

# The Neural Network Approach to Inverse Problems in Differential Equations<sup>☆</sup>

Kailai Xu<sup>a</sup>, Eric Darve<sup>b</sup>

<sup>a</sup>*Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA, 94305*

<sup>b</sup>*Mechanical Engineering, Stanford University, Stanford, CA, 94305*

## Abstract

We proposed a framework for solving inverse problems in differential equations based on neural networks and automatic differentiation. Neural networks are used to approximate hidden fields. We analyze the source of errors in the framework and derive an error estimate for a model diffusion equation problem. Besides, we propose a way for sensitivity analysis, utilizing the automatic differentiation mechanism embedded in the framework. It frees people from the tedious and error-prone process of deriving the gradients. Numerical examples exhibit consistency with the convergence analysis and error saturation is noteworthy predicted. We also demonstrate the unique benefits neural networks offer at the same time: universal approximation ability, regularizing the solution, bypassing the curse of dimensionality and leveraging efficient computing frameworks.

*Keywords:* Neural Networks, Inverse Problems, Partial Differential Equations

## 1. Introduction

### 1.1. Background

Inverse problems are well motivated and challenging in many science and engineering applications. Given the model differential equations and initial/boundary conditions, we can solve the forward problem by numerical PDE schemes and give predictions of data. The inverse problems, on the other hand, start with the measured data and aim at estimating the parameters in the models. The inverse problems are encountered widely in seismic imaging [1], medical tomography [2], plasma diagnosis [3] and so on.

Recent development in machine learning and neural networks has made the neural network approach for solving inverse problems revive. In this article, we propose a framework of calibration of unknown functions in the model using neural networks. The use of neural networks has many benefits. The primary advantage of the proposed method is that the framework is widely applicable; theoretically, it can learn any continuous functions under

<sup>☆</sup>The first author thanks the Stanford Graduate Fellowship in Science & Engineering and the 2018 Schlumberger Innovation Fellowship for the financial support.

*Email addresses:* kailaix@stanford.edu (Kailai Xu), darve@stanford.edu (Eric Darve)

mild assumptions, and potentially non-continuous functions. Besides, the neural networks also offer implicit regularization and can overcome the curse of dimensionalities. Thanks to the active involvement of deep learning researchers in both academia and industries, many inverse problems can be very easily implemented, and the performance is usually optimized for general purpose.

However, one concern that a researcher may have is whether the method has theoretical guarantee. In this article, we seriously consider the error estimate in the neural network approach. We prove that for the diffusion model problem, the error is bounded by  $\mathcal{O}(\Delta t^2 + h^2)$ , where  $\Delta t$  and  $h$  depends on the collected data, under certain assumptions on the optimization. We also demonstrate it through our numerical experiments, which show the consistency with our analysis. To our best knowledge, this is the first time this kind of analysis and numerical demonstration has been done in literature.

The guiding principle is the same as that suggested in [4]: *avoid discretization until the last possible moment*. The motivation is obvious: the discretization will introduce numerical errors (in this article we will use the term *consistency error* for this type of error in the inverse problem) which are undesired. Instead, we approximate the unknown functions directly using neural networks and evaluate it at the numerical discretization points. We find that the deferred discretization yields a more stable calibration.

Finally, the neural network approach provides a natural way for “sensitivity analysis”. We derive a “sensitive region” for an interesting physical quantity under a specified neural network model. We have intentionally avoided the term “uncertainty quantification” since the sensitivity is optimization dependent; that is, it depends on the optimization procedure. The “sensitive region” describes to what extent the solution is credible if we are most interested in the accuracy of a particular physical quantity.

## 1.2. Related Works

There has been much work in the field of inverse problems in differential equations. Here we mention a few works. One family of approach is called *sparsity regularization* [5]. In 2004, Daubechies *et al* [6] provided a first theoretical treatment on sparsity regularization for ill-posed inverse problems and established the convergence of the iterative soft-thresholding algorithm. Sparsity regularization as a paradigm for solving inverse problems has gained much popularity. For a detailed discussion on sparsity regularization in inverse problems, see [7–10]. Another popular approach is the *Bayesian approach*. This approach introduces regularization by means of the prior information and can handle nonlinearity and non-Gaussianity [11]. Besides, it is possible to formulate uncertainty quantification based on the Bayesian approach [12]. For detailed description, see [1, 4, 12–14]. *Adjoint methods* are also popular approaches, especially in seismic tomography [15]. The adjoints method gives an efficient way to evaluate the gradients of an interesting function  $g$  with respect to parameters in PDEs (also called *design parameters*, a.k.a. *control variables* or *decision parameters*) [16]. The gradients are then fed into an optimization algorithm such as conjugate-gradient methods [17], and thereafter the best design parameters for maximizing (or minimizing)  $g$  are found. For more details, see [18–22].

Using neural networks for inverse problems has been in the literature for a while. Back to 30 years ago, Ibrahim [23] considered using a neural network for solving simple inverse problems such as the Fredholm integral equations. Another paper [24] published in 1998 considered the network inversion method for the Fredholm integral equations of the first kind. Unfortunately, due to the limited computation capacity, the authors only considered one- or two-layer neural networks, which have limited representation ability and therefore do not suffice for more complicated tasks. Recently, there has been great progress in using neural networks for engineering problems [25–32]. Our work distinguishes from other works in that

- We abide by the “*avoid discretization until the last possible moment*” principle. For example, for approximating an unknown multivariate scalar function, the neural network exactly takes in a position vector and outputs a number.
- Besides, the algorithm is designed to incorporate the physics, utilizing the existing numerical schemes instead of creating special schemes. This also enables “*hot-plugging*”, where users are still able to use sophisticated numerical schemes.
- Another noteworthy feature is that algorithmically we can pretend we know the hidden fields and solve the forward problem. The objective function obtained is by comparing the obtained solution and observations. We can treat the algorithm as a blackbox: once fed with observations, we get the approximated unknown function immediately. This feature is called *end-to-end* [33] in the machine learning community.
- Finally, under this framework, we can give convergence bound and classify the sources of error in terms of optimization, discretization, and observation. Pioneer works in [34] inspire the latter view.

We summarize the contribution of the article here

- Propose a neural network framework for solving inverse problems.
- Conduct error analysis for the framework. Particularly, we proved the convergence rate of the approach as data is collected in finer granularity.
- Propose a way for sensitive analysis under the framework.
- Solve several calibrating problems for linear and nonlinear differential equations. We also give general comments on the numerical results.

## 2. Calibrating Unknown Functions using Neural Networks

In this paper, we consider the evolution problems. Consider a physical system that is governed by a set of partial differential equations

$$u_t = \mathcal{L}(\mathbf{x}, u; f) \tag{1}$$

here  $f$  is a unknown function which maps  $\mathbf{x}$  to a scalar or a vector. We have a set of observations (possibly noisy) for  $u(\mathbf{x}, t)$ . In this paper, we consider sequential snapshots of the function values, which can be viewed as samples from the set of distributions

$$\mathcal{M} = \{(u(\mathbf{x}, t) + \mathbf{w}_0, u(\mathbf{x}, t + \Delta t) + \mathbf{w}_1, \dots, u(\mathbf{x}, t + (m-1)\Delta t) + \mathbf{w}_{m-1}) | t \in \mathcal{T}\} \quad (2)$$

where  $\mathbf{w}_i$ 's are random noise (not necessarily i.i.d.) and  $\mathcal{T}$  is a finite set of positive values (sampling times).  $\mathcal{M}$  can be seen as that we have taken a continuous sequence of snapshots at each time  $t \in \mathcal{T}$ . For later discussion, we consider a sample  $M$  subject to  $\mathcal{M}$

$$M = \{(U_0, U_1, \dots, U_{m-1}) | t \in \mathcal{T}\} \quad (3)$$

$$= \{(u(\mathbf{x}, t) + \mathbf{W}_0, u(\mathbf{x}, t + \Delta t) + \mathbf{W}_1, \dots, u(\mathbf{x}, t + (m-1)\Delta t) + \mathbf{W}_{m-1}) | \quad (4)$$

$$t \in \mathcal{T}, \mathbf{x} = \{x_1, x_2, \dots, x_n\}\} \quad (5)$$

Our task is to find an appropriate  $f_\theta$  such that the data are consistent with the model. Here  $f_\theta$  is parametrized by  $\theta$ . The basic idea is first to propose a function  $f_\theta$ , and then do the forward simulation and see if the predicted result is consistent with the data. If not, we propose another  $f_{\theta'}$ . The transition of  $f_\theta$  to  $f_{\theta'}$  is carried out by an first-order optimizer that uses automatic differentiation. The utilization of the gradient information will boost the performance tremendously compared to other zeroth order methods, which are intrinsically trial-and-error methods.

Another key component is that we combine the robust and highly accurate numerical methods in partial differential equations with the neural networks. Assume that the numerical scheme for eq. (1) is

$$L(v^{N+m-1}, v^{N+m-2}, \dots, v^N; f_\theta) = 0 \quad (6)$$

where  $v^N(\mathbf{x}) = v(\mathbf{x}, t_N)$  is the numerical solution for  $u(\mathbf{x}, t_N)$  at time  $t = t_N$ . The scheme can be explicit, implicit or semi-implicit; it can also be linear or nonlinear.

The loss function to minimize is given by

$$\text{loss} = \sum_{t_N \in \mathcal{T}} L(U^{N+m-1}, U^{N+m-2}, \dots, U^N; f_\theta)^2 \quad (7)$$

Note even if we minimize the loss function to zero, we are not guaranteed that  $\tilde{f} = f$  since in general

$$L(u^{N+m-1}, u^{N+m-2}, \dots, u^N; f) \neq 0 \quad (8)$$

due to discretization error. However, if the error is small enough, the optimization drives eq. (7) to near zero, the noise is small enough, and the neural network approximation ability is sufficient, we expect  $f_\theta \approx f$ . In the next section, we will conduct a quantified analysis of those errors.

We have to point out it is imperative to adopt the **deferred discretization** here. That is, we do not first learn  $\{f_\theta(\mathbf{x}_i)\}$  at the discretization points  $\{\mathbf{x}_i\}$  and then try to fit these

values using a neural network. Instead, we first form a neural network and evaluate the neural network at the discretization points. The neural network structure will be

$$\mathbf{x} \in \mathbb{R}^d \rightarrow \text{multilayer dense neural networks} \rightarrow y \in \mathbb{R} \quad (9)$$

Like most other inverse problems, such problems are usually ill-posed, and most of the time the stability is questionable (we use the term *ill-conditioned* here). That is, the learned function  $\tilde{f}$  can be very sensitive to the data we have collected, especially those with noise. Neural networks offer a regularization effect in the deferred discretization setting. Indeed, if we use smooth (nonlinear) activation functions in the “multilayer dense neural networks” part in eq. (9), the function  $f_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$  will be smooth. And if we appropriately regularize the neural network, the gradients of the neural networks can also be bounded (for example, by adopting a projected gradient descent during the training process). That results in a “well-behaved” function. In this setting, if the unknown function is also well-behaved, the neural network will offer a good solution in general. The power of the neural network is actually when the unknown function is ill-behaved. In this case, the neural network will either fit the function well or signal us through the unusual behavior of the optimization or the extreme values in the weights.

### 3. Error Analysis and Sensitivity Analysis

#### 3.1. Approximation of Unknown Functions by Neural Networks

*Neural Network Architecture.* We adopt the standard dense neural networks, which can be expressed as a series of function compositions

$$\begin{aligned} \mathbf{y}_2(\mathbf{x}) &= \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\ \mathbf{y}_3(\mathbf{y}_2) &= \tanh(\mathbf{W}_2\mathbf{y}_2 + \mathbf{b}_2) \\ &\dots \\ \mathbf{y}_{n_l}(\mathbf{y}_{n_l-1}) &= \tanh(\mathbf{W}_{n_l-1}\mathbf{y}_{n_l-1} + \mathbf{b}_{n_l-1}) \\ \mathbf{y}_{n_l+1}(\mathbf{y}_{n_l}) &= \mathbf{W}_{n_l}\mathbf{y}_{n_l} + \mathbf{b}_{n_l} \\ f_\theta(\mathbf{x}) &= \mathbf{y}_{n_l+1}(\mathbf{y}_{n_l}(\dots(\mathbf{y}_2(\mathbf{x})))) \end{aligned} \quad (10)$$

here  $\theta$  ensembles all the weights (including biases) parameters

$$\theta = \{\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_{n_l}, \mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{n_l}\} \quad (11)$$

We have used the activation functions  $\tanh$  in this work, but other choices are also possible, such as ReLU, sigmoid, leaky ReLU and so on. Different neural network architectures are also possible, such as those that contain the convolutional layers [35], sparse convolutional neural networks [36], pooling layers, residual connections [37], recurrent cells [38] and so on.

*Uniformly bounded weights.* Regularization techniques are extensively studied to combat overfitting in neural network researches. It was shown that the magnitude of the weights directly impacts the generalization ability of the neural networks. Intuitively, functions are simpler when they vary at a slower rate and thus generalize better [39]. This idea is explored in various articles and shown to be effective to achieve the state-of-the-art performance. Therefore, following the line of research in machine learning, we constrain our neural networks to be equipped with uniformly bounded weights, i.e., there exists a positive constant  $C > 0$ , such that for all training scenarios

$$\|\mathbf{W}_i\|_2 \leq C \quad i = 1, 2, 3, \dots, n_l \quad (12)$$

Here we have chosen  $p = 2$  norm for  $\mathbf{W}_i$ , but other choices such as  $p = 1, \infty$  are possible. In addition, we do not need to make any assumptions on the bias terms  $\mathbf{b}_i$ . For technical reasons, we make the following assumption, which imposes a bound on the last bias term  $\mathbf{b}_{n_l}$

**Assumption 1** (Uniform Bound on Parameters). *There exists a positive constant  $C > 0$  such that*

$$\|\mathbf{W}_i\|_2 \leq C \quad i = 1, 2, 3, \dots, n_l \quad (13)$$

$$\|\mathbf{b}_{n_l}\|_2 \leq C \quad (14)$$

The constraints can be easily enforced through a projection step in the optimization phase and perform a variant of the projected (stochastic) gradient descent method. To be specific, we define a projection function

$$\pi(\mathbf{W}, C) = \frac{1}{\max\{1, \frac{\|\mathbf{W}\|_2}{C}\}} \mathbf{W} \quad (15)$$

which will project the weights back to the closest matrix with bounded 2-norm  $C$ . Under this assumption, we are able to derive an upper bound on the first and second order (and theoretically all orders, albeit the upper bound will go to infinity as the order increases) derivatives of the neural network approximation function.

**Theorem 1.** *Assume eq. (12) holds. Then the neural network given by eq. (10) satisfies*

$$\left\| \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right\|_2 \leq C^{n_l} \quad (16)$$

$$\left\| \frac{\partial^2 f_\theta(\mathbf{x})}{\partial \mathbf{x}^2} \right\|_2 \leq 2C^{n_l+1} \frac{1 - C^{n_l-1}}{1 - C} \quad (17)$$

*Proof.* We use the numerator layout for the matrix calculus. Using the fact that  $\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh(x)^2$  we have according to the chain rule

$$\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}_{n_l+1}}{\partial \mathbf{y}_{n_l}} \frac{\partial \mathbf{y}_{n_l}}{\partial \mathbf{y}_{n_l-1}} \cdots \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}} \quad (18)$$

$$= \mathbf{W}_{n_l} \mathbf{W}_{n_l-1} \cdots \mathbf{W}_1 (1 - \mathbf{y}_{n_l} \otimes \mathbf{y}_{n_l}) (1 - \mathbf{y}_{n_l-1} \otimes \mathbf{y}_{n_l-1}) \cdots (1 - \mathbf{y}_2 \otimes \mathbf{y}_2) \quad (19)$$

here  $\otimes$  denotes element-wise multiplication. Note that  $\tanh(x) \in (-1, 1)$ , we immediately obtain that

$$\left\| \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right\|_2 \leq \|\mathbf{W}_{n_l}\|_2 \|\mathbf{W}_{n_{l-1}}\|_2 \dots \|\mathbf{W}_1\|_2 \leq C^{n_l} \quad (20)$$

Likewise, we have

$$\begin{aligned} \frac{\partial^2 f_\theta(\mathbf{x})}{\partial \mathbf{x}^2} &= \frac{\partial}{\partial \mathbf{x}} \left( \frac{\partial \mathbf{y}_{n_l+1}}{\partial \mathbf{y}_{n_l}} \frac{\partial \mathbf{y}_{n_l}}{\partial \mathbf{y}_{n_{l-1}}} \dots \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}} \right) \\ &= \sum_{i=2}^{n_l} \mathbf{W}_{n_l} \mathbf{W}_{n_{l-1}} \dots \mathbf{W}_1 \left( -2 \mathbf{y}_i \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}} \right) \prod_{j \neq i, j=2}^{n_l} (1 - \mathbf{y}_j \otimes \mathbf{y}_j) \end{aligned} \quad (21)$$

Since we have

$$\frac{\partial \mathbf{y}_i}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}_i}{\partial \mathbf{y}_{i-1}} \frac{\partial \mathbf{y}_{i-1}}{\partial \mathbf{y}_{i-2}} \dots \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}} = \mathbf{W}_{i-1} \mathbf{W}_{i-2} \dots \mathbf{W}_1 \prod_{j=2}^i (1 - \mathbf{y}_j \otimes \mathbf{y}_j)$$

and therefore

$$\left\| \frac{\partial \mathbf{y}_i}{\partial \mathbf{x}} \right\|_2 \leq C^{i-1}$$

Insert the equation into eq. (21) we have

$$\left\| \frac{\partial^2 f_\theta(\mathbf{x})}{\partial \mathbf{x}^2} \right\|_2 \leq 2C^{n_l} \sum_{i=2}^{n_l} C^{i-1} = 2C^{n_l+1} \frac{C^{n_l-1} - 1}{C - 1} \quad (22)$$

□

### 3.2. Approximation Theory of Neural Networks

The approximation degree for one layer neural network is well understood in literature. One such result is [40]

**Theorem 2.** Let  $1 \leq d' \leq d$ ,  $r \geq 1$ ,  $d \geq 1$  be integers,  $1 \leq p \leq \infty$ ,  $\varphi : \mathbb{R}^{d'} \rightarrow \mathbb{R}$  be infinitely many times continuously differentiable in some open sphere in  $\mathbb{R}^{d'}$ . We further assume that there exists  $\mathbf{b}$  in this sphere such that

$$\mathbf{D}^\mathbf{k} \varphi(\mathbf{b}) \neq 0, \quad \mathbf{k} \in \mathbb{Z}^{d'}, \mathbf{k} \geq \mathbf{0} \quad (23)$$

Then there exist  $d' \times d$  matrices  $\{A_j\}_1^n$  with the following property. For any  $f \in W_{r,d}^p$ , there exist coefficients  $a_j(f)$  such that

$$\|f - \sum_{j=1}^n a_j(f) \varphi(A_j(\cdot) + \mathbf{b})\|_p \leq cn^{-r/d} \|f\|_{W_{r,d}^p} \quad (24)$$

Here

$$\|f\|_{W_{r,d}^p} = \sum_{0 \leq |\mathbf{k}| \leq r} \|D^{\mathbf{k}} f\|_p \quad (25)$$

with multi-integer  $\mathbf{k} = (k_1, k_2, \dots, k_s)$

For example, if  $d' = 1$ , and  $\varphi = \sigma$  is the sigmoid function, then  $\varphi$  satisfies the condition in the theorem. If  $f \in W_{1,d}^p$  (and therefore  $r = 1$ ), we obtain a neural network with one hidden layer (hidden size equals  $n$ ). To achieve a predetermined error  $\varepsilon > 0$ , we need

$$n^{-1/d} = \mathcal{O}(\varepsilon) \Rightarrow n = \mathcal{O}(\varepsilon^{-d}) \quad (26)$$

This indicates that one layer neural network suffers from the curse of dimensionalities – the number of required neurons in the hidden layer grows exponentially with the input dimension  $d$ .

One possible way to overcome the difficulty is to consider multilayer neural network. We will now consider this possibility.

In the late 1950s, Kolmogorov proved in a series of papers the Kolmogorov Superposition Theorem that answers (in the negative) Hilbert's 13th problem [41]. It is a theorem about representing instead of approximating and quite deep. It says that for a continuous function defined on  $[0, 1]^d$ , given appropriate activation function, a neural network with hidden sizes  $\mathcal{O}(d)$  will be able to represent the function *exactly*.

**Theorem 3** (Kolmogorov Superposition Theorem [42]). *There exist  $d$  constants  $\lambda_j > 0$ ,  $j = 1, 2, \dots, d$ ,  $\sum_{j=1}^d \lambda_j \leq 1$  and  $2n + 1$  strictly increasing continuous function  $\phi_i$ ,  $i = 1, 2, \dots, 2d + 1$ , which map  $[0, 1]$  to itself, such that every continuous function  $f$  of  $d$  variables on  $[0, 1]^d$  can be represented in the form*

$$f(x_1, \dots, x_d) = \sum_{i=1}^{2d+1} g \left( \sum_{j=1}^d \lambda_j \phi_i(x_j) \right) \quad (27)$$

for some  $g \in C[0, 1]$  depending on  $f$ .

Based on the above result, Maiorov and Pinkus [43] proved that if the fixed number of units in the hidden layers are  $6d + 3$  and  $3d$ , the two-layer neural networks can approximate any function to arbitrary precision.

**Theorem 4.** *There exists an activation function  $\sigma$  which is analytical, strictly increasing and sigmoidal and has the following property: for any  $f \in C[0, 1]^d$  and  $\varepsilon > 0$ , there exist constants  $d_i$ ,  $c_{ij}$ ,  $\theta_{ij}$ ,  $\gamma_i$  and vectors  $\mathbf{w}^{ij} \in \mathbb{R}^d$  for which*

$$\left| f(\mathbf{x}) - \sum_{i=1}^{6d+3} d_i \sigma \left( \sum_{j=1}^{3d} c_{ij} \sigma(\mathbf{w}^{ij} \cdot \mathbf{x} - \theta_{ij}) - \gamma_i \right) \right| < \varepsilon$$

for all  $\mathbf{x} = (x_1, \dots, x_d) \in [0, 1]^d$ .

Theorems 3 and 4 only implies that the approximation ability of two-layer neural network for appropriate activation functions. These activation functions can be quite pathological to achieve the desired accuracy. It is unclear if the activation functions used in practice, such as sigmoid functions, suffice for accurate approximation.

For any  $f \in C[0, 1]^d$ , let the corresponding superposition decomposition be eq. (27). Consider the family of the functions  $f$  where the corresponding  $g \in W_{1,1}^p$  and  $\phi_i(x) \in W_{1,1}^p$ , then by theorem 2, there exist constants  $w_{ri}$ ,  $a_{ri}$ ,  $b_{ri}$  and a constant  $C > 0$  independent of  $r$ , such that

$$\left| g(t) - \sum_{i=1}^r w_{ri} \sigma(a_{ri}t + b_{ri}) \right| \leq \frac{C \|g\|_{W_{1,1}^p}}{r} \quad (28)$$

We call the function family of  $f$  a *regular family* if  $w_{ri}$ ,  $a_{ri}$  are bounded independent of  $r$ , i.e., there exists a constant  $\tilde{C}$ , s.t.

$$|w_{ri}| \leq \tilde{C}, |a_{ri}| \leq \tilde{C} \quad i = 1, 2, \dots, r, \quad \forall r \quad (29)$$

**Remark 1.** The assumptions that  $g \in W_{1,1}^p$  and  $\phi_i(x) \in W_{1,1}^p$  are critical for the error bound. Note that even strictly increasing continuous functions can be very pathological. An example is the the Cantor function  $\mathcal{D}(x)$ .  $\mathcal{D}(x)$  is a nondecreasing continuous function but does not have a weak derivative.  $x + \mathcal{D}(x)$  will then be a strictly increasing continuous function but has no weak derivatives. We want to avoid this kind of functions in this paper.

We now derive an explicit error bound for approximating a function in the regular family by a two-layer neural network with sigmoid functions as the activation functions.

**Theorem 5.** Let  $\sigma$  be the standard sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (30)$$

then for any  $f$  in the regular family and  $\varepsilon > 0$ , there exists a two layer neural network with hidden units  $s = \mathcal{O}\left(\frac{d^2}{\varepsilon^2}\right)$  (the first layer) and  $r = \mathcal{O}\left(\frac{d}{\varepsilon}\right)$  (the second layer), and constants  $d_i$ ,  $c_{ij}$ ,  $\theta_{ij}$ ,  $\gamma_i$  and vectors  $\mathbf{w}^{ij} \in \mathbb{R}^d$  for which

$$\left| f(\mathbf{x}) - \sum_{i=1}^r d_i \sigma \left( \sum_{j=1}^s c_{ij} \sigma(\mathbf{w}^{ij} \cdot \mathbf{x} - \theta_{ij}) - \gamma_i \right) \right| < \varepsilon \quad (31)$$

for all  $\mathbf{x} = (x_1, \dots, x_d) \in [0, 1]^d$ .

*Proof.* See [44]. □

The theory for multiple layer (greater than 2) neural networks is elusive. However, Theorems 2 and 5 shed lights on the potential advantage of deeper neural networks: the hidden neuron sizes grow exponentially with the input dimensionality for one layer; they

grow polynomially for two layers. We conjecture that for sufficiently many layers such as  $\mathcal{O}(d)$  layers, the numbers can be further reduced to  $\mathcal{O}(1)$ . For example, the famous ResNet has hundreds of layers but each layer is a small convolution filter. The deep but thin neural network enables us to learn high dimensional functions and bypass the curse of the dimensionalities. The investigation will be left to the future.

### 3.3. Error Analysis for Diffusion Equations

We now conduct error analysis for the framework. To keep the discussion concrete and simple, we consider the 1D diffusion equation where the conductivity is an unknown continuous function.

$$u_t(x, t) = f(x)u_{xx}(x, t) \quad x \in [-1, 1], t \in (0, T] \quad (32)$$

$$u(\pm 1, t) = \varphi_{\pm 1}(t) \quad t \in (0, T] \quad (33)$$

$$u(x, 0) = u_0(x) \quad x \in [-1, 1] \quad (34)$$

The observed dataset is 2 sequential snapshots at time  $t = t_0$

$$M = \{(U_0 = u(\mathbf{x}, t) + \mathbf{W}_0, U_1 = u(\mathbf{x}, t) + \mathbf{W}_1) | \mathbf{x} = \{x_1, x_2, \dots, x_n\}\} \quad (35)$$

and  $\mathbf{W}_0$  and  $\mathbf{W}_1$  are noise. To put it in another way, given the observed dataset  $M \in \mathbb{R}^{2n}$  and the model specification eq. (32), we want to find the best function  $f_\theta$  that can explain the dataset. We discretize eq. (32) using the uniform grids. Let  $-1 < x_1 < x_2 < \dots < x_n = 1$  be the spacial grids with  $x_{i+1} - x_i = h$ .

Before we give the convergence rate of the neural network approach, we first investigate several error sources under the framework.

*Observation error.*  $\mathbf{W}_0$  and  $\mathbf{W}_1$  in this example represent *observation error*, which may come from the measurement error and/or errors in model specification. The former is easy to understand. For example, since there is inherently unpredictable fluctuations in the readings of a measurement apparatus or the experimenter's interpretation of the instrumental reading, the *random error* is almost inevitable in the observations. The errors in the model specification are due to the unrealistic assumption in our models. For instance, we have assumed that the underlying stochastic process is the Wiener process. However, it is well-known that for many physical phenomena such as hydrodynamics, quantum mechanics and so on may follow sub-diffusion/super-diffusion in some cases [45]. In other words, the observations suffer from *system errors*.

Yet, in this article, we assume that the dataset error is the random error, i.e., the model specification eq. (32) is correct but the observations may have i.i.d. noise  $\mathbf{W}_1, \mathbf{W}_2$ . To quantify the error, we make the following assumption

**Assumption 2** (Observation Error). *There exists a positive constant  $\varepsilon_d > 0$  such that*

$$\|\mathbf{W}_0\|_\infty \leq \varepsilon_o \quad \|\mathbf{W}_1\|_\infty \leq \varepsilon_o \quad (36)$$

*Consistency Error.* Another important source of error comes from the numerical discretization of eq. (32). We adopt the standard Crank Nicolson scheme for this problem, which has second-order accuracy both in time and space, which reads

$$\frac{v_i^{N+1} - v_i^N}{\Delta t} = f_i \frac{v_{i+1}^N + v_{i-1}^N - 2v_i^N}{2h^2} + f_i \frac{v_{i+1}^{N+1} + v_{i-1}^{N+1} - 2v_i^{N+1}}{2h^2} \quad i = 2, 3, \dots, n-1 \quad (37)$$

together with boundary conditions

$$v_n^{N+1} = \varphi(1) \quad v_1^{N+1} = \varphi(-1) \quad (38)$$

Note in eq. (37) we have used one-step discretization. This is because we have adopted a semi-implicit scheme and it is still stable for reasonable large  $\Delta t$ . In other cases such as an explicit scheme is used or high precision in temporal direction is required, we may consider multiple steps and formulate an accrued loss function.

Equations (37) and (38) leads to a tridiagonal linear system. The method is known to have a local error with an upper bound  $\varepsilon_c \leq C_1(\Delta t^2 + h^2) = \mathcal{O}(\Delta t^2 + h^2)$  for some  $C_1 > 0$ .

$$\left| \frac{u_i^{N+1} - u_i^N}{\Delta t} - f_i \frac{u_{i+1}^N + u_{i-1}^N - 2u_i^N}{2h^2} - f_i \frac{u_{i+1}^{N+1} + u_{i-1}^{N+1} - 2u_i^{N+1}}{2h^2} \right| \leq \varepsilon_c \quad i = 2, 3, \dots, n-1$$

The loss function eq. (7) can be then derived easily

$$\text{loss}(\theta) = \sum_{i=2}^{n-1} \left| \frac{U_{1i} - U_{0i}}{\Delta t} - f_\theta(x_i) \frac{U_{1,i+1} + U_{1,i-1} - 2U_{1i}}{2h^2} - f_\theta(x_i) \frac{U_{0,i+1} + U_{0,i-1} - 2U_{0,i}}{2h^2} \right|^2 \quad (39)$$

Note we have not included the boundary term since it does not include information about  $\theta$ . In addition, even if  $\text{loss}(\theta)$  is minimized to zero, it does not guarantee that we will obtain a good approximation to  $f(x)$ . Instead, in this case, the system begins to fit the error, which is known as *overfitting*. There are many techniques to alleviate overfitting; for example, we can collect more observations, use regularizers, terminate the optimization early, etc. We will not dive deep into this topic in the paper since for all the numerical examples we have not observed such issues.

*Optimization error.* Errors of this type may arise because the neural network architecture has been poorly designed or the inverse problems itself have multiple solutions (and therefore the loss function is multimodal). One particular problem the community has recognized is the local minimums for the non-convex loss function. However, the community seems very positive about escaping the local minimums using optimization techniques such as stochastic gradient descent [46] or specific neural network architectures [47]. In addition, local minimums do not necessarily mean bad results. As we have discussed, driving the loss function to zero may cause overfitting. A local minimum may give a reasonable estimation of  $f(x)$  in many cases. In this paper, we will show that if the other sources (observation error and consistency error) of error is small, a small optimization error will yield a good estimation. To avoid sophisticated discussion on the optimization phase, we make the following simple assumption

**Assumption 3** (Optimization Error). *There exists a positive constant  $\varepsilon_{opt} > 0$  such that*

$$\left| \frac{U_{1i} - U_{0i}}{\Delta t} - f_\theta(x_i) \frac{U_{1,i+1} + U_{1,i-1} - 2U_{1i}}{2h^2} - f_\theta(x_i) \frac{U_{0,i+1} + U_{0,i-1} - 2U_{0,i}}{2h^2} \right| \leq \varepsilon_{opt}$$

For the convenience of discussion, we assume that the time corresponds to  $U_0$  and  $U_1$  are  $t_0$  and  $t_1$  respectively. Now we are ready to state our main theorem

**Theorem 6.** *If the following assumptions are satisfied*

- Assumptions 2 and 3 are satisfied.
- According to Assumption 1 and Theorem 1, there exists constants  $F_0, F_2 > 0$  such that
- The exact solution  $u \in C^4([-1, 1] \times [t_0, t_1])$  and

$$\delta_2 = \arg \min_{(x,t) \in [-1,1] \times [t_0, t_1]} |u_{xx}(x, t)| \quad \delta_4 = \arg \max_{(x,t) \in [-1,1] \times [t_0, t_1]} |u_{xxxx}(x, t)| \quad (41)$$

here  $\delta_2$  is strictly positive, i.e.,

$$\delta_2 > 0 \quad (42)$$

The exact conductivity function  $f \in C^2([-1, 1])$  and denote

$$F_2^f = \max_{x \in [-1, 1]} |f''(x)| \quad (43)$$

- $h$  is sufficiently small in the sense that

$$h < \sqrt{\frac{6\delta_2}{\delta_4}} \quad (44)$$

Then the calibrated  $f_\theta(x)$  has error estimate  $\forall x \in [-1 + h, 1 - h]$

$$|f_\theta(x) - f(x)| \leq \frac{2C_1}{\delta_2} \Delta t^2 + \left( \frac{2C_1}{\delta_2} + \frac{F_2 + F_2^f}{2} \right) h^2 + \frac{2}{\delta_2} \varepsilon_{opt} + \frac{4}{\delta_2} \left( \frac{1}{\Delta t} + \frac{2F_0}{h^2} \right) \varepsilon_o \quad (45)$$

In other words,

$$|f_\theta(x) - f(x)| = \mathcal{O} \left( \Delta t^2 + h^2 + \varepsilon_{opt} + \left( \frac{1}{\Delta t} + \frac{1}{h^2} \right) \varepsilon_o \right) \quad (46)$$

Note the assumption eq. (42) is reasonable since if  $u_{xx}(x, t) = 0$  in some region of  $(x, t)$ , the observations will convey little information about  $f(x)$  since  $f(x)u_{xx}(x, t) \equiv 0$ . If  $u_{xx}(x, t)$  is too small, the inverse problem will be quite ill-conditioned and we may have difficulty regularizing the neural network. Nevertheless, in the numerical example even though the second order derivatives of our exact solution vanish near  $x = 0$ , we are still able to infer  $f(0)$  quite accurately. This also demonstrates the robustness of our method.

*Proof.* The proof is split into two parts. We use the notation  $u_{0i} = u(x_i, t_0)$  and  $u_{1i} = u(x_i, t_1)$ .

- Error bound on  $|f_\theta(x_i) - f(x_i)|$ .

Let  $e_{opt,i}$  be the optimization error

$$\frac{U_{1i} - U_{0i}}{\Delta t} - f_\theta(x_i) \frac{U_{1,i+1} + U_{1,i-1} - 2U_{1i}}{2h^2} - f_\theta(x_i) \frac{U_{0,i+1} + U_{0,i-1} - 2U_{0i}}{2h^2} = e_{opt,i} \quad (47)$$

Note that  $U_{0i} = u_{0i} + \mathbf{W}_{0i}$  and  $U_{1i} = u_{1i} + \mathbf{W}_{1i}$ , plug them into eq. (47) we have

$$\begin{aligned} & \frac{u_{1i} - u_{0i}}{\Delta t} - f_\theta(x_i) \frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{2h^2} - f_\theta(x_i) \frac{u_{0,i+1} + u_{0,i-1} - 2u_{0i}}{2h^2} \\ &= e_{opt,i} - \left( \frac{\mathbf{W}_{1i} - \mathbf{W}_{0i}}{\Delta t} - f_\theta(x_i) \frac{\mathbf{W}_{1,i+1} + \mathbf{W}_{1,i-1} - 2\mathbf{W}_{1i}}{2h^2} - f_\theta(x_i) \frac{\mathbf{W}_{0,i+1} + \mathbf{W}_{0,i-1} - 2\mathbf{W}_{0i}}{2h^2} \right) \end{aligned} \quad (48)$$

Let  $e_{c,i}$  be the consistency error at  $x_i$  we have

$$\frac{u_{1i} - u_{0i}}{\Delta t} - f(x_i) \frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{2h^2} - f(x_i) \frac{u_{0,i+1} + u_{0,i-1} - 2u_{0i}}{2h^2} = e_{c,i} \quad (49)$$

Subtracting eq. (49) from eq. (48) we have

$$\begin{aligned} & (f(x_i) - f_\theta(x_i)) \left[ \frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{2h^2} + \frac{u_{0,i+1} + u_{0,i-1} - 2u_{0i}}{2h^2} \right] \\ &= e_{opt,i} - e_{c,i} \\ & - \left( \frac{\mathbf{W}_{1i} - \mathbf{W}_{0i}}{\Delta t} - f_\theta(x_i) \frac{\mathbf{W}_{1,i+1} + \mathbf{W}_{1,i-1} - 2\mathbf{W}_{1i}}{2h^2} - f_\theta(x_i) \frac{\mathbf{W}_{0,i+1} + \mathbf{W}_{0,i-1} - 2\mathbf{W}_{0i}}{2h^2} \right) \end{aligned} \quad (50)$$

Since  $\delta_2 > 0$  we must have  $u_{xx} \geq \delta_2$  or  $u_{xx} \leq \delta_2$  for  $(x, t) \in [-1, 1] \times [t_0, t_1]$ . We might as well assume  $u_{xx} \geq \delta_2 > 0$ .

Note that

$$\frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{h^2} = u_{xx}(x_i, t_1) + \frac{h^2 u_{xxxx}(x_i, t_1)}{12}$$

we have

$$\frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{h^2} \geq \delta_2 - \frac{h^2 \delta_4}{12}$$

invoking eq. (44) we obtain

$$\frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{h^2} \geq \frac{1}{2} \delta_2$$

Likewise

$$\frac{u_{0,i+1} + u_{0,i-1} - 2u_{0i}}{h^2} \geq \frac{1}{2} \delta_2$$

therefore, we have

$$\frac{u_{1,i+1} + u_{1,i-1} - 2u_{1i}}{2h^2} + \frac{u_{0,i+1} + u_{0,i-1} - 2u_{0,i}}{2h^2} \geq \frac{1}{2} \frac{1}{2} \delta_2 + \frac{1}{2} \frac{1}{2} \delta_2 = \frac{1}{2} \delta_2 \quad (51)$$

On the other hand, invoking

$$|e_{opt,i}| \leq \varepsilon_{opt} \quad |e_{c,i}| \leq \varepsilon_c \quad |W_{ki}| \leq \varepsilon_o \quad k = 0, 1 \quad (52)$$

Combining eqs. (50) to (52) we have

$$|f(x_i) - f_\theta(x_i)| \leq \frac{2}{\delta_2} \left[ \varepsilon_{opt} + \varepsilon_c + 2 \left( \frac{1}{\Delta t} + \frac{2F_0}{h^2} \right) \varepsilon_o \right] \quad (53)$$

- Error bound on  $|f_\theta(x) - f(x)|$ .

Define the error function  $F : [-1, 1] \rightarrow \mathbb{R}$  as  $F(x) = f_\theta(x) - f(x)$ , we have already obtained the error bound for  $F(x_i)$  in eq. (53). For any other point  $x \in [-1+h, 1-h]$ , we can always find the interval that contains  $x$ :  $x \in [x_i, x_{i+1}]$ . Using Taylor's theorem we have

$$F(x_i) = F(x) + F'(x)(x_i - x) + \int_x^{x_i} (x_i - y) F''(y) dy \quad (54)$$

$$F(x_{i+1}) = F(x) + F'(x)(x_{i+1} - x) + \int_x^{x_{i+1}} (x_{i+1} - y) F''(y) dy \quad (55)$$

Let  $\alpha = \frac{x_{i+1}-x}{h}$ , then we have

$$\alpha F(x_i) + (1 - \alpha) F(x_{i+1}) \quad (56)$$

$$= F(x) + \alpha \int_x^{x_i} (x_i - y) F''(y) dy + (1 - \alpha) \int_x^{x_{i+1}} (x_{i+1} - y) F''(y) dy \quad (57)$$

We obtain

$$|F(x)| \leq \alpha |F(x_i)| + (1 - \alpha) |F(x_{i+1})| + \frac{1}{2} |F''(x)|_\infty h^2$$

Note

$$|F''(x)| = |f''_\theta(x)| + |f''(x)| \leq F_2 + F_2^f$$

we have

$$|F(x)| \leq \frac{2}{\delta_2} \left[ \varepsilon_{opt} + \varepsilon_c + 2 \left( \frac{1}{\Delta t} + \frac{2F_0}{h^2} \right) \varepsilon_o \right] + \frac{h^2}{2} (F_2 + F_2^f) \quad (58)$$

$$\leq \frac{2C_1}{\delta_2} \Delta t^2 + \left( \frac{2C_1}{\delta_2} + \frac{F_2 + F_2^f}{2} \right) h^2 + \frac{2}{\delta_2} \varepsilon_{opt} + \frac{4}{\delta_2} \left( \frac{1}{\Delta t} + \frac{2F_0}{h^2} \right) \varepsilon_o \quad (59)$$

which finishes the proof.

□

**Remark 2.** We can see that discrepancy between the estimation  $f_\theta(x)$  and the exact function  $f(x)$  is bounded by a linear combination of three error sources  $\varepsilon_o$ ,  $\varepsilon_d$  and  $\varepsilon_{opt}$ .  $\varepsilon_o$  is usually small if precise measurement can be conducted. When  $\Delta t$  or  $h$  is large, the consistency error dominates, and we expect to draw a “convergence plot” with respect to  $\Delta t$  and  $h$ . If  $\Delta t$  and  $h$  decrease to a certain level, the optimization error dominates and we will see the error saturates. It is either because the optimizer got stuck at some local minimum or the theoretical loss minimum is reached (which is called Bayes error in the machine learning community – a statistical lower bound on the error achievable for a given classification problem and associated choice of features [48]). In the first case, it may be possible that we can obtain different loss values for different runs (with a randomized initial guess). In general, the optimization error will be driven to Bayes error with the improvement of optimization techniques.

### 3.4. Sensitivity Analysis

The parametrized function  $f_\theta(x)$  offers us a way to do sensitivity analysis. When we are most concerned about a physical quantity, for example, the maximum value of the conductivity at a particular location  $x = x^*$  in our case, we want to know how reliable the prediction is under our framework. It is similar to uncertainty quantification, but there are two distinct differences:

- The estimation is anchored to a particular physical quantity instead of considering the solution as a whole.
- The estimation will depend on all three sources of errors as well as the neural network structure. In some sense, the estimation will also provide an approach to accessing the quality of the framework in general.

Assume that the physical quantity is expressed as

$$q(\theta) = Q(f_\theta) \quad (60)$$

where  $Q$  is functional. For example

$$Q(f_\theta) = \max_{x \in [-1,1]} f_\theta(x) \quad (61)$$

It is then possible to see the sensitivity of the quantity  $q(\theta)$  with respect to the parameter  $\theta$  by taking the gradient

$$\nabla_\theta q(\theta) = \nabla_\theta Q(f_\theta) \quad (62)$$

the latter can usually be done by first discretization  $Q(f_\theta)$  and then perform the automatic differentiation. For example, for eq. (61) we have

$$\nabla_\theta q(\theta) \approx \nabla_\theta \max_{i=1,2,3,\dots,n} \{f_\theta(x_i)\} \quad (63)$$

$$= \nabla_\theta f_\theta(x_{i^*}) \quad i^* = \arg \max_{i=1,2,3,\dots,n} \{f_\theta(x_i)\} \quad (64)$$

$\nabla_\theta f_\theta(x_{i^*})$  can be easily computed with automatic differentiation. After obtaining  $\nabla_\theta q(\theta)$ , we can give a sensitive region of the  $f_\theta(x)$  by

$$S_{\theta,q,\delta} = \{f_{\theta'}(x) | \theta' = \theta + \alpha \nabla_\theta q(\theta), |\alpha| \leq \delta\} \quad (65)$$

where  $\delta > 0$  is a tunable parameter that quantize the sensitivity.

The intuition is that if we anchor the physical quantity  $q(\theta)$ , it is most sensitive to the change of hidden parameters  $\theta$  in the positive/negative direction  $\nabla_\theta q(\theta)$ . Therefore  $S_{\theta,q,\delta}$  will be a stripe that contains  $f_\theta(x)$  (when  $\alpha = 0$ ). If the stripe deviates from the  $f_\theta$  too much or changes dramatically for a reasonable  $\alpha$ , the framework is then questionable, and we should investigate components such as the neural network structure. The sensitivity analysis provides us with a way to evaluate the quality of the framework.

As an example, we consider fitting the global yearly mean data <sup>1</sup> for some quantity related to CO<sub>2</sub>. We are most concerned about the maximum of the data and want to see how sensitive it is with respect to the neural network parameters. Figure 1 shows the sensitivity region computed using the method described above. Here  $\alpha$  is the largest step size in eq. (65). The width of the stripe describes the impact of a small change in the neural network parameters if the maximum value of the fitted value is the targeted physical quantity.

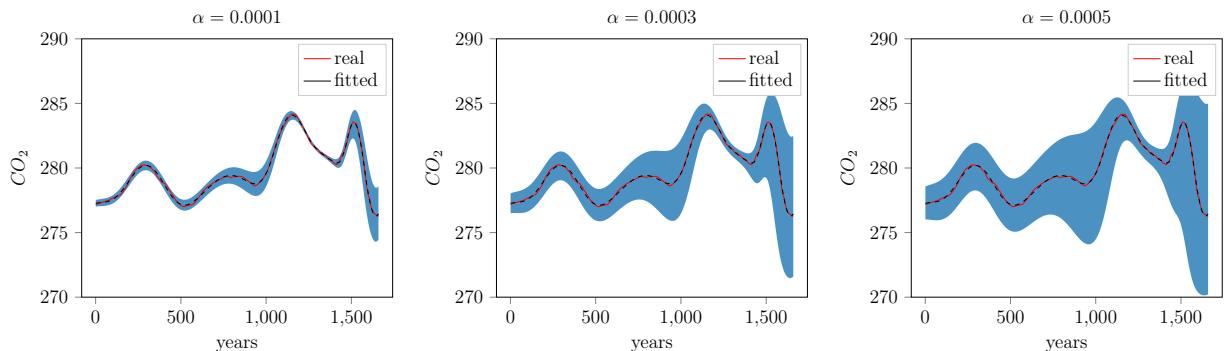


Figure 1: The width of the stripe describes the impact of a small change in the neural network parameters if the maximum value of the fitted value is the targeted physical quantity.

The study of the sensitivity analysis is at an early stage. Many problems need to be addressed. For example, how can we pick the reasonable  $\alpha$ ? Also, the relationship between the sensitivity analysis and uncertainty quantification of the model is not clear and requires more theoretical investigation. That will be left to the future work. Nevertheless, the numerical examples demonstrate that it captures the sensitivity of the physical quantity depending on the neural network model, and can potentially serve as a way for the diagnosis of the framework.

---

<sup>1</sup>[ftp://data.iac.ethz.ch/CMIP6/input4MIPs/UoM/GHGConc/CMIP/yr/atmos/UoM-CMIP-1-1-0/GHGConc/gr3-GMNHSH/v20160701/mole\\_fraction\\_of\\_carbon\\_dioxide\\_in\\_air\\_input4MIPs\\_GHGConcentrations\\_CMIP\\_UoM-CMIP-1-1-0\\_gr3-GMNHSH\\_0000-2014.csv](ftp://data.iac.ethz.ch/CMIP6/input4MIPs/UoM/GHGConc/CMIP/yr/atmos/UoM-CMIP-1-1-0/GHGConc/gr3-GMNHSH/v20160701/mole_fraction_of_carbon_dioxide_in_air_input4MIPs_GHGConcentrations_CMIP_UoM-CMIP-1-1-0_gr3-GMNHSH_0000-2014.csv)

## 4. Numerical Examples

We now apply the framework to three differential equations from various applications and demonstrate its effectiveness. The first equation is a diffusion equation from the elliptic equation family. We try to infer the unknown conductivity function. The second is a wave equation from the hyperbolic equation family. The unknown is the velocity fields. The problem is a significant and challenging one in seismic imaging. Instead of using the popular adjoint method used by the geophysics community, we provide another potential way to infer the velocity fields. The last problem is a variable coefficient nonlinear Burger's equation. The Burger's equation appeared as one of the simplest instances of a nonlinear system out of equilibrium in fluid dynamics [49]. The study may offer new approaches to the inverse problems in turbulence modeling.

For all the numerical experiment below, we have the following common settings

- For the neural network model, we let  $n_l = 3$  and the number of hidden neurons per layer be  $n_h = 20$ . We have tested extensively for different  $n_l = 3, 4, 5, 6, 7, 8, 9, 10$  and  $n_h = 20, 40, 60, 80$  but find basically no difference except for large  $n_l$  and  $n_h$ , the optimizer has difficulty converging within the predetermined number of iterations.
- We use L-BFGS optimizer, which enjoys convergence property similar to second order method. The iterator will stop if either of the following is satisfied

$$|f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)| \leq \varepsilon_1 |f(\mathbf{x}_k)| \quad (66)$$

$$|\nabla f(\mathbf{x}_{k+1})| \leq \varepsilon_2 \quad (67)$$

where  $\mathbf{x}_k$  is the variable value at iteration  $k$ . We set  $\varepsilon_1 = 10^{-12}$ ,  $\varepsilon_2 = 10^{-12}$ . In addition, if not mentioned, the number of maximum iteration is set as 5000.

### 4.1. Diffusion Equation

We let the true model be

$$u_t(x, t) = c(x)u_{xx}(x, t) + f(x) \quad (x, t) \in [-1, 1] \times [0, T] \quad (68)$$

where  $u(x) = e^{-\pi^2 t} \sin(\pi x)$ ,  $c(x) = 1 + e^{-(x-0.5)^2}$  and

$$f(x, t) = \pi^2 \left( 1.0 + e^{-(x-0.5)^2} \right) e^{-\pi^2 t} \sin(\pi x) - \pi^2 e^{-\pi^2 t} \sin(\pi x)$$

Figure 2 shows the case where the snapshot is taken at  $t = 0.1$ , with  $\Delta t = 0.001$  and  $h = 0.002$ . We can see that the calibrated value of  $c(x)$  matches perfectly with the exact value. In this case, we do not introduce any noise to the solution, i.e.,  $\varepsilon_o \equiv 0$ .

We now conduct a systematic study on the influence of  $\Delta t$  and  $h$ . In fig. 3 we take the snapshots at  $t = 0.1$  and  $t = t + \Delta t$ , with various  $\Delta t$  and  $h$ , and  $n = \frac{2}{h}$  in the plots. We see a clear pattern exactly predicted in Theorem 6: a second-order convergence in  $\Delta t$  and  $h$ , and at some point, the error ceases to decrease and remains at a constant level. The numerical results not only demonstrate our theory but also offer us the direction for further

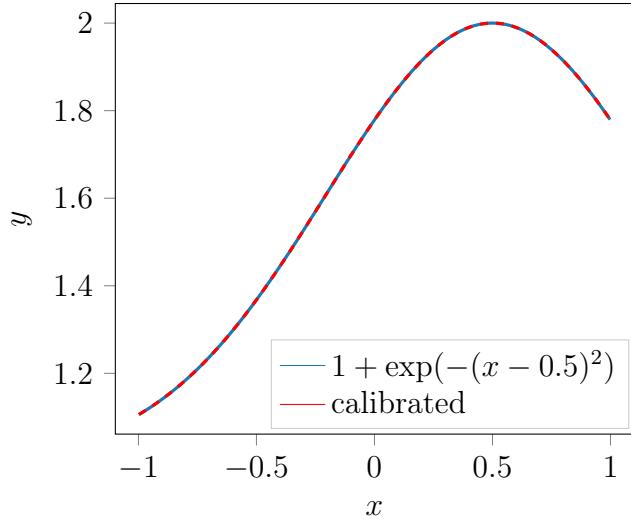


Figure 2: The snapshots are taken at  $t = 0.1$  and  $t = t + \Delta t$ , with  $\Delta t = 0.001$  and  $h = 0.002$ . We can see that the calibrated value of  $c(x)$  matches perfectly with the exact value.

optimization: **collecting more data is of little benefit after the error saturates; instead, we should focus on optimization or noise reduction.** It is quite a surprising implication since typically people believe more data is beneficial.

We can also see that when the error saturates, the level of error are not the same for different  $n$  or  $\Delta t$ . This implies that the spacial discretization error in the first plot or the temporal discretization error in the second plot dominates. If  $h$  or  $\Delta t$  is sufficiently small, the level where the error saturates will be approximately the same, as shown in fig. 3.

Finally, we carry out the sensitivity analysis. The settings are the same as that in fig. 2 except that we add i.i.d. Gaussian noise with mean 0 and standard deviation of  $3 \times 10^{-7}$  to the observations. The physical quantity we are interested is the conductivity function value at  $x = 0$ , i.e., the prediction for  $c(0)$ . Following the discussion in Section 3.4, we generate the sensitive region of this quantity. We see that the stripe gradually grows and eventually incorporates the exact solution. We also see that the variance near the boundary is smaller than that in the center. This observation is consistent with our intuition that if the interesting physical quantity is the value of  $c(x)$  at 0, the far-away region should be less impacted. Therefore a modification to the neural network parameter will not result in significant change in the far-away value of the calibrated function.

#### 4.2. Wave Equation

The second example is concerned about wave equations. Consider the model problem

$$u_{tt}(x, t) = c(x)u_{xx}(x, t) \quad (x, t) \in [-1, 1] \times [0, 1] \quad (69)$$

where  $c(x)$  is the unknown velocity field. We set the initial condition as

$$u(x, 0) = e^{-10x^2} \quad (70)$$

The two manufactured velocity fields are

$$c_1(x) = 1 + \exp(-(x - 0.5)^2) \quad c_2(x) = \begin{cases} 2 & x \in [-1, -0.5) \\ 1 & x \in [-0.5, -0.1) \\ 0.2 & x \in [-0.1, 0.1) \\ 1.0 & x \in [0.2, 0.3) \\ 1.5 & x \in (0.3, 1.0] \end{cases} \quad (71)$$

Here we showcase two modifications to the framework and demonstrate its flexibility for practical problems

- Suppose we have snapshots at different times and three sequential snapshots for each, i.e.,  $|\mathcal{T}| > 1$ . We want to make full use of the observations. Therefore we construct an ensemble loss function as shown in eq. (7).
- Suppose we have prior knowledge  $c(x) \in [0, 2]$ . To incorporate the prior knowledge, we let the last layer in eq. (10) be

$$\mathbf{y}_{n_l+1}(\mathbf{y}_{n_l}) = \tanh(\mathbf{W}_{n_l}\mathbf{y}_{n_l} + \mathbf{b}_{n_l}) + 1 \quad (72)$$

Since we may not have analytical solution for the velocity fields eq. (71), we first run the simulation with  $h = 0.004$ ,  $\Delta t = 0.0001$  using the numerical scheme

$$\frac{v_i^{N+2} - 2v_i^{N+1} + v_i^N}{\Delta t^2} = c_i^2 \frac{v_{i+1}^{N+1} - 2v_i^{N+1} + v_{i-1}^{N+1}}{\Delta t^2} \quad i = 2, 3, \dots, n-1 \quad (73)$$

then we use the values at  $t = \frac{k}{3} - 20\Delta t, \frac{k}{3} - 10\Delta t, \frac{k}{3}, k = 1, 2, 3$  as the observation. The loss function eq. (7) is also formulated using eq. (73) except that the time step is now  $10\Delta t$ . Note that  $\frac{2 \cdot 10\Delta t}{h} = 0.5 < 1$ , the CFL condition is satisfied. We also add i.i.d. Gaussian noise sampled from  $\mathcal{N}(0, (10\Delta t)^2)$  to the observations. Figure 5 shows examples of the snapshots for the velocity fields  $c_1(x)$  and  $c_2(x)$ . Note that since  $10\Delta t$  is quite small, the nearby sequential snapshots almost overlap. However, our algorithm is able to infer the velocity fields from the subtle difference.

We also compare the method with neural networks with that of least square methods. In fig. 6, the left columns show the calibrating results using the neural networks while the right columns correspond to least square methods. For the least square methods, we have used Newton-CG algorithms for minimizing the loss function with discrete velocity field values as unknown variables. We can see that the least square methods are vulnerable to noise and do not yield a solution as smooth as the neural network methods. Especially for  $c_2(x)$ , the neural network captures the transition between different velocity levels and yields a constant value at each level. The results are quite remarkable since it is generally challenging to capture those discontinuities in inverse problems. The fit is not perfect though because we have added noise to the dataset.

### 4.3. Variable Coefficient Burger's Equation

Finally, we study a variable-coefficient Burgers equation arising in the modeling of segregation of dry bidisperse granular mixtures [50]. The numerical PDE is due to [51]

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}(u(1-u)f(x)) - D \frac{\partial^2 u}{\partial x^2} = 0, \quad x \in [-1, 1], t > 0 \quad (74)$$

here  $u = u(x, t)$  is the concentration. The function  $f(x)$  is called the percolation velocity and  $D$  is the constant diffusivity due to inter-particle collisions. In this numerical example, we consider the case where we have partially observed evolution of  $u(x, t)$  and want to infer the percolation velocity  $f(x)$ . To formulate the loss function, we use the following numerical discretization scheme from [50]

$$\begin{aligned} & \left[ \frac{\Delta t}{8h} (2 - v_{j+1}^{N+1} - 2v_j^N) f(x_{j+1}) - \frac{D\Delta t}{2h^2} \right] v_{j+1}^{N+1} + \left[ 1 + \frac{D\Delta t}{h^2} \right] v_j^{N+1} \\ & + \left[ -\frac{\Delta t}{8h} (2 - v_{j-1}^{N+1} - 2v_j^N) f(x_{j-1}) - \frac{D\Delta t}{2h^2} \right] v_{j-1}^{N+1} \\ & = -\frac{\Delta t}{8h} [v_{j+1}^N (2 - v_{j+1}^N) f(x_{j+1}) - v_{j-1}^N (2 - v_{j-1}^N) f(x_{j-1})] + v_j^N \\ & + \frac{D\Delta t}{2h^2} (v_{j+1}^N - 2v_j^N + v_{j-1}^N) \quad j = 2, 3, \dots, n-1 \end{aligned} \quad (75)$$

Note eq. (75) is a nonlinear equation, and in practice, we can use Newton's methods to obtain  $v^{N+1}$  from  $v^N$ . For the inverse problem, the loss function is constructed by taking the square sum of the difference between the left-hand side and the right-hand side in eq. (75) over  $j = 2, 3, \dots, n-1$ .

For this problem, we use a neural network with four hidden layers and each layer has 40 neurons. We have found that the performance is insensitive to these hyper-parameters as long as they stay in a reasonable range (neither too deep nor too heavy). The exact solution is simulated using  $\Delta t = 2 \times 10^{-6}$  and  $h = 0.004$  while we take 20 snapshots at  $t = 0.01, 0.02, 0.03, \dots, 0.20$ . Noise is sampled from i.i.d. Gaussian distribution with zero mean and  $10^{-5}$  standard deviation. The time step is quite coarse, but we show it does not compromise our algorithm to recover  $f(x)$ . We fix  $D = 0.1$ . We also apply  $L_2$  normalization to the neural networks with penalty 0.01 to enforce smoothness. The initial condition is

$$u(x, 0) = \frac{c - f(x)}{2f(x)} \left( -1 + \frac{(x - ct) \tanh(c - f(x))}{2D} \right) \quad (76)$$

where

$$f(x) = -1 + \exp(-(x - 0.5)^2) \quad (77)$$

The corresponding solution profile is shown in fig. 7.

For sensitivity analysis, we consider the maximum value of  $f(x)$  as the interesting physical quantity, i.e.,

$$Q(f_\theta) = \max_{x \in [-1, 1]} f_\theta(x) \quad (78)$$

Figure 8 shows the calibrated results and the corresponding sensitivity regions with different step sizes  $\alpha$ . The stripe gradually grows and finally encloses the exact  $f(x)$ . We also see that the values far-away from the maxima are less sensitive to the parameters.

As a comparison, fig. 9 shows the calibrated results using least square methods with the same settings. We can see that this method is more susceptible to random noise and has stability issues compared to the neural network approach.

## 5. Conclusion

We have introduced a new framework for inverse problems in differential equations based on neural networks and automatic differentiation. It leverages the current development of machine learning techniques, especially deep learning. Thanks to the open source frameworks developed by the machine learning community, such as TensorFlow [52], PyTorch [53], MXNet [54], and so on, the inverse problem solvers can be easily implemented and incorporated into traditional numerical PDE codes. In future works, we show that the approach can also be easily combined with more sophisticated numerical methods such as finite element methods and finite volume methods, which are widely deployed in software for fluid dynamics, solid mechanics, and more. In this framework, a crucial step is to represent the unknown fields using neural networks, instead of discrete values or a linear combination of kernel basis functions. The procedure offers the universal approximation ability of neural networks, along with regularizing solution smoothness, bypassing the curse of dimensionality and leveraging the high-performance computing developed for deep learning. It works seamlessly with forward-simulation schemes and benefits from the numerical stability and consistency. Also, we are also able to provide convergence analysis and contain the error, using theoretical tools from classical computational mathematics.

We performed initiatory demonstrations for the effectiveness of the framework on elliptic and hyperbolic problems, as well as nonlinear problems such as non-constant coefficient Burger’s equation. In the diffusion equation, the convergence rate is shown to be consistent with our analysis. That is noteworthy since we now have a theory guide for performing data collection and optimization.

Based on the framework, we proposed a sensitivity analysis for the solution we obtained. The algorithm is based on the automatic differentiation mechanism and frees researchers from the tedious and error-prone process of deriving the gradients by hand.

There are numerous research opportunities for solving inverse problems in differential equations based on neural networks and automatic differentiation

- Tailored automatic differentiation software for engineering applications. Although the current software – such as TensorFlow we have used for this article – applies to a variety of problems, the performance is not tuned and optimized for engineering usage. Many subtle operators such as special treatment of boundary conditions will easily break the matrix calculation based software for deep learning nowadays.
- Non-convex optimization for engineering problems. Although it is a well-known issue that problems involving neural networks may lead to non-convex optimization prob-

lems, based on authors' experience, such problems are less severe for the engineering problems considered compared to their counterparts in computer vision, speech recognition or natural language processing. Specific optimization techniques and theories may be developed for its own sake.

- Neural network architectures for engineering problems. Currently, there are plenty of neural network architectures, and new research is going on. It would be interesting and fruitful to investigate the proper family of neural networks that are suitable for engineering problems, which have their unique properties (such as known singularity at some locations).
- Convergence theory for inverse problems using the neural networks. As we have seen in the analysis, it is possible to develop general theories about solving inverse problems using neural networks. Currently, our theory is case-dependent. A general theory is more attractive and beneficial for the development of the framework.
- Uncertainty quantification. It is important to quantify the uncertainty under such framework since errors are inevitable in practical problems. We have proposed an tentative method to quantify the "sensitivity" using the automatic differentiation. The sensitivity region can be computed nearly for free thanks to the framework we have used: automatic differentiation is naturally embedded. However, a much more general uncertainty quantification approach shall be developed, and desirably leverages the automatic differentiation mechanism.

In conclusion, we believe the new framework will potentially bring useful tools for analysis and calibration of inverse problems in differential equations.

## References

- [1] Y. Tian, Y. Zhou, J. Chung, M. Chung, and J. Ning. A Bayesian Approach to Linear Inverse Problems in Seismic Tomography. *A Bayesian Approach to Linear Inverse Problems in Seismic Tomography*, 2014.
- [2] S. R. Arridge. Optical Tomography in Medical Imaging. *Inverse problems*, 15(2):R41, 1999.
- [3] Y. Almleaky, J. Brown, and P. Sweet. Density Diagnostics and Inhomogeneous Plasmas. I-Isothermal Plasmas. *Astronomy and Astrophysics*, 224:328–337, 1989.
- [4] A. M. Stuart. Inverse Problems: A Bayesian Perspective. *Acta Numerica*, 19:451–559, 2010.
- [5] B. Jin, P. Maas, and O. Scherzer. Sparsity Regularization in Inverse Problems. *Inverse Problems*, 33(6):060301, 2017.
- [6] I. Daubechies, M. Defrise, and C. De Mol. An Iterative Thresholding Algorithm for Linear Inverse Problems with A Sparsity Constraint. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 57(11):1413–1457, 2004.
- [7] O. Scherzer, M. Grasmair, H. Grossauer, M. Haltmeier, and F. Lenzen. *Variational methods in imaging*. Springer, 2009.
- [8] T. Schuster, B. Hofmann, and B. Kaltenbacher. Tackling Inverse Problems in A Banach Space Environment: From Theory to Applications. *Inverse Problems*, 28(10):100201, 2012.
- [9] T. Schuster, B. Kaltenbacher, B. Hofmann, and K. S. Kazimierski. *Regularization methods in Banach spaces*, volume 10. Walter de Gruyter, 2012.
- [10] I. Daubechies, M. Defrise, and C. De Mol. Sparsity-Enforcing Regularisation and ISTA Revisited. *Inverse problems*, 32(10):104001, 2016.
- [11] Bayesian Inference in Inverse Problems. <http://www.stat.rice.edu/~jrojo/4th-Lehmann/slides/Mallick.pdf>. (Accessed on 01/03/2019).
- [12] M. Dashti and A. M. Stuart. The Bayesian Approach to Inverse Problems. *Handbook of Uncertainty Quantification*, pages 1–118, 2016.
- [13] T. Bui-Thanh. A Gentle Tutorial on Statistical Inversion Using the Bayesian Paradigm. *Institute for Computational Engineering and Sciences, Technical Report ICES-12-18*, 2012.
- [14] D. Talay. Probabilistic Numerical Methods for Partial Differential Equations: Elements of Analysis. Probabilistic Numerical Methods for Partial Differential Equations: Elements of Analysis, pages 148–196. Springer, 1996.
- [15] D. Patella. Geophysical Tomography in Engineering Geology: an Overview. *arXiv preprint physics/0512154*, 2005.
- [16] S. G. Johnson. Notes on Adjoint Methods for 18.336. Online at <http://math.mit.edu/stevenj/18.336/adjoint.pdf>, 2007.
- [17] J. R. Shewchuk et al. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, 1994.
- [18] M. C. Hall, D. G. Cacuci, and M. E. Schlesinger. Sensitivity Analysis of A Radiative-Convective Model By the Adjoint Method. *Journal of the Atmospheric Sciences*, 39(9):2038–2050, 1982.
- [19] A. Jameson. Aerodynamic Shape Optimization Using the Adjoint Method. *Lectures at the Von Karman Institute, Brussels*, 2003.
- [20] A. Fichtner, H.-P. Bunge, and H. Igel. The Adjoint Method in Seismology— II. Applications: Traveltimes and Sensitivity Functionals. *Physics of the Earth and Planetary Interiors*, 157(1-2):105–123, 2006.
- [21] R.-E. Plessix. A Review of the Adjoint-State Method for Computing the Gradient of A Functional with Geophysical Applications. *Geophysical Journal International*, 167(2):495–503, 2006.
- [22] M. B. Giles and N. A. Pierce. An Introduction to the Adjoint Approach to Design. *Flow, turbulence and combustion*, 65(3-4):393–415, 2000.
- [23] I. M. Elshafiey. Neural Network Approach for Solving Inverse Problems. 1991.
- [24] T. Ogawa, Y. Kosugi, and H. Kanada. Neural Network Based Solution to Inverse Problems. *Neural Network Based Solution to Inverse Problems*, volume 3, pages 2471–2476. IEEE, 1998.

- [25] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations. *arXiv preprint arXiv:1711.10561*, 2017.
- [26] 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.
- [27] G. Pang, L. Yang, and G. E. Karniadakis. Neural-Net-Induced Gaussian Process Regression for Function Approximation and PDE Solution. *arXiv preprint arXiv:1806.11187*, 2018.
- [28] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying Total Uncertainty in Physics-Informed Neural Networks for Solving Forward and Inverse Stochastic Problems. *arXiv preprint arXiv:1809.08327*, 2018.
- [29] H. Wang, L. Zhang, J. Han, and E. Weinan. DeePMD-Kit: A Deep Learning Package for Many-Body Potential Energy Representation and Molecular Dynamics. *Computer Physics Communications*, 228:178–184, 2018.
- [30] J. Han, Q. Li, et al. A Mean-Field Optimal Control Formulation of Deep Learning. *arXiv preprint arXiv:1807.01083*, 2018.
- [31] L. Zhang, H. Wang, and E. Weinan. Reinforced Dynamics for Enhanced Sampling in Large Atomic and Molecular Systems. I. Basic Methodology. *space*, 16:17.
- [32] L. Wu, Z. Zhu, C. Tai, et al. Understanding and Enhancing the Transferability of Adversarial Examples. *arXiv preprint arXiv:1802.09707*, 2018.
- [33] T. Glasmachers. LimitsofEnd-To-EndLearning. *arXiv e-prints*, page arXiv:1704.08305, April 2017.
- [34] G. Pang, L. Lu, and G. E. Karniadakis. FPINNs: Fractional Physics-Informed Neural Networks. *arXiv preprint arXiv:1811.08967*, 2018.
- [35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. *Imagenet Classification with Deep Convolutional Neural Networks*, pages 1097–1105, 2012.
- [36] X. Jia, L. Zhao, L. Zhang, J. He, and J. Xu. Modified Regularized Dual Averaging Method for Training Sparse Convolutional Neural Networks. *arXiv preprint arXiv:1807.04222*, 2018.
- [37] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. Deep Residual Learning for Image Recognition, pages 770–778, 2016.
- [38] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.
- [39] H. Gouk, E. Frank, B. Pfahringer, and M. Cree. Regularisation of Neural Networks By Enforcing Lipschitz Continuity. *arXiv preprint arXiv:1804.04368*, 2018.
- [40] H. N. Mhaskar. Neural Networks for Optimal Approximation of Smooth and Analytic Functions. *Neural computation*, 8(1):164–177, 1996.
- [41] X. Liu. Kolmogorov Superposition Theorem and Its Applications. 2015.
- [42] G. G. Lorentz, M. von Golitschek, and Y. Makovoz. *Constructive approximation: advanced problems*, volume 304. Springer Berlin, 1996.
- [43] V. Maiorov and A. Pinkus. Lower Bounds for Approximation By MLP Neural Networks. *Neurocomputing*, 25(1-3):81–91, 1999.
- [44] K. Xu and E. Darve. Calibrating Lévy Process From Observations Based on Neural Networks and Automatic Differentiation with Convergence Proofs. *arXiv e-prints*, page arXiv:1812.08883, December 2018.
- [45] W. Chen, H. Sun, X. Zhang, and D. Korošak. Anomalous Diffusion Modeling By Fractal and Fractional Derivatives. *Computers & Mathematics with Applications*, 59(5):1754–1758, 2010.
- [46] R. Kleinberg, Y. Li, and Y. Yuan. An Alternative View: When Does SGD Escape Local Minima? *arXiv preprint arXiv:1802.06175*, 2018.
- [47] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the Loss Landscape of Neural Nets. Visualizing the Loss Landscape of Neural Nets, pages 6391–6401, 2018.
- [48] K. Turner and J. Ghosh. Bayes Error Rate Estimation Using Classifier Ensembles. *International Journal of Smart Engineering System Design*, 5(2):95–109, 2003.
- [49] J. Bec and K. Khanin. Burgers Turbulence. *Physics Reports*, 447(1-2):1–66, 2007.
- [50] I. C. Christov. On the Numerical Solution of A Variable-Coefficient Burgers Equation Arising in

- Granular Segregation. *arXiv preprint arXiv:1707.00034*, 2017.
- [51] V. Dolgunin. VN Dolgunin and AA Ukolov, Powder Technol. 83, 95 (1995). *Powder Technol.*, 83:95, 1995.
  - [52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TENSORFLOW: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.
  - [53] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic Differentiation in PyTorch. Automatic Differentiation in PyTorch, 2017.
  - [54] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv preprint arXiv:1512.01274*, 2015.

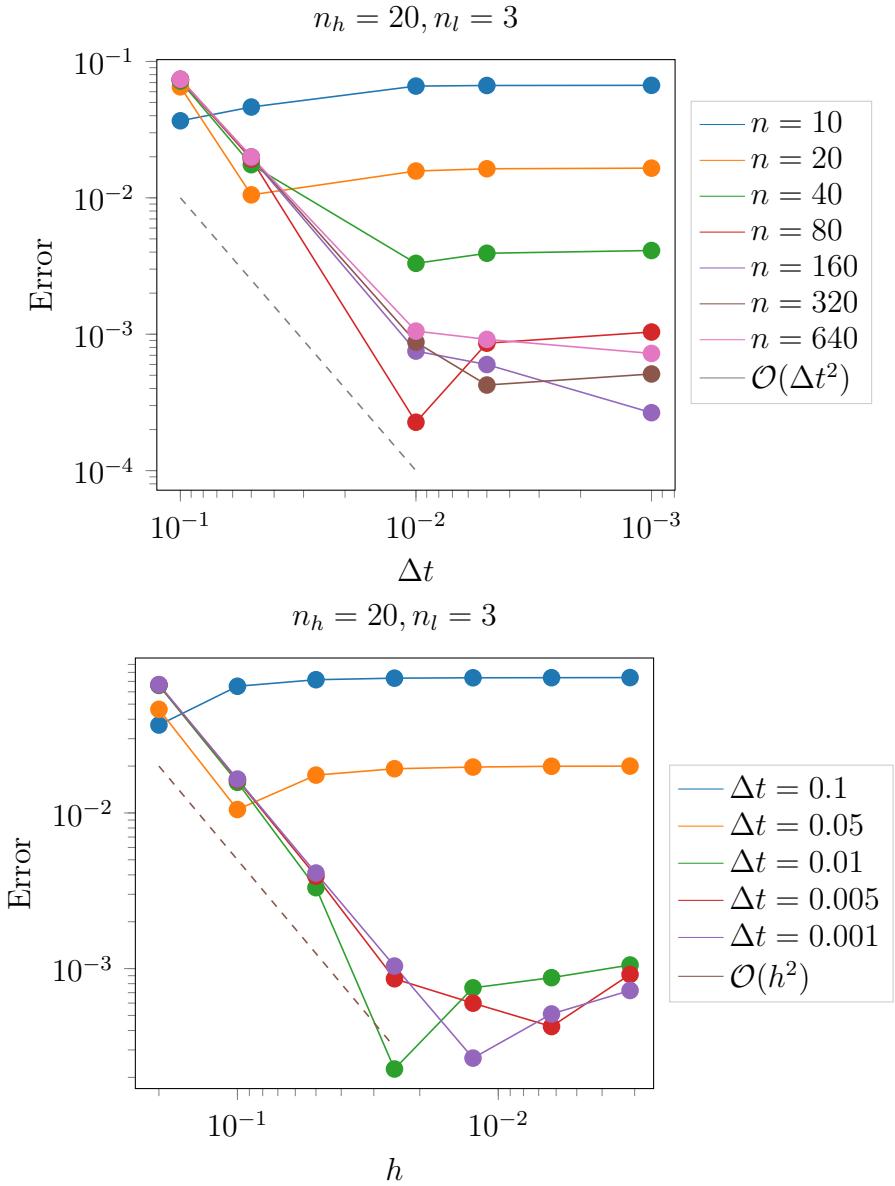
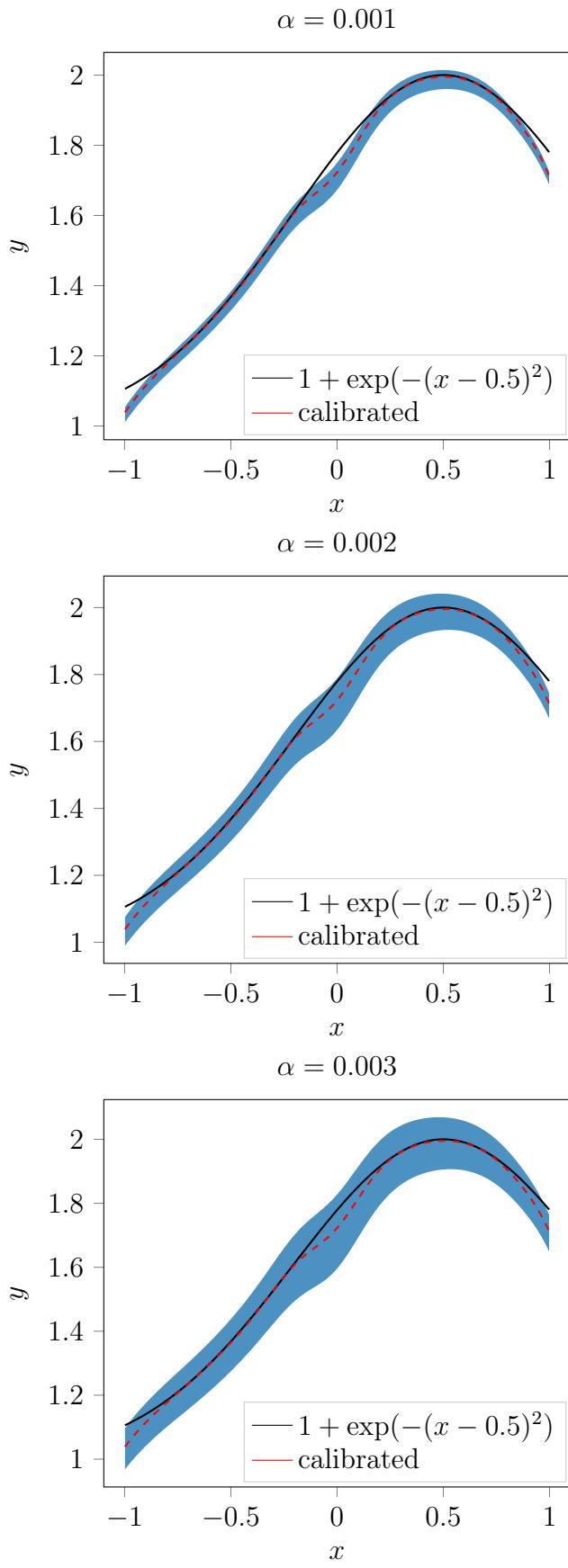


Figure 3: We take the snapshots at  $t = 0.1$  and  $t = t + \Delta t$ , with various  $\Delta t$  and  $h$ , and  $n = \frac{2}{h}$  in the plots. We see a clear pattern exactly predicted in Theorem 6: a second order convergence with respect to  $\Delta t$  and  $h$ , and at some point, the error ceases to decrease and remains at a constant level.



January 24, 2019

Figure 4: Sensitivity analysis for eq. (68). The physical quantity we are interested is the conductivity function value at  $x = 0$ , i.e., the prediction for  $c(0)$ . We have used three different step size  $\alpha = 0.001, 0.002$  and  $0.003$ . The blue region is the sensitivity region defined by eq. (65).

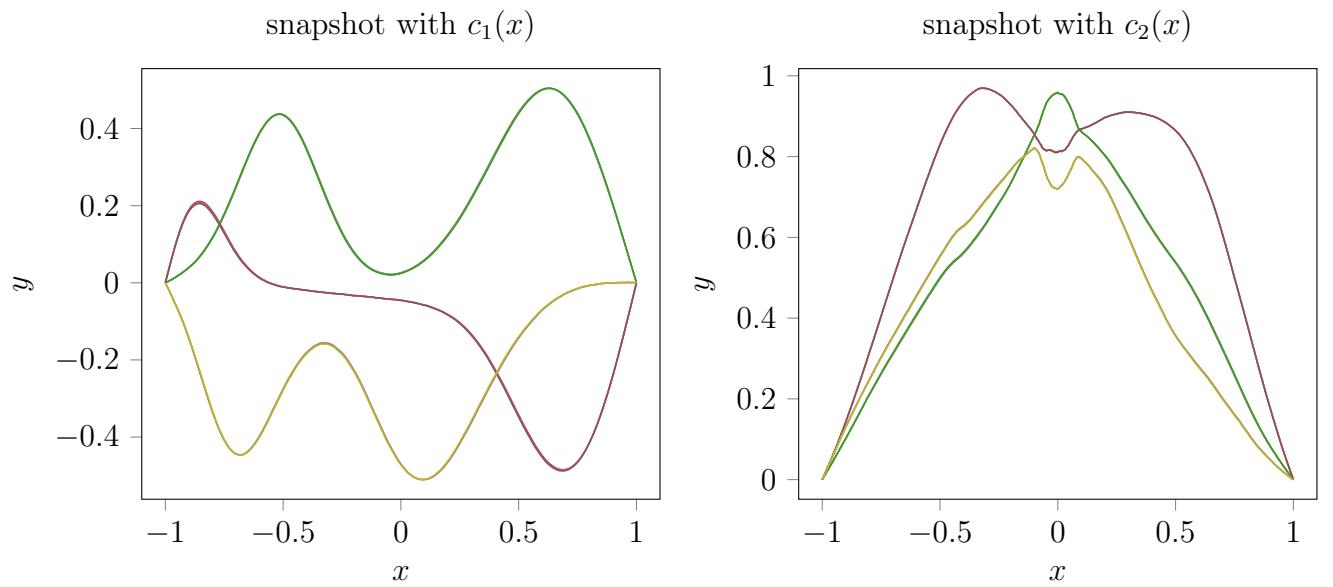


Figure 5: Examples of the snapshots for the velocity fields  $c_1(x)$  and  $c_2(x)$ . Note that since  $10\Delta t$  is quite small, the nearby sequential snapshots almost overlap.

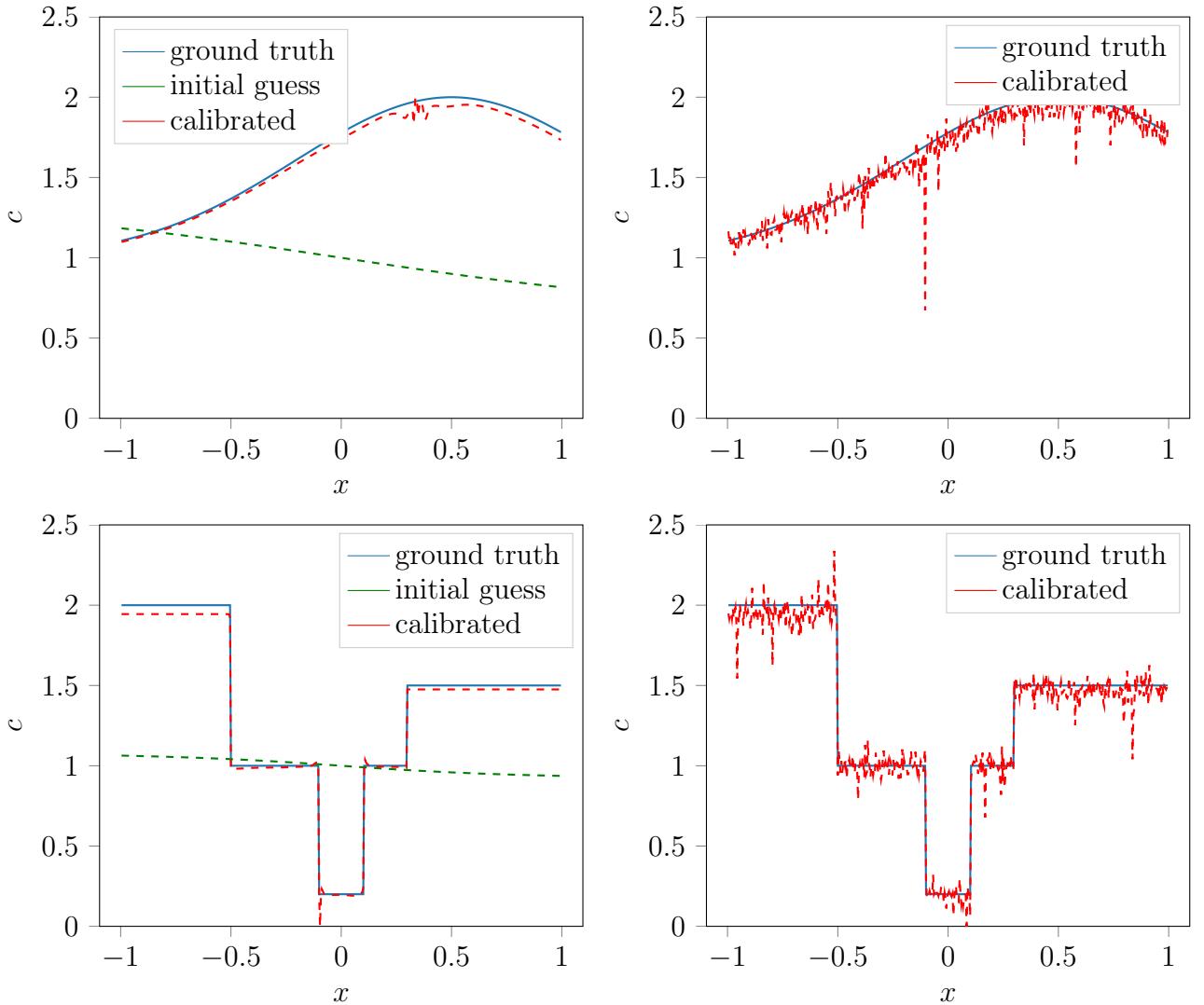


Figure 6: The left columns show the calibrating results using the neural networks while the right columns correspond to least square methods. We can see that the least square methods are vulnerable to noise and do not yield a solution as smooth as the neural network methods. Especially for  $c_2(x)$ , the neural network captures the transition between different velocity levels and yields a constant value at each level. The results are quite remarkable since it is generally challenging to capture those discontinuities in inverse problems. The fit is not perfect though because we have added noise to the dataset.

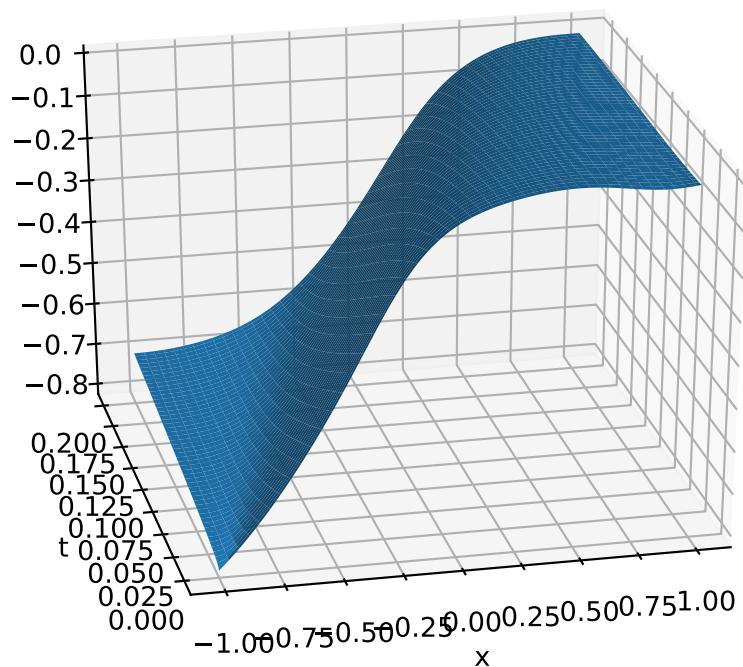


Figure 7: Solution profile for eq. (74) with initial condition eq. (76) and the percolation function eq. (77).

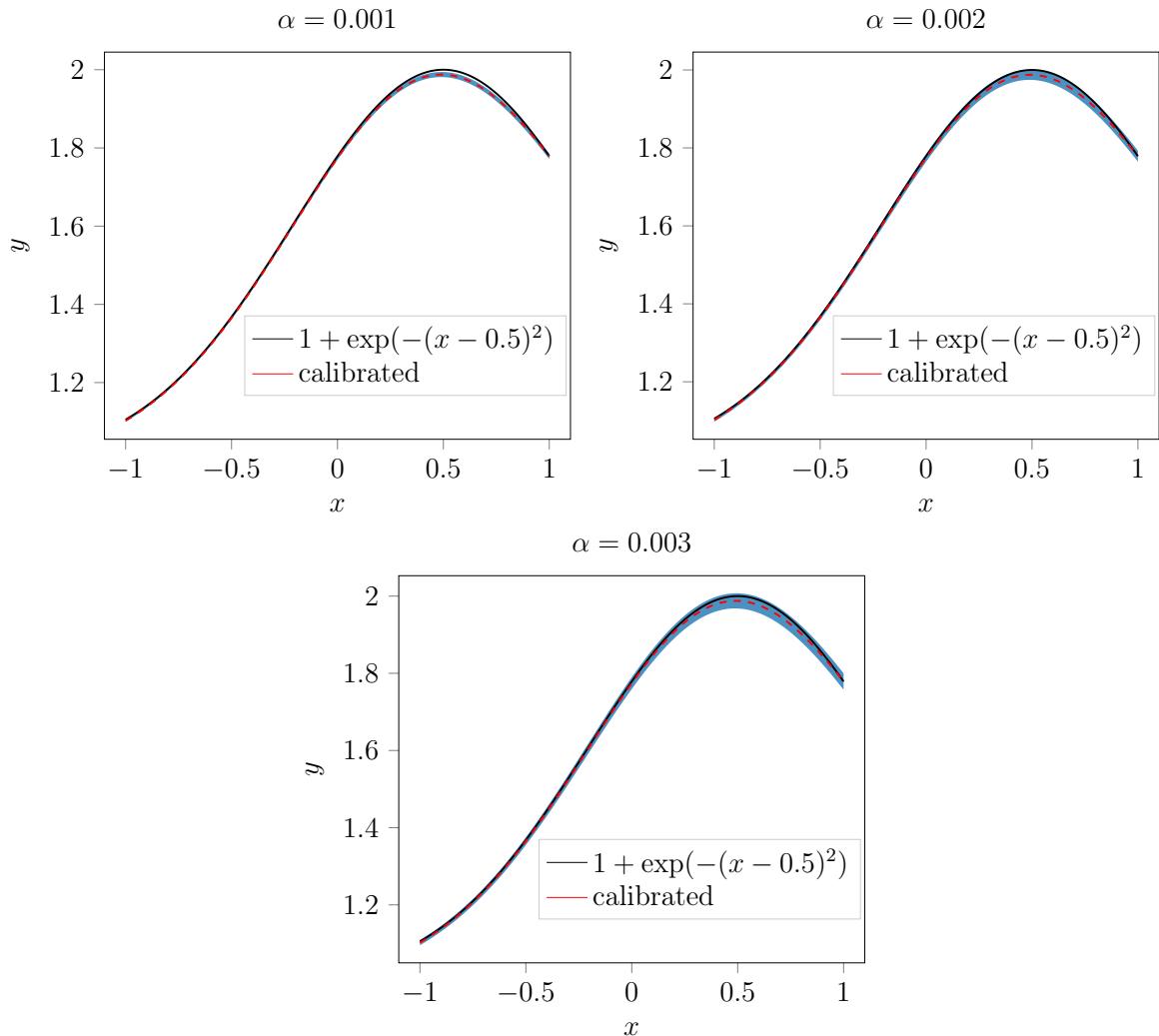


Figure 8: The calibrated results and the corresponding sensitivity regions with different step sizes  $\alpha$ . The stripe gradually grows and finally encloses the exact  $f(x)$ . We also see that the values far-away from the maxima are less sensitive to the parameters.

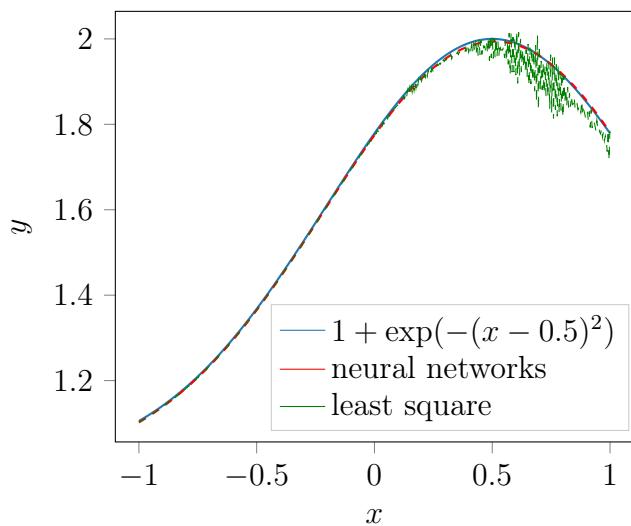


Figure 9: The calibrated results using least square methods with the same settings. We can see that this method is more susceptible to random noise and has stability issues compared to the neural network approach.