Project Report

LEARNING CONTINUOUS-TIME PDES USING MESSAGE PASSING GRAPH NEURAL NETWORK

Abhay Kumar
University of Wisconsin-Madison
Madison, WI, 53715
abhay.kumar@wisc.edu

December 15, 2020

ABSTRACT

With the growing ease of collecting and efficiently storing huge data, there has been increased efforts to explore new opportunities for data-driven discovery of dynamical systems. Several machine learning methods have been proposed to predict dynamics of complex systems and to uncover the underlying hidden PDE models directly from data. Most of these methods are are limited to discrete-time approximations or make the limiting assumption of the observations arriving at regular grids. This report explores a general continuous-time differential model to learn dynamical system PDEs (proposed by Iakovlev et al. [1]). This model assumes an arbitrary space (observation points) and time discretizations and parameterizes the dynamical system using the Message Passing Neural Network (MPNN). The report presents the model's ability to work with unstructured grids, arbitrary time steps, and noisy observations and visualizes the true and learned dynamics for Convection-diffusion and heat equation PDEs. **CODE at** `https://github.com/abhayk1201/799_Project/tree/main/GNN_PDE`

1 Introduction

Partial Differential Equations (PDEs) plays a prominent role in studying the behaviour of many dynamical systems. PDEs are commonly derived based on empirical observations. Many complex systems in modern applications (like in space science, climate science, neuro-science, finance, etc.) still have eluded mechanisms, and the governing equations of these systems are only partially known. However, with the growing ease of collecting and efficiently storing huge data offers new opportunities for data-driven discovery of physical laws. Several machine learning methods have been proposed to accurately predict dynamics of complex systems and to uncover the underlying hidden PDE models directly from data.

1.1 Related Work

Earlier attempts on data-driven discovery of hidden physical laws were mainly based on comparing numerical differentiations of the experimental data with analytic derivatives of candidate functions, and applying the symbolic regression and the evolutionary algorithm to determining the nonlinear dynamical system. Many approaches constructed a dictionary of simple functions and partial derivatives that were likely to appear in the unknown governing equations. Rudy et al. [2] has proposed sparse regression based approach to select the candidates that most accurately represent the data. Raissi et al. [3] proposed a framework to learn the unknown parameters for non-linear dynamical systems when the form of the nonlinear response of a PDE is known. de Bezenac et al. [4] studied the problem of sea surface temperature prediction, assuming advection-diffusion equation as the underlying physical model and designed special neural network according to the general solution of advection-diffusion equations.

However, these models are computationally expensive and does not scale very well to large systems. The sparse regression method requires to fix certain numerical approximations of the spatial differentiations in the dictionary

beforehand, which limits the expressive and predictive power of the dictionary. In many approaches, the explicit form of the PDEs is assumed to be known except for a few scalar learnable parameters. Therefore, extracting governing unknown PDEs directly from data remains challenging and unexplored. Long et al. [5] has presented a method to identify the governing PDE models using properly constrained filters (These constrains are carefully designed by fully exploiting the relation between the orders of differential operators and the orders of sum rules of filters) using convolutional neural network (CNN). Recently, many machine learning based approaches like residual CNNs [6], symbolic neural networks [7], high-order auto-regressive networks [8], and feed-forward networks [9] have been proposed to learn the dynamical systems. However, these methods assumes discrete-time approximations and observations at regular grids, rendering them unable to handle temporally or spatially sparse or non-uniform observations commonly encountered in realistic applications.

The paper [1] proposes a more generic continuous time dynamic differential model (dynamic relational graphs instead of grids), which works for arbitrary space and time discretizations. It achieves efficient graph representation of the domain structure using the method of lines with message passing neural networks. The details are presented in the following sections.

2 Method Description & Techniques Used

Iakovlev et al. [1] has proposed an approach to learn dynamical systems PDEs using arbitrary space and time discretizations. Assume that the continuous dynamical systems with a **state** $u(x,t) \in R$ where, $t \in \mathbb{R}+$ and bounded domain $\mathbf{x} \in \Omega \subset \mathbb{R}^D$ is governed by an unknown partial differential equation (PDE)-

$$\dot{u}(\mathbf{x},t) = \frac{du(\mathbf{x},t)}{dt} = F(\mathbf{x}, u, \nabla_{\mathbf{x}} u, \nabla^2_{\mathbf{x}} u, \ldots) \tag{1}$$

From (1), we can observe the general structure that the temporal evolution $\dot{u}$ of the system depends on the current state u and its spatial partial derivatives w.r.t. the coordinates $\mathbf{x}$. Such PDE models are frequently observed and are widely applicable to modelling of propagative systems, such as behavior of sound waves, fluid dynamics, heat dissipation, weather patterns, disease progression or cellular kinetics etc.

We need to learn the **unknown** function F, from observations $(\mathbf{y}(t_0), ..., \mathbf{y}(t_M)) \in R^{N\times(M+1)}$ of the system's state $\mathbf{u}(t) = (u(\mathbf{x_1},t), ..., u(\mathbf{x_N},t))^T$ at N arbitrary spatial locations $(\mathbf{x_1}, ..., \mathbf{x_N})$ and at M+1 timepoints $(t_0, ..., t_M)$.

2.1 Techniques Used

Following techniques are majorly used for the implementation of the paper idea-

- **Method of Lines (MOL)**: The unknown function, F (1) is numerically approximated to $\hat{F}$ using MOL, by selecting N spatial locations and M+1 time points.
- **Delaunay triangulation**: Delaunay triangulation is used to obtain a good quality space (Ω) discretization. Neighbors for each node were selected by applying Delaunay triangulation to the measurement positions. Two nodes were considered to be neighbors if they lie on the same edge of at least one triangle.
- **Message Passing Neural Networks (MPNN)**[10]: The value of F at a node k must depend only on the nodes k and $neighbors(k)$. Given that the number of arguments and their order in F is not known in advance and might be different for each node, Graph neural networks (GNN) is used to handle arbitrary number of arguments and is permutations-invariant. Adjoint method is used to compute memory efficient gradients for back-propagation for learning continuous-time MPNN surrogates.

2.2 Method of Lines

Method of Lines (MOL) is used to numerically solve (1). This involves selecting N node in Ω and discretizing spatial derivatives at these nodes. These nodes can be used as observation locations $(\mathbf{x_1}, ..., \mathbf{x_N})$. We can estimate F by $\hat{F}$ and get the following system of differential equations (ODEs) whose solution asymptotically approximates the solution of equation (1).

$$\dot{\mathbf{u}}(t) = \begin{bmatrix} \dot{u}_1(t) \\ \vdots \\ \dot{u}_N(t) \end{bmatrix} = \begin{bmatrix} \frac{du(\mathbf{x}_1,t)}{dt} \\ \vdots \\ \frac{du(\mathbf{x}_N,t)}{dt} \end{bmatrix} \approx \begin{bmatrix} \hat{F}(\mathbf{x}_1, \mathbf{x}_{\mathcal{N}(1)}, u_1, u_{\mathcal{N}(1)}) \\ \vdots \\ \hat{F}(\mathbf{x}_N, \mathbf{x}_{\mathcal{N}(N)}, u_N, u_{\mathcal{N}(N)}) \end{bmatrix} \in \mathbb{R}^N \tag{2}$$

In above equation (2), the system's state at $\mathbf{x}_i$ is defined as u_i, and $\mathcal{N}(i)$ is set of indices of neighboring nodes other than i that are required to evaluate $\hat{F}$ at $\mathbf{x}_i$. Similarly, $\mathbf{x}_{\mathcal{N}(i)}$ and $u_{\mathcal{N}(i)}$ are the position and state of the noded $\mathcal{N}(i)$ (where $\mathcal{N}(i)$ is the neighbors of $\mathbf{x}_i$ node).

We can also notice that temporal derivative $\dot{u}$ of u depends not only on the location and state at the node i, but also on locations and states of neighboring nodes, $\mathcal{N}(i)$, resulting in a locally coupled system of ODEs.

Each ODE in the system follows the solution at a fixed location $\mathbf{x}_i$.The full system can be solved using-

$$\mathbf{u}(t) = \mathbf{u}(0) + \int_0^t \dot{\mathbf{u}}(\tau) d\tau \tag{3}$$

where τ is the intermediate time variable

The dicretized $\hat{F}$ inherits the same unknown dynamical system as the true PDE equation function F, we can approximate $\hat{F}$ by a learnable neural surrogate function. The details are mentioned in the following sub-section.

2.3 Message Passing Neural Network Surrogate

We need to consider following requirements to select the surrogate function for the model function $\hat{F}$-

- The value of $\hat{F}$ at a node i must depend only on the nodes i and $\mathcal{N}(i)$.
- the number of arguments and their order in $\hat{F}$ is not known in advance and might be different for each node.

Hence, the model $\hat{F}$ must be able to work with an arbitrary number of arguments and must be invariant to permutations of their order. Graph neural networks (GNNs) satisfy these requirements. In particular, a type of GNNs called message passing neural networks (MPNNs) [10] with parameters θ is used to represent $\hat{F}$ as.

$$\hat{F}_\theta(\underbrace{\mathbf{x}_{\mathcal{N}(i)} - \mathbf{x}_i}_{difference},\ u_i, u_{\mathcal{N}(i)}) \tag{4}$$

$$\mathbf{x}_{\mathcal{N}(i)} - \mathbf{x}_i = \{\mathbf{x}_j - \mathbf{x}_i : j \in \mathcal{N}(i)\} \tag{5}$$

This formulation assumes the absence of position-dependent quantities in F , but models based on this formulation are invariant to translations and rotations of Ω, which makes generalization to systems with different node positions feasible, and prevents overfitting by memorizing position-specific dynamics.

Delaunay triangulation

In this case, we have Graph $G = (V, E)$, where nodes $V = \{x_i\}_{i=1}^N$ are the measurement points and undirected edges (e_{ij}) are defined with respect to the node neighborhood, $\mathcal{N}(i)$. Neighbors for each node were selected by applying **Delaunay triangulation** to the measurement positions. Two nodes were considered to be neighbors if they lie on the same edge of at least one triangle. Delaunay triangulation has such useful properties as maximizing the minimum angle within each triangle in the triangulation and containing the nearest neighbor of each node which helps to obtain a good quality discretization of Ω. I have used scipy package (from scipy.spatial import Delaunay). This is illustrated in figure [1]

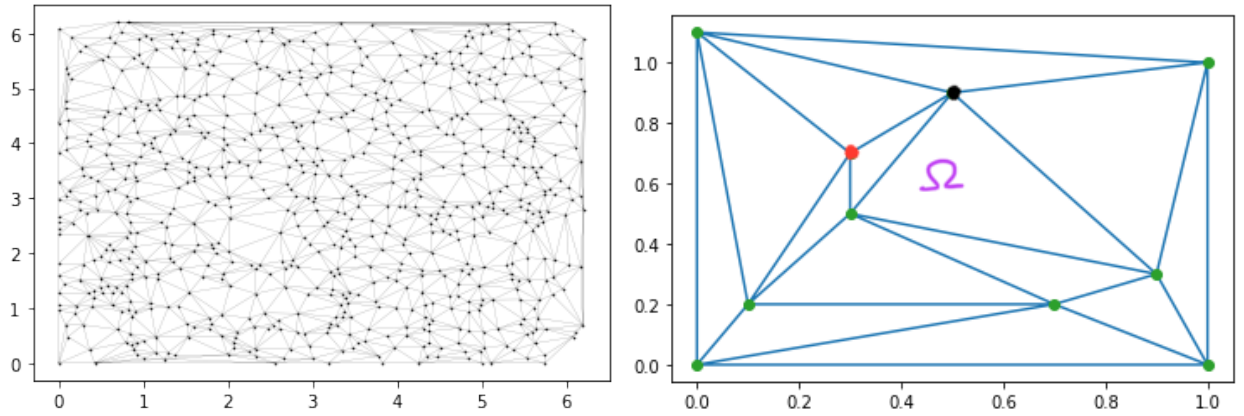
Message Passing Neural Network (MPNN)

We feed-forward a latent state for K graph layers in message passing graph neural networks, parameterized by differentiable functions $\phi^{(k)}$, and $\gamma^{(k)}$ for k layer $(1 \leq k \leq K)$. Each graph layer k consists of-

- Aggregating messages $\mathbf{m}_i^{(k)}$ for each node i.

$$\mathbf{m}_i^{(k+1)} = \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)}(\mathbf{h}_i^{(k)}, \mathbf{h}_j^{(k)}, \mathbf{e}_{ij}) \tag{6}$$

  where $\oplus$ denotes a permutation invariant aggregation function (e.g. sum, mean, max) and edge features is defined as location differences i.e. $\mathbf{e}_{ij} := \mathbf{x}_j - \mathbf{x}_i$



(a) Grid Used (See the Delaunay triangulation)

(b) Sample Delaunay triangulation (Black and red nodes are neighbors as they share an edge.) [1]

Figure 1: Delaunay triangulation for determining the neighbors of a node

- Updating the corresponding node states $\mathbf{h}_i^{(k)}$.

$$\mathbf{h}_i^{(k+1)} = \gamma^{(k)}(\mathbf{h}_i^{(k+1)}, \mathbf{m}_i^{(k+1)}) \tag{7}$$

At any time τ, we initialise the latent states $\mathbf{h}_i^{(0)} = u_i(\tau)$ and node features to the current state $u_i(\tau)$ of the system. The node states at the last graph layer ($\mathbf{h}_i^{(K)}$) of the MPNN is used to evaluate the PDE surrogate.

$$\frac{d\hat{u}(\mathbf{x}_i,t)}{dt} = \hat{F}_\theta(\mathbf{x}_{\mathcal{N}(i)} - \mathbf{x}_i,\ u_i, u_{\mathcal{N}(i)}) = \mathbf{h}_i^{(K)} \tag{8}$$

We can use equation (3) to solve for the estimated states $\hat{\mathbf{u}}(t) = (\hat{u}(\mathbf{x_1},t),...,\hat{u}(\mathbf{x_N},t))^T$

2.4 Adjoint method for learning continuous-time MPNN surrogates

$\hat{F}_\theta$ is parametrized by θ, which represents the parameters of the functions $\phi^{(k)}$, and $\gamma^{(k)}$ for k-th layer $(1 \le k \le K)$ in the MPNN. We minimize the Mean Squared Error (MSE) between the observed states $(\mathbf{y}(t_0),...,\mathbf{y}(t_M))$ and the estimated states $(\hat{\mathbf{u}}(t_0),...,\hat{\mathbf{u}}(t_M))$ to learn the best fit θ parameter.

$$\begin{aligned}\mathscr{L}(\theta) &= \int_{t_0}^{t_M} l(t,\hat{\mathbf{u}})dt \\ &= \int_{t_0}^{t_M} \frac{1}{M+1}\sum_{i=0}^{M} \|\hat{\mathbf{u}}(t_i) - \mathbf{y}(t_i)\|_2^2 \delta(t-t_i)dt \\ &= \frac{1}{M+1}\sum_{i=1}^{M} \|\hat{\mathbf{u}}(t_i) - \mathbf{y}(t_i)\|_2^2\end{aligned} \tag{9}$$

Adaptive solver might requires significantly larger than M time steps to obtain the estimated states. The amount of memory required to evaluate the gradient of $\mathscr{L}(\theta)$ by backpropagation scales linearly with the number of solver time steps. Thus, backpropagation for adaptive methods is typically infeasible due to large memory requirements. The paper [1] uses a memory efficient adjoint method based approach for neural ODE as presented in [11]. The adjoint method involves a single forward ODE pass (3) until state $\mathbf{u}(t_M)$ at the final time t_M, and subsequent backward ODE pass solving the gradients. The backward pass is performed by first solving the below adjoint equation for the adjoint variables λ from $t = t_M$ until $t = 0$ with $\lambda(t_M) = 0$

$$\dot{\lambda}(t)^T = \frac{\partial l)}{\partial \hat{\mathbf{u}}(t)} - \lambda(t)^T \frac{\partial \hat{F}}{\partial \hat{\mathbf{u}}(t)} \tag{10}$$

Then calculate the gradient as-

$$\frac{\partial \mathscr{L}}{\partial \theta} = -\int_0^T \lambda(t)^T \frac{\partial \hat{F}}{\partial \theta} dt \tag{11}$$

3 Benchmark/ Ablation Studies

Multiple ablation studies are performed to benchmark model performance on different measurement grid sizes, interval between observations, irregular sampling, amount of data and amount of noise. Heatmap of true and leaned system dynamics are also presented.

3.1 Convection-diffusion ablation studies

The convection-diffusion equation is widely used PDE to model a variety of physical phenomena related to the transfer of particles, energy, and other physical quantities inside a physical system. The transfer occurs due to two processes: convection and diffusion. Convection-diffusion system for 2D spatial domain is defined below, where u is the concentration of some quantity of interest.

$$\frac{\partial u(x,y,t)}{\partial t} = D\nabla^2 u(x,y,t) - \mathbf{v}.\nabla u(x,y,t) \tag{12}$$

Training and Testing data was obtained by solving the following Initial Boundary Value Problem (IBVP) on spatial domain, $\Omega = [0,2\pi] \times [0,2\pi]$ with periodic boundary conditions.

$$\begin{aligned} &\frac{\partial u(x,y,t)}{\partial t} = D\nabla^2 u(x,y,t) - \mathbf{v}.\nabla u(x,y,t) && (x,y) \in \Omega, t \geq 0 \\ &u(0,y,t) = u(2\pi,y,t) && y \in [0,2\pi], t \geq 0 \\ &u(x,0,t) = u(x,2\pi,t) && x \in [0,2\pi], t \geq 0 \\ &u(x,y,0) = u_0(x,y) && (x,y) \in \Omega \end{aligned} \tag{13}$$

Where, we have set diffusion coefficient D as 0.25, and the velocity field $\mathbf{v}$ as $[5.0, 2.0]^T$. The initial conditions $u_0(x,y)$ were generated as follows-

$$\begin{aligned} \tilde{u}_0(x,y) &= \sum_{k,l=-N}^{N} \lambda_{kl}\, cos(kx+ly) + \gamma_{kl}\, sin(kx+ly) \quad N = 4, \text{ and } \lambda_{kl}, \gamma_{kl} \sim N(0,1) \\ u_0(x,y) &= \frac{\tilde{u}_0(x,y) - \min \tilde{u}_0(x,y)}{\max \tilde{u}_0(x,y) - \min \tilde{u}_0(x,y)} \end{aligned} \tag{14}$$

In all following experiments, the training data contains 24 simulations on the time interval [0, 0.2] sec and the test data contains 50 simulations on the time interval [0, 0.6] sec. And each simulation has values of $u(x,y,t)$ at time points $(t_1, ..., t_M)$ and locations $(\mathbf{x}_1, ..., \mathbf{x}_N)$, where $\mathbf{x}_i = (x_i, y_i)$.

Numerical solutions that represent the true dynamics were obtained using the backward Euler solver with the time step of 0.0002 seconds on a computational grid with 4100 nodes. Training and testing data used in the following experiments is down-sampled from these solutions.

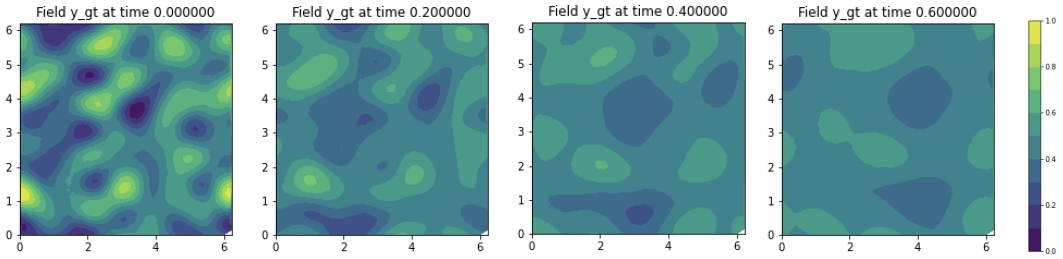
Quality of the model's predictions in the following experiments is evaluated as the relative error between the observed states $\mathbf{y}(t_i)$ and the estimated states $\hat{\mathbf{u}}(t_i)$-

$$Error = \frac{\|\mathbf{y}(t_i) - \hat{\mathbf{u}}(t_i)\|}{\|\hat{\mathbf{u}}(t_i)\|} \tag{15}$$

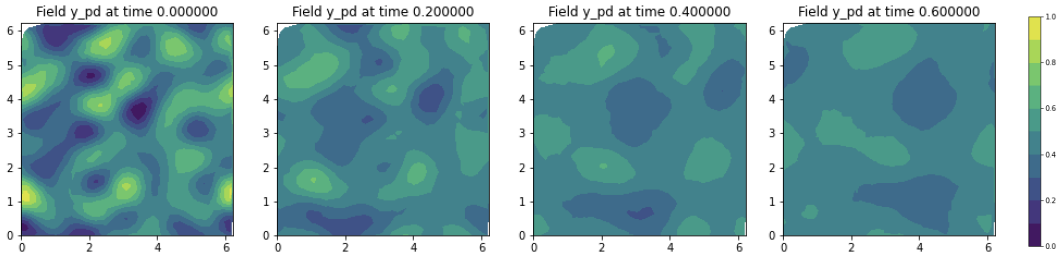
$\varepsilon = 1.0e-6$ is added in the denominator to avoid divide by zero condition.

MPNN surrogate function used: The MPNN surrogate model used for all following experiments contains a single graph layer with mean as the aggregation function. Functions $\phi^{(1)}(u_i,)$ and $\gamma^{(1)}(u_i, u_j - u_i, \mathbf{x}_j - \mathbf{x}_i)$ are represented by 3-hidden layer multilayer perceptrons with *tanh* activation functions. Input/output sizes for $\phi^{(1)}$ and $\gamma^{(1)}$ are set to 4/40 and 41/1 respectively and the number of hidden neurons is 60. This gives approximately 20k trainable parameters.

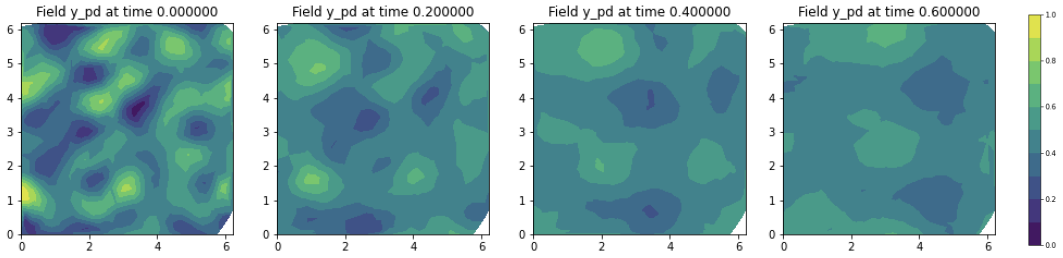
Model Training details: The implementation of the adjoint method and ODE solvers is based on *torchdiffeq* Python package [12]. Adaptive-order implicit euler solver is used with *rtol* and *atol* as 10^{-7} to compute gradients with respect to in the backward pass. *(Note, the paper uses adaptive-order implicit Adams solver, but it was not converging for my case.).* Rprop optimizer with learning rate 10^{-6} and batch size as 24 is used to minimize the MSE error (9).



(a) **Ground Truth Data**



(b) **Grid Size 1500**



(c) **Grid Size 750**

Figure 2: Comparison of the true (top row) and learned system dynamics for grid size 1500 (middle row) and grid size 750 (bottom row) at 4 different time stamps t=0, 0.2s, 0.4s, 0.6s (from left to right)

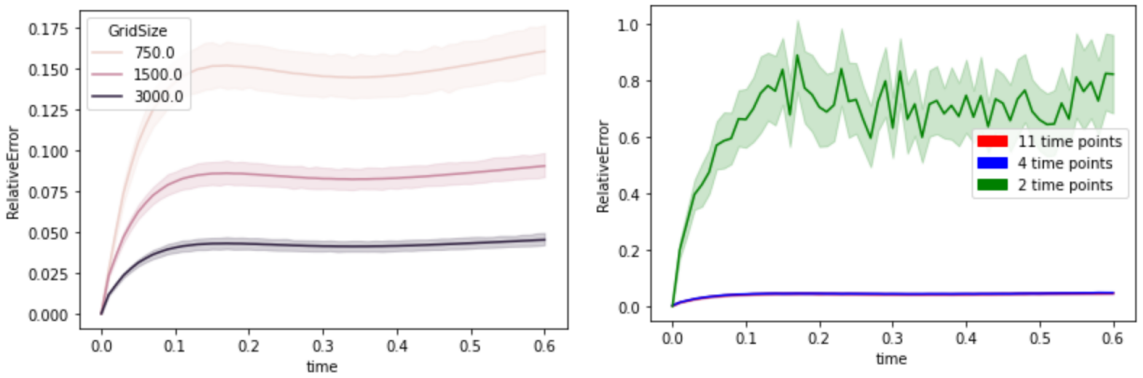
3.1.1 Different grid sizes

In Figure 2, the true and learned system dynamics for grid size 1500 and 750 at 4 different time stamps t=0, 0.2s, 0.4s, and 0.6s for one out of the 50 test cases are shown. For same time stamp, (i.e. in same column), the predicted system dynamics for grid size 1500 is much closer to the real system dynamics as compared to the predicted system dynamics for grid size 750.

The performance of the model decreases with the number of nodes in the grid as shown in 3a. However, for the smallest grid containing 750 nodes, the model is still able to learn a reasonably accurate approximation of the system's dynamics and generalize beyond the training time interval. Also, convergence for smaller grid size was observed to be faster than larger grid size, brings in the trade-off between computational time vs accurate prediction.

3.1.2 Different measurement time interval

Models with a constant time step are sensitive to the length of the time interval between observations. Smaller time steps shows good performance but larger time step (or measurement time interval) fails to generalize. This experiment shows the method's ability to learn a continuous-time model from data with relatively large time intervals between observations.The model was trained and tested for data with time steps of 0.02, 0.0667 and 0.2 sec, which give 11, 4 and 2 time points respectively. The number of nodes was set to 3000. Relative test errors of the model trained on different time grids are shown in Figure 3b. The models' predictions and the true states of the system for one of the 50 test cases are shown in Figure 4. We can see than the model is able to recover the continuous-time system dynamics



(a) Different grid sizes (b) Different measurement time intervals

Figure 3: Relative test errors for (a) different grid sizes and (b) different measurement time interval

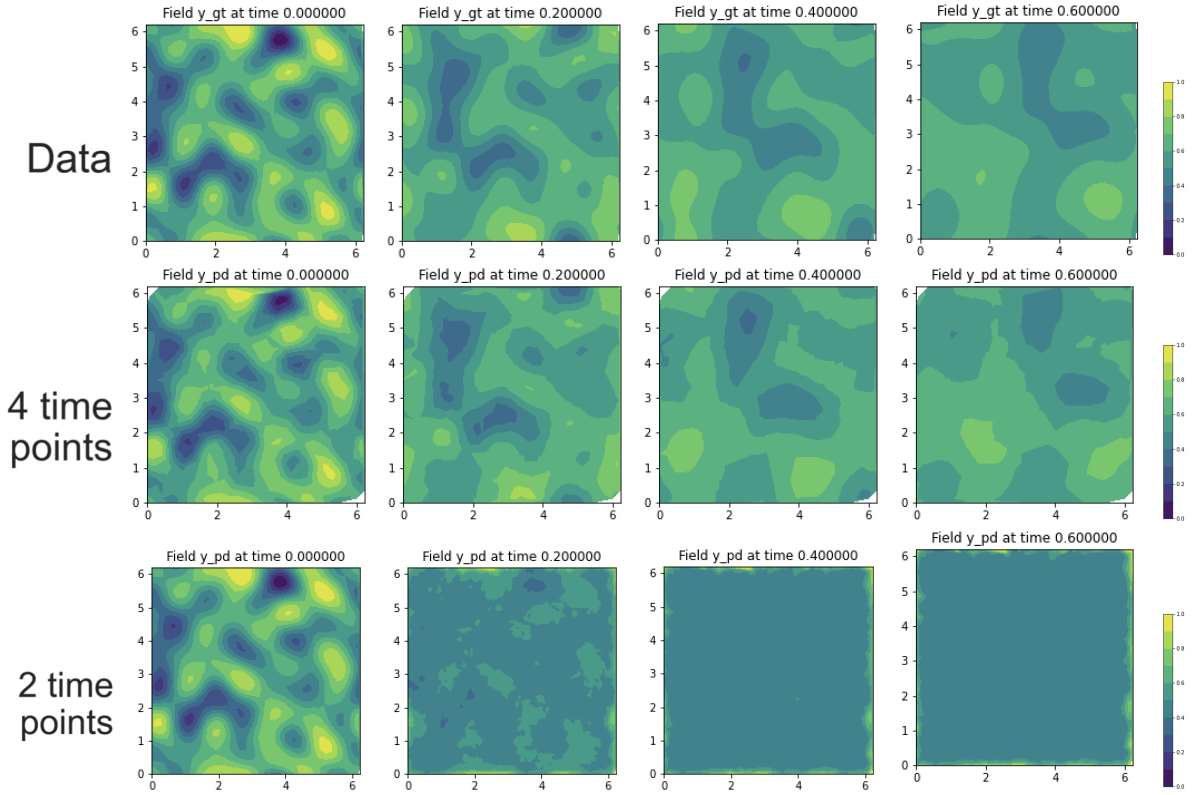


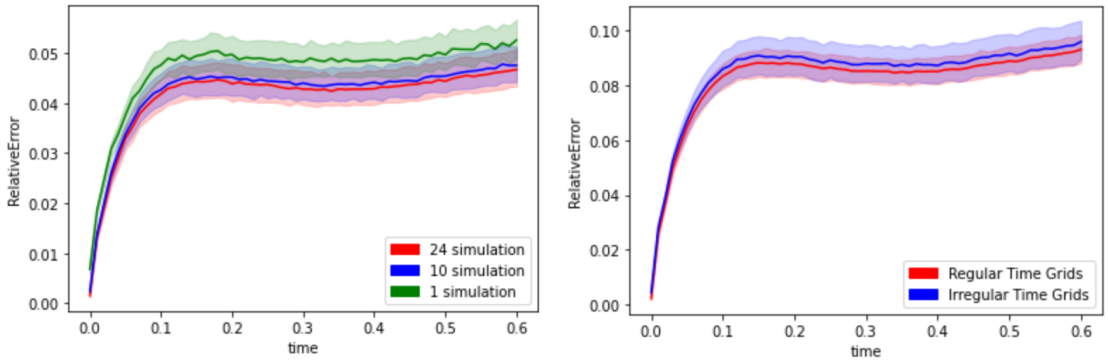
Figure 4: Comparison of the true (top row) and learned system dynamics for 4 time points (middle row) and 2 time points (bottom row) at 4 different time stamps t=0, 0.2s, 0.4s, 0.6s (from left to right)

even when trained with four time point per simulation. Increasing the frequency of observation does not significantly improve performance.

3.1.3 Different amount of data

The model is trained on the data containing 1, 10 and 24 simulations on the time interval $[0, 0.2]$ sec with time step 0.01 sec. The test data has 50 simulations on the time interval [0, 0.6] sec with the same time step. The number of nodes was set to 3000. Relative test errors of the model trained on different numbers of simulations are shown in Figure 5a. Performance of the model improves as the amount of training data increases. However, the relative error does

not converge to zero despite using more data, possibly because of insufficient number of nodes in the training data or insufficient model capacity.



(a) Different amount of data

(b) Irregular time step

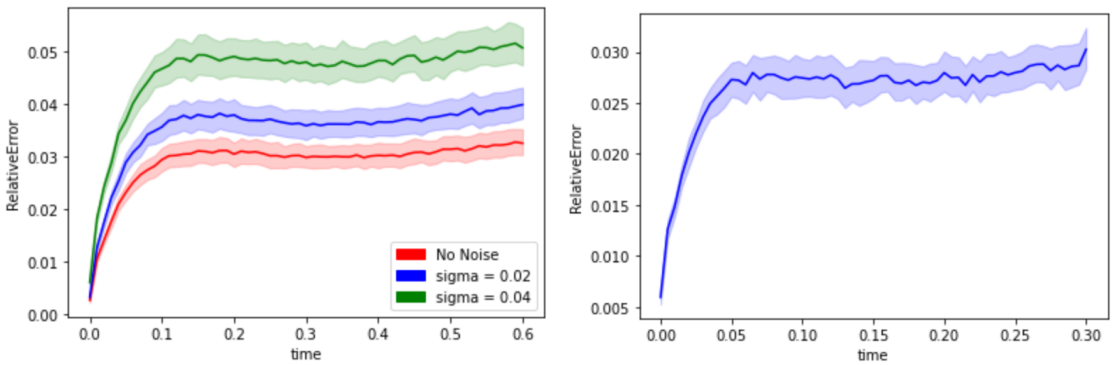
Figure 5: Relative test errors for (a) Different amount of data (Number of simulations) (b) Irregular time step

3.1.4 Irregular time step

The experiment tests our model's ability to learn from data observed at random points in time instead of recorded with a constant time step. The model is trained on two time grids- (i)first time grid has a constant time step 0.02 sec, (ii) second grid is the same as the first one but with each time point perturbed by noise $\varepsilon \sim N(0, (\frac{0.02}{6})^2)$. to have a time grid with an irregular time step. The time step for test data was set to 0.01 sec. The number of nodes was set to 3000. Relative test errors of the model trained with constant and irregular time steps are shown in Figure 5b. The model's similar performance demonstrates the truly continuous-time nature as training and predictions are not restricted to evenly spaced time grids.

3.1.5 Varying amount of additive noise

This experiment tests our model's capability to learn from data with varying amounts of additive noise $\varepsilon \sim N(0, \sigma^2)$. The standard deviation σ was set to 0.02, and 0.04 while the largest magnitude of the observed states is 1. The time step and number of nodes was set to 0.01 sec and 3000 respectively. Noise was added only to the training data. The relative test errors of the model trained on data with different levels of noise are shown in Figure 6a. The results show that the model's performance decreases with increasing noise. Still, the model is able to learn a reasonably accurate approximation of the dynamics of the system.



(a) Varying amount of additive noise

(b) Relative test errors for heat equation

Figure 6: Relative test errors for (a) Varying amount of additive noise (b) Heat Equation

3.2 Heat Equation Experiment

The heat equation describes the behavior of diffusive systems. The equation is defined below in (16). The heat equation describes simpler dynamics than the convection diffusion equation which allowed the model to achieve slightly smaller test errors.

Training and Testing data was obtained by solving the following Initial Boundary Value Problem (IBVP) on spatial domain, $\Omega = [0,1] \times [0,1]$ with Dirichlet boundary conditions.

$$\begin{aligned} &\frac{\partial u(x,y,t)}{\partial t} = D\nabla^2 u(x,y,t) \quad && (x,y) \in \Omega, t \geq 0 \\ &u(x,y,t) = u_0(x,y) && (x,y) \in \text{Boundary}(\Omega), t \geq 0 \\ &u(x,y,0) = u_0(x,y) && (x,y) \in \Omega \end{aligned} \tag{16}$$

Where, we have set diffusion coefficient D as 0.25. The initial conditions $u_0(x,y)$ were generated as follows-

$$\begin{aligned} \tilde{u}_0(x,y) &= \sum_{k,l=-N}^{N} \lambda_{kl}\, cos(kx+ly) + \gamma_{kl}\, sin(kx+ly) \quad N = 10, \text{ and } \lambda_{kl}, \gamma_{kl} \sim N(0,I) \\ u_0(x,y) &= \frac{\tilde{u}_0(x,y) - \min \tilde{u}_0(x,y)}{\max \tilde{u}_0(x,y) - \min \tilde{u}_0(x,y)} \end{aligned} \tag{17}$$

In all following experiments, the training data contains 24 simulations on the time interval [0, 0.2] sec and the test data contains 50 simulations on the time interval [0, 0.6] sec. And each simulation has values of $u(x,y,t)$ at time points $(t_1, ..., t_M)$ and locations $(\mathbf{x}_1, ..., \mathbf{x}_N)$, where $\mathbf{x}_i = (x_i, y_i)$.

Numerical solutions that represent the true dynamics were obtained using the backward Euler solver with the time step of 0.0001 seconds on a computational grid with 4100 nodes. Training and testing data used in the following experiments is down-sampled from these solutions.

Quality of the model's predictions in the following experiments is evaluated as the relative error between the observed states $\mathbf{y}(t_i)$ and the estimated states $\hat{\mathbf{u}}(t_i)$-

$$Error = \frac{\|\mathbf{y}(t_i) - \hat{\mathbf{u}}(t_i)\|}{\|\hat{\mathbf{u}}(t_i)\|} \tag{18}$$

MPNN surrogate function used & Model Training details: MPNN surrogate function used is same as mentioned in above section 3.1 These model Training details details are also same as mentioned in above section 3.1 In the experiment, the training data contains 24 simulations on the time interval [0, 0.1] sec with time step 0.005 sec resulting in 21 time point. The test data contains 50 simulations on the time interval [0, 0.3] sec with the same time step. The number of observation points $\mathbf{x}_i$ was set to 4100.

True and learned system dynamics are presented for heat equation as well to show that the proposed model is able to learn wider range of dynamical systems. Figure 7 demonstrates the real and predicted evolution of the system for one of the test cases and Figure 6b shows relative test errors.

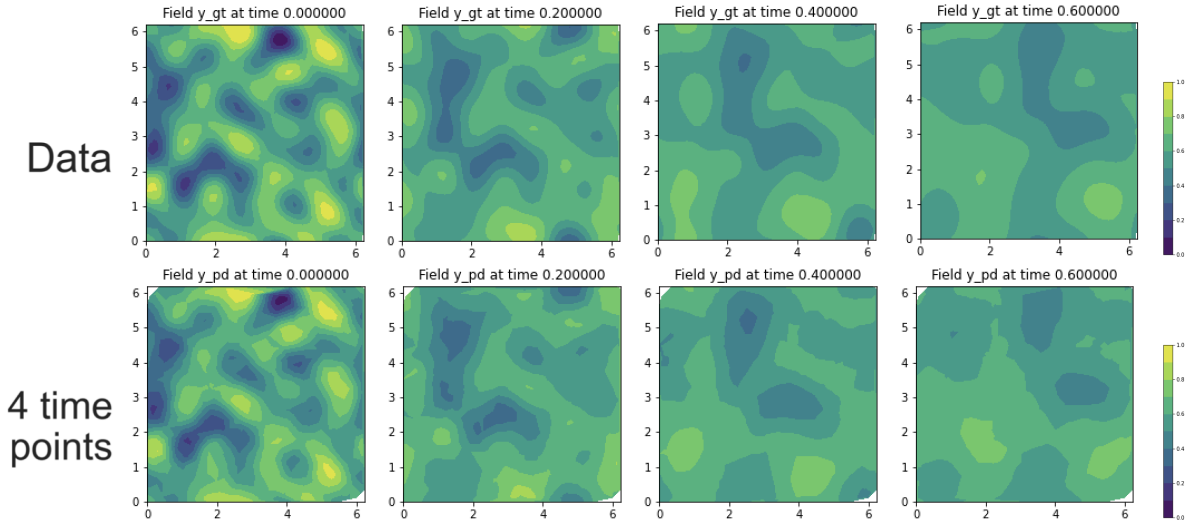


Figure 7: Comparison of the true (top row) and learned system dynamics for grid size 1500 (middle row) and grid size 750 (bottom row) at 4 different time stamps t=0, 0.2s, 0.4s, 0.6s (from left to right)

4 Conclusion

This report explores a continuous-time model to accurately recovers the system's dynamics (governed by PDEs) even when observation points are sparse and the data is recorded at arbitrary space and irregular time intervals. Different ablation studies have been presented for the model for convection-diffusion and heat equations.

5 Challenges & Future work

This report is primarily based on the work of Iakovlev et al. [1], I have tried to reproduce the results presented in this paper. Sometimes, I'm not able to duplicate the results exactly due to variability in different implementation details. However, I have got similar trend as shown in the paper. Few modules needs cuda-enabled PyTorch packages; I've been using Google Colab GPUs for that reason, which has its own limitations of usage (gets disconnected sometimes after certain usage limit, so model requiring too many epoch to converge could not be trained). Convergence is slow and unstable sometimes.As future work, it can be extended or modified to different PDEs systems like wave equation, where we have second-order derivative in time to have a generalized model for different ordered PDEs in time. Additionally, we can try to get some theoretical trade-off (like CFL condition) of these models.

6 Code

CODE at `https://github.com/abhayk1201/799_Project/tree/main/GNN_PDE`
[1]

References

[1] Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time pdes from sparse data with graph neural networks. *arXiv preprint arXiv:2006.08956*, 2020.

[2] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[3] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[4] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

[5] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data, 2018.

[6] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2019.

[7] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[8] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[9] Hao Xu, Haibin Chang, and Dongxiao Zhang. Dl-pde: Deep-learning based data-driven discovery of partial differential equations from discrete and noisy data. *arXiv preprint arXiv:1908.04463*, 2019.

[10] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[11] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

[12] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[1] `https://github.com/abhayk1201/799_Project/tree/main/GNN_PDE`