LEAST-SQUARES NEURAL NETWORK (LSNN) METHOD FOR SCALAR HYPERBOLIC PARTIAL DIFFERENTIAL EQUATIONS

MIN LIU* AND ZHIQIANG CAI†

Abstract. This chapter offers a comprehensive introduction to the least-squares neural network (LSNN) method introduced in [14, 16], for solving scalar first-order hyperbolic partial differential equations, specifically linear advection-reaction equations and nonlinear hyperbolic conservation laws. The LSNN method is built on an equivalent least-squares formulation of the underlying problem on an admissible solution set that accommodates discontinuous solutions. It employs ReLU neural networks (in place of finite elements) as the approximating functions, uses a carefully designed physics-preserved numerical differentiation, and avoids penalization techniques such as artificial viscosity, entropy condition, and/or total variation. This approach captures shock features in the solution without oscillations or overshooting. Efficiently and reliably solving the resulting non-convex optimization problem posed by the LSNN method remains an open challenge. This chapter concludes with a brief discussion on application of the structure-guided Gauss-Newton (SgGN) method developed recently in [21] for solving shallow NN approximation.

Key words. advection-reaction equation, discrete divergence operator, least-squares method, ReLU neural network, nonlinear hyperbolic conservation law

1. Introduction.

Over the past five decades, numerous advanced mesh-based numerical methods have been developed for solving nonlinear hyperbolic conservation laws (HCLs) (see, e.g., [44, 36, 54, 45, 57, 40]). However, accurately approximating solutions to HCLs remains computationally challenging due to two key difficulties. First, the location of the discontinuities in the solution is typically unknown in advance. Second, the strong form of the partial differential equation (PDE) becomes invalid at points where the solution is discontinuous.

Recently, neural networks (NNs) have emerged as a novel class of functions approximators for solving partial differential equations (PDEs) (see, e.g., [17, 33, 53, 55] and Section 1.1). A neural network function is a linear combination of compositions of linear transformations and a nonlinear univariate activation function. One commonly used activation function is the Rectified Linear Unit (ReLU), defined as $\sigma(s) = \max\{0, s\}$. A ReLU neural network thus constructs a function as a piecewise linear approximation, with "break points" determined by its parameters, effectively forming a data-adaptive partition of the domain. As demonstrated in [14, 18, 20], ReLU NNs can approximate discontinuous functions with unknown interfaces far more effectively than traditional approximating functions, such as polynomials or continuous/discontinuous piecewise polynomials defined on a quasi-uniform, predetermined mesh. This makes ReLU NNs particularly suitable for addressing the first challenge.

The strong form of a hyperbolic PDE is typically written with partial derivatives along coordinate directions, supplemented by the Rankine-Hugoniot (RH) jump condition at discontinuity interfaces for HCLs. Due to the unknown location of these interfaces, enforcing the RH condition in computations is difficult, if not impossible. To address this, we reformulate the PDE using physically meaningful derivatives, allowing the new form of the PDE to remain well-defined at the interface (see (2.5) for the directional derivative and (2.21) for the divergence operator). By applying the L^2 least-squares principle to this reformulated PDE, we derive an equivalent least-squares minimization problem on an admissible solution set that accommodates discontinuous solutions. Through appropriate numerical integration for the integral and physics-preserved numerical differentiation for the physically meaningful derivative, the least-squares neural network (LSNN) method is established as minimizing the discrete counterpart of the least-squares functional over the set of

*School of Mechanical Engineering, Purdue University, 585 Purdue Mall, West Lafayette, IN 47907-2088 (liu66@purdue.edu).

†Department of Mathematics, Purdue University, 150 N. University Street, West Lafayette, IN 47907-2067 (caiz@purdue.edu).

NN functions.

Without relying on penalization techniques such as inflow boundary conditions, artificial viscosity, entropy conditions, or total variation constraints, the LSNN method introduced in [14, 16] effectively captures the shock of the underlying problem without oscillations or overshooting. Additionally, the LSNN method is substantially more efficient in terms of degrees of freedom (DoF) compared to adaptive mesh refinement (AMR) methods, which locate the discontinuity interface through an adaptive mesh refinement process.

Despite the impressive approximation capabilities of NNs, the discretization resulting from NN-based methods leads to a non-convex optimization problem in the NN parameters. This high-dimensional, non-convex optimization is often computationally intensive and complex, presenting a significant bottleneck in using NNs for numerically solving PDEs. Nonetheless, considerable research efforts are underway, with some promising progress in developing efficient and reliable iterative solvers (training algorithms) and in designing effective initializations [21, 22, 23].

The chapter is organized as follows. Section 2 describes the advection-reaction equation and the scalar nonlinear HCL, their equivalent least-squares formulations, and preliminaries. ReLU neural network and its approximation property to discontinuous functions are introduced in Section 3. The physics-preserved numerical differentiation and the LSNN method are defined in Section 4. Section 5 discusses efficient iterative solvers. Finally, numerical results for various benchmark test problems are given in Section 6.

1.1. Methodological Remarks and Related Work. Since NN functions are inherently nonlinear with respect to certain parameters, it is both convenient and natural to discretize a PDE by reformulating it as an optimization problem through either natural energy minimization or manufactured least-squares (LS) principles. Consequently, existing NN-based numerical methods for solving PDEs fall into two main categories: (1) Energy-based methods, such as deep Ritz and finite neuron methods [33, 58, 46, 48], which employ the Ritz formulation for the primary variable [34], and the dual neural network (DuNN) [49], which uses complementary energy minimization for the dual variable [13, 34]. (2) Deep LS methods, which formulate PDE residuals into manufactured least squares using various norm choices and problem forms [32, 3, 53, 55, 17, 14]. Energy-based approaches are applicable to a class of self-adjoint and positive definite problems that commonly arise in continuum mechanics. These methods are not only physics informed but also physics preserved, when using NN as approximating functions, it is natural to discretize the underlying problem based on an energy formulation.

However, many scientific and engineering problems are non-self-adjoint and thus lack a natural minimization principle. In such cases, one can then apply the LS principle to create a manufactured one. The LS principle offers great flexibility: all consistent equations and data can be incorporated into the LS functional. Yet, balancing the various terms is challenging; ill-scaled LS functionals can lead to suboptimal accuracy and inefficient training. The minimal requirement for a viable LS formulation is its equivalence to the original PDE; otherwise, the physical fidelity of the model is compromised.

A classical example is the Bramble–Schatz LS (BSLS) formulation [11] for scalar elliptic PDEs, which applies the LS principle directly to the *strong* form of the PDE, the boundary and the initial conditions using appropriate Sobolev norms [12]. When the solution is sufficiently smooth, e.g., belongs to $H^2(\Omega)$ (the collection of functions whose second-order weak derivatives are square-integrable), the BSLS is equivalent to the underlying problem and yields a well-balanced formulation. However, it fails in the presence of singularities, such as those caused by geometric corners or material interfaces. To address this, LS methods have been extended to first-order systems with properly chosen norms, resulting in numerous viable LS finite element methods. These methods, grounded in L^2-based norms, have been thoroughly developed and analyzed for problems such as convection-diffusion-reaction, elasticity, and Stokes equations [7, 8, 6, 9, 10, 28, 27]. In particular, the LS methods introduced in [24, 25, 29] are based on the original physical first-order systems,

ensuring equivalence with the underlying PDEs. When adapted for NN discretizations, these formulations preserve physical laws provided that the numerical differentiation of certain differential operators also respects the underlying physics.

For scalar nonlinear hyperbolic conservation laws, several NN-based numerical methods have been recently introduced by various researchers ([2, 15, 14, 35, 53, 52]). Those methods can be categorized as the physics-informed neural networks (PINNs) [2, 35, 53, 52] and the LSNN methods [15, 14, 16]. Both methods are based on the least-squares principle, but the former uses the discrete l^2 norm while the latter uses the continuous L^2 norm. This difference explains why the latter can employ ReLU activation function, that is not differentiable pointwisely, and adaptively find accurate numerical integration (see Section 4). Fundamentally, the former is based on the strong form ((2.1) or (2.12)) that *violates* the underlying physics (see Section 2), and the latter builds on a correct ("weak") form ((2.5) or (2.21)) that *preserves* the underlying physics.

Due to such a violation, the original PINN produces unacceptable approximate solution of the underlying problem by scientific computing standard. This phenomenon was observed by several researchers, e.g., [35, 52]. Furthermore, [35] modified the loss function by penalizing the artificial viscosity term. [52] applied the discrete l^2 norm to the boundary integral equations over control volumes instead of the differential equations over points and modified the loss function by penalizing the entropy, total variation, and/or artificial viscosity. Even though the least-squares principle permits freedom of various penalizations, choosing proper penalization constants can be challenging in practice and it affects the accuracy, efficiency, and stability of the method. In contrast, the LSNN does not require any penalization constants (see Section 4).

2. Scalar Hyperbolic Partial Differential Equations.

Let Ω be a bounded open domain in $\mathbb{R}^d$ ($d = 1$, 2, or 3) with Lipschitz boundary, and $I = (0, T)$ be the temporal interval. This section describes linear advection-reaction equations defined on Ω and scalar nonlinear hyperbolic conservation laws defined on $\Omega \times I$ and their equivalent least-squares formulations.

2.1. Advection-Reaction Equations.

Let $\boldsymbol{\beta}(\mathbf{x}) = (\beta_1, \cdots, \beta_d)^t \in C^1(\bar{\Omega})^d$ be the advective velocity field having no stagnation point and $\gamma \in C(\bar{\Omega})$ be the reaction coefficient. Without loss of generality, assume that the magnitude of $\boldsymbol{\beta}(\mathbf{x})$ is one in Ω, i.e., $|\boldsymbol{\beta}(\mathbf{x})| \equiv 1$. Otherwise, the equation below in (2.1) may be rescaled by dividing $|\boldsymbol{\beta}(\mathbf{x})|$.

Let $f \in L^2(\Omega)$ and $g \in L^2(\Gamma_-)$ be given scalar-valued functions, where Γ_- is the inflow part of the boundary $\Gamma = \partial\Omega$ given by

$$\Gamma_- = \{\mathbf{x} \in \Gamma : \boldsymbol{\beta}(\mathbf{x}) \cdot \boldsymbol{n}(\mathbf{x}) < 0\}$$

with $\boldsymbol{n}(\mathbf{x})$ the unit outward normal vector to Γ at $\mathbf{x} \in \Gamma$. Consider the following linear advection-reaction equation [44]

$$\begin{cases} \sum_{i=1}^{d} \beta_i(\mathbf{x}) \dfrac{\partial u(\mathbf{x})}{\partial x_i} + \gamma u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_-. \end{cases} \tag{2.1}$$

If the inflow boundary data g is discontinuous, so is the solution $u(\mathbf{x})$. Hence, the solution has no partial derivative along any coordinate direction on the discontinuity interface $\mathcal{I}$, which, in turn, implies that the PDE in (2.1) is not valid on the $\mathcal{I}$.

To deal with this issue, let us introduce characteristic curves of (2.1) defined by

$$\frac{d\mathbf{x}_\xi(t)}{dt} = \boldsymbol{\beta}(\mathbf{x}_\xi(t)) \tag{2.2}$$

with initial condition $\mathbf{x}_\xi(0) = \boldsymbol{\xi}$, where $\boldsymbol{\xi}$ is a point on the inflow boundary Γ_-. Now, the PDE in (2.1) becomes the following ordinary differential equations

$$\frac{d\, u(\mathbf{x}_\xi(t))}{dt} + \gamma u(\mathbf{x}_\xi(t)) = f(\mathbf{x}_\xi(t)) \tag{2.3}$$

with initial condition $u(\mathbf{x}_\xi(0)) = g(\boldsymbol{\xi})$ for all $\boldsymbol{\xi} \in \Gamma_-$. Denote by

$$\Gamma_-^d = \{\boldsymbol{\xi} \in \Gamma_- : g \text{ is discontinuous at } \boldsymbol{\xi}\}$$

the set of discontinuity of g. Then the discontinuity interface of the solution is given by

$$\mathcal{I} = \left\{\mathbf{x}_\xi(t) \in \Omega : \boldsymbol{\xi} \in \Gamma_-^d, t \in [0,\infty), \text{ and } \mathbf{x}_\xi(t) \text{ is the solution of (2.3)}\right\}.$$

The interface $\mathcal{I}$ partitions the domain Ω into submains $\{\Omega_i\}$, and one may compute the solution in each subdomain. This approach requires calculation of the discontinuity interface and is not applicable to nonlinear HCLs.

To circumvent this difficulty, let us describe the approach developed in [14]. To this end, introduce the directional differential operator along the direction $\boldsymbol{\beta}$ by

$$D_{\boldsymbol{\beta}}\, v(\mathbf{x}) = \lim_{\tau \to 0} \frac{u(\mathbf{x}) - u\big(\mathbf{x} - \tau\boldsymbol{\beta}(\mathbf{x})\big)}{\tau} \tag{2.4}$$

for function v whose directional derivative along $\boldsymbol{\beta}$ exists. Then the linear advection-reaction equation with discontinuous solution may be written in the entire domain Ω as follows

$$\left\{\begin{array}{rcll} D_{\boldsymbol{\beta}}\, u + \gamma u &=& f & \text{in } \Omega, \\ u &=& g & \text{on } \Gamma_-. \end{array}\right. \tag{2.5}$$

Now, we are ready to introduce LS formulations of the linear advection-reaction problem based on (2.5). To do so, denote the solution space by

$$V_{\boldsymbol{\beta}} = \left\{v \in L^2(\Omega) : D_{\boldsymbol{\beta}}\, v \in L^2(\Omega)\right\}, \tag{2.6}$$

and define the following least-squares functional

$$\mathcal{L}(v;\mathbf{g}) = \|D_{\boldsymbol{\beta}}\, v + \gamma\, v - f\|_{0,\Omega}^2 + \|v - g\|_{-\boldsymbol{\beta}}^2, \quad \forall\, v \in V_{\boldsymbol{\beta}}, \tag{2.7}$$

where $\mathbf{g} = (f, g)$, and $\|\cdot\|_{0,\Omega}$ and $\|\cdot\|_{-\boldsymbol{\beta}}$ are the $L^2(\Omega)$ and the weighted $L^2(\Gamma_-)$ norms on the domain and the inflow boundary given by

$$\|v\|_{0,\Omega} = (v,v)^{1/2} = \left(\int_\Omega v^2\, d\mathbf{x}\right)^{1/2} \quad \text{and} \quad \|v\|_{-\boldsymbol{\beta}} = \langle v, v\rangle_{-\boldsymbol{\beta}}^{1/2} = \left(\int_{\Gamma_-} |\boldsymbol{\beta}\cdot\boldsymbol{n}|\, v^2\, ds\right)^{1/2},$$

respectively. Then the least-squares formulation of problem (2.5) studied in [4, 30, 5] is to seek $u \in V_{\boldsymbol{\beta}}$ such that

$$\mathcal{L}(u;\mathbf{g}) = \min_{v \in V_{\boldsymbol{\beta}}} \mathcal{L}(v;\mathbf{g}). \tag{2.8}$$

The well-posedness of the least-squares formulation in (2.8) was established in [30].

Problem (2.8) enforces the inflow boundary condition via a penalization in the weighted $L^2(\Gamma_-)$ norm. Alternatively, the condition can be imposed directly within the solution set, or both in the functional and the solution set. Specifically, define

$$V_{\boldsymbol{\beta}}(g) = \left\{v \in V_{\boldsymbol{\beta}} : v|_{\Gamma_-} = g\right\} = \left\{v \in L^2(\Omega) : D_{\boldsymbol{\beta}}\, v \in L^2(\Omega),\, v|_{\Gamma_-} = g\right\}. \tag{2.9}$$

An equivalent least-square formulation is to find $u \in V_{\boldsymbol{\beta}}(g)$ such that

$$\mathcal{L}(u;\mathbf{g}) = \min_{v \in V_{\boldsymbol{\beta}}(g)} \mathcal{L}(v;\mathbf{g}) \quad \text{or} \quad \hat{\mathcal{L}}(u) = \min_{v \in V_{\boldsymbol{\beta}}(g)} \hat{\mathcal{L}}(v). \tag{2.10}$$

where $\hat{\mathcal{L}}(v) = \|D_{\boldsymbol{\beta}}\, v + \gamma\, v - f\|_{0,\Omega}^2$ is the first term of the LS functional $\mathcal{L}(v;\mathbf{g})$ in (2.7). Although the LS formulations in (2.8) and (2.10) are theoretically equivalent, practical considerations, especially the limited understanding of optimizers for non-convex problems, often motivate the use of first formulation in (2.10), with an appropriately weighted second term in $\mathcal{L}(v;\mathbf{g})$.

REMARK 2.1. *The advection-reaction equation is often expressed in a conservative form as follows*

$$\begin{cases} \mathbf{div}\,(\boldsymbol{\beta} u) + \gamma u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_-. \end{cases} \tag{2.11}$$

When the solution u is discontinuous, the divergence operator **div** *must be interpreted in a weak sense, as similarly defined in (2.19). We can then apply the LS principle to (2.11) (see the subsequent section for details).*

2.2. Scalar Nonlinear Hyperbolic Conservation Laws. Let $\mathbf{f}(u) = (f_1(u), ..., f_d(u))$ be the spatial flux vector field, Γ_- be the part of the boundary $\partial\Omega \times I$ where the characteristic curves enter the domain $\Omega \times I \subset \mathbb{R}^{d+1}$, and the boundary data g and the initial data u_0 be given scalar-valued functions defined on Γ_- and Ω, respectively. Consider the following scalar nonlinear hyperbolic conservation law

$$u_t(\mathbf{x}, t) + \sum_{i=1}^{d} \frac{\partial f_i\,(u(\mathbf{x}, t))}{\partial x_i} = 0 \quad \text{in } \Omega \times I \tag{2.12}$$

with the inflow and initial conditions

$$u = g \quad \text{on } \Gamma_- \quad \text{and} \quad u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{on } \Omega, \tag{2.13}$$

respectively, where u_t is the partial derivative of u with respect to the temporal variable t. Without loss of generality, assume that $f_i(u)$ is twice differentiable for all $i \in \{1, \ldots, d\}$.

The solution of (2.12) is often discontinuous due to a discontinuous initial or inflow boundary condition or shock formation. Hence, the strong form in (2.12) is only valid where the solution is differentiable. Let $\mathcal{I}$ denote the discontinuity interface of the solution. To fully characterize the behavior across $\mathcal{I}$, an additional equation, the so-called Rankine-Hugoniot (RH) jump condition (see, e.g., [44, 37]), is required:

$$\big(\mathbf{f}(u^+), u^+\big) \cdot \mathbf{n}^+\big|_{\mathcal{I}} + \big(\mathbf{f}(u^-), u^-\big) \cdot \mathbf{n}^-\big|_{\mathcal{I}} = 0, \tag{2.14}$$

where $\mathbf{n}^+ = (\mathbf{n}_x^+, n_t^+)$ and $\mathbf{n}^- = (\mathbf{n}_x^-, n_t^-)$ are the unit vector normal to the interface $\mathcal{I}$ with opposite directions, and $u(\mathbf{x}, t)$ have two different values on $\mathcal{I}$, $u^+(\mathbf{x}, t)$ and $u^-(\mathbf{x}, t)$, defined by

$$u^{\pm}(\mathbf{x}, t) = \lim_{\epsilon \to 0} u(\mathbf{x} - \epsilon \mathbf{n}_x^{\pm}, t - \epsilon n_t^{\pm}) \quad \forall\ (\mathbf{x}, t) \in \mathcal{I}.$$

Thus, a complete formulation of the scalar nonlinear hyperbolic conservation law must include both the PDE and the RH condition, yielding:

$$\begin{cases} u_t(\mathbf{x}, t) + \sum\limits_{i=1}^{d} \dfrac{\partial f_i\,(u(\mathbf{x}, t))}{\partial x_i} &= 0, \quad \text{in } (\Omega \times I) \setminus \mathcal{I}, \\ (\mathbf{f}(u^+), u^+) \cdot \mathbf{n}^+ + (\mathbf{f}(u^-), u^-) \cdot \mathbf{n}^- &= 0, \quad \text{on } \mathcal{I}, \end{cases} \tag{2.15}$$

supplemented with the inflow and initial conditions in (2.13).

In practice, however, the interface $\mathcal{I}$ is unknown *a priori*, making it extremely challenging to enforce the RH condition directly in numerical simulations. Traditional methods often rely on entropy conditions or artificial viscosity to approximate physically admissible solutions, but these approaches may introduce numerical artifacts such as smearing or oscillations near discontinuities.

To deal with this difficulty, let us introduce the **total flux**

$$\mathbf{F}(u) = (\mathbf{f}(u), u) = (f_1(u), \ldots, f_d(u), u). \tag{2.16}$$

Denote by **div** the space-time divergence operator. When u is differentiable at $(\mathbf{x}, t)$, the classical definition of the divergence gives:

$$\mathbf{div}\,\mathbf{F}(u(\mathbf{x},t)) = u_t(\mathbf{x},t) + \sum_{i=1}^{d} \frac{\partial f_i\left(u(\mathbf{x},t)\right)}{\partial x_i}. \tag{2.17}$$

To interpret the (2.14) condition, we notice that it is indeed expresses a jump condition in the solution. From the perspective of of continuum mechanics, however, it represents the *continuity of the normal component of the total flux* $\mathbf{F}(u)$ across the interface $\mathcal{I}$, i.e.,

$$[\![\mathbf{F}(u)\cdot\mathbf{n}]\!]_{\mathcal{I}} \equiv \left(\mathbf{f}(u^+), u^+\right)\cdot\mathbf{n}^+\big|_{\mathcal{I}} + \left(\mathbf{f}(u^-), u^-\right)\cdot\mathbf{n}^-\big|_{\mathcal{I}} = 0, \tag{2.18}$$

where $[\![\cdot]\!]_{\mathcal{I}}$ denotes the jump over the interface $\mathcal{I}$ and $\mathbf{n}(\mathbf{x},t)$ is the unit normal vector to the interface. This observation motivates a generalized, weak definition of the divergence operator, applicable even at points where u is discontinuous. Specifically, for any $(\mathbf{x},t)\in\mathcal{I}$, we define the space-time divergence operator via the Gauss divergence theorem as:

$$\mathbf{div}\,\mathbf{F}(u(\mathbf{x},t)) = \lim_{\epsilon\to 0}\frac{1}{|B_\epsilon(\mathbf{x},t)|}\int_{\partial B_\epsilon(\mathbf{x},t)}\mathbf{F}(u)\cdot\mathbf{n}\,dS, \tag{2.19}$$

where $B_\epsilon(\mathbf{x},t)\in\mathbb{R}^{d+1}$ is a ball of radius ϵ centered at $(\mathbf{x},t)$, $\partial B_\epsilon(\mathbf{x},t)$ is the boundary of $B_\epsilon(\mathbf{x},t)$, and $\mathbf{n}$ is the outward unit normal to $\partial B_\epsilon(\mathbf{x},t)$.

THEOREM 2.2. *Let u be a solution of (2.15) and (2.13), then the divergence of the total flux $\mathbf{F}(u)$ vanishes on $\mathcal{I}$, i.e.,*

$$\mathbf{div}\,\mathbf{F}(u) = 0 \quad \textit{in } \mathcal{I}. \tag{2.20}$$

Proof. For any $(\mathbf{x},t)\in\mathcal{I}$, let $B_\epsilon(\mathbf{x},t)$ be a ϵ-ball in $\Omega\times I$ centered at $(\mathbf{x},t)$. Then the interface $\mathcal{I}$ partitions $B_\epsilon(\mathbf{x},t)$ into two subdomains denoted by $B_\epsilon^+(\mathbf{x},t)$ and $B_\epsilon^-(\mathbf{x},t)$ sharing part of $\mathcal{I}$. It follows from (2.18), Gauss' divergence theorem, and the first equation of (2.15) that

$$\begin{aligned}\int_{\partial B_\epsilon(\mathbf{x},t)}\mathbf{F}(u)\cdot\mathbf{n}\,dS &= \int_{\partial B_\epsilon^+(\mathbf{x},t)}\mathbf{F}(u)\cdot\mathbf{n}\,dS + \int_{\partial B_\epsilon^-(\mathbf{x},t)}\mathbf{F}(u)\cdot\mathbf{n}\,dS\\ &= \int_{B_\epsilon^+(\mathbf{x},t)}\mathbf{div}\,\mathbf{F}(u)\,dS + \int_{B_\epsilon^-(\mathbf{x},t)}\mathbf{div}\,\mathbf{F}(u)\,dS = 0,\end{aligned}$$

which, together with the weak definition of the divergence operator in (2.19), implies (2.20). This completes the proof of the lemma. ∎

Using Theorem 2.2 and (2.15), the nonlinear scalar hyperbolic conservation law has the following simplified form

$$\mathbf{div}\,\mathbf{F}(u) = 0 \ \text{ in } \ \Omega\times I\in\mathbb{R}^{d+1}, \tag{2.21}$$

supplemented by the inflow and initial conditions in (2.13).

As the linear case studied in Subsection 2.1, the inflow and initial conditions may be enforced either in the functional as penalization terms or in the solution set or in both the functional and the solution set. To this end, denote the collection of square integrable vector fields whose divergence is also square integrable by

$$H(\mathrm{div};\Omega\times I) = \left\{\boldsymbol{\tau}\in L^2(\Omega\times I)^{d+1} : \mathbf{div}\,\boldsymbol{\tau}\in L^2(\Omega\times I)\right\}.$$

Let us introduce the following two solution sets of (2.21)

$$\begin{cases} \mathcal{V}_{\mathbf{f}} = \left\{ v \in L^2(\Omega \times I) : \mathbf{F}(v) = (\mathbf{f}(v), v) \in H(\mathrm{div}; \Omega \times I) \right\} & \text{and} \\ \mathcal{V}_{\mathbf{f}}(\mathbf{g}) = \left\{ v \in \mathcal{V}_{\mathbf{f}} : v|_{\Gamma_-} = g,\, v|_{\Omega \times \{0\}} = u_0 \right\}, \end{cases} \tag{2.22}$$

where $\mathbf{g} = (g, u_0)$. Define two least-squares (LS) functionals as

$$\hat{\mathcal{L}}(v) = \|\mathbf{div}\, \mathbf{F}(v)\|^2_{0,\Omega \times I} \quad \text{and} \quad \mathcal{L}(v; \mathbf{g}) = \hat{\mathcal{L}}(v) + \|v - g\|^2_{0,\Gamma_-} + \|v - u_0\|^2_{0,\Omega \times \{0\}}, \tag{2.23}$$

where $\|\cdot\|_{0,S}$ denotes the standard $L^2(S)$ norm for $S = \Omega \times I$, Γ_-, or $\Omega \times \{0\}$. Now, the corresponding least-squares formulation is to either (1) seek $u \in \mathcal{V}_{\mathbf{f}}$ such that

$$\mathcal{L}(u; \mathbf{g}) = \min_{v \in \mathcal{V}_{\mathbf{f}}} \mathcal{L}(v; \mathbf{g}) \tag{2.24}$$

or (2) seek $u \in \mathcal{V}_{\mathbf{f}}(\mathbf{g})$ such that

$$\mathcal{L}(u; \mathbf{g}) = \min_{v \in \mathcal{V}_{\mathbf{f}}(\mathbf{g})} \mathcal{L}(v; \mathbf{g}) \quad \text{or} \quad \hat{\mathcal{L}}(u) = \min_{v \in \mathcal{V}_{\mathbf{f}}(\mathbf{g})} \hat{\mathcal{L}}(v). \tag{2.25}$$

Comments on theory and practice of these three LS minimization problems in the previous section are valid here as well.

3. ReLU Neural Network and its Approximation to Discontinuous Functions. This section describes l-hidden-layer ReLU neural network as a set of continuous piecewise linear functions and illustrates its striking approximation power to discontinuous functions with *unknown* interface locations [14, 20].

ReLU refers to the rectified linear activation function defined by

$$\sigma(s) = \max\{0, s\} = \begin{cases} s, & s > 0, \\ 0, & s \le 0. \end{cases} \tag{3.1}$$

The $\sigma(s)$ is a continuous piecewise linear function with one *breaking* point $s = 0$. For $k = 1, \ldots, l$, let n_k denote the number of neurons at the k^{th} hidden layer; denote by

$$\mathbf{b}^{(k)} \in \mathbb{R}^{n_k} \quad \text{and} \quad \boldsymbol{\omega}^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$$

the biases and weights of neurons at the k^{th} hidden layer, respectively. Their i^{th} rows are denoted by $b_i^{(k)} \in \mathbb{R}$ and $\boldsymbol{\omega}_i^{(k)} \in \mathbb{R}^{n_{k-1}}$, that are the bias and weights of the i^{th} neuron at the k^{th} hidden layer, respectively, where $n_0 = d$ or $d + 1$ for applications in Section 2. Let

$$\mathbf{x}^{(0)} = \mathbf{x} \in \mathbb{R}^d \quad \text{or} \quad \mathbf{x}^{(0)} = (\mathbf{x}, t) \in \mathbb{R}^{d+1}.$$

For $k = 1, \ldots, l-1$, define vector-valued functions $\mathbf{x}^{(k)} : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}$ by

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k)}\left(\mathbf{x}^{(0)}\right) = \sigma\left(\boldsymbol{\omega}^{(k)} \mathbf{x}^{(k-1)} + \mathbf{b}^{(k)}\right), \tag{3.2}$$

where the application of the activation function σ to a vector-valued function is defined componentwise.

A ReLU neural network with l hidden layers and n_k neurons at the k^{th} hidden layer can be defined as the collection of continuous piecewise linear functions:

$$\mathcal{M}(l) = \left\{ c_0 + \sum_{i=1}^{n_l} c_i \sigma\left(\boldsymbol{\omega}_i^{(l)} \mathbf{x}^{(l-1)} + b_i^{(l)}\right) : c_i,\, b_i^{(l)} \in \mathbb{R},\, \boldsymbol{\omega}_i^{(l)} \in \mathcal{S}^{n_{l-1}} \right\}, \tag{3.3}$$

where $\mathbf{x}^{(l-1)}\left(\mathbf{x}^{(0)}\right)$ is recursively defined as in (3.2), and $\mathcal{S}^{n_{l-1}}$ denotes the unit sphere in $\mathbb{R}^{n_{l-1}}$. The constraint that the weight vector of each neuron lies on the unit sphere arises from normalization for the ReLU activation function (see [47]). This restriction can narrow the set of solutions for a given approximating problem. The total number of parameters of $\mathcal{M}(l)$ is given by

$$M(l) = (n_l + 1) + \sum_{k=1}^{l} n_k \times (n_{k-1} + 1). \tag{3.4}$$

We now explain why any function $v\left(\mathbf{x}^{(0)}\right)$ in $\mathcal{M}(l)$ is always a continuous piecewise linear function, regardless of the dimension, the number of layers, or the number of neurons per layer. To illustrate this, we first consider the shallow NN case, i.e., $l = 1$. For simplicity, let $\mathbf{x}^{(0)} = \mathbf{x} \in \mathbb{R}^d$, then any NN function $v^{(1)}(\mathbf{x}) \in \mathcal{M}(1)$ has the form of

$$v^{(1)}(\mathbf{x}) = c_0 + \sum_{i=1}^{n_1} c_i \sigma\left(\boldsymbol{\omega}_i^{(1)}\mathbf{x} + b_i^{(1)}\right). \tag{3.5}$$

Since the ReLU activation function $\sigma(s)$ is itself a continuous piecewise linear function with a single *breaking point* at $s = 0$, each neuron $\sigma\left(\boldsymbol{\omega}_i^{(1)}\mathbf{x} + b_i^{(1)}\right)$ is a continuous piecewise linear function with a *breaking hyperplane* (see [14, 47]):

$$\mathcal{P}_i^{(1)} = \left\{\mathbf{x} \in \Omega \subset \mathbb{R}^d : \boldsymbol{\omega}_i^{(1)} \cdot \mathbf{x} + b_i^{(1)} = 0\right\}. \tag{3.6}$$

These hyperplanes, together with the boundary of the domain Ω, generate a *physical partition* $\mathcal{K}^{(1)}$ of Ω as described in [47, 26]. This partition $\mathcal{K}^{(1)}$ is uniquely determined by the parameters of the first layer $\left\{\left(\boldsymbol{\omega}_i^{(1)}, b_i^{(1)}\right)\right\}_{i=1}^{n_1}$ and consists of irregular, polygonal subdomains of Ω (see Figure 3(e) below for a visual example for such partition). The NN function $v^{(1)}(\mathbf{x})$ in (3.5) is then a continuous piecewise linear function with respect to $\mathcal{K}^{(1)}$.

Any NN function $v^{(2)}(\mathbf{x})$ in $\mathcal{M}(2)$ has the form

$$v^{(2)}(\mathbf{x}) = c_0 + \sum_{i=1}^{n_2} c_i \sigma\left(\boldsymbol{\omega}_i^{(2)}\mathbf{x}^{(1)} + b_i^{(2)}\right), \tag{3.7}$$

where $\mathbf{x}^{(1)} = \sigma\left(\boldsymbol{\omega}^{(1)}\mathbf{x} + \mathbf{b}^{(0)}\right) \in \mathcal{M}(1)^{n_1}$ is a vector-valued function whose components are continuous piecewise linear functions. Hence, each pre-activated NN function

$$g_i^{(2)}(\mathbf{x}) = \boldsymbol{\omega}_i^{(2)}\mathbf{x}^{(1)} + b_i^{(2)},$$

being an affine function of $\mathbf{x}^{(1)}$, is also a continuous piecewise linear function of $\mathbf{x}$, defined with respect to the physical partition $\mathcal{K}^{(1)}$. The zero level set of $g_i^{(2)}(\mathbf{x})$, denoted by

$$\mathcal{P}_i^{(2)} = \left\{\mathbf{x} \in \Omega \subset \mathbb{R}^d : \boldsymbol{\omega}_i^{(2)}\mathbf{x}^{(1)} + b_i^{(2)} = 0\right\} \tag{3.8}$$

defines a breaking poly-hyperplane that refines the partition $\mathcal{K}^{(1)}$ (see [26] for details). It follows that $v^{(2)}(\mathbf{x})$ is a continuous piecewise linear function with respect to the refined physical partition $\mathcal{K}^{(2)}$, generated by incorporating the breaking poly-hyperplanes $\left\{\mathcal{P}_i^{(2)}\right\}_{i=1}^{n_2}$ into $\mathcal{K}^{(1)}$. By induction, any ReLU NN function $v^{(l)}(\mathbf{x}) \in \mathcal{M}(l)$ is a a continuous piecewise linear function with respect to the physical partition $\mathcal{K}^{(l)}$, which is determined by the network parameters across all hidden layers.

One of the key challenges in numerically solving HCLs lies in the fact that the location of the discontinuities in the solution is typically unknown in advance. To capture these unknown interfaces, traditional mesh-based numerical methods require fine meshes, whick leads to high computational cost and often produces spurious oscillations near the interface, an artifact commonly referred to as the Gibbs phenomenon. In the remainder of this section, we highlight a remarkable approximation property of ReLU NN functions for representing piecewise constant functions with unknown discontinuity interfaces.

To this end, let $\chi(\mathbf{x})$ be a piecewise constant function defined on $\Omega \in \mathbb{R}^d$, given by

$$\chi(\mathbf{x}) = \sum_{i=1}^{k} \alpha_i \mathbf{1}_{\Omega_i}(\mathbf{x}), \tag{3.9}$$

where $\{\Omega_i\}_{i=1}^k$ forms a partition of the domain Ω and $\mathbf{1}_{\Omega_i}(\mathbf{x})$ denotes the indicator function of the subdomain Ω_i. For simplicity of presentation, consider a special case that $k = 2$, $\alpha_1 = 0$, and $\alpha_2 = 1$, i.e.,

$$\chi(\mathbf{x}) = \mathbf{1}_{\Omega_2}(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in \Omega_1, \\ 1, & \mathbf{x} \in \Omega_2, \end{cases} \tag{3.10}$$

where Ω_1 and Ω_2 are open, connected subdomains of Ω satisfying

$$\Omega_1 \cap \Omega_2 = \emptyset \quad \text{and} \quad \bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2.$$

Let $\partial\Omega_i$ be the boundary of the subdomain Ω_i, and assume the interface $\mathcal{I} = \partial\Omega_1 \cap \partial\Omega_2$ is a C^0 surface with finite $(d-1)$-dimensional measure, i.e., $|\mathcal{I}| < \infty$. Although this case only considers a single interface, the extension to general piecewise constant functions as in (3.9) with multiple discontinuities is straightforward.

The complexity of the discontinuity interface $\mathcal{I}$ plays a crucial role in determining both the accuracy of the approximation and the required architecture of NN. To gain a clear understanding, we begin by examining the simplest non-trivial case, when the interface is a hyperplane, and defer a discussion of more general interfaces to Remark 3.2. Specifically, consider the case where the interface $\mathcal{I}$ is a portion of a hyperplane defined by

$$\mathcal{I} = \left\{ \mathbf{x} \in \Omega \subset \mathbb{R}^d : \mathbf{a} \cdot \mathbf{x} = b \right\},$$

where $\mathbf{a}$ is the unit vector normal to $\mathcal{I}$ pointing to the subdomain Ω_2. To approximate the step function $\chi(\mathbf{x})$ associated with this hyperplane interface, we introduce two representative ReLU NN constructions, both capable of capturing the discontinuity sharply and efficiently. For any given $\varepsilon > 0$, define the following NN functions as in [14] and [20]:

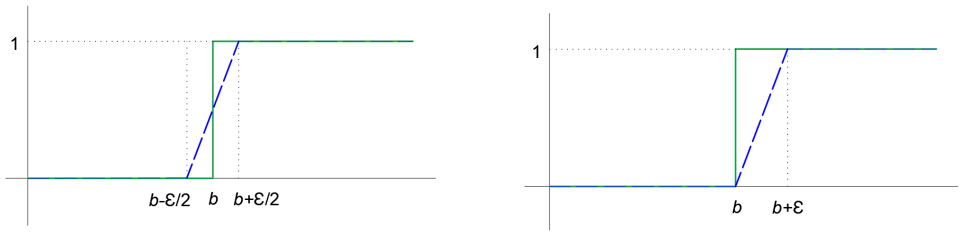
$$p_1(\mathbf{x}) = \frac{1}{2\varepsilon}\big(\sigma(\mathbf{a}\cdot\mathbf{x} - b + \varepsilon) - \sigma(\mathbf{a}\cdot\mathbf{x} - b - \varepsilon)\big) \quad \text{or} \quad p_2(\mathbf{x}) = 1 - \sigma\left(-\frac{1}{\varepsilon}\sigma(\mathbf{a}\cdot\mathbf{x} - b) + 1\right). \tag{3.11}$$

Here, $p_1(\mathbf{x})$ is achieved by a two-layer ReLU NN with just two neurons, while $p_2(\mathbf{x})$ corresponds to a three-layer NN with one neuron per hidden layer. In one dimension, they are illustrated in Figure 1. Define the narrow transition regions as

$$\Omega_{p_1} = \{\mathbf{x} \in \Omega : -\varepsilon < \mathbf{a} \cdot \mathbf{x} - b < \varepsilon\} \quad \text{and} \quad \Omega_{p_2} = \{\mathbf{x} \in \Omega : 0 < \mathbf{a} \cdot \mathbf{x} - b < \varepsilon\}.$$

Then, both $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ serve as smoothed approximations to the ideal step function $\chi(\mathbf{x})$, with the following piecewise-defined formula:

$$p_1(\mathbf{x}) = \begin{cases} 0, & \text{in } \Omega_1 \setminus \Omega_{p_1}, \\ \dfrac{\mathbf{a}\cdot\mathbf{x} - b + \varepsilon}{2\varepsilon}, & \text{in } \Omega_{p_1}, \\ 1, & \text{in } \Omega_2 \setminus \Omega_{p_1} \end{cases} \quad \text{and } p_2(\mathbf{x}) = \begin{cases} 0, & \text{in } \Omega_1 \setminus \Omega_{p_2}, \\ (\mathbf{a}\cdot\mathbf{x} - b)/\varepsilon, & \text{in } \Omega_{p_2}, \\ 1, & \text{in } \Omega_2 \setminus \Omega_{p_2}. \end{cases} \tag{3.12}$$



(a) $p_1(\mathbf{x})$ (b) $p_2(\mathbf{x})$

FIG. 1. *NN Approximation of a 1D unit step function.*

LEMMA 3.1. *There exists a positive constant C such that for all $r \in (0, \infty)$, we have*

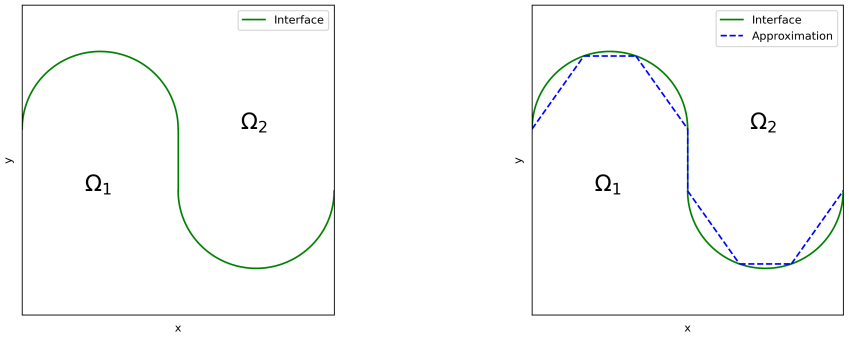
$$\|\chi - p_1\|_{L^r(\Omega)} \leq C\,|\mathcal{I}|^{1/r}\varepsilon^{1/r} \quad \textit{and} \quad \|\chi - p_2\|_{L^r(\Omega)} \leq C\,|\mathcal{I}|^{1/r}\varepsilon^{1/r}, \tag{3.13}$$

where $|\mathcal{I}|$ is the $(d-1)$-dimensional measure of the interface $\mathcal{I}$.

Proof. By (3.10) and (3.12), for $i = 1, 2$, we have

$$\chi(\mathbf{x}) - p_i(\mathbf{x}) = 0, \text{ in } \Omega \setminus \Omega_{p_i} \quad \text{and} \quad \big|\chi(\mathbf{x}) - p_i(\mathbf{x})\big|^r \leq 1, \text{ in } \Omega_{p_i},$$

which, together with the facts that $\big|\Omega_{p_i}\big| \leq C\,\big|\mathcal{I}\big|\varepsilon$, implies the validity of (3.13). This completes the proof of the lemma. ∎



(a) The interface Γ (b) An approximation of the interface by connected series of hyperplanes

FIG. 2. *Approximation of the interface Γ*

These explicit constructions demonstrate the expressive power of ReLU NNs in approximating discontinuous functions, with simple architectures sufficing to localize and resolve interfaces aligned with hyperplanes. In the next remark, we discuss the extension of this approach to more general, non-planar interfaces.

REMARK 3.2. *When the interface $\mathcal{I}$ is not a hyperplane but can be approximated, to a prescribed accuracy $\varepsilon > 0$, by a connected sequence of hyperplanes (see Figure 2 and [20]) in the maximum norm, the indicator function $\chi(\mathbf{x}) = \mathbf{1}_{\Omega_2}(\mathbf{x})$ defined in (3.10) can still be effectively approximated by a ReLU NN of a given architecture, satisfying the error bound in (3.13).*

More precisely, based on the two-layer ReLU NN approximation $p_1(\mathbf{x})$ in (3.11), we showed in [18] that a ReLU NN with at most $\lceil \log_2(d+1) \rceil + 1$ layers suffices to achieve the desired

approximation accuracy ε. However, [18] *does not provide an estimate on the minimum number of neurons at each layer. This result is obtained by constructing a continuous piecewise linear (CPWL) function with a sharp transition layer of ε width, and combining the main findings of [1] regarding the ReLU NN representation of CPWL functions.*

Leveraging the three-layer ReLU NN approximation $p_2(\mathbf{x})$ in (3.11), it was shown in [20] *that $\chi(\mathbf{x})$ can be approximated with the comparable accuracy using a three-layer ReLU NN. In this case, the number and locations of neurons at the first layer are determined by the hyperplanes used to approximate the interface, while the number of neurons in the second layer is dependent on the convexity of the interface* (*see* Theorem 3.2 *in* [20]).

4. Least-Squares Neural Network (LSNN) Method. This section introduces the least-squares neural network (LSNN) method for solving advection-reaction equations in (2.1) and scalar nonlinear hyperbolic conservation laws in (2.21) based on the equivalent least-squares formulations in (2.8)/(2.10) and (2.24)/(2.25), respectively. To evaluate the least-squares functionals, we discuss efficient numerical integration in Subsection 4.1 and physics-preserved numerical differentiation in Subsection 4.2. Finally, the LSNN method is defined in Subsection 4.3.

4.1. Numerical Integration. The evaluation of the least-squares functionals $\mathcal{L}(v;\mathbf{f})$ and $\hat{\mathcal{L}}(v;\mathbf{f})$, as defined in (2.7) or (2.23), involves integrations over the computational domain $\Omega \subset \mathbb{R}^d$ or $\Omega \times I \subset \mathbb{R}^{d+1}$ ($d = 1, 2,$ or 3), as well as portions of their boundaries. In practice, these integrals must be approximated using numerical quadrature. This section outlines the basic setup for numerical integration and discusses key considerations for its implementation within the LSNN framework.

Let

$$\mathcal{T} = \{K : K \text{ is an open subdomain of } \Omega\}$$

denote a partition of the domain Ω, referred to as the *integration mesh*. Here, the partition means that the union of all subdomains of $\mathcal{T}$ equals the whole domain Ω and that any two distinct subdomains of $\mathcal{T}$ have no intersection; more precisely,

$$\bar{\Omega} = \cup_{K\in\mathcal{T}} \bar{K} \quad \text{and} \quad K \cap T = \emptyset, \quad \forall\ K, T \in \mathcal{T}.$$

A composite quadrature over $\mathcal{T}$ is then written as

$$\sum_{K\in\mathcal{T}} \mathcal{Q}_K(w) \approx \sum_{K\in\mathcal{T}} \int_K w(\mathbf{x})\, d\mathbf{x} = \int_\Omega w(\mathbf{x})\, d\mathbf{x},$$

where $\mathcal{Q}_K(w) \approx \int_K w(\mathbf{x})\, d\mathbf{x}$ represents a quadrature rule over subdomain K. The specific quadrature rule $\mathcal{Q}_K$ may vary across different elements $K \in \mathcal{T}$, and can be chosen from standard formulas such as Gaussian quadrature or Newton–Cotes formulas, including the midpoint, trapezoidal, or Simpson's rule (see [56]). For instance, using the midpoint rule for all $K \in \mathcal{T}$, $\mathcal{Q}_K(w) = w(\mathbf{x}_K)|K|$, where $\mathbf{x}_K$ is the centroid of element K and $|K|$ denotes its d- or $(d+1)$-dimensional measure.

In the context of LSNN, the integrands typically depend on NN approximations of the solution u to the underlying PDE. These NN approximations are continuous piecewise linear functions defined over physical partitions (see Section 3), which are generally unknown in advance and evolve dynamically during training. Moreover, the true solution u itself is unknown and may exhibit localized features such as steep gradients, discontinuities, or singularities.

Because of these considerations, adaptive numerical integration was introduced in [47] (see Algorithm 5.2) and in [48] (see Algorithm 3.1). Below, we briefly outline the adaptive mesh refinement strategy for numerical integration with a fixed NN in Algorithm 4.1 for problem (2.1).

As usual, we begin with a uniform and coarse partition $\mathcal{T}'$ of the domain Ω. Based on this initial integration mesh, we apply the standard adaptive numerical quadrature procedure to refine

it according to the given integrands f and g. The goal is to construct an initial integration partition $\mathcal{T}$ such that

$$\left|\int_\Omega f^2\,d\mathbf{x} - \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left(f^2\right)\right| \le \varepsilon \quad \text{and} \quad \left|\int_{\Gamma_-} g^2 dS - \sum_{E\in\mathcal{E}_-} \mathcal{Q}_E\left(g^2\right)\right| \le \varepsilon,$$

where $\mathcal{E}_-$ denotes the collection of inflow boundary faces of $\mathcal{T}$ (see (4.3)).

Let u_τ be the NN approximation obtained using this initial integration mesh $\mathcal{T}$. For each subdomain $K \in \mathcal{T}$, we define a local error indicator

$$\eta_K = \|D_{\boldsymbol{\beta},\tau} u_\tau + \gamma\, u_\tau - f\|_{0,K},$$

where $D_{\boldsymbol{\beta},\tau}$ is a discrete directional derivative operator as defined in (4.1). The global error estimator is then given by $\eta = \left(\sum_{K\in\mathcal{T}} \eta_K^2\right)^{1/2}$. These two error indicators guide the adaptive refinement process, which iteratively updates the integration mesh to better resolve regions with large local error contributions. The refinement procedure is summarized in Algorithm 4.1.

Algorithm 4.1 Adaptive Quadrature Refinement (AQR) with a fixed NN.

(1) for each $K \in \mathcal{T}$, compute the local error indicator η_K;
(2) mark $\mathcal{T}$ by the either bulk or average marking strategy (see, e.g., [47]) and refine marked subdomain to obtain a new partition $\mathcal{T}'$;
(3) compute new NN approximation $u_{\mathcal{T}'}$ on the refined integration mesh $\mathcal{T}'$;
(4) for a given parameter $\gamma \in (0,1)$, if $\eta(u_{\mathcal{T}'}) \le \gamma\eta(u_\mathcal{T})$, then go to Step (1) with $\mathcal{T} = \mathcal{T}'$; otherwise, output $\mathcal{T}$.

For a given parameter $\gamma \in (0,1)$, the stopping criterion used in Step (4) of Algorithm 4.1 indicates that the adaptive procedure terminates when further refinement of the integration mesh produces a negligible reduction in residual. In other words, if the additional refinement does not significantly improve the accuracy of the global error evaluation for the current NN approximation, the algorithm halts and outputs the current integration partition.

REMARK 4.1. *In the case where computational cost is not an issue, one may use a uniform partition $\mathcal{T}$ that is fine enough to approximate the unknown solution well using a piecewise polynomial.*

4.2. Physics-Preserved Numerical Differentiation. Solutions to the hyperbolic PDEs in (2.1) and (2.12) can exhibit discontinuities, making conventional numerical or auto-differentiations along coordinate directions, based directly on (2.1) and (2.12), inadequate. This section introduces physics-preserved numerical differentiation techniques derived from the equivalent formulations in (2.5) and (2.21), as proposed in [14, 16].

When the solution u is discontinuous, as discussed in Subsection 2.1 and Subsection 2.2, both the directional derivative $u_{\boldsymbol{\beta}}(\mathbf{x})$ and the divergence of the total flux $\mathbf{div}\,\mathbf{F}(u)$ can still be defined via limit processes, as given in (2.4) and (2.19), respectively. Approximating these quantities through any reasonable limiting process leads to what we call *physics-preserved numerical differentiation*.

Based on (2.4), for any function $v(\mathbf{x})$ that admits a directional derivative in the $\boldsymbol{\beta}$ direction at $\mathbf{x} \in \Omega$, we define the discrete directional differential operator $D_{\boldsymbol{\beta},\tau}$ as

$$D_{\boldsymbol{\beta},\tau}\, v(\mathbf{x}) = \frac{v(\mathbf{x}) - v\big(\mathbf{x} - \tau\boldsymbol{\beta}(\mathbf{x})\big)}{\tau} \tag{4.1}$$

for $0 < \tau \ll 1$. This operator provides an upwind finite difference approximation to the directional derivative in the direction of $\boldsymbol{\beta}$ with step size τ. Intuitively, for sufficiently small τ, this scheme

avoids crossing the discontinuity interface, thereby preserving the physical meaning of the directional derivative. In contrast, standard numerical or automatic differentiation based on $\sum_{i=1}^{d} \beta_i u_{x_i}$, where u_{x_i} denotes the partial derivative of u with respect to x_i, involves values from both sides of the interface and thus fails to capture the correct behavior at discontinuities. This can be illustrated by a simple model problem in $\Omega = (0,1)\times(0,1) \subset \mathbb{R}^2$ with $\boldsymbol{\beta} = (1,1)^T$. If the inflow boundary condition g is discontinuous at the corner point $(0,0)$, the resulting discontinuity interface is $\mathcal{I} = \{(x,t) \in \Omega : x+t = 0\}$. In this scenario, the standard differentiation $u_x + u_t$ necessarily evaluates across the interface $\mathcal{I}$, leading to non-physical or spurious results.

To define a discrete divergence operator based on (2.19), we associate each integration point $\mathbf{z}$ with a corresponding control volume $K_{\mathbf{z}}$ that contains the point. The discrete divergence of the total flux $\mathbf{F}(v)$ at $\mathbf{z}$ is then given by

$$\mathbf{div}_{\mathcal{T}}\mathbf{F}(v(\mathbf{z})) = \frac{1}{|K_{\mathbf{z}}|}\mathcal{Q}_{\partial K_{\mathbf{z}}}\left(\mathbf{F}(v)\cdot\mathbf{n}\right), \tag{4.2}$$

where $\mathcal{Q}_{\partial K_{\mathbf{z}}}(\cdot)$ denotes a *composite quadrature rule* applied over the boundary $\partial K_{\mathbf{z}}$ of the control volume $K_{\mathbf{z}}$ and $\mathbf{n}$ is the unit outward vector normal to the boundary $\partial K_{\mathbf{z}}$.

Under the midpoint quadrature rule $\mathcal{Q}_K$, each subdomain $K \in \mathcal{T}$ has a single integration point $\mathbf{z}_K = (\mathbf{x}_K, t_K)$, typically chosen as the centroid of K. In this case, the control volume is simple $K_{\mathbf{z}_K} = K$.

More generally, suppose the quadrature rule $\mathcal{Q}_K$ for a subdomain $K \in \mathcal{T}$ has J integration points,

$$\mathbf{z}_{K_j} = (\mathbf{x}_{K_j}, t_{K_j}) \in K \in \mathcal{T}, \quad \text{for } j = 1,\ldots,J.$$

Let $\mathcal{T}_K = \{K_j\}_{j=1}^{J}$ be a partition of K such that $\mathbf{z}_{K_j} \in K_j$, where K_j is referred to as the control volume of the integration point $\mathbf{z}_{K_j}$. Let $\mathcal{Q}_{\partial K_j}(\cdot)$ be a composite quadrature rule over the boundary ∂K_j, then the discrete divergence operator $\mathbf{div}_{\mathcal{T}}$ at the integration point $\mathbf{z}_{K_j}$ can be defined analogously via (4.2).

The general definition of the discrete divergence operator $\mathbf{div}_{\mathcal{T}}$ in (4.2) depends on the quadrature rule $\mathcal{Q}_{\partial K}(\cdot)$ over the boundary ∂K, which, in turn, depends on the geometry of K. Since the integration mesh $\mathcal{T}$ is independent of the physical mesh induced by the NN approximation, it is practically advantageous to construct $\mathcal{T}$ as a composite mesh generated by the AQR procedure in Algorithm 4.1. In this setup, each element $K \in \mathcal{T}$ is taken to be simple cell such as a rectangle, a cuboid, or a hypercube in two, three, or higher dimensions, with all faces aligned parallel to the coordinate hyperplanes. For such integration mesh $\mathcal{T}$ in both two and three dimensions, explicit definitions of $\mathbf{div}_{\mathcal{T}}\mathbf{F}(u(\mathbf{z}_K))$ were proposed and rigorously analyzed in [16], specifically in the context of discontinuous solutions u. These formulations ensure that the numerical divergence remains consistent with the underlying physics, even across discontinuities and sharp interface features.

4.3. The LSNN Method. Denote the collections of the inflow boundary faces and the initial-time faces of the integration mesh $\mathcal{T}$ by

$$\mathcal{E}_- = \{E = \partial K \cap \Gamma_- : K \in \mathcal{T}\} \text{ and } \mathcal{E}_0 = \{E = \partial K \cap (\Omega\times\{0\}) : K \in \mathcal{T}\}, \tag{4.3}$$

respectively. For each face E in $\mathcal{E}_-$ or $\mathcal{E}_0$, let $\mathcal{Q}_E(w)$ denote a quadrature rule applied to an integrand w defined on E. We now define the discrete least-squares functionals. For problem (2.5), the functional is defined by

$$\hat{\mathcal{L}}_{\mathcal{T}}(v) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left((D_{\boldsymbol{\beta},\mathcal{T}}v + \gamma v - f)^2\right) \text{ and } \mathcal{L}_{\mathcal{T}}(v;\mathbf{f}) = \hat{\mathcal{L}}_{\mathcal{T}}(v) + \sum_{E\in\mathcal{E}_-}\mathcal{Q}_E\left(|\boldsymbol{\beta}\cdot\boldsymbol{n}|(v-g)^2\right) \tag{4.4}$$

and for problem (2.21), it is defined as

$$\begin{cases} \hat{\mathcal{L}}_\tau(v) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left(\mathbf{div}_\tau \mathbf{F}(v)\right) & \text{and} \\ \mathcal{L}_\tau(v;\mathbf{f}) = \hat{\mathcal{L}}_\tau(v) + \sum_{E\in\mathcal{E}_-} \mathcal{Q}_E\left((v-g)^2\right) + \sum_{E\in\mathcal{E}_0} \mathcal{Q}_E\left((v-u_0)^2\right) \end{cases} \tag{4.5}$$

To unify the notation across both formulations, let

$$\mathcal{V} = \mathcal{V}_{\boldsymbol{\beta}} \text{ or } \mathcal{V}_{\mathbf{f}} \quad \text{and} \quad \mathcal{V}(h) = \mathcal{V}_{\boldsymbol{\beta}}(g) \text{ or } \mathcal{V}_{\mathbf{f}}(\mathbf{g})$$

for problem (2.5) or (2.21), respectively. The LSNN method then seeks a function $u_{N,\mathcal{T}} \in \mathcal{M}(l)$ that minimizes the discrete least-squares functional. Specifically, it solves either

$$\mathcal{L}_\tau\left(u_{N,\mathcal{T}};\mathbf{f}\right) = \min_{v\in\mathcal{M}(l)} \mathcal{L}_\tau(v;\mathbf{f}). \tag{4.6}$$

or when constrained to the admissible set $\mathcal{V}(h_\tau)$, solves,

$$\mathcal{L}_\tau\left(u_{N,\mathcal{T}};\mathbf{f}\right) = \min_{v\in\mathcal{M}(l)\cap\mathcal{V}(h_\tau)} \mathcal{L}(v;\mathbf{f}) \quad \text{or} \quad \hat{\mathcal{L}}_\tau\left(u_{N,\mathcal{T}}\right) = \min_{v\in\mathcal{M}(l)\cap\mathcal{V}(h_\tau)} \hat{\mathcal{L}}(v), \tag{4.7}$$

where h_τ is an approximation of the prescribed data $h = g$ or $\mathbf{g}$.

We now discuss how to weakly enforce the inflow boundary and initial conditions through the physics-preserved numerical differentiation in Subsection 4.2. For simplicity, assume that the quadrature rule $\mathcal{Q}_K(\cdot)$ is the midpoint rule. In this case, the centroid of K, $\mathbf{z}_K = \mathbf{x}_K$ or $(\mathbf{x}_K, t_K)$, is the sole integration point within each subdomain K. For each inflow boundary or initial face $E \in \mathcal{E}_- \cup \mathcal{E}_0$, there exists a subdomain $K \in \mathcal{T}$ such that $E \in \partial K$. We denote this face by E_K to indicate that E is part of the boundary ∂K of K.

For each inflow boundary face $E_K \in \mathcal{E}_-$, the directional derivative $D_{\boldsymbol{\beta},\tau} v(\mathbf{x}_K)$, as defined in (4.1), is computed by choosing the derivative step size τ so that the backward point $\mathbf{x}_K - \tau\boldsymbol{\beta}(\mathbf{x}_K)$ lies on the boundary face $\mathcal{E}_-$. The directional derivative is then given by

$$D_{\boldsymbol{\beta},\tau} v(\mathbf{x}_K) = \frac{v(\mathbf{x}_K) - g\big(\mathbf{x}_K - \tau\boldsymbol{\beta}(\mathbf{x}_K)\big)}{\tau}, \tag{4.8}$$

where g is the prescribed inflow boundary condition in (2.1). Similarly, for problem (2.12), the discrete divergence operator at the integration point $\mathbf{z}_K$ is modified to incorporate the boundary or initial data on the relevant face E_K. Specifically, the discrete divergence becomes

$$\mathbf{div}_\tau \mathbf{F}(u(\mathbf{z}_K)) = \begin{cases} \frac{1}{|K|}\left(\mathcal{Q}_{\partial K\setminus E_K}\left(\mathbf{F}(u)\cdot\mathbf{n}\right) + \mathcal{Q}_{E_K}\left(\mathbf{F}(g)\cdot\mathbf{n}\right)\right), & E_K \in \mathcal{E}_-, \\ \frac{1}{|K|}\left(\mathcal{Q}_{\partial K\setminus E_K}\left(\mathbf{F}(u)\cdot\mathbf{n}\right) + \mathcal{Q}_{E_K}\left(\mathbf{F}(u_0)\cdot\mathbf{n}\right)\right), & E_K \in \mathcal{E}_0. \end{cases} \tag{4.9}$$

where g and u_0 denote inflow boundary and initial conditions in (2.12), respectively.

5. Efficient and Reliable Iterative Solvers. Both $\mathcal{L}_\tau(v;\mathbf{f})$ and $\hat{\mathcal{L}}_\tau(v)$ are convex functionals with respect to v, but become non-convex when viewed as functions of the NN parameters. As a result, the discrete problems formulated in (4.6) or (4.7) are non-convex optimization problems over the NN parameters. These high-dimensional, non-convex optimization problems are typically computationally intensive and complex, and they remain a major bottleneck in applying NNs to numerically solve PDEs. Despite these challenges, there has been active research and some notable progress in developing more efficient and reliable iterative solvers (training algorithms) and in designing effective initialization strategies for NN-based PDE methods [21, 22, 23].

The Adam (Adaptive Moment Estimation) method [42] is a first-order gradient-based optimization algorithm widely used in training NNs. At each iteration, Adam updates the NN parameters using an adaptive learning rate based on estimates of the first and second moments of the gradients. This adaptive learning rate mechanism makes it particularly effective for large-scale optimization problems with noisy or sparse gradients. However, for solving high-dimensional, non-convex optimization problems arising from NN-based PDE discretizations, first-order methods like Adam often suffer from slow convergence, motivating the development of more efficient second-order solvers that can better exploit the structure of the optimization landscape.

Below, we first explore the algebraic structure of the problem and then introduce a second-order Gauss-Newton based iterative method. As a nonlinear PDE, (2.21) possesses its own intrinsic nonlinearity that warrants special treatment. In this section, we focus solely on the linear problem in (2.5). For simplicity of presentation, we use the second minimization problem defined in (4.7), namely the task of finding $u^*_{N,\mathcal{T}}(\mathbf{x}) \in \mathcal{M}(l)$ such that

$$u^*_{N,\mathcal{T}}(\mathbf{x}) = \underset{v \in \mathcal{M}(l) \cap \mathcal{V}(h_{\mathcal{T}})}{\arg\min} \hat{\mathcal{L}}(v) \tag{5.1}$$

where $\hat{\mathcal{L}}(v)$ defined in (4.4), as an example to illustrate the algebraic structures underlying this non-convex optimization problem. These structures can be leveraged for designing more efficient and reliable iterative solvers.

Problem (5.1) is often called separable nonlinear least-squares (SNLS) problem (see, e.g., [41]), due to the presence of both the linear and nonlinear parameters. There are two approaches for solving SNLS problems: (1) block iterative methods that alternate between updates of the linear and the nonlinear parameters (serving as outer iteration), and (2) the Variable Projection (VarPro) method of Golub-Pereyra [38], which eliminates the linear parameters to reduce the dimensionality of the problem. Although the VarPro method yields a problem with fewer degrees of freedom, it alters the original nonlinear structure of the SNLS problem. Therefore, in this work, we will focus exclusively on the first approach.

To this end, any function $v(\mathbf{x}) \in \mathcal{M}(l)$ has the following form

$$v(\mathbf{x}) = c_0 + \sum_{i=1}^{n_l} c_i \sigma\left(\boldsymbol{\omega}_i^{(l)} \mathbf{x}^{(l-1)} + b_i^{(l)}\right), \tag{5.2}$$

where $\mathbf{x}^{(l-1)}$ is recursively defined in (3.2). Clearly, $v(\mathbf{x})$ is determined by the linear and nonlinear parameters

$$\hat{\mathbf{c}} = (c_0, \mathbf{c}) = (c_0, c_1, \ldots, c_{n_l}) \in \mathbb{R}^{n_l+1} \text{ and } \boldsymbol{\Theta} = \left\{\mathbf{r}^{(k)}\right\}_{k=1}^{l} = \left\{\left(\mathbf{r}_1^{(k)}, \ldots, \mathbf{r}_{n_k}^{(k)}\right)^T\right\}_{k=1}^{l},$$

respectively, where $\mathbf{r}_i^{(k)} = \left(b_i^{(k)}, \boldsymbol{\omega}_i^{(k)}\right)$ is the bias and weights of the i^{th} neuron at the k^{th} hidden layer. To indicate this dependence, we often write $v(\mathbf{x}) = v(\mathbf{x}; \hat{\mathbf{c}}, \boldsymbol{\Theta})$. Now, a solution

$$u^*_{N,\mathcal{T}}(\mathbf{x}) = u_{N,\mathcal{T}}(\mathbf{x}; \hat{\mathbf{c}}^*, \boldsymbol{\Theta}^*)$$

of the problem in (5.1) is to minimize the least-squares functional $\hat{\mathcal{L}}(v)$ over $\mathcal{M}(l) \cap \mathcal{V}(h_{\mathcal{T}})$; or equivalently, the optimal parameters $(\hat{\mathbf{c}}^*, \boldsymbol{\Theta}^*)$ minimize the function $\hat{\mathcal{L}}\left(v(\cdot; \hat{\mathbf{c}}, \boldsymbol{\Theta})\right)$ of variables $(\hat{\mathbf{c}}, \boldsymbol{\Theta})$ in a domain of high dimension. Hence, $(\hat{\mathbf{c}}^*, \boldsymbol{\Theta}^*)$ satisfies the following optimality conditions

$$\nabla_{\mathbf{c}} \hat{\mathcal{L}}_{\mathcal{T}}\left(u_{N,\mathcal{T}}(\cdot; \hat{\mathbf{c}}^*, \boldsymbol{\Theta}^*)\right) = \mathbf{0} \quad \text{and} \quad \nabla_{\boldsymbol{\Theta}} \hat{\mathcal{L}}_{\mathcal{T}}\left(u_{N,\mathcal{T}}(\cdot; \hat{\mathbf{c}}^*, \boldsymbol{\Theta}^*)\right) = \mathbf{0}, \tag{5.3}$$

where $\nabla_{\hat{\mathbf{c}}}$ and $\nabla_{\boldsymbol{\Theta}}$ denote the gradients with respect to $\hat{\mathbf{c}}$ and $\boldsymbol{\Theta}$, respectively.

Below we discuss algebraic structures of (5.3). First, this is a coupled nonlinear system of algebraic equations on $\hat{\mathbf{c}}^*$ and $\boldsymbol{\Theta}^*$. Next, we derive specific forms of these algebraic equations. To do so, let $\sigma_0(\mathbf{x}) = 1$ and $\sigma_i(\mathbf{x}) = \sigma\left(\boldsymbol{\omega}_i^{(l)}\mathbf{x}^{(l-1)} + b_i^{(l)}\right)$ for $i = 1, \ldots, n_l$, and set

$$\boldsymbol{\Sigma}(\mathbf{x}) = (\sigma_1(\mathbf{x}), \ldots, \sigma_{n_l}(\mathbf{x}))^T \quad \text{and} \quad \hat{\boldsymbol{\Sigma}}(\mathbf{x}) = (\sigma_0(\mathbf{x}), \sigma_1(\mathbf{x}), \ldots, \sigma_{n_l}(\mathbf{x}))^T.$$

Since the ReLU activation function $\sigma(s)$ is not point-wisely differentiable, it is then difficult to compute gradients of the discrete LS functional $\hat{\mathcal{L}}_\tau\left(v(\cdot; \hat{\mathbf{c}}, \boldsymbol{\Theta})\right)$. Instead, we introduce an intermediate functional before numerical integration,

$$\hat{\mathcal{L}}_\tau\left(v(\cdot; \hat{\mathbf{c}}, \boldsymbol{\Theta})\right) = \int_\Omega \left(D_{\boldsymbol{\beta},\tau} v + \gamma v - f\right)^2 d\mathbf{x}, \tag{5.4}$$

compute gradients of this functional, and then approximate the coefficient matrix and the right-hand side vector by numerical quadrature.

Because $\sigma(s)$ has the first order weak derivative, by the facts that $v(\mathbf{x}) = \hat{\boldsymbol{\Sigma}}(\mathbf{x})^T\hat{\mathbf{c}}$ and that $\nabla_{\hat{\mathbf{c}}} v(\mathbf{x}; \hat{\mathbf{c}}, \boldsymbol{\Theta}) = \hat{\boldsymbol{\Sigma}}(\mathbf{x})$, we have

$$\frac{1}{2}\nabla_{\hat{\mathbf{c}}}\hat{\mathcal{L}}_\tau\left(v(\cdot; \hat{\mathbf{c}}, \boldsymbol{\Theta})\right) = \int_\Omega \left(D_{\boldsymbol{\beta},\tau}\, v + \gamma\, v - f\right) \nabla_{\hat{\mathbf{c}}}\left(D_{\boldsymbol{\beta},\tau} + \gamma\right) v\, d\mathbf{x}$$

$$= \int_\Omega \left(D_{\boldsymbol{\beta},\tau}\, v + \gamma\, v - f\right)\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}(\mathbf{x}) d\mathbf{x} = \int_\Omega \left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}(\mathbf{x})\left\{\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}(\mathbf{x})^T\hat{\mathbf{c}} - f\right\} d\mathbf{x},$$

which, together with the optimality condition, implies

$$\mathbf{0} = \frac{1}{2}\nabla_{\hat{\mathbf{c}}}\hat{\mathcal{L}}_\tau\left(v(\cdot; \hat{\mathbf{c}}, \boldsymbol{\Theta})\right) = \int_\Omega \left\{\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}\right]\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}^T\right]\hat{\mathbf{c}} - f\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}\right]\right\} d\mathbf{x}. \tag{5.5}$$

Approximating the integral by numerical quadrature, then the first equation in (5.3) implies the following system of algebraic equations

$$\boldsymbol{A}\left(\boldsymbol{\Theta}\right)\hat{\mathbf{c}} = F\left(\boldsymbol{\Theta}\right), \tag{5.6}$$

where $\boldsymbol{A}\left(\boldsymbol{\Theta}\right)$ and $F\left(\boldsymbol{\Theta}\right)$ are the coefficient matrix of order $(n_l + 1) \times (n_l + 1)$ and the right-hand side vector $(n_l + 1) \times 1$ given by

$$\begin{cases} \boldsymbol{A}\left(\boldsymbol{\Theta}\right) = \sum\limits_{K\in\mathcal{T}} \mathcal{Q}_K\left(\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}\right]\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}^T\right]\right) & \text{and} \\ F\left(\boldsymbol{\Theta}\right) = \sum\limits_{K\in\mathcal{T}} \mathcal{Q}_K\left(f\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right)\hat{\boldsymbol{\Sigma}}\right]\right), \end{cases} \tag{5.7}$$

respectively. Here the actions of the numerical integration and numerical differentiation operators $\mathcal{Q}_K$ and $D_{\boldsymbol{\beta},\tau}$ are applied component-wisely.

At each step of a block iterative method for solving the SNLS problem in (5.3), given the current estimate of the nonlinear parameters $\boldsymbol{\Theta}$, (5.6) is a linear system with the coefficient matrix $\boldsymbol{A}\left(\boldsymbol{\Theta}\right)$. Let $a_{ij}\left(\boldsymbol{\Theta}\right)$ be the ij-element of $\boldsymbol{A}\left(\boldsymbol{\Theta}\right)$, then

$$a_{ij}\left(\boldsymbol{\Theta}\right) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left(\left[D_{\boldsymbol{\beta},\tau}\sigma_i + \gamma\sigma_i\right]\left[D_{\boldsymbol{\beta},\tau}\sigma_j + \gamma\sigma_j\right]\right) = a_{ji}\left(\boldsymbol{\Theta}\right),$$

which shows that $\boldsymbol{A}\left(\boldsymbol{\Theta}\right)$ is symmetric. Under reasonable assumptions (e.g., the assumption for one hidden layer is that the corresponding breaking hyperplanes are distinct), the set $\{\sigma_i(\mathbf{x})\}_{i=1}^{n_l}$ is linearly independent, implying that $\boldsymbol{A}\left(\boldsymbol{\Theta}\right)$ is positive definite (see, e.g., [21] for shallow NN).

However, due to the global support of the basis functions $\{\sigma_i(\mathbf{x})\}_{i=1}^{n_l}$, $\boldsymbol{A}(\boldsymbol{\Theta})$ is a dense matrix and can be highly ill-conditioned (see, e.g., [22, 23], especially for shallow NNs in one dimension). Consequently, gradient-based optimization methods may suffer from slow convergence, limiting their efficiency for solving such systems.

Similar systems of algebraic equations to (5.7) arise from the least-squares approximation using shallow ReLU NNs [21] and the shallow Ritz method for one-dimensional diffusion and diffusion-reaction problems [22, 23]. In those special cases, efficient and reliable iterative solvers have been developed and analyzed. However, in broader NN applications, the design of fast solvers for the linear parameters remains a critical and largely unresolved challenge. When the number of the linear parameters is relatively small, techniques such as truncated singular value decomposition (SVD) can effectively mitigate the issues caused by large condition numbers and provide accurate solutions despite the ill-conditioning [21]. Nevertheless, scalable and efficient algorithms for larger systems are still an active area of research.

At each step of the block iterative method, the second equation in (5.3) remains a nonlinear system of algebraic equations for $\boldsymbol{\Theta}$, given the current estimate of the linear parameters $\hat{\mathbf{c}}$. A natural choice for solving such NLS problems is the Gauss-Newton (GN) method [31, 51], since its associated GN matrix is always positive semi-definite. A major limitation of the GN method is the potential singularity of the GN matrix, which often necessitates regularization strategies, most notably the shifting technique used in the Levenberg-Marquardt (LM) method [43, 50], to ensure invertibility. These regularization techniques, while widely adopted, modify the original optimization problem and introduce additional complexity. In particular, selecting an appropriate shift parameter is highly problem-dependent and remains a challenging and delicate task in practice.

For the least-squares approximation of a target function using shallow NNs, in [21] we addressed the limitations of the LM by deriving a structured form of the GN matrix and explicitly eliminating its singularity. The resulting structure-guided Gauss-Newton (SgGN) method achieves faster convergence and significantly greater accuracy than the LM approach. An extension of the SgGN method to least-squares approximation problems using deep ReLU NNs will be presented in a forthcoming paper.

Next, we describe the SgGN method for solving the second equation in (5.3) when using shallow ReLU NNs. Any NN function $v(\mathbf{x}) \in \mathcal{M}(1)$ has the form of

$$v(\mathbf{x}) = v(\mathbf{x}; \hat{\mathbf{c}}, \mathbf{r}) = c_0 + \sum_{i=1}^{n} c_i \sigma(\boldsymbol{\omega}_i \mathbf{x} + b_i). \tag{5.8}$$

Here and thereafter, we drop the subscript 1 of n_1 and the superscript (1) for notational simplicity. The nonlinear parameters consist of $\mathbf{r} = (\mathbf{r}_1, \ldots, \mathbf{r}_n)$, with each $\mathbf{r}_i = (b_i, \boldsymbol{\omega}_i)$. Substituting this into the second equation in (5.3), we obtain

$$\mathbf{0} = \nabla_{\mathbf{r}} \hat{\mathcal{L}}_{\mathcal{T}} \left(u_{N,\mathcal{T}}(\cdot; \hat{\mathbf{c}}^*, \mathbf{r}^*) \right) \tag{5.9}$$

Again, we use the intermediate functional $\hat{\mathcal{L}}_{\tau}$ to derive a structured form of the GN matrix and then approximate the integral by quadrature.

Denote by $H(s) = \sigma'(s) = \begin{cases} 0, & s < 0, \\ 1, & s > 0 \end{cases}$ the Heaviside step function, where $\sigma'(s)$ is the weak derivative of $\sigma(s)$ in the distribution sense. Let I_{d+1} be the order-$(d+1)$ identity matrix, $D(\mathbf{c})$ be the diagonal matrix with the i^{th}-diagonal elements c_i, and $\mathbf{y} = (1, x_1, \ldots, x_d)^T$ be the homogeneous-coordinates. A direct calculation (see (4.5) in [21]) gives

$$\nabla_{\mathbf{r}} v(\mathbf{x}; \hat{\mathbf{c}}, \mathbf{r}) = (D(\mathbf{c}) \otimes I_{d+1}) (\mathbf{H}(\mathbf{x}) \otimes \mathbf{y}), \tag{5.10}$$

where $\mathbf{H}(\mathbf{x}) = \left(H_1(\mathbf{x}; \mathbf{r}_1), \ldots, H_n(\mathbf{x}; \mathbf{r}_n) \right)^T$ with $H_i(\mathbf{x}; \mathbf{r}_i) = H(\boldsymbol{\omega}_i \mathbf{x} + b_i) = H(\mathbf{r}_i \mathbf{y})$ and the

symbol $\otimes$ denotes the Kronecker product of two matrices/vectors. Denote by

$$\mathbf{G}(\hat{\mathbf{c}}, \mathbf{r}) = \int_{\Omega} \left(D_{\boldsymbol{\beta},\tau}\, v + \gamma\, v - f\right) \left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \left(\mathbf{H}(\mathbf{x}) \otimes \mathbf{y}\right) d\mathbf{x} \tag{5.11}$$

a scaled gradient vector of $\hat{\mathcal{L}}_\tau\left(v(\cdot; \hat{\mathbf{c}}, \mathbf{r})\right)$ with respect to $\mathbf{r}$. By (5.10), we have

$$\nabla_{\mathbf{r}} \hat{\mathcal{L}}_\tau\left(v(\cdot; \hat{\mathbf{c}}, \mathbf{r})\right) = 2 \int_{\Omega} \left(D_{\boldsymbol{\beta},\tau}\, v + \gamma\, v - f\right) \left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \nabla_{\mathbf{r}} v \, d\mathbf{x} = 2 \left(D(\mathbf{c}) \otimes I_{d+1}\right) \mathbf{G}(\hat{\mathbf{c}}, \mathbf{r})$$

The principal part of the Hessian without the second-order derivative with respect to $\mathbf{r}$ is called the GN matrix given by

$$\begin{cases} \mathcal{G}(\mathbf{c}, \mathbf{r}) = 2 \left(D(\mathbf{c}) \otimes I_{d+1}\right) \mathcal{H}(\mathbf{r}) \left(D(\mathbf{c}) \otimes I_{d+1}\right) \\ \text{with } \mathcal{H}(\mathbf{r}) = \int_{\Omega} \left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \left(\mathbf{H}(\mathbf{x}) \otimes \mathbf{y}\right)\right] \left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \left(\mathbf{H}(\mathbf{x}) \otimes \mathbf{y}\right)^T\right] d\mathbf{x}, \end{cases}$$

where $\mathcal{H}(\mathbf{r})$ is referred to as the layer GN matrix. Approximating the integral by numerical quadrature, we then have the corresponding components

$$\begin{cases} \mathbf{G}_\tau(\hat{\mathbf{c}}, \mathbf{r}) = \sum\limits_{K \in \mathcal{T}} \mathcal{Q}_K \left\{\left(D_{\boldsymbol{\beta},\tau}\, v + \gamma\, v - f\right) \left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \left(\mathbf{H} \otimes \mathbf{y}\right)\right\}, \\ \mathcal{H}_\tau(\mathbf{r}) = \sum\limits_{K \in \mathcal{T}} \mathcal{Q}_K \left\{\left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \left(\mathbf{H} \otimes \mathbf{y}\right)\right] \left[\left(D_{\boldsymbol{\beta},\tau} + \gamma\right) \left(\mathbf{H} \otimes \mathbf{y}\right)^T\right]\right\}, \text{ and} \\ \mathcal{G}_\tau(\mathbf{c}, \mathbf{r}) = 2 \left(D(\mathbf{c}) \otimes I_{d+1}\right) \mathcal{H}_\tau(\mathbf{r}) \left(D(\mathbf{c}) \otimes I_{d+1}\right). \end{cases} \tag{5.12}$$

Now, we describe the SgGN method [21] for solving problem in (5.1). The algorithm starts with an initial function approximation $u_{n,\tau}^{(0)}(\mathbf{x}) = u_{n,\tau}\left(\mathbf{x}; \hat{\mathbf{c}}^{(0)}, \mathbf{r}^{(0)}\right)$ by initializing the nonlinear parameters $\mathbf{r}^{(0)}$, as they define the breaking hyperplanes that partition the domain, and effectively forming a computational "mesh" for our approximation. Given $\mathbf{r}^{(0)}$, we compute the optimal linear parameters $\hat{\mathbf{c}}^{(0)}$ on the current physical partition by solving

$$\boldsymbol{A}\left(\mathbf{r}^{(0)}\right) \hat{\mathbf{c}}^{(0)} = F\left(\mathbf{r}^{(0)}\right), \tag{5.13}$$

where $\boldsymbol{A}\left(\mathbf{r}^{(0)}\right)$ and $F\left(\mathbf{r}^{(0)}\right)$ are defined in (5.7). For a detailed discussion of this initialization strategy, see [47, 14]. Then, given the approximation $u_{n,\tau}^{(k)}(\mathbf{x}) = u_{n,\tau}\left(\mathbf{x}; \hat{\mathbf{c}}^{(k)}, \mathbf{r}^{(k)}\right)$ at the k^{th} iteration, the process of obtaining

$$u_{n,\tau}^{(k+1)}(\mathbf{x}) = u_{n,\tau}\left(\mathbf{x}; \hat{\mathbf{c}}^{(k+1)}, \mathbf{r}^{(k+1)}\right)$$

proceeds as follows:

(i) First, identify a set of *active* neurons in the current approximation $u_{n,\tau}^{(k)}(\mathbf{x})$ by defining

$$\mathcal{I}_{\text{active}} = \left\{ i \in \{1, \ldots, n\} \,\middle|\, \left|c_i^{(k)}\right| \geq \epsilon_{\mathbf{c}} \right\}, \tag{5.14}$$

where $\epsilon_{\mathbf{c}}$ is a prescribed tolerance. The parameters corresponding to these active neurons, denoted as $\tilde{\mathbf{c}}^{(k)}$ and $\tilde{\mathbf{r}}^{(k)}$, are extracted to form a reduced system.

(ii) Next, we form $\tilde{\mathbf{G}}_\tau\left(\hat{\mathbf{c}}^{(k)}, \tilde{\mathbf{r}}^{(k)}\right)$, $D\left(\tilde{\mathbf{c}}^{(k)}\right)$, and $\tilde{\mathcal{H}}_\tau\left(\tilde{\mathbf{r}}^{(k)}\right)$, compute the search direction in the reduced space by

$$\tilde{\mathbf{p}}^{(k+1)} = \left(D^{-1}\left(\tilde{\mathbf{c}}^{(k)}\right) \otimes I_{d+1}\right) \tilde{\mathcal{H}}^{-1}\left(\tilde{\mathbf{r}}^{(k)}\right) \tilde{\mathbf{G}}_\tau\left(\hat{\mathbf{c}}^{(k)}, \tilde{\mathbf{r}}^{(k)}\right), \tag{5.15}$$

and then map it back to the full parameter space by initializing $\mathbf{p}^{(k+1)} = \mathbf{0}$ and setting $\mathbf{p}_i^{(k+1)} = \tilde{\mathbf{p}}_i^{(k+1)}$ only for indices $i \in \mathcal{I}_{\text{active}}$.

(iii) Then, the nonlinear parameter is updated by

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \gamma_{k+1}\mathbf{p}^{(k+1)},$$

where the optimal step size γ_{k+1} is computed by minimizing one dimensional function:

$$\gamma_{k+1} = \underset{\gamma\in\mathbb{R}_0^+}{\arg\min}\ \hat{\mathcal{L}}_\tau\left(u_{n,\tau}\left(\cdot;\hat{\mathbf{c}}^{(k)},\mathbf{r}^{(k)}-\gamma\mathbf{p}^{(k+1)}\right)\right).$$

(iv) Finally, $u_{n,\tau}^{(k+1)}(\mathbf{x}) = u_{n,\tau}\left(\mathbf{x};\hat{\mathbf{c}}^{(k+1)},\mathbf{r}^{(k+1)}\right)$ is obtained by solving

$$\boldsymbol{A}\left(\mathbf{r}^{(k+1)}\right)\hat{\mathbf{c}}^{(k+1)} = F\left(\mathbf{r}^{(k+1)}\right). \tag{5.16}$$

We conclude this section with a few remarks on initialization. The optimization problem in (5.1) is inherently non-convex, making initialization a critical factor for the success of any optimization/iterative/training scheme. This challenge can be addressed by leveraging (1) the physical interpretations of the linear and nonlinear parameters and (2) method of various continuations.

For the shallow ReLU neural network, since the breaking hyper-planes of neurons form a partition of the computational domain, initialization of the nonlinear parameters $\mathbf{r}$ is given by lying those hyper-planes that uniformly partition the domain. Initialization of the linear parameters $\mathbf{c}$ is then the solution of (5.13) (see [14, 16, 21, 22, 23]).

The adaptive neuron enhancement (ANE) method introduced in [47, 26] provides a natural method of continuation. The method of model continuation for linear advection-reaction problems with variable advection field was studied in [14]. Finally, the method of subdomain continuation for the block space-time LSNN method was introduced in [16] for the nonlinear hyperbolic conservation laws.

6. Numerical Experiment. In this section, we present three numerical examples to demonstrate the performance of the LSNN method for linear and nonlinear hyperbolic problems. In each experiment, the discrete LS functionals were minimized using the Adam first-order optimization algorithm [42]. Implementation of the second-order Gauss-Newton method, as presented in Section 5, is beyond the scope of the current experiments and is deferred to future work or for readers to explore. The structure of the ReLU NN used is denoted as d-n_1-$n_2\cdots n_{l-1}$-d_o for a l-layer network, where n_1, n_2 and n_{l-1} represent the number of neurons in the first, second, and $(l-1)$th layers, respectively. Here, d and d_o indicate the input and output dimensions of the problem.

6.1. A 2D linear problem with a variable advection velocity field. Consider a variable advective velocity field $\boldsymbol{\beta}(x,y) = (1,2x)$, $(x,y)\in\Omega = (0,1)\times(0,1)$, and the boundary of the input of the problem is $\Gamma_- = \{(0,y) : y\in(0,1)\}\cup\{(x,0) : x\in(0,1)\}$. The inflow boundary condition is given by

$$g(x,y) = \begin{cases} y+2, & (x,y)\in\Gamma_-^1 \equiv \{(0,y) : y\in[\frac{1}{5},1)\}, \\ (y-x^2)e^{-x}, & (x,y)\in\Gamma_-^2 = \Gamma_-\setminus\Gamma_-^1. \end{cases}$$

The exact solution of this linear advection-reaction problem is

$$u(x,y) = \begin{cases} (y-x^2)e^{-x}, & (x,y)\in\Omega_1 \equiv \{(x,y)\in\Omega : y < x^2+\frac{1}{5}\}, \\ (y-x^2+2)e^{-x}, & (x,y)\in\Omega_2 = \Omega\setminus\Omega_1. \end{cases} \tag{6.1}$$

The LSNN method was implemented using a 2–60–60–1 ReLU NN model and a uniform integration grid of size $h = 0.01$ (see [19] for experiment details). The directional derivative $v_{\boldsymbol{\beta}}$ was approximated by the backward finite difference quotient (4.1) with $\rho = h/10$. We report

the numerical results after 200,000 Adam iterations in Figure 3 and Table 1. As shown in Figures 3(b) to 3(d), the LSNN method is capable of approximating the discontinuous solution with the curved interface with non-constant jump accurately without any oscillation or overshooting. In Figure 3(e), the graph of the physical mesh created by the trained ReLU NN function shows that the optimization process tends to distribute the breaking polylines in the second layer along the interface (see Figure 3(a)) presented in the problem, allowing the discontinuous solution to be accurately approximated using a piecewise linear function. Table 1 shows the relative numerical errors measured in different norms. With 3841 parameters, the ReLU NN can accurately approximate the solution with reasonable accuracy.

TABLE 1
Relative errors of the linear advection-reaction problem.

Network structure	$\frac{\|u-u_{N,\mathcal{T}}\|_0}{\|u\|_0}$	$\frac{\|\|\|u-u_{N,\mathcal{T}}\|\|\|_{\boldsymbol{\beta}}}{\|\|\|u\|\|\|_{\boldsymbol{\beta}}}$	$\frac{\mathcal{L}^{1/2}(u_{N,\mathcal{T}};\mathbf{g})}{\mathcal{L}^{1/2}(u_{N,\mathcal{T}};\mathbf{0})}$	Parameters
2–60–60–1	0.071953	0.115680	0.035981	3841

TABLE 2
Relative L^2 errors of LSNN for the Riemann problem with $f(u)=\frac{1}{4}u^4$

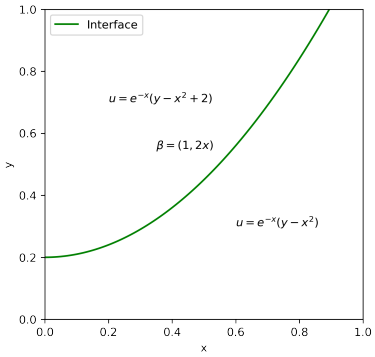
Time block		Number of sub-intervals		
		$\hat{m}=\hat{n}=2$	$\hat{m}=\hat{n}=4$	$\hat{m}=\hat{n}=6$
$\Omega_{0,1}$	Trapezoidal rule	0.067712	0.010446	0.004543
	Mid-point rule	0.096238	0.007917	0.003381
$\Omega_{1,2}$	Trapezoidal rule	0.108611	0.008275	0.009613
	Mid-point rule	0.159651	0.007169	0.005028

6.2. A 1D Riemann problem with a spatial flux $f(u)=\frac{1}{4}u^4$. The second numerical example is a Riemann problem with a convex flux $\mathbf{f}(u)=(f(u),u)=(\frac{1}{4}u^4,u)$ and an initial condition with a unit jump $u_{_L}=1>0=u_{_R}$ at $(0,0)$ [16]. The computational domain is chosen as $\Omega=(-1,1)\times(0,0.4)$ and is subdivided into two blocks, $\Omega_{0,1}=(-1,1)\times(0,0.2)$ and $\Omega_{1,2}=(-1,1)\times(0.2,0.4)$, allowing efficient computation. The numerical integration is performed using a uniform grid of size $h_x=h_t=0.01$ and for the discrete divergence operator $\mathbf{div}_{\mathcal{T}}$ 4.9, we tested two quadrature methods for calculating the line integral $\mathcal{Q}_{\partial K}(\cdot)$: the composite trapezoidal rule and the midpoint rule. Furthermore, we also investigated the impact of the number of sub-intervals, along each boundary edge of ∂K, on the precision of the LSNN method.

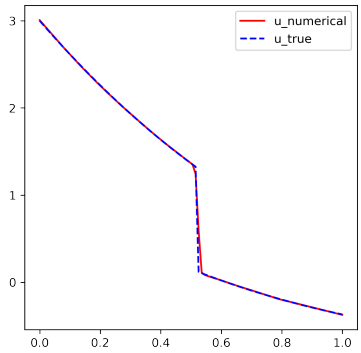
A 2–10–10–1 ReLU NN model was used as an approximate function, and the Adam optimizer trains its associated parameters in $50,000$ iterations, the resulting relative L^2 errors are reported in Tables 2.And the traces of the exact and numerical solutions in $t=0.2$ and $t=0.4$ are plotted in Fig. 4.

From Table 2, it is evident that the accuracy of the LSNN method depends on the number of sub-intervals, with $\hat{m}$ and $\hat{n}$ denoting the number of sub-intervals along the spatial and temporal directions, respectively. In general, larger values of $\hat{m}$ and $\hat{n}$ lead to higher accuracy in the LSNN approximation. However, increasing $\hat{m}$ and $\hat{n}$ also raises the computational cost, particularly for evaluating the line integral $\mathcal{Q}_{\partial K}(\cdot)$, which becomes more expensive with finer partitions. Moreover, the accuracy achieved using the composite trapezoidal and midpoint rules in the LSNN method is comparable; both are capable of accurately simulating this Riemann problem and capturing the correct shock propagation speed.

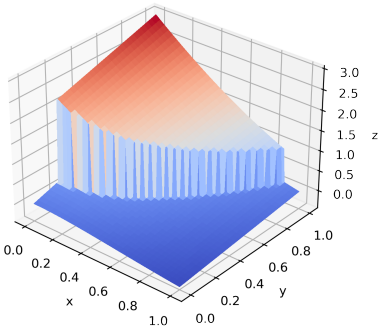
6.3. A 2D inviscid Burgers equation. The last numerical test considers a two-dimensional inviscid Burgers equation, where the spatial flux vector field is $\tilde{\mathbf{f}}(u)=\frac{1}{2}(u^2,u^2)$. Given a piecewise



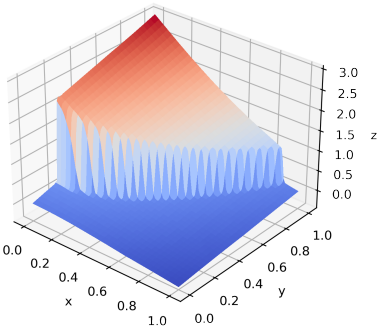
(a) The interface



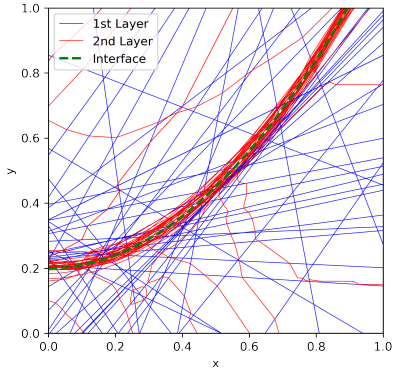
(b) The trace of Figure 3(d) on $y = 1 - x$



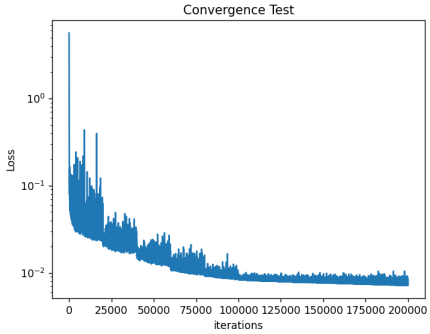
(c) The exact solution



(d) A 2–60–60–1 ReLU NN function approximation



(e) The breaking hyper-planes of the approximation in Figure 3(d)

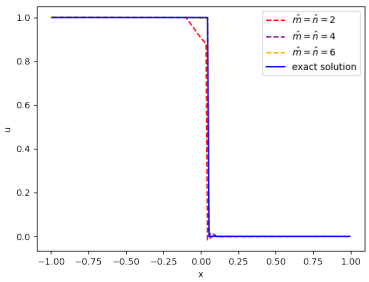


(f) The loss curve

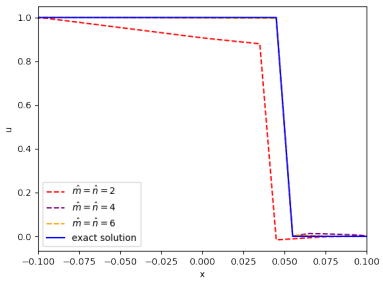
FIG. 3. *Approximation results for the linear advection-reaction problem in Sec. 6.1.*

constant initial data,

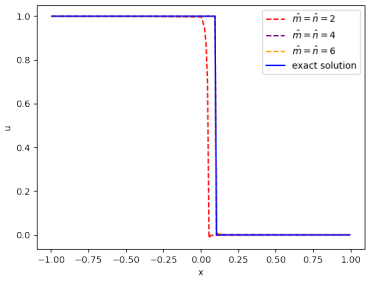
$$u_0(x, y) = \begin{cases} -0.2, & \text{if } x < 0.5 \text{ and } y > 0.5, \\ -1.0, & \text{if } x > 0.5 \text{ and } y > 0.5, \\ 0.5, & \text{if } x < 0.5 \text{ and } y < 0.5, \\ 0.8, & \text{if } x > 0.5 \text{ and } y < 0.5, \end{cases}$$



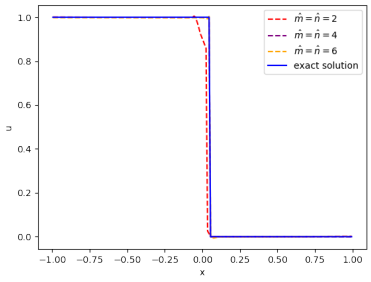
(a) Traces at $t = 0.2$ (trapezoidal)



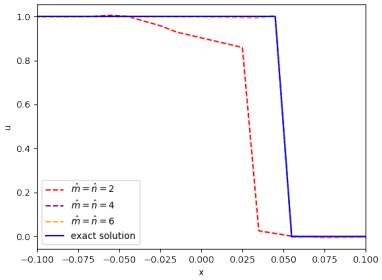
(b) Zoom-in plot near the discontinuous interface of sub-figure (a)



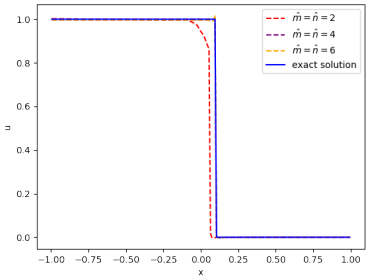
(c) Traces at $t = 0.4$ (trapezoidal)



(d) Traces at $t = 0.2$ (mid-point)



(e) Zoom-in plot near the discontinuous interface of sub-figure (d)



(f) Traces at $t = 0.4$ (mid-point)

FIG. 4. *Numerical results of the problem with* $f(u) = \frac{1}{4}u^4$ *using the composite trapezoidal and mid-point rules*

the exact solution to this problem is as follows [39],

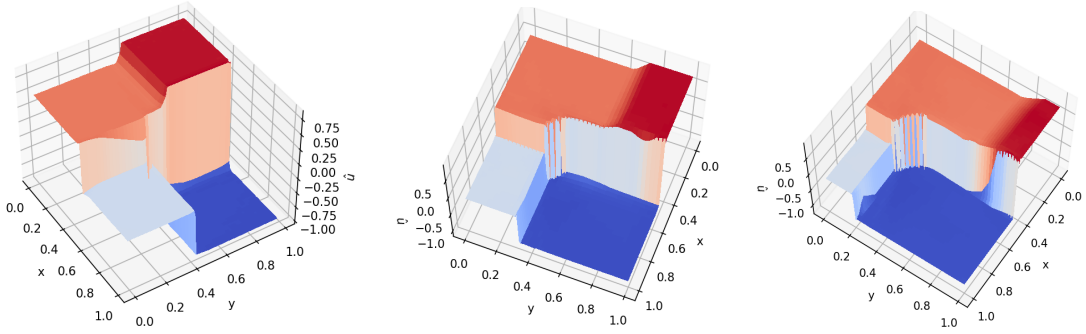
$$u(x,y,t) = \begin{cases} \begin{cases} -0.2, & y > 0.5 + 3t/20, \\ 0.5, & \text{otherwise} \end{cases} & \text{and } x < 0.5 - 3t/5, \\ \begin{cases} -1, & y > -8x/7 + 15/14 - 15t/28, \\ 0.5, & \text{otherwise} \end{cases} & \text{and } 0.5 - 3t/5 < x < 0.5 - t/4, \\ \begin{cases} -1, & y > x/6 + 5/12 - 5t/24, \\ 0.5, & \text{otherwise} \end{cases} & \text{and } 0.5 - t/4 < x < 0.5 + t/2, \\ \begin{cases} -1, & y > x - \frac{5}{18t}(x + t - 0.5)^2, \\ \frac{1}{2t}(2x - 1), & \text{otherwise} \end{cases} & \text{and } 0.5 + t/2 < x < 0.5 + 4t/5, \\ \begin{cases} -1, & y > 0.5 - t/10, \\ 0.8, & \text{otherwise} \end{cases} & \text{and } x > 0.5 + 4t/5. \end{cases}$$

Setting the computational domain $\Omega = (0,1)^2 \times (0,0.5)$, and the inflow boundary conditions prescribed using the exact solution, a 4-layer ReLU NN (3–48–48–48–1) was used as the model function. Again, the numerical integration was performed on uniform grids of size $h_x = h_y = h_t = 0.01$, and the computation domain is decomposed into five time blocks of equal sizes, namely $\Omega_{0,1}, \Omega_{1,2}, \cdots, \Omega_{4,5}$. The three-dimensional discrete divergence operator $\mathbf{div}_{\mathcal{T}}$ is computed using the mid-point quadrature rule with $\hat{m} = \hat{n} = \hat{k} = 2$, where $\hat{m}$, $\hat{n}$ and $\hat{k}$ are the number of sub-intervals along the spatial x, spatial y and the temporal direction. Table 3 reported the relative L^2errors of LSNN in each time block. Specifically, 30,000 iterations of Adam optimization were performed for the first time block, and the rest blocks were trained with 20,000 iterations. Fig.5 presents the numerical results at time $t = 0.1$, 0.3, and 0.5. This experiment shows that the LSNN method can be extended to two-dimensional problems and is capable of simulating the shock and rarefaction waves simultaneously.

As anticipated, numerical error accumulated when using a block space-time method, mentioned in the previous paragraph that decomposes the time interval $[0,5]$ into 5 blocks (see [15]). By $t = 0.5$, the relative error L^2 reached 21.3% (see Table 3) . This result raises an important question for future research: how to enhance the accuracy of the LSNN method for high-dimensional hyperbolic problems. Theoretical studies suggest that a three-layer ReLU NN is sufficient for such problems from a function approximation standpoint [20]. However, developing an efficient and reliable iterative solver suitable for these high-dimensional, non-convex optimization problems remains a challenge. The discussion in Sec. 5 offers insights into leveraging the unique structure of NNs to guide the iterative process, though the problem remains unresolved.

TABLE 3
Relative L^2 errors of LSNN for a 2D Burgers' equation

Network structure	Block	$\frac{\|u^k - u^k_{\mathcal{T}}\|_0}{\|u^k\|_0}$
3-48-48-48-1	$\Omega_{0,1}$	0.093679
	$\Omega_{1,2}$	0.121375
	$\Omega_{2,3}$	0.163755
	$\Omega_{3,4}$	0.190460
	$\Omega_{4,5}$	0.213013



(a) $t = 0.1$ (b) $t = 0.3$ (c) $t = 0.5$

FIG. 5. *Numerical results of 2D Burgers' equation.*

REFERENCES

[1] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*, 2016.

[2] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Science of USA*, 116 (31):15344–15349, 2019.

[3] J. Berg and K. Nystrom. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[4] B. Bochev and J. Choi. Improved least-squares error estimates for scalar hyperbolic problems. *Comput. Methods Appl. Math.*, 1(2):115–124, 2001.

[5] B. Bochev and M. Gunzburger. Least-squares methods for hyperbolic problems. *Handbook of Numerical Analysis*, 17:289–317, 2016.

[6] P. B. Bochev, Z. Cai, T. Manteuffel, and S. McCormick. Analysis of velocity-flux least-squares principles for navier-stokes equations: Part i. *SIAM J. Numer. Anal*, 35:990–1009, 1998.

[7] P. B. Bochev and M. D. Gunzburger. Analysis of least-squares finite element methods for the stokes equations. *Math. Comp*, 63:479–506, 1994.

[8] P. B. Bochev and M. D. Gunzburger. Least-squares methods for the velocity-pressure-stress formulation of the stokes equations. *Comput. Methods Appl. Mech. Engrg.*, 126:267–287, 1995.

[9] J. Bramble, R. Lazarov, and J. Pasciak. A least-squares approach based on a discrete minus one inner product for first order system. *Math. Comp*, 66:935–955, 1997.

[10] J. Bramble, R. Lazarov, and J. Pasciak. Least-squares methods for linear elasticity based on a discrete negative norm. *Comput. Methods in Appl. Mech. Engrg.*, 152:520–543, 2001.

[11] J. H. Bramble and A. H. Schatz. Rayleigh-Ritz-Galerkin-methods for Dirichlet's problem using subspaces without boundary conditions. *Comm. Pure Appl. Math*, 23:653–675, 1970.

[12] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer-Verlag, New York, 1994.

[13] F. Brezzi and M. Fortin. *Mixed and Hybrid Finite Element Methods*. Springer-Verlag, New York, 1991.

[14] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *J. Comput. Phys.*, 443:110514, 2021.

[15] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for scalar nonlinear hyperbolic conservation law. *Appl. Numer. Math.*, 174:163–176, 2022.

[16] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for scalar nonlinear hyperbolic conservation laws: discrete divergence operator. *J. Comput. Appl. Math.*, 433:115298, 2023.

[17] Z. Cai, J. Chen, M. Liu, and X. Liu. Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic PDEs. *J. Comput. Phys.*, 420 (2020) 109707.

[18] Z. Cai, J. Choi, and M. Liu. Least-squares neural network (LSNN) method for linear advection-reaction equation: discontinuity interface. *SIAM J. Sci. Comput.*, 46:C448–C478, 2024.

[19] Z. Cai, J. Choi, and M. Liu. Least-squares neural network (LSNN) method for linear advection-reaction equation: non-constant jumps. *International Journal of Numerical Analysis & Modeling*, 21(5):C609–C628, 2024.

[20] Z. Cai, J. Choi, and M. Liu. ReLU neural network approximation to piecewise constant functions. *arXiv:2410.16506 [math.FA]*, 2024.

[21] Z. Cai, T. Ding, M. Liu, X. Liu, and J. Xia. A structure-guided Gauss-Newton method for shallow neural network. *arXiv:2404.05064 [cs.LG]*, 2024.

[22] Z. Cai, A. Doktorova, R. D. Falgout, and C. Herrera. Efficient shallow Ritz method for 1D diffusion problems. *Comput. Math. Appl.*, 200:349–363, 2025.

[23] Z. Cai, A. Doktorova, R. D. Falgout, and C. Herrera. Efficient shallow Ritz method for 1D diffusion-reaction problems. *SIAM J. Sci. Comput.*, page S414–S435, 2025.

[24] Z. Cai, R. Lazarov, T. A. Manteuffel, and S. F. McCormick. First-order system least squares for second-order partial differential equations: Part i. *SIAM Journal on Numerical Analysis*, 31(6):1785–1799, 1994.

[25] Z. Cai, B. Lee, and P. Wang. Least-squares methods for incompressible newtonian fluid flow: linear stationary problems. *SIAM J. Numer. Anal*, 42(2):843–859, 2004.

[26] Z. Cai and M. Liu. Self-adaptive ReLU neural network method in least-squares data fitting. In *Principles and Applications of Adaptive Artificial Intelligence*, pages 242–262. IGI Global, 2024.

[27] Z. Cai, T. Manteuffel, and S. McCormick. First-order system least squares for the stokes equations, with applications to linear elasticity. *SIAM J. Numer. Anal*, 34:1727–1741, 1997.

[28] Z. Cai, T. A. Manteuffel, and S. F. McCormick. First-order system least squares for second-order partial differential equations: Part ii. *SIAM J. Numer. Anal*, 34(2):425–454, 1997.

[29] Z. Cai and G. Starke. Least-squares methods for linear elasticity. *SIAM J. Numer. Anal*, 42(2):826–842, 2004.

[30] H. De Sterck, T. A. Manteuffel, S. F. McCormick, and L. Olson. Least-squares finite element methods and algebraic multigrid solvers for linear hyperbolic pdes. *SIAM Journal on Scientific Computing*, 26(1):31–54, 2004.

[31] J. E. Dennis Jr and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

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

[33] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational

problems. *Communications in Mathematics and Statistics*, 6(1), 3 2018.
[34] I. Ekeland and R. Témam. *Convex Analysis and Variational Problems*. North-Holland and American Elsevier, Amsterdam and New York, 1976.
[35] O. Fuks and H. Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport porous media. *J. Machine Learning for Modeling and Computing*, 1(1):19–37, 2020.
[36] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*. Springer, New York, 1996.
[37] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*, volume 118. Springer Science & Business Media, 2013.
[38] G. H. Golub and V. Pereyra. The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate. *SIAM J. Numer. Anal.*, 10(2):413–432, 1973.
[39] J.-L. Guermond and M. Nazarov. A maximum-principle preserving c0 finite element method for scalar conservation equations. *Computer Methods in Applied Mechanics and Engineering*, 272:198–213, 2014.
[40] J. S. Hesthaven. *Numerical Methods for Conservation Laws: From Analysis to Algorithms*. SIAM, 2017.
[41] L. Kaufman. A variable projected method for solving separable nonlinear least squares problems. *BIT*, 15:49–57, 1975.
[42] D. P. Kingma and J. Ba. ADAM: A method for stochastic optimization. In *International Conference on Representation Learning, San Diego*, 2015; arXiv preprint arXiv:1412.6980.
[43] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quart. Appl. Math.*, 2:164–168, 1944.
[44] R. J. LeVeque. *Numerical Methods for Conservation Laws*. Birkhäuser, Boston, 1992.
[45] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, Cambridge, 2002.
[46] M. Liu and Z. Cai. Adaptive two-layer relu neural network: II. Ritz approximation to elliptic PDEs. *Comput. Math. Appl.*, 113:103–116, 2022.
[47] M. Liu, Z. Cai, and J. Chen. Adaptive two-layer ReLU neural network: I. best least-squares approximation. *Comput. Math. Appl.*, 113:34–44, 2022.
[48] M. Liu, Z. Cai, and K. Ramani. Deep Ritz method with adaptive quadrature for linear elasticity. *Comput. Methods Appl. Mech. Engrg.*, 415:116229, 2023.
[49] M. Liu, Z. Cai, and K. Ramani. Dual neural network (dunn) method for elliptic partial differential equations and systems. *J. Comput. Appl. Math.*, 467:116596, 2025.
[50] D. W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *J. SIAM*, 11(2):431–441, 1963.
[51] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. SIAM, 2000.
[52] R. G. Patel, I. Manickam, N. A. Trask, M. A. Wood, M. Lee, I. Tomas, and E. C. Cyr. Thermodynamically consistent physics-informed neural networks for hyperbolic systems. *J. Comput. Phys.*, 449, 2022.
[53] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.
[54] C.-W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, pages 325–432. Springer, 1998.
[55] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1139–1364, 2018.
[56] A. H. Stroud. *Approximate Calculation of Multiple Integrals*. Englewood Cliffs, N.J.: Prentice-Hall, 1971.
[57] J. W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22. Springer Science & Business Media, 2013.
[58] J. Xu. The finite neuron method and convergence analysis. *arXiv:2010.01458vl*, 2020.