Universal Physics-Informed Neural Networks: Symbolic Differential Operator Discovery with Sparse Data

Lena Podina [*1] **Brydon Eastman** [*2] **Mohammad Kohandel** [3]

Abstract

In this work we perform symbolic discovery of differential operators in a situation where there is sparse experimental data. This small data regime in machine learning can be made tractable by providing our algorithms with prior information about the underlying dynamics. Physics Informed Neural Networks (PINNs) have been very successful in this regime (reconstructing entire ODE solutions using only a single point or entire PDE solutions with very few measurements of the initial condition). The Universal PINN approach (UPINN) adds a neural network that learns a representation of unknown hidden terms in the differential equation. The algorithm yields both a surrogate solution to the differential equation and a black-box representation of the hidden terms. These hidden term neural networks can then be converted into symbolic equations using symbolic regression techniques like AI Feynman. In order to achieve convergence of the neural networks, we provide our algorithms with (noisy) measurements of both the initial condition as well as (synthetic) experimental data obtained at later times. We demonstrate strong performance of UPINNs even when provided with very few measurements of noisy data in both the ODE and PDE regime.

1. Introduction

Machine learning algorithms, for instance neural networks (NN), are particularly helpful in representing unknown quantities in a data-driven way (Belohlav et al., 1997). NNs with a wide enough hidden layer can be used to approximate any function (Pinkus, 1999) by tuning its parameters (henceforth 'NN parameters'). Hence, NNs have been used to infer DE parameters or even entire DE models, due to their ability to approximate functions. Many recent applications use NNs augmented with prior knowledge in order to learn underlying DE models[1] from data (Chakraborty, 2020; Raissi et al., 2019; Lu et al., 2021b; Rackauckas et al., 2020; Meng & Karniadakis, 2020; Lu et al., 2021a). However, acquiring sufficient data to fit these values accurately using NNs is difficult. A method that can function in low-data regimes by leveraging the known structure of the model is needed.

Two prominent NN-based methods that learn DE models from data are physics-informed neural networks (PINN) (Raissi et al., 2019) and universal differential equations (UDE) (Rackauckas et al., 2020). In UDEs, each unknown component of the DE model is approximated by a NN and a hard DE constraint is employed. That is, the best-fit DE is satisfied at all times during training. However, UDEs are not robust to noise, require a lot of data, and SINDy, as employed in (Rackauckas et al., 2020), does not succeed in finding the true mechanistic model reliably. PINNs assume the form of the true DE and fits its parameters via a soft constraint (relaxing the requirement that the NN should satisfy the best-fit DE exactly), which is added to the NN loss function as an additional loss term referred to as the 'PINN loss'. A drawback of PINNs is that the structure of the DE model must be determined in advance, and there is no way to learn its unknown components using the method as originally proposed. Additionally, as iterative optimization is computationally expensive, PINN loss can fail on stiff DEs (Wang et al., 2021).

Our approach, Universal PINNs (UPINNs), bridges the limitations of both PINNs (cannot be used when the structure of the DE is not fully known) (Raissi et al., 2019) and UDEs (not robust to noise and requires lots of data) (Rackauckas et al., 2020). To address this, we replace the hard constraint of the UDE with that of PINN loss, which allows the approach to learn unknown components of the DE model from data. This approach is robust to noise and performs well in low-data regimes. Additionally, using the AI Feynman

[*]Equal contribution [1]Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada [2]OpenAI, San Francisco, USA [3]Department of Applied Mathematics, University of Waterloo, Waterloo, Canada. Correspondence to: Lena Podina <lpodina@uwaterloo.ca>, Mohammad Kohandel <kohandel@uwaterloo.ca>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

[1]The (underlying or true) DE model of a process refers to the DEs that produced the experimental data.

algorithm (Udrescu & Tegmark, 2020) yields good results in identifying the underlying hidden terms of the DE model.

In this paper, we claim the following contributions:

1. We propose a novel training methodology, Universal PINN, which combines PINN loss with the UDE framework to allow a PINN-based approach to learn unknown parts of the DE model from data
2. Our method is robust to noise and learns the unknown DE model components to significantly higher accuracy in the Lotka-Volterra model compared to the state of the art (UDE approach)
3. UPINNs perform very well in systems biology ODEs, and the Viscous Burgers' equation partial differential equation
4. Using a symbolic regression algorithm, we reached better identification of the tested systems than in the original UDE paper

2. Background

As per (Raissi et al., 2019), suppose that the following DE governs a physical process. $u(t, x)$ is an unknown real-valued function of time (t) and position (x). Its time derivative is related to its value for each tuple (t, x) with a known function $\mathcal{N}$, and unknown vector of parameters θ. Furthermore, there are N potentially noisy DE measurements $\{t_i, x_i, u_i\}$.

$$\frac{\partial u(t,x)}{\partial t} = \mathcal{N}[u;\theta], x \in \Omega, \Omega \in R^D, t \in [0,T] \quad (1)$$

Although θ is required in order to find a numerical or analytical function u that satisfies 1, θ is unknown in this setup. Using the given data, the PINN method from (Raissi et al., 2019) can estimate θ and u simultaneously. Its key component is a neural network U, which predicts u given any tuple (t, x). The following loss function is used to train U:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}|U(t^i, x^i) - u^i| + \frac{1}{M}\sum_{j=1}^{M}\left|\frac{dU}{dt_j} - \mathcal{N}[u;\theta]\right| \quad (2)$$

where $\frac{dU}{dt_j}$ is auto-differentiated through the neural network and evaluated at time t_j and position x_j. The first term penalizes U for making predictions that do not match the DE at a predefined set of M collocation points $\{t_j, x_j\}$. The second term penalizes U for making predictions that do not match the data. Note that the second term contains θ, which allows parameter estimate $\hat{\theta}$ to be updated using its gradient with respect to $\mathcal{L}$. At every optimization iteration, the parameters of U are updated along with $\hat{\theta}$.

A UDE (Rackauckas et al., 2020) is a DE which is defined in part using universal approximators, e.g. a neural network. These NNs can be used to approximate unknown components of $\mathcal{N}$. We will assume that possibly noisy data $\{t_i, x_i, u_i\}$ is available from Eq. 1. Suppose that $\mathcal{N}$ is a function g composed of k unknown functions h_i and known parameters θ:

$$\mathcal{N}[u;\theta] = g(u, h_1(u;\theta), \ldots, h_k(u;\theta);\theta) \quad (3)$$

In (Rackauckas et al., 2020), the h_i terms are approximated by a single neural network H with k outputs and fit using iterative optimization such as Adam (Kingma & Ba, 2014) or gradient descent (Bishop & Nasrabadi, 2006). Since Eq. (1) always holds, the loss function only consists mean squared error between the DE solution and the data. Note that with any particular approximation H, the DE (1) is defined fully and u can be solved for numerically. Hence, the training loop for UDEs involves numerically solving Eq. (1), computing the error between the solution and the data, and updating H to better approximate the unknown components of $\mathcal{N}$. At the end of training, H will represent the unknown component of $\mathcal{N}$ and the numerical solution of Eq. (1) will yield u that matches the solution of the true DE.

Symbolic regression is a general technique of finding a model that fits data while balancing the simplicity of the model with its accuracy. This problem has been solved with various genetic algorithms (see, among others, (McKay et al., 1995; Schmidt & Lipson, 2009)) but since this method is computationally expensive, newer techniques are becoming more popular. For example, AI Feynman (Udrescu & Tegmark, 2020) leverages NNs and symmetry, units, compositionality, etc. and finally returns a list of potential models ranked by error and complexity. Some work (Valipour et al., 2021; Kamienny et al., 2022) makes use of transformers to find the correct functional form. In our work, we only use them at the final stage after our method has learned an approximate representation of the missing components.

3. Methods

Our proposed method, Universal PINNs, is a modification of the PINNs to discover the functional form of an unknown term within a differential equation. Suppose $\vec{u}(\vec{x}, t) \in \mathbb{R}^m$ for $\vec{x} \in \mathbb{R}^d$. Let $\mathcal{N}$ be a (potentially nonlinear) differential operator, then consider time t in the domain $[0, T] \subset \mathbb{R}$ along with a d dimensional, bounded spatial domain $\Omega \subset \mathbb{R}^d$ where $\partial\Omega$ denotes the boundary of Ω. Notably, if $\mathcal{N}$ contains any derivatives, we assume that those derivatives are with respect to the spatial variables only. We then consider problems of the form

$$\frac{d}{dt}\vec{u}(\vec{x}, t) = \mathcal{N}[\vec{u}](\vec{x}, t), \quad t \in [0, T], \quad \vec{x} \in \Omega$$

subject to initial condition

$$\vec{u}(\vec{x}, 0) = \vec{u}_0(\vec{x}), \quad \vec{x} \in \Omega$$

and boundary conditions

$$\beta[\vec{u}](\vec{x}, t) = 0, \quad \vec{x} \in \partial\Omega, \quad t \in [0, T]$$

where β is a (potentially non-linear) differential operator whose derivatives are only with respect to the spatial variables.

Further, suppose

$$\mathcal{N}[\vec{u}](\vec{x}, t) = \mathcal{N}_K[\vec{u}](\vec{x}, t) + \mathcal{F}[\vec{u}](\vec{x}, t)$$

where $\mathcal{N}_K$ is some differential operator with known functional form and $\mathcal{F}$ represents some unknown, target differential operator. Similarly, suppose $\beta = \beta_K + \mathcal{B}$ for some known β_K and some unknown $\mathcal{B}$.

Finally, one can consider $\Omega = \varnothing$, in which case the underlying differential law is governed by an ordinary differential equation (ODE). In this situation, there is no boundary condition and so no need for β (or, equivalently, β is the empty function).

Suppose we have n data points $D = \{(t_k, \vec{x}_k, \vec{u}_k)\}_{k=0}^{n-1}$ where $\vec{u}_k = \vec{u}(t_k, \vec{x}_k) + \epsilon_k$ where ϵ_k is some noise term (potentially $\epsilon_k = 0$). We will use this measured data to fit the parameters of (up to) three neural networks. The first network, $F(\vec{u}; \theta_F)$, will approximate the target differential operator $\mathcal{F}[\vec{u}]$ by using a neural network with parameters θ_F. The second network, $U(\vec{x}, t; \theta_U)$, will approximate the value of $\vec{u}(x, t)$ by a neural network with parameters θ_U. The third network, $B(\vec{u}; \theta_B)$, will approximate the value of $\mathcal{B}[\vec{u}]$, the unknown target for the boundary condition, with a neural network parameterized by θ_B. To fit these networks, we consider another two sets of collocation points: these sets are $X_P = \{(\vec{x}_k, t_k)\}_{k=0}^{n_P-1} \subset (\Omega \setminus \partial\Omega) \times (0, T]$ and $X_B = \{(\vec{x}_k, t_k)\}_{k=0}^{n_B-1} \subset (\partial\Omega) \times (0, T]$. These sets correspond to locations in the space-time domain where we enforce that our network U satisfies the underlying differential equation (in the case of X_P) and the boundary conditions (in the case of X_B).

To calculate the gradients for fitting these networks, we consider the loss function

$$L(\theta_U, \theta_B, \theta_F) = L_M(\theta_U) + L_B(\theta_U, \theta_B) + L_P(\theta_U, \theta_F).$$

The first component of the loss is the MSE loss. This loss is the difference in MSE between the measurement value of $\vec{u} \approx \vec{u}_k$ from the input data with the neural network approximation of $\vec{u} \approx U(\vec{x}_k, t_k)$, evaluated at the same space-time location and is given by

$$L_M(\theta_U) = \frac{1}{n} \sum_{(\vec{x}_k, t_k, \vec{u}_k) \in D} (U(\vec{x}_k, t_k; \theta_U) - \vec{u}_k)^2.$$

The second component of the loss is the boundary loss. This loss is the mean squared value of the approximated value of the boundary condition and is given by

$$L_B(\theta_U, \theta_B) = \frac{1}{n_B} \sum_{(\vec{x}_k, t_k) \in X_B} (\beta_K[U](\vec{x}_k, t_k; \theta_U) + B(U(\vec{x}_k, t_k; \theta_U); \theta_B))^2$$

The final component of the loss is the PINN loss. This loss is the mean squared error between the value U_t, the time derivative of the neural network approximation of U, and the value $\mathcal{N}_K[U] + F(U)$.

$$L_P(\theta_U, \theta_F) = \frac{1}{n_P} \sum_{(\vec{x}_k, t_k) \in X_P} (\mathcal{N}_K[U](\vec{x}_k, t_k; \theta_U) + F(U(\vec{x}_k, t_k; \theta_U); \theta_F) - U_t(\vec{x}_k, t_k; \theta_U))^2.$$

This loss function is quite similar to the loss function for PINNs given in (Raissi et al., 2019), however here we insert two additional neural networks into the loss function corresponding to the unknown parts of the underlying dynamics in the boundary conditions and the differential equation. To compensate for these additional parameters, we extend the first component of the loss to include more than just initial data (but solution data as well). In this way, D could contain data from the initial condition, data from the boundary, or data from the interior of the domain.

Practically, one way to select X_P is to simply choose n_P and use Latin hypercube sampling to select n_P points in $(\Omega \setminus \partial\Omega) \times (0, T]$. A similar construction works for selecting X_B. In this way, we are sampling the domain in a space-filling manner.

The architecture of the fully-connected neural networks is as follows, for each of our models examined in Results:

1. **Burgers**: two inputs for t and x followed by scaling layer; 8 hidden layers of 20 units for the surrogate network and the hidden component network; sigmoid activation
2. **Lotka-Volterra** and **Apoptosis model**: one input for t followed by a scaling layer; 2 hidden layers of 64 units for the surrogate solution; 2 hidden layers of 16 units for the hidden component approximation; sigmoid activation

4. Results

To demonstrate our approach, we show high accuracy in identifying the hidden terms in three test-cases: the Lotka-Volterra equations (an ODE model), and the viscous Burgers' equation (a parabolic PDE model), and a model for cell apoptosis (an ODE model). Additionally, we show that AI Feynman is able to correctly identify the functional form of hidden terms within the Lotka-Volterra model from the output of our model.

4.1. Lotka-Volterra System

We begin our analysis by testing our method on the Lotka-Volterra (LV) model (Berryman, 1992) of predator-prey interactions. The DE is formulated as follows:

$$\begin{aligned}\frac{dx}{dt} &= \alpha x - \beta x y\\ \frac{dy}{dt} &= -\delta y + \gamma x y\ .\end{aligned}$$

We take the known portion of the differential equation as $\mathcal{N}_{\mathcal{K}}[U] = [\alpha\, x, -\delta\, y]$ for known parameters α and δ, and seek to learn $F = [F_1, F_2] \approx [-\beta\, x\, y, \gamma\, x\, y]$ from data only, without knowing the target form and without knowing β and γ.

To generate the synthetic data, $\alpha, \beta, \gamma, \delta$ were fixed at $(1.3, 0.9, 0.8, 1.8)$ respectively, with initial conditions at $(x_0, y_0) = (0.44249296, 4.6280594)$ just as in (Rackauckas et al., 2020). The time interval was chosen as $[0, 3]$ and stayed the same throughout every LV experiment.

An ODE solver was used to generate data satisfying the LV equations. This yields a set of points $\{t_i, x_i, y_i\}$. Then, Gaussian noise is added to each x_i and y_i. Given a particular noise level ϵ, Gaussian noise was added to the data as follows:

$$\begin{aligned}(x_i)_{noise} &= x_i + \epsilon \cdot \bar{x} \cdot N(0,1)\\ (y_i)_{noise} &= y_i + \epsilon \cdot \bar{y} \cdot N(0,1)\end{aligned}$$

where $\bar{x}$ denotes the element-wise mean of x_i over all i (similarly for y).

First, we demonstrate our approach on noise-free data (Table 1) and data with $\epsilon = 5 \times 10^{-3}$ noise (Table 2) for various values of n (number of data points) and n_P (number of collocation points). We want to show how the hard-to-acquire data can be augmented by taking more collocation points which require no experiments/measurements and come at only the cost of increased computing power. We see that, in contrast to a standard PINN approach, we need to provide more data than just the initial condition. However, even with very sparse measurement data, we can acquire a good discovery by only increasing the number of collocation points. The additional benefit gained from increasing the collocation points is only realized when there is already ample enough experimental data for the algorithm to leverage.

Table 1. MSE between F and the true hidden target after training for various values of n and n_P – noiseless data

n \ n_P	10^2	10^3	10^4
1	2×10^1	2×10^1	2×10^1
5	9×10^{-4}	1×10^{-3}	9×10^{-4}
10	2×10^{-4}	4×10^{-5}	5×10^{-6}

Table 2. MSE between F and the true hidden target after training for various values of n and n_P – noisy ($\epsilon = 5 \times 10^{-3}$) data

n \ n_P	10^2	10^3	10^4
1	2×10^1	2×10^1	2×10^1
5	6×10^{-2}	4×10^{-3}	5×10^{-3}
10	1×10^{-3}	6×10^{-4}	8×10^{-4}

Next, we compare UPINN performance to the UDE method. We test the two methods on noiseless sparse data (1) and on noisy data (Fig 2). The error is computed as a mean squared error (MSE) taken with respect to the true interaction. At minimal noise level, the UDE approach and UPINN approach perform similarly and for the densest data UDEs slightly outperform UPINNs. Although increasing either noise or sparsity degrades the performance of both methods, the UPINN method consistently attains a lower MSE compared to the UDE method as noise or sparsity increases.

Figures 6 and 9 show the surrogate solution and hidden terms as recovered by the UDE and UPINN methods. The noise level of the noisy data was set at 0.1 and, for the noiseless sparse data, there were 5 points each 0.6 units apart. It is clear that UPINNs are quite robust to noise and perform well in low-data regimes. The UDE approach performs reasonably on sparse data, but is not robust to noise.

Finally, AI Feynman symbolic regression is run on the neural network output from both our approach and the UDE approach, in order to find the best functional form. These results are presented in Table 4. A dash indicates AI Feynman did not recover the functional form Cxy. The best performance between the two methods is bolded. In cases of both sparse and noisy data, AI Feynman correctly recovers the hidden interaction terms more often for our method than it does for the UDE method. If a formula is recovered for both methods, the one recovered for the PINN method is often more accurate.

The terms $\gamma\, x\, y$ and $-\beta\, x\, y$ in the LV equations correspond to the predator's uptake function in the ecological model.

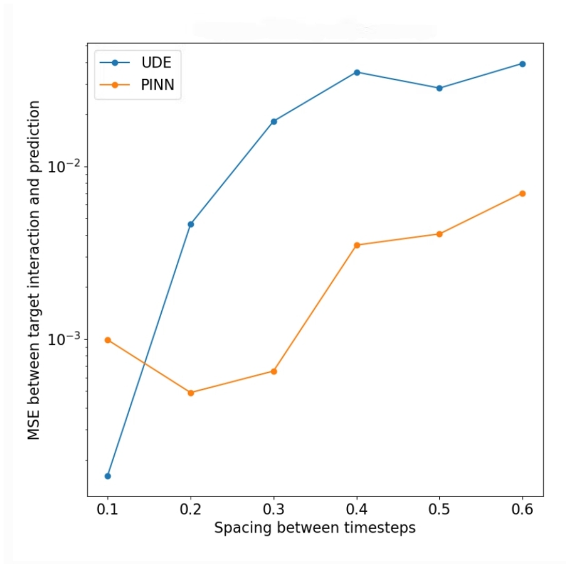

Figure 1. Sparse data regime

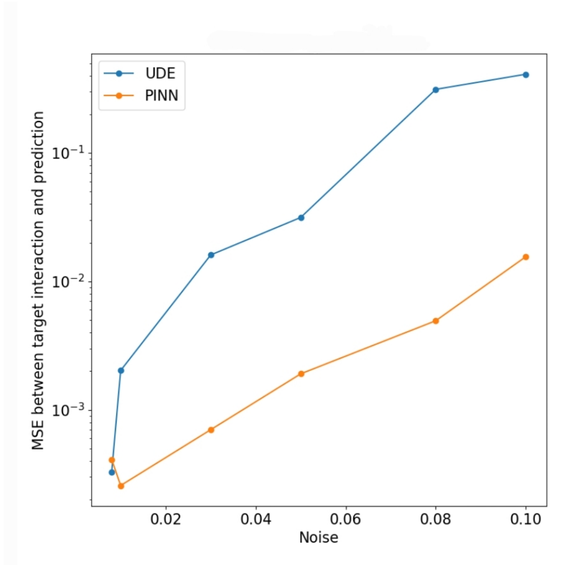

Figure 2. Noisy data regime

Figure 3. Mean squared error (MSE) of the recovery of the true interaction, comparing between the UPINN and UDE method. The spacing parameter determines how much time passes between datapoints, but the overall time interval $[0, 3]$ remains the same.

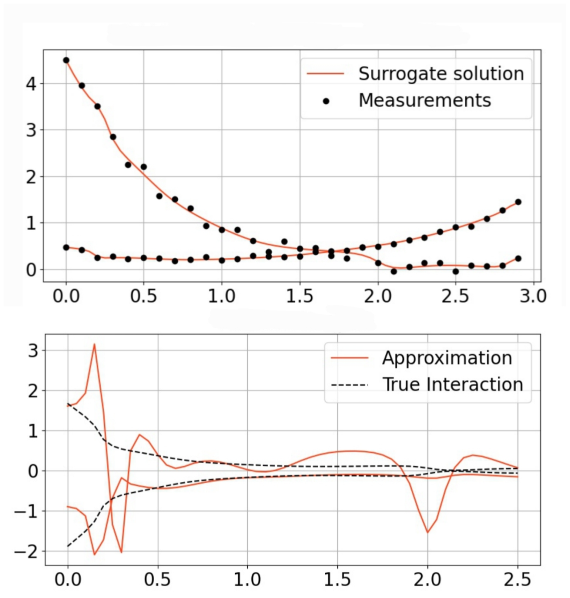

Figure 4. Noisy data regime. Reconstructed trajectory (top) and learned hidden interaction (bottom).

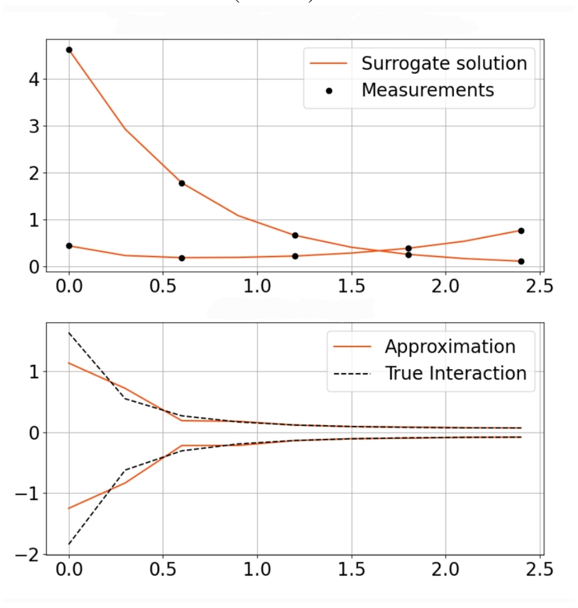

Figure 5. Sparse data regime. Reconstructed trajectory (top) and learned hidden interaction (bottom).

Figure 6. UDE method performance on the Lotka-Volterra model

This represents the predators' feeding habits as a function of prey population and its resulting effect on both the prey population and the predator's population. The actual form of these functions can take various forms in predator-prey models (see, for instance, (Harrison, 1979; Bolger et al., 2020)). While we initially modelled this as two unknown, decoupled functions F_1 and F_2 and learned them independently, we could also have modeled them by a single function with an additional learned parameter as a scaling factor. That is, we could take $F_1 = -\phi F_2$ and then only explicitly learn F_2 and a single parameter ϕ. This results in regressions that are near identical to the ones presented above, but showcases an important modelling methodology that our method is amenable to and, for more complicated models than LV, may be necessary in order to achieve a high-quality

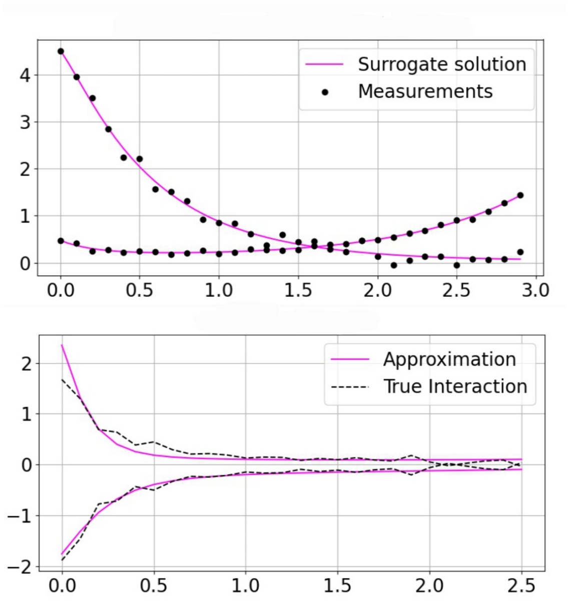

Figure 7. Noisy data regime. Reconstructed trajectory (top) and learned hidden interaction (bottom).

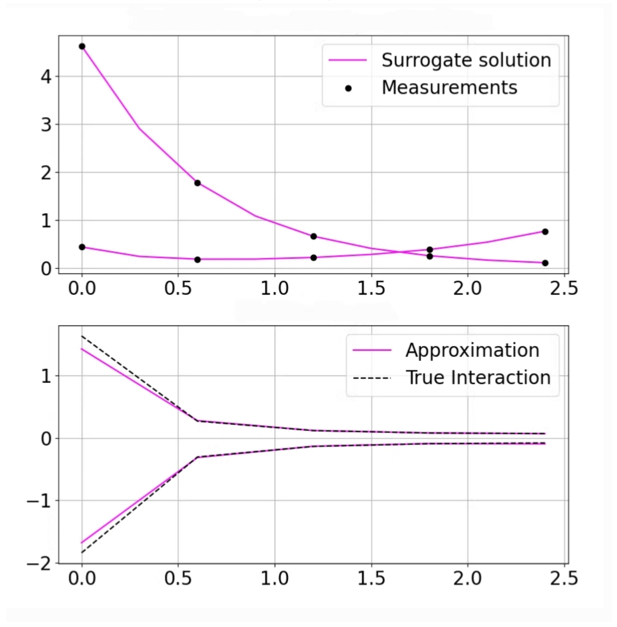

Figure 8. Sparse data regime. Reconstructed trajectory (top) and learned hidden interaction (bottom).

Figure 9. UPINN performance on the Lotka-Volterra model

regression.

4.2. Viscous Burger's Equation

Finally, our method is easily applied to PDEs (as in the original PINN implementation). Here we present the discovery

Table 3. Coefficients (with MSE) recovered by AI Feynman from the approximations of F_1, comparing over datasets (rows) and method of finding F_1 (columns). The true coefficient is -0.9.

spacing	noise level	F_1 (UDE)	F_1 (UPINN)
0.1	0	**-0.901 (2.8e-7)**	–
0.2	0	–	–
0.3	0	–	**-0.897 (4e-6)**
0.4	0	–	**-0.888 (8.2e-5)**
0.5	0	–	**-0.889 (8.9e-5)**
0.6	0	-0.892 (4e-3)	**-0.890 (1e-5)**
0.1	8e-3	-9.25 (1.8e-3)	**-0.906 (1e-5)**
0.1	1e-2	–	**-0.911 (3.45e-5)**
0.1	3e-2	–	**-0.960 (1e-3)**
0.1	5e-2	–	–
0.1	8e-2	–	–
0.1	1e-1	–	–

Table 4. Coefficients (with MSE) recovered by AI Feynman from the approximations F_2, comparing over datasets (rows) and method of finding F_2 (columns). The true coefficient is 0.8 for F_2.

spacing	noise level	F_2 (UDE)	F_2 (UPINN)
0.1	0	**0.802 (1.1e-6)**	0.797 (2.5e-6)
0.2	0	0.797 (3.4e-6)	**0.799 (3.8e-7)**
0.3	0	–	**0.798 (1.9e-6)**
0.4	0	–	**0.797 (5.2e-6)**
0.5	0	**0.760 (1e-3)**	–
0.6	0	–	**0.800 (1e-32)**
0.1	8e-3	–	**0.798 (3e-5)**
0.1	1e-2	**0.791 (2.3e-5)**	0.777 (1.5e-4)
0.1	3e-2	–	**0.777 (1.5e-4)**
0.1	5e-2	–	**0.740 (1.1e-3)**
0.1	8e-2	–	–
0.1	1e-1	**0.887 (2e-3)**	–

of both the solution to the PDE where the underlying hidden dynamics of the operator were partially hidden. This reconstruction used only noisy ($\epsilon = 5 \times 10^{-3}$) data obtained from two time points (the initial condition, $t = 0$, and a later time at $t = 0.5$). While this method can be used to discover the form of the boundary condition as well, here we assume that the homogeneous Dirichlet boundary conditions are known. The PDE in question is

$$\frac{\partial u}{\partial t} = -u\,\frac{\partial u}{\partial x} + \nu\,\frac{\partial^2 u}{\partial x^2},\;\; \nu = \frac{1}{1000\,\pi},\;\; u(x,0) = -\sin(\pi\,x)$$

Here we took $\mathcal{N}_{\mathcal{K}} = \nu\, u_{xx}$ and let the algorithm learn the hidden term $-u\,u_x$. To do this, we gave the F network u, u_x, and u_t as inputs. This represents an inductive prior where we are assuming that the hidden term depends on first order and lower derivatives of the solution. In our approach, such a prior is necessary (that is, the algorithm cannot learn what order of derivatives to include or not include, it can merely choose which inputs presented to it to utilize). For collocation data we used $n_P = 10^4$ and $n_B = 10^2$ points sampled from the appropriate parts of the domain $[-1, 1] \times [0, 1]$ via Latin hypercube sampling. The PDE solution was reconstructed with MSE of 3×10^{-4} and the hidden term was discovered with MSE of 2×10^{-2}. The resulting solution is visualized in Figure 10.

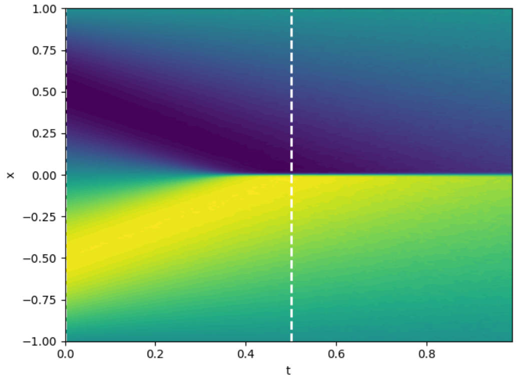

Figure 10. The reconstructed solution of Burgers' equation. The two vertical dashed white lines indicate the noisy experimental data that were sampled for the algorithm.

4.3. Cell Apoptosis Model

We also test the method on a biological application, which is the Q1 cell apoptosis model from (Wee & Aguda, 2006). This is an ODE with three variables, serine-threonine kinase Akt_s (active Akt), Akt (inactive Akt) and tumour suppressor protein $p53$. p53 promotes cell apoptosis, or programmed cell death, and Akt inhibits it. Here, we only focus on the system of ODEs and learning its nonlinear terms. We refer readers to the paper for the biological motivation and discussion. We denote the concentrations of p53, active Akt and inactive Akt as x, y, z respectively.

$$
\begin{aligned}
v_0 &= k_0 \\
v_1 &= k_1 \cdot z \cdot (j_1 + y) \\
v_{m1} &= \frac{k_{m1} \cdot y}{j_{m1} + y} \\
v_2 &= \frac{k_2 \cdot y \cdot x}{j_2 + x} \\
v_{m3} &= \frac{k_{m3} \cdot x \cdot y}{j_{m3} + y} \\
\frac{dx}{dt} &= v_0 - v_2 - k_d \cdot x \\
\frac{dy}{dt} &= v_1 - v_{m1} - v_{m3} \\
\frac{dz}{dt} &= \frac{-dy}{dt}
\end{aligned}
$$

All parameter values were taken from the paper. With the initial condition $(x, y, z) = (0.248, 0.0973, 0.0027)$ and 30 noiseless datapoints, both the v_1 and v_2 interactions (including the parameters) were learned to a high degree of accuracy. In Figures 11 and 12 it can be seen that although the general shape does not match the true interaction 100%, the mean squared error between the true interaction and the learned function is in fact very small (on the order of 10^{-4}) and the surrogate solution fits the data very well. This case study reveals a key trait of the method – in some DE's, the hidden interaction is not unique given a particular trajectory and data. Furthermore, when the derivatives of the trajectory are very small (as can be seen by the saturation past $t = 100$) the method can have difficulty learning the hidden term. As is done in (Yazdani et al., 2020), if this method were augmented to handle very small and very large values through scaling, more accurate learning of the interaction would be possible.

5. Conclusion

In conclusion, the Universal PINN approach is able to recover, with a great degree of accuracy, the symbolic functional form of hidden terms within a differential operator using very sparse measurements of noisy data. This approach is robust to both noise and sparsity of the data by increasing the number of collocation points (an operation that doesn't require any additional experimentation, just stronger compute capacities). This approach can be applied to discovering the functional form of an unknown ordinary differential equation (ODE) as well as both the functional form of a partial differential operator in a partial differential equation (PDE) and unknown terms in the boundary condition of a PDE. Although PINNs have been noted to perform sub-optimally on stiff equations without modification (Ji

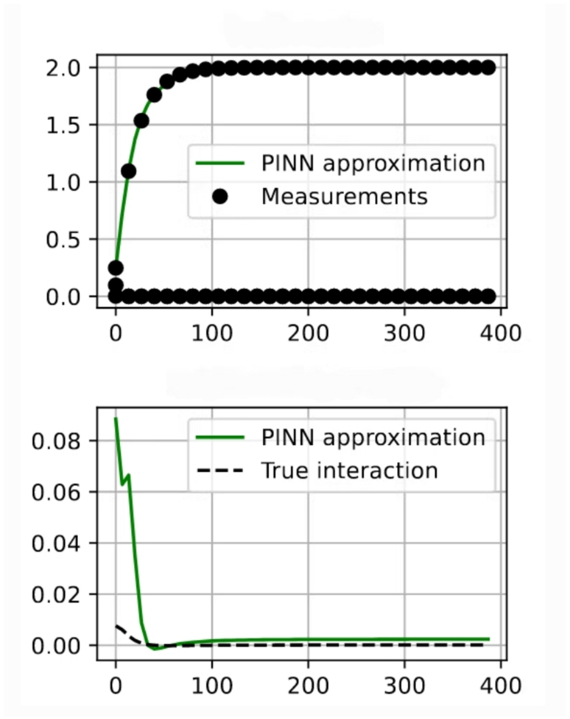

Figure 11. Learning the v_1 term using UPINNs. Reconstructed trajectory (top) and learned hidden interaction (bottom).

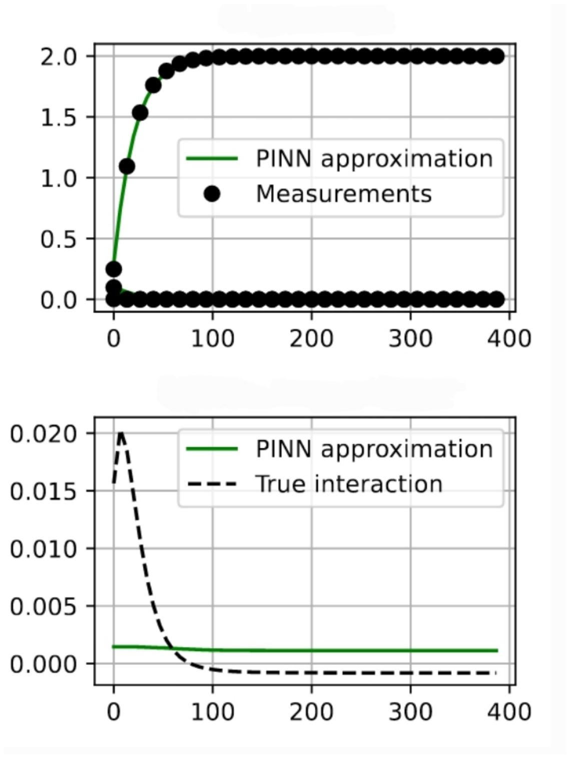

Figure 12. Learning the v_2 term using UPINNs. Reconstructed trajectory (top) and learned hidden interaction (bottom).

et al., 2021; Moya & Lin, 2021), we have noted promising results in this direction. However, more investigation is needed.

Software and Data

A GitHub link will be included with the "camera-ready" version of the manuscript.

References

Belohlav, Z., Zamostny, P., Kluson, P., and Volf, J. Application of random-search algorithm for regression analysis of catalytic hydrogenations. *The Canadian Journal of Chemical Engineering*, 75(4):735–742, 1997.

Berryman, A. A. The orgins and evolution of predator-prey theory. *Ecology*, 73(5):1530–1535, 1992.

Bishop, C. M. and Nasrabadi, N. M. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Bolger, T., Eastman, B., Hill, M., and Wolkowicz, G. A predator-prey model in the chemostat with holling type ii response function. *Mathematics in Applied Sciences and Engineering*, 1(4):333–354, 2020.

Chakraborty, S. Transfer learning based multi-fidelity physics informed deep neural network. *CoRR*, abs/2005.10614, 2020. URL `https://arxiv.org/abs/2005.10614`.

Harrison, G. W. Global stability of predator-prey interactions. *Journal of Mathematical Biology*, 8(2):159–171, 1979.

Ji, W., Qiu, W., Shi, Z., Pan, S., and Deng, S. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *The Journal of Physical Chemistry A*, 125(36): 8098–8106, 2021.

Kamienny, P.-A., d'Ascoli, S., Lample, G., and Charton, F. End-to-end symbolic regression with transformers. *arXiv preprint arXiv:2204.10532*, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lu, J., Bender, B., Jin, J. Y., and Guan, Y. Deep learning prediction of patient response time course from early data via neural-pharmacokinetic/pharmacodynamic modelling. *Nature machine intelligence*, 3(8):696–704, 2021a.

Lu, J., Deng, K., Zhang, X., Liu, G., and Guan, Y. Neural-ode for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens. *Iscience*, 24(7):102804, 2021b.

McKay, B., Willis, M. J., and Barton, G. W. Using a tree structured genetic algorithm to perform symbolic regression. In *First international conference on genetic algorithms in engineering systems: innovations and applications*, pp. 487–492. IET, 1995.

Meng, X. and Karniadakis, G. E. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems. *Journal of Computational Physics*, 401:109020, 2020.

Moya, C. and Lin, G. Dae-pinn: A physics-informed neural network model for simulating differential-algebraic equations with application to power networks. *arXiv preprint arXiv:2109.04304*, 2021.

Pinkus, A. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999. doi: 10.1017/S0962492900002919.

Rackauckas, C., Ma, Y., Martensen, J., Warner, C., Zubov, K., Supekar, R., Skinner, D., Ramadhan, A., and Edelman, A. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

Raissi, M., Perdikaris, P., and Karniadakis, G. 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, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Schmidt, M. and Lipson, H. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. doi: 10.1126/science.1165893. URL https://www.science.org/doi/abs/10.1126/science.1165893.

Udrescu, S.-M. and Tegmark, M. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

Valipour, M., Panju, M., You, B., and Ghodsi, A. Symbolicgpt: A generative transformer model for symbolic regression. In *Preprint Arxiv*, 2021. URL https://arxiv.org/abs/2106.14131. Under Review.

Wang, S., Teng, Y., and Perdikaris, P. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

Wee, K. B. and Aguda, B. D. Akt versus p53 in a network of oncogenes and tumor suppressor genes regulating cell survival and death. *Biophysical journal*, 91(3):857–865, 2006.

Yazdani, A., Lu, L., Raissi, M., and Karniadakis, G. E. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS computational biology*, 16(11):e1007575, 2020.