Deep Neural Network Approach to Forward-Inverse Problems

Hyeontae Jo[a,1], Hwijae Son[a,1], Hyung Ju Hwang[a,*], Eunheui Kim[b]

[a] *Department of Mathematics, Pohang University of Science and Technology, South Korea*
[b] *Department of Mathematics and Statistics, California State University Long Beach, US*

Abstract

In this paper, we construct approximated solutions of Differential Equations (DEs) using the Deep Neural Network (DNN). Furthermore, we present an architecture that includes the process of finding model parameters through experimental data, the inverse problem. That is, we provide a unified framework of DNN architecture that approximates an analytic solution and its model parameters simultaneously. The architecture consists of a feed forward DNN with non-linear activation functions depending on DEs, automatic differentiation [2], reduction of order, and gradient based optimization method. We also prove theoretically that the proposed DNN solution converges to an analytic solution in a suitable function space for fundamental DEs. Finally, we perform numerical experiments to validate the robustness of our simplistic DNN architecture for 1D transport equation, 2D heat equation, 2D wave equation, and the Lotka-Volterra system.

Keywords: Differential equation, Approximated solution, Inverse problem, Artificial neural networks

1. Introduction

This paper marks the first step toward a comprehensive study on deep learning architectures to solve forward-inverse problems for differential equations. Recent advances in deep learning show its capability to handle various types of model problems in many disciplines. In particular, deep learning techniques have been applied to understand data augmented differential equations. While most of such studies have been centered around heuristics and modeling prospectives, to the best of our knowledge, there is little to no theoretical analysis to confirm whether the deep learning architectures give rise to the correct solutions to the governing differential equations. An overreaching goal of this paper is to provide a comprehensive analysis of Deep Neural Networks (DNNs) to solve data-driven differential equations. This paper reports a novel architecture leveraging recent progress in deep learning techniques that solves forward-inverse problems for differential equations. The paper further includes a convergence analysis and its experimental results for our deep learning architecture.

Forward-inverse problems (or inverse problems in short) for differential equations in this paper are related to data augmented differential equations. Namely, we consider equations of states for physical systems as governing differential equations and model parameters such as advection rates, reaction diffusion coefficients, for example, that need to be fitted by the given data. Hence numerical methods solving forward-inverse problems typically become constraint problems that require an ensemble of two steps, (1) solve the state equations, which is called the forward problems and (2) find the correct model parameters that fit the given data set. Inverse problem is an actively studied field and many numerical algorithms for the inverse problems are robust enough to handle sufficiently large data sets, see for example [1, 5, 16, 17, 19, 25, 26] references therein. However, such algorithms can be computationally expensive and they may be too sophisticated for non-experts in inverse problems to implement them. This calls for simplistic methods that unify two steps in solving forward-inverse problems.

The contributions of this paper are three-fold. First, the DNN architecture presented in this paper highlights its simplistic approach to handling forward-inverse problems simultaneously. Second, a rigorous analysis

*Corresponding author
Email addresses: jht0116@postech.ac.kr (Hyeontae Jo), son9409@postech.ac.kr (Hwijae Son), hjhwang@postech.ac.kr (Hyung Ju Hwang), EunHeui.Kim@csulb.edu (Eunheui Kim)

[1] Both authors contributed equally to this work

of the convergence of the DNN solutions to the actual solutions for the governing problems is provided. Third, numerical experiments validate the robustness of our simple architecture.

The paper comprises the followings. A short overview of related works on data-driven differential equations and the problem formulation are presented in the rest of Section 1. The methodology including the DNN architecture and loss function is described in Section 2. Theoretical results are given in Section 3. Section 4 is devoted to the experiments done for the problem. In Section 5, we conclude the paper.

1.1. Background

There are many works to utilize an Artificial Neural Network (ANN) to solve Differential Equations (DEs) in place of using well established numerical methods such as finite difference, finite element, and finite volume methods. Those finite schemes are heavily depending on mesh-grid points, and they may become a hindrance when the state equations reside in a domain with complex geometry. As such, a mesh-free approximation using clever constructions of basis functions has been introduced, see for example [10, 24]) and references therein. The concept of using ANN to solve DEs can be related to mesh-free approximations as one can consider Multi-Layer Perceptrons (MLPs) as an approximation to solutions to DEs.

The ANN structure was first introduced in [20]. Several studies follow to identify a class of functions that can be recognized by ANNs. The following is a brief overview and a few highlights of such studies. Cybenko [8] established sufficient conditions for which a continuous function can be approximated by finite linear combinations of single hidden layer neural networks with the same univariate function. About the same time, Hornik *et al* [11] showed that measurable functions can be approximated by multi-layer feedforward networks with a monotone sigmoid function as the authors called them "universal approximators". Cotter [6] extends the result of [11] to new architectures, and later Li [18] showed that a MLP with one hidden layer can approximate a function and its higher partial derivatives on a compact set.

The concept of using ANN to solve DEs are not new, and it has gained much attention recently. Some of the highlights include the following. Lagaris *et al* [14] studied to solve DEs and PDEs using an ANN with architecture including 1 single layer and 10 units and they next extended in [15] their results to a domain with complex geometry. Jianyu *et al* [12] used ANN with a radial basis function as an activation function for Poisson equations. More recently, Berg *et al* in [4] used a DNN to solve steady (time-independent) problems for a domain with complex geometry in one and two space dimensions, and later in [3] they studied DNN architectures to solve augmented Poisson equations (inverse problems) including three space dimensions.

The recent work by Raissi *et al* [22] can be perhaps closely related to our work in the sense that their DNN architectures (they called "continuous time models") resemble ours. We note however the aim of this paper is to establish a comprehensive study that includes a rigorous convergence analysis of DNNs to solve forward-inverse problems for DEs and PDEs. Since our convergence result is for linear equations at this point, we present our experiments for the well known linear equations as well. While our experiments cover simpler equations than those studied in [22], as their focuses were on architectures for "data-efficient and physics informed learning machines", we hope that our result can enhance the experiments shown in [22]. We believe our first comprehensive results can shed lights onto further studies toward more complex and complicated systems using machine learning architectures.

1.2. Problem formulation

We consider the equations of states as the following time dependent initial boundary value problems:

$$L_p u = 0,\ t \in (0,T],\ x \in \Omega, \tag{1.1}$$

$$Iu = f,\ x \in \Omega, \tag{1.2}$$

$$Bu = g,\ t \in (0,T],\ x \in \partial\Omega, \tag{1.3}$$

where L_p is a differential operator, p is a set of model parameters, $\Omega \subset \mathbb{R}^d$ is a bounded domain (for the position of the state u), $\partial\Omega$ is a boundary of Ω, $f(x)$ is an initial distribution, I is an initial operator, B is a boundary operator, and $g(t,x)$ is a boundary data. Next the governing equation is equipped with a set of observation data, which may be provided from actual experiments, as following:

$$D = \{(t_i, x_j, u_{ij}) | i = 1,2,...,n,\ j = 1,2,...,m\}, \tag{1.4}$$

where u_{ij} denotes the output value (observation from the experiment) at position $x_j \in \Omega$ and time $0 < t_i \leq T$ with the final time T. n and m refer the numbers of time and spatial observations respectively. Since our experiments cover ODEs and PDEs with one and two spatial dimensions, x_j dependencies and the index j for x_j will be adjusted for each example.

We apply DNNs to solve governing equations (1.1)-(1.3) and the observation data (1.4). Our loss function includes the governing equation, observations, initial and boundary conditions. The loss function is minimized by using the standard gradient descent method. The results show that our DNN architecture can handle much lesser numbers of observations compared to known numerical solvers such as finite difference, element and volume methods. The results presented in this paper demonstrate that our DNN architecture is perhaps the most simplistic way to solve forward-inverse problems, yet robust enough to handle many different cases. Furthermore, we establish the convergence result of forward-inverse problems for parabolic or hyperbolic linear PDEs with Dirichlet boundary conditions. Specifically we show that the sequence of DNN solutions converges to an analytic solution in $L^\infty(0, T; H_0^1(\Omega))$ sense.

2. Methodology

This section provides our DNN architecture and the mathematical formulation of the training procedure.

2.1. DNN architecture

The DNN architecture can be written as a repeated composition of some affine transformations and some nonlinear activation functions. We denote by u_N the DNN solution and assume that the architecture consists of $L+1$ layers. The first layer takes (t, x) as an input, and the last layer gives the output value $u_N(t, x)$. The $L-1$ layers between the first and the last layers are called hidden layers. We have used common nonlinear activation functions such as sigmoid, rectified linear units, and hyperbolic tangents through the DNN. Each neuron in the DNN contains a bias except for the input neurons. In two consecutive layers, the connections between neurons are written as weight matrices. Relations between $(l-1)^{th}$ and l^{th} layers are defined by :

$$z_j^l = \sum_k^{N_{l-1}} w_{jk}^l \sigma_{l-1}(z_k^{l-1}) + b_j^l, \tag{2.1}$$

where

- z_k^{l-1}: k^{th} neuron in $(l-1)^{th}$ layer
- N_{l-1}: the number of neurons in $(l-1)^{th}$ layer
- b_j^l: the bias of j^{th} neuron in l^{th} layer
- w_{jk}^l: the weights between k^{th} neuron in $(l-1)^{th}$ layer and j^{th} neuron in l^{th} layer
- σ_{l-1} the activation function in $(l-1)^{th}$ layer

For convenience, we denote $z^0 = (z_1^0, ..., z_{N_0}^0)$ as (t, x) and $z^L = (z_1^L, ..., z_{N_L}^L)$ as $u_N(t, x)$, respectively. The values N_{l-1}, L, and the form of σ_{l-1} should be chosen before training. In the training procedure, we have to calculate the optimal weights and biases w_{jk}^l, b_j^l which minimize a suitable loss function.

2.2. Loss function

The training procedure of the DNN is equivalent to the optimization problem of the loss function with respect to the DNN parameters. We denote the DNN solution by $u_N(t, x) = u_N(t, x; w, b)$, where (w, b) are the set of weights and biases defined in (2.1). Denote the number of grid points of time, spatial variables, initial and boundary domains by N_t, N_x, I, B_t, B_x respectively. Now we define the loss function using (1.1),

$$Loss_{GE}(w, b, p) = \int_{[0,T]} \int_\Omega (L_p u_N(t, x; w, b))^2 dx dt \approx \sum_{i,j=1}^{N_t, N_x} (L_p u_N(t_i, x_j; w, b))^2, \tag{2.2}$$

where the last approximated sum is obtained by sampling a grid point $\{(t_i, x_j)|t_i \in [0,T], x_j \in \Omega, \text{ for } i = 1, ..., N_t, j = 1, ..., N_x\}$. Note that the reason that we define the above loss function is to find the optimal weights (w, b) which minimize $Loss_{GE}$. However, (2.2) is not sufficient because it excludes information about initial and boundary conditions. Therefore we define two loss functions from (1.2)-(1.3)

$$Loss_{IC}(w,b) = \int_{\Omega} (Iu_N(0, x; w, b) - f(x))^2 dx \approx \sum_{i=1}^{I} (u_N(0, x_i; w, b) - f(x_i))^2, \tag{2.3}$$

$$Loss_{BC}(w,b) = \int_{[0,T]} \int_{\partial\Omega} (Bu_N(t, x; w, b) - g(t, x))^2 dS dt \approx \sum_{i,j=1}^{B_t, B_x} (u_N(t_i, x_j; w, b) - g(t_i, x_j)|)^2. \tag{2.4}$$

Combining all loss functions (2.2)-(2.4) is still not enough because the solution of DEs (1.1)-(1.3) could differ depending on the choice of the equation parameter p. Due to this reason, we should make one additional loss function to calibrate p using the observed data (1.4).

$$Loss_{Obs}(w,b) = \sum_{(t_i,x_j)\in D} |u_{ij} - u_N(t_i, x_j; w, b)|^2. \tag{2.5}$$

Finally, we define the forward loss and total loss as a summation of all three, and four loss functions respectively. For the forward loss, the model parameter p is considered to be fixed.

$$Loss_{Forward}(w,b) = Loss_{GE}(w,b) + Loss_{IC}(w,b) + Loss_{BC}(w,b), \tag{2.6}$$

$$Loss_{Total}(w,b,p) = Loss_{GE}(w,b,p) + Loss_{IC}(w,b) + Loss_{BC}(w,b) + Loss_{Obs}(w,b). \tag{2.7}$$

Algorithm 1 Training

procedure TRAIN(number of epochs)
  for number of epochs **do**
    sample minibatch of m samples $z^1, z^2, ..., z^m$ from uniform distribution $p_\Omega(z)$
    sample minibatch of m samples $z_I^1, z_I^2, ..., z_I^m$ from uniform distribution $p_{\{0\}\times\Omega}(z)$
    sample minibatch of m samples $z_B^1, z_B^2, ..., z_B^m$ from uniform distribution $p_{\partial\Omega}(z)$
    sample k observation points $z_O^1, z_O^2, ..., z_O^k$
    Find the true value $u_j = u_p(z_O^j)$ for $j = 1, 2, ..., k$
    Update the neural network by descending its stochastic gradient :

$$\nabla_{w,b}[\frac{1}{m}\sum_{i=1}^{m}[L_p(u_N)(z^i)^2 + (u_N(z_I^i) - f(z_I^i))^2 + (u_N(z_B^i) - g(z_B^i))^2] + \frac{1}{k}\sum_{j=1}^{k}(u_N(z_O^j) - u_j)^2]$$

  end for
end procedure

3. Theoretical result

This section provides a theoretical proof that there exists a sequence of weights such that the corresponding sequence of DNN solutions converges to an analytic solution on any compact subset of the domain. We focus on the DEs (1.1)-(1.3) where the existence and the uniqueness of solutions are guaranteed. We establish the result in two steps. We first show that a sequence of DNN solutions converges to an analytic solution for the corresponding model parameters, called the forward problem. We next show that both the estimated parameter and the DNN solutions converge to the model parameter and the analytic solution simultaneously, called the inverse problem.

3.1. Forward problem

For the forward problem, we fix the model parameter p and denote the analytic solution to (1.1)-(1.3) by u_p. We also denote the DNN solution in (2.1) by u_N. In u_N, activation functions σ are any non-polynomial functions in $C^k(\mathbb{R}^n)$.

Next we quote the following Definition 3.1 and Theorem 3.1 from [18]

Definition 3.1. Let $I_n = [0,1]^n$ be the unit interval on $\mathbb{R}^n$, k be a non-negative integer. Then we say a function f is contained in $\widehat{C}^k(I_n)$ if $f \in C^k(\Omega)$ for some open set U containing I_n

Theorem 3.1. (Li, Theorem 2.1 in [18]) Let $f \in \widehat{C}^k(I_n)$. Then, given $\varepsilon > 0$, we can find parameters of a neural network u_N, defined in (2.1) with $L = 1$, so that

$$\|D^{\underline{k}} f - D^{\underline{k}} u_N\|_{L^\infty(I_n)} < \varepsilon$$

holds for any multi-index $\underline{k} = (k_1, k_2, ..., k_n)$, $|k_1| + |k_2| + ... + |k_n| \le k$, and k_i's are non-negative integers.

Remark 3.1. Since the above result can be generalized to multi-layer architectures (for example, [11]) and to a general compact set K instead of I_n, we may assume that the architecture contains only one hidden layer ($L = 1$) and the domain Ω is I_n ($\Omega = I_n$).

Theorem 3.2. For a non-negative integer k, assume that the highest order of linear operator (1.1) is k and $u_p \in \widehat{C}^k(I_n)$, and the activation function $\sigma(x)$ and its (k-th order) derivatives are continuous and discriminatory. Then, there exists $\{m_j, w_j, b_j\}_{j=1}^{\infty}$ such that a sequence of the DNN solutions with m_j nodes, denoted by $\{u_j(w_j, b_j)\}_{j=1}^{\infty}$ satisties

$$Loss_{Forward}(w_j, b_j) \to 0 \text{ as } j \to \infty \tag{3.1}$$

Proof. Let $\epsilon > 0$ be given. By Theorem 3.1, there exists a neural network $u_j(x) = \sum_{i=1}^{m_j} w_i^1 \sigma(w_i^2 x + b_i)$ such that $\|D^{\underline{k}} u_p - D^{\underline{k}} u_j\|_\infty < \epsilon$, where $\underline{k}$ is a non-negative multi-index up to differentiability of u_p. By integrating $|L_p u_j|^2$, $|Iu_j - f|^2$, $|Bu_j - g|^2$ over $[0,T] \times \Omega$, Ω, $[0,T] \times \partial\Omega$ respectively, we obtain the desired result. □

Remark 3.2. The assumption $u_p \in \widehat{C}^k(I_n)$ in Theorem 3.2 is a strong condition. Since we can also approximate it by a sequence of compactly supported smooth functions in I_n, we can extend the assumption to a general Sobolev space.

The Theorem 3.2 states that we can always find parameters of a DNN architecture which can reduce $Loss_{Forward}$ if the DE has a smooth analytic solution. However, since we can not directly use information of an analytic solution, we next show that the DNN architecture equipped with parameters which minimize $Loss_{Forward}$ converges to an analytic solution in Theorem 3.3.

Theorem 3.3. Let $L_p = \partial_t + L$ in (1.1) be a second order parabolic operator and $Bu = 0$ in (1.3) be a Dirichlet boundary condition. Also we define the DNN solution $u_j = u_j(t, x; w_j, b_j)$ with m_j nodes and the corresponding loss $Loss_{Forward}(w, b)$. Then, $Loss_{Forward}(w, b) \to 0$ implies

$$u_j(t, x; w_j, b_j) \to u_p \text{ in } L^\infty([0,T]; H_0^1(\Omega)), \tag{3.2}$$

where u_p is a solution to (1.1)-(1.3)

Proof. First we assume that the activation function satisfies the Dirichlet boundary condition by replacing it with $b(x)\sigma(t,x)$, where $b(x)$ is an arbitrary smooth function that satisfies $b = 0$ on $\partial\Omega$. By evaluating $u_p - u_j$ in (1.1)-(1.3), we have the following

$$\begin{aligned} \partial_t(u_p - u_j) + L(u_p - u_j) &= \varepsilon_m(t,x),\ t \in (0,T],\ x \in \Omega, \\ I(u_p - u_j) &= \eta_m(x),\ x \in \Omega, \\ B(u_p - u_j) &= 0,\ t \in (0,T],\ x \in \partial\Omega \end{aligned}$$

Then the energy estimates for the second order parabolic equation (see Theorem 5, Chapter 7 in [9]) are applied to obtain that

$$\operatorname*{esssup}_{0\le t\le T} \|u_p - u_j(\cdot,t)\|_{H_0^1(\Omega)} + \|u_p - u_j\|_{L^2([0,T];H_0^1(\Omega))} + \|\partial_t (u_p - u_j)\|_{L^2([0,T];L^2(\Omega))} \tag{3.3}$$
$$\le C\left(\|\varepsilon_m\|_{L^2([0,T];L^2(\Omega))} + \|\eta_m\|_{H_0^1(\Omega)}\right),$$

where the constant C in (3.3) depends only on Ω, T and the coefficients of L. Note that the right hand side in (3.3) is equivalent to $Loss_{Forward}(w,b)$. This shows that the sequence of DNN solutions converges to the analytic solution when $Loss_{Forward}(w,b) \to 0$. □

Remark 3.3. The convergence result also holds when $L_p = \partial_{tt} + L$ is a second order hyperbolic operator.

3.2. Inverse problem

Definition 3.2. Let P be the set of all possible model parameters. Define $S := \{u_p \mid p \in P\}$ be the set of solutions corresponding to each model parameter $p \in P$.

Definition 3.3. We say the observation set D_p is *clear* if for any $u_p, u_q \in S$, $u_p|_D = u_q|_D$ if and only if $p = q$

Theorem 3.4. Let $Loss_{total}(w,b,p)$ be the total loss defined in (2.7) and let the observation set D_p with $p \in P$ be given and clear. We assume that for given $\{\epsilon_j\}_{j=1}^{\infty}$ with $\epsilon_j \to 0$, there exists (m_j, w_j, b_j, p_j) such that $Loss_{Total}(w_j, b_j, p_j) < \epsilon_j$, and the parameter set $\{p_j\}$ is contained in P, then

$$u_j(w_j, b_j) \to u_p \text{ a.e. and } p_j \to p \text{ as } j \to \infty \tag{3.4}$$

Proof. We first divide the total loss into $Loss_{Total} = Loss_{Forward,p_j} + Loss_{Obs,p_j}$. For m fixed, we set $u_{N_{m,k}}(w_{m,k}, b_{m,k})$ as a DNN solution with $k \ge m$ nodes, where $(w_{m,k}, b_{m,k})$ is a minimizer of $Loss_{Forward}$ defined in Definition 3.1. Then, Theorem 3.3 implies $u_{N_{m,k}} \to u_{p_m}$ as $k \to \infty$. Also, $Loss_{Obs,p_m} \to 0$ implies $p_m \to p$ by definition of D_p

□

4. Experiments

In this section, we provide experimental results based on several differential equations including 1D transport equation, 2D heat equation, 2D wave equation, and the Lotka-Volterra system. For each equation, we have calculated an analytic (if possible) or numerical solution with fixed model parameters in order to generate a small amount of true solution points which will be regarded as the observation points. We apply our DNN model to find an approximated solution and the optimal equation parameter at the same time. In this step, we have used a neural network with variable depth and width, and ReLU activations. We used the Adam optimizer [13, 23] with $(\beta 1, \beta 2) = (0.9, 0.999)$ in order to find the minimizer w, b, and p defined in (2.7). Also, for higher order derivatives in (1.1) we have applied the reduction of order technique to express it as a system of differential equations. This step dramatically reduces the computational cost for calculating (2.2). For example, the second-order PDE $u_{xx} = f$ can be replaced by $v_x = f$ together with the equation $v = u_x$. That is, we derive two first-order PDEs $v_x = f, u_x = v$ from one second-order PDE $u_{xx} = f$, then the output layer of DNN should be changed into (u, v). This method is applied to 2D Heat and Wave equations. Also, different types of activation functions are used depending on the behavioral characteristics of the governing equations. Finally, we provide two differences between 1) actual and model output values and 2) actual and calculated model parameters. Observation points were calculated from analytic (transport), series (heat, wave), numerical (Lotka-Volterra) solution and sampled randomly among them.

In the rest of this section, we present the experimental results. For each figure, top left, top right figures show our neural network solution and the analytic solution respectively. Bottom left shows the absolute error between the neural network solution and the analytic solution. Bottom right figure shows the convergence of estimated parameters to the real parameters. We have implemented our method by using Pytorch [21], which is one of the most famous machine learning library. We first present the detailed experimental settings. Table 1 and 2 show the summarized information of the number of grid points and DNN architectures respectively.

Table 1: Information of grid and observation points

	Data Generation		
	Grid Range	Number of Grid Points	Number of Observations
1D Transport	$(t, x) \in [0, 1] \times [0, 1]$	17×100	17
2D Heat	$(t, x, y) \in [0, 1] \times [0, 1] \times [0, 1]$	$100 \times 100 \times 100$	13
2D Wave	$(t, x, y) \in [0, 1] \times [0, 1] \times [0, 1]$	$100 \times 100 \times 100$	61
Lotka-Volterra	$t \in [0, 100]$	20,000	40

Table 2: Neural network architecture

	Neural Network Architecture		
	Fully Connected Layers	Activation Functions	Learning Rate
1D Transport	2(input)-128-256-128-1(output)	ReLU	10^{-5}
2D Heat	3(input)-128-128-1(output)	Sin, Sigmoid	10^{-5}
2D Wave	3(input)-128-256-128-1(output)	Sin, Tanh	10^{-5}
Lotka-Volterra	1(input)-64-64-2(output)	Sin	10^{-4}

4.1. 1D Transport equation

1D transport equation consists of

$$\partial_t u + a\partial_x u = 0, \tag{4.1}$$
$$u(0, x) = \begin{cases} \sin^4 0.25 * \pi(x - 0.1) & \text{if } 0.1 \le x \le 0.5 \\ 0 & \text{otherwise} \end{cases},$$

where $a = \pi/10 (\simeq 0.314...)$. We have generated the observations from the analytic solution, by method of characteristics, of (4.1).

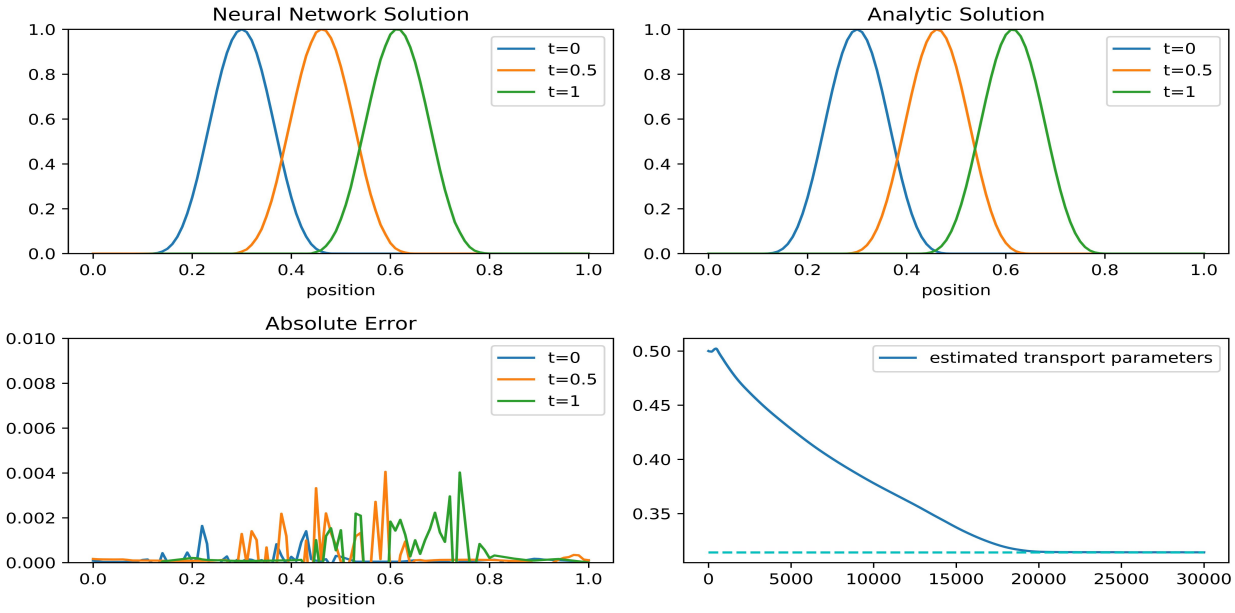

Figure 1: Experimental result for 1D transport equation

4.2. 2D Heat equation

$$\begin{aligned} &\partial_t u = a^2 \left(\partial_{xx} u + \partial_{yy} u\right), \\ &u(t,0,y) = u(t,1,y) = 0, \\ &u(t,x,0) = u(t,x,1) = 0, \end{aligned} \tag{4.2}$$

where $a = 1$. We have generated the observations from the partial sum of the series solution, by separation of variables, of (4.2)

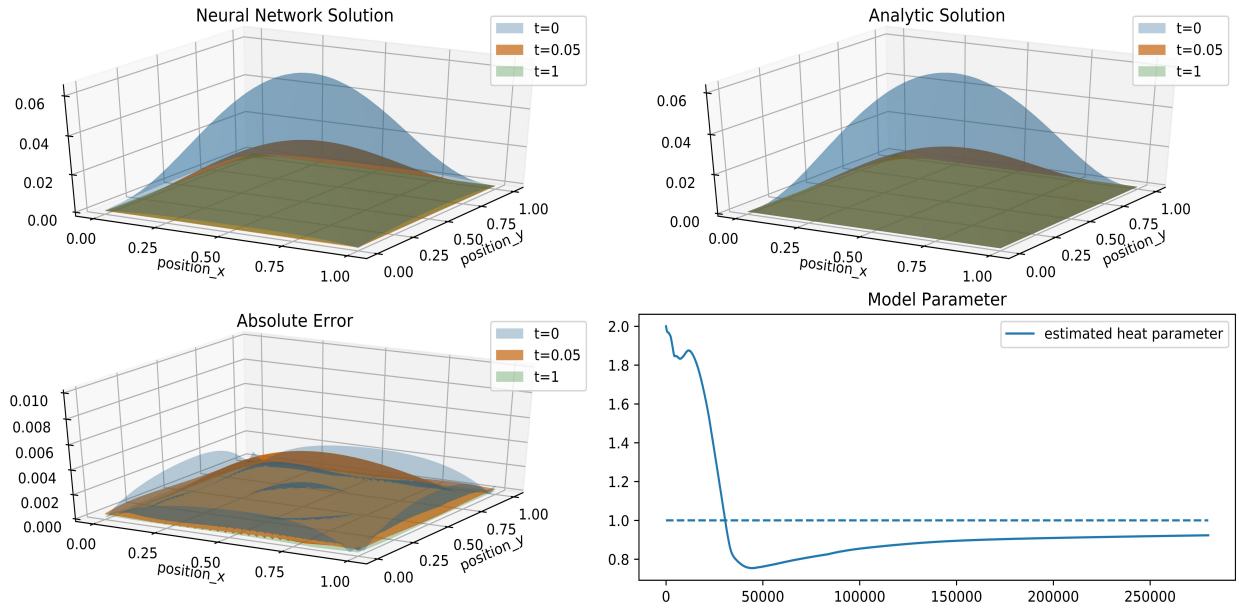

Figure 2: Experimental result for 2D heat equation

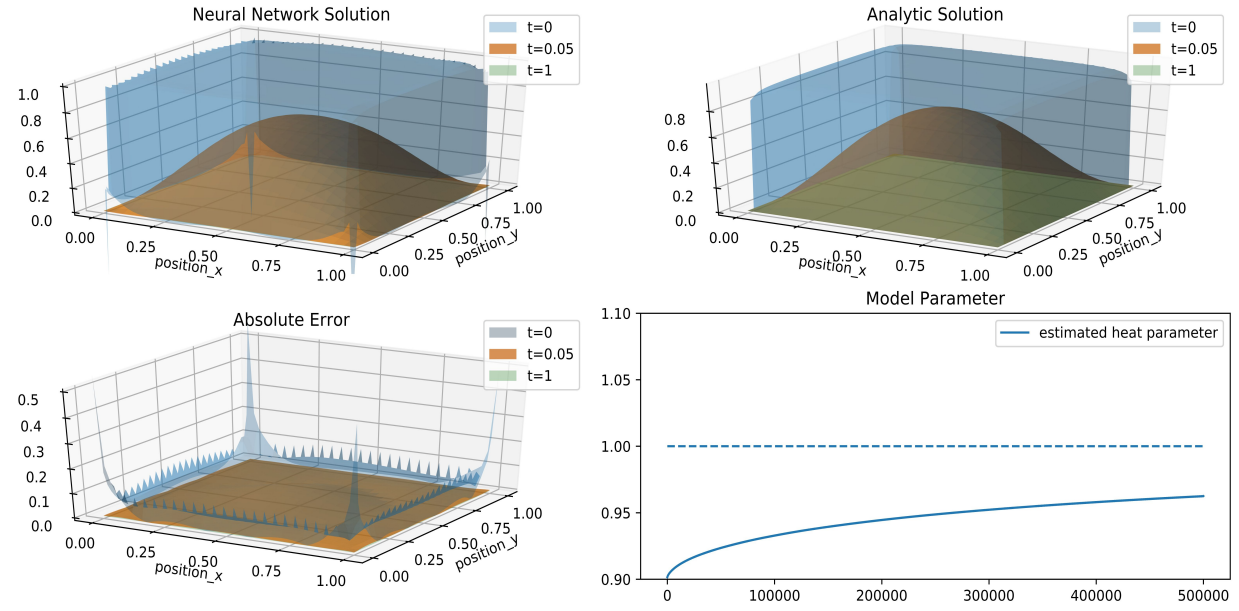

Figure 3: Experimental result for 2D heat equation

4.3. 2D Wave equation

$$
\begin{aligned}
&\partial_{tt} u = a^2 \left(\partial_{xx} u + \partial_{yy} u\right), \\
&u(0, x) = xy(1-x)(1-y), \\
&\partial_t u(0, x, y) = 0, \\
&u(t, 0, y) = u(t, 1, y) = 0, \\
&u(t, x, 0) = u(t, x, 1) = 0,
\end{aligned} \tag{4.3}
$$

where $a = 1$. We have generated the observations from the partial sum of the series solution, by separation of variables, of (4.3)

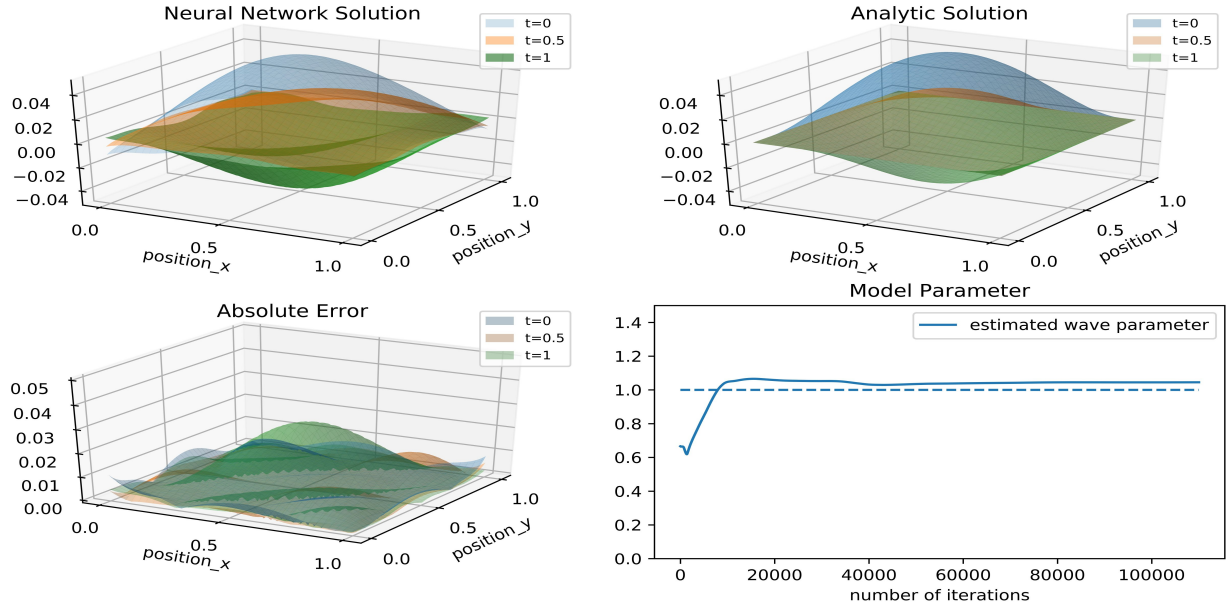

Figure 4: Experimental result for 2D wave equation

4.4. Lotka-Volterra system

$$
\begin{aligned}
&u'(t) = \alpha u - \beta uv, \\
&v'(t) = \delta uv - \gamma v, \\
&u(0) = 1, v(0) = 1,
\end{aligned} \tag{4.4}
$$

where $\alpha = 1, \beta = 0.4, \delta = 0.4, \gamma = 0.1$. We have generated the observations from a numerical solution by the Runge-Kutta method of (4.4). We used the *sin* function as the activation function for Lotka-Volterra system. Considering the periodic nature of the solution, the periodic activation function is a natural choice.

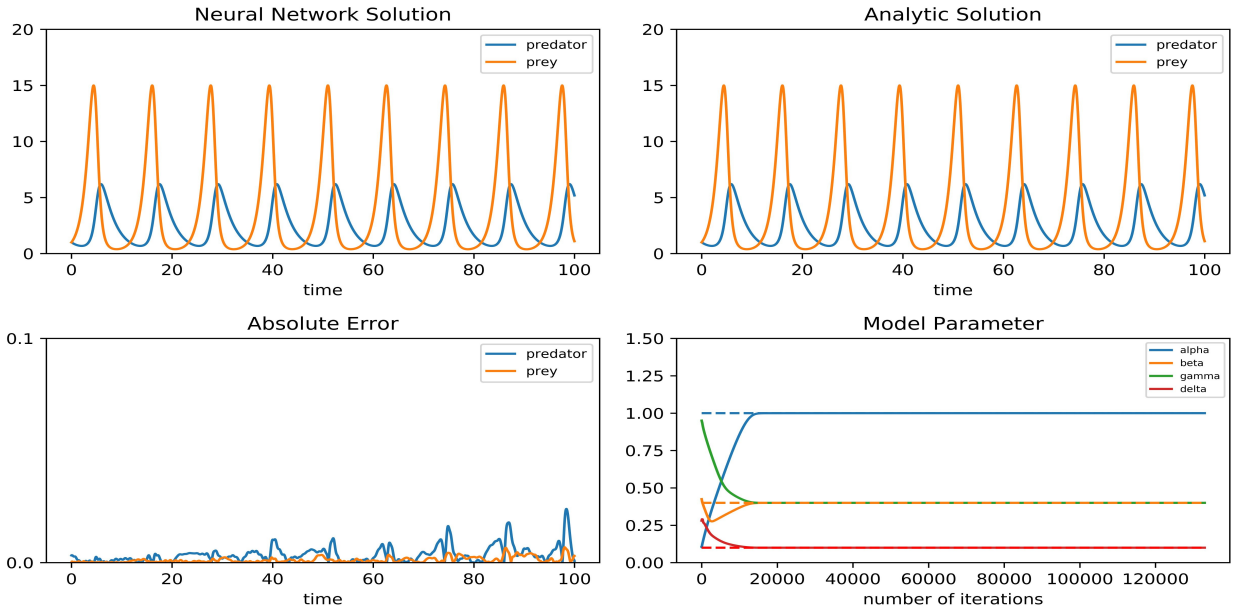

Figure 5: Experimental result for Lotka-Volterra equation

4.5. Stability Condition

In this section, we address the Courant-Friedrichs-Lewy (CFL) condition [7] which is a necessary condition while solving certain partial differential equations numerically. We compare the results of transport equation with three different Courant numbers which all violate the convergence condition. As we can see in figure 6, our method shows the convergence well regradless of CFL condition.

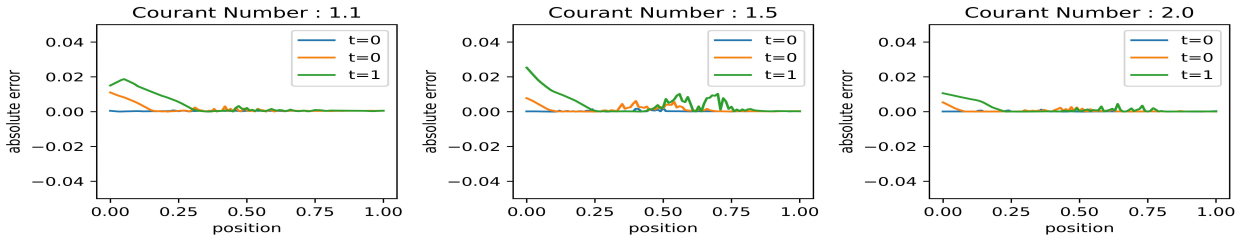

Figure 6: Experimental result for 2D wave equation

Note that our experiments in section 4.2 give nice results while the settings violate the well known stability condition called the Von Neumann stability condition.

5. Conclusion

First we summarize our theoretical results. For linear differential equations, we have shown that the DNN solution can reduce the proposed loss function as much as we want. A key point in the proof is a continuation of the Theorem 2.1 in [18] which states the fact that a linear combination of the dilated activations can approximate the target function and the derivative of such a linear combination can approximate its derivative in L_∞ sense. Next we have proved that the DNN which minimizes the loss converges to an analytic solution for linear parabolic or hyperbolic equations. In this step, we have applied basic energy estimates for each equation. Theoretical results for the inverse problem is also included as a continuation of the forward problem. We provide numerical experiments which show that our method indeed works accurately. We emphasize that our method can easily be implemented without any background knowledge about numerical analysis (for example, stability conditions) but about some libraries for implementing neural networks. Although it performs well

for fundamental DEs, it might be hard to apply it to more complex equations. We have recognized that the error between a NN solution and an analytic/numerical solution is slightly increasing depending on time.

For future directions, we may consider two problems. First, we can use more complicated neural network architectures such as CNN, RNN. Since we have dealt with time dependent PDEs, the combination of CNN and RNN would be a great choice for modelling. Second, the theoretical results for non-linear PDEs should be explored. The convergence results of our work are only applicable to linear PDEs. However, as in the experiment for the Lotka-Volterra system, our method is successful in approximating solutions even for non-linear systems. We hope proper convergence results for non-linear systems to be explored.

6. Acknowledgement

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF-2017R1E1A1A03070105, NRF-2019R1A5A1028324)

References

References

[1] William Arloff, Karl RB Schmitt, and Luke J Venstrom. A parameter estimation method for stiff ordinary differential equations using particle swarm optimisation. *International Journal of Computing Science and Mathematics*, 9(5):419–432, 2018.

[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153), 2018.

[3] Jens Berg and Kaj Nyström. Neural network augmented inverse problems for pdes. *arXiv preprint arXiv:1712.09685*, 2017.

[4] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[5] Guy Chavent. *Nonlinear least squares for inverse problems: theoretical foundations and step-by-step guide for applications*. Springer Science & Business Media, 2010.

[6] Neil E Cotter. The stone-weierstrass theorem and its application to neural networks. *IEEE Transactions on Neural Networks*, 1(4):290–295, 1990.

[7] Richard Courant, Kurt Friedrichs, and Hans Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

[8] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[9] Lawrence C. Evans. *Partial differential equations*. American Mathematical Society, Providence, R.I., 2010.

[10] Gregory E Fasshauer. Solving partial differential equations by collocation with radial basis functions. In *Proceedings of Chamonix*, volume 1997, pages 1–8. Vanderbilt University Press Nashville, TN, 1996.

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[12] Li Jianyu, Luo Siwei, Qi Yingjian, and Huang Yaping. Numerical solution of elliptic partial differential equation using radial basis function neural networks. *Neural Networks*, 16(5-6):729–734, 2003.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[15] Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

[16] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944.

[17] Junhong Li and Xiao Li. Particle swarm optimization iterative identification algorithm and gradient iterative identification algorithm for wiener systems with colored noise. *Complexity*, 2018, 2018.

[18] Xin Li. Simultaneous approximations of multivariate functions and their derivatives by neural networks with one hidden layer. *Neurocomputing*, 12(4):327–343, 1996.

[19] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[20] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[21] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[22] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[23] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[24] Scott A Sarra. Adaptive radial basis function methods for time dependent partial differential equations. *Applied Numerical Mathematics*, 54(1):79–94, 2005.

[25] Panagiotis Tsilifis, Ilias Bilionis, Ioannis Katsounaros, and Nicholas Zabaras. Computationally efficient variational approximations for bayesian inverse problems. *Journal of Verification, Validation and Uncertainty Quantification*, 1(3):031004, 2016.

[26] Fatih Yaman, Valery G Yakhno, and Roland Potthast. A survey on inverse problems for applied sciences. *Mathematical problems in engineering*, 2013, 2013.