Advances in Artificial Intelligence and Machine Learning; Research 2 (2) 366-384 Received 10-05-22; Accepted 24-05-22; Published 30-05-22

Using Neural Architectures to Model Complex Dynamical Systems

Nicholas Gabriel ngabriel@email.gwu.edu
Physics Department
George Washington University
Washington, DC 20052, USA

Neil F. Johnson neiljohnson@gwu.edu
Physics Department
George Washington University
Washington, DC 20052, USA

Corresponding Author: Neil F. Johnson

Abstract

The natural, physical and social worlds abound with feedback processes that make the challenge of modeling the underlying system an extremely complex one. This paper proposes an end-to-end deep learning approach to modelling such so-called complex systems which addresses two problems: (1) scientific model discovery when we have only incomplete/partial knowledge of system dynamics; (2) integration of graph-structured data into scientific machine learning (SciML) using graph neural networks. It is well known that deep learning (DL) has had remarkable success in leveraging large amounts of unstructured data into downstream tasks such as clustering, classification, and regression. Recently, the development of graph neural networks has extended DL techniques to graph structured data of complex systems. However, DL methods still appear largely disjointed with established scientific knowledge, and the contribution to basic science is not always apparent. This disconnect has spurred the development of physics-informed deep learning, and more generally, the emerging discipline of SciML. Modelling complex systems in the physical, biological, and social sciences within the SciML framework requires further considerations. We argue the need to consider heterogeneous, graph-structured data as well as the effective scale at which we can observe system dynamics. Our proposal would open up a joint approach to the previously distinct fields of graph representation learning and SciML.

Keywords: Neural networks, Complex systems, Deep learning.

1. INTRODUCTION

The natural, physical and social world is full of examples of so-called complex systems. These are systems which contain many interacting parts, each of which may itself evolve in time, and where feedback loops can occur across scales. Human society provides a good example at all scales – from families to communities to organizations and nations. Moreover, such human systems are now enhanced by the existence of additional interactions in the online world. Unfortunately, there is no easy way to describe the dynamics of such systems. Traditional disciplines have tried, but given the explosion in real data both online and offline, we argue here that machine learning is in an ideal position to explore appropriate descriptions going forward.

Even within the computer science community, no unified approach to describing complex systems exists. While this may appear problematic, it also provides an opportunity which we explore here. Specifically, in this paper we discuss how neural architectures can provide novel descriptions of complex dynamical systems that solve several key challenges. Having motivated the challenges, we propose an end-to-end deep learning approach to modelling complex systems which addresses two key problems: (1) scientific model discovery when we have only incomplete/partial knowledge of system dynamics; (2) integration of graph-structured data into SciML using graph neural networks. In this way, our proposed approach combines two previously distinct fields of graph representation learning and SciML.

This paper lays out a roadmap for solving current problems of scalability and data integration in complex systems rather than providing definitive conclusions/comparison of the best models. Furthermore, there is value in assessing how such hitherto distinct tools in machine learning and artificial intelligence can be combined in a way to circumvent existing problems. We hope that this paper can therefore stimulate work along this line of thinking.

1.1 Complex Dynamical Systems and Networks

Many real world systems can be modelled as networks – specifically, discrete sets of objects and their interactions. In particular, the interactions of physical, biological, and social objects can frequently be represented by set of connections, or edges, between the different objects. For a sufficient number of objects (say, $N \geq 100$) the aggregate systems formed by these objects and their edges is complex: they display collective behavior which is not predictable even with full knowledge of the system at the individual level. Such behaviour of complex systems is called emergent behavior, a formalization of the adage "the whole is greater than the sum of its parts". Emergent phenomena tend to have their roots in the self-organization of complex systems. There is no rigorous set of criteria for self-organization, but common features include [1]:

1. sufficient 'energy' or impetus within the system to enable endogenous transitions between configurations of the system
2. non-linearity of interactions, often by a feedback mechanism
3. slowly-driven, metastable, out of equilibrium dynamics

Specific examples of systems and their associated emergent phenomena are as follows:

system	degrees of freedom	interactions	emergent phenomena
brain	neurons	action potential	thoughts, behaviour
society	humans	communication	ideologies, group formation
N-body	atoms/molecules	electromagnetic	magnetization, crystallization

In the case of an N-body system in statistical physics, there are many more specific conditions under which self organization, or self organized *criticality*, can occur. Specifically, some physical systems are speculated to "self tune" towards system parameters β near their critical values β_c, in which region a spatial or temporal quantity a displays scale-free behavior $a \sim a_o|\beta - \beta_c|^\alpha$. Examples of systems with such scale-free behavior have been identified in geophysics [2], plasma physics [3], and neuroscience [4], among others. Furthermore, social systems exhibit approximately scale-free network structure [5], but it is not clear to what extent we can apply results from physical systems in this context. In particular, physical systems are invariably modeled as being at or near equilibrium whereas many real-world systems of societal importance are instead in a continual state of evolution.

Data-driven approaches, particularly deep learning with neural networks, provide a complementary approach to study complex systems. In their most basic form, such models would not assume any interaction or structure, but would *learn* the form of such interactions directly from data. In most scenarios this is far too much freedom: we typically have some idea of how the interactions or statistics of the system should look. So far there is no dominant paradigm for embedding such intuitions into machine learning models, though the emerging field of scientific machine learning is gaining traction. For the remainder of the introduction, we develop the concepts needed for applying deep learning to complex systems.

To provide a basic framework for the remaining sections, we now give a generic description of a complex system. Consider a time-dependent system of N objects possessing a microscopic state $\sigma_i(t)$, with $\mathbf{M}$ such that the dynamics of the system can be written as

$$\frac{d\sigma_i}{dt} = F\big(\{\sigma_i\}, \mathbf{M}\big) \tag{1}$$

where $i = 1, \ldots, N$. For a physical system we might consider σ_i which are set of scalars, vectors, matrices, or tensors, and $\mathbf{M} = \{M^\alpha\}$ a set of matrices or tensors with M^α_{ij} parameterizing the αth interaction between nodes i and j. The functional form of the dynamics, F, can be deterministic or probabilistic in nature. At this point the system dynamics could be extremely complicated, but we would not consider it complex. In principle, given some boundary conditions or initial conditions on the σ_i we can calculate a unique solution of the system, or in other words the system has (in principle) a closed-form, analytic solution. Additionally, if we were to limit ourselves to classical and statistical physics, we could generally assume that systems of this form are ergodic and possibly path-independent [6]. By contrast, for the system to be complex we require that the interactions themselves are also time dependent, or formally that the system is described by a system of equations of the form:

$$\begin{aligned} \frac{d\sigma_i}{dt} &= F\big(\{\sigma_i\}, M^\alpha_{ij}\big) \\ \frac{dM^\alpha_{ij}}{dt} &= G\big(\{\sigma_i\}, M^\alpha_{ij}\big) \end{aligned} \tag{2}$$

In almost all cases, systems of this type will not have a closed-form solution and in general their evolution will be path-dependent and *non*-ergodic.

1.2 A Simple Network Representation of Complex Dynamical Systems

The simplest definition of a complex network is arguably equation (2) with an additional topological constraint called adjacency, whereby the existence of an edge between two nodes ($A_{ij} = 1$ indicates an edge, $A_{ij} = 0$ no edge) dictates whether they interact or not. In other words, $M_{ij}^{\alpha} = 0$ when $A_{ij} = 0$. The study of network science, loosely speaking, is the study of the statistical, structural, and functional properties of networks formed by real world systems. In general the number of adjacent nodes, or edges, in a complex network is far less than the number of possible edges between N nodes, or

$$\sum_{i \neq j} A_{ij} \ll \frac{N(N-1)}{2} \tag{3}$$

and the number of edges scales sublinearly with the number of nodes so that

$$\lim_{N \to \infty} \frac{\sum_{i,j} A_{ij}}{N} = 0. \tag{4}$$

These properties are often referred to (at least colloquially) as sparsity conditions of a network. This property of networks is not only a robust empirical finding, but a necessary mathematical condition for many important network properties such as modularity (i.e. where nodes form community structures) and hyperbolicity (i.e. some nodes have many more edges than others) (FIGURE 1).

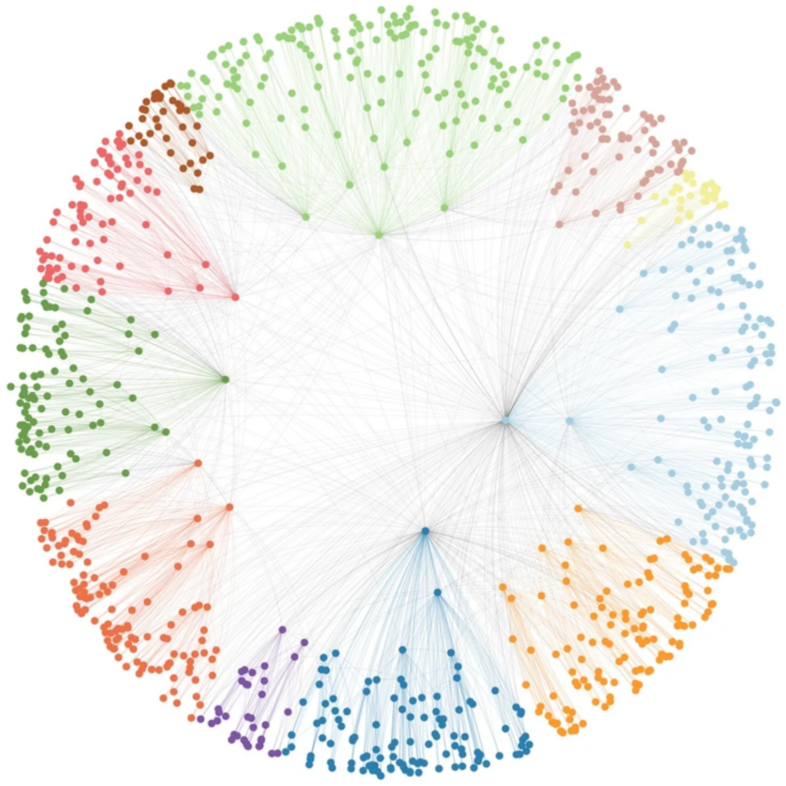

Figure 1: A network that with a modular (modules are color coded) and hyperbolic structure (there are ~4 "parent" nodes from which many "descendant" nodes attach) See Ref. [7], for details.

1.3 Graph Embeddings

For the remaining sections, we will define a graph to be the configuration specified by A_{ij}. We call this structure a graph rather than a network to avoid confusion with *neural* networks, which will be central to the ensuing discussion. Graph Embedding, broadly construed, is the systematic assignment of low-dimensional vectors $z_i \in \mathbb{R}^d$ to each object of a system such that the vectors encode some meaningful properties of the system. For a system size N we would typically define low-dimensional to as $d \ll N$. Given some pairwise data S_{ij} (where $S_{ij} = 0$ when $A_{ij} = 0$) or nodewise data y_i, for example, we might seek some z_i such that

1. fitting pairwise data: $g(z_i, z_j) \approx S_{ij}$
2. fitting nodewise data: $f(z_i) \approx y_i$

where the functions f and g can be imposed (e.g. matrix factorization below) or learned (e.g. graph neural networks). The motivation for defining such low dimensional embeddings typically reflects our belief that the underlying process that produces the data S_{ij} and y_i has a much fewer degrees of freedom than the data itself. Enforcing a low dimension for the embeddings can also provide a form of regularization against noise in the process that produces the data. Embeddings are conceptually similar to spectral methods for dimensionality reduction such as principal component analysis or singular value decomposition. Much of the inspiration for applying neural networks to produce embeddings was to respond to the shortcomings of spectral methods in text analysis [8]. In the next section we develop an optimization approach to graph embedding which will motivate a neural embedding approach.

1.4 Embeddings via Matrix Factorization

Suppose we wish to solve the pairwise problem above of approximating $g(z_i, z_j) \approx S_{ij}$. If we pack each of the embeddings into a matrix $\mathbf{Z}$ as

$$\mathbf{Z} = (z_1\ z_2\ \dots\ z_N) \tag{5}$$

defined by

$$\min_{\mathbf{Z}} ||\mathbf{S} - \mathbf{Z}^T\mathbf{Z}||_F \tag{6}$$

where $||\cdot||_F$ is the Frobenius norm, then $\mathbf{Z} \in \mathbb{R}^{N\times d}$ compresses the edgewise data $\mathbf{S} \in \mathbb{R}^{N\times N}$ when $d < N$. A typical procedure for determining the $\{z_i\}$ that satisfy (6) is as follows:

1. define a loss $\mathcal{L} = \sum_{ij}(S_{ij} - \sigma(z_i^T z_j))^2$, where $\sigma(\cdot)$ is e.g. a sigmoid function
2. select a random subset of nodes indexed by i_k
3. calculate the sum of gradients with respect to each component of z_{i_k}

$$\begin{aligned}\bar{\nabla}\mathcal{L} &= \sum_{i_k} \nabla_{i_k}\mathcal{L},\\ \nabla_i &= \left(\frac{\partial}{\partial z_{i1}}, \dots, \frac{\partial}{\partial z_{id}}\right)\end{aligned} \tag{7}$$

4. update all embeddings according to $z_i \leftarrow z_i - \eta\bar{\nabla}\mathcal{L}$.

We would then repeat steps 2-4 until some stop condition is met, e.g. $\mathcal{L}$ no longer decreases beyond some threshold for additional iterations. This procedure is called batch gradient descent, where i_k defines the batch at each update. Since we only considered the first derivative with respect to the loss, this method would be classified as a first-order optimization method. Currently, the most popular optimizers have adaptive learning rates. Examples of adaptive optimizers are AdaGrad, ADAM, and RMSProp empirically have better performance on a wide range of tasks since they can automatically adjust to heterogenous optimization landscapes without manual intervention. Second-order/Newtonian can more efficiently locate local minima but are computationally infeasible due to a Hessian matrix of size $\mathbb{R}^{(N\times d)^2}$ required for such procedures. Quasi-Newtonian methods such as L-BFGS store an approximate representation of the Hessian (Cholesky factor) to make second order techniques feasible for large systems. We refer to Ref. [9] for a more comprehensive review of optimizers.

1.5 Neural Networks

There are several limitations to the matrix factorization approach in the last sections

(I) If we have some nodewise features $x_i \in \mathbb{R}^m$ for our system, there is no way to easily incorporate them.

(II) If a new node i' is added to the system, we have to repeat our optimization procedure to obtain an embedding $z_{i'}$

(III) The topology A_{ij} is not naturally utilized (though is can be incorporated into S_{ij})

We can address (I) and (II) with a neural network defined by equations (8) and (9) as follows (FIGURE 2):

1. use the x_i as input into a neural network
2. define the lth hidden layer to be the embedding of the input data $z_i = h_i^{(l)}$
3. define the output layer f_i in terms of some predictive task such that each $f_i \approx y_i$

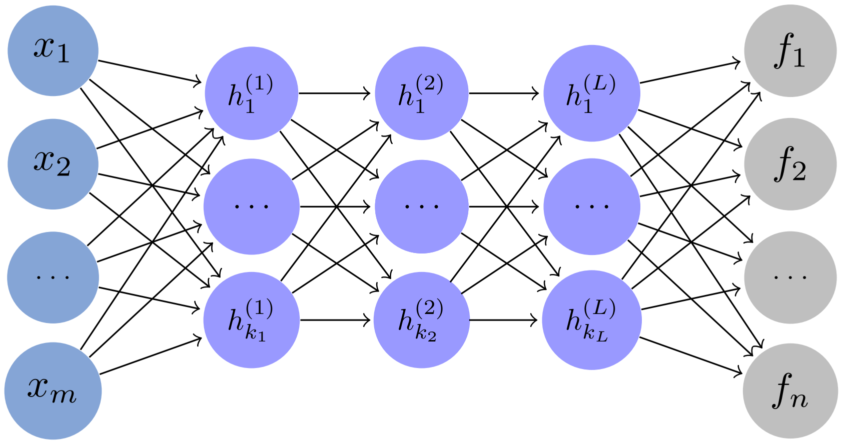

Figure 2: An MLP with $L = 3$ hidden layers.

The loss for this procedure could then be obtained by adding a squared error term for the predictive task:

$$\mathcal{L} = \sum_{ij}(S_{ij} - \sigma(z_i^T z_j))^2 + \sum_{ik}(f_{ik} - y_{ik})^2. \tag{8}$$

Arguably the simplest type of neural network is the multi-layer perceptron (MLP) defined by

$$h^{(l+1)} = \sigma\left(W^{(l)}h^{(l)} + b^{(l)}\right) \tag{9}$$

where the activation function $\sigma(\cdot)$ is chosen so that the mapping between layers is non-linear, and the matrix $W^{(l)}$ and vector $b^{(l)}$ are called the weights and biases of the lth layer. Common choices for the activation function are the sigmoid $\text{sig}(x) = 1/(1 + e^{-x})$, hyperbolic tangent $\tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$, or the rectified linear unit $\text{ReLU}(x) = \max(0, x)$, each of which may be suitable for different purposes.

The defacto procedure for determining the parameters $W^{(l)}$ and $b^{(l)}$ for each layer is similar to the batch gradient descent procedure outlined earlier. The only additional consideration is that to calculate the gradient of the loss with respect a particular parameter in the lth layer of a neural network, we must consider the functional dependencies in layers $l + 1, l + 2, \ldots$ up to the loss. This leads us to first calculate the gradients of parameters starting at the *last* layer in the network, and accumulate gradients "backwards" through the network via the chain rule. This backwards accumulation of gradients is called backpropagation [10].

In general, neural networks provide a flexible way to model data x_i, y_i, S_{ij} with unknown relationships in a way that is relatively straightforward: we simply learn parameterized relationships through optimization. Moreover, since neural networks are universal function approximators [11], we can potentially learn relationships of arbitrary complexity. In many settings, this flexibility allows us to fruitfully utilize large amount of unstructured data, ultimately leading to increased performance in downstream tasks such as classification or regression. This is in contrast to traditional statistical or physical models which may require transformation and sub-setting of our data prior to modelling and analysis. For example, it might be non-sensical to try to directly incorporate text or images into such models, and we would need to subset our data or transform it in order to make it comparable with some other quantities. Replacing statistical or physical models with deep learning models, however, has the known drawback of making model interpretation unclear if not impossible all together. For example, the weights and biases (commonly denoted $W^{(l)}$ and $b^{(l)}$ for the lth layer) may be less interpretable than the coefficients β_i in linear regression. This may exclude neural networks as a viable model choice in many settings where interpretable parameters are not only desirable, but the main purpose of the model. To this end there has been great amount of work in explaining machine learning models, so called explainable AI (XAI). We refer to Ref. [12] for a survey.

Even if interpretability were not a concern, there is another well known limitation of neural networks: data requirements. A rule of thumb is that for each parameter of a model, one should have an order of magnitude more data samples. For a linear model with 10 parameters, we would want at least 10^2 data samples. For a neural network with 10^5 parameters, we would need at least 10^6 data samples, and so on. In many situations, acquiring this amount of data is either impractical if not impossible altogether. However, depending on the problem at hand, we may be able to intelligently introduce some bias into the model such that we can actually train a neural network with far less than 10 samples/parameter *and* achieve better model performance. These biases can be classified into three main categories as follows [13]:

Observational Bias: selection/augmentation of data such that it densely covers the desired function domain.

Inductive Bias: The design of our model architecture is such that it favors the desired function.

Learning Bias: Augmenting the loss function such that the training procedure is guided towards our function.

Some of the most celebrated forms of bias in deep learning are inductive biases: Recurrent neural networks assume a causal structure of sequential features, while convolutional neural networks assume that features are translation and rotation invariant. Graph Neural Networks are designed such that operations applied to nodes as a function of A_{ij} are permutation invariant. So instead of learning specific relationships for each pair of adjacent nodes, GNNs learn a single transformation by which we treat all pairs of nodes. This treatment also serves as a solution to (III), naturally incorporating A_{ij} into a neural network.

Physics Informed Neural Networks introduce physical models specified by differential equations into deep neural networks by specifying the differential equations as learning biases. That is, we essentially penalize the neural network when its output function does not satisfy a specified differential equation. In this way we can implicitly use neural networks as a solver for differential equations [13]. So in some sense, physics can serve deep learning by introducing learning biases which make training more efficient, and machine learning can serve physics by providing a flexible solver which easily interfaces with arbitrary data. PINNs are therefore a natural choice for complex systems which often have extremely rich and varied data *and* differential equations that describe the system. In practice, a class of candidate differential equations might be derivable independently of the data through some first principles mathematical derivation based on plausible microscopic mechanisms that respect known scientific principles of the system. Hence this class of differential equations can be regarded as coarse-grained or 'mean-field' approximations, or they may come from some more phenomenological argument – for example, generalized Burgers-like equations of the form discussed later in equations (14) and (16) which are themselves akin to differential equations commonly used in fluid dynamics, turbulence and hypersonics.

2. GRAPH NEURAL NETWORKS

Graph neural networks have allowed fruitful applications of neural networks in settings which have graph structured data including telecommunications, biological, and social networks. In physics, there have recently been successful applications of GNNs in particle physics [14] and astrophysics [15]. The utility of applying GNNs in the context of complex systems with a graph structure has several parts, namely they allow us to do the following

(A) efficiently define a dense representation of each node in an embedding space

(B) systematically incorporate information from nearby nodes to facilitate nodewise predictive tasks

(C) transfer knowledge learned from one graph to another graph without requiring retraining

(D) define a meaningful coordinate systems for graphs which otherwise have an ambiguous meaning of "space"

Point (A) is a result of the relational inductive bias of GNNs. Point (B) and (C) result from the fact that GNNs treat all pairs of nodes in the same way, i.e. GNN operators are invariant under permutation of nodes, allowing them to generalize learned relationships. Point (D) will be particularly relevant for the next chapter on PINNs, as we need some notion of space to define spatial operators ∂_x and ∇ in differential equations.

2.1 Graph Convolutional Networks

One of the most popular types of GNNs is the Graph Convolutional Network (GCN) [16], with layers defined as

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{10}$$

where $\tilde{A} = A + I$ is the adjacency matrix with inserted self-loops, $\tilde{D}_{ii} = \sum_j \hat{A}_{ij}$ is the diagonal degree matrix, $W^{(l)}$ is the weight matrix for the layer, and σ is an activation function. The operator

$\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ essentially acts like a convolutional filter on $H^{(l)}$: $\tilde{A} = A + I$ dictates that the convolution is with respect to each nodes' neighbors and itself; the factors of $\tilde{D}^{-\frac{1}{2}}$ enforces the resulting convolution such that it has values in the range [0,1] for numerical stability. The role of $W^{(l)}$ and σ then follow the same logic as an MLP.

2.2 Graph Attention Networks

One problem with GCNs is that they treat all relationships defined by A_{ij} as being the same, which in many cases (such as social networks) is an unrealistic assumption. Graph attention networks (GAT) [17] address this issue by learning attention coefficients α_{ij} which essentially reweights A_{ij} as a function of the last layer's features. Specifically, if we define a matrix $(\mathcal{A})_{ij} = \alpha_{ij}$ then we can write the GAT layers as

$$\begin{aligned} H^{(l+1)} &= \sigma\left(\mathcal{A}H^{(l)}W^{(l)}\right), \\ \alpha_{ij} &= \sigma_a(a^T[W_a h_i^{(l)}||W_a h_j^{(l)}]) \end{aligned} \tag{11}$$

where $h_i^{(l)} \in \mathbb{R}^{k_l}$ is the embedding of the ith node at layer l, $||$ denotes concatenation, $a \in \mathbb{R}^{2k_l}$ and $W_a \in \mathbb{R}^{k'_l \times k_l}$ are attention weights, and σ_a is the attention activation. Not only does reweighting the adjacency matrix make GATs more realistic, the attention mechanism can naturally models sequences of data, whether they are a sequence of words in a sentence or a sequence of values in a time series, by learning the appropriate weights between each position in the sequence.

3. PHYSICS INFORMED NEURAL NETWORKS

There are several theorems which state that neural networks of sufficient width or depth can approximate any function of the input parameters [11]. For this reason deep neural networks are called universal approximators. Interestingly, this implies that the solution to any differential equation can be learned by a deep neural network. Say we have a function $u(t,\vec{x})$ obeying the heat equation

$$u_t = \alpha\nabla^2 u \tag{12}$$

and N data points across the spatiotemporal domain $\{u(t_i,\vec{x}_i)|i = 1, ..., N\}$. Following the PINN methodology, we can train a neural network to approximate the function $u(t,\vec{x})$ as in FIGURE 3.

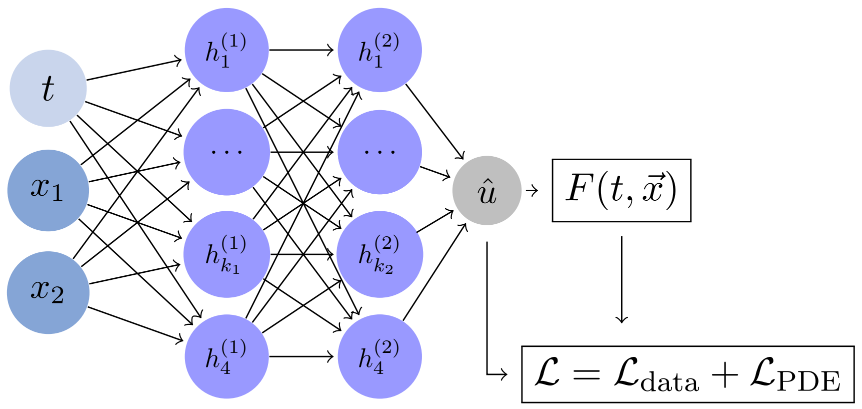

Figure 3: A PINN architecture

where we define the following quantities

$$\begin{aligned} &\text{PDE residual: } F_i = [\hat{u}_t - \nabla^2 \hat{u}]|_{t=t_i, \vec{x}=\vec{x}_i} \\ &\text{PDE loss: } \quad \mathcal{L}_{\text{PDE}} = w_{\text{PDE}} \sum_i |F_i|^2 \\ &\text{data loss: } \quad \mathcal{L}_{\text{data}} = w_{\text{data}} \sum_i (u(t_i, \vec{x}_i) - \hat{u}(t_i, \vec{x}_i))^2 \end{aligned} \tag{13}$$

A notable innovation that allows this methodology to work is that the derivatives with respect to t and $\vec{x}$ in the PDE residual can be determined via backpropagation, i.e. can be calculated by accumulating gradients through each successive layer of the neural network via the chain rule.

The PINN methodology basically amounts to enforcing physical models as a learning bias or soft constraint. An alternative approach is to actually design a neural network architecture such that certain aspects of our PDE/system are automatically satisfied such as boundary conditions [18], multiscale features [19], and symmetries/ conservation laws [20]. We can still add these considerations into PINNs as additional learning biases, and doing so requires no modification or amendment of our architecture. The chief advantages of PINNs compared to more tailored architectures include

mesh-free: they can incorporate and interpolate irregularly sampled time and space domains

inverse and ill-posed problems: for a model $u_t - \alpha \nabla^2 u = g(u; t, \vec{x})$, we do not need to know α nor the function $g(u; t, \vec{x})$ in advance, and instead make them learnable parameters

flexibility: the simplicity of the PINN approach makes interfacing with various data and architectures easy

The learnable function $g(u; t, \vec{x})$ in our model is frequently referred to as "hidden" physics. Such a notion becomes highly relevant when we turn our attention to complex systems, where we typically cannot even derive or identify the correct differential equations from the first principles mechanisms which are actually present in the real-world system of interest.

4. A NEURAL ARCHITECTURE FOR COMPLEX SYSTEMS

For the purposes of defining learnable physics of complex systems, we first need to define some notion of spatial coordinates on an irregular graph structure A_{ij}. For example, it might not be obvious how to define such coordinates for a graph such as in FIGURE 4.

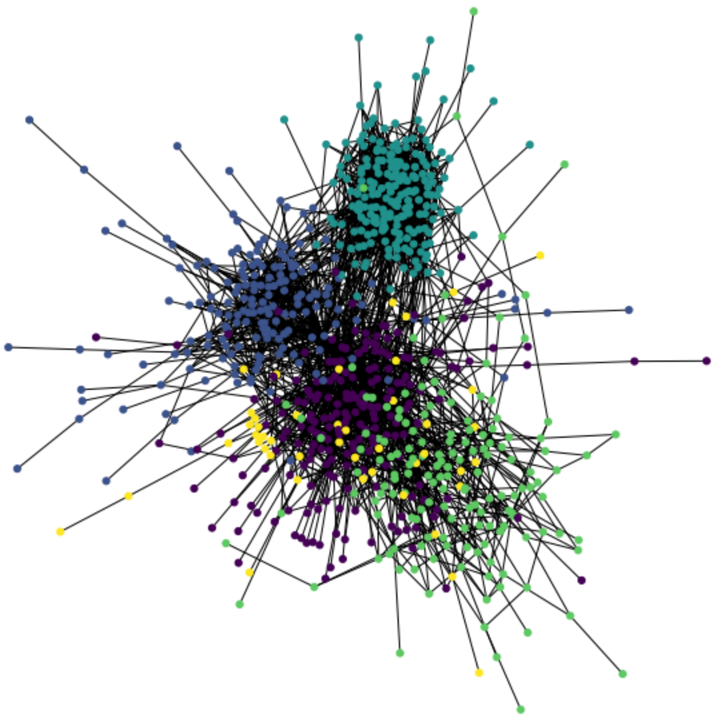

Figure 4: A graph with 500 nodes. Communities are denoted by different colors and are determined by the Louvain algorithm for community detection [21]

.

These coordinates should ideally be low-dimensional and preserve standard notions of distance and similarity within the graph (i.e. graphs which are close topologically should be close in the learned coordinate space). GNNs are well suited to this purpose [22]. Further, we can compose a GNN, or any encoder, with a PINN as in FIGURE 5 such that we jointly learn coordinates and the output function of the PINN. In modern language, such a joint learning approach is frequently called "end-to-end" learning.

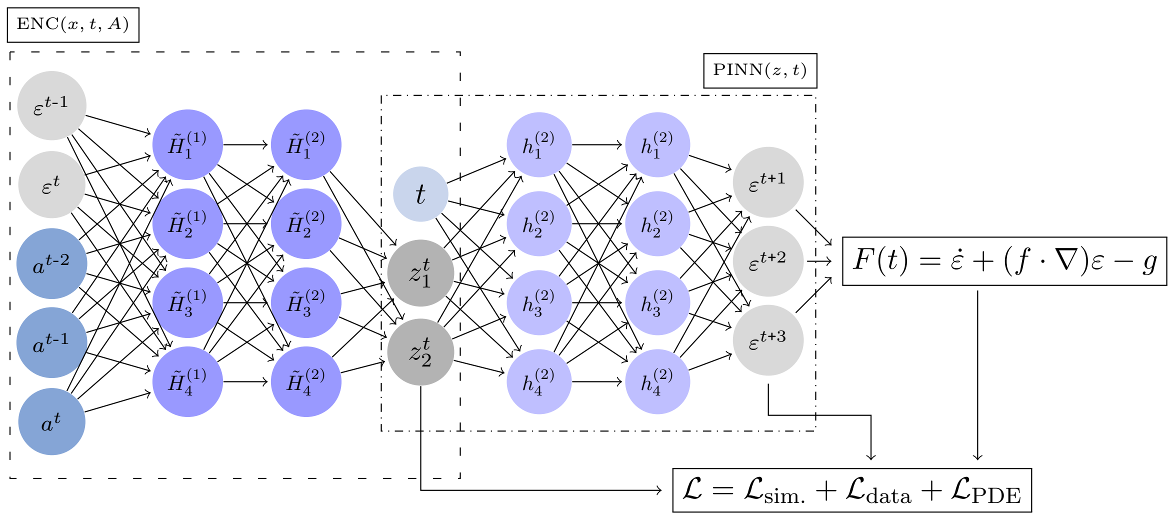
ENC(x, t, A)
PINN(z, t)
ε^{t-1}
ε^{t}
a^{t-2}
a^{t-1}
a^{t}
$\tilde{H}_1^{(1)}$ $\tilde{H}_2^{(1)}$ $\tilde{H}_3^{(1)}$ $\tilde{H}_4^{(1)}$
$\tilde{H}_1^{(2)}$ $\tilde{H}_2^{(2)}$ $\tilde{H}_3^{(2)}$ $\tilde{H}_4^{(2)}$
t
z_1^t
z_2^t
$h_1^{(2)}$ $h_2^{(2)}$ $h_3^{(2)}$ $h_4^{(2)}$
ε^{t+1}
ε^{t+2}
ε^{t+3}
$F(t) = \dot{\varepsilon} + (f \cdot \nabla)\varepsilon - g$
$\mathcal{L} = \mathcal{L}_{\text{sim.}} + \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{PDE}}$

Figure 5: A PINN with an coordinate encoder ENC(x, t, A). For systems with graph structure we might choose an encoder with GCN or GAT layers. We show here a lagged covariate input window of size 3 $\{a^{t-1}, a^{t-2}, a^{t-3}\}$, a lagged autocovariate window of size 2 $\{\varepsilon^{t-1}, \varepsilon^{t}\}$ and an output window of size 3 $\{\varepsilon^{t+1}, \varepsilon^{t+2}, \varepsilon^{t+3}\}$.

4.1 Our Proposed Approach

In order to provide a concrete challenge example around which we can illustrate our discussion, we propose here to apply the architecture in FIGURE 5 to forecast the population $G_i(t)$ of pages on social media with an associated graph A_{ij} (i.e. the number of "followers" on a page.). Based on a first-principles mathematical analysis of many-body aggregation that does not concern us here, an approximate model for the evolution of $G_i(t)$ as an aggregation process is given by [23]

$$G_i(t) = N(t) - \varepsilon_i(0, t)$$

$$\frac{\partial \varepsilon_i}{\partial t} + \frac{2F_i}{N(t)}\left[1 - \frac{\varepsilon_i}{N(t)}\right]\frac{\partial \varepsilon_i}{\partial z} = e^z \frac{dN(t)}{dt} \tag{14}$$

where $N(t)$ is a monotonically increasing reservoir of the system which is known in advance and $G_i(t)$ is the population data value for each node i at a given time t (FIGURE 6). The form of equation (14) is derived from several plausible assumptions about how individuals form into groups online as described in Ref. [23].

In order to produce a coupled system of equations with each component equation described by (14), we first start with independent realizations of (14) assigned to each node in the system (FIGURE 6).

We then couple the G_i and backcast a nodewise predictor variable a_i through successive iterations of

$$\begin{aligned} \dot{G}_i^{k+1}(t) &= \dot{G}_i^k(t) + \sum_{j,t'} w_{ij}\dot{G}_j^k(t-t')\delta_j \\ a_i^{k+1}(t) &= \sum_{t'} \sigma\left(\dot{G}_i^k(t+t')\right)e^{-\lambda t'}\delta_{t'} + \sum_{j,t'} w_{ij}\sigma\left(a_j^k(t-t')\right) \end{aligned} \tag{15}$$

with $\delta_j, \delta_{t'} \sim$ uniform(0,1) and $w_{ij} \sim A_{ij}\times$ uniform(0,1). The form of $a_i^{k+1}(t)$ is such that future $G_i(t+t')$ can be predicted from $a_i(t)$ through a nonlinear ($\sigma(\cdot)$), diffuse ($e^{-\lambda t'}$), and noisy ($\delta_{t'}$) process. Furthermore, although $\dot{G}_i^{k+1}(t)$ depends linearly upon $\dot{G}_j^k(t-t')$, this dependence needs to be determined from a nonlinear contribution $\sum_{j,t'} w_{ij}\sigma\big(a_j(t-t')\big)$ to a_i^{k+1}.

The resulting ε_i^k implied by (15) may in principle be calculable, but for the purposes of generalizing to real world systems where we do not know the exact form of the dynamics in advance, we instead propose the following generalization of (14)

$$\begin{aligned} G_i(t) &= N(t) - \varepsilon_i(0,t), \\ \frac{\partial \varepsilon_i}{\partial t} &+ (f_i \cdot \nabla)\varepsilon_i = g_i, \end{aligned} \tag{16}$$

where $f_i(z,t) \equiv f(z,t,h_i^{(1)},...,h_i^{(L)})$, $g_i(z,t) \equiv g(z,t,h_i^{(1)},...,h_i^{(L)})$ and f and g are learnable functions.

Notable advantages of this architecture over other forecasting methods (described in the next section) include the following:

1. there is a natural incorporation of graph structure A_{ij}
2. it is easy to generalize partial physics knowledge: e.g. from (14) to (16)
3. scalability: the model size does not need to grow with the system size
4. inductive: forecasting for unseen nodes does not require retraining/refitting of any parameters

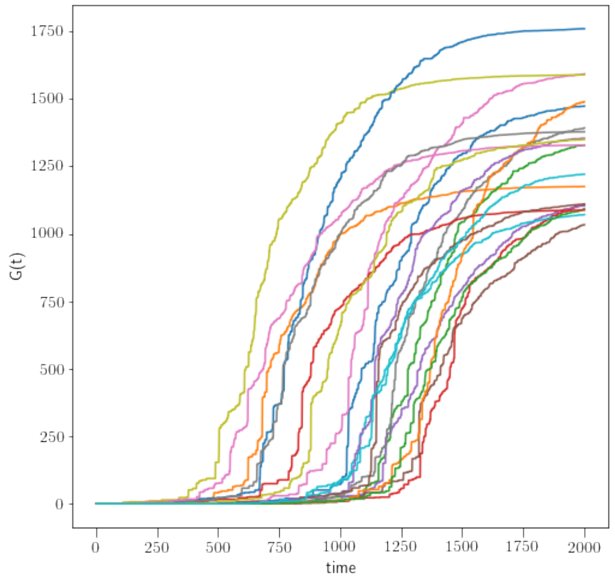

Figure 6: Population data $G_i(t)$ for 20 nodes in the system.

Ultimately, the challenge would be to:

1. Generalize partial physics knowledge of social systems, i.e. equation (14) to (16)
2. Forecast $G_i(t)$ over long horizons (e.g. 30-60 timesteps)(FIGURE 6)

5. PROOF-OF-PRINCIPLE EXPERIMENT

As a measure of performance for this architecture on forecasting, we can calculate the root mean squared error of τ steps ahead forecasting for a set of nodes $j \in \Omega$ with $N = |\Omega|$ as

$$\mathrm{RMSE}^{(\tau)}(\Omega) = \sum_{t \in \{T\}} \sum_{j \in \Omega} \sqrt{\left(y_j^{t+\tau} - G_j(t+\tau)\right)^2 / N}. \tag{17}$$

where $\{T\}$ is a random sample over some subset of the time domain and y_j^t is the timeseries data for which $G_j(t)$ is a predictor. Then if we define a set of training nodes as $j \in \Omega$ and a set of nodes as $j \in \Omega'$ with $\Omega \cap \Omega' = \emptyset$ we have the following

$$\begin{aligned} \mathrm{RMSE}^{(\tau)}_{\text{val.}} &= \mathrm{RMSE}^{(\tau)}(\Omega) \\ \mathrm{RMSE}^{(\tau)}_{\text{test.}} &= \mathrm{RMSE}^{(\tau)}(\Omega') \end{aligned} \tag{18}$$

As a preliminary comparison, we test a PINN with a GCN encoder against two simple baselines: a vanilla NN and a naive forecast. We use as predictors a κ lagged autocovariate window $\{y_j^{t-\kappa}, ..., y_j^t\}$

and covariate window $\{a_j^{t-\kappa}, ..., a_j^t\}$ and predict a single $\tau = 30$ step ahead forecast $G_j(t+\tau)$. For the naive forecast we linearly extrapolate the lagged window:

$$G_j(t+\tau) = y_j^t + \tau \frac{(y_j^t - y_j^{t+\kappa})}{\kappa}. \tag{19}$$

The vanilla NN consists of three hidden layers with 64 hidden units each. We test on a system of $N = 100$ nodes with an 80/20 test/train split and over a random sample of 200 points in the time domain $500 < t < 1500$.

For the PINN + GCN we implement the system of equations in (14), use 2 hidden layers with 32 hidden units in the encoder, 3 hidden layers with 48 hidden units in the PINN, an input window $\kappa = 20$ for the covariates, and a latent dimension $z \in \mathbb{R}^8$. For both the vanilla NN and the PINN + GCN we use the Adam optimizer with learning rate 10^{-3} (TABLE 1).

Table 1: Comparison on $\tau = 30$ step ahead forecasting on a validation and test set.

model	$\text{RMSE}_{\text{val.}}^{(\tau=30)}$	$\text{RMSE}_{\text{test.}}^{(\tau=30)}$
naïve	35.99	34.64
NN	16.20	51.75
PINN + GCN	12.87	16.23

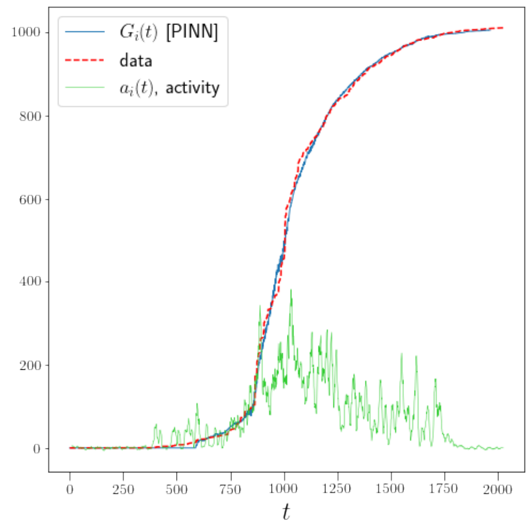

Figure 7: A PINN + GCN forecast for a node in the validation set. Note the robustness of the forecast to noise in the data y_i^t and the covariate a_i^t.

6. DISCUSSION AND FUTURE WORK

Our preliminary results (e.g. FIGURE. 7) are indeed promising for the PINN architecture with a GNN encoder. In particular, we see excellent generalization from the validation set to the test set as compared with the vanilla NN. This is understandable since we expect both a GNN encoder and PDE to allow us to make more intelligent forecasts based on graph structure and dynamics.

To better understand what elements of the architecture provide benefit to a forecasting task, we will in the future need to implement a wider range of PINN + DEC models, as well as compare with a variety of timeseries forecasting models. Relevant comparisons could be made with traditional techniques such as vector autoregression (VAR), neural time-series forecasting architectures like N-BEATS [24], and dynamic GNNs [25,26]. Additionally, we also need to compare with other techniques that perform simultaneous regression and governing equation discovery, i.e. system identification methods such as NARMAX [27] and SINDy [28].

Additionally, we need to incorporate into our architecture an appropriate error model (i.e. along with each forecast $G_i(t)$ there is an accompanying error estimate). Uncertainty quantification for PINNs is fairly well studied at this point, so we can simply adapt one of the methods described in Ref. [29].

7. CONCLUSION

Our paper has tried to confront the problem that no unified approach yet exists to describing complex systems, and to leverage how particular neural architectures seem poised to provide scalable models which accurately describe them. Specifically, we proposed an end-to-end deep learning approach to modelling complex systems which addresses two problems: (1) scientific model discovery when we have only incomplete/partial knowledge of system dynamics; (2) integration of graph-structured data into SciML using graph neural networks. The present work still needs to be greatly expanded to investigate the effectiveness of our approach. However we see value in starting discussions about how distinct tools in machine learning and artificial intelligence can be combined in a way to overcome the significant and urgent challenge of modeling real-world complex dynamical systems.

8. ACKNOWLEDGMENTS

We are grateful to Prof. George Karniadakis for very valuable discussions about PINNs. We are also grateful for funding for this research from the U.S. Air Force Office of Scientific Research under award numbers FA9550-20-1-0382 and FA9550-20-1-0383. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

References

[1] Carlos Gershenson. Self-Organization and Emergence in Life Sciences. Artificial Life. 2008; 14:239–240.

[2] Jr. Smalley RF, Turcotte D L, S. A. Solla. A Renormalization Group Approach to The Stick-Slip Behavior of Faults. Journal of Geophysical Research. 1985;90(B2):1894–1900.

[3] Nicholas WW, Pruessner G, Chapman SC, Crosby N B, Jensen HJ. 25 Years of Self-Organized Criticality: Concepts and Controversies. Space Science Reviews. 2016;198:3–44.

[4] Plenz D, Ribeiro TL, Miller SR, Kells PA, Vakili A, et al. Capek. Self-Organized Criticality in the Brain. Frontiers in Physics. 9, 2021.

[5] Broido AD, Clauset A. Scale-Free Networks Are Rare. Nature Communications. 2019;10.

[6] Holovatch Y, Kenna R, Thurner S. Complex Systems: Physics Beyond Physics. European Journal of Physics. 2017;38:023002.

[7] Kovács B, Palla G. The Inherent Community Structure of Hyperbolic Networks. Scientific Reports. 2021;11:16050.

[8] Altszyler E, Ribeiro S, Sigman M, Fernández Slezak D. The Interpretation of Dream Meaning: Resolving Ambiguity Using Latent Semantic Analysis in a Small Corpus of Text. Consciousness and Cognition. 2017;56:178–187.

[9] https://arxiv.org/pdf/1906.06821.pdf

[10] Yu X, Efe MO, Kaynak O. A General Backpropagation Algorithm for Feedforward Neural Networks Learning. IEEE Transactions on Neural Networks. 2002;13:251–254.

[11] Hornik K, Stinchcombe M, White H. Multilayer Feedforward Networks Are Universal Approximators. Neural Networks. 1989;2:359–366.

[12] Adadi A, Berrada M. Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). IEEE Access. 2018;6:52138–52160.

[13] Em Karniadakis G, Kevrekidis IG, Lu L, Perdikaris P, Wang S, et al. Physics-Informed Machine Learning. Nature Reviews Physics. 2021;3:422–440.

[14] Shlomi J, Battaglia P, Vlimant JR. Graph Neural Networks in Particle Physics. Machine Learning: Science and Technology. 2021;2:021001.

[15] https://arxiv.org/pdf/2111.08683.pdf

[16] https://arxiv.org/pdf/1609.02907.pdf

[17] https://arxiv.org/abs/1710.10903, 2017.

[18] McFall KS, Mahan JR. Artificial Neural Network Method for Solution of Boundary Value Problems With Exact Satisfaction of Arbitrary Boundary Conditions. IEEE Transactions on Neural Networks. 2009;20:1221–1233.

[19] Liu Z. Multi-Scale Deep Neural Network (MscaleDNN) For Solving Poisson-Boltzmann Equation in Complex Domains. Communications in Computational Physics. 2020;28:1970–2001.

[20] https://arxiv.org/abs/1904.08991

[21] Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast Unfolding of Communities in Large Networks. Journal of Statistical Mechanics: Theory and Experiment. 2008;2008:P10008.

[22] https://arxiv.org/abs/1806.01261

[23] https://arxiv.org/abs/2108.01940

[24] https://arxiv.org/abs/1905.10437

[25] https://arxiv.org/abs/2005.11650

[26] Yu B, Yin H, Zhu Z. Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting. In Proceedings of the Twenty-Seventh. International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence Organization, July 2018.

[27] Rahrooh A, Shepard S. Identification of Nonlinear Systems Using Narmax Model. Nonlinear Analysis: Theory, Methods Applications. 2009;71:e1198–e1202.

[28] Rudy SH, Brunton SL, Proctor JL, Kutz JN. Data-Driven Discovery of Partial Differential Equations. Science Advances. 2017;3:e1602614.

[29] https://arxiv.org/abs/2201.07766