Geometric NeuralPDE (GNPnet) Models for Learning Dynamics

Oluwadamilola Fasina[1,3] **Smita Krishnaswamy**[1] **Aditi Krishnapriyan**[2,3*]

[1]Yale University [2] University of California, Berkeley [3] Lawerence Berkeley National Lab

Abstract

Real-world phenomena, such as cellular dynamics, electromagnetic wave propagation, and heat diffusion vary with respect to space and time and therefore can described by partial differential equations (PDEs). We focus on the problem of finding a dynamic model and parameterization that can generate and match observed time-series data. For this purpose, we introduce Geometric Neural PDE network (GNPnet), a neural network that learns to match and interpolate measured phenomenon using an autoregressive framework. GPNnet has several novel features including a geometric scattering network that leverages spatial problem structure, and an FEM solver that is incorporated within the network. GPNnet learns parameters of a PDE via an FEM solver that generates solution values that are compared with measured phenomenon. By using the adjoint sensitivity method to differentiate the output loss function, we can train the model end-to-end. We demonstrate GPNnet by learning the parameters of a simulated wave equation.

1 Introduction

Dynamical systems are encountered in many settings: fluid flow, cellular progressions, electromagnetic wave propagation, weather forecasting, financial forecasting, and neutron transport are just a few applications among many [1],[2],[3] [4]. Physical, practical, and engineering limitations mean that dynamical systems are measured at discrete time-steps despite an underlying continuous signal that exists for real-world phenomena. These discrete measurements can be enough to capture the underlying true dynamics of a given system, but there are scenarios where this is not the case. Consider, for example, cellular dynamics: due to technological limitations, one can only measure how a cell progresses in a few discrete time steps; this poses a problem if a measurement is not taken during the time period when some underlying diseases manifest in the cell's lifetime [5]. Of course, there are other situations that may not be as dire, but the example demonstrates many applications where interpolating or extrapolating from static measurements of a dynamical system is desirable.

To this end, several data driven approaches have been utilized for learning dynamics [6],[7],[5] [8]. Each of these approaches attempt to learn the dynamics of a system by leveraging the power of neural networks (NNs) with domain knowledge. While fully connected neural networks are promising for learning dynamics, there have been shortcomings in learning physics [9, 10] because of a lack of awareness of structure within the data during training. Crucially, there are advantages to incorporating PDE solvers into NNs. This can both constrain the system outputs to be more physically meaningful, and can be also be a key component of learning dynamics since most systems vary spatially, in addition to temporally. Consequently, we introduce a novel NN framework, geometric neural PDE network (GNPnet), that harnesses the power of NNs in tandem with the geometric structure of the data and a PDE solver.

*Corresponding author: aditik1@berkeley.edu

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

GNPnet is a graph network that leverages the structure of the available measured scientific data via random walks on a graph, and also incorporates a PDE solver within this framework to generate data for phenomena that vary with respect to space and time. GNPnet is also a fully differentiable architecture that can be trained end-to-end via the adjoint method. In recent years, the combination of deep learning and differential equations has been notable [11, 12, 13, 14, 5]. One aspect of this, notably seen in neural ODEs [11], is the ability to circumvent automatic differentiation through the ODE solver by using the adjoint method to implicitly differentiate the loss function and train the NN. In this setting, the assumption is that there are training datapoints, and the NN outputs the change in the system over time that is then acted upon by an ODE solver. Our setting is different; instead, GNPnet learns the dynamical system by predicting PDE parameters, which are then given to a PDE solver to generate solution data. We then minimize the loss between the predicted solution data and the ground truth input time-series data. This enables us to learn physical parameters, initial conditions, or any properties defining the system in consideration. Importantly, we can learn the dynamics without necessarily having the parameters themselves in our dataset.

2 Details of GNPnet

GNPnet is a NN architecture that leverages the geometric structure of the data to predict PDE parameters (physical parameters, initial conditions, etc.), and utilizes this information to then generate and match time-series data. GNPnet is equipped with a geometric scattering transform, a GNN, and an FEM solver (a widely-used method for solving PDEs). GNPnet takes time-series data as input graphs whose node features are augmented with geometric scattering features. This augmented graph data is then fed into a GNN to predict initial conditions or physical parameters governing the system in consideration. These parameters are then used by an FEM solver to predict the system dynamics. GNPnet is capable of keeping track of the gradient of the FEM output with respect to the FEM input by leveraging the adjoint method [15]. This allows GNPnet to be trained end-to-end from the GNN input through the FEM output such that the predicted dynamics can match the ground truth dynamics (see the GNPnet schematic in Figure 1). Since the GNN outputs parameters or initial conditions that govern the system in consideration, in certain problem settings, we can also verify that the outputs are physically meaningful. In the subsequent section, the components of GNPnet are described in detail.

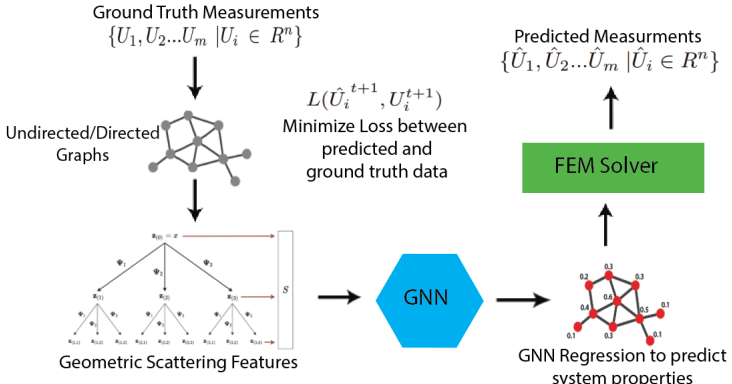

Figure 1: **GNPNet training procedure.** Schematic outlining the training process.

2.1 Geometric Scattering Transform

In general, time-series measurements can be defined in the space-time domain as $U : \Omega \times (0, T) \to R^n$. Given a set of measurements$\{U_1, U_2 ... U_m \, | U_i \in R^n\}$, each consisting of n nodes for m time points/measurements $t_1, t_2, ..., t_m$, GNPnet first constructs a graph for each measurement U_i at a corresponding time point t_i. GNPnet achieves this by computing a new matrix $U_{i_{GT}} \in R^{n \times 3}$, where the rows represent the set of nodes $v_1, v_2 ... v_n$ in a given measurement, U_i, and the dimensions of the coordinates $(x, y) \in R^2$ are concatenated with the value $g(v_i) \in R$ at that node, $(x, y, g(v_{i_n}))$. The graph can then be constructed by using a kernel, broadly represented as $K(x, y) = e^{-||x-y||_2^2/\sigma}$, which in our setting is $K(v_i, v_j) = e^{-||v_i - v_j||_2^2/\sigma} = G_{i_{GT}}$. Here, (v_i, v_j) represents pairwise nodes in the matrix $U_{i_{GT}}$.

In [16] the authors demonstrated the utility of the geometric scattering transform in the context of graph representations, i.e., geometric scattering features computed on dirac signals placed on a graph, are capable of capturing multigranular graph structure. Since, then geometric scattering has been used for various machine learning tasks and shown promise in a variety of applications in deep learning [17, 18, 19] because of the rich geometric structure that the transform captures.

The geometric scattering transform was introduced in [16] and provides a rich way of describing signals on graphs and graphs themselves. The scattering transform is based off of diffusion wavelets [20] of the form $\mathbf{\Psi}_j \coloneqq \mathbf{P}_{1/2}^{2^{j-1}} - \mathbf{P}_{1/2}^{2^j}$ where ${}_{1/2} \coloneqq \frac{1}{2}(_n + ^{-1})$ is a matrix which describes the transition probabilities of a lazy random walker, and j is the scale of the wavelet. Given these wavelets, we geometric scattering computes an alternating cascade of wavelets and element-wise absolute value nonlinearities to form $\mathbf{U}_p \coloneqq \mathbf{\Psi}_{j_m}|\mathbf{\Psi}_{j_{m-1}}\ldots|\mathbf{\Psi}_{j_2}|\mathbf{\Psi}_{j_1}|\ldots|$ for a scattering path $p \coloneqq (j_1, \ldots, j_m)$. Given $\mathbf{U}_p$, we then compute scattering moments of the form $\mathbf{S}_{p,q} = \sum_{i=1}^{n} |\mathbf{U}_p[v_i]|^q$..

In this study, we use three dyadic scales, two layers, and five moments. For this set of hyperparameters, the geometric scattering transform always outputs p = 20 scattering features for each node, which are concatenated to the original node features $(x, y, g(v_{i_m}))$ of the matrix $U_{i_{GT}}$ to create the new set of augmented measurement data: $\{\tilde{U}_{1_{GT}}, \tilde{U}_{2_{GT}}...\tilde{U}_{m_{GT}}|\tilde{U}_{i_{GT}} \in R^{n\times p}\}$. From the new set of measurement data, GNPnet computes a kNN-graph (with k=4) to construct a new set of augmented graphs$\{\tilde{G}_{1_{GT}}, \tilde{G}_{2_{GT}}....\tilde{G}_{m_{GT}}|\tilde{G} \in R^{n\times n}\}$ that are inputs to the GNN.

2.2 Learning Dynamics

To learn the dynamics of the system in consideration, GNPnet minimizes the discrepancy between the predicted measurements generated by the PDE solver and the ground truth measurements. Once the scattering transform is computed, the GNN takes the set of augmented graphs and predicts parameters or initial conditions that define the system in consideration. An input graph $G = (V, E)$ has nodes $i \in V$ and edges $ij \in E$ that define the graph connectivity. It also has a node position matrix with shape $n \times 2$ of the node feature vector, $f_i \in R^d = (x, y, g(v_i), s_i)$ contains the spatial coordinates (x, y), the solution value $g(v_i)$ at node v_i, and the scattering features $s_i \in R^p$. Formally, the GNN learns the system properties by learning the mapping $GNN :^{nxd}\rightarrow R^{n\times 1}$ for a given graph such that each node in a graph has one output value which corresponds to the value of the parameter or initial condition at that node. The node features are updated through a standard MLP message passing layer, $v_i = MLP_v(v_i, \sum_j e_{ij})$.

We train GNPnet in an autoregressive manner to learn the dynamics of the predicted system. To accomplish this, the measurement data must be passed into the GNN chronologically. For an observed set time series measurements, $\{U_1, U_2...U_m\ |U_i \in R^n\}$, the set of augmented graphs $\{\tilde{G}_{1_{GT}}, \tilde{G}_{2_{GT}}....\tilde{G}_{m_{GT}}\}$ must be fed in chronological order, such that GNPNet learns the dynamical process of the full system. The GNN outputs a new matrix $U_{i_{pred}} \in R^{n\times 1}$ whose nodes values represent the physical parameters or initial conditions at the input measurement data at t_0. Thus far, we can summarize the steps that take us from the ground truth measurement U_i at time t_i to the output of the GNN as:

$$U_i \rightarrow U_{i_{GT}} \rightarrow K(U_{i_{GT}}) \rightarrow G_{i_{GT}} \rightarrow GST(G_{i_{GT}}) \rightarrow \tilde{G^t}_{GT} \rightarrow GNN(\tilde{G^t}_{i_{GT}}) \rightarrow U_s \quad (1)$$

The output of the GNN, U_s, (which defines system properties/parameters for a given system) is fed as input to the PDE solver, which then generates predicted solution values at different time steps. The network is trained autoregressively by constructing an element-wise mean-squared error loss between the predicted $\hat{U_i}^{t+1}$ and the ground truth U_i^{t+1} value for j nodes in a given solution U at the next $t + 1$ time step. The adjoint method is then used to compute the derivatives of the output of the FEM solver with respect to the NN parameters.

$$U_s \rightarrow FEM \rightarrow \hat{U}^{t+1} \quad (2)$$

$$L(\hat{U}^{t+1}, U^{t+1}) = \frac{1}{n}\sum_{j=1}^{n}(U_j^{t+1} - \hat{U^{t+1}}_j) \quad (3)$$

$$\frac{\partial L}{\partial U_s} * \frac{\partial U_s}{\partial \theta} \tag{4}$$

3 Experiments

The wave equation is a hyperbolic PDE defined as $\frac{\partial^2 u}{\partial t^2} - c^2(x)\frac{\partial^2 u}{\partial x^2} = 0$, where u is the field value and $c(x)$ is the wave velocity. We use the wave equation as a toy example to demonstrate the efficacy of GNPnet. The solution data for the wave equation was generated using a [51 x 51] square mesh containing 2601 nodes, a spatial domain of [0,1] in the x and y direction, T = 30 time steps, Dirichlet boundary conditions, $u|_{\partial\Omega} = 0$, with the time derivatives and initial condition equal to an arbitrary constant: $u'_0 = u_0 = 5$. We train GNPnet to match the ground truth dynamics by having the GNN learn the wave velocity parameter defining the wave equation. The wave velocity parameter used to generate the data varies spatially and is defined as: $c(x, y) = e^{(-\alpha * x - \alpha * y)}$, with 40 parameter values of α ranging from [2,6]. We ablate GNPnet against a baseline (a single pass of the field values through our GNN) and GNPnet without geometric scattering. The mean absolute and relative error along with the cross-correlation between predicted and ground truth wave velocity are shown in Table 1.

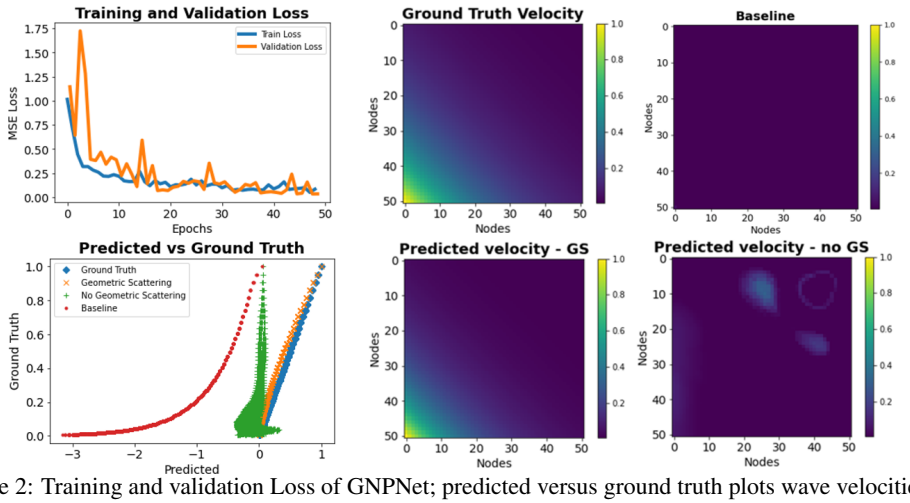

Figure 2: Training and validation Loss of GNPNet; predicted versus ground truth plots wave velocities for different experiments

Table 1: Relative error, absolute error, and cross correlation between ground truth and predicted wave velocity

Experiment	Absolute Error	Relative Error	Cross-correlation
Baseline	1.723E+0	4.666E+1	3.288E-1
No geometric scattering	2.267E-1	4.150E+0	8.522E-1
Geometric scattering (ours)	**4.994E-2**	**5.335E-1**	**9.893E-1**

4 Conclusions

Interpolation and extrapolation of dynamical systems is paramount across disciplines because we are limited to finite, discrete measurements of continuous systems. To address this challenge, we propose GNPnet. We equip GNPnet with rich scattering features that exploit the geometry of the data, and a fully-integrated PDE solver to generate predicted time-series data. Here, we demonstrated GNPnets' ability to learn dynamics on the wave equation. As shown in Figure 1, GNPnet is able to learn the dynamics of the wave equation—the training and validation loss decrease over time—and is capable of recovering the wave velocity parameter that defines the system. However, it should be noted that the inversion of a PDE is not always possible, as there could be many parameter sets that generate a given time series. The autoregressive loss that GNPNet is trained on is capable of finding parameters that can generate the correct dynamics, and potentially different solutions. In the future, we hope to test GNPnet on time-series measurements from systems where ground truth parameters for generations are not known, such as biophysical systems.

5 Broader Impact

Our method is a general framework for interpolating and extrapolating at different time points for dynamical systems. This method follows the foot steps of recent works in machine learning that are focused on leveraging the power of neural networks in the natural sciences. Although more rigorous testing for this method still remains, we envision this method having a net positive societal impact: we hope others can build on the concept of intertwining PDEs and neural networks that leverage the structure of the data in an application-agnostic manner.

References

[1] Elmer Eugene Lewis and Warren F Miller. Computational methods of neutron transport. 1984.

[2] Yaser S Abu-Mostafa and Amir F Atiya. Introduction to financial forecasting. *Applied intelligence*, 6(3):205–213, 1996.

[3] Anita K Hopper, Dave A Pai, and David R Engelke. Cellular dynamics of trnas and their genes. *FEBS letters*, 584(2):310–317, 2010.

[4] Tilmann Gneiting and Adrian E Raftery. Weather forecasting with ensemble methods. *Science*, 310(5746):248–249, 2005.

[5] Alexander Tong, Jessie Huang, Guy Wolf, David Van Dijk, and Smita Krishnaswamy. Trajectorynet: A dynamic optimal transport network for modeling cellular dynamics. In *International conference on machine learning*, pages 9526–9536. PMLR, 2020.

[6] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, pages 1–12, 2022.

[7] Maziar Raissi, Paris Perdikaris, and George 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.

[8] Jeehyun Hwang, Jeongwhan Choi, Hwangyong Choi, Kookjin Lee, Dongeun Lee, and Noseong Park. Climate modeling with neural diffusion equations. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 230–239. IEEE, 2021.

[9] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

[10] Aditi S Krishnapriyan, Alejandro F Queiruga, N Benjamin Erichson, and Michael W Mahoney. Learning continuous models for continuous physics. *arXiv preprint arXiv:2202.08494*, 2022.

[11] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[12] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

[13] Qingqing Zhao, David B Lindell, and Gordon Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. *arXiv preprint arXiv:2206.00711*, 2022.

[14] Sebastian K Mitusch, Simon W Funke, and Miroslav Kuchta. Hybrid fem-nn models: Combining artificial neural networks with the finite element method. *Journal of Computational Physics*, 446:110651, 2021.

[15] Sebastian K Mitusch, Simon W Funke, and Jørgen S Dokken. dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. *Journal of Open Source Software*, 4(38):1292, 2019.

[16] Feng Gao, Guy Wolf, and Matthew Hirn. Geometric scattering for graph data analysis. In *International Conference on Machine Learning*, pages 2122–2131. PMLR, 2019.

[17] Joyce Chew, Matthew Hirn, Smita Krishnaswamy, Deanna Needell, Michael Perlmutter, Holly Steach, Siddharth Viswanath, and Hau-Tieng Wu. Geometric scattering on measure spaces. *arXiv preprint arXiv:2208.08561*, 2022.

[18] Alexander Tong, Frederik Wenkel, Dhananjay Bhaskar, Kincaid Macdonald, Jackson Grady, Michael Perlmutter, Smita Krishnaswamy, and Guy Wolf. Learnable filters for geometric scattering modules. *arXiv preprint arXiv:2208.07458*, 2022.

[19] Yimeng Min, Frederik Wenkel, and Guy Wolf. Geometric scattering attention networks. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8518–8522. IEEE, 2021.

[20] Ronald R Coifman and Mauro Maggioni. Diffusion wavelets. *Applied and computational harmonic analysis*, 21(1):53–94, 2006.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [No]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]