Summary of the paper: Universal Physics-Informed Neural Networks

Jefin Paul[1]

Abstract

This is the summary of the paper Universal Physics-Informed Neural Networks: Symbolic Differential Operator Discovery with Sparse Data by the authors Lena Podina, Brydon Eastman and Mohammad Kohandel. This paper is about a method called Universal Physics-Informed Neural Networks (UPINN). It discovers symbolic representations of differential operators when there is limited experimental data. In their work, they performed symbolic discovery of differential operators in situations where sparse experimental data occurs. This approach leverages prior knowledge about the underlying physical dynamics to address this issue. It provides first, a surrogate solution to the differential equation and then generates a black-box representation of the hidden terms. They have used three methods in their experiments, Lotka-volterra Model, viscous burger's equation and Cell apoptosis model. The hidden term neural networks were transformed into symbolic equations using AI Feynman technique in of their methods to reconstruct the symbolic expressions.

1. Introduction

This paper will be studied in terms of the introduction of the topic introduced by the authors, the exploration and their approach on various methods, the degree of improvement they have done over existing approaches, and later on continue by the comparison with eleven other topics discussed in the Advanced Machine Learning class, and, ultimately, the conclusion.

The authors introduce the utilization of Machine Learning algorithms, specifically Neural Networks (NN), for data-driven representation of unknown quantities. Neural networks, especially those with a wide hidden layer, have the capacity to approximate any function by fine-tuning their parameters. If the System's Differential Equation (DE) is available, these Neural Networks can analyze data and iteratively learn the optimal parameter values that closely align with the observed behavior.

The authors introduce two well-known methods for learning Differential Equations: Physics-Informed Neural Networks (PINN) and Universal Differential Equations (UDE). PINNs are a type of neural network-based method used for solving and learning solutions to partial differential equations. However, they have limitations when the structure of the DE is not fully known. On the other hand, UDE is a specific type of differential-algebraic equation with a unique property: its solution can closely approximate any continuous function on any interval of the real number line to any desired level of accuracy. However, UDE tends to struggle with noise and requires a substantial amount of data.

To overcome this, they proposed their approach, Universal PINNs (UPINNs), aims to overcome the limitations of both PINNs and UDEs. By substituting the rigid constraint of Universal Differential Equations (UDE) with the PINN loss, it becomes possible to learn the unknown components of the Differential Equation (DE) model directly from the available data. They have also used the AI-Feynman algorithm to one of their model to enhance the identification of underlying hidden terms in the DE model.

2. Methods

The proposed method, Universal Physics-Informed Neural Networks (Universal PINNs), presents a modification of Physics-Informed Neural Networks (PINNs) aimed at discovering the functional form of an unknown term within a differential equation.

In the paper introduced by (Raissi et al., 2019) The differential equation involves an unknown real-valued function $u(t, x)$ of time (t) and position (x). The time derivative of u is related to its value for each tuple (t, x) through a known function $\mathcal{N}$ with an unknown vector of parameters θ

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

The vector θ is required in order to find a numerical value but it is unknown. Time domain is from 0 to T. x is a vector representing spatial coordinates, and $x \in \Omega$ specifies that x

[1]University Jean Monney, Saint-Etienne, France. Correspondence to: Jefin Paul<jefin.paul@etu.univ-st-etienne.fr>.

belongs to the spatial domain Ω. $\Omega \in \mathbb{R}^D$ means that the spatial domain Ω is a subset of the D-dimensional Euclidean space $\mathbb{R}^D$, where D is the dimensionality of the space.

The goal is to estimate both the unknown function u and the parameters θ simultaneously using a method called PINN. The key component of this method is a neural network that predicts the function u for any given tuple. The authors describe a loss function(L) to train this neural network.

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

The first term in the above equation penalizes the neural network for making predictions that do not match the differential equation at a predefined set of M collocation points. The second term penalizes neural networks for making predictions that do not match with the data.

In the paper by Rackauckas et al., 2020, they introduced the concept of UDE, it is a DE which defined in part using universal approximators like Neural networks. Noisy data is represented as ti, xi, ui, where ti is the time, xi is the input, and ui is the observed output, which is available from equation 1. The solution $\mathcal{N}$ is expressed as a function g. This function is composed of k unknown functions h_i and known parameters θ, represented as:

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

The unknown functions h_i are approximated by a neural network H with k outputs. This neural network is trained using iterative optimization methods like Adam or gradient descent. The training process involves numerically solving the differential equation (Eq. 1) using the current approximation H. The error between the numerical solution and the observed data is computed, using a mean squared error. The neural network H is updated to improve its approximation of the unknown components of $\mathcal{N}$. This training loop is repeated iteratively to refine the neural network's approximation.

In the author's version, introduce a modified version of Physics-Informed Neural Networks (PINNs) called Universal PINNs. The goal of Universal PINNs is to discover the functional form of an unknown term within a differential equation. They consider a differential operator N that contains a potentially nonlinear differential term and explore problems of the form:

$$\frac{d\vec{u}(\vec{x}, t)}{dt} = N[\vec{u}](\vec{x}, t), \quad t \in [0, T], \quad \vec{x} \in \Omega \tag{4}$$

subject to the initial condition

$$\vec{u}(\vec{x}, 0) = \vec{u}_0(\vec{x}), \quad \vec{x} \in \Omega \tag{5}$$

and boundary conditions

$$\beta[\vec{u}](\vec{x}, t) = 0, \quad \vec{x} \in \partial\Omega, \quad t \in [0, T] \tag{6}$$

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

Furthermore, suppose

$$N[\vec{u}](\vec{x}, t) = NK[\vec{u}](\vec{x}, t) + F[\vec{u}](\vec{x}, t) \tag{7}$$

where NK is some differential operator with a known functional form, and F represents some unknown target differential operator. Similarly, suppose $\beta = \beta_K + B$ for some known β_K and some unknown B.

Finally, one can consider $\Omega = \emptyset$, 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 there are n data points $D = \{(t_k, \vec{x}_k, \vec{u}_k)\}_{k=0}^{n-1}$ where $\vec{u}_k = \vec{u}(t_k, \vec{x}_k) + \epsilon_k$ and ϵ_k is some noise term (potentially $\epsilon_k = 0$). These measured data are used to fit the parameters of up to three neural networks.

The authors propose using neural networks to approximate the unknown terms and functions in the differential equation. They introduce three neural networks:

1. $F(\vec{u}; \theta_F)$ approximates the target differential operator $F[\vec{u}]$.
2. $U(\vec{x}, t; \theta_U)$ approximates the solution $\vec{u}(\vec{x}, t)$.
3. $B(\vec{u}; \theta_B)$ approximates the unknown target for the boundary condition.

To fit these networks, another two sets of collocation points are considered. These sets are $XP = \{(\vec{x}_k, t_k)\}_{k=0}^{nP-1} \subset (\Omega \setminus \partial\Omega) \times (0, T]$ and $XB = \{(\vec{x}_k, t_k)\}_{k=0}^{nB-1} \subset (\partial\Omega) \times (0, T]$. These sets correspond to locations in the space-time domain where it is enforced that the network U satisfies the underlying differential equation (in the case of XP) and the boundary conditions (in the case of XB).

The training process involves fitting these neural networks using measured data points $D = \{(t_k, \vec{x}_k, \vec{u}_k)\}$, where $\vec{u}_k = \vec{u}(t_k, \vec{x}_k) + \epsilon_k$ includes a potentially noisy term ϵ_k. The networks are trained based on three sets of collocation points: XP for enforcing the underlying differential equation, XB for enforcing boundary conditions, and the measured data points D.

The loss function is defined as:

$$L(\theta_U, \theta_B, \theta_F) = LM(\theta_U) + LB(\theta_U, \theta_B) + LP(\theta_U, \theta_F) \tag{8}$$

The loss function $L(\theta_U, \theta_B, \theta_F)$ consists of three components:

- **MSE Loss (LM):** Measures the difference between the measured values and the neural network approximations at the data points.

- **Boundary Loss (LB):** Measures the mean squared value of the approximated boundary condition.

- **PINN Loss (LP):** Measures the mean squared error between the time derivative of the neural network approximation and the value predicted by the differential equation.

The loss function is used to minimize the errors between the neural network predictions and the actual solution, which ensures that the neural networks accurately represent the unknown terms in the differential equation and satisfy the given boundary conditions. An enhancement is introduced by incorporating two additional neural networks into the loss function. These networks correspond to the unknown parts of the underlying dynamics in the boundary conditions and the differential equation. To account for the additional parameters, the first component of the loss is extended to incorporate not only initial data but also solution data. This allows the set D to include data from the initial condition, data from the boundary, or data from the interior of the domain.

In practical terms, one approach for selecting XP is to simply choose np and utilize Latin hypercube sampling to select np points in $(\Omega \setminus \partial\Omega) \times (0, T]$. A similar construction is applied for selecting XB. This method involves sampling the domain in a space-filling manner.

Finally, the authors provide details about the architecture of the fully-connected neural networks used for different models, such as Burgers, Lotka-Volterra, and Apoptosis.

3. Result

To prove their method works well, they achieved high accuracy in finding hidden terms in three different scenarios: the Lotka-Volterra equations (which are a type of mathematical model for how populations of species change over time), the viscous Burgers' equation (a model for fluid dynamics), and a model for cell apoptosis (which is programmed cell death). They also demonstrated that their AI model, called AI Feynman, can correctly figure out the mathematical expressions hidden in the Lotka-Volterra model based on the model's output.

3.1. Lotka-Volterra System

The first ODE where the PINN is applied is the Lotka-Volterra Predictor-Prey model. Referenced (Berryman) They used the data shown in Figure 2 in order to fit the hidden terms.

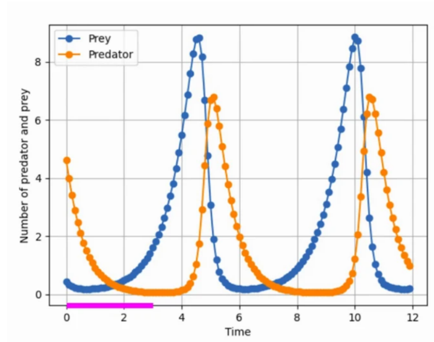


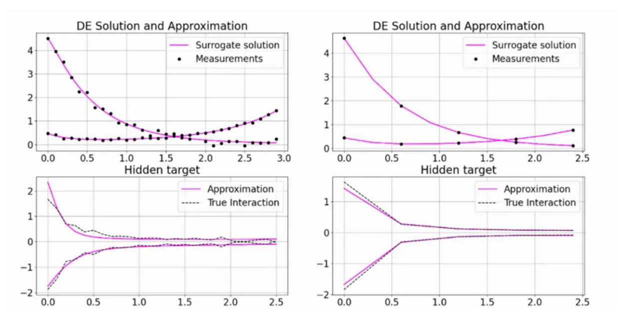
Figure 1. Data used for UPINN

The equation is given by

$$\frac{dx}{dt} = \alpha x - \beta xy \tag{9}$$

$$\frac{dy}{dt} = -\delta y + \gamma xy \tag{10}$$

The unknown term is passed to the neural networks. Here β and γ are the unknown terms which is denoted as F1 and F2 and they used two neural networks, F1 and F2 to model the hidden terms Beta and and gamma. Gaussian noise is added to the data. The analysis is conducted on both noise-free and noisy data for different numbers of data points and collocation points.



(a) Noisy data (b) Sparse data

Figure 2. Noisy and Sparse data regime. Reconstructed trajectory and learned hidden interaction.

They obtained good approximation for the data of the hidden terms in both noisy and sparse data

The analysis also includes the application of symbolic regression (AI Feynman) to find the symbolic or functional form for the hidden terms. The reconstructed symbolic expressions are shown in Table 1 and Table 2 for the hidden terms F1 and F2.

Table 1. Coefficients (with MSE) recovered by AI Feynman from the approximations of F1, comparing over datasets (rows) and method of finding F1 (columns). The true coefficient is -0.9.

Spacing	Noise Level	F1 (UDE)	F1 (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 2. Coefficients (with MSE) recovered by AI Feynman from the approximations F2, comparing over datasets (rows) and the method of finding F2 (columns). The true coefficient is 0.8 for F2.

Spacing	Noise Level	F2 (UDE)	F2 (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)	–

From the table 1 and table 2 we could see that their model UPINN is working better than UDE and they have got more reconstructed expressions than UDE.

3.2. Cell Apoptosis Model

This is another model that they tested on, in this case, they used two separate networks to approximate V1 and V2. Because here V1 and V2 are unknown, (all the equations and the parameters are obtained from the paper referenced (Wee & Aguda, 2006)) This is an ODE with three variables, serine-threonine kinase Akts (active Akt), Akt (inactive Akt) and tumour suppressor protein p53. p53 promotes cell apoptosis, or programmed cell death, and Akt inhibits it. They denote the concentrations of p53, active Akt and inactive Akt as x, y, z espectively.

After solving for the equation shown from 11 to 18. The result are obtained as shown in the figure 3

$$v_0 = k_0 \tag{11}$$
$$v_1 = k_1 \cdot z \cdot (j_1 + y) \tag{12}$$
$$v_{m1} = k_{m1} \cdot \frac{y}{j_{m1} + y} \tag{13}$$
$$v_2 = k_2 \cdot \frac{y \cdot x}{j_2 + x} \tag{14}$$
$$v_{m3} = k_{m3} \cdot \frac{x \cdot y}{j_{m3} + y} \tag{15}$$
$$\frac{dx}{dt} = v_0 - v_2 - k_d \cdot x \tag{16}$$
$$\frac{dy}{dt} = v_1 - v_{m1} - v_{m3} \tag{17}$$
$$\frac{dz}{dt} = -\frac{dy}{dt} \tag{18}$$

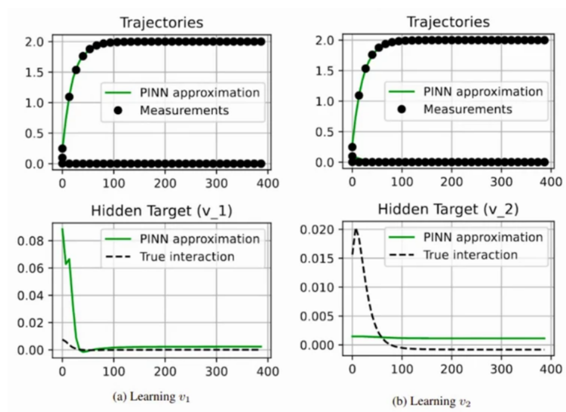


Figure 3. Learning the v1 term using UPINNs. Reconstructed trajectory (top left) and learned hidden interaction (bottom left). and the top right and bottom right shows the similar approach for v2 term.

In Figures 3, it can be observed 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 differential equations (DEs), 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.

3.3. Viscous Burger's Equation

Finally, they applied their method on the Burgers equation, they used only noisy data here, finding a hidden term from the Burgers equation.

The authors showcase the discovery of the solution to the partial differential equation (PDE) where the underlying hidden dynamics of the operator were partially hidden. This reconstruction utilized only noisy ($\epsilon = 5 \times 10^{-3}$) data obtained from two time points (the initial condition, $t = 0$, and a later time at $t = 0.5$).

The partial differential equation is given by:

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} + \frac{1}{1000\pi}\frac{\partial^2 u}{\partial x^2}, \tag{19}$$

where $\nu = \frac{1}{1000\pi}$, and the initial condition is $u(x, 0) = -\sin(\pi x)$.

They took $NK = \nu u_{xx}$ and let the algorithm learn the hidden term $-uu_x$. This was done by providing the F network with inputs u, u_x, and u_t.

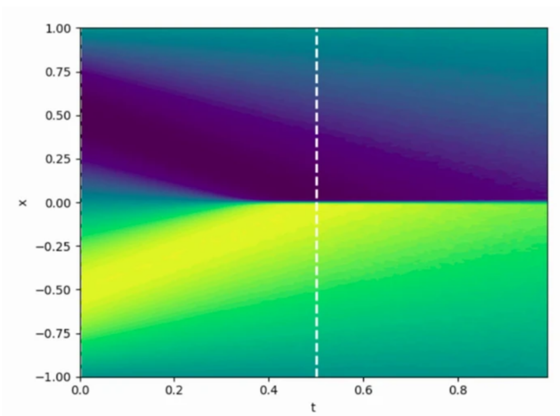


Figure 4. The reconstructed solution of Burgers' equation. The vertical dashed white lines indicate the noisy experimental data that were sampled for the algorithm

For collocation data, the authors utilized $n_P = 104$ and $n_B = 102$ points sampled from the relevant parts of the domain $[-1, 1] \times [0, 1]$ via Latin hypercube sampling. The partial differential equation (PDE) solution was reconstructed with a mean squared error (MSE) of 3×10^{-4}, and the hidden term was discovered with an MSE of 2×10^{-2}.

4. Conclusion by authors

In their conclusion, the Universal PINN approach demonstrates the capability to accurately recover the symbolic functional form of hidden terms within a differential operator The method exhibits robustness to both noise and sparsity of the data by increasing the number of collocation points.

5. Topics integration/relation with the paper

In this section, we will discuss about the relation of the paper with the topics that are closely related or how we can integrate them into this model, three topics that closely related are Physics-Informed Machine Learning, Reinforcement Learning and Expressiveness of Recurrent Neural Networks.

5.1. Physics-Informed Machine Learning

This paper is itself about Physics Informed Neural network which is coming under the concept Physics Informed Machine Learning(PIML). Physics Informed Machine Learning is introduced as a way to leverage prior knowledge of physical systems during the training of machine learning models. The key idea of physics-informed machine learning is to constrain the data according to known physical models, reducing the randomness in the training data. It significantly reduce the number of training samples required, making the training process more efficient. In many real-world problems, the systems generating data are governed by underlying physical laws or principles. Thus in the cases where there are less data, we can make the model to align with the known physics of the system and it then can be integrated into machine learning models.

5.1.1. Adding Noise to the Data of Methods using Fourier Transform

We could use the Fourier transform to generate noise by simplifying the analysis of differential equations by transforming them into algebraic equations in the frequency domain. In the paper, for the Lotka-Volterra System, the authors have used Gaussian Noise, which they have added to the data, and they have added the noise to make it more realistic to real-life problems. We could also add the noise by using Fourier transform. We can consider incorporating noise into our data by perturbing the true solution or measurements. For example if there exists a true solution u_true(x, t) , then we can generate noisy data by adding random noise.

$$u_{\text{noisy}}(x, t) = u_{\text{true}}(x, t) + \text{noise} \tag{20}$$

Here, the noise can be generated using a random process or by applying a Fourier transform to the true solution and adding appropriately scaled random values. We could convert the the first methods that they have used into Fourier Transform. Then we could add random noise by Introducing random noise to the Fourier coefficients. This step can be done by multiplying the Fourier coefficients by a complex number with a random phase while keeping the magnitude fixed. The amplitude of the noise can be controlled by adjusting the magnitude of the complex numbers. Then we could convert it back to original equation by applying inverse Fourier transform.

5.1.2. Context-informed dynamics adaptation(CoDA) on UPINN

CoDA, as a data-driven approach, struggles to generalize to unseen systems with similar dynamics but different contexts. This challenge may arise in cases involving external forces, spatio-temporal conditions, boundary conditions, sensor characteristics, and system parameters. The key aspect is its ability to leverage multiple environments, each associated with a different dynamic, and to condition the dynamics model on contextual parameters specific to each environment. This is achieved by employing a hypernetwork to condition the dynamics model, learned jointly with a context vector obtained from observed data. Such an approach allows for better adaptation to new dynamics and improved generalization across environments, even with limited samples. To extend or integrate CoDA into the PINN framework and include context parameters, we can first define the architecture of the physics-informed neural network (PINN). This architecture encompasses the input, output, and hidden layers of the neural network, incorporating the differential equations that describe the underlying dynamics. These parameters should ideally capture variations in physical contexts, such as external forces, boundary conditions, or other relevant factors specific to each environment.

We could also integrate the CoDA mechanism into the PINN architecture. This involves adding a hypernetwork that conditions the dynamics model on context parameters. We then would have to design the hypernetwork to learn the relationship between the context parameters and the dynamics model. The hypernetwork's weights can be adjusted to adapt the model to different contexts. We could start this by collecting training data from multiple environments. Each associated with a different set of context parameters. This data should include observations of the system's behavior under various conditions and then we have to annotate them with context parameters for each environment.

As for Loss function modification, we could modify the loss function used in the PINN to include terms that encourage the adaptation of the dynamics model to different contexts. This modification ensures that the network not only fits the observed data but also generalizes well across diverse environments. We could also add regularization terms that penalize deviations from context-specific dynamics, promoting robustness and adaptability.

Finally, we could do some training, using the modified loss function and the training data from multiple environments. Ensuring that the training process should optimize both the neural network weights and the hypernetwork parameters.

5.2. Reinforcement learning(RL)

Reinforcement learning is a type of machine learning paradigm where an agent learns to make decisions by interacting with an environment. The agent takes action, and the environment provides feedback in the form of rewards or penalties. The objective of the agent is to learn a strategy (policy) that maximizes the cumulative reward over time. The proper tuning of hyperparameters is essential for successful RL training such as Learning Rate, discount factor, exploration and exploitation trade-off, etc.

5.2.1. Integrating RL with UPINN

Reinforcement Learning has wide variety of applications in Deep learning and machine learning, we could implement Reinforcement learning with UPINN. The paper (Martin & Schaub, 2022) gives an idea about how the PINN and RL has been integrated, The primary focus is on devising an ”Enhanced Safe Mode” for a spacecraft engaged in orbital manoeuvres around a small, irregularly shaped celestial body, such as an asteroid. The objective here is to train an RL agent, termed ”Enhanced Safe Mode,” to adeptly guide the spacecraft into a secure and stable orbit when confronted with the activation of Safe Mode. The agent is designed to prioritize three critical safety objectives avoiding collisions with the asteroid, conserving fuel resources, and ensuring the spacecraft maintains sufficient proximity to the asteroid.

The RL environment is formulated as a Markov Decision Process (MDP), encompassing a state space that includes parameters like position, velocity, and remaining fuel. The reward is defined as avoiding collisions, and fuel depletion and rewarding the spacecraft for maintaining a safe distance from the asteroid. For the RL training process, the Soft Actor-Critic (SAC) algorithm is selected. For the training Enhanced Safe Mode agent through RL is used and three differnt scenarios are considered. One employs hybrid approach, using the PINN.

The PINN Gravity Model is trained to understand and predict the gravitational forces acting on the spacecraft in orbit around a small body, like an irregularly shaped asteroid. The PINN Gravity Model is integrated into the RL training process. During RL training, the agent interacts with the environment, and the PINN Gravity Model is used to provide accurate gravitational information for the simulations. This helps the RL agent learn and make decisions based on realistic gravitational effects without sacrificing computational speed.

We could implement similar approach into UPINN by first defining the RL problem that we want to solve, this involves specifying the environment, states, actions, rewards, and the goal. Then we will have to design RL framework for implementing the RL algorithm, and decide how to represent the state in RL problem. Here, maybe we can involve using the output of UPINN as part of the state representation. We could define action space. Designing a reward function that incorporates both the RL objectives and the physics-informed constraints.

Then in the RL training loop, enforce physics-informed constraints using the PINN or UPINN. This involves adding terms to the loss function that penalize deviations from the physics equations. We could then implement a joint training, which trains the combines system using a joint optimization approach. This involves updating the weights of both the RL policy and the neural network approximating the physics-based solution. Then we can implement the training provess, which Iterate between RL training and physics-informed training. This may involve alternating between policy updates and updates to the UPINN parameters. Finally we could evaluate the performance of integrated RL and physics-informed model.

5.2.2. Implementing UPINN with RL for the Lotka-Volterra model

First we have to specify the state, action, and reward components. Then we have to define the state space which could be prey population (x) and predator population (y). Action space can be actions that the RL agent can take to influence the predator-prey interactions. This could involve controlling certain aspects of the environment or the LV model. The reward components can be population dynamics, where we could reward the RL agent based on the stability and desired dynamics of the prey and predator populations, or balance which could encourage the agent for maintaining a certain ratio or balance between prey and predator populations. The learning process can be defined as when the agent could receive a reward for reducing uncertainty in its predictions or for discovering accurate hidden interaction terms.

Then we could integrate the UPINN model as a component of the RL agent. UPINN will serve as a tool for learning the hidden interaction terms and dynamics of the Lotka-Volterra model from the data generated during RL training. Then we have to train the RL agent with RL algorithms. We could design exploration strategies within the RL algorithm to allow the agent to explore different actions and learn more about the system by exploiting the learned knowledge from UPINN to make informed decisions and optimize the predator-prey interactions and finally evaluation. As an add-on, we could also apply symbolic regression techniques to extract symbolic expressions representing the learned interactions from the UPINN model.

5.3. Expresiveness of RNN

Here, expressiveness refers to the ability of the network to capture and represent complex relationships and patterns in sequential data. The expressiveness of an RNN is crucial because it determines the range of functions or computations that the network can approximate.

A Recurrent Neural Network (RNN) is a type of artificial neural network designed for processing sequences of data. Unlike traditional feedforward neural networks, RNNs are focused on Temporal dependencies, which is dependencies over time, that is our current output does not only depend on current input but also in past inputs. One of the disadvantages of RNN is the difficulty of learning long-term dependencies. Traditional RNNs struggle to retain information over many time steps, which can result in a loss of context and hinder performance on tasks requiring understanding of long-range dependencies.

There are many advanced types of RNN. One of them is Long Short-Term Memory(LSTM), it helps capture the long-term memory of data and it addresses the vanishing gradient problem.

Gated Recurrent unit (GRU), itis another variant of RNN that simplifies the architecture compared to LSTM. It combines the memory cell and hidden state, resulting in a more streamlined structure.

Probabilistic Deterministic Finite Automaton(PDFA) is a type of weighted automaton designed for modeling sequences of data. It explicitly defines states, transitions, and weights, providing an interpretable representation.

We could integrate RNN, their advanced methods in UPINNs, there are several paper available where ODE or PDE and the dynamics of physics is integrated with RNN.

5.3.1. Background on PINN with RNN

Physics-informed recurrent neural network (PIRNN) developed by (Zheng et al., 2023). The authors aim to develop a modelling framework that integrates data-driven and physics-informed techniques to construct high-fidelity recurrent neural network (RNN) models. They derived a generalization error bound for the PIRNN model. This involves assessing how well the model can adapt to previously unseen data of the same distribution, considering a nominal system model. The PIRNN model is then used in a model predictive control (MPC) scheme. MPC is an optimization-based control method that finds optimal control actions based on a process model while considering the dynamic behaviour of the system. The researchers created a special type of computer program called a physics-informed recurrent neural network (PIRNN). This program combines real-world data with mathematical models to better learn and understand complex systems. They also developed a way to control nonlinear systems (systems with complex interactions) using this PIRNN.

5.3.2. Background on PINN with LSTM

In the paper by (Lahariya et al., 2022), they did a project to model and control the flexibility in an evaporative cooling system, which is part of energy-intensive industrial processes. The goal is to use machine learning models to better understand and utilize the flexibility of these systems. They integrated PINN with LSTM to model the relationship between control inputs and system responses while respecting the physical constraints of the process.

They create two types of models: Physics Informed Neural Networks (PhyNN) and Physics Informed LSTM (PhyL-

STM). They also introduce two metrics, State of Charge (SoC) and Rate of Charge (RoC), to quantify the operational flexibility of the cooling system. SoC represents the system's state of operation, while RoC measures the transition speed between different states.

They use two types of models - black-box models (Neural Networks and LSTMs) and grey-box models (Physics Informed Neural Networks and Physics Informed LSTMs) to approximate the time-varying relationship between the control inputs (like fan power) and system responses (like basin temperature).

The performance of the PhyLSTMWF improves with longer training data window, as opposed to PhyNN.

5.3.3. Integrating RNN or LSTM with UPINNs

This can be a powerful approach to solve differential equations. After identifying the problem and the physical context and the dynamics involved in the problem. We formulate the ODE/PDE that describes the physical system. Specify the known terms and the unknown terms in the equations. Then we have to design the architecture or integrate the network for example, feeding the data through LSTM and using its output as input feature for UDE coming inside UPINN. Then we validate and test the model. We could take the method used in the paper, like Burger's equation, to solve them. The Burgers' equation is given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 \tag{21}$$

Since they are already depending on time, the presence of the partial derivative with respect to time in the Burgers' equation indicates that the equation involves temporal dynamics. We could implement this as sequential data for LSTM networks. The sequential nature of the data corresponds to the evolution of the system over time. Each time step represents a snapshot of the system, and the LSTM can capture the temporal dependencies in these sequences of snapshots.

So we could take the input data as each input sequence corresponds to the spatial distribution of the variable u at different points along the x-axis. Each time step could be system at specific point in time. Maybe how the behaviour of viscous fluid change over time. The target could be the next step in time for same spatial points. The goal is to predict how the system evolves from one time step to the next. LSTM is trained to learn the temporal patterns and loss function is designed to ensure that predicted sequences adhere to the dynamics described by Burgers' equation. After training, the LSTM can be used to predict the evolution of the system over time.

Since we are implementing hybrid architecture, the model could get more complex and might take a lot of time for training. Combining different types of neural networks requires careful tuning of hyperparameters to achieve optimal performance. Finding the right set of hyperparameters for both LSTM and UPINN components is quite challenging.

5.4. Advanced Kernel Methods

The topic and paper are not related concepts. But we could use Kernel Methods in the paper. Combining Advanced Kernel methods and Physics-Informed Neural Networks (UPINN) can be a powerful for solving complex problems. Kernel methods are great at understanding intricate patterns in data, especially when dealing with lots of features. They use a clever trick to map data into a space where patterns are easier to see. On the other hand, Neural Networks, like UPINN, are versatile learners that can understand complex relationships directly from the data. Neural networks can benefit from kernel methods by utilizing the implicit feature mappings provided by kernels. This can enhance the ability of neural networks to learn representations from the data. It will allows the model to capture both explicit and implicit features in the data, potentially improving its expressiveness and ability to model intricate relationships.

5.5. Statistical Learning Theory - Generalization Bound

Generalization bounds are theoretical tools used to analyze the performance of machine learning models on unseen data. It provides theoretical guarantees on the model's ability to generalize to new, unseen data. Though both topics are not related, the combination can benefit from both the theoretical assurances of generalization bounds and the physical consistency provided by UPINN. Also, it could improve the generalization performance on unseen data when combined with UPINN.

5.6. Online Learning

In Online learning we can operate sequentially, and adapting to new data points as they arrive. It refers to the incremental learning paradigm where a model is updated continuously as new data points arrive. Which is not related to the paper because the paper discuses about using Physics dynamics and concentrates on the application of UPINN to solve differential equations in a physics-informed manner applying neural network to them. They both address different aspects of learning rate. We could maybe adapt UPINN to take model to changing or streaming data over time, making it more responsive to dynamic scenarios

5.7. Proximal Splitting Algorithm

A proximal splitting algorithm is a type of optimization algorithm used for solving convex optimization problems, especially those were nonsmooth functions are involved. The lecture slides consists of different type of algorithms to solve optimization problems. Solving physics-informed problems often requires optimization techniques to train neural networks effectively. There could be chance of generally using subgradient Algorithm for optimization when the objective function is not necessarily smooth especially when dealing with UDE's. Though in the paper they have mentioned about using Adam optimizer, they have different purposes and are mostly applied to different types of problems.

5.8. Optimal Transport

Optimal Transport and Universal Physics-Informed Neural Networks (UPINN) are not directly related concepts, and they belong to different areas of mathematics and machine learning. Optimal Transport, also known as Monge-Kantorovich Transport or Wasserstein distance, is a mathematical framework for quantifying the optimal way to transport mass from one distribution to another while minimizing the associated cost and the goal here is to find the most efficient and cost-effective way to transform one distribution into another, which is not the problem-case for this paper but maybe we could integrate them.

5.9. Sparsity Inducing Norms

The main idea of this topic is about regularization techniques that encourage the learning algorithm to prefer sparse models. Here the model should have fewer non-zero parameters or features. The goal is to generate simpler models with fewer active components, which can lead to improved generalization performance, interpretability, and high computational efficiency. The topic is not related to the paper and is not implemented in the paper. However, implementing sparsity-inducing norm could encourage models to have a smaller number of non-zero parameters. We could use it when we want more interpretable or parsimonious model.

5.10. Transfer Learning

Transfer Learning and UPINNs serve different purposes. Transfer learning involves training a model on one task and then using the knowledge gained to improve performance on a related but different task. We could integrate these concepts in the paper. For example, in a physics-based problem with limited data in a specific domain (target domain), we could use transfer learning to leverage knowledge from a related physics domain (source domain). The UPINN approach could then be applied to refine the model's understanding of the system's dynamics based on the limited data available in the target domain.

6. Conclusion

The UPINN is implemented with the integration of Universal Differential Equation and Physics- Informed Neural Network, this method can accurately identify hidden terms from sparse and noisy data, it has a wide variety of applications in ecology cell biology, and Performs well on PDEs as well as ODES. The key takeaway is that UPINNs demonstrate strong performance even when given very limited and noisy data in both ordinary and partial differential equation contexts. The method, thus combines neural networks, physics-informed techniques, and symbolic regression to discover symbolic representations of differential operators, which can be useful in situations with sparse experimental data.

Universal Physics-Informed Neural Networks (UPINNs) stand as a powerful and versatile approach in the realm of scientific computing and machine learning. The integration of UPINNs brings forth a unique fusion of physics-informed constraints and neural network-based architectures, allowing for the effective incorporation of domain knowledge into the learning process. At the conclusion of the paper, a comprehensive comparison was drawn between the content presented and other relevant topics, including Physics-Informed Machine Learning, Reinforcement Learning, and expressiveness of Recurrent Neural Networks (RNNs),and concisely with respect to the other subjects.

References

Berryman, A. A. The origins and evolution of predator-prey.

Lahariya, M., Karami, F., Develder, C., and Crevecoeur, G. Physics informed lstm network for flexibility identification in evaporative cooling systems. *Journal Name*, 2022. https://arxiv.org/pdf/2205.09353.pdf.

Martin, J. R. and Schaub, H. Reinforcement learning and orbit discovery enhanced by small-body physics-informed neural network gravity models. 2022. URL https://hanspeterschaub.info/Papers/Martin2022.pdf.

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: 10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

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.

Zheng, Y., Hu, C., Wang, X., and Wu, Z. Physics-informed recurrent neural network modeling for predictive control of nonlinear processes. *Journal of Process Control*, 128: 103005, 2023. doi: 10.1016/j.jprocont.2023.103005.