A PINN Approach to Symbolic Differential Operator Discovery with Sparse Data

Anonymous Author(s)
Affiliation
Address
`email`

1 Introduction

Mechanistic models, used for modeling real-world processes in biology, chemistry and physics [14, 2, 15], are in the form of a system of differential equations (DE). If correct values for the initial conditions and DE parameters are known, the DE can be used to interpolate between experimental datapoints, and predict the future state of a dynamical system. Many recent applications use NNs augmented with prior knowledge in order to learn underlying DE models[1] from data [5, 13, 9, 12, 10, 8]. 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) [13], and universal differential equations (UDE) [12]. In UDE, 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 [12] does not succeed in finding the true mechanistic model reliably. PINN assumes the form of the true DE and fits its parameters via a soft constraint (relaxing the requirement that NN should satisfy best-fit DE exactly), which is added to the NN loss function 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. Additionally, as iterative optimization is computationally expensive, and PINN loss can fail on stiff DEs [17].

Our approach bridges the limitations of both PINN (cannot be used when the structure of the DE is not fully known) [13] and UDE (not robust to noise and requires lots of data) [12]. To address this, we replace the hard constraint of the UDE with that of PINN loss, which allows the approach to learn both parameters and 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 algorithm [16] yielded good results in identifying the underlying DE model.

2 Method

Suppose $\vec{u}(\vec{x}, t) \in \mathbb{R}^m$ for $\vec{x} \in \mathbb{R}^d$. Let $\mathcal{N}$ be a (potentially non-linear) differential operator (that is, $\mathcal{N}$ is a function not only of $\vec{u}$, $\vec{x}$, and t but also of any derivatives of $\vec{u}$ with respect to $\vec{x}$. 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. We then consider problems of the form

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

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

subject to initial and boundary conditions

$$\vec{u}(\vec{x},0) = \vec{u}_0(\vec{x}), \quad \vec{x} \in \Omega, \quad \beta[\vec{u}](\vec{x},t) = 0, \quad \vec{x} \in \partial\Omega, \quad t \in [0,T]$$

where β is a (potentially non-linear) differential operator containing derivatives with respect to $\vec{x}$.

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 $\mathcal{F}[\vec{u}]$; the second network, $U(\vec{x},t;\theta_U)$, will approximate the value of $\vec{u}(x,t)$; and the third network, $B(\vec{u};\theta_B)$, will approximate the value of $\mathcal{B}[\vec{u}]$, the unknown target for the boundary condition. For all these networks we consider the architecture to be fully connected networks activated by the sigmoid function. To fit these networks, we consider another two sets of collocation points: these sets are $X_P = \{(\vec{x}_k, t_k)\}_{k=0}^{n_P-1}$ from the interior of the domain and $X_B = \{(\vec{x}_k, t_k)\}_{k=0}^{n_B-1}$ from the boundary of the domain. These sets correspond to locations in the space-time domain where we enforce that our network U satisfies the underlying differential equation and the boundary conditions.

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. 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 and the value $\mathcal{N}_K[U] + F(U)$.

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

This loss function is quite similar to the loss function for PINNs given in [13], 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 the domain. A similar construction works for selecting X_B. In this way, we are sampling the domain in a space-filling manner. In practice, increasing n requires acquiring data from a (potentially noisy) experiment. However, increasing n_P or n_B only costs extra computing power, as the data in X_P and X_B are just the $\vec{x}$-t points. In this regard, increasing n_P or n_B is effectively "free" from a modeling point of view (within reason), and lends the method increased accuracy without the need to acquire more data. As can be seen in the Results section, typically $n \ll n_P$ and $n \ll n_B$.

3 Results

3.1 Lotka Volterra

We begin our analysis by testing our method on the Lotka-Volterra (LV) model [3] of predator-prey interactions. The DE is formulated as follows:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha x - \beta xy$$
$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\delta y + \gamma xy \; .$$

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, the ODE is solved with the same parameters as in [12], and Gaussian noise is added to x_i and y_i proportionally to their means. For details see Appendix A.

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 and n_P. We want to show how the hard-to-acquire data (in D, of size n) can be augmented by taking more collocation points (X_P of size n_P) which require no experiments/measurements and comes 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 best realized when there is already ample enough experimental data for the algorithm to leverage.

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 1: Noise Free 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}

Table 2: Noisy ($\epsilon = 5 \times 10^{-3}$) Data

Table 3: Tables demonstrating the MSE between F and the true hidden target after training for various values of n and n_P.

Next, we compare our method's performance to the UDE method. We test the two methods on noiseless sparse data (Fig 1a), and on noisy data (Fig 1b). The error is computed with respect to the true interaction. At minimal noise level the UDE approach and PINN approach perform similarly, and for the densest data UDE slightly outperforms PINNs. Although increasing either noise or sparsity degrades the performance of both methods, the PINN method consistently attains a lower MSE compared to the UDE method as noise or sparsity increases.

Finally, AI Feynman symbolic regression[2] is run on the neural network output from both our approach and the UDE approach. In all cases, the NNs approximating F were given the training data as input. Then each NN's output was subsequently given to AI Feynman to find the best functional form. This data is presented in Table 4. 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 almost always more accurate.

The terms $\gamma\, x\, y$ and $-\beta\, x\, y$ in the LV equations correspond to the predator's uptake function in the ecology 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, [6, 4]). While we

[2]An algorithm which searches for the mathematical model that best fits a dataset, balancing model fit and simplicity

spacing	noise level	F_1 (UDE)	F_1 (PINN)	F_2 (UDE)	F_2 (PINN)
0.1	0	**-0.901 (2.8e-7)**	–	**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	–	**-0.897 (4e-6)**	–	**0.798 (1.9e-6)**
0.4	0	–	**-0.888 (8.2e-5)**	–	**0.797 (5.2e-6)**
0.5	0	–	**-0.889 (8.9e-5)**	**0.760 (1e-3)**	–
0.6	0	-0.892 (4e-3)	**-0.890 (1e-5)**	–	**0.800 (1e-32)**
0.1	8e-3	-9.25 (1.8e-3)	**-0.906 (1e-5)**	–	**0.798 (3e-5)**
0.1	1e-2	–	**-0.911 (3.45e-5)**	**0.791 (2.3e-5)**	0.777 (1.5e-4)
0.1	3e-2	–	**-0.960 (1e-3)**	–	**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)**	–

Table 4: Coefficients (with MSE) recovered by AI Feynman from the approximations F_1 and F_2, comparing over datasets (rows) and method of finding F_1 and F_2 (columns). True coefficients are -0.9 for F_1 and 0.8 for F_2. A dash indicates AI Feynman did not recover the functional form Cxy. The best performance is in bold.

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 regression.

3.2 Viscous Burger's Equation

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

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

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 4.

4 Conclusion

In conclusion, our 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 by utilizing a modification of PINNs. This approach is robust to both noise and sparsity of the noisy measurement 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 [7, 11], we have noted promising results in this direction. More investigation is needed, however.

References

[1] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[2] Zdenek Belohlav, Petr Zamostny, Petr Kluson, and Jiri Volf. Application of random-search algorithm for regression analysis of catalytic hydrogenations. *The Canadian Journal of Chemical Engineering*, 75(4):735–742, 1997.

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

[4] Tedra Bolger, Brydon Eastman, Madeleine Hill, and Gail Wolkowicz. A predator-prey model in the chemostat with holling type ii response function. *Mathematics in Applied Sciences and Engineering*, 1(4):333–354, 2020.

[5] Souvik Chakraborty. Transfer learning based multi-fidelity physics informed deep neural network. *CoRR*, abs/2005.10614, 2020.

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

[7] Weiqi Ji, Weilun Qiu, Zhiyu Shi, Shaowu Pan, and Sili Deng. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *The Journal of Physical Chemistry A*, 125(36):8098–8106, 2021.

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

[9] James Lu, Kaiwen Deng, Xinyuan Zhang, Gengbo Liu, and Yuanfang Guan. Neural-ode for pharmacokinetics modeling and its advantage to alternative machine learning models in predicting new dosing regimens. *Iscience*, 24(7):102804, 2021.

[10] Xuhui Meng and George Em Karniadakis. 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.

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

[12] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, Ali Ramadhan, and Alan Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[13] M. Raissi, P. Perdikaris, and G.E. Karniadakis. 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.

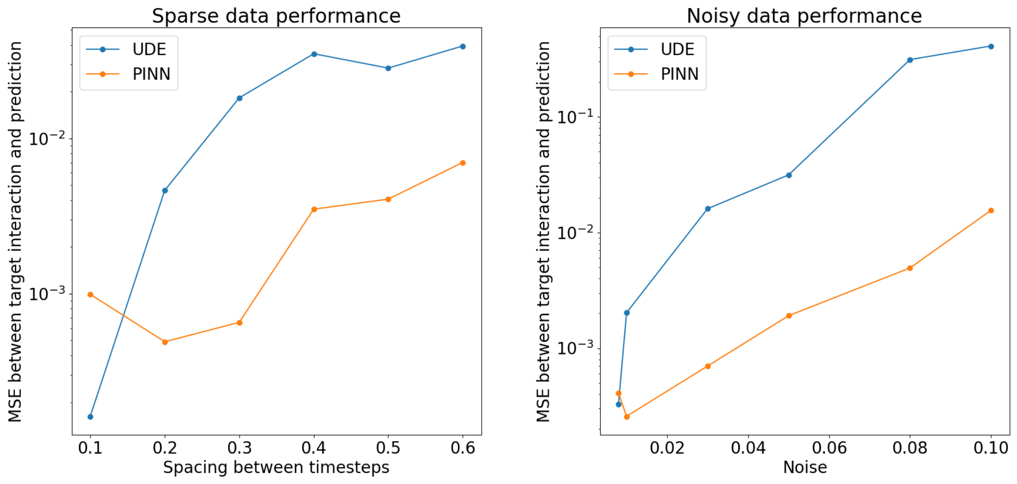
[14] Peter Ruoff, Melinda K Christensen, Jana Wolf, and Reinhart Heinrich. Temperature dependency and temperature compensation in a model of yeast glycolytic oscillations. *Biophysical Chemistry*, 106(2):179–192, 2003.

[15] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. CRC press, 2018.

[16] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020.

[17] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

A Appendix



(a) Sparse data (b) Noisy data

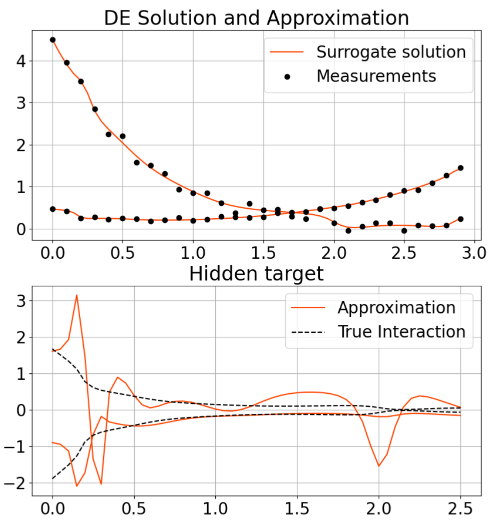
Figure 1: Mean squared error (MSE) of the recovery of the true interaction, comparison between UDE and PINN method. The spacing parameter determines how much time passes between data-points, but the overall time interval $[0,3]$ remains the same.

To generate the synthetic data for the LV equations, $\alpha, \beta, \gamma, \delta$ 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 [12]. 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 (sampled from the standard normal distribution) is added to each x_i and y_i. Given a particular noise level ϵ, the Gaussian noise added to the data follows:

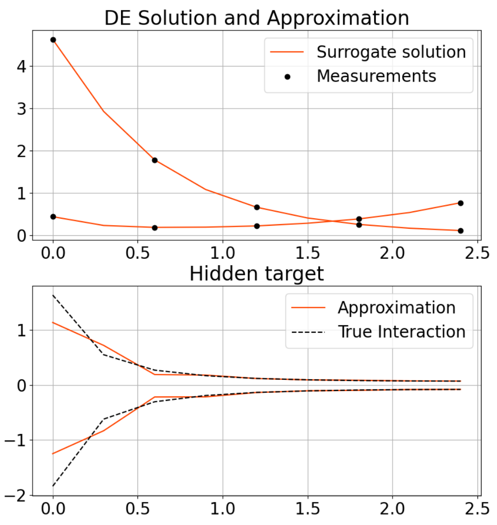
$$(x_i)_{noise} = x_i + \epsilon \cdot \bar{x} \cdot N(0,1)$$
$$(y_i)_{noise} = y_i + \epsilon \cdot \bar{y} \cdot N(0,1)$$

where $\bar{x}$ denotes the mean of x_i over all i, and similarly for y. Similarly, to generate the synthetic data for the Burgers' equation parameters were fixed and a solution was generated using FEniCS [1]. We then discretized the FEniCS solution to a grid of 256 equispaced x-points and 100 equispaced t-points in the domain. This data was then perturbed to add noise at the level of $\epsilon = 5 \times 10^{-3}$ as described above.

Figures 2 and 3 show the surrogate solution and hidden terms as recovered by the UDE and PINN methods. The noise level of the noisy data was set at 1e-1, and for the noiseless sparse data, there were 5 points each 0.6 apart. It is clear that the PINN approach is quite robust to noise, but performs well in low-data regimes. The UDE approach performs reasonably on sparse data, but is not robust to noise.

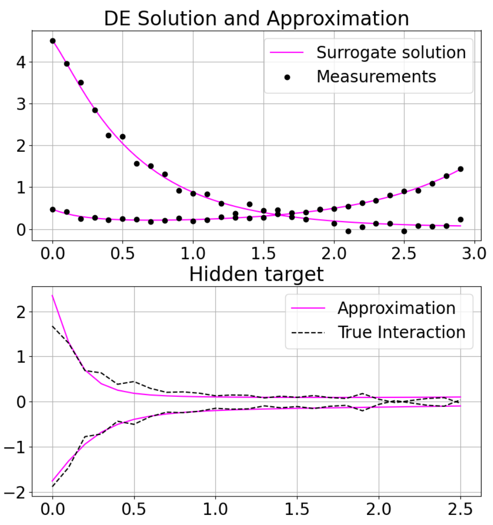


(a) Noisy data

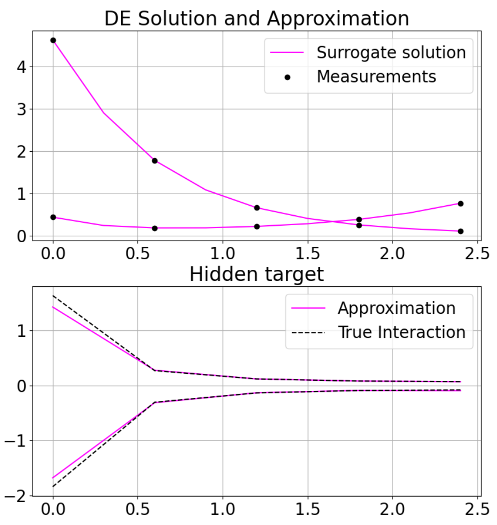


(b) Sparse data

Figure 2: UDE method performance



(a) Noisy data



(b) Sparse data

Figure 3: PINN performance

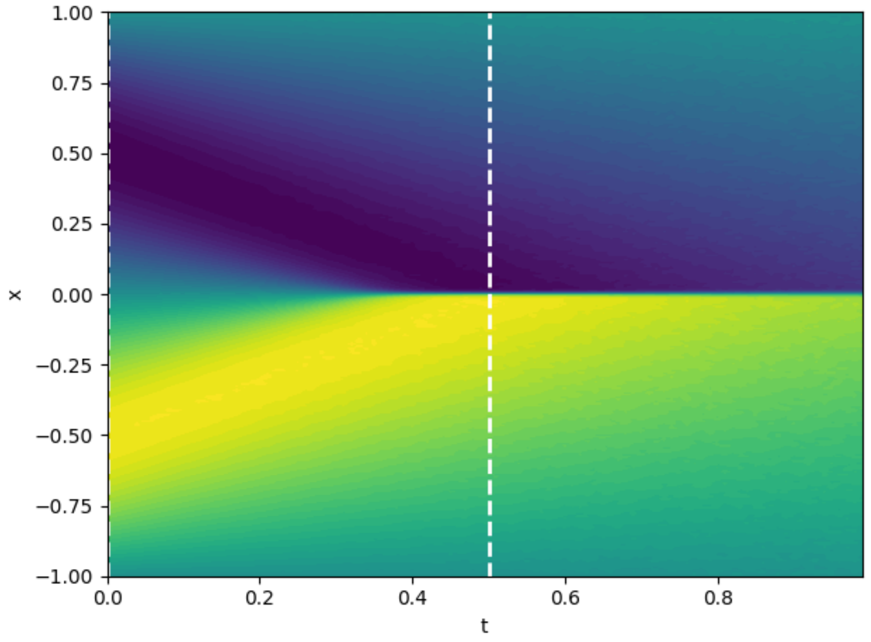


Figure 4: The reconstructed solution of Burgers' equation. The two vertical dashed white lines indicate the noisy experimental data that were sampled for the algorithm.