Weak Adversarial Networks for High-dimensional Partial Differential Equations

Yaohua Zang * Gang Bao † Xiaojing Ye ‡ Haomin Zhou §

Abstract

Solving general high-dimensional partial differential equations (PDE) is a long-standing challenge in numerical mathematics. In this paper, we propose a novel approach to solve high-dimensional linear and nonlinear PDEs defined on arbitrary domains by leveraging their weak formulations. We convert the problem of finding the weak solution of PDEs into an operator norm minimization problem induced from the weak formulation. The weak solution and the test function in the weak formulation are then parameterized as the primal and adversarial networks respectively, which are alternately updated to approximate the optimal network parameter setting. Our approach, termed as the weak adversarial network (WAN), is fast, stable, and completely mesh-free, which is particularly suitable for high-dimensional PDEs defined on irregular domains where the classical numerical methods based on finite differences and finite elements suffer the issues of slow computation, instability and the curse of dimensionality. We apply our method to a variety of test problems with high-dimensional PDEs to demonstrate its promising performance.

Keywords. High Dimensional PDE, Deep Neural Network, Adversarial Network, Weak Solution

1 Introduction

Solving general high-dimensional partial differential equations (PDEs) has been a long-standing challenge in numerical analysis and computation [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. In this paper, we present a novel method that leverages the form of weak solutions and adversarial networks to compute solutions of PDEs, especially to tackle problems posed in high dimensions. To instantiate the derivation of the proposed method, we first consider the following second-order elliptic PDE with either Dirichlet's or Neumann's boundary conditions on *arbitrary domain* $\Omega \subset \mathbb{R}^d$,

$$\begin{cases} -\sum_{i=1}^d \partial_i(\sum_{j=1}^d a_{ij}\partial_j u) + \sum_{i=1}^d b_i\partial_i u + cu - f = 0, & \text{in } \Omega \\ u(x) - g(x) = 0 \quad \text{(Dirichlet)} \quad \text{or} \quad (\partial u/\partial \vec{n})(x) - g(x) = 0 \quad \text{(Neumann)}, & \text{on } \partial\Omega \end{cases} \tag{1}$$

where $a_{ij}, b_i, c : \Omega \to \mathbb{R}$ for $i, j \in [d] \triangleq \{1, \ldots, d\}$, $f : \Omega \to \mathbb{R}$ and $g : \partial\Omega \to \mathbb{R}$ are all given, and $(\partial u/\partial \vec{n})(x)$ denotes the directional derivative of u along the outer normal direction $\vec{n}$ at the boundary point $x \in \partial\Omega$. In addition, we assume that the elliptic operator has a strong ellipticity, meaning there exists a constant $\theta > 0$ such that $\xi^\top A(x)\xi \geq \theta|\xi|^2$ for any $\xi = (\xi_1, \ldots, \xi_d) \in \mathbb{R}^d$ with $|\xi|^2 = \sum_{i=1}^d |\xi_i|^2$ and $x \in \Omega$ a.e., where $a_{ij} = a_{ji}$ for all $i, j \in [d]$ and $A(x) \triangleq [a_{ij}(x)] \in \mathbb{R}^{d\times d}$, i.e., $A(x)$ is symmetric positive definite with all eigenvalues no smaller than θ almost everywhere in Ω. We also consider solving PDEs involving time, such as the linear second-order parabolic PDE (of finite time horizon):

$$\begin{cases} u_t - \sum_{i=1}^d \partial_i(\sum_{j=1}^d a_{ij}\partial_j u) + \sum_{i=1}^d b_i\partial_i u + cu - f = 0, & \text{in } \Omega \times [0,T] \\ u(x,t) - g(x,t) = 0 \quad \text{(Dirichlet)} \quad \text{or} \quad (\partial u/\partial \vec{n})(x,t) - g(x,t) = 0 \quad \text{(Neumann)}, & \text{on } \partial\Omega \times [0,T] \\ u(x,0) - h(x) = 0, & \text{on } \Omega \end{cases} \tag{2}$$

*School of Mathematical Sciences, Zhejiang University, Hangzhou, Zhejiang, China. Email: 11535015@zju.edu.cn.
†School of Mathematical Sciences, Zhejiang University, Hangzhou, Zhejiang, China. Email: baog@zju.edu.cn.
‡Department of Mathematics and Statistics, Georgia State University, Atlanta, Georgia 30303, USA. Email: xye@gsu.edu.
§School of Mathematics, Georgia Institute of Technology, Atlanta, Georgia 30332, USA. Email: hmzhou@math.gatech.edu.

where $a_{ij}, b_i, c : \Omega \times [0,T] \to \mathbb{R}$ for $i,j \in [d]$ as before, $f : \Omega \times [0,T] \to \mathbb{R}$ and $g : \partial\Omega \times [0,T] \to \mathbb{R}$ and $h : \Omega \to \mathbb{R}$ are given. In either case, we will see that the method developed in this paper can be directly applied to *general high-dimensional PDEs, including both linear and nonlinear ones.*

PDEs are prevalent and have extensive applications in science, engineering, economics, and finance [13, 14]. The most popular standard approaches to calculate numerical solutions of PDEs include finite difference and finite element methods (FEM) [15]. These methods discretize the time interval $[0,T]$ and the domain Ω using mesh grids or triangulations, create simple basis functions on the mesh, convert a continuous PDE into its discrete counterpart, and finally solve the resulting system of basis coefficients to obtain numerical approximations of the true solution. Although these methods have been significantly advanced in the past decades and are able to handle rather complicated and highly oscillating problems, they suffer the so-called "curse of dimensionality" since the number of mesh points increases exponentially fast with respect to the problem dimension d. Hence they quickly become computationally intractable for high dimensional problem in practice. As a consequence, these numerical methods are rarely useful for general high-dimensional PDEs, e.g. $d \geq 4$, especially when a sufficiently high-resolution solution is needed and/or the domain Ω is irregular.

Facing the challenge, our goal is to provide a computational feasible alternative approach to solve general high-dimensional PDEs defined on arbitrarily shaped domains. More specifically, using the weak formulation of PDEs, we parametrize the weak solution and the test function as the primal and adversarial neural networks respectively, and train them in an unsupervised form where only the evaluations of these networks (and their gradients) on some sampled collocation points in the interior and boundary of the domain are needed. Our approach retains the continuum nature of PDEs for which partial derivatives can be carried out directly without any spatial discretization, and is fast and stable in solving general high-dimensional PDEs. Moreover, our method is completely mesh-free and can be applied to PDEs defined on arbitrarily shaped domains, without suffering the issue of the curse of dimensionality.

In the remainder of this paper, we first provide an overview of related work on solving PDEs using machine learning approaches in Section 2. In Section 3, we introduce the weak formulation of stationary PDEs, reformulate the PDE as a saddle-point problem based on the operator norm induced from the weak formulation, and present our proposed algorithm with necessary training details. Then we extend the method to solve PDEs involving time. In Section 4, we provide a number of numerical results to show that our method can solve high-dimensional PDEs efficiently and accurately. Our examples also demonstrate some numerical understandings about selections of neural networks structures including the numbers of layers and nodes in the computations. Section 5 concludes this paper.

2 Related Work

Deep learning techniques have been used to solve PDEs in the past few years. As an emergent research direction, great potentials have been demonstrated by many promising results, even though many fundamental questions remain to be answered. Based on the strategies, these work can be roughly classified into two categories.

In the first category, deep neural networks (DNN) are employed to assist the classical numerical methods. In [16], parallel neural networks are used to improve the efficiency of the finite difference method. In [17], neural network is used to accelerate the numerical methods for matrix algebra problems. Neural network is also applied to improve the accuracy of finite difference method in [18], which can be extended to solve two-dimensional PDEs [19]. In [20, 21], the solution of ordinary differential equations (ODE) is approximated by the combination of splines, where the combination parameters are determined by training a neural network with piecewise linear activation functions. A constrained integration method called GINT is proposed to solving initial boundary value PDEs in [1], where neural networks are combined with the classical Galerkin method. In [22], convolutional neural networks (CNN) is used to solve the large linear system derived from the discretization of incompressible Euler equations. In [23], a neural network-based discretization scheme is developed for the nonlinear differential equation using regression analysis technique. Despite of the improvement over classical numerical methods, these methods still suffer the exponentially increasing problem size and are not tractable for high-dimensional PDEs.

In the second category, the deep neural networks are employed to directly approximate the solution of PDE, which may be more advantageous in dealing with high dimensional problems. In [2], the solution

of the PDE is decomposed into two parts, where the first part is explicitly defined to satisfy the initial boundary conditions and the other part is a product of a mapping parametrized as a neural network and an explicitly defined function that vanishes on the boundary. Then the neural network is trained by minimizing the squared residuals over specified collocation points. An improvement of this method by parametrizing both parts using neural networks in [24], The singular canonical correlation analysis (SVCCA) is introduced to further improve this method in [25]. In contrast to decomposing the solution into two parts, the idea of approximating the solution of PDEs by a single neural network is considered in [26], which is not capable of dealing with high-dimensional problems. In [3, 4], a class of physics informed (PI) deep learning models are developed to approximate the solution of PDEs by incorporating observed data points and initial boundary conditions into the loss function for training. A similar model is present in [12] for high dimensional free boundary parabolic PDEs. A different approach that represents a class of nonlinear PDEs by forward-backward stochastic differential equations is proposed and studied In [5, 6, 7, 8].

Another appealing approach that exploits the variational form of PDEs is considered in [9, 10, 11]. In [9], a committer function is parameterized by a neural network whose weights are obtained by optimizing the variational formulation of the corresponding PDE. In [10], deep learning technique is employed to solve low-dimensional random PDEs based on both strong form and variational form. More recently, an adaptive collocation strategy is presented for a method in [27]. In [28], an adversarial inference procedure is used for quantifying and propagating uncertainty in systems governed by non-linear differential equations, where the discriminator distinguishes the real observation and the approximation provided by the generative network through the given physical laws expressed by PDEs and the generator tries to fool the discriminator. To the best of our knowledge, none of the existing methods models the solution and test function in the weak solution form of the PDE as primal and adversarial networks as proposed in the present work. We will show in the experiment section that the use of weak form is more advantageous especially when the PDEs have singularities where classical solutions do not exist.

3 Proposed Method

To demonstrate the main idea, we first focus on the boundary value problems (BVP) (1). We consider the weak formulation of the PDE, and pose the weak solution as an operator norm minimization. The weak solution and the test function are both parametrized as deep neural networks, where the parameters are learned by an adversarial training governed by the weak formulation. Important implementation details are also provided. Finally, we extend the proposed method to the IBVP (2), where the PDEs are time-dependent.

3.1 PDE and weak formulation

In general, a solution $u \in C^2(\Omega)$ of a BVP (1) requires sufficient regularity of the problem and may not exist in the classical sense. Instead, we consider the *weak formulation* of (1) by multiplying both sides by a test function $\varphi \in H_0^1(\Omega;\mathbb{R})$ and integrating by parts:

$$\begin{cases} \langle \mathcal{A}[u], \varphi \rangle \triangleq \int_\Omega \left(\sum_{j=1}^d \sum_{i=1}^d a_{ij} \partial_j u \partial_i \varphi + \sum_{i=1}^d b_i \varphi \partial_i u + cu\varphi - f\varphi \right) \mathrm{d}x = 0 \\ \mathcal{B}[u] = 0, \quad \text{on } \partial\Omega \end{cases} \tag{3}$$

where $H_0^1(\Omega;\mathbb{R})$ denotes the Sobolev space, a Hilbert space of functions who themselves and their weak partial derivatives are L^2 integrable on Ω with vanishing trace on the boundary $\partial\Omega$. Note that the boundary terms of (3) after integration by parts disappears due to $\varphi = 0$ on $\partial\Omega$. If $u \in H^1(\Omega;\mathbb{R})$ with possibly nonzero trace satisfies (3) for all $\varphi \in H_0^1$, we say that u is a *weak solution* (or *general solution*) of (1). In general, the weak solution to (1) may exist while a classical one may not. In this paper, we therefore seek for the weak solution characterized in (3) so that we can provide an answer to a BVP (1) to the best extent even if it does not admit a solution in the classical sense.

3.2 Induced operator norm minimization

A novel point of view for the weak solution u can be interpreted as follows. We can consider $\mathcal{A}[u] : H_0^1(\Omega) \to \mathbb{R}$ as a linear functional (operator) such that $\mathcal{A}[u](\varphi) \triangleq \langle \mathcal{A}[u], \varphi \rangle$ as defined in (3). Then the operator norm of

$\mathcal{A}[u]$ induced from L^2 norm is defined by

$$\|\mathcal{A}[u]\|_{op} \triangleq \max\{\langle \mathcal{A}[u], \varphi\rangle / \|\varphi\|_2 \mid \varphi \in H_0^1, \varphi \neq 0\}, \tag{4}$$

where $\|\varphi\|_2 = (\int_\Omega |\varphi(x)|^2 \, \mathrm{d}x)^{1/2}$. Therefore, u is a weak solution of (1) if and only if $\|\mathcal{A}[u]\|_{op} = 0$ and the boundary condition $\mathcal{B}[u] = 0$ is satisfied on $\partial\Omega$. As $\|\mathcal{A}[u]\|_{op} \geq 0$, we know that a weak solution u to (1) thus solves the following two equivalent problems in observation of (4):

$$\min_{u \in H^1} \|\mathcal{A}[u]\|_{op}^2 \quad \Longleftrightarrow \quad \min_{u \in H^1} \max_{\varphi \in H_0^1} |\langle \mathcal{A}[u], \varphi\rangle|^2 / \|\varphi\|_2^2. \tag{5}$$

This result is summarized in the following theorem.

Theorem 1. *Suppose u^* satisfies the boundary condition $\mathcal{B}[u^*] = 0$, then u^* is a weak solution of the BVP (1) if and only if u^* solves the problems in (5) and $\|\mathcal{A}[u^*]\|_{op} = 0$.*

Proof. For any fixed $u \in H^1(\Omega)$, we can see that the maximum of $\langle \mathcal{A}[u], \varphi\rangle$ is achievable over $Y \triangleq \{\varphi \in H_0^1(\Omega) \mid \|\varphi\|_2 = 1\}$ since $\langle \mathcal{A}[u], \cdot\rangle$ is continuous and Y is closed in $H_0^1(\Omega)$. Denote $h(u)$ as the maximum of $\langle \mathcal{A}[u], \varphi\rangle$ over Y, then $h(u) = \|\mathcal{A}[u]\|_{op}$ in (4). On the other hand, the space of functions $u \in H^1(\Omega)$ satisfying the boundary condition $\mathcal{B}[u] = 0$, denoted by X, is also closed in $H^1(\Omega)$. Therefore, the minimum of $h(u)$ over X is also achievable. Hence the minimax problem (5) is well-defined.

Now we show that u^* is the solution of the minimax problem (5) if and only if it is the weak solution of the problem (1). Suppose u^*, satisfying the boundary condition $\mathcal{B}[u^*] = 0$, is the weak solution of the problem (1), namely u^* satisfies (3) for all $\varphi \in Y$, then $\langle \mathcal{A}[u^*], \varphi\rangle \equiv 0$ for all $\varphi \in Y$. Therefore, $\|\mathcal{A}[u^*]\|_{op} = 0$, and u^* is the solution of the minimax problem (5). On the other hand, suppose a weak solution $\hat{u}$ of (1) exists. Assume that u^* is the minimizer of the problem (5), i.e., $u^* = \arg\min_{u \in X} h(u)$, but not a weak solution of the problem (1), then there exists $\varphi^* \in Y$ such that $\langle \mathcal{A}u^*, \varphi^*\rangle > 0$. Therefore $h(u^*) = \max_{\varphi \in Y} |\langle \mathcal{A}[u^*], \varphi\rangle| > 0$. However, as we showed above, $h(\hat{u}) = 0$ since $\hat{u}$ is a weak solution of (1), which contradicts to the assumption that u^* is the minimizer of $h(u)$ over X. Hence u^* must also be a weak solution of (1). □

Theorem 1 implies that, to find the weak solution of (1), we can instead seek for the optimal solution u that minimizes (5).

3.3 Weak adversarial network for solving PDE

The formulation (5) inspires an adversarial approach to find the weak solution of (1). More specifically, we seek for the function $u_\theta : \mathbb{R}^d \to \mathbb{R}$, realized as a deep neural network with parameter θ to be learned, such that $\mathcal{A}[u_\theta]$ minimizes the operator norm (5). On the other hand, the test function φ, is a deep adversarial network with parameter η, also to be learned, challenges u_θ by maximizing $\langle \mathcal{A}[u_\theta], \varphi_\eta\rangle$ modulus its own norm $\|\varphi_\eta\|_2$ for every given u_θ in (5).

To train the deep neural network u_θ and the adversarial network φ_η such that they solve (5), we first need to formulate the objective functions of u_θ and φ_η. Since logarithm function is monotone and strictly increasing, we can for convenience reformulate (5) and obtain the objective of u_θ and φ_η in the interior of Ω as follows,

$$L_{\text{int}}(\theta, \eta) \triangleq \log |\langle \mathcal{A}[u_\theta], \varphi_\eta\rangle|^2 - \log \|\varphi_\eta\|_2^2. \tag{6}$$

In addition, the weak solution u_θ also need to satisfy the boundary condition $\mathcal{B}[u] = 0$ on $\partial\Omega$ as in (1). Let $\{x_b^{(j)}\}_{j=1}^{N_b}$ be a set of N_b collocation points on the boundary $\partial\Omega$, then the squared error of u_θ for Dirichlet boundary condition $u = g$ on $\partial\Omega$ is given by

$$L_{\text{bdry}}(\theta) \triangleq (1/N_b) \cdot \textstyle\sum_{j=1}^{N_b} |u_\theta(x_b^{(j)}) - g(x_b^{(j)})|^2. \tag{7}$$

If the Neumann boundary condition in (1) is imposed in the BVP (1), then one can form the loss function $L_{\text{bdry}}(\theta) = (1/N_b) \cdot \sum_{j=1}^{N_b} |\sum_{i=1}^{d} n_i(x_b^{(j)}) \, \partial_i u_\theta(x_b^{(j)}) - g(x_b^{(j)})|^2$ instead, where $\vec{n}(x) = (n_1(x), \ldots, n_d(x))$ is the outer normal direction at $x \in \partial\Omega$. The total objective function is the weighted sum of the two objectives (6) and (7), for which we seek for a saddle point that solves the minimax problem:

$$\min_\theta \max_\eta L(\theta, \eta), \quad \text{where} \quad L(\theta, \eta) \triangleq L_{\text{int}}(\theta, \eta) + \alpha L_{\text{bdry}}(\theta), \tag{8}$$

where $\alpha > 0$ is user-chosen balancing parameter. In theory, the weak solution attains zero for both L_{int} and L_{bdry}, so any choice of α would work. However, different α values impact the performance of the training and we will give examples in Section 4.

3.4 Training algorithm for the weak adversarial network

Given the objective function (8), the key ingredients in the network training are the gradients of $L(\theta, \eta)$ with respect to the network parameters θ and η. Then θ and η can be optimized by alteranting gradient descent and ascent of $L(\theta, \eta)$ in (8) respectively.

To obtain the gradients of L_{int} in (8), we first denote the integrand of $\langle \mathcal{A}[u_\theta], \varphi_\eta \rangle$ in (3) as $I(x; \theta, \eta)$ for every given θ and η. For instance, for the second-order elliptic PDE (1), $I(x; \theta, \eta)$, $\nabla_\theta I(x; \theta, \eta)$, $\nabla_\eta I(x; \theta, \eta)$ are given below in light of the weak formulation (3):

$$\begin{aligned}
I(x;\theta,\eta) &= \textstyle\sum_{j=1}^d \sum_{i=1}^d a_{ij}(x)\partial_j u_\theta(x)\partial_i \varphi_\eta(x) + \sum_{i=1}^d b_i(x)\varphi_\eta(x)\partial_i u_\theta(x) \\
&\quad + c(x)u_\theta(x)\varphi_\eta(x) - f(x)\varphi_\eta(x) \\
\nabla_\theta I(x;\theta,\eta) &= \textstyle\sum_{j=1}^d \sum_{i=1}^d a_{ij}(x)\partial_j \nabla_\theta u_\theta(x)\partial_i \varphi_\eta(x) + \sum_{i=1}^d b_i(x)\varphi_\eta(x)\partial_i \nabla_\theta u_\theta(x) \\
&\quad + c(x)\nabla_\theta u_\theta(x)\varphi_\eta(x) - f(x)\varphi_\eta(x) \\
\nabla_\eta I(x;\theta,\eta) &= \textstyle\sum_{j=1}^d \sum_{i=1}^d a_{ij}(x)\partial_j u_\theta(x)\partial_i \nabla_\eta \varphi_\eta(x) + \sum_{i=1}^d b_i(x)\nabla_\eta \varphi_\eta(x)\partial_i u_\theta(x) \\
&\quad + c(x)u_\theta(x)\nabla_\eta \varphi_\eta(x) - f(x)\nabla_\eta \varphi_\eta(x)
\end{aligned} \tag{9}$$

where $\nabla_\theta u_\theta$ and $\nabla_\eta \varphi_\eta$ are the standard gradients of the networks u_θ and φ_η with respect to their network parameters θ and η. Furthermore, the algorithm and numerical experiments conducted in this paper are implemented in the TensorFlow [29] framework. In this situation, we take advantage of TensorFlow to calculate those derivatives automatically within the framework. To be more specific, due to the definition of L_{int} in (6) and the integrands in (9), we can obtain that $\nabla_\theta L_{\text{int}}(\theta, \eta) = 2(\int_\Omega I(x; \theta, \eta)\,\mathrm{d}x)^{-1}(\int_\Omega \nabla_\theta I(x; \theta, \eta)\,\mathrm{d}x)$. Then we randomly sample N_r collocation points $\{x_r^{(j)} \in \Omega \mid j \in [N_r]\}$ uniformly in the interior of the region Ω, and approximate the gradient $\nabla_\theta L_{\text{int}}(\theta, \eta) \approx 2 \cdot (\sum_{j=1}^{N_r} I(x_r^{(j)}; \theta, \eta))^{-1}(\sum_{j=1}^{N_r} \nabla_\theta I(x_r^{(j)}; \theta, \eta))$. The gradients $\nabla_\eta L_{\text{int}}$, $\nabla_\theta L_{\text{bdry}}$ can be approximated similarly, and hence we omit the details here. With the gradients of $\nabla_\theta L$ and $\nabla_\eta L$, we can apply alternating updates to optimize the parameters θ and η. The resulting algorithm, termed as the weak adversarial network (WAN), is summarized in Algorithm 1.

3.5 Efficiency and stability improvements of WAN

During our experiments, we observed that several small modifications can further improve the efficiency and/or stability of Algorithm 1 in practice. One of these modifications is that, to enforce $\varphi_\eta = 0$ on Ω, we can factorize $\varphi_\eta = w \cdot v_\eta$, where w vanishes on $\partial\Omega$ and v_η is allowed to take any value on $\partial\Omega$. To obtain w for the domain Ω in a given BVP (1), we can set it to the signed distance function of Ω, i.e., $w(x) = \text{dist}(x, \partial\Omega) \triangleq \inf\{|x - y| : y \in \partial\Omega\}$ if $x \in \Omega$ and $-\text{dist}(x, \partial\Omega)$ if $x \notin \Omega$. This signed distance function can be obtained by the fast marching method. Alternatively, one can pretrain w as a neural network such that $w(x) > 0$ for $x \in \Omega$ and $w(x) = 0$ for $x \in \partial\Omega$. To this end, one can parametrize $w_\xi : \Omega \to \mathbb{R}$ as a neural network and optimize its parameter ξ by minimizing the loss function $\sum_{j=1}^{N_b} |w_\xi(x_b^{(j)})| - \varepsilon \sum_{j=1}^{N_r} \log w_\xi(x_r^{(j)})$. In either way, we precompute such w and fix it throughout Algorithm 1 WAN, then the updates of parameters are performed for u_θ and v_η only. In this case, $\varphi_\eta = w \cdot v_\eta$ always vanishes on $\partial\Omega$ so we do not need to worry about the boundary constraint of φ_η during the training.

In addition, our experiments show that in the training process for the weak solution neural network u_θ, applying gradient descent directly to $|\langle \mathcal{A}[u_\theta], \varphi_\eta \rangle|^2 / \|\varphi\|_2^2$ instead of the logarithm term appears to improve efficiency. Finally, formulating the loss function for the boundary condition with absolute error rather than squared error appears empirically to be more efficient in some cases. The computer code that implements these modifications for all test problems in Section 4 will be released upon request.

Algorithm 1 Weak Adversarial Network (WAN) for Solving High-dimensional static PDEs.

Input: N_r/N_b: number of region/boundary collocation points; K_u/K_φ: number of solution/adversarial network parameter updates per iteration.
Initialize: Network architectures $u_\theta, \varphi_\eta : \Omega \to \mathbb{R}$ and parameters θ, η.
while not converged **do**
  Sample collocation points $\{x_r^{(j)} \in \Omega : j \in [N_r]\}$ and $\{x_b^{(j)} \in \partial\Omega : j \in [N_b]\}$
  `# update weak solution network parameter`
  for $k = 1, \ldots, K_u$ **do**
    Update $\theta \leftarrow \theta - \tau_\theta \nabla_\theta L$ where $\nabla_\theta L$ is approximated using $\{x_r^{(j)}\}$ and $\{x_b^{(j)}\}$.
  end for
  `# update test function network parameter`
  for $k = 1, \ldots, K_\varphi$ **do**
    Update $\eta \leftarrow \eta + \tau_\eta \nabla_\eta L$ where $\nabla_\eta L$ is approximated using $\{x_r^{(j)}\}$.
  end for
end while
Output: Weak solution $u_\theta(\cdot)$ of (1).

3.6 Weak adversarial network for PDEs involving time

In this subsection, we consider extending the proposed weak adversarial network method to solve IBVPs with time-dependent PDEs. We provide two approaches for such case: one is to employ semi-discretization in time and iteratively solve $u(x, t_n)$ from a time-independent PDE for each t_n, where Algorithm 1 directly serves as a subroutine; the other one is to treat x and t jointly and consider the weak solution and test functions in the whole region $\Omega \times [0, T]$ without any discretization.

3.6.1 Semi-discretization in time

The weak adversarial network approach can be easily applied to time-dependent PDEs, such as the parabolic equation (2), by discretizing the time and solving an elliptical-type static PDE for each time point. To this end, we partition $[0, T]$ into N uniform segments using time points $0 = t_0 < t_1 < \cdots < t_N = T$, and apply the Crank-Nicolson scheme [30] in classical finite difference method for (2) at each time t_n for $n = 0, \ldots, N-1$ to obtain

$$u(x, t_{n+1}) - u(x, t_n) = \frac{h}{2}\Big(\mathcal{L}(x, t_{n+1}; u(x, t_{n+1})) + f(x, t_{n+1}) + \mathcal{L}(x, t_n; u(x, t_n)) + f(x, t_n)\Big) \tag{10}$$

where $h = T/N$ is the time step size in discretization, $t_n = nh$, and

$$\mathcal{L}(x, t; u) \triangleq \sum_{i=1}^d \partial_i \Big(\sum_{j=1}^d a_{ij}(x, t)\partial_j u(x, t)\Big) - \sum_{i=1}^d b_i(x, t)\partial_i u(x, t) - c(x, t)u(x, t). \tag{11}$$

More precisely, we start with $u(x, t_0) = u(x, 0) = h(x)$, and solve for $u(x, t_1)$ from (10) for $n = 1$. Since (10) is an elliptical-type PDE in $u(x, t_1)$ with boundary value $u(x, t_1) = g(x, t_1)$ on $\partial\Omega$, we can apply Algorithm 1 directly and obtain $u(x, t_1)$ as the parametrized neural network $u_{\theta_1}(x)$ with parameter θ_1 output by Algorithm 1. Following this procedure, we can solve (10) for $u(x, t_n) = u_{\theta_n}(x)$ for $n = 2, 3, \ldots, N$ in order. This process is summarized in Algorithm 2. Other types of time discretization can be employed and the IBVP can be solved with similar idea.

3.6.2 Solving PDE with space and time variables jointly

The proposed weak adversarial network approach can also be generalized to solve the IBVP (2) with space and time variables jointly. In this case, the *weak formulation* of (2) can be obtained by multiplying both sides of (2) by a test function $\varphi(\cdot, t) \in H_0^1(\Omega)$ a.e. in $[0, T]$ and integrating by parts:

$$\begin{aligned} 0 = \langle \mathcal{A}[u], \varphi\rangle \triangleq & \int_\Omega \big(u(x, T)\varphi(x, T) - h(x)\varphi(x, 0)\big)\,\mathrm{d}x - \int_0^T \int_\Omega u\partial_t\varphi\,\mathrm{d}x\,\mathrm{d}t \\ & + \int_0^T \int_\Omega \Big(\sum_{j=1}^d \sum_{i=1}^d a_{ij}\partial_j u\partial_i\varphi + \sum_{i=1}^d b_i\varphi\partial_i u + cu\varphi - f\varphi\Big)\,\mathrm{d}x\,\mathrm{d}t \end{aligned} \tag{12}$$

Algorithm 2 Solving parabolic PDE (2) with semi-discretization in time and Algorithm 1 as subroutine

Input: N_r, N_b, K_u, K_φ as in Algorithm 1. N: number of time points; $h = T/N$.
Initialize: Network architectures $u_\theta, \varphi_\eta : \Omega \to \mathbb{R}$ and parameters θ, η for each t_n. Set $u(x, t_0) = u(x, 0) = h(x)$.
for $n = 0, \cdots, N-1$ **do**
  Solve for $u(x, t_{n+1}) = u_{\theta_{n+1}}(x)$ from the elliptical equation (10) using Algorithm 1
end for
Output: Weak solution $u_\theta(\cdot, t_n)$ of (2) for $n = 1, \ldots, N$.

Algorithm 3 Weak Adversarial Network (WAN) for Solving high-dimensional PDEs in whole space $\Omega \times [0, T]$

Input: $N_r/N_b/N_a$: number of region/boundary/initial collocation points; K_u/K_φ. $\Omega_T \triangleq \Omega \times [0, T]$.
Initialize: Network architectures $u_\theta, \varphi_\eta : \Omega_T \to \mathbb{R}$ and parameters θ, η.
while not converged **do**
  Sample points $\{(x_r^{(j)}, t_r^{(j)}) : j \in [N_r]\} \subset \Omega_T$, $\{(x_b^{(j)}, t_b^{(j)}) : j \in [N_b]\} \subset \partial\Omega \times [0, T]$, $\{x_a^{(j)} : j \in [N_a]\} \subset \Omega$
  `# update weak solution network parameter`
  for $k = 1, \ldots, K_u$ **do**
    Update $\theta \leftarrow \theta - \tau_\theta \nabla_\theta L$ where $\nabla_\theta L$ in (13) is approximated using $\{(x_r^{(j)}, t_r^{(j)})\}$ and $\{(x_b^{(j)}, t_b^{(j)})\}$ and $\{x_a^{(j)}\}$.
  end for
  `# update test function network parameter`
  for $k = 1, \ldots, K_\varphi$ **do**
    Update $\eta \leftarrow \eta + \tau_\eta \nabla_\eta L$ where $\nabla_\eta L$ in (13) is approximated using $\{(x_r^{(j)}, t_r^{(j)})\}$.
  end for
end while
Output: Weak solution $u_\theta(x, t)$ in Ω_T.

Following the same idea presented in Section 3.2–3.3, we parametrize the weak solution u and test function φ as deep neural networks $u_\theta, \varphi_\eta : \Omega \times [0, T] \to \mathbb{R}$ with parameters θ and η respectively. Then we form the objective function in the saddle-point problem of θ and η as

$$L(\theta, \eta) \triangleq L_{\text{int}}(\theta, \eta) + \gamma L_{\text{init}}(\theta) + \alpha L_{\text{bdry}}(\theta), \tag{13}$$

where $\alpha, \gamma > 0$ are user-chosen balancing parameters. In (13), the loss function L_{int} of the interior of $\Omega \times [0, T]$ has the same form as (6) but with $\langle \mathcal{A}[u_\theta], \varphi_\eta \rangle$ defined in (12) and $\|\varphi_\eta\|_2^2 \triangleq \int_0^T \int_\Omega |\varphi(x, t)|^2 \, \mathrm{d}x \, \mathrm{d}t$; L_{init} of the initial value condition in Ω and L_{bdry} of the boundry value condition on $\partial\Omega \times [0, T]$ are given by

$$L_{\text{init}}(\theta) \triangleq (1/N_a) \cdot \textstyle\sum_{j=1}^{N_a} |u_\theta(x_a^{(j)}, 0) - h(x_a^{(j)})|^2 \tag{14}$$

$$L_{\text{bdry}}(\theta) \triangleq (1/N_b) \cdot \textstyle\sum_{j=1}^{N_b} |u_\theta(x_b^{(j)}, t_b^{(j)}) - g(x_b^{(j)}, t_b^{(j)})|^2 \tag{15}$$

where $\{x_a^{(j)} : j \in [N_a]\} \subset \Omega$ are N_a collocation points for the initial condition and $\{(x_b^{(j)}, t_b^{(j)}) : j \in [N_b]\} \subset \partial\Omega \times [0, T]$ are N_b collocation points for the boundary condition.

Similar as in Section 3.5, we factorize $\varphi_\eta = w \cdot v_\eta$ where $w : \Omega_T \to \mathbb{R}$ is set to a function which vanishes on $\partial\Omega$ in advance. The training process is similar as above, which is summarized in Algorithm 3.

4 Numerical Experiments

4.1 Experiment setup

In this section, we conduct a series of numerical experiments of the proposed algorithms (Algorithms 1–3) on BVP and IBVP with high-dimensional linear and *nonlinear* PDEs defined on regular and *irregular* domains. To quantitatively evaluate the accuracy of a solution u_θ, we compute the L_2 relative error $\|u_\theta - u^*\|_2 / \|u^*\|_2$,

Table 1: List of model and algorithm parameters.

Notation	Stands for ...
d	Dimension of $\Omega \subset \mathbb{R}^d$
K_φ	Inner iteration to update test function φ_η
K_u	Inner iteration to update weak solution u_θ
τ_η	Learning rate for network parameter η of test function φ_η
τ_θ	Learning rate for network parameter θ of weak solution u_θ
N_r	Number of sampled collocation points in the region Ω
N_b	Number of sampled collocation points on the boundary $\partial\Omega$ or $\partial\Omega \times [0, T]$
N_a	Number of sampled collocation points in $\Omega = \Omega \times \{0\}$ at initial time
α	Weight parameter of $L_{\text{bdry}}(\theta)$ on the boundary $\partial\Omega$
γ	Weight parameter of $L_{\text{init}}(\theta)$ for the initial value condition

where u^* is the exact solution of the problem and u_θ is the result obtained by the our algorithm. In all experiments, we set both of the primal network (weak solution u_θ) and the adversarial network (test function φ_η) as fully-connected feedforward networks. The u_θ network has a total of 7 layers including the input and output layers, where each hidden layer contains 20 neurons. The activation functions is tanh for layers 1,2,4,6 and elu for layers 3,5 for the problem (16) and softplus for other problems, and identity for the last layer. For the network φ_η, it consists of a total of 9 layers with each hidden layer containing 50 neurons. The activation functions is tanh for layers 1,2, softplus for layers 3,5,8, sinc for layers 2,5,7, and identity for the last layer. These two networks are trained by minimizing the loss function (8) or (13) by AdamGrad [31]. All model and algorithm parameters are summarized in Table 1 for quick reference. The values of these parameters are given in the description of each experiment below.

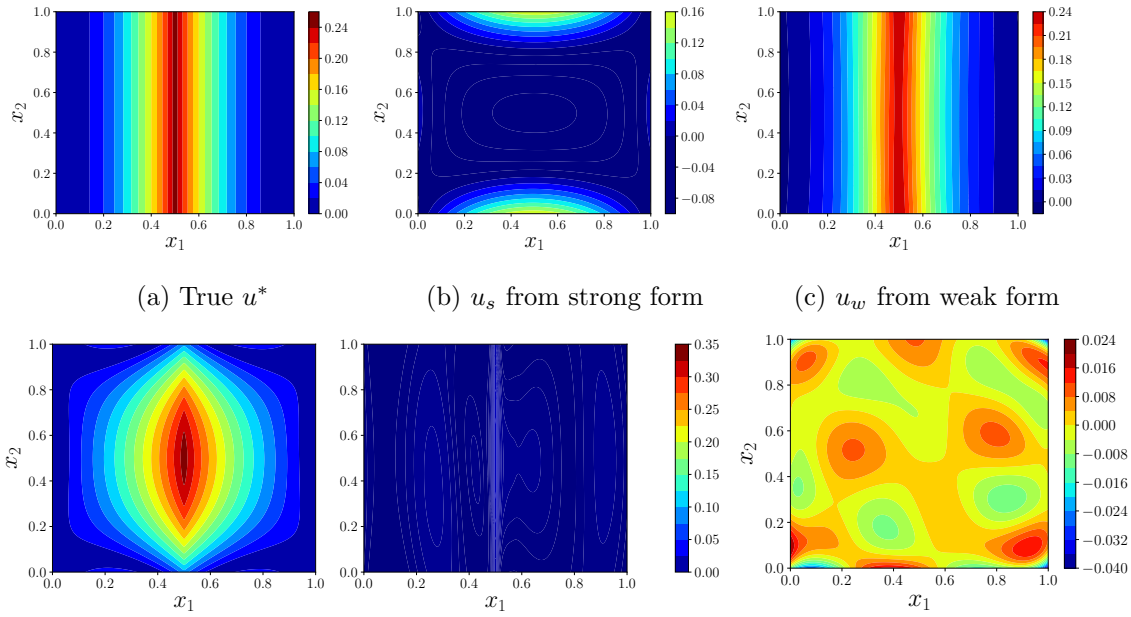
4.1.1 Weak form versus strong form

In the first test, we show that Algorithm 1 based on the weak formulation of PDEs can be advantageous for problems where strong solutions do not exist. Consider the following Poisson equation with Dirichlet boundary condition on $\Omega = (0, 1)^2$:

$$\begin{cases} \Delta u = 2, & \text{in } \Omega \\ u = g, & \text{on } \partial\Omega \end{cases} \tag{16}$$

where $g(x_1, 0) = g(x_1, 1) = x_1^2$ for $0 \le x_1 \le \frac{1}{2}$, $g(x_1, 0) = g(x_1, 1) = (x_1 - 1)^2$ for $\frac{1}{2} \le x_1 \le 1$, and $g(0, x_2) = g(1, x_2) = 0$ for $0 \le x_2 \le 1$. This problem dose not admit a strong (classical) solution, but only a unique weak solution $u^*(x) = u^*(x_1, x_2) = x_1^2$ when $0 \le x_1 \le \frac{1}{2}$ and $u^*(x) = (x_1 - 1)^2$ when $\frac{1}{2} \le x_1 \le 1$, which is shown in Figure 1(a).

We applied the proposed Algorithm 1 to (16) with $K_\varphi = 1, K_u = 2$, $\tau_\eta = 0.04, \tau_\theta = 0.015$, $N_r = 10^4$, $N_b = 4 \times 100$ (100 collocation points on each hyperplane), and $\alpha = 10000 \times N_b$ for 10,000 iterations, after which we obtain an approximation u_w (a solution based on weak form) shown in Figure 1(c). For comparison, we also applied a recent deep neural network method based on strong form of PDEs, which called Physics-informed neural networks(PINN) [3], to the same problem. This neural network contains 9 layers with each hidden layer contains 20 neurons and a hyperbolic tangent activation function. It takes the summation of the mean square error for boundary/initial condition and the mean square error for the physics-law which represent by PDEs as loss. We use the same experimental setup as that offered in `https://github.com/maziarraissi/PINNs` and set collocation points in the domain and on the boundary are 10000 and 400 respectively for 20,000 iterations, which yields a solution u_s shown in Figure 1(b). Due to the nonexistence of strong solution, the result u_s obtained based on strong form fails to capture the singularities of u^*, although it approximately satisfies the PDE with small error $\Delta u_s - f$ in Ω as shown in Figure 1(e) (note that $f = 2$ and the error is small within $[-0.04, 0.024]$). We tried a large range of different parameters for [3], but all return solutions with similar patterns as that of u_s. Furthermore, similar results are obtained when we use the algorithm reported in [32]. In particular, the larger weight boundary just enforces better alignment of u_s with g on $\partial\Omega$, but results even worse matching of Δu_s and f in Ω, and



(a) True u^* (b) u_s from strong form (c) u_w from weak form

(d) $|u_s - u|$ vs. $|u_w - u|$ (e) $\Delta u_s - f$

Figure 1: Comparison of the solutions to (16) based on strong and weak form. (a) The true solution u^*; (b) Approximation u_s obtained by [3] based on the strong form; (c) Approximation u_w obtained by the proposed Algorithm 1 based on the weak form; (d) The pointwise absolute errors of u_s and u_w to u^* under the same scale; (c) The pointwise error $\Delta u_s - f$ in Ω.

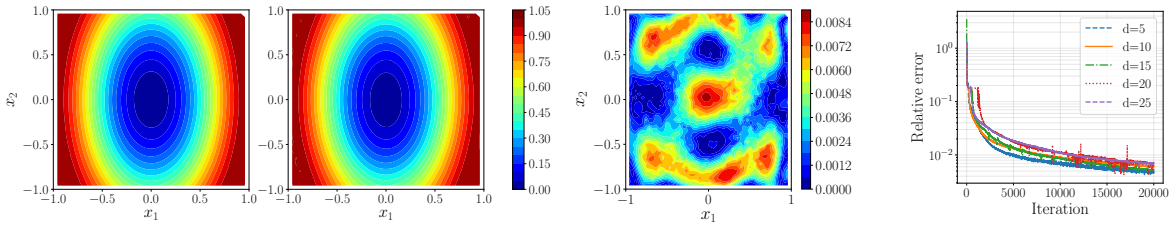
vice versa, which is as expected. We believe it is because the solution u_s based on the strong form tends to enforce smoothness in Ω and aligns with the boundary value g on $\partial\Omega$ by violating both slightly, which results in a smooth solution severely biased from u^* as shown in Figure 1(b). On the other hand, the solution u_w, shown in Figure 1(c), obtained by Algorithm 1 can capture the singularities at the center line and faithfully recover the solution u^*. The comparison of the pointwise absolute errors of u_s and u_w in Figure 1(d) show the significant improvement using weak form in this case.

4.1.2 High-dimensional nonlinear elliptic PDEs with Dirichlet boundary condition

In the second example, we apply Algorithm 1 to a *nonlinear* elliptic PDEs with Dirichlet boundary condition. We test the problem with different dimensions $d = 5, 10, 15, 20, 25$ as follows,

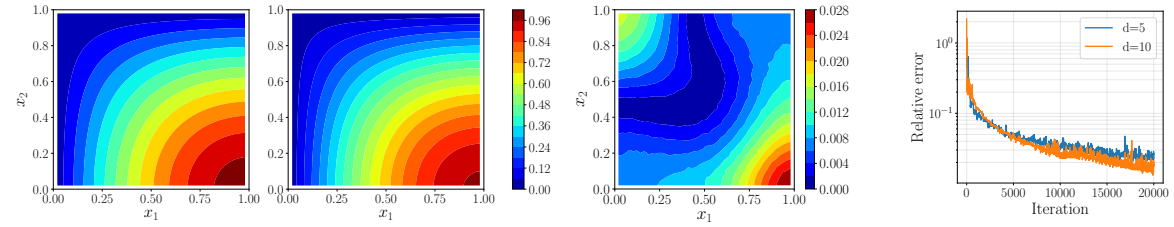
$$\begin{cases} -\nabla \cdot (a(x)\nabla u) + \frac{1}{2}|\nabla u|^2 = f(x) & \text{in } \Omega \triangleq (-1,1)^d, \\ u(x) = g(x) & \text{on } \partial\Omega \end{cases} \tag{17}$$

where $a(x) = 1 + |x|^2$ in Ω, $f(x) = 4\rho_1^2(1+|x|^2)\sin\rho_0^2 - 4\rho_0^2\cos(\rho_0^2) - (\pi+1)(1+|x|^2)\cos(\rho_0^2) + 2\rho_1^2\cos^2(\rho_0^2)$ in Ω, and $g(x) = \sin(\frac{\pi}{2}x_1^2 + \frac{1}{2}x_2^2)$ on $\partial\Omega$, with $\rho_0^2 \triangleq \frac{\pi}{2}x_1^2 + \frac{1}{2}x_2^2$, $\rho_1^2 \triangleq \frac{\pi^2}{4}x_1^2 + \frac{1}{4}x_2^2$. The exact solution of (17) is $u^*(x) = \sin(\frac{\pi}{2}x_1^2 + \frac{1}{2}x_2^2)$ in Ω, the cross section (x_1, x_2) of which is shown in the left panel of Figure 2(a). In this test, we set $K_\varphi = 1$, $K_u = 2$, $\tau_\eta = 0.04$, $\tau_\theta = 0.015$, $N_r = 4,000d$, $N_b = 40d^2$, and $\alpha = 10,000 \times N_b$ for $d = 5, 10$, and $20,000 \times N_b$ for $d = 15, 20$, and $25,000 \times N_b$ for $d = 25$. The solution u_θ after 20,000 iterations for $d = 20$ case is shown in the right panel of Figure 2(a), and the absolute pointwise error $|u_\theta - u^*|$ is shown in Figure 2(b). We show the progresses of the relative error versus iteration in Figure 2(c). After 20,000 iterations, the relative error reaches $0.44\%, 0.62\%, 0.52\%, 0.66\%, 0.69\%$ for $d = 5, 10, 15, 20, 25$ cases, respectively. As we can see, the Algorithm 1 can solve high-dimensional nonlinear PDEs accurately.



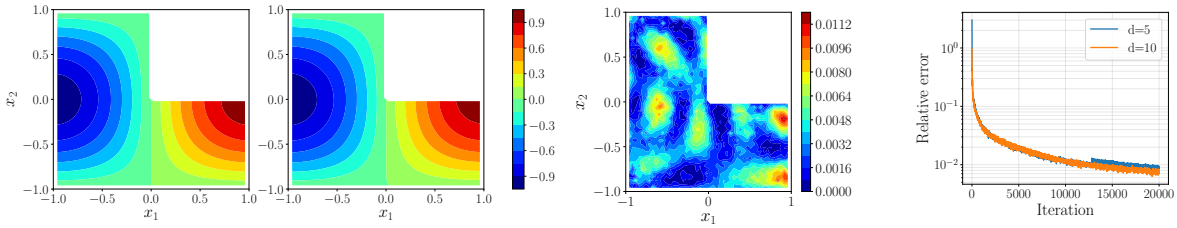
(a) u^* vs u_θ (b) $|u_\theta - u^*|$ (c) Relative error vs iteration

Figure 2: Result for the BVP (17) with nonlinear elliptical PDE and Dirichlet boundary condition. (a) True solution u^* and the approximation u_θ obtained by Algorithm 1 after 20,000 iterations for $d = 20$; (b) The absolute difference $|u_\theta - u^*|$ for $d = 20$; (c) Relative errors versus iteration numbers for $d = 5, 10, 15, 20, 25$ cases. For display purpose, images (a) and (b) only show the slices of $x_3 = \cdots = x_d = 0$ for $d \geq 3$.



(a) u^* vs u_θ (b) $|u_\theta - u^*|$ (c) Relative error vs iteration

Figure 3: Result for the BVP (18) with Neumann boundary condition. (a) True solution u^* and the approximation u_θ obtained by Algorithm 1 after 20,000 iterations for $d = 10$; (b) The absolute difference $|u_\theta - u^*|$ for $d = 10$; (c) Relative errors versus iteration numbers for $d = 5, 10$ cases. For display purpose, images (a) and (b) only show the slices of $x_3 = \cdots = x_d = 0$ for $d \geq 3$.



(a) u^* vs u_θ (b) $|u_\theta - u^*|$ (c) Relative error vs iteration

Figure 4: Result for the BVP (19) with an isotropic Poisson equation and Dirichlet boundary condition on *nonconvex* domain. (a) True solution u^* and the approximation u_θ obtained by Algorithm 1 after 20,000 iterations for $d = 10$; (b) The absolute difference $|u_\theta - u^*|$ for $d = 10$; (c) Relative errors versus iteration numbers for $d = 5, 10$ cases. For display purpose, images (a) and (b) only show the slices of $x_3 = \cdots = x_d = 0$ for $d \geq 3$.

4.1.3 High-dimensional elliptic PDEs with Neumann boundary condition

In this experiment, we show that the proposed Algorithm 1 can be applied to high-dimensional PDEs with *Neumann* boundary condition. Consider the following boundary value problem:

$$\begin{cases} -\Delta u + 2u = f & \text{in } \Omega \triangleq (0,1)^d, \\ \partial u / \partial \vec{n} = g & \text{on } \partial\Omega \end{cases} \tag{18}$$

where $\vec{n}(x) \in \mathbb{R}^d$ is the outer normal at $x \in \partial\Omega$. We set $f(x) = (\frac{\pi^2}{2} + 2)\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)$ in Ω and $g(x) = \left[\frac{\pi}{2}\cos\left(\frac{\pi}{2}x_1\right)\cos\left(\frac{\pi}{2}x_2\right),\ -\frac{\pi}{2}\sin\left(\frac{\pi}{2}x_1\right)\sin\left(\frac{\pi}{2}x_2\right),\ 0, \cdots, 0\right] \cdot \vec{n}$ on $\partial\Omega$. The exact solution of (18) in this case is $u^*(x) = \sin(\frac{x_1}{2}x_1)\cos(\frac{\pi}{2}x_2)$ in Ω as shown in the left panel of Figure 3(a). In this test, we set $K_\varphi = 2$, $K_u = 5$, $\tau_\eta = 0.05$, $\tau_\theta = 0.02$, $N_r = 8 \times 10^4$, $N_b = 2d \times 400$, and $\alpha = 1000 \times N_b$. After 20,000 iterations, the relative error of u_θ to u^* is 2.03% and 1.31% for $d = 5$ and 10 respectively. The solution is shown in the right panel of Figure 3(a), whose pointwise absolute error is shown in Figure 3(b). The progresses of relative error versus iteration number for $d = 5, 10$ are shown in Figure 3(c). This test shows that the proposed algorithm can also work effectively for BVPs with Neumann boundary conditions.

4.1.4 Poisson equation on irregular nonconvex domain

We now consider a BVP with anisotropic Poisson equation and Dirichlet boundary condition on irregular *nonconvex* domain for problem dimension $d = 5, 10$ as follows:

$$\begin{cases} -\nabla \cdot (a(x)\nabla u) = f(x) & \text{in } \ \Omega \triangleq (-1,1)^d \setminus [0,1)^d \\ u(x) = g(x) & \text{on } \ \partial\Omega \end{cases} \tag{19}$$

where $a(x) = 1 + |x|^2$ and $f(x) = \frac{\pi^2}{2} \cdot (1 + |x|^2)\sin(\tilde{x}_1)\cos(\tilde{x}_2) + \pi x_2 \sin(\tilde{x}_1)\sin(\tilde{x}_2) - \pi x_1 \cos(\tilde{x}_1)\cos(\tilde{x}_2)$ in Ω and $g(x) = \sin(\tilde{x}_1)\cos(\tilde{x}_2)$ on $\partial\Omega$, with $\tilde{x}_i \triangleq (\pi/2) \cdot x_i$ for $i = 1, 2$. The true solution is $u^* = \sin(\tilde{x}_1)\cos(\tilde{x}_2)$ in Ω as shown in the left panel of Figure 4(a). In this test, $K_\varphi, K_u, \tau_\eta, \tau_\theta, N_r, N_b$ are set the same as those in problem (17), and $\alpha = 10,000 \times N_b$ and $20,000 \times N_b$ for $d = 5, 10$ respectively. The solution u_θ after 20,000 iterations for $d = 10$ case is shown in the right panel of Figure 4(a), and the absolute pointwise error $|u_\theta - u^*|$ is shown in Figure 4(b). Again, for both values of problem dimension d, we show the progresses of the relative error versus iteration in Figure 2(c). After 20,000 iterations, the relative error reaches 0.86% and 0.80% for $d = 5, 10$ cases, respectively. As we can see, the proposed Algorithm 1 can easily handle PDEs defined on irregular domains.

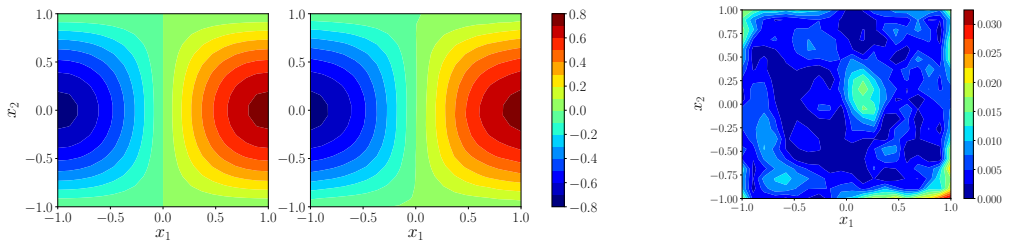
4.1.5 Solving high dimensional parabolic equation involving time

Next, we consider solving the following *nonlinear* diffusion-reaction equation involving time:

$$\begin{cases} u_t - \Delta u - u^2 = f(x,t), & \text{in } \Omega \times [0,T] \\ u(x,t) = g(x,t), & \text{on } \partial\Omega \times [0,T] \\ u(x,0) = h(x), & \text{in } \Omega \end{cases} \tag{20}$$

where $\Omega = (-1,1)^d \subset \mathbb{R}^d$. We first give an example of solving the IBVP (20) in dimension $d = 5$ using Algorithm 2 which discretizes time and uses the Crank-Nicolson scheme (10). In this test, we set $f(x,t) = (\pi^2 - 2)\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)e^{-t} - 4\sin^2(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)e^{-2t}$ in $\Omega \times [0,T]$, $g(x,t) = 2\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)e^{-t}$ on $\partial\Omega \times [0,T]$ and $h(x) = 2\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)$ in Ω. In this case, the exact solution of the IBVP (20) is $u(x,t) = 2\sin(\frac{\pi}{2}x_1)\cos(\frac{\pi}{2}x_2)e^{-t}$. We take $T = 1$ and discretize the time interval $[0,1]$ into $N = 10$ equal segments, and then solve the IBVP using Algorithm 2 with the setup of $K_\varphi, K_u, \tau_\eta, \tau_\theta, N_r, N_b, \alpha$ at each time step are the same as those in Section 4.1.2. Figure 5(a) shows the exact solution u^* (left) and the solution u_θ (right) obtained by Algorithm 2 at final time T. Figure 5(b) shows the point-wise absolute error $|u_\theta(x,T) - u^*(x,T)|$. The absolute relative error reaches 2.8% after 10,000 iterations. The small error implies that the solution obtained by Algorithm 2 is a close approximation to the true solution u^*.

We also considered solving the diffusion-reaction equation (20) for space dimension $d = 5, 10$ using Algorithm 3 by dealing with (x,t) jointly without discretization. In this experiment, we set $f(x,t) = (\frac{\pi^2}{2} - 2)\sin(\frac{\pi}{2}x_1)e^{-t} - 4\sin^2\left(\frac{\pi}{2}x_1\right)e^{-2t}$ in $\Omega \times [0,T]$, $g(x,t) = 2\sin(\frac{\pi}{2}x_1)e^{-t}$ on $\partial\Omega \times [0,T]$, and $h(x) = 2\sin(\frac{\pi}{2}x_1)$ in Ω. The exact solution is $u^*(x,t) = 2\sin(\frac{\pi}{2}x_1)e^{-t}$ in $\Omega \times [0,T]$. In this test, we set $K_\varphi, K_u, \tau_\eta, \tau_\theta, N_r, N_b, \alpha$ the same as in Section 4.1.2 , and $N_a = N_b$ and $\gamma = \alpha$. The solution u_θ after 20,000 iterations for $d = 10$ case is shown in the right panel of Figure 6(a), and the absolute pointwise error $|u_\theta - u^*|$ is shown in Figure 6(b). As has done before, we show the progresses of the relative error versus iteration in Figure 6(c). After 20,000 iterations, the relative error reaches 0.78% and 0.66% for $d = 5, 10$ cases, respectively. Clearly, the Algorithm 3 can solve high-dimensional nonlinear PDEs involving time accurately.



(a) (b)

Figure 5: (a): Exact solution $u^*(x,T)$ (left) and the approximation $u_\theta(x,T)$ (right) obtained by Algorithm 2 at final time $T = 1$ for the IBVP (20) with problem dimension $d = 5$. (b): The corresponding point-wise absolute error $|u(x,T) - u^*(x,T)|$. All images only show the 2D slice of $x_3 = x_4 = x_5 = 0$ for display purpose.

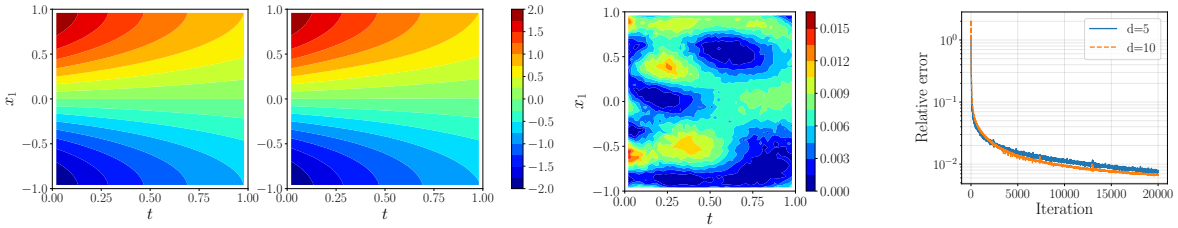
4.1.6 Stability and scalability

In this last set of tests, we evaluate the performance of Algorithm 1 with different parameter settings and increasing problem dimensionality. First, we test the effects of different numbers of collocation points N_r and N_b on the problem (17) with $d = 5$. We set $K_\varphi = 1$, $K_u = 2$, $\tau_\eta = 0.04$ and $\tau_\theta = 0.015$ for this test, and then use different numbers of collocations points N_r in the region and N_b on the boundary (each time with one fixed and the other one varying). In particular, we run Algorithm 1 for varying N_b with fixed (a) $N_r = 500$ and (b) $N_b = 16,000$, and then for varying N_r with fixed (c) $N_b = 5$ and (d) $N_b = 10$. The progresses of relative error versus running time (the iteration stops when the L_2 relative error reaches 1%) for these cases are shown in Figure 7. As we can see, in all cases, Algorithm 1 stably makes progresses towards the weak solution. We also tested the effect of network architectures of u_θ. We tried a number of different combinations of layer and neuron numbers for u_θ. With fixed layer numbers 3 and 9, we show the progresses of training with different number of per-layer neurons in Figure 8(a)(b) respectively. Similarly, with fixed per-layer neuron numbers 10 and 20, we show the same training process with varying numbers of layers in Figure 8(c)(d) respectively. In all of these tests, we can see the relative error of u_θ gradually decays towards 0 except the case when the number of neuron is 5, which indicates sufficient neurons are required to accurately approximate the solution. In most cases, more layers and/or neurons yield faster decay of relative error, but this is not always the case. We know that more layers and/or neurons increase representation capacity of the neural network u_θ, but they can introduce much more parameters to train, yield longer training time, and may result in overfitting of the represenation. We plan to investigate this problem in more depth in our future work.

To show the scalability of our method, we plotted the total computation time (in seconds) of Algorithm 1 applied to BVP (17) for $d = 5, 10, 15, 20, 25$ in Figure 8(e). This figures shows the times when u_θ first hit 1% relative error to u^* for these problem dimensions. It appears that, with the parameter setting we selected, the computation time increases approximately linearly in problem dimension d. This shows that Algorithm 1 has great potential in scalability for high dimensional PDEs empirically.

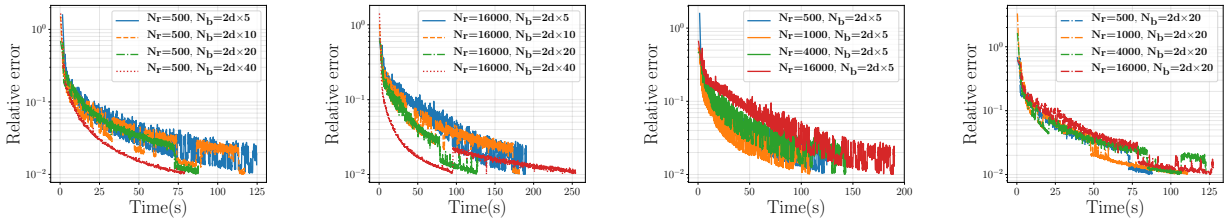
5 Concluding Remarks

In this paper, we developed a novel approach, called *weak adversarial network* or WAN, to solve general high-dimensional linear and nonlinear PDEs defined on arbitrary domains. Inspired by the weak formulation of PDEs, we rewrite the problem of finding the weak solution of the PDE as a saddle-point problem, where the weak solution and the test function are parameterized as the primal and adversarial networks, respectively. The objective function is completely determined by the PDE, the initial and boundary conditions, of the IBVP; and the parameters of these two networks are alternately updated during the training to reach optimum. The training only requires evaluations of the networks on randomly sampled collocations points in the interior and boundary of the domain, and hence can be completed quickly on desktop-level machines with standard deep learning configuration. We demonstrated the promising performance of WAN on a variety



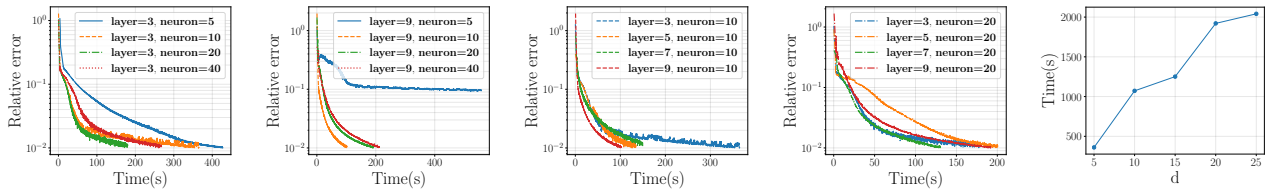
(a) u^* vs u_θ (b) $|u_\theta - u^*|$ (c) Relative error vs iteration

Figure 6: Result for the IBVP (20) with a *nonlinear* diffusion-reaction equation. (a) True solution u^* and the approximation u_θ obtained by Algorithm 3 after 20,000 iterations for $d = 5$; (b) The absolute difference $|u_\theta - u^*|$ for $d = 5$; (c) Relative errors versus iteration numbers for $d = 5, 10$ cases. For display purpose, images (a) and (b) show the slices of $x_2 = \cdots = x_d = 0$ for $d \geq 2$.



(a) $N_r = 500$ (b) $N_r = 16,000$ (c) $N_b = 5$ (d) $N_b = 20$

Figure 7: Effects of the numbers of sampled region collocation points N_r and boundary collocation points N_b on the nonlinear elliptical PDE (17) with $d = 5$. The progresses of relative error versus running time are shown with varying N_b for (a) $N_r = 500$ and (b) $N_r = 16,000$, and with varying N_r for (c) $N_b = 2d \times 5$ and (d) $N_b = 2d \times 20$.



(a) 3-layer (b) 9-layer (c) 10-neuron (d) 20-neuron (e) Time(s) vs d

Figure 8: Effects of the numbers of layers and neurons, as well as the dimensionality, on the nonlinear elliptical PDE problem (17). The progresses of relative error versus running time are shown with varying number of neurons per layer for a total of (a) 3 layers and (b) 9 layers, and with number of layers for the same number of (c) 10 and (d) 20 neurons per layer; (e) The computation time (in seconds) versus problem dimension $d = 5, 10, 15, 20, 25$.

of PDEs with high dimension, nonlinearity, and nonconvex domain which are challenging issues in classical numerical PDE methods. In all tests, WAN exhibits high efficiency and strong stability without suffering these issues.

References

[1] K. Rudd, S. Ferrari, A constrained integration (cint) approach to solving partial differential equations using artificial neural networks, Neurocomputing 155 (2015) 277–285.

[2] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE transactions on neural networks 9 (5) (1998) 987–1000.

[3] M. Raissi, P. Perdikaris, G. 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 (2019) 686–707.

[4] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561.

[5] C. Beck, W. E, A. Jentzen, Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations, Journal of Nonlinear Science (2017) 1–57.

[6] M. Fujii, A. Takahashi, M. Takahashi, Asymptotic expansion as prior knowledge in deep learning method for high dimensional bsdes, Asia-Pacific Financial Markets (2017) 1–18.

[7] J. Han, A. Jentzen, W. E, Overcoming the curse of dimensionality: Solving high-dimensional partial differential equations using deep learning, arXiv preprint arXiv:1707.02568 (2017) 1–13.

[8] W. E, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Communications in Mathematics and Statistics 5 (4) (2017) 349–380.

[9] Y. Khoo, J. Lu, L. Ying, Solving for high-dimensional committor functions using artificial neural networks, Research in the Mathematical Sciences 6 (1) (2019) 1.

[10] M. A. Nabian, H. Meidani, A deep neural network surrogate for high-dimensional random partial differential equations, arXiv preprint arXiv:1806.02957.

[11] W. E, B. Yu, The deep ritz method: A deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics 6 (1) (2018) 1–12.

[12] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics 375 (2018) 1339–1364.

[13] A. Quarteroni, A. Valli, Numerical approximation of partial differential equations, Vol. 23, Springer Science & Business Media, 2008.

[14] J. W. Thomas, Numerical partial differential equations: finite difference methods, Vol. 22, Springer Science & Business Media, 2013.

[15] T. J. Hughes, The finite element method: linear static and dynamic finite element analysis, Courier Corporation, 2012.

[16] H. Lee, I. S. Kang, Neural algorithm for solving differential equations, Journal of Computational Physics 91 (1) (1990) 110–131.

[17] L. Wang, J. Mendel, Structured trainable networks for matrix algebra, in: 1990 IJCNN International Joint Conference on Neural Networks, IEEE, 1990, pp. 125–132.

[18] D. Gobovic, M. E. Zaghloul, Analog cellular neural network with application to partial differential equations with variable mesh-size, in: Proceedings of IEEE International Symposium on Circuits and Systems-ISCAS'94, Vol. 6, IEEE, 1994, pp. 359–362.

[19] R. Yentis, M. Zaghloul, Vlsi implementation of locally connected neural network for solving partial differential equations, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications 43 (8) (1996) 687–690.

[20] A. J. Meade Jr, A. A. Fernandez, The numerical solution of linear ordinary differential equations by feedforward neural networks, Mathematical and Computer Modelling 19 (12) (1994) 1–25.

[21] A. J. Meade Jr, A. A. Fernandez, Solution of nonlinear ordinary differential equations by feedforward neural networks, Mathematical and Computer Modelling 20 (9) (1994) 19–44.

[22] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating eulerian fluid simulation with convolutional networks, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 3424–3433.

[23] Y. Suzuki, Neural network-based discretization of nonlinear differential equations, Neural Computing and Applications (2017) 1–16.

[24] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complex geometries, Neurocomputing 317 (2018) 28–41.

[25] M. Magill, F. Qureshi, H. de Haan, Neural networks trained to solve differential equations learn general representations, in: Advances in Neural Information Processing Systems, 2018, pp. 4071–4081.

[26] M. Dissanayake, N. Phan-Thien, Neural-network-based approximations for solving partial differential equations, communications in Numerical Methods in Engineering 10 (3) (1994) 195–201.

[27] C. Anitescu, E. Atroshchenko, N. Alajlan, T. Rabczuk, Artificial neural network methods for the solution of second order boundary value problems, Computers, Materials & Continua 59 (1) (2019) 345–359.

[28] Y. Yang, P. Perdikaris, Adversarial uncertainty quantification in physics-informed neural networks, Journal of Computational Physics 394 (2019) 136–152.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: A system for large-scale machine learning, in: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016, pp. 265–283.

[30] J. Crank, P. Nicolson, A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, Advances in Computational Mathematics 6 (1) (1996) 207–226.

[31] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, Journal of Machine Learning Research 12 (Jul) (2011) 2121–2159.

[32] K. Xu, B. Shi, S. Yin, Deep learning for partial differential equations, 2018.