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

ZHIQIANG CAI† AND MIN LIU‡

Abstract. This chapter offers a comprehensive introduction to the least-squares neural network (LSNN) method introduced in [5, 4], for solving scalar hyperbolic partial differential equations (PDEs), 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 appropriate solution space that accommodates discontinuous solutions. It employs ReLU neural networks (in place of finite elements) as the approximating functions, uses a carefully designed physics-preserving numerical differentiation, and avoids penalization techniques such as the 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.

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., [25, 18, 31, 26, 35, 22]). 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 becomes invalid at points where the solution is discontinuous.

Recently, neural networks (NNs) have emerged as a novel class of approximating functions for solving partial differential equations (see, e.g., [6, 16, 30, 32]). A neural network function is a linear combination of compositions of linear transformations and a nonlinear univariate activation function. As demonstrated in [5, 7, 8], 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. Due to the unknown location of these interfaces, enforcing the RH condition in computations is difficult. 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 (2.3) for the directional derivative and (2.11) 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 a suitable solution space that accommodates discontinuous solutions. Through appropriate numerical integration for the integral and physics-preserved numerical differentiation for the physically meaningful derivative, the LSNN method is established as minimizing the discrete counterpart of the least-squares functional over the set of 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 [5, 4] 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

*This work was supported in part by the National Science Foundation under grant DMS-2110571.

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

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

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 [9, 10, 11].

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.

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 and $\gamma \in C(\bar{\Omega})$ be the reaction coefficient. 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

$$\left\{\begin{array}{rcll} \displaystyle\sum_{i=1}^{d} \beta_i(\mathbf{x}) \frac{\partial u(\mathbf{x})}{\partial x_i} + \gamma u & = & f & \text{in } \Omega, \\ u & = & g & \text{on } \Gamma_-. \end{array}\right. \tag{2.1}$$

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 in (2.1) may be rescaled by dividing $|\boldsymbol{\beta}(\mathbf{x})|$. If the inflow boundary data g is discontinuous, so is the solution $u(\mathbf{x})$. Hence, the PDE in (2.1) is not valid at where u is discontinuous.

To deal with this issue, let us define the directional derivative of u along the direction $\boldsymbol{\beta}$ by

$$u_{\boldsymbol{\beta}} = \lim_{\rho \to 0} \frac{u(\mathbf{x}) - u\big(\mathbf{x} - \rho\boldsymbol{\beta}(\mathbf{x})\big)}{\rho}. \tag{2.2}$$

Then (2.1) may be rewritten as

$$\left\{\begin{array}{rcll} u_{\boldsymbol{\beta}} + \gamma u & = & f & \text{in } \Omega, \\ u & = & g & \text{on } \Gamma_-. \end{array}\right. \tag{2.3}$$

Note that (2.3) is well-defined in the entire domain Ω.

Denote the solution space by

$$V_{\boldsymbol{\beta}} = \{v \in L^2(\Omega) : v_{\boldsymbol{\beta}} \in L^2(\Omega)\}, \tag{2.4}$$

and define the following least-squares functional

$$\mathcal{L}(v; \mathbf{f}) = \|v_{\boldsymbol{\beta}} + \gamma\, v - f\|_{0,\Omega}^2 + \|v - g\|_{-\boldsymbol{\beta}}^2, \quad \forall\, v \in V_{\boldsymbol{\beta}}, \tag{2.5}$$

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

$$\|v\|_{-\boldsymbol{\beta}} = \langle v, v\rangle_{-\boldsymbol{\beta}}^{1/2} = \left(\int_{\Gamma_-} |\boldsymbol{\beta}\cdot\boldsymbol{n}|\, v^2\, ds\right)^{1/2}.$$

Then the least-squares formulation of problem (2.3) studied in [1, 13, 2] is to seek $u \in V_{\boldsymbol{\beta}}$ such that

$$\mathcal{L}(u; \mathbf{f}) = \min_{v \in V_{\boldsymbol{\beta}}} \mathcal{L}(v; \mathbf{f}). \tag{2.6}$$

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

$$\left\{\begin{array}{rcll} \mathbf{div}\,(\boldsymbol{\beta} u) + \gamma u & = & f & \text{in } \Omega, \\ u & = & g & \text{on } \Gamma_-. \end{array}\right. \tag{2.7}$$

If the solution u is discontinuous, then the divergence operator **div** *should be understood in a weak sense as similarly defined in (2.9).*

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

$$\left\{\begin{array}{rcll} u_t(\mathbf{x}, t) + \displaystyle\sum_{i=1}^{d} \frac{\partial f_i\,(u(\mathbf{x}, t))}{\partial x_i} & = & 0, & \text{in } \Omega \times I, \\ u & = & g, & \text{on } \Gamma_-, \\ u(\mathbf{x}, 0) & = & u_0(\mathbf{x}), & \text{in } \Omega, \end{array}\right. \tag{2.8}$$

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 al $i \in \{1, \ldots, d\}$.

The solution of (2.8) is often discontinuous due to a discontinuous initial or inflow boundary condition, or a shock formation. Hence, the strong form in (2.8) is only valid at where the solution is differentiable. The Rankine-Hugoniot (RH) jump condition (see, e.g., [25, 19]) is supplemented at the discontinuity interface. But the interface is unknown *a priori*, it is then difficult to enforcing the RH jump condition in computation.

To deal with this difficulty, denote the total flux by

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

and define the space-time divergence operator **div** in a weak sense as follows:

$$\mathbf{div}\,\mathbf{F}(u(\mathbf{x}, t)) = \lim_{\epsilon \to 0} \frac{1}{|B_\epsilon(\mathbf{x}, t)|} \int_{\partial B_\epsilon(\mathbf{x}, t)} \mathbf{F}(u) \cdot \mathbf{n}\, dS, \tag{2.9}$$

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

$$\mathbf{div}\,\mathbf{F}(u(\mathbf{x}, t)) = u_t(\mathbf{x}, t) + \sum_{i=1}^{d} \frac{\partial f_i\,(u(\mathbf{x}, t))}{\partial x_i}. \tag{2.10}$$

If u is discontinuous at $(\mathbf{x},t)$, then $\mathbf{div}\,\mathbf{F}(u(\mathbf{x},t))$ defined in (2.9) leads to the continuity condition of the normal component of the space-time flux $\mathbf{F}(u)$ that is identical to the RH jump condition.

Now, problem (2.8) may be rewritten as the following form

Find $u \in \mathcal{V}_{\mathbf{F}} = \left\{ v \in L^2(\Omega\times I) \,|\, \mathbf{F}(v) \in H(\mathrm{div};\Omega\times I) \right\}$ such that

$$u = \arg\min_{v\in\mathcal{V}_{\mathbf{F}}} \mathcal{L}(v;\mathbf{f}), \quad \text{where } \mathcal{L}(v;\mathbf{f}) = \|\mathbf{div}\,\mathbf{F}(v)\|^2_{0,\Omega\times I} + \|v-u_0\|^2_{0,\Omega\times\{0\}}$$

$$\left\{ \begin{array}{rcll} \mathbf{div}\,\mathbf{F}(u) & = & 0, & \text{in } \ \Omega\times I \in \mathbb{R}^{d+1}, \\ u & = & g, & \text{on } \ \Gamma_-, \\ u(\mathbf{x},0) & = & u_0(\mathbf{x}), & \text{in } \ \Omega. \end{array} \right. \tag{2.11}$$

Denote the collection of square integrable vector fields whose divergence is also square integrable by

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

It is then easy to see that solutions of (2.11) are in the following subset of $L^2(\Omega\times I)$

$$\mathcal{V}_{\mathbf{F}} = \left\{ v \in L^2(\Omega\times I) \,|\, \mathbf{F}(v) \in H(\mathrm{div};\Omega\times I) \right\}. \tag{2.12}$$

Define the least-squares (LS) functional as

$$\mathcal{L}(v;\mathbf{f}) = \|\mathbf{div}\,\mathbf{F}(v)\|^2_{0,\Omega\times I} + \|v-g\|^2_{0,\Gamma_-} + \|v-u_0\|^2_{0,\Omega\times\{0\}}, \tag{2.13}$$

where $\mathbf{f} = (g, u_0)$, $\|\cdot\|_{0,S}$ 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 seek $u \in \mathcal{V}_{\mathbf{F}}$ such that

$$\mathcal{L}(u;g,u_0) = \min_{v\in\mathcal{V}_{\mathbf{F}}} \mathcal{L}(v;g,u_0). \tag{2.14}$$

PROPOSITION 2.2. *Assume that $u \in L^\infty(\Omega\times I)$ is a piece-wise C^1 function. Then u is a weak solution of (2.11) if and only if u is a solution of the minimization problem in (2.14).*

Proof. The proposition is a direct consequence of Theorem 2.5 in [14]. □

3. ReLU Neural Network and its Approximation to Discontinuous Functions.

This section describes l-hidden-layer ReLU neural network as a set of continuous piece-wise linear functions and illustrates its striking approximation power to discontinuous functions with *unknown* interface locations [5, 8].

ReLU refers to the rectified linear activation function defined by

$$\sigma(t) = \max\{0,t\} = \left\{ \begin{array}{ll} t, & t>0, \\ 0, & t\le 0. \end{array} \right. \tag{3.1}$$

The $\sigma(t)$ is a continuous piece-wise linear function with one *breaking* point $t=0$. 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. Introduce a vector-valued function $\mathbf{N}^{(k)}: \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$ as

$$\mathbf{N}^{(k)}\left(\mathbf{x}^{(k-1)}\right) = \sigma\left(\boldsymbol{\omega}^{(k)}\mathbf{x}^{(k-1)} + \mathbf{b}^{(k)}\right) \quad \text{for } \mathbf{x}^{(k-1)} \in \mathbb{R}^{n_{k-1}}, \tag{3.2}$$

where application of the activation function σ to a vector-valued function is defined component-wisely and n_0 is the input dimension.

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

$$\mathcal{M}(l) = \left\{ \begin{array}{ll} \mathbf{c}_1 \left(\mathbf{N}^{(l)} \circ \cdots \circ \mathbf{N}^{(1)}(\mathbf{x})\right) + c_0 : & (c_0, \mathbf{c}_1) \in \mathbb{R}^{n_l+1},\ \boldsymbol{\omega}^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}, \\ & \mathbf{b}^{(k)} \in \mathbb{R}^{n_k} \text{ for } k = 1, \ldots, l \end{array} \right\}, \tag{3.3}$$

where the symbol $\circ$ denotes the composition of functions. The total number of parameters of $\mathcal{M}(l)$ is given by

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

In the remainder of this section, we use the step function with a hyper-plane interface to illustrate the remarkable approximation property of the ReLU NN function. To this end, let $\chi(\mathbf{x})$ be a piece-wise constant function defined on Ω given by

$$\chi(\mathbf{x}) = \left\{ \begin{array}{ll} 0, & \mathbf{x} \in \Omega_1, \\ 1, & \mathbf{x} \in \Omega_2, \end{array} \right. \tag{3.5}$$

where Ω_1 and Ω_2 are open, connected subdomains of Ω such that

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

Let $\partial\Omega_i$ be the boundary of the subdomain Ω_i, assume that the interface $\Gamma = \partial\Omega_1 \cap \partial\Omega_2$ is C^0 and that its $(d-1)$-dimensional measure $|\Gamma|$ is finite.

When the interface Γ is part of a hyper-plane

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

the step function in (3.5) can be approximated by either a two-layer or a three-layer NN function:

$$p_1(\mathbf{x}) = \frac{1}{2\varepsilon}\big(\sigma(\mathbf{a}\cdot\mathbf{x} - b + \varepsilon) - \sigma(\mathbf{a}\cdot\mathbf{x} - b - \varepsilon)\big) \quad \text{or} \quad p_2(\mathbf{x}) = 1 - \sigma\left(-\frac{1}{\varepsilon}\sigma(\mathbf{a}\cdot\mathbf{x} - b) + 1\right) \tag{3.6}$$

within any prescribed accuracy $\varepsilon > 0$, where $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ were introduced in [5] and [8], respectively.

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

$$\|\chi - p\|_{L^r(\Omega)} \le C\,|\Gamma|^{1/r}\varepsilon^{1/r} \quad \textit{and} \quad \|\chi - \mathcal{N}\|_{L^r(\Omega)} \le C\,|\Gamma|^{1/r}\varepsilon^{1/r}, \tag{3.7}$$

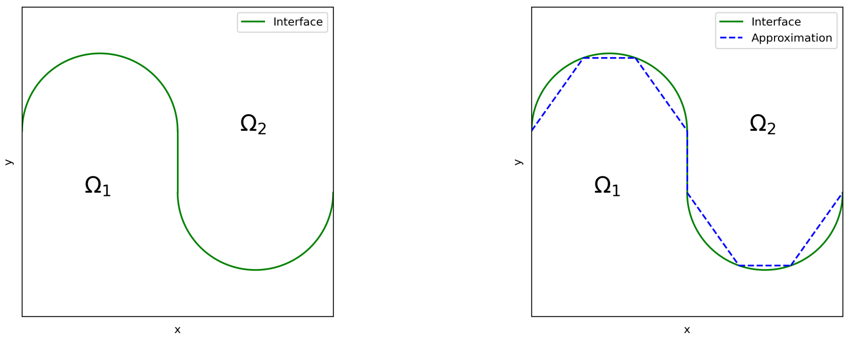
where $|\Gamma|$ is the $(d-1)$-dimensional measure of the interface Γ.

Proof. Let

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

Clearly, we have

$$\chi(\mathbf{x}) - p_1(\mathbf{x}) = 0,\ \forall\, \mathbf{x} \in \Omega \setminus \Omega_{p_1} \quad \text{and} \quad \chi(\mathbf{x}) - p_2(\mathbf{x}) = 0,\ \forall\, \mathbf{x} \in \Omega \setminus \Omega_{p_2}. \tag{3.8}$$



(a) The interface Γ

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

FIG. 1. *Approximation of the interface* Γ

It is then easy to see that

$$|\chi(\mathbf{x}) - p_1(\mathbf{x})|^r \le 1,\ \forall\, \mathbf{x} \in \Omega \setminus \Omega_{p_1} \quad \text{and} \quad |\chi(\mathbf{x}) - p_2(\mathbf{x})|^r \le 1,\ \forall\, \mathbf{x} \in \Omega \setminus \Omega_{p_2}, \tag{3.9}$$

which, together with the facts that

$$\left|\Omega_{p_1}\right| \le C\left|\Gamma\right|\varepsilon \quad \text{and} \quad \left|\Omega_{p_2}\right| \le C\left|\Gamma\right|\varepsilon,$$

implies the validity of (3.7). This completes the proof of the lemma. ∎

REMARK 3.2. *In the case that the interface* Γ *is not a hyper-plane, but can be approximated by a connected series of hyper-planes with a prescribed accuracy* $\varepsilon > 0$ (*see* Figure 1 and [8]), *then the piece-wise constant function* $\chi(\mathbf{x})$ *may be approximated by a ReLU NN function with given architecture satisfying the error bound in* (3.7).

More precisely, based on the one-hidden-layer ReLU NN approximation $p_1(\mathbf{x})$ *in* (3.6), *we showed in* [7] *that a ReLU NN with at most* $\lceil \log_2(d+1) \rceil + 1$ *layers is sufficient to achieve the prescribed accuracy* ε. *However,* [7] *does not provide an estimate on the minimum number of neurons at each layer.*

Based on the two-hidden-layer ReLU NN approximation $p_2(\mathbf{x})$ *in* (3.6), *we showed in* [8] *that* $\chi(\mathbf{x})$ *may be approximated with the same accuracy by a two-hidden-layer ReLU NN. Moreover, the number of neurons at the first hidden-layer and their locations depend on the hyper-planes used for approximating the interface and the number of neurons of the second hidden-layer depends on convexity of the interface* (*see* Theorem 3.2 *in* [8]).

REMARK 3.3. *Let* $\{\Omega_i\}_{i=1}^k$ *be a partition of the domain* Ω. *Let* $\chi(\mathbf{x})$ *be a piece-wise constant function with respect to the partition with* $\chi(\mathbf{x}) = \alpha_i$ *in* Ω_i *for* $i = 1, \ldots, k$. *Then we have*

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

where $\mathbf{1}_{\Omega_i}(\mathbf{x})$ *is the indicator function of the subdomain* Ω_i. *As indicated in* Remark 3.2, *each indicator function may be approximated by a ReLU function with a prescribed accuracy, and so is* $\chi(\mathbf{x})$.

4. Least-Squares Neural Network (LSNN) Method.

This section introduces the least-squares neural network (LSNN) method for solving advection-reation equations in (2.1) and scalar nonlinear hyperbolic conservation laws in (2.11) based on the equivalent least-squares formulations in (2.6) and (2.14), 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. Evaluation of the least-squares functional $\mathcal{L}(v;\mathbf{f})$ defined in (2.5) or (2.13) requires integrations over the computational domain $\Omega \subset \mathbb{R}^d$ or $\Omega \times I \subset \mathbb{R}^{d+1}$ ($d = 1, 2$, or 3) and their partial boundaries. In practice, each integration is approximated by a numerical integration. This section describes basic numerical integration and discusses some strategies in the application of the LSNN method.

To this end, let

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

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

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

On the *integration mesh* $\mathcal{T}$, we denote a composite numerical integration as follows

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

where $\mathcal{Q}_K(w) \approx \int_K w(\mathbf{x})\, d\mathbf{x}$ denotes a quadrature rule over K. First, $\mathcal{Q}_K$ may vary on $K \in \mathcal{T}$. Second, its choice is one of the standard quadrature rules like the Gaussian quadrature or Newton–Cotes formulas such as the midpoint, trapezoidal, or Simpson rule (see [33]). In the case of 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 K and $|K|$ is the d-dimensional measure of K.

In the application of the LSNN method, integrands depend on NN approximations to the solution u of the underlying PDE. Each NN approximation is a continuous piece-wise linear function with respect to a physical partition [27] that is in general unknown and moving. Moreover, the solution u is unknown and has some local features.

Because of these considerations, adaptive numerical integration was introduced in [27] (see Algorithm 5.2) and in [28] (see Algorithm 3.1). Below, we briefly describe the adaptive mesh refinement for numerical integration with a fixed NN in Algorithm 4.1 for problem (2.1). As usual, we start with a uniform and coarse partition $\mathcal{T}$ of the domain Ω. Assume that the inflow boundary data g can be approximated with a prescribed accuracy by continuous linear function with respect to the partition $\mathcal{T}$. Let $u_{\mathcal{T}}$ be a NN approximation based on an initial partition $\mathcal{T}$. For each subdomain $K \in \mathcal{T}$, the local error indicator is given by

$$\eta_{\scriptscriptstyle K} = \|\, (u_{\mathcal{T}})_{\boldsymbol{\beta}} + \gamma\, u_{\mathcal{T}} - f\|_{0,K}.$$

Then the global error estimator is given by $\eta = \left(\sum\limits_{K\in\mathcal{T}} \eta_{\scriptscriptstyle K}^2\right)^{1/2}$. The adaptive mesh refinement is summarized in Algorithm 4.1.

As indicated in [27, 28], the stopping criterion used in Algorithm 4.1 is based on whether or not the quadrature refinement on numerical integration improves approximation accuracy. When the refinement does not improve accuracy much, the adaptive quadrature stops and outputs the current integration mesh.

Remark 4.1. *In the case that computational cost is not an issue, one may use a uniform partition $\mathcal{T}$ that is fine enough to approximate the unknown solution well by a piece-wise constant function.*

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

```
(1) for each $K \in \mathcal{T}$, compute the local error indicator $\eta_K$;
(2) mark $\mathcal{T}$ by the either bulk or average marking strategy (see, e.g., [27]) 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) if $\eta(u_{\mathcal{T}'}) \le \gamma\eta(u_{\mathcal{T}})$, go to Step (1) with $\mathcal{T} = \mathcal{T}'$; otherwise, output $\mathcal{T}$.
```

4.2. Physics-preserved Numerical Differentiation. Solutions of hyperbolic PDEs in (2.1) and (2.8) could be discontinuous. This indicates that numerical and auto-differentiations along coordinates based on (2.1) and (2.8) are inadequate. In this section, we describe the physics-preserved numerical differentiation based on (2.3) and (2.11) introduced in [5, 4].

When u is discontinuous, as discussed in Subsection 2.1 and Subsection 2.2, the directional derivative $u_{\boldsymbol{\beta}}(\mathbf{x})$ and the divergence of the total flux $\mathbf{div}\,\mathbf{F}(u)$ may be defined, respectively, in (2.2) and (2.9) through the limit process. Any approximation to those limits leads to the so-called physics-preserved numerical differentiation.

Based on (2.2), for any $\mathbf{x} \in \Omega$, define the discrete differential operator $D_{\boldsymbol{\beta}}$ by

$$D_{\boldsymbol{\beta}} v(\mathbf{x}) := \frac{v(\mathbf{x}) - v\big(\mathbf{x} - \rho\boldsymbol{\beta}(\mathbf{x})\big)}{\rho} \approx v_{\boldsymbol{\beta}}(\mathbf{x}), \tag{4.1}$$

where ρ is the directional derivative "mesh" size and the $0 < \rho \ll 1$ is a parameter. That is, the directional derivative $v_{\boldsymbol{\beta}}$ along the $\boldsymbol{\beta}$ direction is approximated by the backward finite difference quotient with the "mesh" size ρ. The $D_{\boldsymbol{\beta}}$ defined in (4.1) ensures that the derivative is computed without crossing the discontinuous interface.

To define the discrete divergence operator based on (2.9), for each integration point $\mathbf{z}$, we associate with a subdomain (control volume) $K_{\mathbf{z}}$ containing the point. Then the discrete divergence operator at $\mathbf{z}$ is defined as

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

where $\mathcal{Q}_{\partial K_{\mathbf{z}}}(\cdot)$ is a *composite quadrature rule* 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 $\partial K_{\mathbf{z}}$.

For the midpoint rule $\mathcal{Q}_K$, there is only one integration point $\mathbf{z}_K = (\mathbf{x}_K, t_K)$ that is the centroid of the subdomain $K \in \mathcal{T}$, then the control volume is the subdomain K, i.e., $K_{\mathbf{z}_K} = K$. For a quadrature rule $\mathcal{Q}_K$ having 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 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 similarly defined as in (4.2).

The generic definition of the discrete divergence operator $\mathbf{div}_{\mathcal{T}}$ in (4.2) depends on the quadrature rule over the boundary ∂K, $\mathcal{Q}_{\partial K}(\cdot)$, and in turn on the shape of K. Since the partition $\mathcal{T}$ is an integration mesh independent of the physical partition of the NN approximation, in practice, it is then convenient to choose the integration mesh $\mathcal{T}$ to be a composite mesh generated by the AQR in Algorithm 4.1 so that each $K \in \mathcal{T}$ is a rectangle, cuboid, or hypercube in two, three, or four dimensions, respectively; moreover each face of K is parallel to one of the coordinate hyper-planes. For such integration mesh $\mathcal{T}$ in both two and three dimensions, explicit definitions of $\mathbf{div}_{\mathcal{T}}\mathbf{F}(u(\mathbf{z}_K))$ was introduced and analyzed in [4] in the case that u is discontinuous.

4.3. Least-Squares Neural Network (LSNN) Method. Denote the collections of the inflow boundary faces and the initial faces of the integration mesh $\mathcal{T}$ by

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

respectively. For each E in $\mathcal{E}_-$ or $\mathcal{E}_0$, let $\mathcal{Q}_E(w)$ denote a quadrature rule for integrand w defined on E. Define the discrete least-squares functionals by

$$\mathcal{L}_{\mathcal{T}}\big(v;\mathbf{f}\big) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left((D_{\boldsymbol{\beta}}v + \gamma v - f)^2\right) + \sum_{E\in\mathcal{E}_-} \mathcal{Q}_E\left(|\boldsymbol{\beta}\cdot\boldsymbol{n}|(v-g)^2\right) \tag{4.3}$$

for problem (2.1) and by

$$\mathcal{L}_{\mathcal{T}}\big(v;\mathbf{f}\big) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left(\mathbf{div}_{\mathcal{T}}\mathbf{F}(v)\right) + \sum_{E\in\mathcal{E}_-} \mathcal{Q}_E\left((v-g)^2\right) + \sum_{E\in\mathcal{E}_0} \mathcal{Q}_E\left((v-u_0)^2\right) \tag{4.4}$$

for problem (2.8). Then the least-squares least-squares (LSNN) method for problems (2.1) or (2.8) is to seek $u_{{}_{N,\mathcal{T}}} \in \mathcal{M}(l)$ such that

$$\mathcal{L}_{\mathcal{T}}\big(u_{{}_{N,\mathcal{T}}};\mathbf{f}\big) = \min_{v\in\mathcal{M}(l)} \mathcal{L}_{\mathcal{T}}\big(v;\mathbf{f}\big). \tag{4.5}$$

The least-squares functionals in (4.3) and (4.4) enforce the inflow boundary and initial conditions through penalization: the summation terms over E in $\mathcal{E}_-$ and $\mathcal{E}_0$. Below, we impose them weakly through the physics-preserved numerical differentiation in Subsection 4.2.

For simplicity of presentation, let us assume that the $\mathcal{Q}_K(\cdot)$ is the midpoint rule. Then the centroid of K, $\mathbf{z}_K = \mathbf{x}_K$ or $(\mathbf{x}_K, t_K)$, is the only integration point in K. For each inflow boundary or initial face $E \in \mathcal{E}_-$ or $\mathcal{E}_0$, there exists a subdomain $K \in \mathcal{T}$ such that $E \in \partial K$. For convenience, denote it by E_K to indicate that E is part of the boundary ∂K of K.

For each boundary face $E_K \in \mathcal{E}_-$, to compute the directional derivative $D_{\boldsymbol{\beta}}v(\mathbf{x}_K)$ defined in (4.1), we choose the directional derivative "mesh" size ρ such that $\mathbf{x}_K - \rho\boldsymbol{\beta}(\mathbf{x}_K)$ lies on $\mathcal{E}_-$. Then the directional derivative is given by

$$D_{\boldsymbol{\beta}}v(\mathbf{x}_K) = \frac{v(\mathbf{x}_K) - g\big(\mathbf{x}_K - \rho\boldsymbol{\beta}(\mathbf{x}_K)\big)}{\rho}, \tag{4.6}$$

where g is the given inflow boundary condition in (2.1). In a similar fashion, for problem (2.8), the discrete divergence operator at $\mathbf{z}_K$ is modified as

$$\mathbf{div}_{\mathcal{T}}\mathbf{F}(u(\mathbf{z}_K)) = \begin{cases} \dfrac{1}{|K|}\left(\mathcal{Q}_{\partial K\setminus E_K}\big(\mathbf{F}(u)\cdot\mathbf{n}\big) + \mathcal{Q}_{E_K}\big(\mathbf{F}(g)\cdot\mathbf{n}\big)\right), & E_K \in \mathcal{E}_-, \\[2ex] \dfrac{1}{|K|}\left(\mathcal{Q}_{\partial K\setminus E_K}\big(\mathbf{F}(u)\cdot\mathbf{n}\big) + \mathcal{Q}_{E_K}\big(\mathbf{F}(u_0)\cdot\mathbf{n}\big)\right), & E_K \in \mathcal{E}_0. \end{cases} \tag{4.7}$$

where g and u_0 are the given inflow boundary and initial conditions in (2.8), respectively.

Denote the modified least-squares functionals by

$$\mathcal{G}_{\mathcal{T}}\big(v;\mathbf{f}\big) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left((D_{\boldsymbol{\beta}}v + \gamma v - f)^2\right) \quad \text{and} \quad \mathcal{G}_{\mathcal{T}}\big(v;\mathbf{f}\big) = \sum_{K\in\mathcal{T}} \mathcal{Q}_K\left(\mathbf{div}_{\mathcal{T}}\mathbf{F}(v)\right) \tag{4.8}$$

for problems (2.1) and (2.8), respectively, where the discrete directional and divergence operators at subdomains $K \in \mathcal{T}$, whose boundary intersects $\mathcal{E}_-$ or $\mathcal{E}_0$, are modified in the respective (4.6) and (4.7). Then the modified least-squares least-squares (LSNN) method for problems (2.1) or (2.8) is to seek $u_{{}_{N,\mathcal{T}}} \in \mathcal{M}(l)$ such that

$$\mathcal{G}_{\mathcal{T}}\big(u_{{}_{N,\mathcal{T}}};\mathbf{f}\big) = \min_{v\in\mathcal{M}(l)} \mathcal{G}_{\mathcal{T}}\big(v;\mathbf{f}\big). \tag{4.9}$$

5. Efficient and Reliable Iterative Solvers. Both $\mathcal{L}_{\mathcal{T}}\big(v;\mathbf{f}\big)$ and $\mathcal{G}_{\mathcal{T}}\big(v;\mathbf{f}\big)$ are convex as functionals of v but non-convex as functions of the NN parameters, the resulting discrete problem in (4.5) or (4.9) is then 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 [9, 10, 11].

As a nonlinear PDE, (2.8) has its own nonlinearity that deserves a special treatment. In this section, we only consider the linear problem in (2.1). To this end, we first describe algebraic structures of the resulting non-convex optimization problems in (4.5) and (4.9), that may be used for designing efficient and reliable iterative solvers.

The least-squares problems in (4.5) and (4.9) are nonlinear due to the nonlinear parameters: the biases and weights of all hidden-layers

$$\boldsymbol{\Theta} = \left\{\mathbf{r}^{(k)}\right\}_{k=1}^{l} = \left\{\left(\mathbf{r}_1^{(k)},\dots,\mathbf{r}_{n_k}^{(k)}\right)^T\right\}_{k=1}^{l} \tag{5.1}$$

with $\mathbf{r}_i^{(k)} = \left(b_i^{(k)}, \boldsymbol{\omega}_i^{(k)}\right)$ the bias and weights of the ith neuron at the kth hidden-layer. The output bias and weights

$$\mathbf{c} = (c_0, \mathbf{c}_1) = (c_0, c_1, \dots, c_{n_l}) \in \mathbb{R}^{n_l+1}$$

are referred to as the linear parameters. A least-squares problem with both the linear and nonlinear parameters are usually called as the separable nonlinear least-squares (SNLS) problem (see, e.g., [23]). There are two approaches for solving a SNLS problem: (1) block iterative methods between the linear and the nonlinear parameters as outer iteration and (2) the Variable Projection (VarPro) method of Golub-Pereyra [20] in 1973 that eliminates the linear parameters.

Since the VarPro method changes the nonlinear structure of a SNLS problem and the number of the linear parameters is often much smaller than that of the nonlinear parameters, i.e.,

$$n_l + 1 \ll \sum_{k=1}^{l} n_k \times (n_{k-1}+1),$$

this section discusses only the first approach: block iterative methods. To this end, let

$$\sigma_0(\mathbf{x}) = 1 \quad \text{and} \quad \sigma_i(\mathbf{x}) = \sigma\left(\boldsymbol{\omega}_i^{(l)}\left(\mathbf{N}^{(l-1)}\circ\cdots\circ\mathbf{N}^{(1)}(\mathbf{x})\right) + b_i^{(l)}\right). \tag{5.2}$$

Let $u_{N,\mathcal{T}} \in \mathcal{M}(l)$ be a solution of problem (4.5) or (4.9), then

$$u_{N,\mathcal{T}} = \sum_{i=0}^{n_l} c_i\sigma_i(\mathbf{x}) = \mathbf{c}^T\boldsymbol{\Sigma}(\mathbf{x}), \tag{5.3}$$

where $\boldsymbol{\Sigma}(\mathbf{x}) = (\sigma_0(\mathbf{x}),\dots,\sigma_{n_l}(\mathbf{x}))^T$, and the linear parameter $\mathbf{c} = (c_1,\dots,c_n)^T$ and the nonlinear parameter $\boldsymbol{\Theta}$ satisfy the following optimality conditions

$$\nabla_{\mathbf{c}}\mathcal{G}_{\mathcal{T}}\big(u_{N,\mathcal{T}};\mathbf{f}\big) = \mathbf{0} \quad \text{and} \quad \nabla_{\boldsymbol{\Theta}}\mathcal{G}_{\mathcal{T}}\big(u_{N,\mathcal{T}};\mathbf{f}\big) = \mathbf{0}, \tag{5.4}$$

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

Clearly, the functional $\mathcal{G}_{\mathcal{T}}\big(u_{N,\mathcal{T}};\mathbf{f}\big)$ is quadratic with respect to the linear parameters $\mathbf{c}$. Hence, the first equation in (5.4) implies the following system of linear equations

$$\boldsymbol{A}(\boldsymbol{\Theta})\,\mathbf{c} = F(\boldsymbol{\Theta}), \tag{5.5}$$

where $\boldsymbol{A}(\boldsymbol{\Theta})$ and $F(\boldsymbol{\Theta})$ 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

$$\left\{\begin{array}{rl} \boldsymbol{A}(\boldsymbol{\Theta}) = & \sum\limits_{K\in\mathcal{T}} \mathcal{Q}_K\left(\left[D_{\boldsymbol{\beta}}\boldsymbol{\Sigma}+\gamma\boldsymbol{\Sigma}\right]\left[D_{\boldsymbol{\beta}}\boldsymbol{\Sigma}+\gamma\boldsymbol{\Sigma}\right]^T\right) \\ \text{and}\quad F(\boldsymbol{\Theta}) = & \sum\limits_{K\in\mathcal{T}} \mathcal{Q}_K\left(f\left[D_{\boldsymbol{\beta}}\boldsymbol{\Sigma}+\gamma\boldsymbol{\Sigma}\right]\right), \end{array}\right. \tag{5.6}$$

respectively. Here the actions of the numerical integration and differentiation operators $\mathcal{Q}_K$ and $D_{\boldsymbol{\beta}}$ are applied component-wisely. Let $a_{ij}(\boldsymbol{\Theta})$ be the ij-element of the coefficient matrix $\boldsymbol{A}(\boldsymbol{\Theta})$, then

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

Hence, $\boldsymbol{A}(\boldsymbol{\Theta})$ is symmetric. Due to non-local supports of $\{\sigma_i\}_{i=0}^{n_l}$, $\boldsymbol{A}(\boldsymbol{\Theta})$ is dense; moreover, it could be highly ill-conditioned. This fact, in turn, implies inefficiency of the optimization methods of gradient descent type.

Similar systems of linear equations to (5.6) arise from the least-squares approximation using shallow ReLU NN [9] and the shallow Ritz method for one-dimensional diffusion and diffusion-reaction problems [10, 11]. Efficient and reliable iterative solvers in those special cases were discussed in those papers, but how to design fast iterative solvers for the linear parameters in many NN applications is important and remains an open question. When the number of the linear parameters is not very large, methods like the truncated SVD would overcome the difficulty of large condition number [9].

For the nonlinear parameters satisfying the second equation in (5.4), one may employ the commonly used first-order gradient-based methods (see, e.g., survey papers [3, 17, 34]), second-order methods (see, e.g., survey papers [3, 17, 34]), or the Gauss-Newton (GN) method [15, 29] for nonlinear least-squares optimization. Nevertheless, it is non-trivial to derive a second-order or Gauss-Newton method due to the fact that the ReLU activation function $\sigma(t)$ has only first-order weak derivative. A damped block Newton/Gauss-Newton for the second equation in (5.4) will be studied in a forthcoming paper. Basic idea follows those of recent works on fast iterative solvers introduced in [9] for the least-squares function approximation in $\mathbb{R}^d$ and in [10, 11] for the shallow Ritz method solving one dimensional diffusion and diffusion-reaction problems.

REMARK 5.1. *The resulting discrete problems in (4.5) and (4.9) are non-convex optimization, and hence initialization is critical for the success of any optimization/iterative/training scheme. The initialization issue may be addressed through (1) the physical meaning of the linear and non-linear 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 (5.5) with fixed* $\mathbf{r}$ *that is a linear problem (see [5, 4, 9, 10, 11]).*

The adaptive neuron enhancement (ANE) method introduced in [27, 12] provides a natural method of continuation. The method of model continuation for linear advection-reaction problems with variable advection field was studied in [5]. Finally, the method of subdomain continuation for the block space-time LSNN method was introduced in [4] 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 [24]. [1] 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

[1]The second-order Gauss-Newton method as discussed in Section 5 is not implemented in this chapter.

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 $h_x = h_y = 0.01$. The directional derivative $v_{\boldsymbol{\beta}}$ was approximated by the backward finite difference quotient (4.1) with $\rho = h_x/2$. The numerical results after 200,000 Adam iterations is reported in Figure 2 and Table 1. As shown in Figures 2(b) to 2(d), the LSNN method is capable of approximating the discontinuous solution with a curved interface and non-constant jump accurately without any oscillation or overshooting. In Figure 2(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 2(a)) presented in the problem, allowing the discontinuous solution to be accurately approximated using a piecewise linear function. Table 1 lists the relative numerical errors measured in different norms. With 3841 parameters (DoFs), the ReLU NN can accurately approximate the solution with reasonable accuracy.

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

Network structure	$\frac{\|u-u_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-u_{\mathcal{T}}^N\|\|\|_{\boldsymbol{\beta}}}{\|\|\|u\|\|\|_{\boldsymbol{\beta}}}$	$\frac{\mathcal{L}^{1/2}(u_{\mathcal{T}}^N,\mathbf{f})}{\mathcal{L}^{1/2}(u_{\mathcal{T}}^N,\mathbf{0})}$	Parameters
2–60–60–1	0.07184	0.1145	0.02609	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 $u_L = 1 > 0 = u_R$. 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 $\Omega_{1,2} = (-1,1)\times(0.2,0.4)$ during LSNN training, to allow for an efficient computation. The numerical integration is performed using a uniform grid of size $h_x = h_t = 0.01$. For the discrete divergence operator $\mathbf{div}_{\mathcal{T}}$ (4.7), two quadrature methods were tested for calculating the line integral $\mathcal{Q}_{\partial K}(\cdot)$: the composite trapezoidal rule and the midpoint rule. Furthermore, the impact of the number of sub-intervals used, along each boundary edge of ∂K, on the precision of the LSNN method was investigated.

A 2–10–10–1 ReLU NN model was used as an approximate function, and the Adam optimizer trained 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. 3.

From Tables 2, it is expected that the accuracy of the LSNN method depends on the number of sub-intervals ($\hat{m}$ and $\hat{n}$ are the corresponding number of sub-intervals along the spatial and temporal directions, respectively); that is, the larger the $\hat{m}$ and $\hat{n}$, the more accurate the LSNN method is. Moreover, the accuracy using the composite trapezoidal and mid-point rules in the LSNN method is comparable, both are capable of simulating this Riemann problem with accurate shock propagating 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 piece-wise constant initial data

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

we refer the readers to an exact solution to this problem in [21].

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.4 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 blocked space-time method. 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 [8]. 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.

REFERENCES

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

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

[2] B. Bochev and M. Gunzburger. Least-squares methods for hyperbolic problems. *Handbook of Numerical Analysis*, 17:289–317, 2016.
[3] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.
[4] 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 (2023) 115298, arXiv:2110.10895v3 [math.NA].
[5] Z. Cai, J. Chen, and M. Liu. Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *J. Comput. Phys.*, 443 (2021) 110514.
[6] 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.
[7] 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.
[8] Z. Cai, J. Choi, and M. Liu. ReLU neural network approximation to piecewise constant functions. *arXiv:2410.16506 [math.FA]*, 2024.
[9] 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.
[10] Z. Cai, A. Doktorova, R. D. Falgout, and C. Herrera. Fast iterative solver for neural network method: I. 1D diffusion problems. *arXiv:2404.17750 [math.NA]*, 2024.
[11] Z. Cai, A. Doktorova, R. D. Falgout, and C. Herrera. Fast iterative solver for neural network method: II. 1D diffusion-reaction roblems and "data fitting". *arXiv:2407.01496 [math.NA]*, 2024.
[12] 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.
[13] 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 J. Sci. Comput.*, 26(1):31–54, 2004.
[14] H. De Sterck, T. A. Manteuffel, S. F. McCormick, and L. Olson. Numerical conservation properties of H(div)-conforming least-squares finite element methods for the Burgers equation. *SIAM J. Sci. Comput.*, 26(5):1573–1597, 2005.
[15] J. E. Dennis Jr and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* SIAM, Philadelphia, 1996.
[16] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Comm. Math. Stat.*, 6(1):1–12, 2018.
[17] C. Gambella, B. Ghaddar, and J. Naoum-Sawaya. Optimization problems for machine learning: a survey. *arXiv:1901.05331 [math.OC]*, 2019.
[18] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws.* Springer, New York, 1996.
[19] E. Godlewski and P.-A. Raviart. *Numerical Approximation of Hyperbolic Systems of Conservation Laws*, volume 118. Springer Science & Business Media, 2013.
[20] 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.
[21] J.-L. Guermond and M. Nazarov. A maximum-principle preserving C^0 finite element method for scalar conservation equations. *Comput. Methods Appl. Mech. Engrg.*, 272:198–213, 2014.
[22] J. S. Hesthaven. *Numerical Methods for Conservation Laws: From Analysis to Algorithms.* SIAM, Philadelphia, 2017.
[23] L. Kaufman. A variable projected method for solving separable nonlinear least squares problems. *BIT*, 15:49–57, 1975.
[24] 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.
[25] R. J. LeVeque. *Numerical Methods for Conservation Laws.* Birkhäuser, Boston, 1992.
[26] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems.* Cambridge University Press, Cambridge, 2002.
[27] 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.

[28] M. Liu, Z. Cai, and K. Ramani. Deep Ritz method with adaptive quadrature for linear elasticity. *Comput. Methods Appl. Mech. Engrg.*, 415 (2023) 116229.

[29] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* SIAM, Philadelphia, 2000.

[30] 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.

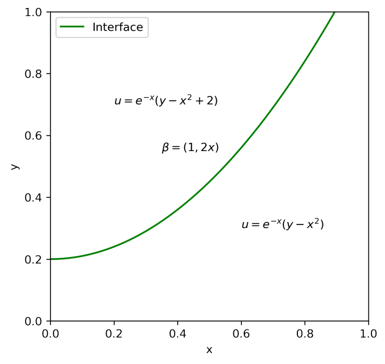
[31] C.-W. Shu. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations*, pages 325–432. Springer, 1998.

[32] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1139–1364, 2018.

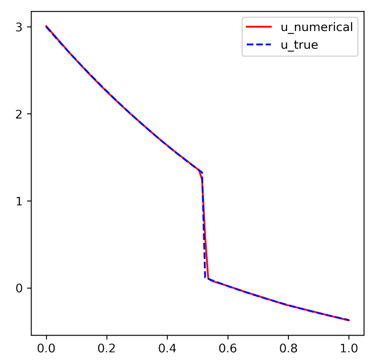
[33] A. H. Stroud. *Approximate Calculation of Multiple Integrals.* Englewood Cliffs, N.J.: Prentice-Hall, 1971.

[34] S. Sun, Z. Cao, H. Zhu, and J. Zhao. A survey of optimization methods from a machine learning perspective. *IEEE Trans. on Cybernetics*, 50(8):3668 – 3681, 2020.

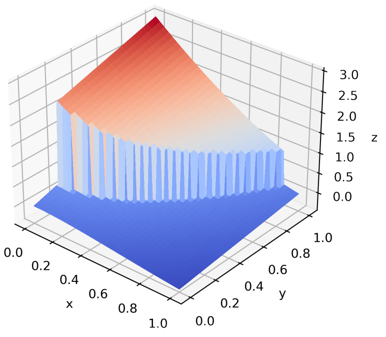
[35] J. W. Thomas. *Numerical Partial Differential Equations: Finite Difference Methods*, volume 22. Springer Science & Business Media, 2013.



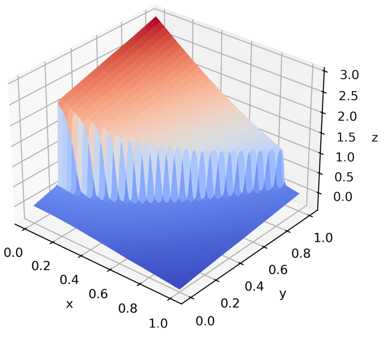
(a) The interface



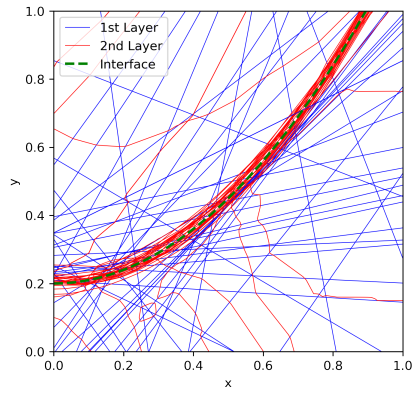
(b) The trace of Figure 2(d) on $y = 1 - x$



(c) The exact solution

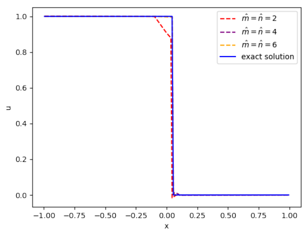


(d) A 2–60–60–1 ReLU NN function approximation

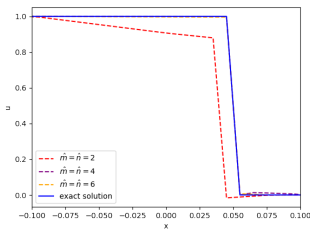


(e) The breaking hyper-planes of the approximation in Figure 2(d)

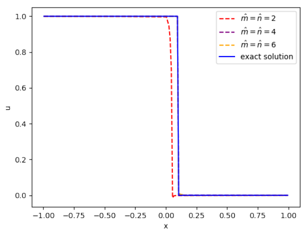
FIG. 2. *Approximation results for the linear advection-reaction problem in Sec. 6.1.*



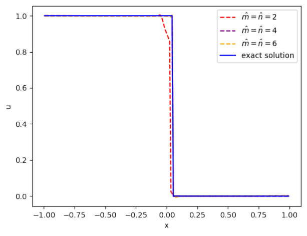
(a) Traces at $t = 0.2$ (trapezoidal)



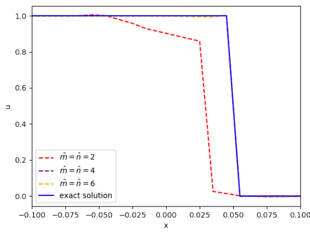
(b) Zoom-in plot near the discontinuous interface of sub-figure (a)



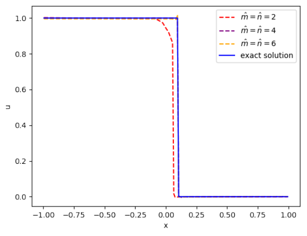
(c) Traces at $t = 0.4$ (trapezoidal)



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

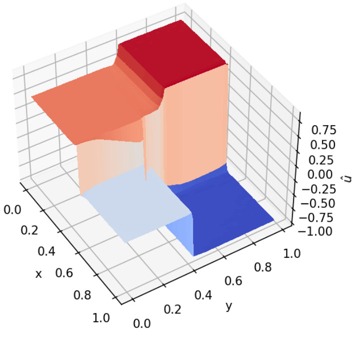


(e) Zoom-in plot near the discontinuous interface of sub-figure (d)

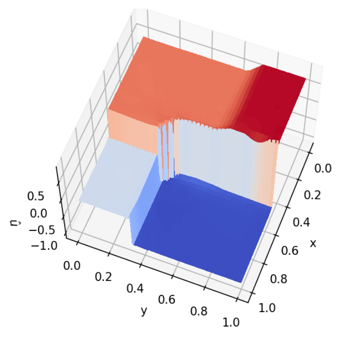


(f) Traces at $t = 0.4$ (mid-point)

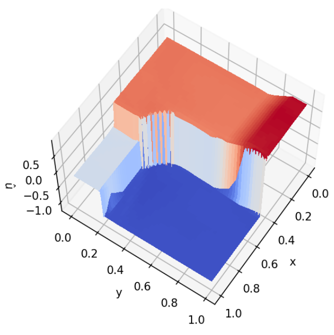
FIG. 3. *Numerical results of the problem with $f(u) = \frac{1}{4}u^4$ using the composite trapezoidal and mid-point rules*



(a) $t = 0.1$



(b) $t = 0.3$



(c) $t = 0.5$

FIG. 4. *Numerical results of 2D Burgers' equation.*