(x)$ is the initial condition at $t = 0$. Similarly to the derivation for ODEs, we analytically compute W_{out} of Eqn. 9, namely we solve the equation $\partial\mathcal{L}/\partial W_{\text{out}} = 0$ considering a neural solution of the form $\psi = HW_{\text{out}}$. Starting with the first term of Eqn. 9, we read:

$$\frac{\partial\mathcal{L}_{\text{diffeq}}}{\partial W_{\text{out}}} = 2\hat{D}_H^T(\hat{D}_H W_{\text{out}} - f(t,x)) \tag{10}$$

where $\hat{D}_H = \hat{D}H$. Accordingly, for the IC loss component we obtain:

$$\frac{\partial\mathcal{L}_{\text{IC}}}{\partial W_{\text{out}}} = 2H_0^T(H_0 W_{\text{out}} - g(0,x)), \tag{11}$$

with $H_0 = H(0,x)$. For the BCs loss components we have:

$$\frac{\partial\mathcal{L}_{\text{BCs}}}{\partial W_{\text{out}}} = \sum_{\mu=L,R} 2H_\mu^T\left(H_\mu W_{\text{out}} - B_\mu(t)\right), \tag{12}$$

where $H_\mu = H(t,\mu)$. Piecing this all together yields:

$$W_{\text{out}} = \left(\hat{D}_H^T\hat{D}_H + \sum_{\mu=0,L,R} H_\mu^T H_\mu\right)^{-1} \left(\hat{D}_H^T f(t,x) + \sum_{\mu=0,L,R} H_\mu^T Q_\mu(t,x)\right), \tag{13}$$

where $Q_0 = g(x)$, $Q_L = B_L(t)$, and $Q_R = B_R(t)$.

We therefore show it is equally feasible to obtain an analytic set of linear weights to determine solutions to PDEs. Indeed, the accuracy of the solution depends heavily on how well the hidden states H span the solution space. To encourage a representative hidden space, we typically bundle train a single network on different equations, namely a network with multiple outputs.

5. Results

We investigate our method on numerous well known differential equations of practical interest. We present a summary of our results in Table 1. The full training regime, network design and test conditions can be found in the appendix. For ODEs we report accuracy as the MSE of the residual: $|\hat{D}_n\phi_\theta(t) - f(t)|^2$. For PDEs we report accuracy as the MSE between the predicted solution and the analytic solution: $|\psi_{gt} - \psi_{pred}|^2$.

In addition, we highlight how our solver can be used to tackle specific challenges with learning from these equations.

5.1. ODEs

5.1.1. Linear ODEs

To test the performance of our approach, we train both first- and second-order methods of linear non-homogeneous differential equations. For first order ODEs, the equation is defined by the operator of Eqn. 2 with $n = 1$ and by speci-

fying three quantities: the time-dependent coefficients a_0, the forces f, and the ICs. As such, any first-order linear non-homogeneous ODE can be defined by the tuple (a_0, f, ICs). Given a pre-defined list of options for each quantity in the tuple (see Appendix), we randomly sample 10 tuples for training. We batch train our model on all the equations simultaneously (i.e. $W_{\text{out}} \in \mathbb{R}^{t\times 10}$). We then carry out inference on 1000 randomly sampled test tuples using analytic W_{out} from Eqn. 7. Results are presented in Table 1. It is clear that for first-order differential equations, transfer learning analytic W_{out} is significantly advantageous since we obtain high-fidelity solutions. As such, we take a similar approach for second order differential equations described by the operator of Eqn. 2 for $n = 2$. We plot the results of 20 ODEs from the test set and compute their residuals in Fig. 1 where $\dot{\psi} = d\psi/dt$. Indeed, the overall test accuracy depends on how well the hidden states span the space of differential equations. We typically find that more training bundles results in better test performance (see Appendix).

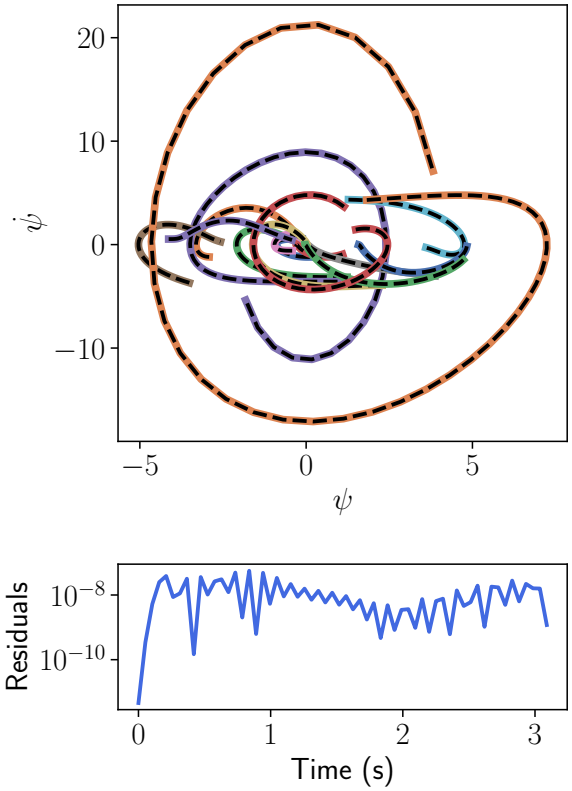

Figure 1. Predicted (colored) versus ground truth (dashed black) phase space, namely a plot of space against velocity for different times, to 20 second-order non-homogeneous ordinary differential equations. Average residuals are shown in the bottom panel.

5.1.2. Systems of ODEs

Since an analytic W_{out} can be computed for linear ODEs, the method naturally extends to systems of linear differential equations (see Appendix for derivation). To highlight this, we investigate a system of linear second-order ODEs of the form $\ddot{\boldsymbol{\psi}} = A\boldsymbol{\psi}$ that describe a system of two coupled oscillators where $\boldsymbol{\psi} = [\psi_1, \psi_2]$ and A describes the coupling. The equation is of the form:

$$\begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix} \begin{bmatrix} \ddot{\psi}_1 \\ \ddot{\psi}_2 \end{bmatrix} = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_1 + k_2 \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \end{bmatrix}. \tag{14}$$

We train the network to satisfy 10 different $\{m, k_1, k_2\}$ values and initial conditions $[\boldsymbol{\psi}_0, \dot{\boldsymbol{\psi}}_0]$ (see Appendix for sampling details) such that the pre-trained network can be used to instantaneously compute accurate solutions for different ICs of the coupled masses. We report the result of testing 100 different systems sampled from the same range as training in Table 1. Furthermore, we investigate an interesting application in which the network can be exploited to identify initial conditions of a coupled-oscillator system capable of inducing *beats* - when two normal mode frequencies come close (see Fig. 2) (Schwartz, 2017).

5.1.3. Nonlinear ODE

Until this point, we have only investigated the success of fine-tuning W_{out} analytically for linear ODEs, nevertheless our network proposal can still be exploited to transfer learn nonlinear ODEs. To do so, we replace the computation of analytic W_{out} with gradient-based optimization. Note, since the hidden weights are frozen, the final hidden activations can be pre-computed given sampling points t for efficient optimization. We use this approach to solve a Hamiltonian nonlinear system described by the ODE:

$$\ddot{\psi} = -\psi - \psi^3, \tag{15}$$

that conserves energy given by the Hamiltonian:

$$\mathcal{H} = \frac{\dot{\psi}^2}{2} + \frac{\psi^2}{2} + \frac{\psi^4}{4}. \tag{16}$$

We train the system on 5 initial positions ψ_0 randomly sampled in the range [0.5, 2.0] and with initial velocity $\dot{\psi}_0 = 0$. The loss function during training consists of (1) a differential equation loss, (2) an initial condition loss, and (3) an energy conservation loss penalty. The energy loss enforces the Hamiltonian at all points in time to be the same, namely $\mathcal{L}_{\text{E}} = (\mathcal{H}(\psi, \dot{\psi}) - \mathcal{H}(\psi_0, \dot{\psi}_0))^2$ (Mattheakis et al., 2022). We then evaluate the performance of the hidden states on 30 ICs sampled in the same range (see Fig. 3). Since we freeze the hidden layers, we can pre-compute the hidden activations $H(\bar{t})$ at fixed time $\bar{t}$ and then fine-tune W_{out} using gradient descent for 5000 epochs. Note that the optimization can be done using other methods as well, including L-BFGS since the entire problem is reduced to convex optimization.

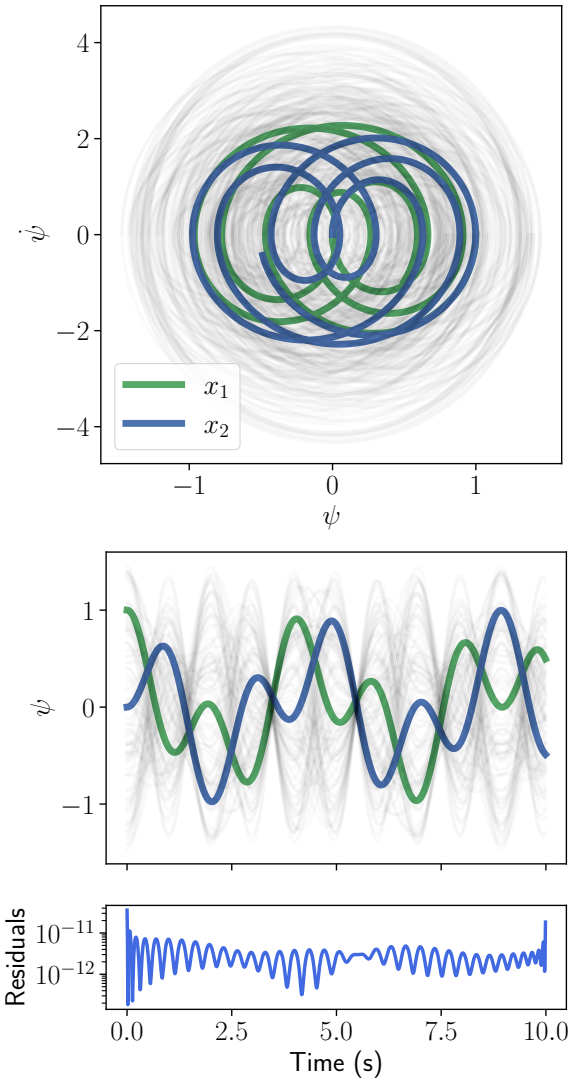

Figure 2. Phase space trajectories of the coupled oscillator system for fixed mass and spring constants (top) and spatial solutions (middle). One solution that induces beats is highlighted in color while the other solutions appear in grey. The average residuals of the total realizations are shown in the bottom panel. The initial state of the masses influences how close the normal mode frequencies get. Our network can identify solutions to all 100 initial conditions in $\sim 10^{-2}$ seconds.

5.2. PDEs

PDEs can be used to model complex spatio-temporal systems making them of practical interest in numerous domains. Typically, the most well-studied PDEs are linear and include the diffusion, Poisson, and wave equations. To benchmark the performance of this approach, we investigate the Poisson equation and the time-dependent Schrödinger equation.

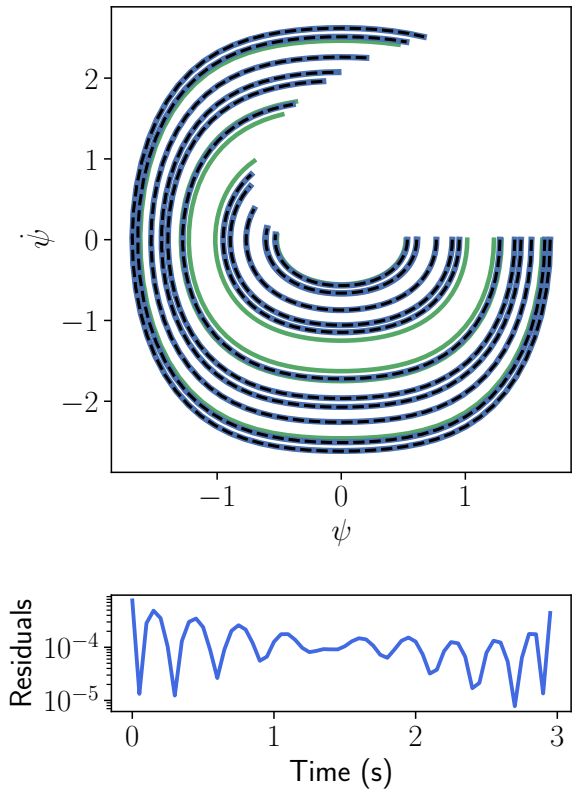

Figure 3. Top: phase space of predicted trajectories of a nonlinear oscillator system. The training curves ares shown in green and the test in blue. Dashed black lines represent ground truth solutions. Bottom: average residuals of 30 predicted solutions across different initial conditions.

5.2.1. Poisson Equation

The Poisson equation is an extensively studied PDE in physics, typically used to identify an electrostatic potential ψ given a charge distribution ρ. In 2-D, it can be described by:

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = \rho(x, y). \tag{17}$$

We define this PDEs in the domain $x \in [x_L, x_R]$ and $y \in [y_B, y_T]$ with BCs:

$$\psi(x_L, y) = \psi(x_R, y) = \psi(x, y_B) = \psi(x, y_T) = 0. \tag{18}$$

We train the network on 4 different charge distributions $\rho(x, y) = \sin(k\pi x)\sin(k\pi y)$ for $k \in 1, 2, 3, 4$. We then evaluate the performance of our network in two settings. The first is an ablation across 100 linearly spaced values of k in $[1, 4]$ (see Table 1). As a second experiment we test the proposed transfer learning method on a harder testing force function of the form:

$$\rho_{\text{test}} = \frac{1}{4}\sum_{k=1}^{4}(-1)^{k+1} 2k \sin(k\pi x)\sin(k\pi y). \tag{19}$$

The solution is shown in the top graph of Fig. 4. To assess the network performance, we plot in the lower graph of Fig. 4 the mean square error (MSE) computed between the predicted $\psi(x, y)$ and the analytical solution which reads:

$$\psi(x, y) = \frac{-1}{2(k\pi)^2} \sin(k\pi x) \sin(k\pi y). \tag{20}$$

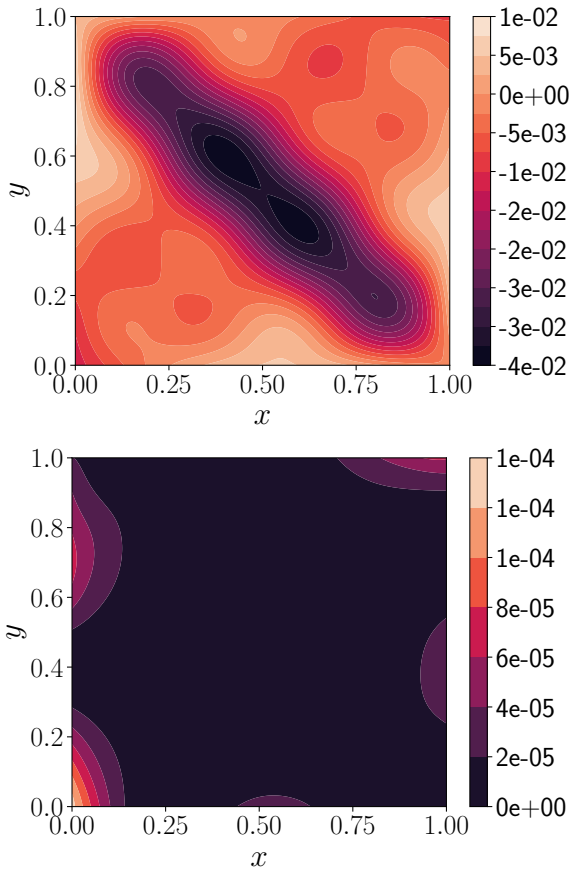

Figure 4. Predicted solution (top) of the Poisson equation with an initial charge distribution $\rho(x, y)$ composed of multiple frequencies k. The network is pre-trained on the individual frequencies and can obtain the solution to the combination in one-shot (35s) with high fidelity/low MSE (bottom).

5.2.2. Time-Dependent Schrödinger Equation

In quantum mechanics the time-dependent Schrödinger equation describes the propagation of a wavefunction through space and time. The PDE in one-dimensional space is of the form:

$$i\hbar \frac{\partial}{\partial t}\psi(x, t) = \left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\psi(x, t), \tag{21}$$

where $\psi(x, t)$ is a complex-valued function called wavefunction, $V(x)$ is a stationary potential function, m is the mass, and $\hbar$ is a constant. We investigate the quantum free-particle evolution for which $V(x) = 0$ for several initial states $\psi(x, 0)$.

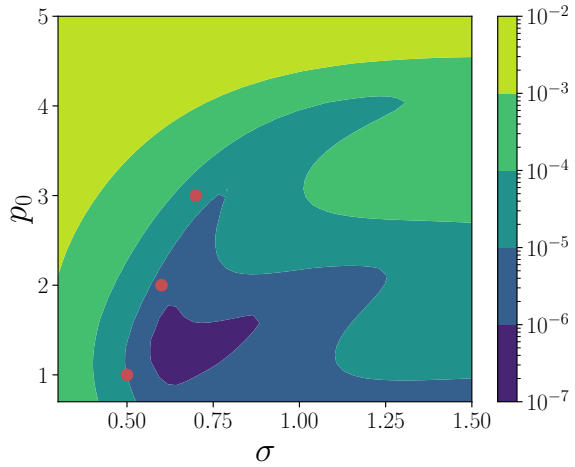

Figure 5. MSE between predicted and analytic solutions $|\psi|^2$ as a function of σ and p_0. Red circles represent the three configurations for which the network was batch trained. We see that as p_0 increases, transfer learning W_{out} becomes less effective because of the F-principle bottleneck for PINNs.

To train the complex-valued wave-function, we separate the real ψ_R and imaginary ψ_I parts (Raissi et al., 2019), namely $\psi(x, t) = \psi_R(x, t) + i\psi_I(x, t)$. By plugging the above form of ψ into Eqn. 21 we obtain a coupled system of real-valued PDEs as:

$$\frac{\partial}{\partial t}\begin{bmatrix}\psi_R \\ \psi_I\end{bmatrix} = \begin{bmatrix}0 & -\hbar/2m \\ \hbar/2m & 0\end{bmatrix}\frac{\partial^2}{\partial x^2}\begin{bmatrix}\psi_R \\ \psi_I\end{bmatrix}, \tag{22}$$

which is of the form $\boldsymbol{\psi}_t = A\boldsymbol{\psi}_{xx}$. The system is linear and thus, we can obtain analytic W_{out}. We consider a network with two outputs per equation associating with ψ_R and ψ_I, where each output is, respectively, described by a set of weights as $W_{\text{out}} = [W_{\text{R}}, W_{\text{I}}]^T$. Then, the network solutions read:

$$\begin{bmatrix}\psi_R \\ \psi_I\end{bmatrix} = \begin{bmatrix}H & 0 \\ 0 & H\end{bmatrix}\begin{bmatrix}W_R \\ W_I\end{bmatrix}, \tag{23}$$

By taking the L_2 loss of Eqn. 22 as well as the BCs and ICs we obtain:

$$\begin{aligned} W_{\text{out}} =& (D_H^T D_H + H_0^T H_0 \\ &+ H_d^T H_d + \dot{H}_d^T \dot{H}_d)^{-1}(H_0^T \psi_0), \end{aligned} \tag{24}$$

where $H_0 = H(0, x)$, $H_d = H(t, L) - H(t, R)$, and $\dot{H}_d = H_x(t, L) - H_x(t, R)$.

To investigate a particular set of solutions, we define the

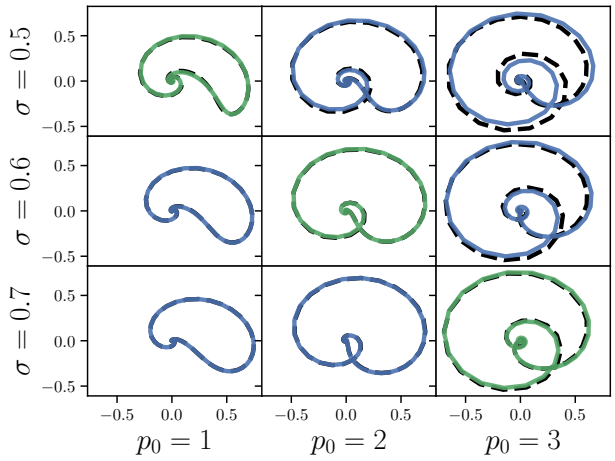

Figure 6. Real (x-axis) against imaginary (y-axis) wave-functions of the predicted solutions $\psi(T_{\max}, x)$ for different realizations of σ, p_0 with solid and dashed lines representing the predicted and ground truth solutions. The diagonal configurations are used for training. Non-diagonals constitute test configurations.

initial condition for this problem as:

$$\psi(x,0) = \frac{1}{\pi^{1/4}\sqrt{\sigma}} e^{-(x-x_0)^2/(2\sigma^2)+ip_0x/\hbar}, \tag{25}$$

that leads to the exact solution:

$$\psi(x,t) = \frac{e^{-\frac{(x-(x_0+p_0t/m))^2}{2\sigma^2(1+i\hbar t/m\sigma^2)}} e^{i(p_0x-Et)/\hbar}}{\pi^{1/4}\sqrt{\sigma(1+i\hbar t/m\sigma^2)}}, \tag{26}$$

where $E = p_0^2/2m$.

We train a network simultaneously on three solutions of Eqn. (21) with pairs of σ, p_0 such that the network is trained for $(\sigma, p_0) = \{(0.5, 1), (0.6, 2), (0.6, 3)\}$. We show that by using only 3 training ICs, accurate solutions to multiple other configurations can be obtained instantly and with high accuracy (see Fig. 5). We measure and present in Fig. 5 the accuracy of our predicted solution by computing the MSE across time and space against the analytic solution Eqn. 26. Furthermore, we show that near the bundles, the predicted real-imaginary complex space under different σ, p_0 pairs tightly couples to the analytic solution (see Fig. 6). Our results also highlight the F-principle, a conclusion drawn about deep networks which shows that high frequency components require more training (Xu et al., 2019) as can be seen in Fig. 5 by looking at higher p_0 values which induce higher frequency components in the solution ψ.

6. Conclusion

We have extensively shown how PINNs can be batch trained on a family of differential equations to learn a rich latent space that can be exploited for transfer learning. For linear systems of ODEs and PDEs, the transfer can be reduced to computing a closed-form solution for W_{out} resulting in one-shot inference. This analytic solution significantly speeds up inference on unseen differential equations by orders of magnitude, and can therefore replace or augment traditional transfer learning. In particular, we show that such a network can identify first-order ODEs, second-order coupled ODEs, Poisson and Schrödinger equations with high levels of accuracy within a few seconds. Furthermore, in the nonlinear setting, where a closed-form analytic W_{out} is not derived, we show that our approach can still be used with gradient descent to identify accurate solutions to the nonlinear oscillator system. These results are particularly important to practitioners who seek to rapidly identify accurate solutions to differential equations of the same type, namely of the same order, but under different conditions and coefficients. Indeed many new applications may arise as a consequence of this approach, from transfer learning on large domains, to solving high dimensional linear PDEs. Future work in this direction may adapt to data-dependent settings, incorporate non-linear outputs to develop one-shot training for nonlinear equations, and investigate properties of the learnt hidden space.

References

Chen, F., Sondak, D., Protopapas, P., Mattheakis, M., Liu, S., Agarwal, D., and Giovanni, M. D. Neurodiffeq: A python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46): 1931, 2020. doi: 10.21105/joss.01931.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural Ordinary Differential Equations. pp. 6571–6583, 2018. URL http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf.

Choudhary, A., Lindner, J. F., Holliday, E. G., Miller, S. T., Sinha, S., and Ditto, W. L. Physics enhanced neural networks predict order and chaos. *arXiv:1912.01958 [physics]*, November 2019. URL http://arxiv.org/abs/1912.01958. arXiv: 1912.01958.

Cranmer, M., Greydanus, S., Hoyer, S., Battaglia, P., Spergel, D., and Ho, S. Lagrangian Neural Networks. *arXiv:2003.04630 [physics, stat]*, March 2020. URL http://arxiv.org/abs/2003.04630. arXiv: 2003.04630.

de Wolff, T., Carrillo, H., Martí, L., and Sanchez-Pi, N. Towards Optimally Weighted Physics-Informed Neural Networks in Ocean Modelling. *arXiv:2106.08747 [physics]*,

June 2021. URL http://arxiv.org/abs/2106.08747. arXiv: 2106.08747.

Desai, S., Mattheakis, M., Sondak, D., Protopapas, P., and Roberts, S. J. Port-Hamiltonian Neural Networks for Learning Explicit Time-Dependent Dynamical Systems. *CoRR*, abs/2107.08024, 2021a. URL https://arxiv.org/abs/2107.08024. _eprint: 2107.08024.

Desai, S. A., Mattheakis, M., and Roberts, S. J. Variational integrator graph networks for learning energy-conserving dynamical systems. *Phys. Rev. E*, 104: 035310, Sep 2021b. doi: 10.1103/PhysRevE.104.035310. URL https://link.aps.org/doi/10.1103/PhysRevE.104.035310.

Dormand, J. R., El-Mikkawy, M. E. A., and Prince, P. J. Families of Runge-Kutta-Nystrom Formulae. *IMA Journal of Numerical Analysis*, 7(2):235–250, 1987. ISSN 0272-4979, 1464-3642. doi: 10.1093/imanum/7.2.235. URL https://academic.oup.com/imajna/article-lookup/doi/10.1093/imanum/7.2.235.

Flamant, C., Protopapas, P., and Sondak, D. Solving Differential Equations Using Neural Network Solution Bundles. *arXiv:2006.14372 [physics]*, June 2020. URL http://arxiv.org/abs/2006.14372. arXiv: 2006.14372.

Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian Neural Networks. In Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F. d., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 15379–15389. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9672-hamiltonian-neural-networks.pdf.

Guo, Y., Zhang, H., Wang, L., Fan, H., and Wang, X. Transfer learning of chaotic systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(1): 011104, January 2021. ISSN 1054-1500, 1089-7682. doi: 10.1063/5.0033870. URL http://arxiv.org/abs/2011.09970. arXiv: 2011.09970.

Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018. doi: 10.1073/pnas.1718942115. URL https://www.pnas.org/doi/abs/10.1073/pnas.1718942115.

Hennigh, O., Narasimhan, S., Nabian, M. A., Subramaniam, A., Tangsali, K., Rietmann, M., del Aguila Ferrandis, J., Byeon, W., Fang, Z., and Choudhry, S. Nvidia simnet: an ai-accelerated multi-physics simulation framework. *arXiv:2012.07938 [physics.flu-dyn]*, 2020. URL https://arxiv.org/abs/2012.07938.

Hochlehnert, A., Terenin, A., Sæmundsson, S., and Deisenroth, M. P. Learning Contact Dynamics using Physically Structured Neural Networks. *arXiv:2102.11206 [cs, stat]*, February 2021. URL http://arxiv.org/abs/2102.11206. arXiv: 2102.11206.

Howse, J. W., Abdallah, C. T., and Heileman, G. L. Gradient and Hamiltonian Dynamics Applied to Learning in Neural Networks. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E. (eds.), *Advances in Neural Information Processing Systems 8*, pp. 274–280. MIT Press, 1996.

Jaeger, H. and Haas, H. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. doi: 10.1126/science.1091277.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, June 2021. ISSN 2522-5820. doi: 10.1038/s42254-021-00314-5. URL http://www.nature.com/articles/s42254-021-00314-5.

Kaxiras, E., Neofotistos, G., and Angelaki, E. The first 100 days: modeling the evolution of the COVID-19 pandemic. *arXiv:2004.14664 [q-bio]*, April 2020. URL http://arxiv.org/abs/2004.14664. arXiv: 2004.14664.

Lagaris, I. E., Likas, A., and Fotiadis, D. I. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*, 9(5): 987–1000, September 1998. ISSN 10459227. doi: 10.1109/72.712178. URL http://arxiv.org/abs/physics/9705023. arXiv: physics/9705023.

Lee, K., Trask, N. A., and Stinis, P. Machine learning structure preserving brackets for forecasting irreversible processes. *arXiv:2106.12619 [physics]*, June 2021. URL http://arxiv.org/abs/2106.12619. arXiv: 2106.12619.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. M., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *CoRR*, abs/2010.08895, 2020. URL https://arxiv.org/abs/2010.08895.

Lu, L., Meng, X., Mao, Z., and Karniadakis, G. E. A deep learning library for solving differential equations. *SIAM Review*, 63:208–228, 2021. doi: 10.1137/19M1274067.

Maclaurin, D., Duvenaud, D., and Adams, R. P. Autograd: Effortless Gradients in Numpy. pp. 3.

Mattheakis, M., Joy, H., and Protopapas, P. Unsupervised reservoir computing for solving ordinary differential equations. *arXiv:2108.11417 [cs:LG]*, 2021. URL http://arxiv.org/abs/2108.11417. arXiv: 2108.11417.

Mattheakis, M., Sondak, D., Dogra, A. S., and Protopapas, P. Hamiltonian neural networks for solving equations of motion. *Phys. Rev. E*, 105:065305, 2022. doi: 10.1103/PhysRevE.105.065305. URL https://link.aps.org/doi/10.1103/PhysRevE.105.065305.

McClenny, L. and Braga-Neto, U. Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism. *arXiv:2009.04544 [cs, stat]*, September 2020. URL http://arxiv.org/abs/2009.04544. arXiv: 2009.04544.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045. URL http://www.sciencedirect.com/science/article/pii/S0021999118307125.

Saemundsson, S., Terenin, A., Hofmann, K., and Deisenroth, M. Variational Integrator Networks for Physically Structured Embeddings. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 3078–3087. PMLR, June 2020. URL https://proceedings.mlr.press/v108/saemundsson20a.html. ISSN: 2640-3498.

Sahli Costabal, F., Yang, Y., Perdikaris, P., Hurtado, D. E., and Kuhl, E. Physics-Informed Neural Networks for Cardiac Activation Mapping. *Frontiers in Physics*, 8, 2020. ISSN 2296-424X. doi: 10.3389/fphy.2020.00042. URL https://www.frontiersin.org/articles/10.3389/fphy.2020.00042/full. Publisher: Frontiers.

Sanchez-Gonzalez, A., Bapst, V., Cranmer, K., and Battaglia, P. Hamiltonian Graph Networks with ODE Integrators. *arXiv:1909.12790 [physics]*, September 2019. URL http://arxiv.org/abs/1909.12790. arXiv: 1909.12790.

Schwartz, M. Lecture 3: Coupled oscillators. pp. 6, 2017.

Sirignano, J. and Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, December 2018. ISSN 00219991. doi: 10.1016/j.jcp.2018.08.029. URL http://arxiv.org/abs/1708.07469. arXiv: 1708.07469.

Wang, H., Planas, R., Chandramowlishwaran, A., and Bostanabad, R. Train Once and Use Forever: Solving Boundary Value Problems in Unseen Domains with Pre-trained Deep Learning Models. *arXiv:2104.10873 [physics]*, apr 2021. URL http://arxiv.org/abs/2104.10873. arXiv: 2104.10873.

Wang, S. and Perdikaris, P. Deep learning of free boundary and Stefan problems. *Journal of Computational Physics*, 428:109914, March 2021. ISSN 00219991. doi: 10.1016/j.jcp.2020.109914. URL http://arxiv.org/abs/2006.05311. arXiv: 2006.05311.

Xu, Z.-Q. J., Zhang, Y., and Xiao, Y. Training behavior of deep neural network in frequency domain. *arXiv:1807.01251 [cs, math, stat]*, October 2019. URL http://arxiv.org/abs/1807.01251. arXiv: 1807.01251.

Yang, L., Zhang, D., and Karniadakis, G. E. Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations. *arXiv:1811.02033 [cs, math, stat]*, November 2018. URL http://arxiv.org/abs/1811.02033. arXiv: 1811.02033.

Yin, Y., Guen, V. L., Dona, J., de Bézenac, E., Ayed, I., Thome, N., and Gallinari, P. AUGMENTING PHYSICAL MODELS WITH DEEP NET- WORKS FOR COMPLEX DYNAMICS FORECASTING. pp. 22, 2021.

Zhai, H. and Hu, G. Inferring micro-bubble dynamics with physics-informed deep learning. *arXiv:2105.07179 [physics]*, may 2021. URL http://arxiv.org/abs/2105.07179. arXiv: 2105.07179.

Zhang, R., Liu, Y., and Sun, H. Physics-Informed Multi-LSTM Networks for Metamodeling of Nonlinear Structures. *Computer Methods in Applied Mechanics and Engineering*, 369:113226, September 2020. ISSN 00457825. doi: 10.1016/j.cma.2020.113226. URL http://arxiv.org/abs/2002.10253. arXiv: 2002.10253.

Zhong, Y. D., Dey, B., and Chakraborty, A. Dissipative SymODEN: Encoding Hamiltonian Dynamics with Dissipation and Control into Deep Learning. *arXiv:2002.08860 [cs, eess, stat]*, April 2020. URL http://arxiv.org/abs/2002.08860. arXiv: 2002.08860.

A. System of second order differential equations

To compute the analytic W_{out} for a system of second order differential equation we begin by defining:

$$\ddot{\boldsymbol{\psi}} = A\boldsymbol{\psi}$$

where dots denote time derivatives and

$$\boldsymbol{\psi} = \begin{bmatrix} H & 0 \\ 0 & H \end{bmatrix} \begin{bmatrix} W_q \\ W_p \end{bmatrix}.$$

Then, by computing the $L2$ loss on the equation above including initial conditions $\psi_0 = \psi(0), \dot{\psi}_0 = \dot{\psi}(0)$ we obtain:

$$W_{\text{out}} = \left(D_H^T D_H + H_0^T H_0 + H_{0d}^T H_{0d}\right)^{-1} \left(H_0^T \psi_0 + H_{0d}^T \dot{\psi}_0\right)$$

where

$$D_H = \begin{bmatrix} \ddot{H} & 0 \\ 0 & \ddot{H} \end{bmatrix},$$
$$H_0 = \begin{bmatrix} H(0) & 0 \\ 0 & H(0) \end{bmatrix},$$
$$H_{0d} = \begin{bmatrix} \dot{H}(0) & 0 \\ 0 & \dot{H}(0) \end{bmatrix}.$$

B. QR Decomposition

In the main manuscript, we define the loss function for ODEs as:

$$\mathcal{L} = (\hat{D}_n u_\theta(t) - f(t))^2 + (\bar{D}_0 u_\theta(t) - u_{\text{ic}})^2, \tag{27}$$

For ease of notation, let $\hat{D}_n u_\theta(t) = \hat{Y}$, $f(t) = Y$, $\bar{D}_0 u_\theta(t) = \hat{Y}_0$ and $u_{\text{ic}} = Y_0$. The loss function above can be re-written as a single loss function s.t.:

$$\mathcal{L} = \left(\begin{bmatrix} \hat{Y} \\ \hat{Y}_0 \end{bmatrix} - \begin{bmatrix} Y \\ Y_0 \end{bmatrix} \right)^2. \tag{28}$$

To see this, we can expand the vector notation as:

$$\mathcal{L} = \left(\begin{bmatrix} \hat{Y} & \hat{Y}_0 \end{bmatrix} - \begin{bmatrix} Y & Y_0 \end{bmatrix} \right) \left(\begin{bmatrix} \hat{Y} \\ \hat{Y}_0 \end{bmatrix} - \begin{bmatrix} Y \\ Y_0 \end{bmatrix} \right), \tag{29}$$

which when expanded resolves to Eqn.27.

By differentiating the above loss equation with respect to W_{out} and setting it to zero, we obtain a linear least squares problem as

$$\left(H^T H + H_0^T H_0\right) W_{\text{out}} = \left(H^T Y + H^T Y_0\right), \tag{30}$$

which is of the form

$$A^T A \, W_{\text{out}} = A^T Y. \tag{31}$$

Since the left hand-side is a square matrix $A^T A$, it is possible to take its pseudo-inverse. However, for a number of problems, it is possible that the matrix A has a large condition number and therefore the squaring procedure of the normal equations squares the condition number making it unstable. Although it is possible to take a pseudo-inverse, in such cases, rather than using the normal equations to obtain a solution to W_{out}, it is possible to solve for W_{out} using QR decomposition. In other words:

$$AW_{\text{out}} = Y, \tag{32}$$

and since A is not square, we can decompose it into $A = QR$ such as

$$W_{\text{out}} = R^{-1}Q^T Y. \tag{33}$$

One advantage of using QR is that it avoids forming the gram matrix $A^T A$ of the normal equations which can be singular.

C. Computational Complexity

One special outcome of our formalism is that it separates the homogeneous part of the differential equation from the initial conditions and forces. In other words, if we investigate the normal equations for W_{out}, we see that

$$W_{\text{out}} = (\hat{D}_H^T \hat{D}_H + \bar{D}_H^T \bar{D}_H)^{-1}(D_H^T f'(t) + \bar{D}_H^T u'_{\text{ic}}). \tag{34}$$

Notice that the forces and initial conditions appear outside the matrix inversion. This is a particularly important feature as it allows us to scale rapidly when the differential equation is fixed and the solution to many initial conditions or forces is required. In fact, the computational complexity of the inversion is $\mathcal{O}(h^3)$ where h is the number of output neurons of the final hidden layer and the multiplication is $\mathcal{O}(h^2 m)$, where m is the number of initial conditions and forces. Therefore, if the differential equation is fixed the total computational cost of computing W_{out} is $\mathcal{O}(h^3 + mh^2)$. However, if the differential equation is not fixed and varies across all m samples, then the inversion has to be computed m times such that the total computational cost is $\mathcal{O}(mh^3 + mh^2)$.

D. Training Configuration

All models are trained using an Adam optimizer with a learning rate of 10^{-3}. The networks are all trained on a Macbook Pro, 2.2 GHz Intel Core i7, 16 Gb RAM. We use 64-bit tensors.

Differential Equation	Architecture (N bundles)	# training Bundles	Training Iterations	# Training Collocation Points	Evaluation Domain	Evaluation Deltas	Activations	Optimizer
First-Order Linear Inhomogeneous	1-100-100-1*N	10	10000	30	t in [0,3]	dt = 0.1	tanh	Adam
Second-Order Linear Inhomogeneous	1-100-100-1*N	10	10000	30	t in [0,3]	dt = 0.05	tanh	Adam
Coupled-Oscillator	1-100-100-2*N	10	10000	50	t in [0,10]	dt = 0.01	sin	Adam
Non-Linear Oscillator	1-100-100-1*N	5	10000	60	t in [0,3]	dt = 0.05	sin	Adam
Poisson	2-100-100-1*N	4	40000	1000	x in [0,1], t in [0,1]	dx = 0.01, dt = 0.01	sin	Adam
Schroedinger	2-100-100-2*N	3	40000	1000	x in [-10,10], t in [0,1]	dx = 0.1, dt = 0.01	$\alpha \sin + (1-\alpha)\tanh$	Adam

First-Order Linear Inhomogeneous

The equation is of the form:

$$\dot{u} + a(t)u = f(t), \tag{35}$$

subjected to an initial condition u_0. We sample within:

$$\begin{aligned} f &\in \{\cos(t), \sin(t), t\}, \\ a &\in \{t, t^2, 1\}, \\ (u_0) &\in [-5, 5]. \end{aligned}$$

Second-Order Linear Inhomogeneous

The equation is of the form:

$$\ddot{u} + a_1(t)\dot{u} + a(t)u = f(t) \tag{36}$$

with initial conditions u_0 and $\dot{u}_0$. We sample within:

$$\begin{aligned} f &\in \{1, t, \cos(t), \sin(t)\}, \\ a &\in \{1, 3t, t^2\}, \\ a_1 &\in \{1, t^2, t^3\}, \\ (u_0, \dot{u}_0) &\in [-5, 5]. \end{aligned}$$

Coupled oscillator

The equation is of the form:

$$\begin{bmatrix} m & 0 \\ 0 & m \end{bmatrix} \begin{bmatrix} \ddot{u}_1 \\ \ddot{u}_2 \end{bmatrix} = \begin{bmatrix} k_1 + k_2 & -k_2 \\ -k_2 & k_1 + k_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}. \tag{37}$$

We sample:

$$\begin{aligned} m &\in [1, 2], \\ (k_1, k_2) &\in [0.5, 4.5], \\ (u_0, u_1, \dot{u}_{1,0}, \dot{u}_{2,0}) &\in [-1.5, 1.5]. \end{aligned}$$

Nonlinear oscillator

The equation is:

$$\ddot{u} + u + u^3 = 0. \tag{38}$$

We sample within:

$$(u_0, \dot{u}_0) \in [0.5, 2].$$

Poisson equation

$$\nabla^2 u = \rho \tag{39}$$

we sample:

$$\rho \in \{\sin(x)\sin(y), \sin(2x)\sin(2y), \sin(3x)\sin(3y), \sin(4x)\sin(4y)\}.$$

Schrodinger

$$\begin{bmatrix} \dot{u}_R \\ \dot{u}_I \end{bmatrix} = \begin{bmatrix} 0 & -\hbar/2m \\ \hbar/2m & 0 \end{bmatrix} \begin{bmatrix} u''_R \\ u''_I \end{bmatrix} \tag{40}$$

where $\psi = (u_R, u_I)$ and:

$$\psi(x, 0) = \frac{1}{\pi^{1/4}\sqrt{\sigma}} e^{-(x-x_0)^2/(2\sigma^2) + ip_0 x/\hbar}. \tag{41}$$

and

$$\begin{aligned} \sigma &\in \{0.5, 0.6, 0.7\}, \\ p_0 &\in \{1, 2, 3\}. \end{aligned}$$

E. Training Bundles

Typically, the more training bundles we use, the more diverse the hidden states have to be in order to generate accurate solutions for all the training differential equations. As such, we would expect that a diverse set of hidden states should also result in better inference for unseen differential equations. Experimentally, we find and show in Fig. 7 that for first order differential equations this is true:

Of course doing this also increases the training overhead, thus empirically we use 10 consistently across our experiments.

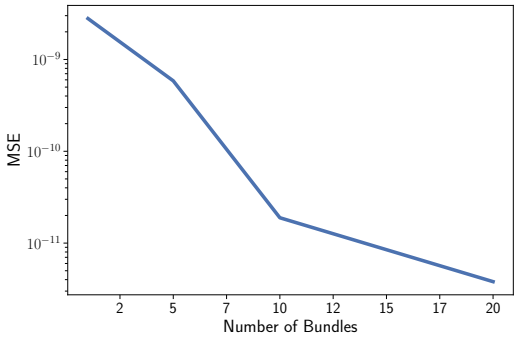

Figure 7. Test MSE as a function of number of bundles. The more bundles we use to train, the better our test accuracy gets.