RECOVERING GOVERNING EQUATIONS OF PARTIAL DIFFERENTIAL EQUATIONS VIA FRIEDRICHS LEARNING

Jonathn Chang
Department of Mathematics; University of California, Los Angeles
Los Angeles, CA 90095
jchang153@ucla.edu

Chunmei Wang
Department of Mathematics; University of Florida
Gainesville, FL 32611
chunmei.wang@ufl.edu

Haizhao Yang
Department of Mathematics; University of Maryland
College Park, MD 20742
hzyang@umd.edu

July 28, 2022

ABSTRACT

Data-driven recovery of ODE/PDEs has been an active field in recent years. However, most existing learning algorithms focus on the strong form of equations which involve derivatives of observations that may be noisy, discontinuous, or collected non-uniformly. As such, algorithms that typically use finite differences or other pointwise approximations for derivatives are not robust to noise and not able to capture discontinuity well. This preliminary paper proposes Friedrichs Learning for recovering time-dependent equations in the weak form to solve these challenges.

Keywords: Partial Differential Equation; Weak Formulation; Deep Neural Network; Data-Driven Recovery

1 Introduction

Partial differential equations (PDEs) are used to model complex systems in numerous fields such as physics, engineering, biology, finance, etc. Traditionally, the governing equations of famous PDEs are determined through fundamental physical laws. However, many of the governing equations in PDEs describing modern-day systems are still undetermined and are commonly extracted through numerical methods given observed data through data-driven recovery of ODEs and PDEs [1, 7, 8, 4, 10]. Although data has become increasingly easier to gather and store with abundance in the past few years, the challenge is that these measurements may be noisy, discontinuous, or non-uniform in time or space. Unfortunately, this is common with data observed in the real-world, such as sensor stations that may not be uniformly spaced, experimental design flaws, and in general, environmental randomness.

Learning algorithms that attempt to solve these PDEs in the strong form encounter inaccuracies due to the inability to capture the derivative of discontinuous data, the highly ill-conditioned evaluation of derivatives of noisy data, and instability errors in approximating derivatives of non-uniform data. Some models such as [6] that avoid the spatial derivatives still require the derivative in time. Such data for which derivatives may not exist at all satisfies the PDE only in a weak sense, so the weak formulation of the PDE is proposed to combat these challenges, which avoids the unstable evaluation of derivatives by transferring derivatives of the data to derivatives of a smooth, test function. To the best of our knowledge, although the weak form has been used to learn ODEs [9], it has not been used to learn PDEs.

The weak formulation is converted into a minimax optimization problem which can be trained using a deep neural network (DNN). This is directly drawn from [2], which proposes Friedrichs Learning as a deep learning method to find weak solutions to PDEs using a minimax formulation, hence the title of this paper. The idea is that updating the test function to maximize the loss will complement and enhance the learning of the unknown equation(s) which attempts to minimize the loss.

The goal of this preliminary paper is to introduce the proposed minimax algorithm and implementation, demonstrate its performance through examples, compare its results with latest methods, and discuss future research paths. Section 2 uses the weak formulation of a PDE to derive the Friedrichs Learning algorithm. Section 3 reveals the implementation of the algorithm using DNNs. Section 4 conducts several experiments to compare the performance of Friedrichs Learning against the strong form. Section 5 discusses the results and further improvements that can be made to the algorithm. And finally, Section 6 makes acknowledgments about the support received in conducting this research.

2 Friedrichs' Minimax Formulation with Reaction Diffusion PDE

We'll introduce Friedrichs Learning for recovery with an example of the time-dependent reaction diffusion PDE in one dimension. Higher-order PDEs can be reduced to lower-order ones through auxiliary functions, so they can be formulated similarly.

$$\frac{\partial u}{\partial t} = \frac{d}{dx}\left(H\frac{\partial u}{\partial x}\right) + Gu \tag{2.1}$$

The goal is to recover H and G as functions of x given data for u, where u is defined on a bounded domain $\Omega \times T$ with boundary $\partial\Omega$ and ∂T. In the single-dimensional case, $\Omega \subset \mathbb{R}$. The initial deep learning approach is to reformulate this equation in the strong form into a mean squared error (MSE) loss function with the population risk,

$$(\bar{\theta}_H, \bar{\theta}_G) = \arg\min_{\theta_H,\theta_G} \mathscr{L}(\theta) = \arg\min_{\theta_H,\theta_G} \mathbb{E}_{(x,t)\in\Omega\times T}\left[\left|\frac{\partial u}{\partial t} - \frac{d}{dx}\left(H(x;\theta_H)\frac{\partial u}{\partial x}\right) - G(x;\theta_G)u\right|^2\right], \tag{2.2}$$

where θ_H and θ_G are parameterized as DNNs and $(\bar{\theta}_H, \bar{\theta}_G)$ is the set of parameters (θ_H, θ_G) that minimizes the population risk.

On the other hand, Friedrichs Learning uses the weak form of equation (2.1) which involves a smooth test function $v \in \mathcal{H}_0^1(\Omega) \times \mathcal{H}_0^1(T)$, where $\mathcal{H}_0^1(\Omega)$ and $\mathcal{H}_0^1(T)$ are sets of functions in the standard Sobolev spaces $\mathcal{H}^1(\Omega)$ and $\mathcal{H}^1(T)$ which vanish at the boundaries. We denote the inner product on Ω and T as

$$(u, v)_{\Omega,T} = \int_T \int_\Omega uv \, dxdt \tag{2.3}$$

To convert (2.1) into its weak form, we take the inner product of both sides with respect to the test function v; that is, multiply the equation by v and integrate over the spacetime domain,

$$\int_T \int_\Omega \frac{\partial u}{\partial t} v \, dxdt = \int_T \int_\Omega \left[\frac{d}{dx}\left(H\frac{\partial u}{\partial x}\right) v + Guv\right] dxdt. \tag{2.4}$$

Then, we integrate by parts the terms which involve derivatives in u to reach the weak form,

$$\int_T \left[\left[uH \frac{\partial v}{\partial x} \right]_{\partial\Omega} - \int_\Omega u \frac{\partial v}{\partial t}\, dx \right] dt = \int_T \int_\Omega \left[u \frac{d}{dx} \left(H \frac{\partial v}{\partial x} \right) + Guv \right] dxdt. \tag{2.5}$$

We use this to formulate the supervised learning problem into a minimax optimization problem with v, H, and G parameterized as DNNs $v(x,t;\theta_v), H(x;\theta_H), G(x;\theta_G)$ respectively. Let $v^{\theta_v} = v(x,t;\theta_v)$, $H^{\theta_H} = H(x;\theta_H)$, and $G^{\theta_G} = G(x;\theta_G)$. Then, we have

$$(\bar{\theta}_H, \bar{\theta}_G, \bar{\theta}_v) = \arg \min_{\theta_H,\theta_G} \max_{\theta_v} \mathscr{L}(\theta) =$$

$$= \arg \min_{\theta_H,\theta_G} \max_{\theta_v} \frac{1}{||v^{\theta_v}||_{L^2}} \left| \int_T \left[\left[uH^{\theta_H} \frac{\partial v^{\theta_v}}{\partial x} \right]_{\partial\Omega} - \int_\Omega \left[u \frac{\partial v^{\theta_v}}{\partial t} + u \frac{d}{dx} \left(H^{\theta_H} \frac{\partial v^{\theta_v}}{\partial x} \right) + G^{\theta_G} u v^{\theta_v} \right] dx \right] dt \right|, \tag{2.6}$$

where $(\bar{\theta}_H, \bar{\theta}_G, \bar{\theta}_v)$ is the set of parameters that solves the minimax problem; that is, $(\bar{\theta}_H, \bar{\theta}_G)$ minimize the value $\max_{\theta_v} \mathscr{L}(\theta)$. The normalization term $||v^{\theta_v}||_{L^2}$ guarantees that the solution of the minimax problem is well-posed.

3 Deep Learning Solver

Now we will discuss the discretization of the minimax problem (2.6), the structure of the neural networks used to parameterize H, G, and v, and the algorithm used to train these networks. We will first focus on uniformly spaced data, but discretization can be performed with randomly generated data through Monte Carlo methods.

3.1 Discretization

Assume $\Omega = [0, X]$ and $T = [0, T]$, and data for u is given as $\{u(x_k, t_j)\}$, where $x_k = \frac{kX}{K}$ for $k = 0, \ldots, K$ and $t_j = \frac{jX}{J}$ for $j = 0, \ldots, J$. That is, a grid of $(K+1)(J+1)$ many evenly spaced points in the domain $[0, X] \times [0, T]$. Given that K and J are even, Simpson's 1/3 rule will be used to discretize the single and double integrals, including the $||\cdot||_{L^2}$ norm. Denote $h_x := \frac{X}{K}$ and $h_t := \frac{T}{J}$ as the spacings in the spatial and temporal domains, respectively. Define the Simpson weight matrix as the $J \times K$ matrix with entries $(1,4,2,\ldots,2,4,1)^T(1,4,2,\ldots,2,4,1)$. Then, Simpson's rule gives

$$\int_T [f(t)]\, dt \approx \frac{h_t}{3}(1,4,2,\ldots,2,4,1)(f_0, f_1, f_2, \ldots, f_{J-2}, f_{J-1}, f_J)^T \tag{3.1}$$

for the 1-dimensional integral in T, with $f_j := f(\frac{jT}{J})$,

$$\int_T \int_\Omega [f(x,t)]\, dxdt \approx \frac{h_x h_t}{9} \left(\begin{pmatrix} \text{Simpson} \\ \text{weight} \\ \text{matrix} \end{pmatrix}, \begin{pmatrix} f_{0,0} & f_{1,0} & \cdots & f_{K-1,0} & f_{K,0} \\ f_{0,1} & f_{1,1} & \cdots & f_{K-1,1} & f_{K,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{0,J-1} & f_{1,J-1} & \cdots & f_{K-1,J-1} & f_{K,J-1} \\ f_{0,J} & f_{1,J} & \cdots & f_{K-1,J} & f_{K,J} \end{pmatrix} \right)_F \tag{3.2}$$

for the double integral in Ω and T, with $f_{k,j} := f(\frac{kX}{K}, \frac{jT}{J})$ and $(\cdot,\cdot)_F$ denoting the Frobenius inner product, and

$$||f(x,t)||_{L^2} = \left(\int_T \int_\Omega |f(x,t)|^2 \, dxdt\right)^{\frac{1}{2}} \approx \tag{3.3}$$

$$\approx \frac{h_x h_t}{9} \left(\begin{pmatrix} \text{Simpson} \\ \text{weight} \\ \text{matrix} \end{pmatrix}, \begin{pmatrix} f_{0,0} & f_{1,0} & \cdots & f_{K-1,0} & f_{K,0} \\ f_{0,1} & f_{1,1} & \cdots & f_{K-1,1} & f_{K,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{0,J-1} & f_{1,J-1} & \cdots & f_{K-1,J-1} & f_{K,J-1} \\ f_{0,J} & f_{1,J} & \cdots & f_{K-1,J} & f_{K,J} \end{pmatrix}^{\circ 2} \right)_F^{\frac{1}{2}} \tag{3.4}$$

for the discretization of $||\cdot||_{L^2}$, where $A^{\circ n}$ denotes the Hadamard power of n.

3.2 Neural Network Structure

Similarly to the previous Friedrichs Learning paper [2], we employ a ResNet structure for each DNN, $\phi(\boldsymbol{x};\theta)$. This is defined recursively as follows:

$$\begin{aligned} \boldsymbol{h}_0 &= \boldsymbol{V}\boldsymbol{x} \\ \boldsymbol{g}_l &= \sigma(\boldsymbol{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l) \\ \boldsymbol{h}_l &= \boldsymbol{g}_l + \boldsymbol{h}_{l-1} \\ \phi(\boldsymbol{x};\theta) &:= \boldsymbol{a}^T \boldsymbol{h}_L \end{aligned} \tag{3.5}$$

where $l = 1,\ldots,L$, $\boldsymbol{V} \in \mathbb{R}^{m\times d}$ (input layer), $\boldsymbol{W}_l \in \mathbb{R}^{m\times m}$ (lth weight matrix), $\boldsymbol{b}_l \in \mathbb{R}^m$ (lth bias vector), and $\boldsymbol{a} \in \mathbb{R}^{m\times 1}$ (output layer). The activation function σ is flexible, but usually defined as either Tanh or ReLU^k $:= \max\{0,x\}^k$, where k depends on the differentiability requirement of the function being parameterized. The variable d is the input dimension of the function being parameterized, and m and L are the width and hidden layer depth of the DNN, respectively, and are hyperparameters that can be adjusted during training. This ResNet structure is essentially building each hidden layer ($\boldsymbol{h}_l$) by performing the usual weight, bias, and activation on the previous layer ($\boldsymbol{g}_l$), added to the previous layer without any modifications ($\boldsymbol{h}_{l-1}$).

To address the boundary requirement of the DNN for $v(x,t;\theta_v) \in \mathcal{H}_0^1(\Omega)\times\mathcal{H}_0^1(T)$, we can manipulate the output of the DNN so that it satisfies the conditions automatically. Suppose $\hat{\phi}(x,t;\theta)$ is the output described in (3.5). Then, we can define

$$v(x,t;\theta_v) = a(x,t)\hat{\phi}(x,t;\theta) \tag{3.6}$$

where $a(x,t)$ is a function specifically constructed such that $a(x,t) = 0$ on $\partial\Omega$ and ∂T.

3.3 Network Training

The networks are trained to solve the minimax problem in (2.6) using gradient descent (in particular, the novel Adam optimizer [5] is used). The training algorithm is illustrated in Algorithm 1, where there are n outer iterations and n_t and n_s inner iterations for updating θ_v and (θ_H,θ_G), respectively. In each inner iteration n_t, we hold the parameters (θ_H,θ_G) constant while updating the parameters θ_v according to $\theta_v \leftarrow \theta_v + \eta_v g_t$, where η_v is the step size associated with v at the specified outer iteration and g_t is the gradient of the loss function $\mathscr{L}(\theta)$ given in (2.6), calculated with respect to θ_v. This gradient is evaluated via autograd in PyTorch. Similarly, in each inner iteration n_s, we hold the parameters θ_v constant while updating the parameters θ_H according to $\theta_H \leftarrow \theta_H - \eta_H g_s$ and θ_G according to $\theta_G \leftarrow \theta_G - \eta_G g_s$, where g_s is calculated with respect to (θ_H,θ_G). The learning rates for each network decrease after every outer iteration according to the following exponential decay scheme,

$$\eta_v^{(k)} = \eta_v^{(0)} \left(\frac{1}{10}\right)^{k/\upsilon_v}, \tag{3.7}$$

where υ_v is the decay rate for θ_v, and a hyperparameter in the training. This follows identically for η_H and η_G.

The table of hyperparameters required for training are listed in Table 1. Although the values of these hyperparameters are problem-dependent, the depth of each network is usually fixed at 3 hidden layers with a total depth of 5 layers, and the activation functions are commonly chosen as Tanh for both the testing and solution networks.

Algorithm 1 Friedrichs Learning for Recovering the Reaction Diffusion PDE in One Dimension

Require: The PDE with unknown parameters H and G along with measured data for the solution over some spacetime domain

Ensure: The parameters θ_H, θ_G, and θ_v that solve the minimax problem in (2.6).

Set hyperparameters n (number of outer iterations), n_t (number of inner iterations for v), and n_s (number of inner iterations for H and G).

Set step sizes $\eta_v^{(k)}$, $\eta_H^{(k)}$, and $\eta_G^{(k)}$ for v, H, and G, respectively, in the kth outer iteration.

Initialize $\phi_v(x,t;\phi_v^{0,0})$, $\phi_H(x;\phi_H^{0,0})$, and $\phi_H(x;\phi_G^{0,0})$.

for k in $1,\dots,n$ **do**

  for j in $1,\dots,n_t$ **do**

    Compute the gradient of the loss function in (2.6) with the parameters $(\theta_v^{k-1,j-1},\theta_H^{k-1,0},\theta_G^{k-1,0})$ with respect to θ_v and denote it as g_t.

    Update $\phi_v^{k-1,j} \leftarrow \phi_v^{k-1,j-1} + \eta_v^{(k)} g_t$ with the step size $\eta_v^{(k)}$.

  end for

  $\theta_v^{k,0} \leftarrow \theta_v^{k-1,n_t}$

  for j in $1,\dots,n_s$ **do**

    Compute the gradient of the loss function in (2.6) with the parameters $(\theta_v^{k,0},\theta_H^{k-1,j-1},\theta_G^{k-1,j-1})$ with respect to (θ_H,θ_G) and denote it as g_s.

    Update $\phi_H^{k-1,j} \leftarrow \phi_H^{k-1,j-1} + \eta_H^{(k)} g_s$ with the step size $\eta_H^{(k)}$.

    Update $\phi_G^{k-1,j} \leftarrow \phi_G^{k-1,j-1} + \eta_G^{(k)} g_s$ with the step size $\eta_G^{(k)}$

  end for

  $\theta_H^{k,0} \leftarrow \theta_H^{k-1,n_s}$

  $\theta_G^{k,0} \leftarrow \theta_G^{k-1,n_s}$

end for

Parameter	Definition	Parameter	Definition
n	number of outer iterations	L	depth of each ResNet network
n_t	number of inner iterations for v	m_t	width of each layer in v
n_s	number of inner iterations for H and G	m_s	width of each layer in H and G
$\eta_v^{(0)}$	initial learning rate for v	υ_v	decay rate for η_v
$\eta_H^{(0)}$	initial learning rate for H	υ_H	decay rate for η_H
$\eta_G^{(0)}$	initial learning rate for G	υ_G	decay rate for η_G

Table 1: Hyperparameters for Training

4 Numerical Experiments

Now, we will display some of the numerical experiments done to test the performance of Friedrichs Learning in (2.6) against the performance of the empirical version of the MSE loss function in (2.2), along with some of the issues arising during these tests and the proposal of a different loss function. We will focus on the same reaction diffusion PDE in one

dimension used to introduce the Friedrichs Learning algorithm, where the given data is uniform and non-noisy, but future experiments will be done to motivate the advantage of (2.6) with non-uniform and noisy data.

To measure the accuracy of the recovered equations, the following discrete relative L^2 error will be applied to uniformly spaced points on the spatial domain:

$$e_{L^2}(\theta) := \left(\frac{\sum_i ||\phi(x_i;\theta) - T(x_i)||^2_{L^2}}{\sum_i ||T(x_i)||^2_{L^2}} \right)^{\frac{1}{2}} \tag{4.1}$$

where $\phi(x;\theta)$ denotes either the network approximation for H or G, and T is the corresponding exact solution.

4.1 Friedrichs Learning Tests

We begin with the following example. Choose $\Omega = [0,1]$ and $T = [0,2]$, with the boundary and initial value conditions $u(0,t) = 0$, $u(1,t) = 0$, and $u(x,0) = x(1-x)$, and the following choices of H and G.

$$\begin{aligned} H(x) &= \frac{e^{-\sin(2\pi x)}}{20} \\ G(x) &= \cos(2\pi x) \end{aligned} \tag{4.2}$$

These choices of H and G do not provide an exact solution for u, so we use a FTCS finite difference scheme to generate an approximate grid of solution points, as well as approximating the spatial and temporal derivatives of this grid with central differences in the interior and foward/backward differences at the boundaries for the derivatives in (2.2).

For the choice of $a(x,t)$ in (3.6), we choose $a(x,t) = \frac{x(1-x)t(2-t)}{0.5^2}$, which is 0 on the spatial and temporal boundaries and reaches a maximum of 1 at $(x,t) = (0.5,1)$.

The choices of parameters for Friedrichs Learning are given in Table 2 and for the MSE strong form in Table 3.

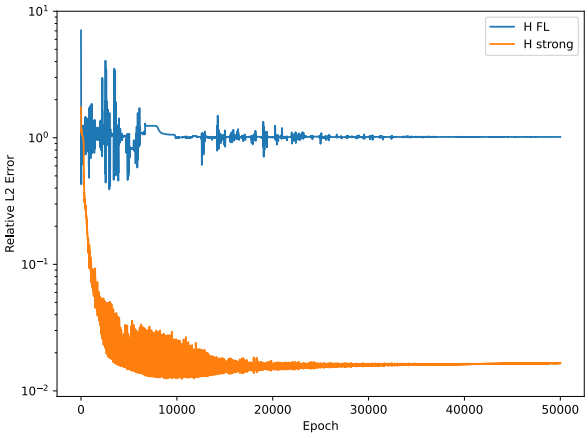
Parameter	Value	Parameter	Value	Parameter	Value
n	50000	$\eta_v^{(0)}$	3×10^{-3}	υ_v	20000
n_t	10	$\eta_H^{(0)}$	3×10^{-4}	υ_H	20000
n_s	1	$\eta_G^{(0)}$	3×10^{-4}	υ_G	20000
L	5	m_t	50	m_s	50

Table 2: Hyperparameters for Friedrichs Learning Tests, FL

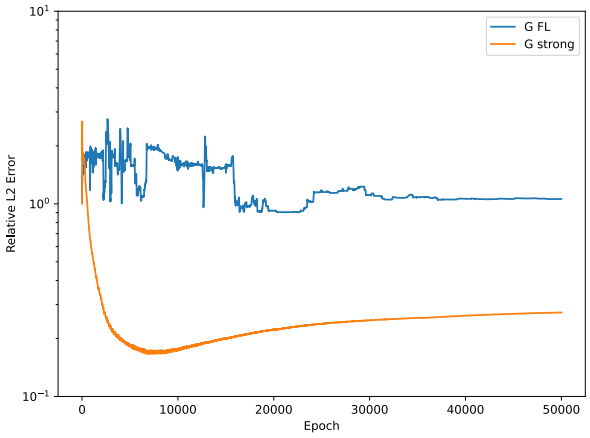
Parameter	Value	Parameter	Value	Parameter	Value
n	50000	$\eta_H^{(0)}$	1×10^{-3}	υ_H	20000
m_s	50	$\eta_G^{(0)}$	1×10^{-3}	υ_G	20000

Table 3: Hyperparameters for Friedrichs Learning Tests, MSE

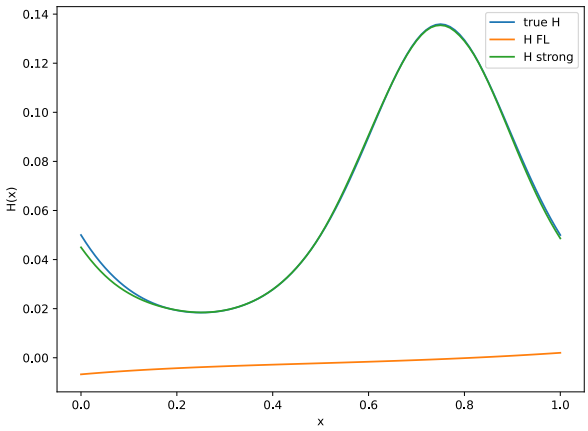
The graph of the relative L^2 error in Friedrichs Learning against the strong form can be seen in Figure 1a for H and Figure 1b for G. Unfortunately, we can see that the error produced by Friedrichs Learning is two orders of magnitude worse in H and one order of magnitude worse in G than the error produced by the strong form. As observed in Figures 1c and 1d, Friedrichs Learning has a difficult time learning the curvature in the true graphs. One disadvantage that can also be noted for the strong form is the inaccuracy near the boundaries in the graphs produced by the strong form, such



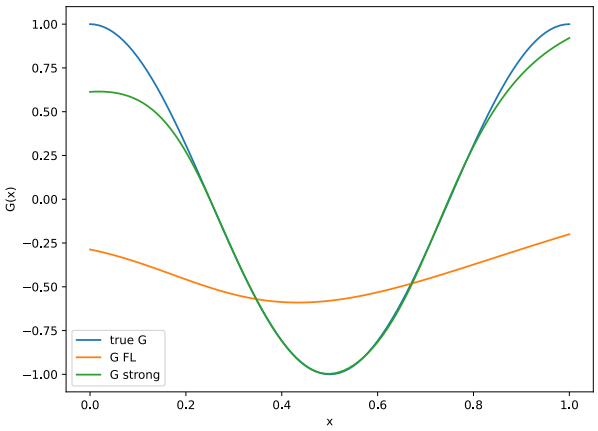
(a) Error History for H



(b) Error History for G



(c) Graph for H



(d) Graph for G

Figure 1: Graphs for (4.2)

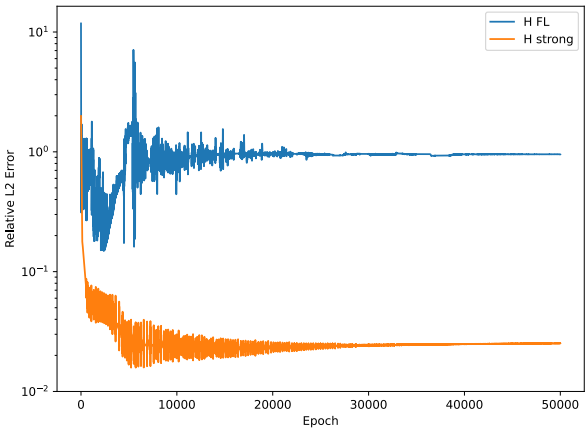
as in Figure 1d for G. This may be attributed to the approximation of derivatives of u in its loss function, which can get highly inaccurate with higher orders.

We'll try another example with simpler H and G to see if FL performs better in recovering them. We'll choose the same domain, boundary, and initial value conditions with the following choices of H and G:

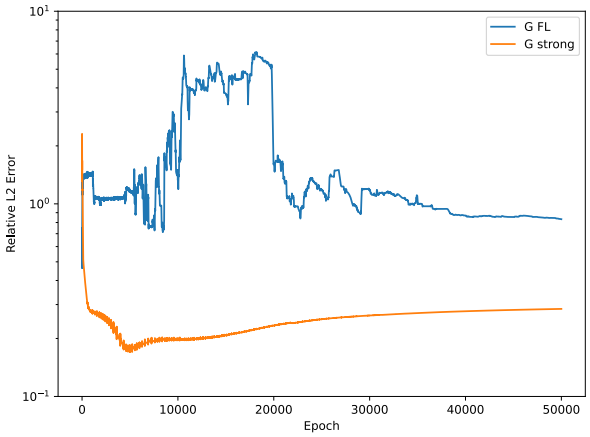
$$\begin{aligned} H(x) &= \frac{x^2}{10} \\ G(x) &= x^2 \end{aligned} \tag{4.3}$$

Again, we use the same finite difference schemes to approximate u and its derivatives, as well as the same $a(x,t)$ and parameters in Table 2 and Table 3 for Friedrichs Learning compared with the strong form, respectively.

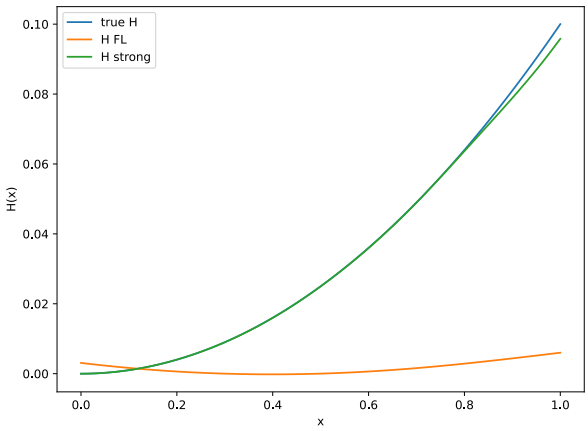
The graph of the relative L^2 error in Friedrichs Learning against the strong form can be seen in Figure 2a for H and Figure 2b for G. Again, Friedrichs Learning is unable to match the error produced by the strong form. We see again in



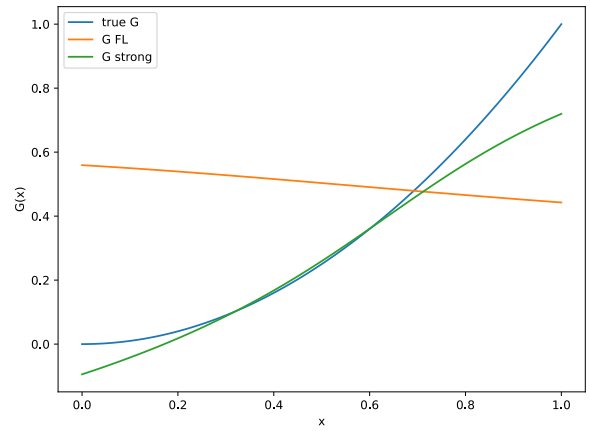
(a) Error History for H



(b) Error History for G



(c) Graph for H



(d) Graph for G

Figure 2: Graphs for (4.3)

Figures 2c and 2d that Friedrichs Learning tends to learn lines and has difficulty adapting to curvature. We also see the same disadvantage in the strong form near the boundaries of the graph in G in Figure 2d.

4.2 Other Tests

With the discouraging results in the previous section, we look to reformulate the loss function in equation (2.6). We construct two new loss functions to test the weak form in Friedrichs Learning. The first uses the similar mean-squared scheme in (2.2), but multiplies the test function v to each term, which turns it into a minimax problem:

$$(\bar{\theta}_H, \bar{\theta}_G, \bar{\theta}_v) = \arg \min_{\theta_H, \theta_G} \max_{\theta_v} \mathscr{L}(\theta) =$$

$$= \arg \min_{\theta_H, \theta_G} \max_{\theta_v} \mathbb{E}_{(x,t)\in\Omega\times T} \left[\left| \left(\frac{\partial u}{\partial t} - \frac{d}{dx} \left(H(x;\theta_H) \frac{\partial u}{\partial x} \right) - G(x;\theta_G) u \right) v(x,t;\theta_v) \right|^2 \right]. \tag{4.4}$$

The second proceeds with the inner product in (2.4), but does not perform any integration by parts and places the absolute value on the inside:

$$(\bar{\theta}_H, \bar{\theta}_G, \bar{\theta}_v) = \arg\min_{\theta_H,\theta_G}\max_{\theta_v}\mathscr{L}(\theta) =$$
$$= \arg\min_{\theta_H,\theta_G}\max_{\theta_v}\frac{1}{||v^{\theta_v}||_{L^2}}\int_T\int_\Omega\left|\left(\frac{\partial u}{\partial t} - \frac{d}{dx}\left(H(x;\theta_H)\frac{\partial u}{\partial x}\right) - G(x;\theta_G)u\right)v(x,t;\theta_v)\right|dxdt. \tag{4.5}$$

The choice of (4.4) is used to identify the influence and benefit of the test function v in the minimax problem, and the choice of (4.5) is used to test the effect of the absolute value bar placement. This formulation is similar to one that has been proven to be successful [3]. Through testing (4.4), we discover that the choice of v produces a large difference in accuracy. In particular, choosing $v \in \mathcal{H}_0^1(\Omega) \times \mathcal{H}_0^1(T)$ produces inaccuracies in recovering H and G. So, we proceed with testing these functions without multiplying the result of v with the constraint function $a(x,t)$ in equation (3.6).

We display the results of an experiment with $\Omega = [0,\pi]$ and $T = [0,2]$, with boundary and initial value conditions $u(0,t) = 0$, $u(1,t) = 0$, and $u(x,0) = x(\pi - x)$, and the following choices of H and G, as well as an exact solution u.

$$H(x) = 2.0$$
$$G(x) = 0.5$$
$$u(x,t) = \sum_{n=1}^{\infty}\left[\frac{-4}{\pi n^3}\sin(nx)\exp(t(-2n^2+0.5))((-1)^n - 1)\right] \tag{4.6}$$

These choices provide an exact solution for which smooth derivatives exist so that the error in approximating the solution and its derivatives will not influence the results experiment. The choices of parameters for both (4.4) and (4.5) are presented in Table 4 and the choices of parameters for the MSE strong form (2.2) are presented in Table 5 .

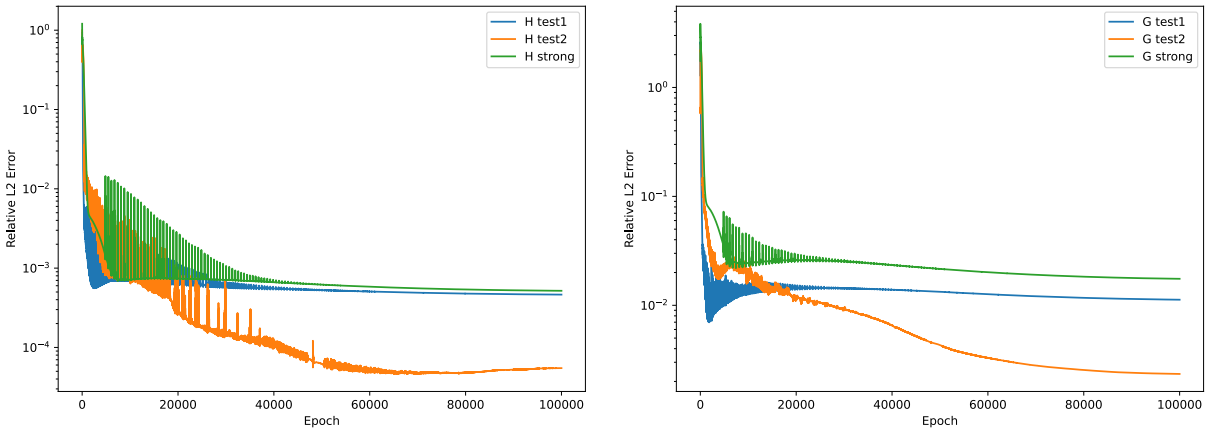
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
n	100000	$\eta_v^{(0)}$	3×10^{-3}	υ_v	20000
n_t	1	$\eta_H^{(0)}$	3×10^{-4}	υ_H	20000
n_s	1	$\eta_G^{(0)}$	3×10^{-4}	υ_G	20000
L	5	m_t	50	m_s	50

Table 4: Hyperparameters for Test (4.6), New Functions

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
n	100000	$\eta_H^{(0)}$	1×10^{-4}	υ_H	20000
m_s	50	$\eta_G^{(0)}$	1×10^{-4}	υ_G	20000

Table 5: Hyperparameters for Test (4.6), MSE

The graph of the relative L^2 error for the two new loss functions (denoted "test1" for (4.4) and "test2" for (4.5)) against the strong form can be seen in Figures 3a and 3b. In both graphs, we see (4.4) slightly outperforming the strong form, and (4.5) outperforming the strong form by around one order of magnitude. In particular, (4.5) is able to reach a minimum error of $e_{L^2} \approx 4.6 \times 10^{-5}$ for H and $e_{L^2} \approx 2.3 \times 10^{-3}$ for G, indicating remarkably high accuracy. We can also see that the graphs of both new loss functions are much more consistent in converging to a minimum compared to the fluctuating behavior of the graphs of FL in Figures 1 and 2.



(a) Error History for H

(b) Error History for G

Figure 3: Graphs for (4.6)

5 Conclusion and Further Research

Despite the discouraging results in Section 4.1, we were able to reach far better results in Section 4.2, indicating merit in utilizing a test function v to reformulate the recovery problem into a minimax optimization problem.

From here, research will be conducted to further explore the new functions proposed in Section 4.2, particularly the use of a test function that does not vanish on the boundary, as well as the placement of the absolute value signs in the loss function. With $v \notin \mathcal{H}_0^1\Omega \times \mathcal{H}_0^1(T)$, the weak form must be reformulated, as certain terms will no longer be able to be cancelled out during integration by parts. This new weak form will be tested, and if results are favorable, we will proceed to test with noisy and non-uniform solution data.

In particular, we plan to compare this method with Weak SINDy in [9], which has proven to achieve high accuracy with the weak form against noisy, low dimensional, uniform data. The use of DNNs in Friedrichs Learning gives the opportunity to improve on Weak SINDy with the ability to recover equations from higher-dimensional PDEs.

6 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. DMS-2050133. 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 National Science Foundation.

Jonathn Chang is extremely grateful to receive this opportunity to conduct research under the mentorship of Dr. Wang and Dr. Yang. He also acknowledges the support from Texas Tech University and the NSF in hosting this program and making this opportunity possible. Finally, he thanks all the mentors and students in the Texas Tech REU for an amazing experience this summer.

References

[1] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, Mar 2016.

[2] Fan Chen, Jianguo Huang, Chunmei Wang, and Haizhao Yang. Friedrichs learning: Weak solutions of partial differential equations via deep learning, 2020.

[3] Yiqi Gu, Haizhao Yang, and Chao Zhou. SelectNet: Self-paced learning for high-dimensional partial differential equations. *Journal of Computational Physics*, 441:110444, Sep 2021.

[4] Jiequn Han, Chao Ma, Zheng Ma, and Weinan E. Uniformly accurate machine learning-based hydrodynamic models for kinetic equations. *Proceedings of the National Academy of Sciences*, 116(44):21983–21991, Oct 2019.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[6] Quanjun Lang and Fei Lu. Learning interaction kernels in mean-field equations of 1st-order systems of interacting particles, 2020.

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

[8] Chao Ma, Jianchun Wang, and Weinan E. Model reduction with memory and the machine learning of dynamical systems, 2018.

[9] Daniel A. Messenger and David M. Bortz. Weak sindy for partial differential equations, Dec 2020.

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