Finite Difference Neural Networks: Fast Prediction of Partial Differential Equations

Zheng Shi∗†¶, Nur Sila Gulgec¶, Albert S. Berahas§¶, Shamim N. Pakzad‖‡ and Martin Takáč¶

Department of Industrial & Systems Engineering, Lehigh University, Bethlehem, USA.
†IBM, Armonk, USA. ‡Thornton Tomasetti, San Francisco, USA.
§Department of Industrial & Operations Engineering, University of Michigan, Ann Arbor, USA.
‖Department of Civil & Environmental Engineering, Lehigh University, Bethlehem, USA.

Email: {shi.zheng.tfls, sgulgec, albertberahas, takac.mt}@gmail.com, ‡pakzad@lehigh.edu.

Abstract—Discovering the underlying behavior of complex systems is an important topic in many science and engineering disciplines. In this paper, we propose a novel neural network framework, finite difference neural networks (FD-Net), to learn partial differential equations from data. Specifically, our proposed finite difference inspired network is designed to learn the underlying governing partial differential equations from trajectory data, and to iteratively estimate the future dynamical behavior using only a few trainable parameters. We illustrate the performance (predictive power) of our framework on the heat equation, with and without noise and/or forcing, and compare our results to the Forward Euler method. Moreover, we show the advantages of using a Hessian-Free Trust Region method to train the network.

Index Terms—Finite difference neural networks, Partial differential equations, Hessian-free trust region method

I. INTRODUCTION

Partial differential equations (PDEs) are widely adopted in a plethora of science and engineering fields to explain a variety of phenomena such as heat, diffusion, electrodynamics, fluid dynamics, elasticity, and quantum mechanics, to mention a few. This is primarily due to their ability to model and capture the behavior of complex systems as well as their versatility. However, solving PDEs is far from a trivial task. Often incredible amounts of computing power and time are required to get reasonable results, and the methods used can be complicated and highly-sensitive to the choice of parameters.

The rapid development in data sensing (collection) and data storage capabilities provides scientists and engineers with another avenue for understanding and making predictions about these phenomena. The massive amounts of data collected from highly complex and multi-dimensional systems have the potential to provide a better understanding of the underlying system dynamics.

Utilizing this abundance of data to solve PDEs has been exploited in several recent studies; see e.g., [1]–[10]. In [1], [2] and [3], [4], the authors applied symbolic regression and sparse regression techniques, respectively, to explain non-linear dynamical systems. In [5], [6], the authors introduced physics informed neural networks using Gaussian processes. Moreover, Chen et al. [7] proposed continuous-depth residual networks and continuous-time latent variable models to train ordinary neural networks. Finally, in [8], the authors proposed conditional generative adversarial networks to predict solutions for steady state heat conduction and incompressible fluid flow, and, in [9], [10], the authors proposed PDE-Net inspired by Wavelet theory to approximate the unknown nonlinear responses of diffusion and convection processes. Possibly the closest work to ours is PDE-Net [9], [10], which the authors proposed to learn differential operators by learning convolution filters. The key differentiating features of our approach can be summarized as follows: (i) our approach is computationally efficient since it trains finite difference inspired, small and linear filters; (ii) our network architecture can be adapted and enhanced to learn PDEs with forcing; and, (iii) we use a second-order optimization method to improve accuracy and computational time of training.

In this paper, inspired by finite-difference approximations and residual neural networks [11], we propose a novel neural network framework, finite difference neural networks (FD-Net), to learn the governing partial differential equations from trajectory data, and iteratively estimate future dynamical behavior. Mimicking finite-difference approximations, FD-Net employs “finite-difference” block(s) (FD-Block) with artificial time steps to learn first-, second- and/or higher-order partial derivatives, and thus learn the underlying PDEs from neighboring spatial points over the time horizon. As a proof-of-concept, we deploy our proposed method to learn and predict the underlying dynamics of PDEs using trajectory data from the heat equation in different cases: (1) simple homogeneous heat equation; (2) heat equation with noise; and, (3) heat equation with a forcing term.

Stochastic first-order methods have been very successful in training machine learning models in various applications [12]. However, there are several drawbacks to using such methods, and it has been shown that, for certain applications, employing stochastic second-order methods can be beneficial [13]–[17]. In this paper, we show that training our networks is one such application; training time can be significantly reduced and the accuracy of the solutions can be drastically improved by using a second-order method. Specifically, we employ a second-order Hessian-Free method, Trust-Region (TR) Newton Conjugate Gradient (CG) [18], [19].

The paper is organized as follows. In Section II, we introduce the PDE used in our case study and discuss the four
different classes of problems that we investigate. We discuss in
detail the fundamentals of FD-Net in Section III. Extensive
numerical results are presented in Section IV. Finally, in
Section V, we make some concluding remarks and discuss
avenue for future research.

II. THE HEAT EQUATION

Consider a linear partial differential equation (PDE) in
canonical form:
\[ \mathcal{F}(x, t, u, u_t, u_x, u_{xx}, u_{xxx}, \ldots) = 0, \]  
(1)
where \( \mathcal{F} \) is a linear function of \( u \) and its partial derivatives with
respect to time and/or space. The objective of our study is to
implicitly learn \( \mathcal{F} \) given a series of measurements (trajectory
data) at specific time and spatial instances, and predict the
solutions to the equation throughout the time horizon.

For our case study, we consider the heat equation, one of
the most frequently used PDEs in physics, mathematics, engi-
neering and more. The heat equation describes the evolution
of heat flow over time in an object [20]. Let \( u(x, t) \) denote the
temperature at a spatial point \( x \) and time \( t \). The heat equation
for a 1-D bar of length \( L \) can be expressed as
\[ \frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2}, \]  
(2)
where \( \beta \) is a rate of diffusion of the material. Under the
assumption of perfectly insulated boundaries, the boundary
conditions (BCs) can be expressed as
\[ u(0, t) = 0, \quad u(L, t) = 0. \]  
(3)
We consider the following initial condition (IC)
\[ u(x, 0) = \sum_{i=1}^{N} C_i \sin\left(\frac{i\pi x}{L}\right), \]  
(4)
where \( C_i \in \mathbb{R} \) for \( i \in \{1, 2, \ldots, N\} \). The exact solution of (2)
with BCs (3) and IC (4) can be expressed as
\[ u(x, t) = \sum_{i=1}^{N} C_i \sin\left(\frac{i\pi x}{L}\right)e^{-\beta(i\pi/L)^2t}. \]  
(5)
The reasons we choose this PDE are three-fold: (1) it is
an extensively used PDE that will allow us to investigate
the merits and limitations of our proposed approach; (2)
although the PDE is simple, it has several characteristics
that are interesting to investigate (e.g., first- and second-order
derivatives) and the behavior of the PDE can be complex in
the presence of noise and/or a forcing term; and (3) we can
derive the exact solution.

Given \( x \in [0, L] \) and \( t \in [0, T] \), (2) can be approximately
solved via forward Euler method [21]. To this end, the domain
is discretized (both in \( x \) and \( t \) and \( u(x, t) \) is computed recursively as follows:
\[ u(x, t + \Delta t) = u(x, t) + \frac{\Delta t}{\Delta x}\left[ u(x + \Delta x, t) - 2u(x, t) + u(x - \Delta x, t) \right], \]  
(6)
where \( \delta = \beta \frac{\Delta x}{(\Delta x)^2} \).

The performance of the Euler method, in terms of the
accuracy of the solution, is highly dependent on the choice
of the granularity of the discretization, both in time \( (\Delta t) \) and
space \( (\Delta x) \). Specifically, the Euler method fails to generate
accurate approximations, and may even diverge, if \( \Delta t \) and
\( \Delta x \) do not satisfy \( \delta \leq 0.5 \), known as the stability criterion
[22]. We should note that of course higher-order numerical
procedures (or even implicit schemes) exist and could be used
to solve (2) and mitigate some of the stability problems at the
cost of more complex updates.

In addition to the instability associated with sparsely dis-
etrazed time steps, in real-life applications the measurements of
\( u(x, t) \) are often contaminated with noise, e.g., Gaussian
noise \( \epsilon \sim N(\mu, \sigma^2) \), which can severely impact the stability
and quality of the solutions. Moreover, the PDE could also
have a forcing term, e.g.,
\[ \frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2} + f(x) \quad \text{and} \quad f(x) = \sum_{i=1}^{N} D_i \sin\left(\frac{i\pi x}{L}\right), \]  
(7)
where \( D_i \in \mathbb{R} \) for \( i \in \{1, 2, \ldots, N\} \). The exact solution of (7)
is
\[ u(x, t) = \sum_{i=1}^{N} \left( C_i - \frac{D_i}{\pi(i\pi/L)^2} \right) \sin\left(\frac{i\pi x}{L}\right)e^{-\beta(i\pi/L)^2t} + \sum_{i=1}^{N} \frac{D_i}{\pi(i\pi/L)^2} \sin\left(\frac{i\pi x}{L}\right). \]  
(8)
The challenges that arise from instability, noisy measure-
ments and forcing terms can make conventional approaches,
such as the Euler method, vulnerable and result in inaccurate
approximations. These challenges have inspired researchers in
the fields of computational mathematics and machine learning
to develop solution techniques that utilize the power of deep
neural networks and exploit the massive amounts of measure-
ments (i.e., trajectory data) that are readily available to solve
PDEs and make predictions.

In this paper we investigate the performance of our pro-
sed method, FD-Net, on the heat equation for all four
aforementioned cases: (1) stable; (2) unstable; (3) noisy; and,
(4) forced.

III. FUNDAMENTALS OF FD-Net

In this section, we describe the fundamental components of
FD-Net.

The building blocks of FD-Net are FD-Blocks, whose
design is inspired by finite-difference approximations of partial
derivatives. Fig. 1 shows an instance of FD-Block. An
FD-Block is a deep residual learning block [11] that aims to
learn the evolution of a dynamical system for one artificial time
step on \([t, t + \Delta t]\). It is composed of groups of convolutional
layers, a fully connected (FC) layer, and a multi-step skip
connection.

Specifically, for each group of convolutional layers, a cer-
tain number of “finite-difference” filters (FD-Filters) are
defined in space: for \( x \in \{\Delta x, 2\Delta x, \ldots, L - \Delta x\} \), the size of
the filter is three (one parameter for \( x \) itself, one for its left
neighbor and one for its right neighbor); for the boundaries,
i.e., \( x = 0 \) or \( x = L \), the size of a filter is two as there is only
one neighbor, either on the left or right. The outputs of one
group of layers with FD-Filters are concatenated to form a
learned representation of partial derivatives of a certain order.
In order to capture and mimic higher-order partial derivatives,
FD-Block ∈ \mathbb{R}^{16}(1,3) introduces multiple groups of convolutional layers with such filters are employed. The representation from a previous group is used as input of the subsequent group in order to learn a higher-order representation. The learned representations of partial derivatives, by all groups, are then concatenated and passed as input to the FC layer in order to learn the evolution dynamics. Next, a skip connection is applied and the network proceeds to the following artificial time step.

Moreover, to imitate finite-difference approximations and to capture the behavior of linear equations, FD-Net defines the parameters of each layer without bias terms, and the outputs of the layers without applying nonlinear activation functions. In addition to the main architecture of the FD-Block, FD-Net constructs a learnable representation via an FC layer and concatenates it with the outputs of the convolutional layers to learn forcing functions that are potentially present in the PDE.

Overall, FD-Net is formed by stacking multiple FD-Blocks sequentially in order to produce an approximate solution of the PDE at $t + \Delta t$ given a solution at $t$. Incorporating $k (> 1)$ FD-Blocks introduces $k - 1$ artificial time steps between $t$ and $t + \Delta t$ to FD-Net, which enhances the learning capability of FD-Net, especially when $\Delta t$ is large. We discuss this further in Section IV. The number of FD-Blocks is the first hyper-parameter of FD-Net.

Furthermore, instead of defining distinct FD-Filters for each FD-Block, FD-Net shares the same FD-Filters along the sequence. As a result, the size of an instance of our networks does not depend on the number of FD-Blocks but rather on the number and size of the FD-Filters.

---

**Fig. 1: An illustration of FD-Block and the artificial time step.** In this particular instance, there are $k$ FD-Blocks defined in the network and thus $k - 1$ artificial time steps on $[t, t + \Delta t]$. For each FD-Block, there are 16 FD-Filters, two groups of convolutional layers, an FC layer, a forcing function representation and a skip connection. At the (artificial) time step $t + \frac{j\Delta t}{k}$ for $j = 0, 1, ..., k - 1$, the input $u(\cdot, t + \frac{j\Delta t}{k})$ is passed through the convolutional layers to learn the first- and second-order partial derivatives. Concatenated with the representation of the forcing function, the outputs are then passed through the FC layer with the skip connection to predict the function behavior at $t + \frac{(j+1)\Delta t}{k}$.

**TABLE I: Number of Parameters in FD-Net.**

| # Parameters w/o forcing | 4  | 8  | 16 | 32 | 64 |
|--------------------------|----|----|----|----|----|
| # Parameters w/o forcing $\dagger$ | 148 | 520 | 1936 | 7456 | 29248 |
| # Parameters $\ddagger$ | 468 | 840 | 2256 | 7776 | 29568 |

$\dagger$ does not count parameters for learning forcing function. $\ddagger$ includes all parameters in an instance of FD-Net, the forcing function is in the form of (7) with $N = 10, x \in [0, \pi]$ and $\Delta x = 0.1$.

---

FD-Net uses the same number of FD-Filters across all convolutional layers for a consistent input/output shape. We use this quantity as the second hyper-parameter of FD-Net and refer to it as "the number(s) of FD-Filters".

Table I shows the sizes of the networks, for different numbers of FD-Filters, used to learn PDEs with first- and second-order partial derivatives. Our experiments (see Section IV) indicate that 16 FD-Filters are sufficient for FD-Net to produce predictions with high precision for our case-study PDE for all aforementioned cases.

**IV. NUMERICAL EXPERIMENTS**

In this section, we present numerical experiments to demonstrate the empirical performance of FD-Net on the heat equation under the four scenarios described in Section II: (1) stable case; (2) unstable case; (3) noisy case; and, (4) forcing case. We first describe the data we used in our experiments, then discuss the optimization methods employed for training the networks and finally show numerical results.

The main goal of the experiments is to study if FD-Net is capable of making accurate predictions throughout the time horizon by solely relying on trajectory data and iteratively learning the short-term (i.e., between $t$ and $t + \Delta t$) evolutions. We illustrate the training and testing (prediction) performance of our proposed approach, compare the predictions made by FD-Net with the approximate solutions generated by the forward Euler method, demonstrate the advantages of training our networks with Trust-Region (TR) Newton CG method, and investigate the sensitivity of our networks to the hyper-parameters. For brevity, we only show a subset of our results in this paper and defer the full experimental results to the Appendix in [23].

**A. Data, Training and Testing**

For each aforementioned case, we generated synthetic data using the exact solutions in (5) and (8) to the heat equations; see Section II. Specifically, for each case, we generated 200
different trajectories, each with a randomly generated initial condition, i.e., $C_i \sim \mathcal{N}(0, 1)$, for $i \in \{1, 2, \ldots, 10\}$ in (4). We considered the 1D bar of length $L = \pi$, and the rate of diffusion parameter was set to $\beta = 2 \cdot 10^{-4}$. We set the spatial discretization to $\Delta x = 0.1$ on $[0, \pi]$, the time horizon as $[0, 1000]$, and the temporal discretization to $\Delta t = 1$ (namely, $\delta = 0.02 < 0.5$ in (6)), for the stable, noisy and forcing cases, and $\Delta t = 200$ (namely, $\delta = 4 > 0.5$ in (6)), for the unstable case. For the noisy case, we considered multiplicative noise of the form $u(x, t) = u(x, t)(1 + \gamma \epsilon_{x,t})$ for $i \in \{1, 2, 3\}$, where $\epsilon_{x,t}, \gamma \sim \mathcal{N}(0, 1)$, and $\gamma_1 = 10^{-8}$ (low), $\gamma_2 = 10^{-4}$ (medium) and $\gamma_3 = 10^{-2}$ (high). For the case with a forcing function, we generated the function with $D_i \sim \mathcal{N}(0, 1)$ for $i \in \{1, 2, \ldots, 10\}$ in (7) and applied it to the PDEs of all initial conditions.

We denote the randomly generated data set by $A = \{u_s(x, t) \mid s \in S, x \in \{0, \Delta x, \ldots, L\}, t \in \{0, \Delta t, \ldots, T\}\}$, where $S$ is the index set of ICs, $u_s(x, t)$ is the data (of measurements) and $\{u_s(x, t)\}_{s \in S}$ is the trajectory data for a specific IC. We randomly selected 150 ICs as our training set, and the remainder (50 ICs) were used for testing purposes. We denote $S_{\text{train}}$ and $S_{\text{test}}$ the subsets of indices of ICs for training and testing, respectively.

For training purposes, we adopted a “one-step ahead” procedure. Let

$$A_{\text{train}} = \{u_s(x, t), u_s(x, t + \Delta t) \mid s \in S_{\text{train}}, x \in \{0, \Delta x, \ldots, L\}, t \in \{0, \Delta t, \ldots, T - \Delta t\}\}$$

denote the training data set, where $(u_s(x, t), u_s(x, t + \Delta t))$ is a training tuple (sample), where $u_s(x, t)$ is the input and $u_s(x, t + \Delta t)$ is the target. We defined the MSE loss of the stochastic mini-batch as

$$\text{MSE}_\theta = \frac{1}{|A_\theta|} \sum_{s,x,t} (u_s(x, t + \Delta t) - \hat{u}_s(x, t + \Delta t))^2, \quad (9)$$

where $A_\theta \subseteq A_{\text{train}}$ is a mini-batch and $\hat{u}_s(x, t + \Delta t)$ is the output of a network. On the other hand, for testing, we used a “1000-step” sequential prediction procedure (and refer to it as “1000-step prediction”). Let

$$A_{\text{test}} = \{u_s(x, 0), u_s(x, T) \mid s \in S_{\text{test}}, x \in \{0, \Delta x, \ldots, L\}\}$$

denote the testing data set, $u_s(x, 0)$ be the input and $u_s(x, T)$ be the target. We used $u_s(x, 0)$ as the initial input and sequentially made predictions through the time horizon until reaching $T$, where the final prediction $\hat{u}_s(x, T)$ was made. Specifically, $u_s(x, 0)$ was used as the input to make the next prediction $\hat{u}_s(x, \Delta t)$, which in turn was used as the input to make the next prediction $\hat{u}_s(x, 2\Delta t)$, and this was repeated throughout the whole time horizon. The error metric we used was MSE and was defined as

$$\text{MSE}_{\text{test}} = \frac{1}{|A_\theta|} \sum_{s,x} (u_s(x, T) - \hat{u}_s(x, T))^2. \quad (10)$$

For each case, we configured the networks with different numbers of FD-Blocks and FD-Filters and used two optimization methods, ADAM (with learning rates $10^{-3}$ and $10^{-4}$) [24] and Trust-Region (TR) Newton CG method [18], both with mini-batch sizes of 64. We prescribed a fixed budget of 100 iterations for the TR method on the stable, noisy and forcing cases and 300 iterations on the unstable case, but allowed the ADAM algorithm to run for 12000 iterations on all cases. For each configuration, algorithm and case, we used 10 random seeds to initialize network parameters and to generate stochastic mini-batches.

Fig. 2: Evolution of training error. The marked dashes represent the average mini-batch MSE loss over 10 random seeds and the filled areas represent their 95% confidence intervals.

B. Results and Discussion

In this section, we present numerical results and discuss the strengths and limitations of FD-Net. We consider all the aforementioned cases: (1) stable; (2) unstable; (3) noisy (medium); and (4) forcing. For brevity, among all configurations investigated, we show results for the best configuration for each case (from an average performance perspective given the budget) and defer the results of other configurations in [23]. Specifically, we show the results for 1 FD-Block & 16 FD-Filters for the stable, forcing and noisy cases, and 10 FD-Blocks & 16 FD-Filters for the unstable case. Furthermore, we investigate the sensitivity of FD-Net to the hyper-parameters, i.e., numbers of FD-Blocks and FD-Filters.

We begin our presentation by showing the evolution of the training errors, i.e., $\text{MSE}_\theta$ (9), for different optimization algorithms in Fig. 2. We compare the performance of the algorithms in terms of the number of gradient and Hessian-vector computations. As is clear from the figure, the TR method is able to achieve smaller $\text{MSE}_\theta$ than ADAM within the given budget for all cases. This is true for other FD-Block and FD-Filter configurations, as well as different noise levels; see [23] for more results.

Having demonstrated that our networks can be adequately trained within a budget, we proceed to show the testing (prediction) accuracy of FD-Net and compare against a standard benchmark numerical scheme, the forward Euler method, in Fig. 3. This figure shows the sequences of predictions made by FD-Net, for the stable case at 5 time steps in the horizon; see results of other cases in [23]. We chose the sequences of
training our networks with the TR method using the TR method results in higher accuracy with FD-Net. 16 FD-Blocks suffices for the cases with small \( k = 1000 \) FD-Filters to choices varies by random seeds, and that utilizing a \( \Delta \) FD-Block reduces this variance. That FD-Block and \( \Delta \) FD-Filters models depends on the number of FD-Filters with a small number of \( \Delta = 0 \) FD-Filters size of \( k = 1000 \) is \( \Delta \) FD-Filters \( 200 \leq \) − artificial time steps to the time interval FD-Net FD-Blocks for one \( \in \{ \tilde{u} \}_{t} \) and \( 1 \in \{ u \}_{t} \) FD-Net FD-Blocks x FD-Net x FD-Filters \( t \in \{ x, t \} \) propagation throughout the time horizon, and any imperfect intermediate predictions can severely deteriorate the final prediction at \( t = 1000 \). This is evident for the networks trained by ADAM for the stable, noisy and forcing cases, where the testing errors for certain random seeds are very large. This effect is less severe for the unstable case as there are far fewer time steps from \( t = 0 \) to \( t = 1000 \). We should note, however, that the TR method is able to reduce the testing errors for all cases, and the effect of error propagation is not evident. This is true across the different cases so long as the networks are appropriately configured; see [23] for more details.

Next, we investigate the sensitivity of FD-Net to choices of the hyper-parameters (the numbers of FD-Blocks and FD-Filters). The main results are given in Fig. 5 and 6; see [23] for more results. To fully reveal the learning capabilities of FD-Net, we used the TR method without imposing any budget, and trained the network of each configuration with 10 random seeds.

As mentioned in Section III, stacking \( k \) FD-Blocks introduces \( k - 1 \) artificial time steps to the time interval \([t, t + \Delta t]\). While \( k = 1 \) suffices for the cases with small \( \Delta t \) (i.e., \( \Delta t \leq 1 \)), it is crucial to introduce a sufficient number of artificial time steps in order to achieve good training and testing performance in the setting where \( \Delta t \) is large. Fig. 5 shows the results of training the networks with different numbers of FD-Blocks and 16 FD-Filters for the unstable case (\( \Delta t = 200 \)). As is clear, the larger the number of FD-Blocks, the lower the minimum and eventual training or testing errors at the cost of training a more complex network.

The size of FD-Net models depends on the number of FD-Filters. Thus far, we illustrated the performance of FD-Net with 16 FD-Filters. Fig. 6 shows the results of training the networks with different numbers of FD-Filters and 1 FD-Block on the stable case. The figure clearly shows that the performance of FD-Net with a small number of FD-Filters varies by random seeds, and that utilizing a larger number of FD-Filters reduces this variance. That being said, the results highlight that there is little benefit to using more than 8 FD-Filters, as the average testing error only improves marginally with more FD-Filters. Similar conclusion can be drawn for the other cases; see [23].

**Abbreviation:** Euler, forward Euler method.

Fig. 3: Sequence of predictions for the stable case at \( t \in \{ 200, 400, 600, 800, 1000 \} \) and \( x \in \{ 0, \Delta x, 2\Delta x, \ldots, \pi \} \) for one specific IC. Top Left corner: \( u_{s}(x, 0) \); Rest of top row: Targets (\( u_{s}(x, t) \)) and predictions (\( \tilde{u}_{s}(x, t) \)); Bottom row: Squared errors (\( (u_{s}(x, t) - \tilde{u}_{s}(x, t))^{2} \)).

Fig. 4: Testing errors for different cases, algorithms and random seeds. Top row: minimum testing errors; bottom row: final testing errors. Top/middle/bottom line of boxplot is the upper quartile/median/lower quartile, respectively, the whiskers represent the range, and the circled dots are individual observations.

1000-step predictions with the minimum testing errors (10) over the course of training, and compared them with the predictions made by the forward Euler method. We show results for a single IC. These figures clearly indicate that: (1) FD-Net, when trained sufficiently well, is able to make higher quality predictions across the time horizon than the forward Euler method; (2) training our networks with the TR method allows for better predictions than the ADAM optimizer; (3) the performance of the forward Euler method is highly dependent on the case. Specifically, the Euler method, as predicted by the theory, cannot adequately capture the dynamics of the PDE in the unstable setting.

To further illustrate the testing (prediction) performance of FD-Net, in Fig. 4, we show the minimum testing errors over the training process and the final testing errors, respectively, for every case, algorithm and random seed. Clearly, training FD-Net using the TR method results in higher accuracy predictions with lower variance for all cases. Indeed, this is true for all configurations of FD-Net; see [23]. It is worth noting that 1000-step prediction is a challenging task. This can be attributed to the fact that the error at each time step propagates throughout the time horizon, and any imperfect intermediate predictions can severely deteriorate the final...
In this paper, we presented a novel neural network framework, FD-Net, for learning the dynamics of PDEs and making predictions solely based on trajectory data. The architecture of FD-Net is inspired by finite differences and residual neural networks. For our case study, we showed that high-precision predictions can be made by our networks with only a small number of parameters. Compared with the forward Euler method, our networks are able to achieve better predicting performance for the stable, unstable, noisy and forcing cases. While first-order methods are ubiquitous in training deep neural networks, we proposed to leverage a Hessian-free optimization method, Trust Region (TR) Newton Conjugate Gradient (CG), to train the networks. We demonstrated that the TR method is, computationally, more efficient than the ADAM algorithms in training the networks. Trained by the TR method, FD-Net is able to efficiently learn the dynamics and make fast predictions for the heat equations. However, this was only a proof-of-concept study of our FD-Net model. As future work, we aim to study the applicability of FD-Net for solving different PDEs (e.g., higher-order, nonlinear, etc.) and compare against higher-order and implicit numerical schemes, to extend the territory to discovering hidden PDEs, and to develop customized optimization algorithms for training the networks.

ACKNOWLEDGMENT

This work was partially supported by the U.S. National Science Foundation, under award number CCF:1618717 and CCF:1740796.

REFERENCES

[1] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” Science, vol. 324, no. 5923, pp. 81–85, 2009. [Online]. Available: https://science.sciencemag.org/content/324/5923/81

[2] J. Bongard and H. Lipson, “Automated reverse engineering of nonlinear dynamical systems,” Proceedings of the National Academy of Sciences, vol. 104, no. 24, pp. 9943–9948, 2007. [Online]. Available: https://www.pnas.org/content/104/24/9943

[3] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, “Data-driven discovery of partial differential equations,” Science Advances, vol. 3, no. 4, p. e1602614, 2017.

[4] H. Schaeffer, “Learning partial differential equations via data discovery and sparse optimization,” Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 473, no. 2197, p. 20160646, 2017.

[5] M. Raissi and G. E. Karniadakis, “Hidden physics models: Machine learning of nonlinear partial differential equations,” Journal of Computational Physics, vol. 357, pp. 125–141, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0021999117309014

[6] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations,” arXiv preprint arXiv:1711.10566, 2017.

[7] J. Bongard, and H. Lipson, “Automated reverse engineering of nonlinear dynamical systems,” Science, vol. 324, no. 5923, pp. 81–85, 2009.

[8] M. Schmidt and H. Lipson, “Distilling free-form natural laws from experimental data,” Science, vol. 324, no. 5923, pp. 81–85, 2009. [Online]. Available: https://science.sciencemag.org/content/324/5923/81

[9] Z. Long, Y. Lu, X. Ma, and B. Dong, “PDE-net: Learning PDEs from data,” in Proceedings of the 35th International Conference on Machine Learning, vol. 80, Stockholm, Sweden: PMLR, 10–15 Jul 2018, pp. 3208–3216. [Online]. Available: http://proceedings.mlr.press/v80/long18a.html

[10] L. Bottou, F. E. Curtis, and J. Nocedal, “Optimization methods for large-scale machine learning,” Siam Review, vol. 60, no. 2, pp. 223–311, 2018.

[11] A. S. Berahas and M. Takić, “A robust multi-batch l-bfgs method for machine learning,” Optimization Methods and Software, vol. 35, no. 1, pp. 191–219, 2020. [Online]. Available: https://doi.org/10.1080/10556788.2019.1658108

[12] A. S. Berahas and M. Takić, “Finite difference neural networks: Fast prediction of partial differential equations,” in Proceedings of the National Academy of Sciences, vol. 117, no. 25, pp. 13906–13908, 2020.

[13] P. Xu, F. Roosta, and M. W. Mahoney, “Second-order optimization for non-convex machine learning: An empirical study,” in Proceedings of the 2020 SIAM International Conference on Data Mining. SIAM, 2020, pp. 199–207.

[14] A. S. Berahas, M. Jahani, and M. Takić, “Quasi-newton methods for deep learning: Forget the past, just sample,” arXiv preprint arXiv:1901.09997, 2019.

[15] A. S. Berahas, R. Bollapragada, and J. Nocedal, “An investigation of newton-sketch and subsampled newton methods,” Optimization Methods and Software, pp. 1–20, 2020.

[16] J. Nocedal and S. J. Wright, Numerical Optimization, 2nd ed. Springer-Verlag New York, 2006.

[17] T. Steihaug, “The conjugate gradient method and trust regions in large scale optimization,” SIAM Journal on Numerical Analysis, vol. 20, no. 3, pp. 626–637, 1983.

[18] F. P. Incropera, A. S. Lavine, T. L. Bergman, and D. P. DeWitt, Fundamentals of heat and mass transfer. Wiley, 2007.

[19] L. Euler, Institutionum calculi integralis. impensis Academiae imperialis scientiarum, 1824, vol. 1.

[20] L. Olsen-Kettle, “Numerical solution of partial differential equations,” Lecture notes at University of Queensland, Australia, 2011.

[21] Z. Shi, N. S. Gulgec, A. S. Berahas, S. N. Pakzad, and M. Takić, “Finite Difference Neural Networks: Fast Prediction of Partial Differential Equations,” arXiv preprint arXiv:2006.01892, 2020.

[22] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.