LEAST-SQUARES RELU NEURAL NETWORK METHOD FOR SCALAR HYPERBOLIC CONSERVATION LAW

by

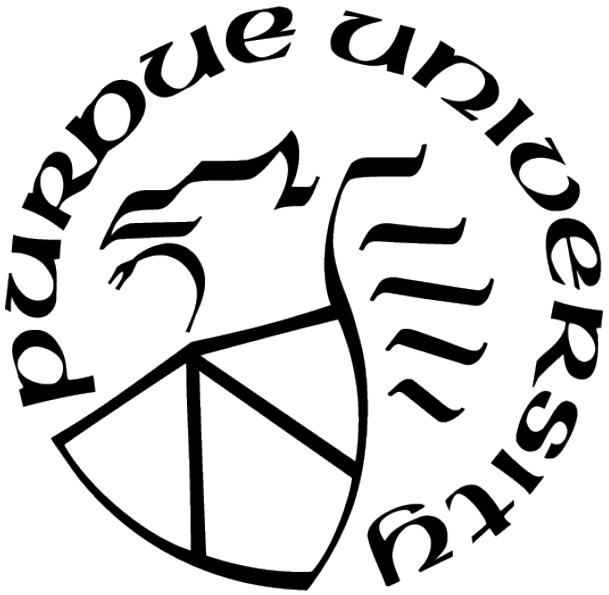
Jingshuang Chen

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



Department of Mathematics

West Lafayette, Indiana

May 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Zhiqiang Cai, Chair

Department of Mathematics

Dr. Jie Shen

Department of Mathematics

Dr. Xiangxiong Zhang

Department of Mathematics

Dr. Min Liu

School of Mechanical Engineering

Approved by:

Dr. Plamen Stefanov

Dedicated to my family for their unconditional love and support.

ACKNOWLEDGMENTS

First, I would like to thank my Ph.D advisor, Dr. Zhiqiang Cai, who introduced me to the field of numerical analysis and always encourages me during my Ph.D. I would not have made it without his academic guidance and continuous support. Also, I would like to extend my gratitude to my committee members. Thank you Dr. Min Liu for providing valuable suggestions and help in coding. Dr. Jie Shen and Dr. Xiangxiong Zhang: your insightful comments and expertise enlighten my thesis with different perspectives. In addition, I would like to thank all the knowledge professors I met in Purdue and I learned a lot from them. Furthermore, I extend my appreciation to the Department of Mathematics for creating such a good academic atmosphere and providing financial support for my Ph.D study.

Also, many thanks to my friends for making my six years at Purdue enjoyable. In particular, I would like to thank Dr. Difeng Cai, Xinyu Liu, Xiaodong Huang and Jiahao Zhang for discussing research problems. I am also grateful to my officemates, Wei Deng and Dr. Joan Ponce. Special thanks also goes to my former roommate Yun Huang, and classmates. Thanks for sharing the wonderful time during my six years at Purdue. Finally, I was so lucky to meet Dali here at Purdue. Thanks for your love and support throughout my Ph.D.

Most importantly, I would like to thank my family, especially my parents and grandmother, for the continuous support and love they have given me throughout my life. I could not complete my Ph.D without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\|\cdot\|$	L^2 norm (unless otherwise stated)
$(\cdot,\cdot)$	inner product
$[\![\cdot]\!]$	jump of a function on the interface
Ω	a computational domain
$\partial\Omega$	boundary of Ω
Γ_-	inflow boundary
$\mathcal{T}$	an integration mesh of the computational domain Ω
∇	gradient operator
$\nabla\cdot$	divergence operator
$\boldsymbol{n}$	a unit outward normal vector to the boundary of the computational domain Ω
$\boldsymbol{\beta}$	advection velocity field
$\mathcal{N}$	a deep neural network
$\boldsymbol{\theta}$	parameters of a neural network
$\mathcal{M}(\boldsymbol{\theta}, L)$	set of DNN functions
σ	DNN activation function
$\boldsymbol{\omega}$	DNN weights
$\boldsymbol{b}$	DNN bias
$\bar{u}_{\mathcal{T}}^N$	DNN approximation (unless otherwise stated)

ABSTRACT

The thesis introduces the least-squares ReLU neural network method for solving scalar hyperbolic conservation laws with discontinuous solutions. The method is a discretization of an equivalent least-squares formulation in the set of neural network functions with the ReLU activation function. Evaluation of the LS functional is done by using numerical integration and proper finite difference/volume scheme.

We theoretically and numerically show that the least-squares ReLU neural network is capable of approximating the discontinuous interface of the underlying problem automatically through the free hyper-planes of the ReLU neural network and, hence, outperforms the traditional mesh-based numerical methods in terms of the number of degrees of freedom. Numerical results of some benchmark test problems for linear advection-reaction equations as well as nonlinear equations show that the method can not only accurately approximate the solution with the least number of parameters, but also avoid the common Gibbs phenomena along the discontinuous interface.

1. INTRODUCTION

Recently, deep neural networks (DNNs) have achieved astonishing performance in computer vision, natural language processing, and many other artificial intelligence tasks. A special feature of DNN is its new way to approximate functions through a composition of multiple linear and activation functions. This feature leads to wide applications to other fields, including some recent studies (see, e.g., [1]–[5]) of using DNN models to numerically solve partial differential equations (PDEs).

The idea of solving differential equations using neural networks may be traced back to a paper in 1994 by Dissanayake and Phan-Thien [6]. For a differential equation $\mathcal{L}(u) = 0$ defined on the domain Ω with boundary condition $B(u) = 0$ on $\partial\Omega$, a neural network was trained to minimize the following least-square functional which is defined based on the original PDEs:

$$\tilde{\mathcal{L}}(v) = \int_{\Omega} \left|\mathcal{L}(v)(x)\right|^2 dx + \int_{\partial\Omega} \left|B(v)(x)\right|^2 ds \equiv \|\mathcal{L}(v)\|_{0,\Omega}^2 + \|B(v)\|_{0,\partial\Omega}^2, \tag{1.1}$$

where $\|\cdot\|_{0,S}$ is the L^2 norm over subdomain $S = \Omega$ or $\partial\Omega$. Several follow-up works use similar ideas with one hidden layer and sampling points from a mesh to numerically approximate the integrals in $\tilde{\mathcal{L}}$ at each iteration [7]–[9]. More recently, there is a limited emerging literature on the use of deeper hidden layers to solve PDEs [1], [3], [5]. It is also illustrated that the sampling points can be obtained by a random sampling of the domain rather than using a physical mesh, which is beneficial in higher-dimensional problem [1], [5].

The purpose of the thesis is to introduce a numerical approach which uses deep neural networks to solve PDEs. The approach makes use of a deep neural network to approximate the solutions of PDEs through the compositional construction and employs appropriate loss function to determine parameters of the deep neural network through an iterative process. Specifically, we restrict our attention on using the least-squares formulation to solve scalar hyperbolic conservation laws in the thesis.

The least-squares (LS) methodology has been intensively studied for many PDEs including problems arising from solid and fluid dynamics, transport, magnetohydrodynamics, etc.

The two striking features of the least-squares method are (i) it naturally symmetrizes and stabilizes the original problem; and (ii) value of the corresponding LS functional of the current approximation is an accurate a posteriori error estimator. The first property enables us to work on complex systems which might not have underlying minimization principles, and the second one provides feedback for automatically controlling numerical processes such as the location of quadrature points for evaluating LS functional and the neural network structure [10].

During the past several decades, numerical methods for scalar hyperbolic conservation law have been intensively studied by many researchers and many numerical schemes have been developed. For linear advection-reaction problems, when the inflow boundary data is discontinuous, so is the solution. For the nonlinear scalar hyperbolic conservation law, the solution is often discontinuous as well due to the discontinuous initial condition or the shock formation. It is well-known that traditional mesh-based numerical methods often exhibit oscillations near discontinuity interface (called the Gibbs phenomena). Usually, such spurious oscillations are unacceptable for many applications (see, e.g, [11]). To eliminate or reduce the Gibbs phenomena, finite difference and finite volume methods often use numerical techniques such as limiters, filters, Roe, ENO/WENO, etc. [11]–[15]; and finite element methods usually employ discontinuous finite elements [16]–[18] or adaptive mesh refinement (AMR) to generate locally refined elements along discontinuous interfaces (see, e.g., [19]–[22]).

Instead of using the traditional mesh-based method, the thesis focuses on the discussion about employing the least-squares functional as the loss function of the DNN for solving scalar hyperbolic conservation laws. The main reason for choosing DNNs is because of its powerful approximation property and it can output compositions of functions cheaply. As we observed, one of the striking features of DNN is that it generates a class of functions which is not subject to a hand-crafted geometric mesh or point cloud as the traditional, well-studied finite difference, finite volume, and finite element methods. DNN partitions the computational domain Ω by using free hyper-planes, which is capable of automatically adapting to the target function. For instance, the location of the hyper-planes can be adjusted according to the location of the discontinuity interface through an iterative process

(see section 3.5). On the other hand, the mesh generated by the AMR strategy is based on a geometric mesh and subject to mesh conformity requirement. In addition, it is not an easy task to remove unnecessary elements or points once the discontinuity interface is found. From this perspective, DNN is favorable for approximating the discontinuous solutions of scalar hyperbolic conservation laws.

The study on DNN approximation theory draws much attention in recent years and we present a short list of some papers studying the approximation theory of ReLU DNNs. In [23], [24], the authors showed that a ReLU DNN with at most $\lceil \log_2(d+1) \rceil$ hidden layers can represent a piecewise linear function $\mathbb{R}^d \to \mathbb{R}$. Besides, with tailored neural network structures and more neurons as well as layers in the network, ReLU DNN is capable of approximating a large class of functions other than linear [25]–[27]. In addition, the authors theoretically show that the performance of neural networks possess greater approximation power than the traditional methods of nonlinear approximation [25]. Despite the fact that the approximation theory of ReLU neural networks has been intensively studied, to the best of our knowledge, we did not find a result which is applicable to the discontinuous solutions of scalar hyperbolic conservation laws.

On the other hand, however, the powerful approximation property of DNN comes with a price. The procedure for determining the values of the parameters of the network is now a problem in non-convex optimization even though the underlying PDE is linear. In practice, this high dimensional, non-convex optimization problem tends to be computationally intensive and complicated. So far, we have limited understanding about the optimization process and it could be a future research direction. Current method is to use the iterative optimization methods such as gradient descent (GD), Stochastic GD, Adam, etc. (see, e.g., [28] for a review paper in 2018 and references therein). To obtain a desired solution from the optimization problems, the only approach is to start from a close enough first approximation since the non-convex optimization problems usually have many solutions. To achieve that, we propose the initialization strategy for a two-layer neural network (see section 3.4) as well as the method of model continuation (see section 3.6). We present numerical results to demonstrate such continuation method is able to reduce the total number of the network parameters in a large scale.

The major contributions of the thesis include the following:

1. Set up the framework for the least-squares neural network (LSNN) method to solve scalar hyperbolic conservation laws. For the linear advection-reaction problems, the numerical results indicate that the method is able to not only outperform the traditional mesh-based numerical methods in terms of the number of degrees of freedom, but also avoid the common Gibbs phenomena along the discontinuous interface without any post-processing techniques. For the nonlinear problems, a modified LSNN method is proposed and the DNN approximation results suggest the method is capable of resolving the shock;

2. Show theoretically that a three-layer ReLU neural network is sufficient to approximate the discontinuous solutions accurately without oscillation;

3. Introduce the initialization strategy and the method of model continuation for providing a good starting point for training the neural network.

The rest of the thesis is organized as follows. We introduce the deep neural network structure through the functional terminology in Chapter 2. Chapter 3 presents a general framework of the least-squares neural network method for linear advection-reaction equations. The approximation property of the ReLU neural network is analyzed and the numerical experiments are presented to demonstrate the advantage over the traditional mesh-based approaches. In Chapter 4, we employ the method for nonlinear scalar hyperbolic conservation law and introduce the modified block space-time least-squares neural network method. Numerical experiments for the benchmark one dimensional problems are presented to show the capability of the LSNN method for resolving the shock. Finally, concluding remarks, limitations and future research work are presented in Chapter 5.

Throughout the thesis, the standard notations and definitions are used for the Sobolev space $H^s(\Omega)^d$ and $H^s(\Gamma_-)^d$ when $s \geq 0$. The associated norms with these two spaces are denoted by $\|\cdot\|_{s,\Omega}$ and $\|\cdot\|_{s,\Gamma_-}$, and their respective inner products are denoted as $(\cdot,\cdot)_{s,\Omega}$ and $(\cdot,\cdot)_{s,\Gamma_-}$. For $s = 0$ case, $H^s(\Omega)^d$ is the same as $L^2(\Omega)^d$, then the norm and inner product

are simply denoted as $\|\cdot\|$ and $(\cdot,\cdot)$, respectively. The subscripts Ω in the designation of norms will be suppressed when there is no ambiguity.

2. DEEP NEURAL NETWORK STRUCTURE

In this chapter, we briefly describe the deep neural network (DNN) structure through the functional terminology. To this end, we consider a deep neural network (DNN) with a multi-dimensional output:

$$\mathcal{N} : \mathbf{x} \in \mathbb{R}^d \longrightarrow y = \mathcal{N}(\mathbf{x}) \in \mathbb{R}^c,$$

where d and c are dimensions of input $\mathbf{x} \in \mathbb{R}^d$ and output $y = \mathcal{N}(\mathbf{x}) \in \mathbb{R}^c$, respectively. A special feature of DNN is its new way to approximate functions through a composition of multiple linear and activation functions. Typically, the DNN function $\mathcal{N}(\mathbf{x})$ is represented as compositions of many different layers of functions:

$$y = \mathcal{N}(\mathbf{x}) = N^{(L)} \circ \cdots N^{(2)} \circ N^{(1)}(\mathbf{x}), \tag{2.1}$$

where the symbol $\circ$ denotes the composition of functions: $f \circ g(x) = f(g(x))$, and L is the depth of the network. In this case, $N^{(1)}$ is called the first layer of the neural network, $N^{(2)}$ is called the second layer, and so on. All layers except for the last one $N^{(L)}$ are called hidden layers since they are hidden in between input and output layers. Figure 2.1 presents an illustration of a deep neural network.

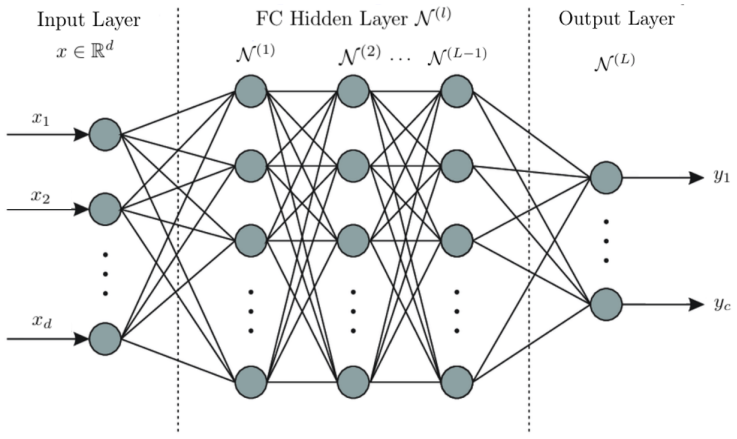


Figure 2.1. Fully-Connected Neural Network with each circle representing a neuron

Each layer is typically a vector-valued function. The choice of the function $N^{(l)}(\mathbf{x}^{(l-1)})$ is guided by many mathematical and engineering disciplines. Specifically, throughout the thesis, we employ fully connected (FC) hidden layers for the network. A FC layer $N^{(l)} : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$ is defined as a composition of a linear transformation $T^{(l)} : \mathbb{R}^{n_{l-1}} \to \mathbb{R}^{n_l}$ and an activation function $\sigma^{(l)} : \mathbb{R} \to \mathbb{R}$ as follows:

$$N^{(l)}(\mathbf{x}^{(l-1)}) = \sigma^{(l)}\Big(T^{(l)}(\mathbf{x}^{(l-1)})\Big) = \sigma^{(l)}(\boldsymbol{\omega}^{(l)}\mathbf{x}^{(l-1)} - \mathbf{b}^{(l)}) \quad \text{for } \mathbf{x}^{(l-1)} \in \mathbb{R}^{n_{l-1}}, \tag{2.2}$$

where $\boldsymbol{\omega}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$, $\mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$, $\mathbf{x}^{(0)} = \mathbf{x}$, and application of $\sigma^{(l)}$ to a vector is defined component-wise. There is typically no activation function in the output layer. Components of $\boldsymbol{\omega}^{(l)}$ and $\mathbf{b}^{(l)}$ are called weights and bias, respectively, and are parameters to be determined (trained). Each component of the vector-valued function $N^{(l)}$ is interpreted as a neuron and the dimension n_l defines the width or the number of neurons of the l^{th} layer in a network. The $n_0 = d$ and $n_L = c$ are the respective dimensions of input and output. There are $n_l \times (n_{l-1} + 1)$ parameters at the l^{th} layer, and the total number of parameters of the DNN function $\mathcal{N}(x)$ defined in (2.1) is given by

$$N = \sum_{l=1}^{L} n_l \times (n_{l-1} + 1). \tag{2.3}$$

Choices of the activation function σ have influences on the model output, approximation accuracy, and computational efficiency of the training. A widely used activation function is the rectified linear unit (ReLU) given by

$$\sigma(t) = \max\{0, t\} = \begin{cases} 0, & \text{if } t \leq 0, \\ t, & \text{if } t > 0. \end{cases} \tag{2.4}$$

Another commonly used activation function is a modification of ReLU activation and it is termed "leaky ReLU":

$$\sigma(t) = \begin{cases} 0.01t, & \text{if } t \leq 0, \\ t, & \text{if } t > 0. \end{cases}$$

Both ReLU and leaky ReLU activation are piece-wise linear functions. The reason for modifying ReLU activation function is to avoid possible gradient vanishing issue in the training process. Besides the linear activation functions, sigmoid function is also widely used in practice. Specifically, the sigmoid activation function is defined as:

$$\sigma(t) = \frac{1}{1 + \mathrm{e}^{-t}}, \quad t \in \mathbb{R}. \tag{2.5}$$

All three activation functions are depicted in Figure 2.2. Both ReLU and leaky ReLU are easier to compute compared with the non-linear sigmoid function. But using a smooth activation function such as the sigmoid function is essential for some applications. More details can be found in [2].

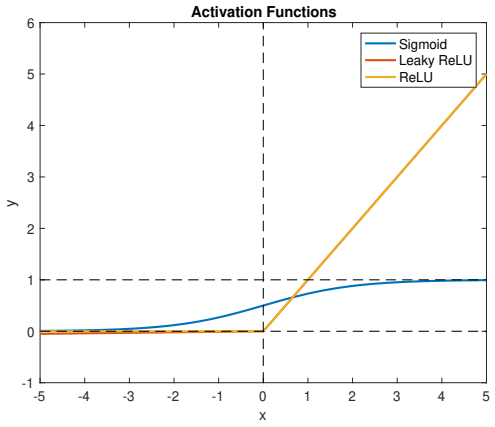


Figure 2.2. Activation functions

Besides the deep neural network structure introduced in this chapter, there are some other types of network models which are commonly used in practice. For instance, the convolution network network (CNN) is employed mainly for image processing and the recurrent network network (RNN) is most commonly applied to natural language processing. The discussion of the applications for these neural network structures are beyond the scope of the thesis. In particular, we will use the fully-connected neural network with ReLU as the activation function in the thesis.

3. LEAST-SQUARES NEURAL NETWORK METHOD FOR LINEAR ADVECTION-REACTION EQUATIONS

A version of this chapter has been submitted for publication [29].

In this chapter, we introduce the least-squares ReLU neural network (LSNN) method for solving the linear advection-reaction problem with discontinuous solution. Since DNN functions are nonlinear functions of the parameters, then the advection-reaction equation will be discretized through least-squares principles. In the context of finite element approximations, several least-squares methods have been studied (see, e.g., [22], [30]–[35]). Basically, there are two least-squares formulations which are equivalent to the original differential equation. One is a direct application of least-squares principle (see, e.g., [30], [34]) with a weighted L^2 norm for the inflow boundary condition, where the weight is the magnitude of the normal component of the advection velocity field. The other is to apply the least-squares principle to an equivalent system of the underlying problem by introducing an additional flux variable (see [22], [35]). Some numerical techniques such as feedback least-squares finite element method [31], adaptive local mesh refinement with proper finite elements [22], etc. were introduced in order to reduce the common Gibbs phenomena for problems with discontinuous solutions.

The LSNN method is based on the least-squares formulation studied in ([30], [34]), i.e., a direct application of the lease-squares principle to the underlying problem, and on the ReLU neural network as the approximation class of functions. The class of neural network functions enables the LSNN method to automatically approximate the discontinuous solution without using *a priori* knowledge of the location of the discontinuities. Compared to various AMR methods that locate the discontinuous interface through local mesh refinement, the LSNN method is much more effective in terms of the number of the degrees of freedom (see, e.g., Fig. 3.1(c) and Fig. 3.2(c)).

Theoretically, it is proved in [34] that the homogeneous least-squares functional is equivalent to a natural norm in the solution space $V_{\boldsymbol{\beta}}$ consisting of all square-integrable functions whose directional derivative along $\boldsymbol{\beta}$ is also square-integrable (see section 3.1). This equivalence enables us to prove Ceá's lemma for the LSNN approximation, i.e., the error of the

LSNN approximation is bounded by the approximation error of the set of ReLU neural network functions. This result is extended to the LSNN method with numerical integration as well. Even though approximation theory of the ReLU neural network has been intensively studied by many researchers (see, e.g., [36] for work before 2000 and [37], [38]), we are not able to find a result which is applicable to the discontinuous solution of the advection-reaction problem.

To explore how well the ReLU neural network approximates the discontinuous solution, we consider two-dimensional transport problems, i.e., (3.4) with $\hat{\gamma} = 0$. When the boundary data g is discontinuous at point $\mathbf{x}_0 \in \Gamma_-$, the solution of the transport problem is discontinuous across an interface: the streamline of the advection velocity field starting at $\mathbf{x}_0$. The solution of this problem can be decomposed as the sum of a piece-wise constant function and a continuous piece-wise smooth function (see, e.g., (3.19)). We show that the piece-wise constant function can be approximated well without the Gibbs phenomena by either a two- or a three-layer ReLU neural network with the minimal number of neurons depending on the geometric property of the interface (see Lemmas 3.3.2 and 3.7.1). Together with the universal approximation property, this implies that a two- or three-layer ReLU neural network is sufficient to well approximate the solution of the linear transport problem without oscillation. These theoretical results are confirmed by the numerical results in section 3.5.

However, the procedure for determining the values of the parameters of the network becomes a non-convex optimization even though the underlying PDE is linear. In order to obtain the desired solution given the fact that the non-convex optimization problem usually has many solutions, it is required to start from a close enough first approximation and a common way to do so is by the method of continuation. In this chapter, we also propose the method of model continuation through approximating the advection velocity field by a family of piece-wise constant vector fields. Numerical results for a test problem with a variable velocity field show that this method is able to reduce the total number of the parameters in a large scale.

This chapter is organized as follows. Section 3.1 introduces the advection-reaction problem formulation, its least-squares formulation, and preliminaries. The least-squares neural network method is described in section 3.2. The ReLU NN approximation of discontinuous

solutions is analyzed in section 3.3 and section 3.7. Initialization for the two-layer neural networks and the method of model continuation for initialization are presented in sections 3.4 and 3.6, respectively. Finally, numerical results for various benchmark test problems are given in section 3.5.

3.1 Problem Formulation

Let Ω be a bounded domain in $\mathbb{R}^d$ with Lipschitz boundary, and denote the advective velocity field by $\boldsymbol{\beta}(\mathbf{x}) = (\beta_1, \cdots, \beta_d)^T \in C^1(\bar{\Omega})^d$. Define the inflow and outflow parts of the boundary $\Gamma = \partial\Omega$ by

$$\Gamma_- = \{\mathbf{x} \in \Gamma : \boldsymbol{\beta}(\mathbf{x}) \cdot \boldsymbol{n}(\mathbf{x}) < 0\} \quad \text{and} \quad \Gamma_+ = \{\mathbf{x} \in \Gamma : \boldsymbol{\beta}(\mathbf{x}) \cdot \boldsymbol{n}(\mathbf{x}) > 0\}, \tag{3.1}$$

respectively, where $\boldsymbol{n}(\mathbf{x})$ is the unit outward normal vector to Γ at $\mathbf{x} \in \Gamma$.

As a model hyperbolic boundary value problem, we consider the linear advection-reaction equation

$$\begin{cases} \nabla\cdot(\boldsymbol{\beta} u) + \gamma u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_-, \end{cases} \tag{3.2}$$

where $\gamma \in C(\bar{\Omega})$, $f \in L^2(\Omega)$, and $g \in L^2(\Gamma_-)$ are given scalar-valued functions. As usual, we assume that there exist a positive constant γ_0 such that

$$\gamma(\mathbf{x}) + \frac{1}{2}\nabla \cdot \boldsymbol{\beta}(\mathbf{x}) \geq \gamma_0 > 0 \quad \text{for all } \mathbf{x} \in \Omega. \tag{3.3}$$

For simplicity of presentation, we also assume that g is bounded so that streamline functions from Γ_- to Γ_+ is not needed (see [34]).

Denote by $v_{\boldsymbol{\beta}} = \boldsymbol{\beta} \cdot \nabla v$ the directional derivative along the advective velocity field $\boldsymbol{\beta}$, then (3.2) may be rewritten as follows

$$\begin{cases} u_{\boldsymbol{\beta}} + \hat{\gamma}\, u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_-, \end{cases} \tag{3.4}$$

where $\hat{\gamma} = \gamma + \nabla \cdot \boldsymbol{\beta}$. The solution space of (3.2) is given by

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

which is equipped with the norm as

$$|||v|||_{\boldsymbol{\beta}} = \left(\|v\|_{0,\Omega}^2 + \|v_{\boldsymbol{\beta}}\|_{0,\Omega}^2\right)^{1/2}.$$

Denote the weighted $L^2(\Gamma_-)$ norm over the inflow boundary 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 following trace and Poincaré inequalities are proved in [34] (see also [31]) that there exist positive constants C_t and C_p such that

$$\|v\|_{-\boldsymbol{\beta}} \le C_t\, |||v|||_{\boldsymbol{\beta}}, \quad \forall\ v \in V_{\boldsymbol{\beta}} \tag{3.5}$$

and

$$\|v\|_{0,\Omega} \le C_p \left(\|v\|_{-\boldsymbol{\beta}} + D\, \|v_{\boldsymbol{\beta}}\|_{0,\Omega}\right), \quad \forall\, v \in V_{\boldsymbol{\beta}}, \tag{3.6}$$

respectively, where $D = \mathrm{diam}(\Omega)$ is the diameter of the domain Ω.

Remark 3.1.1. *Let $\mathcal{C}$ be the streamline of the advection velocity field $\boldsymbol{\beta}$ starting at $\mathbf{x}_0 \in \Gamma_-$ in two dimensions. Assume that the inflow boundary condition g is discontinuous at $\mathbf{x}_0$. Then it is easy to see that the solution of (3.2) is also discontinuous across $\mathcal{C}$ because the restriction of the solution on $\mathcal{C}$ satisfies the same differential equation but different initial condition. Moreover, if $\hat{\gamma} = 0$, then the jump of the solution along $\mathcal{C}$ is a constant $|g(\mathbf{x}_0^+) - g(\mathbf{x}_0^-)|$, where $g(\mathbf{x}_0^+)$ and $g(\mathbf{x}_0^-)$ are the values of g at $\mathbf{x}_0$ from different sides. The streamline $\mathcal{C}$ is referred to be the discontinuous interface.*

In the remainder section, we introduce the least-squares (LS) formulation for the linear advection-reaction equations (3.4) following [31], [34]. To this end, we define the LS functional

$$\mathcal{L}(v;\mathbf{f}) = \|v_{\boldsymbol{\beta}} + \hat{\gamma}\, v - f\|_{0,\Omega}^2 + \|v - g\|_{-\boldsymbol{\beta}}^2 \tag{3.7}$$

for all $v \in V_{\boldsymbol{\beta}}$, where $\mathbf{f} = (f, g)$. Now, the corresponding least-squares formulation is to seek $u \in V_{\boldsymbol{\beta}}$ such that

$$\mathcal{L}(u;\mathbf{f}) = \min_{v \in V_{\boldsymbol{\beta}}} \mathcal{L}(v;\mathbf{f}). \tag{3.8}$$

It follows from the trace, triangle, and Poincaré inequalities and assumptions on $\boldsymbol{\beta}$ and γ that the homogeneous LS functional $\mathcal{L}(v;\mathbf{0})$ is equivalent to the norm $|||v|||_{\boldsymbol{\beta}}^2$, i.e., there exist positive constants α and M such that

$$\alpha\, |||v|||_{\boldsymbol{\beta}}^2 \leq \mathcal{L}(v;\mathbf{0}) \leq M\, |||v|||_{\boldsymbol{\beta}}^2. \tag{3.9}$$

In addition, the problem (3.8) has a unique solution $u \in V_{\boldsymbol{\beta}}$ which satisfies the following *a priori* estimate

$$|||u|||_{\boldsymbol{\beta}} \leq C\, \left(\|f\|_{0,\Omega} + \|g\|_{-\boldsymbol{\beta}}\right). \tag{3.10}$$

Denote the bilinear and linear forms by

$$a(u,v) = (u_{\boldsymbol{\beta}} + \hat{\gamma}\, u, v_{\boldsymbol{\beta}} + \hat{\gamma}\, v) + \langle u, v\rangle_{-\boldsymbol{\beta}} \quad \text{and} \quad f(v) = (f, v_{\boldsymbol{\beta}} + \hat{\gamma}\, v) + \langle g, v\rangle_{-\boldsymbol{\beta}}\,,$$

respectively. Then the minimization problem in (3.8) is to find $u \in V_{\boldsymbol{\beta}}$ such that

$$a(u,v) = f(v), \quad \forall\ v \in V_{\boldsymbol{\beta}}. \tag{3.11}$$

3.2 LSNN Method for Linear Advection-Reaction Equations

The main idea of the least-squares neural network (LSNN) method is to employ DNN functions for approximating the solution $u(\mathbf{x})$ of the LS minimization problem in (3.8). For each $\mathbf{x} \in \Omega \subset \mathbb{R}^d$ and given integers $\{n_l\}_{l=1}^L$, we denote the set of DNN functions by

$$\mathcal{M}(\boldsymbol{\theta}, L) = \Big\{\mathcal{N}(\mathbf{x}) = N^{(L)} \circ \cdots \circ N^{(1)}(\mathbf{x}) : \boldsymbol{\omega}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}},\ \mathbf{b}^{(l)} \in \mathbb{R}^{n_l} \text{ for } l = 1, ..., L\Big\},$$

where $N^{(l)}(\mathbf{x}^{(l-1)})$ is defined in (2.2) and $\boldsymbol{\theta}$ denotes all parameters in the network: $\boldsymbol{\omega}^{(l)}$ and $\mathbf{b}^{(l)}$ for $l = 1, ..., L$. It is easy to see that $\mathcal{M}(\boldsymbol{\theta}, L)$ is a subset of $V_{\boldsymbol{\beta}}$, but not a linear subspace. For each $\mathbf{x} \in \Omega \subset \mathbb{R}^d$, a DNN is implemented to compute an approximation $u_{_N}(\mathbf{x}; \boldsymbol{\theta})$ at the point $\mathbf{x}$. Then the LSNN method for the linear advection-reaction equations is to find $u_{_N}(\mathbf{x}; \boldsymbol{\theta}^*) \in \mathcal{M}(\boldsymbol{\theta}, L)$ such that

$$\mathcal{L}\Big(u_{_N}(\mathbf{x}; \boldsymbol{\theta}^*); \mathbf{f}\Big) = \min_{v \in \mathcal{M}(\boldsymbol{\theta}, L)} \mathcal{L}\Big(v(\mathbf{x}; \boldsymbol{\theta}); \mathbf{f}\Big) = \min_{\boldsymbol{\theta} \in \mathbb{R}^N} \mathcal{L}\Big(v(\mathbf{x}; \boldsymbol{\theta}); \mathbf{f}\Big), \tag{3.12}$$

where N is the total number of parameters in $\mathcal{M}(\boldsymbol{\theta}, L)$ which is given in (2.3).

Instead of evaluating the LS functional analytically, we consider numerical approximation to the LS functional. This means that we will use numerical quadrature to approximate integrals of the LS functional. For simplicity and generality in high dimensions, we will adopt composite "mid-point" quadrature rule. 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 $\mathcal{T}$ equal the whole domain Ω and that any two distinct subdomains of $\mathcal{T}$ have no intersection; more precisely,

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

In addition, we denote

$$\mathcal{E}_{-} = \{E = \partial K \cap \Gamma_{-} :\ K \in \mathcal{T}\}$$

as a partition of the inflow boundary Γ_-. Let $\mathbf{x}_{_K}$ and $\mathbf{x}_{_E}$ be the centroids of $K \in \mathcal{T}$ and $E \in \mathcal{E}_-$, respectively. Define the discrete LS functional as follows:

$$\mathcal{L}_{\mathcal{T}}\Big(v(\mathbf{x};\boldsymbol{\theta});\mathbf{f}\Big) = \sum_{K\in\mathcal{T}} \Big(v_{\boldsymbol{\beta}} + \hat{\gamma}\, v - f\Big)^2(\mathbf{x}_{_K};\boldsymbol{\theta})\,|K| + \sum_{E\in\mathcal{E}_-} \Big(|\boldsymbol{\beta}\cdot\boldsymbol{n}|(v-g)^2\Big)(\mathbf{x}_{_E};\boldsymbol{\theta})|E|, \tag{3.13}$$

where $|K|$ and $|E|$ are the d and $d-1$ dimensional measures of K and E, respectively; and $v_{\boldsymbol{\beta}}$ is the discrete directional derivative which will be defined in the subsequent sections. We shall see later that the discretization scheme is critical to ensure the success of the LSNN method. Then the discrete least-squares approximation is to find $u_{\mathcal{T}}^N(\mathbf{x},\boldsymbol{\theta}^*) \in \mathcal{M}(\boldsymbol{\theta}, L)$ such that

$$\mathcal{L}_{\mathcal{T}}\Big(u_{\mathcal{T}}^N(\mathbf{x},\boldsymbol{\theta}^*);\mathbf{f}\Big) = \min_{v\in\mathcal{M}(\boldsymbol{\theta},L)} \mathcal{L}_{\mathcal{T}}\Big(v(\mathbf{x};\boldsymbol{\theta});\,\mathbf{f}\Big) = \min_{\boldsymbol{\theta}\in\mathbb{R}^N} \mathcal{L}_{\mathcal{T}}\Big(v(\mathbf{x};\boldsymbol{\theta});\mathbf{f}\Big). \tag{3.14}$$

To understand approximation property of the LSNN method, by the triangle inequality, we have

$$\|u - u_{\mathcal{T}}^N\| \le \|u - u_{_N}\| + \|u_{_N} - u_{\mathcal{T}}^N\|, \tag{3.15}$$

where the first term represents the approximation error caused by the deep neural network and the second term is the numerical error by evaluating the LS functional through numerical quadrature. How to estimate the former is still an open problem. The latter can be computed to a desired accuracy through either a uniform or an adaptive partition of the Ω and Γ_- (See [10] for more details).

In (3.15), $u_{\mathcal{T}}^N(\mathbf{x},\boldsymbol{\theta})$ is assumed to be the exact solution of the minimization problem in (3.14). In practice, problem (3.14) is solved numerically by an iterative method such as the method of gradient decent. Let $u_{\mathcal{T}}^k(\mathbf{x},\boldsymbol{\theta})$ be the algebraic approximation at the k^{th} iterate, then the total error of the discrete LSNN method is bounded by the sum of the DNN approximation error, the quadrature error, and the algebraic error as follows:

$$\|u - u_{\mathcal{T}}^k\| \le \|u - u_{_N}\| + \|u_{_N} - u_{\mathcal{T}}^N\| + \|u_{\mathcal{T}}^N - u_{\mathcal{T}}^k\|, \tag{3.16}$$

which can be easily obtained by the triangle inequality.

3.3 ReLU NN Approximation of Discontinuous Solutions: I. Discontinuous Interface Along a Straight Line

In this section, we will theoretically show that a two-layer ReLU neural network is capable of accurately approximating a piece-wise defined solution with a constant jump on the discontinuous interface consisting of a straight line.

First, we consider the equation (3.4) with $\hat{\gamma} = 0$. When the boundary data g is discontinuous at point $\mathbf{x}_0 \in \Gamma_-$, the solution of the linear advection-reaction problem is discontinuous across an interface: the streamline of the advection velocity field starting at $\mathbf{x}_0$. Then the key to approximate such discontinuous solution is to decompose the solution as the sum of a piece-wise constant function as well as a continuous piece-wise smooth function (see, e.g., (3.19)).

Lemma 3.3.1. *Let u and $u_{_N}$ be the solutions of problems (3.7) and (3.12), respectively. Then we have*

$$|||u - u_{_N}|||_{\boldsymbol{\beta}} \leq \left(\frac{M}{\alpha}\right)^{1/2} \inf_{v \in \mathcal{M}(\boldsymbol{\theta}, L)} |||u - v|||_{\boldsymbol{\beta}}, \tag{3.17}$$

where α and M are constants in (3.9).

Proof. For any $v \in \mathcal{M}(\boldsymbol{\theta}, L) \subset V_{\boldsymbol{\beta}}$, it follows from the coercivity and continuity of the homogeneous functional $\mathcal{L}\big(v; \mathbf{0}\big)$ in (3.9), problem (3.2), and (3.12) that

$$\begin{aligned} \alpha\, |||u - u_{_N}|||_{\boldsymbol{\beta}}^2 &\leq \mathcal{L}\big(u - u_{_N}; \mathbf{0}\big) = \mathcal{L}\big(u_{_N}(\mathbf{x}; \boldsymbol{\theta}^*); \mathbf{f}\big) \\ &\leq \mathcal{L}\big(v(\mathbf{x}; \boldsymbol{\theta}); \mathbf{f}\big) = \mathcal{L}\big(u - v; \mathbf{0}\big) \leq M\, |||u - v|||_{\boldsymbol{\beta}}^2, \end{aligned}$$

which implies (3.17). This completes the proof of the lemma. □

For a given vector $\boldsymbol{\xi} \in \mathbb{R}^d$ and $c \in \mathbb{R}$, assume that the hyper-plane $\mathcal{P} : \boldsymbol{\xi} \cdot \mathbf{x} = c$ divides the domain Ω into two non-empty subdomains Ω_1 and Ω_2, i.e.,

$$\Omega_1 = \{\mathbf{x} \in \Omega : \boldsymbol{\xi} \cdot \mathbf{x} < c\} \quad \text{and} \quad \Omega_2 = \{\mathbf{x} \in \Omega : \boldsymbol{\xi} \cdot \mathbf{x} > c\}.$$

Let $\chi(\mathbf{x};\boldsymbol{\xi},c)$ be a piece-wise constant function defined by

$$\chi(\mathbf{x};\boldsymbol{\xi},c)=\begin{cases}\alpha_1, & \mathbf{x}\in\Omega_1,\\ \\ \alpha_2, & \mathbf{x}\in\Omega_2.\end{cases}$$

Lemma 3.3.2. *Let $p(\mathbf{x})$ be a two-layer neural network function given by*

$$p(\mathbf{x})=\alpha_1+\frac{\alpha_2-\alpha_1}{2\varepsilon}\Big(\sigma(\boldsymbol{\xi}\cdot\mathbf{x}-c+\varepsilon)-\sigma(\boldsymbol{\xi}\cdot\mathbf{x}-c-\varepsilon)\Big)$$

for any $\varepsilon>0$ such that intersections between the domain Ω and the hyper-planes $\boldsymbol{\xi}\cdot\mathbf{x}=c\pm\varepsilon$ are not empty. Then we have

$$\|\chi-p\|_{0,\Omega}=\Big(\|\chi-p\|_{0,\Omega}^2+\|\chi_{\boldsymbol{\eta}}-p_{\boldsymbol{\eta}}\|_{0,\Omega}^2\Big)^{1/2}\leq\sqrt{\frac{D}{6}}\,\big|\alpha_1-\alpha_2\big|\,\sqrt{\varepsilon},\tag{3.18}$$

where $\boldsymbol{\eta}$ is a vector normal to $\boldsymbol{\xi}$ and D is the diameter of the domain Ω.

Proof. Let

$$\Omega_\varepsilon=\Omega_{\varepsilon,1}\cup\Omega_{\varepsilon,2}\equiv\{\mathbf{x}\in\Omega:\ c-\varepsilon<\boldsymbol{\xi}\cdot\mathbf{x}<c\}\cup\{\mathbf{x}\in\Omega:\ c<\boldsymbol{\xi}\cdot\mathbf{x}<c+\varepsilon\}.$$

The equality in (3.18) follows from the fact that $\chi_{\boldsymbol{\eta}}-p_{\boldsymbol{\eta}}=0$. To show the validity of the inequality in (3.18), first we have

$$\chi-p=\begin{cases}\dfrac{\alpha_1-\alpha_2}{2\varepsilon}\Big(\boldsymbol{\xi}\cdot\mathbf{x}-c+\varepsilon\Big), & \mathbf{x}\in\Omega_{\varepsilon,1},\\ \dfrac{\alpha_1-\alpha_2}{2\varepsilon}\Big(\boldsymbol{\xi}\cdot\mathbf{x}-c-\varepsilon\Big), & \mathbf{x}\in\Omega_{\varepsilon,2},\\ \qquad 0, & \mathbf{x}\in\Omega\setminus\Omega_\varepsilon.\end{cases}$$

By a rotation of the coordinates, $\mathbf{x}=(s,\mathbf{y})$, it is easy to see that the domain $\Omega_{\varepsilon,1}$ is bounded by the hyper-planes $s=c-\varepsilon$ and $s=c$ and the hyper-surfaces $\varphi_1(s)$ and $\varphi_2(s)$ on $\partial\Omega$. Hence, we have

$$\int_{\Omega_{\varepsilon,1}}\Big(\boldsymbol{\xi}\cdot\mathbf{x}-c+\varepsilon\Big)^2\,d\mathbf{x}=\int_{c-\varepsilon}^{c}\int_{\varphi_1(s)}^{\varphi_2(s)}\Big(s-c+\varepsilon\Big)^2\,d\mathbf{y}\,ds\leq D^{d-1}\int_{c-\varepsilon}^{c}\Big(s-c+\varepsilon\Big)^2\,ds=\frac{D^{d-1}}{3}\,\varepsilon^3.$$

In a similar fashion,

$$\int_{\Omega_{\varepsilon,2}} \left(\boldsymbol{\xi}\cdot\mathbf{x}-c-\varepsilon\right)^2 d\mathbf{x} \le \frac{D^{d-1}}{3}\varepsilon^3.$$

The above two inequalities imply

$$\|\chi-p\|_{0,\Omega}^2 \le \left(\frac{\alpha_1-\alpha_2}{2\varepsilon}\right)^2 \frac{2D^{d-1}}{3}\varepsilon^3 = \frac{D^{d-1}(\alpha_1-\alpha_2)^2\varepsilon}{6}.$$

This proves the inequality in (3.18) and, hence, the lemma. □

Assume that u is a piece-wise smooth function with respect to the partition $\{\Omega_1,\Omega_2\}$ such that the jump of u on the interface $\mathcal{P}=\Omega_1\cap\Omega_2$ is a constant $\alpha_2-\alpha_1$, i.e.,

$$[\![u]\!]_{\mathcal{P}} \equiv u_2|_{\mathcal{P}} - u_1|_{\mathcal{P}} = \alpha_2-\alpha_1.$$

Then u has the following decomposition

$$u = \chi(\mathbf{x};\boldsymbol{\xi},c)+\hat{u}, \tag{3.19}$$

where $\boldsymbol{\xi}$ is a vector normal to $\boldsymbol{\beta}$. It is easy to see that $\hat{u}$ is continuous in Ω and piece-wise smooth. By decomposing the discontinuous function into the sum of a piece-wise constant function and a continuous piece-wise smooth function, we are able to prove the following Theorem.

Theorem 3.3.1. *Assume that the advection velocity field $\boldsymbol{\beta}$ is a constant vector field and that $f\in C(\Omega)$. Let u and $u_{_N}$ be the solutions of problems (3.7) and (3.12), respectively. Then we have*

$$|||u-u_{_N}|||_{\boldsymbol{\beta}} \le C\left(\left|\alpha_1-\alpha_2\right|\sqrt{\varepsilon}+\inf_{v\in\mathcal{M}(\boldsymbol{\theta},L)}|||\hat{u}-v|||_{\boldsymbol{\beta}}\right), \tag{3.20}$$

where $\hat{u}\in C(\Omega)$ is given in (3.19).

Proof. The assumptions on $\boldsymbol{\beta}$ and f imply that the exact solution u has the decomposition in (3.19). Now, (3.20) is a direct consequence of Lemmas 3.3.1 and 3.3.2. □

Lemma 3.3.3. *Let u, $u_{\scriptscriptstyle N}$, and $u_{\mathcal{T}}^N$ be the solutions of problems (3.7), (3.12), and (3.13), respectively. Then there exists a positive constant C such that*

$$|||u-u_{\scriptscriptstyle N}|||_{\boldsymbol{\beta}} \le C\left(\inf_{v\in\mathcal{M}(\boldsymbol{\theta},L)}|||u-v|||_{\boldsymbol{\beta}} + \left|(\mathcal{L}-\mathcal{L}_{\mathcal{T}})(u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N;\mathbf{0})\right| + \left|(\mathcal{L}-\mathcal{L}_{\mathcal{T}})(u-u_{\scriptscriptstyle N};\mathbf{0})\right|\right). \tag{3.21}$$

Proof. By the triangle inequality, the fact that $\mathcal{L}_{\mathcal{T}}(u_{\mathcal{T}}^N;\mathbf{f}) \le \mathcal{L}_{\mathcal{T}}(u_{\scriptscriptstyle N};\mathbf{f})$, and the continuity of the homogeneous functional $\mathcal{L}\big(v;\,\mathbf{0}\big)$ in (3.9), we have

$$\begin{aligned}
\frac{1}{2}\,\mathcal{L}_{\mathcal{T}}(u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N;\mathbf{0}) &\le \mathcal{L}_{\mathcal{T}}(u_{\scriptscriptstyle N}-u;\mathbf{0}) + \mathcal{L}_{\mathcal{T}}(u-u_{\mathcal{T}}^N;\mathbf{0}) = \mathcal{L}_{\mathcal{T}}(u_{\scriptscriptstyle N};\mathbf{f}) + \mathcal{L}_{\mathcal{T}}(u_{\mathcal{T}}^N;\mathbf{f}) \\
&\le 2\,\mathcal{L}_{\mathcal{T}}(u_{\scriptscriptstyle N};\mathbf{f}) = 2\left((\mathcal{L}_{\mathcal{T}}-\mathcal{L})(u_{\scriptscriptstyle N}-u;\mathbf{0}) + \mathcal{L}(u_{\scriptscriptstyle N}-u;\mathbf{0})\right) \\
&\le 2\,(\mathcal{L}_{\mathcal{T}}-\mathcal{L})(u_{\scriptscriptstyle N}-u;\mathbf{0}) + 2M\,|||u-u_{\scriptscriptstyle N}|||_{\boldsymbol{\beta}}^2,
\end{aligned}$$

which, together with the coercivity of the homogeneous functional $\mathcal{L}\big(v;\,\mathbf{0}\big)$ in (3.9), implies that

$$\begin{aligned}
\alpha\left|\!\left|\!\left|u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N\right|\!\right|\!\right|_{\boldsymbol{\beta}}^2 &\le \mathcal{L}\big(u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N;\,\mathbf{0}\big) = \big(\mathcal{L}-\mathcal{L}_{\mathcal{T}}\big)\big(u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N;\,\mathbf{0}\big) + \mathcal{L}_{\mathcal{T}}\big(u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N;\,\mathbf{0}\big) \\
&\le \big(\mathcal{L}-\mathcal{L}_{\mathcal{T}}\big)\big(u_{\scriptscriptstyle N}-u_{\mathcal{T}}^N;\,\mathbf{0}\big) + 4\,(\mathcal{L}_{\mathcal{T}}-\mathcal{L})(u_{\scriptscriptstyle N}-u;\mathbf{0}) + 4M\,|||u-u_{\scriptscriptstyle N}|||_{\boldsymbol{\beta}}^2.
\end{aligned}$$

Now, (3.21) is a direct consequence of the triangle inequality, the above inequality, and Lemma 3.3.1. This completes the proof of the lemma. □

Lemma 3.3.3 indicates that the total error of the LSNN approximation with numerical integration is bounded by the approximation error of the neural network and the error of the numerical integration.

3.4 Initialization of two-layer neural networks

Network initialization is important in the training, inappropriate initialization may cause high computational cost as well as poor results. In this section, we briefly describe the initialization strategy introduced in [10] for two-layer neural networks.

In d-dimension, for i = 1, 2, ..., n_1, let $\boldsymbol{\omega}_{\rm i} \in \mathbb{R}^d$ and $b_{\rm i} \in \mathbb{R}$ be the weights and bias of the input layer, respectively; and let $c_{\rm i} \in \mathbb{R}$ and $c_0 \in \mathbb{R}$ be the respective weights and bias of the output layer. Then a two-layer ReLU neural network with n_1 neurons produces the following set of functions for $\mathbf{x} \in \mathbb{R}^d$:

$$\mathcal{M}(\boldsymbol{\theta}, 1) = \left\{ c_0 + \sum_{\rm i=1}^{n_1} c_{\rm i}\sigma(\boldsymbol{\omega}_{\rm i} \cdot \mathbf{x} - b_{\rm i}) \,:\, c_{\rm i},\, b_{\rm i} \in \mathbb{R},\ \boldsymbol{\omega}_{\rm i} \in \mathbb{R}^d \right\},$$

where $\boldsymbol{\theta}$ represents all parameters in the network. The $\sigma(t)$ is a continuous piece-wise linear function with the discontinuity at $t = 0$ and belongs to a class of activation functions of the form

$$\sigma_k(t) = \max\{0,\, t^k\} = \begin{cases} 0, & t < 0, \\ & \qquad \text{for} \quad k \in \mathbb{Z}_+, \\ t^k, & t \geq 0 \end{cases}$$

where $\mathbb{Z}_+$ is the set of all positive integers. Note that $\sigma_k(t) \in C^{k-1}(\mathbb{R})$ is a piece-wise polynomial of degree k with the discontinuity at $t = 0$. For simplicity of presentation, we only discuss the initialization strategy for two-layer ReLU neural networks. Extension of results to general activation functions $\sigma_k(t)$ may be foreseen.

In total, there are $(d + 2)n_1 + 1$ parameters for functions in the set $\mathcal{M}(\boldsymbol{\theta}, 1)$, where $n_1 + 1$ of them are the output weights and bias $\{c_{\rm i}\}_{\rm i=0}^{n_1}$; and $(d+1)n_1$ of them are the input weights $\{\boldsymbol{\omega}_{\rm i}\}_{\rm i=1}^{n_1}$ and bias $\{b_{\rm i}\}_{\rm i=1}^{n_1}$. We refer to the former as linear parameters and the later as nonlinear parameters. Hence, $\mathcal{M}(\boldsymbol{\theta}, 1)$ consists of $n_1 + 1$ linear and $(d + 1)n_1$ nonlinear parameters. It remains a challenge to determine the values of nonlinear parameters in the training process. Then to remove n_1 nonlinear parameters, we notice that

$$\sigma(\boldsymbol{\omega} \cdot \mathbf{x} - b) = |\boldsymbol{\omega}|\, \sigma\left(\frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} \cdot \mathbf{x} - \frac{b}{|\boldsymbol{\omega}|} \right),$$

where $|\boldsymbol{\omega}| = \sqrt{\omega_1^2 + \cdots + \omega_d^2}$ is the length of a vector $\boldsymbol{\omega} \in \mathbb{R}^d$. This implies that $\mathcal{M}(\boldsymbol{\theta}, 1)$ is equal to

$$\hat{\mathcal{M}}(\boldsymbol{\theta}, 1) = \left\{ c_0 + \sum_{\rm i=1}^{n_1} c_{\rm i}\sigma(\boldsymbol{\omega}_{\rm i} \cdot \mathbf{x} - b_{\rm i}) \,:\, c_{\rm i},\, b_{\rm i} \in \mathbb{R},\ \boldsymbol{\omega}_{\rm i} \in \mathcal{S}^{d-1} \right\}, \tag{3.22}$$

where $\mathcal{S}^{d-1}$ is the unit sphere in $\mathbb{R}^d$. The number of parameters in $\hat{\mathcal{M}}(\boldsymbol{\theta}, 1)$ is reduced to

$$N = (d+1)n_1 + 1.$$

First, we restrict our attention to the two dimensional case. When $d = 2$, $\mathcal{S}^1$ is a unit circle given by:

$$\mathcal{S}^1 = \Big\{\boldsymbol{\omega} = (\omega_1,\, \omega_2)^T \in \mathbb{R}^2 \,:\, \omega_1^2 + \omega_2^2 = 1\Big\} = \Big\{\boldsymbol{\omega} = \Big(\cos\gamma,\, \sin\gamma\Big)^T \,:\, 0 \le \gamma \le 2\boldsymbol{\pi}\Big\}.$$

This gives

$$\hat{\mathcal{M}}(\boldsymbol{\theta}, 1) = \left\{c_0 + \sum_{\mathrm{i}=1}^{n_1} c_{\mathrm{i}}\sigma\Big((\cos\gamma_{\mathrm{i}})\, x_1 + (\sin\gamma_{\mathrm{i}})\, x_2 - b_{\mathrm{i}}\Big) \,:\, c_{\mathrm{i}},\, b_{\mathrm{i}} \in \mathbb{R},\ \gamma_{\mathrm{i}} \in [0,\, 2\boldsymbol{\pi}]\right\}, \tag{3.23}$$

which is the set of continuous piece-wise linear functions with n_1 free lines

$$l_{\mathrm{i}} :\ (\cos\gamma_{\mathrm{i}})\, x_1 + (\sin\gamma_{\mathrm{i}})\, x_2 - b_{\mathrm{i}} = 0 \quad \text{for i} = 1,\, ...,\, n_1. \tag{3.24}$$

Similarly, in the d-dimension, $\hat{\mathcal{M}}(\boldsymbol{\theta}, 1)$ is the set of continuous piece-wise linear functions with n_1 free hyper-planes

$$\mathcal{P}_{\mathrm{i}} :\ \boldsymbol{\omega}_{\mathrm{i}} \cdot \mathbf{x} - b_{\mathrm{i}} = 0 \quad \text{for i} = 1,\, ...,\, n_1. \tag{3.25}$$

Let

$$\varphi_0(\mathbf{x}) = 1 \quad \text{and} \quad \varphi_{\mathrm{i}}(\mathbf{x}) = \sigma(\boldsymbol{\omega}_{\mathrm{i}} \cdot \mathbf{x} - b_{\mathrm{i}}) \quad \text{for i} = 1, ..., n_1.$$

For a given input weights and bias

$$\boldsymbol{\omega} = (\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_{n_1}) \quad \text{and} \quad \mathbf{b} = (b_1, ..., b_{n_1}),$$

problem (3.11) may be approximated by finding $u_{n_1} = \sum_{\mathrm{i}=0}^{n_1} c_{\mathrm{i}}\varphi_{\mathrm{i}}(\mathbf{x})$ such that

$$a(u_{n_1}, \varphi_{\mathrm{j}}) = f(\varphi_{\mathrm{j}}) \tag{3.26}$$

for $\mathrm{j} = 0, 1, ..., n_1$. Let

$$A(\boldsymbol{\omega}, \mathbf{b}) = (a(\varphi_\mathrm{j}, \varphi_\mathrm{i}))_{(n_1+1)\times(n_1+1)} \quad \text{and} \quad F(\boldsymbol{\omega}, \mathbf{b}) = (f(\varphi_\mathrm{j}))_{(n_1+1)\times 1},$$

then the coefficients, $\mathbf{c} = (c_0, c_1, ..., c_n)$, of u_{n_1} is the solution of the system of linear algebraic equations

$$A(\boldsymbol{\omega}, \mathbf{b})\, \mathbf{c} = F(\boldsymbol{\omega}, \mathbf{b}). \tag{3.27}$$

Lemma 3.4.1. *Assume that hyper-planes* $\{\boldsymbol{\omega}_\mathrm{i} \cdot \mathbf{x} = b_\mathrm{i}\}_{\mathrm{i}=1}^{n_1}$ *are distinct. Then the coefficient matrix* $A(\boldsymbol{\omega}, \mathbf{b})$ *is symmetric, positive definite.*

Proof. Obviously, $A(\boldsymbol{\omega}, \mathbf{b})$ is symmetric. Positive definiteness of $A(\boldsymbol{\omega}, \mathbf{b})$ follows from the lower bound in (3.9) and the linear independence of $\{\varphi_\mathrm{i}\}_{\mathrm{i}=0}^{n_1}$ (see Lemma 2.1 of [10]). □

As discussed in [10], the hyper-planes (3.25) and the boundary of the domain Ω form a physical partition of the domain Ω. It is then natural to initialize the input weights $\boldsymbol{\omega}$ and bias $\mathbf{b}$ such that the corresponding hyper-planes $\{\mathcal{P}_\mathrm{i}\}_{\mathrm{i}=1}^{n_1}$ form a uniform partition of the domain Ω. The initial for the output weights and bias $\mathbf{c}$ may be chosen to be the solution of problem (3.27). To conclude, initialization of nonlinear parameters are based on their physical partitioning of the domain and the initial of linear parameters are obtained by solving a system of linear equations given the nonlinear parameters.

3.5 Numerical Experiments

In this section, we present numerical results for test problems with constant, piece-wise constant, or variable advection velocity fields. The solutions of these test problems are discontinuous along an interface which is a line segment, a piece-wise line segment, or a curve.

In all experiments, the integration mesh $\mathcal{T}$ is obtained by uniformly partitioning the domain Ω into identical squares with mesh size $h = 10^{-2}$. To train (numerically compute) parameters $\boldsymbol{\theta}$ associated with the DNN functions $u_{\mathcal{T}}^N(\mathbf{x}, \boldsymbol{\theta})$, the Adam optimizer version of gradient descent [39] is implemented as an iterative method to numerically solve the min-

imization problem in (3.13). The iterative parameter (may vary at each iteration) of the method of gradient decent is called the step size or learning rate.

Choosing a proper approach to approximate the differential operator in the least-squares functional (3.13) is crucial for success of the LSNN method. In general, there are two numerical approaches. The first approach is to use the backward finite difference quotient along coordinate directions. Let $\mathbf{x}_{K} = (x_{K}^{1}, x_{K}^{2})$, then the differential operator in (3.13) is approximated by

$$\begin{aligned} v_{\boldsymbol{\beta}}(x_{K}^{1}, x_{K}^{2}) &\equiv (\beta_1, \beta_2) \cdot \nabla v(x_{K}^{1}, x_{K}^{2}) \\ &\approx \beta_1 \frac{v(x_{K}^{1}, x_{K}^{2}) - v(x_{K}^{1} - h_1, x_{K}^{2})}{h_1} + \beta_2 \frac{v(x_{K}^{1}, x_{K}^{2}) - v(x_{K}^{1}, x_{K}^{2} - h_2)}{h_2} \end{aligned} \tag{3.28}$$

with $h_1 \in \mathbb{R}$ and $h_2 \in \mathbb{R}$ being the finite difference step size along x_1 and x_2 directions, receptively. The second method is to employ the backward finite difference quotient along the $\boldsymbol{\beta}$ direction:

$$v_{\boldsymbol{\beta}}(\mathbf{x}_{K}) \approx \frac{v(\mathbf{x}_{K}) - v\Big(\mathbf{x}_{K} - \rho\bar{\boldsymbol{\beta}}(\mathbf{x}_{K})\Big)}{\rho} \tag{3.29}$$

where $\rho \in \mathbb{R}$ is chosen to be smaller than the integration mesh size h, and $\bar{\boldsymbol{\beta}}$ is the unit vector along the $\boldsymbol{\beta}$ direction. In section 3.5.2, we will present the numerical results for both schemes.

Let u be the exact solution of problem (3.2) and $\bar{u}_{\tau}^{N}$ be the LSNN approximation. Tables 3.1–3.7 report the numerical errors in the relative L^2, $V_{\boldsymbol{\beta}}$, and graph norms. In these tables, a network structure is expressed by 2-n-1 for a two-layer network with n neurons, by 2-n_1-n_2-1 for a three-layer network with n_1 and n_2 neurons in the respective first and second layers, and so on. Figures 3.2–3.9 depict the traces of the exact solution and the numerical approximation on a plane perpendicular to both the x_1x_2-plane and the discontinuous interface, which accurately illustrate the quality of the numerical approximation.

3.5.1 Problems with a constant advection velocity fields: I. Discontinuity along a vertical line segment

First, we present numerical results for two test problems with constant advection velocity fields whose solutions are piece-wise constants (see, e.g., [22]). A two-layer neural network is employed and the network is initialized by the method described in section 3.4.

The first test problem is the equation in (3.2) with the domain $\Omega = (0,2) \times (0,1)$, the inflow boundary $\Gamma_- = \{(x,0) : x \in (0,2)\}$, a constant advection velocity field $\boldsymbol{\beta} = (0,1)^T$, $\gamma = f = 0$, and the inflow boundary data $g(x) = 0$ for $x \in (0, \pi/3)$ and $g(x) = 1$ for $x \in (\pi/3, 2)$. Let $\Omega_1 = \{(x,y) \in \Omega : 0 < x < \pi/3\}$ and $\Omega_2 = \{(x,y) \in \Omega : \pi/3 < x < 2\}$, it is then easy to see that the exact solution is a piece-wise constant given by

$$u(x,y) = \begin{cases} 0, & (x,y) \in \Omega_1, \\ 1, & (x,y) \in \Omega_2. \end{cases}$$

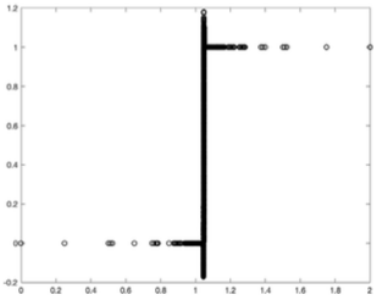
The discontinuous interface is the vertical line $x = \pi/3$.

This problem was used to test various adaptive least-squares finite element methods in [22]. In particular, the discontinuous interface $x = \pi/3$ was chosen so that if the initial mesh does not align with the interface, so is the mesh generated by either global or local mesh refinements.

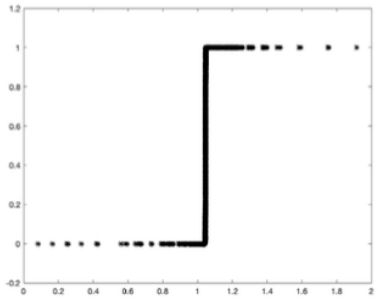
Numerical results in [22] (see Fig. 3.1) showed that the conforming least-squares finite element method (C-LSFEM) exhibits the Gibbs phenomena even with very fine mesh and that the newly developed flux-based LSFEM in [22] using a pair of the lowest-order elements is able to avoid overshooting on an adaptively refined mesh.

The LSNN method using the directional derivative is implemented with $\rho = h/2$ in (3.29) and a fixed learning rate 0.003 with 50000 iterations. Our first set of experiments are done by using networks: 2-200-1 and 2-25-15-15-1. These two networks have 601 and 705 parameters, respectively, and provide good approximations (similar to Fig. 3.2(a,b)) to the exact solution.

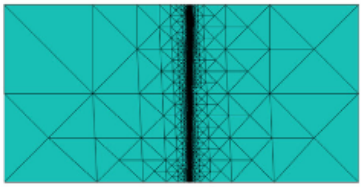
Lemma 3.3.2 indicates that a two-layer network with 2 neurons is sufficient to approximate the exact solution accurately. Our second set of experiments are done by using net-



(a) C-LSFEM vertical cross section



(b) flux-based LSFEM vertical cross section



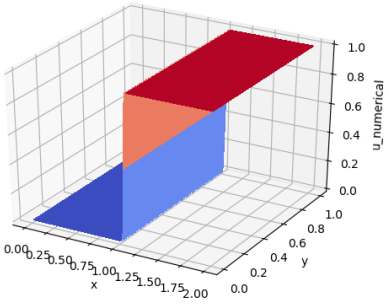
(c) An adaptively refined mesh

Figure 3.1. Numerical results in [22] of the problem with discontinuity along a vertical line segment

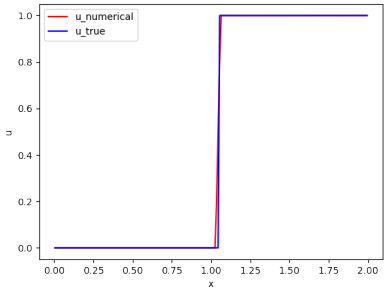
works: 2-2-1 and 2-4-1 with the respective 7 and 13 parameters. The 2-2-1 network fails to approximate the exact solution when the initial breaking lines are chosen to be the vertical line $x = 1$ and the horizontal line $y = 1/2$. This is because the iterative solver of the non-convex optimization is not capable of moving the initial horizontal breaking line to the right place. The initial breaking lines for the 2-4-1 network are chosen to be the vertical lines $x = 2/3$ and $x = 4/3$ and the horizontal lines $y = 1/3$ and $y = 2/3$.

Table 3.1. Relative errors of the problem with discontinuity along a vertical line segment

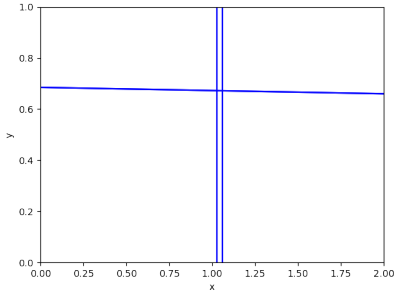
Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_\beta}{\|\|\|u\|\|\|_\beta}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-4-1	0.058046	0.058304	0.050491	13
2-200-1	0.058745	0.058926	0.048537	601



(a) Numerical solution $\bar{u}_{\mathcal{T}}^N$



(b) Vertical cross section



(c) Network breaking lines

Figure 3.2. Approximation results of the problem with discontinuity along a vertical line segment

Errors of numerical results are presented in Table 3.1. The second and third columns in Table 3.1 show that the approximation of the small network is slightly more accurate than

that of the large network while the values of the loss functions are reversed. This indicates that the large network is trapped in a local minimum. The numerical solution of the 4-neuron network is depicted in Fig. 3.2(a). The traces of the exact and numerical solutions on the plane $y = 1$ are depicted in Fig. 3.2(b), which shows no oscillation. Fig. 3.2(c) displays breaking lines of the network with two vertical lines $x = 1.02882$ and $x = 1.06114$ closing to the interface $x = \pi/3$. This indicates that breaking lines of neural network are capable of automatically adapting to the discontinuous interface. This simple test problem shows that the LSNN method out-performs the traditional mesh-based numerical methods.

3.5.2 Problems with a constant advection velocity fields: II. Discontinuity along the diagonal

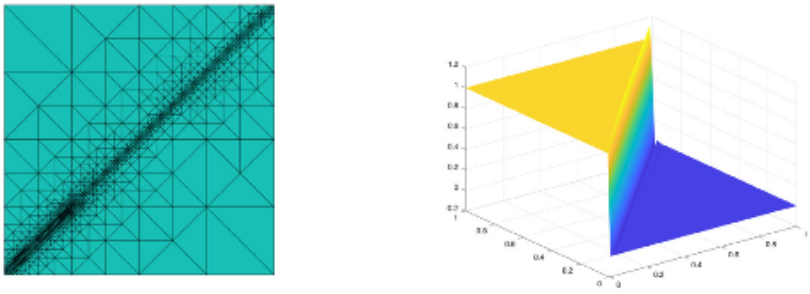
The second test problem is again equation (3.2) with a constant advection velocity vector and a piece-wise constant inflow boundary condition. Specifically, $\boldsymbol{\beta} = (1,1)^T/\sqrt{2}$, $\Omega = (-1,1)^2$, $\Gamma_- = \Gamma_-^1 \cup \Gamma_-^2 \equiv \{(-1,y) : y \in (-1,1)\} \cup \{(x,-1) : x \in (-1,1)\}$, $\gamma = 1$, g and f are piece-wise constants given by

$$g(x,y) = \begin{cases} 1, & (x,y) \in \Gamma_-^1, \\ 0, & (x,y) \in \Gamma_-^2, \end{cases} \quad \text{and} \quad f(x,y) = \begin{cases} 1, & (x,y) \in \Omega_1, \\ 0, & (x,y) \in \Omega_2, \end{cases}$$

where $\Omega_1 = \{(x,y) \in \Omega : y > x\}$ and $\Omega_2 = \{(x,y) \in \Omega : y < x\}$. The exact solution of the test problem is $u(x,y) = f(x,y)$ with the discontinuous interface: $y = x$.

Again, we presented the numerical results produced by traditional mesh-based method. Fig. 3.3 shows the results generated by the conforming least-squares finite element method (C-LSFEM) in [22], which indicates that the C-LSFEM exhibits Gibbs phenomena along the discontinuous interface even though a fine mesh is used.

For the LSNN method, we present two sets of numerical experiments. The first set is done by using the directional derivative (3.29) with $\rho = h/2$. Numerical results of a 2-6-1 network are presented in Table 3.2 and Figure 3.4. The traces of the exact and numerical solutions on the plane $y = -x$ are depicted in Fig. 3.4(b). Clearly, the LSNN method with only 19 parameters approximates the exact solution accurately without the Gibbs phenomena. This



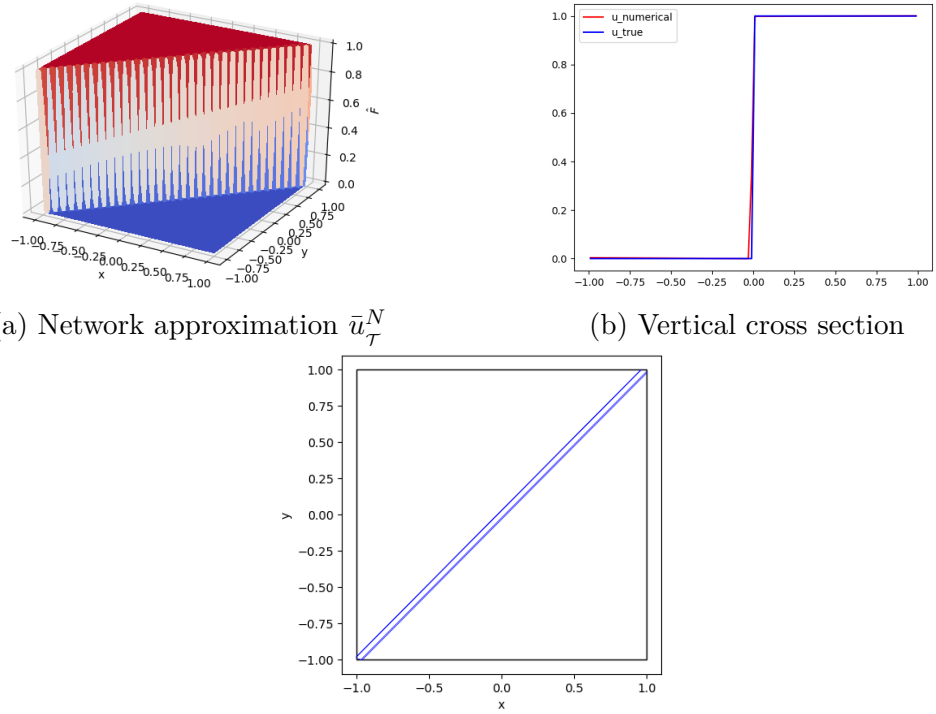
(a) Adaptive mesh (b) Numerical solution

Figure 3.3. Numerical results in [22] of the problem with discontinuity along the diagonal

test problem shows that the LSNN method is able to rotate and shift the initial breaking lines to approximate the discontinuous interface. Again, the LSNN method outperforms the traditional mesh-based approach.

Table 3.2. Relative errors of the problem with discontinuity along the diagonal using directional derivative

Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_\beta}{\|\|\|u\|\|\|_\beta}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-6-1	0.073534	0.073826	0.067531	19



(a) Network approximation $\bar{u}_{\mathcal{T}}^N$

(b) Vertical cross section

(c) Network breaking lines

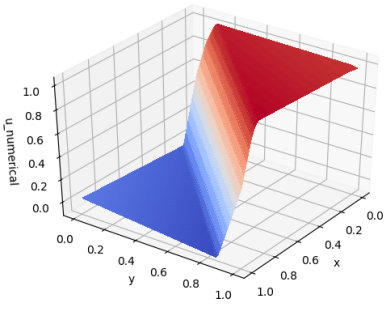
Figure 3.4. Approximation results of the problem with discontinuity along the diagonal using directional derivative

Comparably, the second set of experiment is done by approximating the differential operator in the least-squares functional (3.13) by the finite difference quotient along each coordinate direction (3.28). With $h_1 = h_2 = h/2$, we report the results for a 2-6-1 network in Table 3.3 and Figure 3.5. The results indicate that the LSNN method fails to resolve the

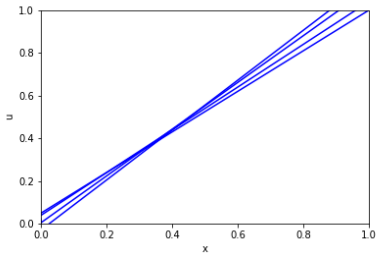
discontinuous interface. From this perspective, we see that the discretization approach is critical for the success of the LSNN method.

Table 3.3. Relative errors of the problem with discontinuity along the diagonal using finite difference quotient

Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_\beta}{\|\|\|u\|\|\|_\beta}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-6-1	0.192157	0.210532	0.120123	19



(a) Network approximation $\bar{u}_{\mathcal{T}}^N$



(b) Network breaking lines

Figure 3.5. Approximation results of the problem with discontinuity along the diagonal using finite difference quotient

3.5.3 Problems with a a piecewise smooth solution: I. Constant jump on the interface

The third test problem is a modification of the second test problem by changing the inflow boundary condition from the piece-wise constant to a discontinuous piece-wise smooth function and the domain from $\Omega = (-1,1)^2$ to $\Omega = (0,1)^2$, i.e.,

$$g(x,y) = \begin{cases} \sin(y), & (x,y) \in \Gamma_-^1 = \{(0,y) : y \in (0,1)\}, \\ \cos(x), & (x,y) \in \Gamma_-^2 = \{(x,0) : x \in (0,1)\}. \end{cases}$$

Set $\gamma = f = 0$, the exact solution of this test problem is

$$u(x,y) = \begin{cases} \sin(y-x), & (x,y) \in \Omega_1 = \{(x,y) \in (0,1)^2 : y > x\}, \\ \cos(x-y), & (x,y) \in \Omega_2 = \{(x,y) \in (0,1)^2 : y < x\}. \end{cases}$$

The LSNN method is employed with $\rho = h/2$ and a fixed learning rate 0.003 for 30000 iterations. Numerical results of three network models are reported in Table 3.4. Figure 3.6 presents the NN approximation of the 2-40-1 network. The traces of the exact and numerical solutions on the plane $y = 1 - x$ are depicted in Fig. 3.6(b), which exhibits no oscillation. It is expected that the network with additional neurons is needed in order to approximate the solution well since the solution of the test problem is a piece-wise smooth function. Moreover, this test problem conforms Theorem 3.3.1 that a piece-wise smooth function having a constant jump on a line segment discontinuous interface may be approximated well by a two-layer network.

Table 3.4. Relative errors of the problem with a piece-wise smooth solution

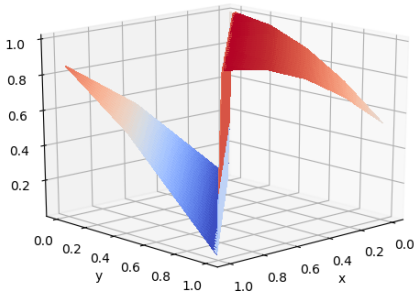
Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_\beta}{\|\|\|u\|\|\|_\beta}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-20-1	0.110745	0.110754	0.035571	61
2-30-1	0.107525	0.107641	0.013568	91
2-40-1	0.101411	0.101413	0.003509	121

3.5.4 Problems with a a piecewise smooth solution: II. Non-constant jump on the interface

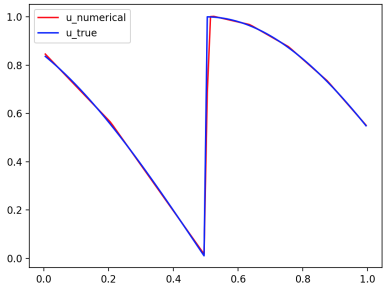
The fourth test problem is a modification of the third by changing γ from 0 to 1. Choose g and f accordingly such that the exact solution u is

$$u(x,y) = \begin{cases} \sin(x+y), & (x,y) \in \Omega_1 = \{(x,y) \in (0,1)^2 : y > x\}, \\ \cos(x+y), & (x,y) \in \Omega_2 = \{(x,y) \in (0,1)^2 : y < x\}. \end{cases}$$

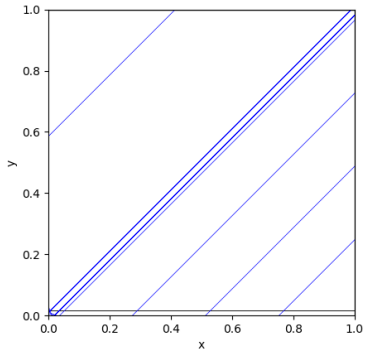
The discontinuous interface of the problem is along the diagonal $y = x$. Clearly, the jump on the interface is not a constant value.



(a) Network approximation $\bar{u}_{\mathcal{T}}^{N}$



(b) Vertical cross section



(c) Network breaking lines

Figure 3.6. Approximation results of the problem with a piece-wise smooth solution having a constant jump on the interface

The LSNN method uses $\rho = h/2$ in (3.29) and a fixed learning rate 0.003. Three neural network structures are tested for the problem: 2-40-1, 2-200-1 and 2-5-5-1. The initialization of the first layer employs the method described in the section 3.4, and that of the subsequent layers are randomly generated. The numerical results are presented in Table 3.5. The traces of the exact and numerical solutions on the plane $y = 1 - x$ are depicted in Fig. 3.7. Obviously, even though the discontinuous interface is a straight line, a two-layer network fails to approximate the solution which has a non-constant jump on the interface. Although the breaking lines capture the discontinuous interface, certain level of oscillation is exhibited which is not acceptable for this type of application. Hence, a three-layer network is needed to accurately approximate the solution.

Table 3.5. Relative errors of problem with a piece-wise smooth solution having a non-constant jump

Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_{\boldsymbol{\beta}}}{\|\|\|u\|\|\|_{\boldsymbol{\beta}}}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-50-1	0.046258	0.091179	0.078722	151
2-200-1	0.045368	0.083268	0.072652	601
2-15-15-1	0.003964	0.045277	0.035135	286

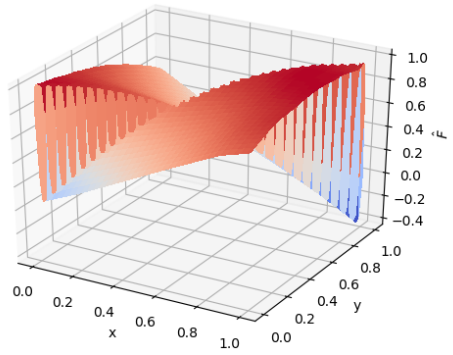
3.5.5 Problem with a piece-wise constant advection velocity field

The fifth test problem is equation (3.2) defined on $\Omega = (0,1)^2$ with $\gamma = f = 0$ and a piece-wise constant advection velocity field. Specifically, the advection velocity field is given by

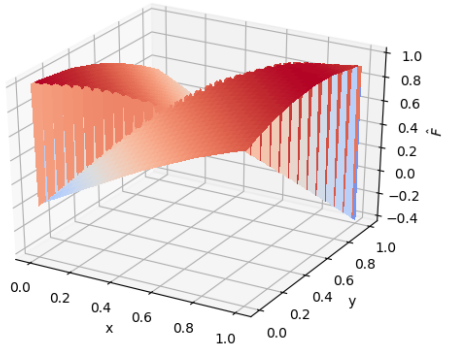
$$\boldsymbol{\beta} = \begin{cases} (1-\sqrt{2},1)^T, & (x,y) \in \Upsilon_1 = \{(x,y) \in \Omega : \, y < x\}, \\ (-1,\sqrt{2}-1)^T, & (x,y) \in \Upsilon_2 = \{(x,y) \in \Omega : \, y \geq x\}. \end{cases} \tag{3.30}$$

and, hence, the inflow boundary of the problem is

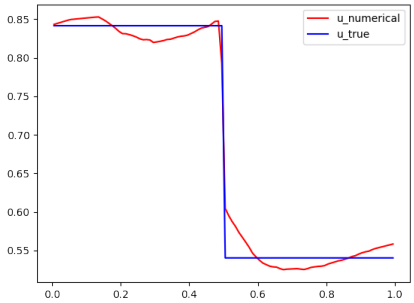
$$\Gamma_- = \{(x,0) : \, x \in (0,1)\} \cup \{(1,0)\} \cup \{(1,y) : \, y \in (0,1)\}. \tag{3.31}$$



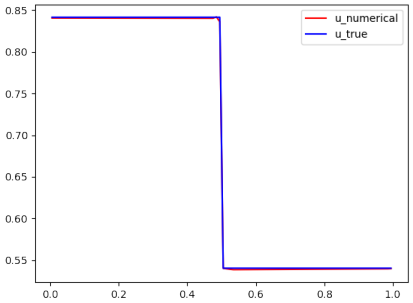
(a) 2-layer network approximation $\bar{u}_{\mathcal{T}}^{N}$



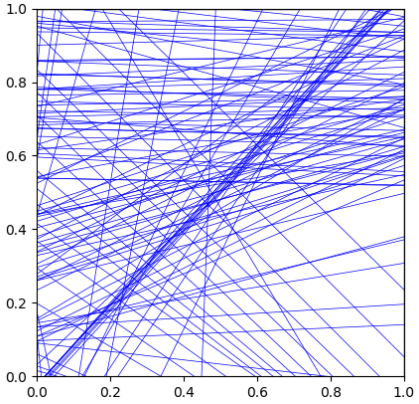
(b) 3-layer network approximation $\bar{u}_{\mathcal{T}}^{N}$



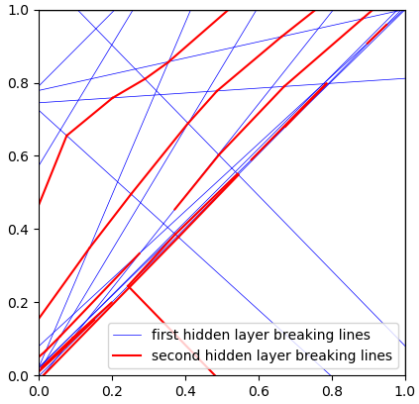
(c) 2-layer network vertical cross section



(d) 3-layer network vertical cross section



(e) 2-layer network breaking lines



(f) 3-layer network breaking lines

Figure 3.7. Approximation results of the problem with a piece-wise smooth solution having a non-constant jump on the interface

Let $\Gamma_-^1 = \{(x,0) : x \in (0,a)\}$ with $a = 43/64$. For the inflow boundary condition

$$g(x,y) = \begin{cases} -1, & (x,y) \in \Gamma_-^1, \\ \\ 1, & (x,y) \in \Gamma_-^2 = \Gamma_- \setminus \Gamma_-^1, \end{cases} \tag{3.32}$$

then the exact solution is a piece-wise constant: $u = -1$ in Ω_1 and $u = 1$ in Ω_2, where $\Omega_2 = \Omega \setminus \bar{\Omega}_1$ and

$$\Omega_1 = \{\mathbf{x} \in \Upsilon_1 : \boldsymbol{\xi}_1 \cdot \mathbf{x} < a\} \cup \{\mathbf{x} \in \Upsilon_2 : \boldsymbol{\xi}_2 \cdot \mathbf{x} < a\}.$$

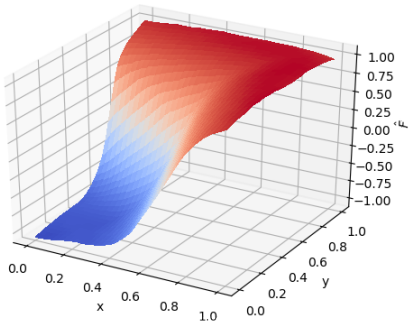
Here, $\boldsymbol{\xi}_1 = (1, \sqrt{2}-1)^T$ and $\boldsymbol{\xi}_2 = (\sqrt{2}-1, 1)^T$ are two vectors normal to $\boldsymbol{\beta}|_{\Upsilon_1}$ and $\boldsymbol{\beta}|_{\Upsilon_2}$, respectively.

The LSNN method with $\rho = h/2$ in (3.29) and a fixed learning rate 0.003 with 50000 iterations is implemented for networks: 2-30-1, 2-200-1, and 2-5-5-1. Initialization of the first layer is done by the approach described earlier, and that of the subsequent layers are randomly generated. The experiment is replicated three times to reduce the variability of random initialization of some parameters in the network. The best result is presented in Table 3.6 and Figure 3.8, and the figures of the two-layer network is for the 2-200-1 model. The traces of the exact and numerical solutions on the plane $x = 0$ and the breaking lines of these two networks are depicted in Fig. 3.8(c,d) and Fig. 3.8(e,f), respectively.

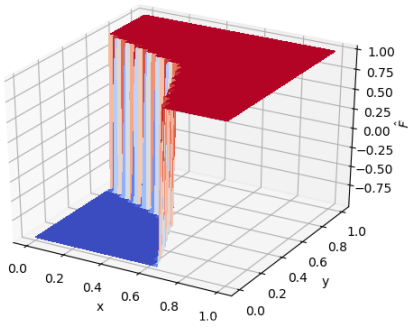
Clearly, the two-layer network with 200 neurons (over 600 parameters) fails to approximate the solution well in average (see Table 3.6) and point-wise (see Figure 3.8). A three-layer network with less than 8% of parameters outperforms this large two-layer network in every aspects including breaking lines. Comparing these two networks, obviously, a three-layer network is necessary to accurately approximate the solution having a constant jump on a piece-wise line segment discontinuous interface. In the subsequent sections, we will theoretically show that a three-layer network is necessary and sufficient in order to well approximate a discontinuous solution with an interface in $\mathbb{R}^2$ that is not a straight line.

Table 3.6. Relative errors of the problem with a piece-wise constant advection velocity field

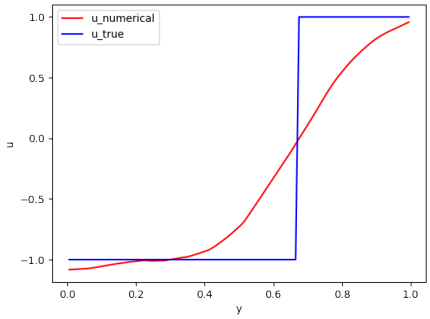
Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_{\boldsymbol{\beta}}}{\|\|\|u\|\|\|_{\boldsymbol{\beta}}}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-30-1	0.487306	0.556949	0.386919	91
2-200-1	0.317839	0.402699	0.259592	601
2-5-5-1	0.086122	0.086131	0.016945	46



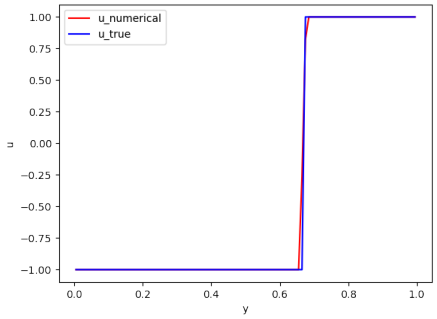
(a) 2-layer network approximation $\bar{u}_T^N$



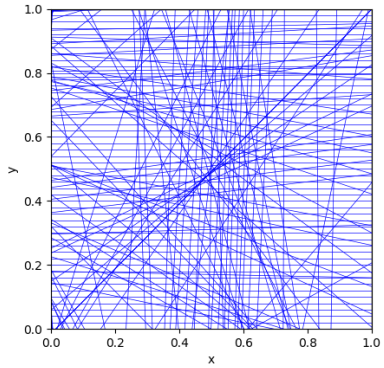
(b) 3-layer network approximation $\bar{u}_T^N$



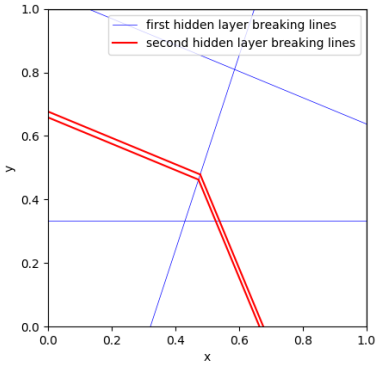
(c) 2-layer network vertical cross section



(d) 3-layer network vertical cross section



(e) 2-layer network breaking lines



(f) 3-layer network breaking lines

Figure 3.8. Approximation results of the problem with a piece-wise constant advection velocity field

3.5.6 Problem with a variable advection velocity field

The sixth test problem is equation (3.2) defined on the domain $\Omega = (0,1)^2$ with a *variable* advection velocity field $\boldsymbol{\beta} = (-y, x)^T$ and $\gamma = f = 0$ (see, e.g., [31], [40]). With the inflow boundary condition g given in (3.32), the exact solution is a piece-wise constant given by

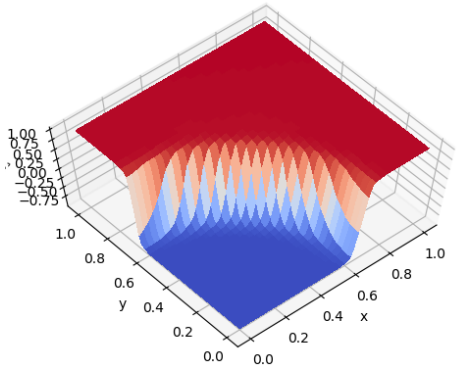
$$u(x,y) = \begin{cases} -1, & (x,y) \in \Omega_1, \\ \\ 1, & (x,y) \in \Omega_2, \end{cases} \tag{3.33}$$

where $\Omega_1 = \{(x,y) \in \Omega : \, x^2 + y^2 < a^2\}$ and $\Omega_2 = \{(x,y) \in \Omega : \, x^2 + y^2 > a^2\}$.

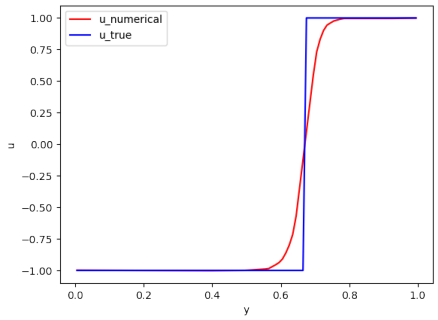
For the LSNN method, again we use a uniform integration mesh $\mathcal{T}$ with the mesh size $h = 10^{-2}$; the finite difference quotient in (3.29) is calculated with $\rho = h/10$ to avoid the use of values on both sides of the interface, if possible; and the parameters are initialized by the strategy described in section 3.4 for the first layer and randomly for the subsequent layers. The learning rate starts with 0.005, and is reduced by half for every 25000 iterations. This learning rate decay strategy is used with 150000 iterations. Due to the random initialization of some parameters, numerical experiments are replicated three times and the best results for the three- and four-layer networks are reported in Table 3.7 and Figure 3.9. The traces of the exact and numerical solutions at the plane $x = 0$ are depicted in Fig. 3.9 (b) and (d) for the respective three- and four-layer networks. As shown in Fig. 3.9 (b), the LSNN approximation of the three-layer network with 40 neurons at each layer smears the discontinuity. A careful examination of the iterative process, it seems to us that the smear is due to the initialization (see Fig. 3.11).

Table 3.7. Relative errors of the problem with a variable advection velocity field

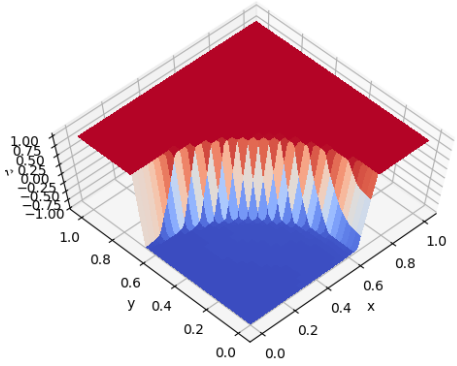
Network structure	$\frac{\|u-\bar{u}_{\mathcal{T}}^N\|_0}{\|u\|_0}$	$\frac{\|\|\|u-\bar{u}_{\mathcal{T}}^N\|\|\|_{\boldsymbol{\beta}}}{\|\|\|u\|\|\|_{\boldsymbol{\beta}}}$	$\frac{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{f})}{\mathcal{L}^{1/2}(\bar{u}_{\mathcal{T}}^N;\mathbf{0})}$	Parameters
2-40-40-1	0.146226	0.187823	0.108551	1761
2-30-30-30-1	0.109266	0.122252	0.039993	1951



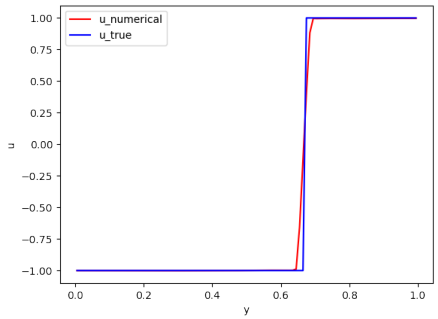
(a) 3-layer network approximation $\bar{u}_{\mathcal{T}}^{N}$



(b) 3-layer network vertical cross section



(c) 4-layer network approximation $\bar{u}_{\mathcal{T}}^{N}$



(d) 4-layer network vertical cross section

Figure 3.9. Approximation results of the problem with a variable advection velocity field

3.6 Method of model continuation

As observed from our numerical experiments for the test problem with a curved discontinuous interface, initial of the parameters plays an important role in training neural networks. This is because the high dimensional non-convex optimization usually have many solutions. Without a good initial, our previous simulations rely on over-parameterized neural networks to approximate the underlying problem well. In practice, the strategy of over-parameterization is computationally expensive.

Based on our numerical experiments in the previous sections, to generate a good initial for the parameters, we introduce the method of continuation through models for the advection-reaction problem in (3.2) with a variable advection velocity field $\boldsymbol{\beta}(\mathbf{x})$. To this end, let $\{\boldsymbol{\beta}_n(\mathbf{x})\}$ be a sequence of piece-wise constant vector fields. Consider the following advection-reaction problem with the advection velocity field $\boldsymbol{\beta}_n(\mathbf{x})$:

$$\begin{cases} (u_n)_{\boldsymbol{\beta}_n} + \hat{\gamma}\, u_n &= f, \quad \text{in } \Omega, \\ u_n &= g, \quad \text{on } \Gamma_-. \end{cases} \tag{3.34}$$

Let u be the solution of (3.2), it is easy to see that $u - u_n$ satisfies

$$\begin{cases} (u-u_n)_{\boldsymbol{\beta}_n} + \hat{\gamma}\,(u-u_n) &= u_{\boldsymbol{\beta}_n} - u_{\boldsymbol{\beta}}, \quad \text{in } \Omega, \\ u - u_n &= 0, \quad \text{on } \Gamma_-, \end{cases} \tag{3.35}$$

which, together with the stability estimate in (3.10), implies

$$\|u-u_n\|_{0,\Omega} \le |||u-u_n|||_{\boldsymbol{\beta}_n} \le C\, \|u_{\boldsymbol{\beta}_n} - u_{\boldsymbol{\beta}}\|_{0,\Omega} = C \left(\int_\Omega \left((\boldsymbol{\beta}_n - \boldsymbol{\beta}) \cdot \nabla u \right) d\mathbf{x} \right)^{1/2}.$$

Hence, if $\boldsymbol{\beta}_n$ is a good approximation to $\boldsymbol{\beta}$, then u_n is a good approximation to u. This indicates that (3.34) provides a continuation process on the parameter n for (3.2).

For the test problem in section 3.5.6, since streamlines of the advection velocity field $\boldsymbol{\beta} = (-y, x)^T$ are quarter circles in $\Omega = (0,1)^2$ oriented counterclockwise, it is natural to

approximate the quarter-circle by n line segments. To this end, let $t_{\rm i} = \dfrac{{\rm i}\pi}{2n}$ for ${\rm i} = 0, 1, ..., n$ and

$$\Upsilon_{{\rm i}+1} = \{(x, y) \in \Omega : \ (\sin t_{\rm i})x < (\cos t_{\rm i})y \ \ \text{and} \ \ (\sin t_{{\rm i}+1})x \geq (\cos t_{{\rm i}+1})y\}.$$

Then $\{\Upsilon_{{\rm i}+1}\}_{{\rm i}=0}^{n-1}$ forms a partition of Ω (see Fig. 3.10 for $n = 4$ case). This type of approximations leads to

$$\boldsymbol{\beta}_n = (\cos t_{{\rm i}+1} - \cos t_{\rm i}, \sin t_{{\rm i}+1} - \sin t_{\rm i})^T \quad \text{in } \Upsilon_{{\rm i}+1}$$

for ${\rm i} = 0, 1, ..., n-1$. Note that $\boldsymbol{\beta}_2$ is the same vector field given in (3.30). Hence, (3.34) with $n = 2$ and the test problem in section 3.5.5 are the same.

The method of model continuation starts with a three-layer neural network (2-5-5-1) to approximate u_2 (see the third row of Table 3.6 and Fig. 3.8 (b,d)). This trained network is used as an initial for the parameters in the hidden layers of the 2-6-6-1 network to approximate u_3 by randomly generated the parameters of new neurons. The initial for the output weights and bias may be chosen as the solution of the system (3.27). The adaptive learning rate strategy which starts with 0.01 and decays by 20% for every 50000 iterations is implemented with the method. The networks for u_n with $n = 4, 5$ are initialized sequentially in a similar fashion. Numerical results for approximating u_n and u are reported in Table 3.8, and the traces of the exact and numerical solutions at the plane $x = 0$ are depicted in Fig. 3.11. The third and fourth columns show that the difficulty of the corresponding problems increase as the number of line segments increase. The fifth column shows that u_n approaches to u monotonously. Comparing Table 3.7 with the last row of Table 3.8, it is clear that the method of model continuation is capable of reducing the number of parameters in the network significantly.

Table 3.8. Relative errors of the problem with discontinuity along line segments

n	Network structure	$\frac{\|u_n - \tilde{u}_{\mathcal{T}}^N\|_0}{\|u_n\|_0}$	$\frac{\|\|\|u_n - \tilde{u}_{\mathcal{T}}^N\|\|\|}{\|\|\|u_n\|\|\|}$	$\frac{\|u - \tilde{u}_{\mathcal{T}}^N\|}{\|u\|}$	$\frac{\mathcal{L}^{1/2}(\tilde{u}_{\mathcal{T}}^N; \mathbf{f})}{\mathcal{L}^{1/2}(\tilde{u}_{\mathcal{T}}^N; \mathbf{0})}$	Parameters
3	2-6-6-1	0.075817	0.080026	0.244483	0.059422	61
4	2-6-6-1	0.104372	0.110954	0.216481	0.064744	61
5	2-8-8-1	0.097836	0.109648	0.135606	0.049938	97
curve	2-25-25-1	0.141261	0.187616	0.141261	0.077233	726

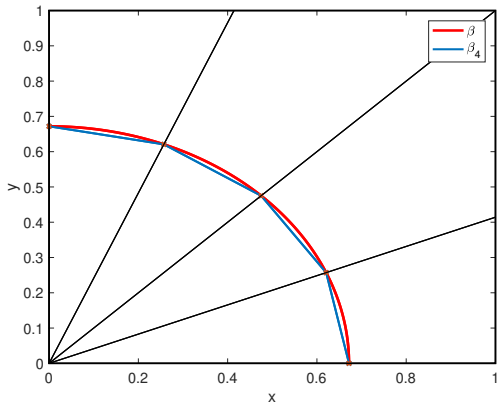
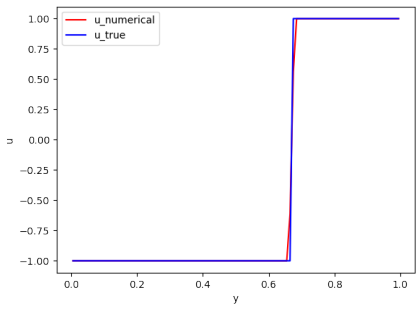
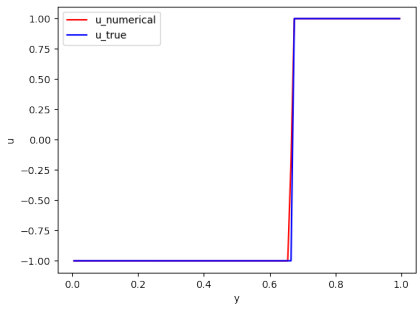


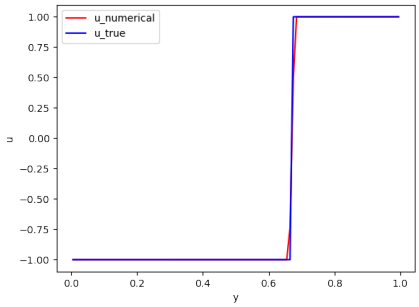
Figure 3.10. Discontinuous interface for $n = 4$



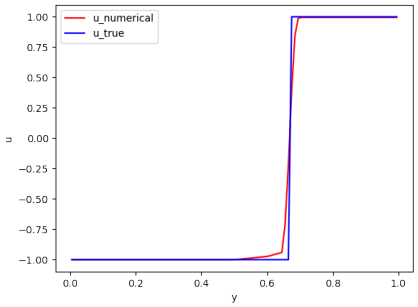
(a) 3 line segments



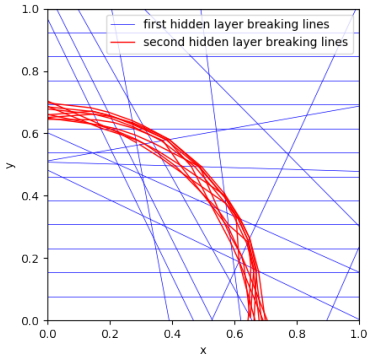
(b) 4 line segments



(c) 5 line segments



(d) Curved discontinuity



(e) Network breaking lines of the problem with curved discontinuity

Figure 3.11. Traces of the exact and numerical solutions for the problem with discontinuity along line segments

3.7 ReLU NN Approximation of Discontinuous Solutions: II. Discontinuous Interface Along Line Segments

In section 3.3, we theoretically showed that a two-layer ReLU neural network is sufficient to accurately approximate a piece-wise defined solution with a constant jump on the discontinuous interface consisting of a straight line. Numerical experiments in section 3.5.1-3.5.3 confirm our theoretical results. Besides, in section 3.5.5 and 3.6, we also numerically show that a three-layer ReLU neural network is needed to approximate the solution well if the discontinuity interface is not a straight line. Then the goal of this section is to theoretically show that a three-layer ReLU is necessary to approximate such solution.

First, we restrict our attention to the solution of problem in section 3.5.5, i.e., the solution is a piece-wise constant and the discontinuous interface consists of two line segments. To make it slightly general, let

$$\chi = \begin{cases} \alpha_1, & \mathbf{x} \in \Omega_1, \\ \\ \alpha_2, & \mathbf{x} \in \Omega_2. \end{cases}$$

Without loss of generality, assume that $\alpha_1 < \alpha_2$. Let $p_1(\mathbf{x})$ and $p_2(\mathbf{x})$ be two-layer neural network functions given by

$$p_{\mathrm{i}}(\mathbf{x}) = \alpha_1 + \frac{\alpha_2 - \alpha_1}{2\varepsilon}\Big(\sigma(\boldsymbol{\xi}_{\mathrm{i}} \cdot \mathbf{x} - a + \varepsilon) - \sigma(\boldsymbol{\xi}_{\mathrm{i}} \cdot \mathbf{x} - a - \varepsilon)\Big)$$

for any $\varepsilon > 0$ such that intersections between the domain Ω and the hyper-planes $\boldsymbol{\xi}_{\mathrm{i}} \cdot \mathbf{x} = a \pm \varepsilon$ are not empty.

Lemma 3.7.1. *Let $p(\mathbf{x}) = \max\{p_1(\mathbf{x}), p_2(\mathbf{x})\}$, then we have*

$$\|\chi - p\|_{0,\Omega} = \left(\|\chi - p\|_{0,\Omega}^2 + \|\chi_{\boldsymbol{\beta}} - p_{\boldsymbol{\beta}}\|_{0,\Omega}^2\right)^{1/2} \leq \sqrt{\frac{2}{3}} D^{(d-1)/2} \left|\alpha_1 - \alpha_2\right| \sqrt{\varepsilon}, \tag{3.36}$$

where D is the diameter of the domain Ω.

Proof. Since $p(\mathbf{x}) = p_{\mathrm{i}}(\mathbf{x})$ in Υ_{i} for i = 1, 2 and $\Omega = \Upsilon_1 \cup \Upsilon_2$, we have

$$\|\chi - p\|_{0,\Omega}^2 = \|\chi - p_1\|_{0,\Upsilon_1}^2 + \|\chi - p_2\|_{0,\Upsilon_2}^2.$$

Combining with the fact that $\chi_\beta - p_\beta = 0$ in Ω, (3.36) is then a direct consequence of Lemma 3.3.2. □

Similar as the discussion in [41], the maximum operation can be constructed by using an additional hidden layer of the ReLU network with 4 neurons:

$$\max\{a,b\} = \frac{a+b}{2} + \frac{|a-b|}{2} = \mathbf{v}\,\sigma\left(\boldsymbol{\omega}\begin{bmatrix} a \\ b \end{bmatrix}\right)$$

where the row vector and the 4×2 matrix are given by

$$\mathbf{v} = \frac{1}{2}[1,-1,1,1] \quad \text{and} \quad \boldsymbol{\omega} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix},$$

respectively.

Lemma 3.7.1 indicates that a three-layer neural network is sufficient when the interface consists of two line segments and the jump is a constant. This result may be extended to discontinuous interfaces consisting of more than two line segments in a similar fashion. In the remainder of this section, we theoretically show that a three-layer neural network is able to accurately approximate the solution with the discontinuous interface along multiple line segments.

The result is established on the lattice representation theorem [24]. Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is a continuous function that are piecewise linear on m subdomains of the domain Ω:

$$\Omega_{\mathrm{i}}, \quad \mathrm{i} = 1, \cdots, m.$$

This is the same as on each Ω_{i}, f is a linear function:

$$f(x) = f_{\mathrm{i}}(x) = a_{\mathrm{i}} \cdots x + b_{\mathrm{i}}, \quad x \in \Omega_{\mathrm{i}}$$

where $a_{\mathrm{i}} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. Then there exists M unique-order subdomains $\tilde{\Omega}_k$ for $k = 1, \cdots M$ such that the sign of $f_{\mathrm{i}} - f_{\mathrm{j}}$ does not change on each $\tilde{\Omega}_k$. The main theorem states as follows:

Theorem 3.7.1. *For every continuous piece-wise linear function $f : \mathbb{R}^n \to \mathbb{R}$ with finite pieces defined by the distinct local linear functions l_{i} for $\mathrm{i} = 1, \cdots, m$ and $\{\tilde{\Omega}_k\}_{k=1}^M$ be the unique order subdomains. Then there exists finite non-empty subsets of $\{1, 2, \cdots, m\}$, say s_k for $1 \leq k \leq M$ such that*

$$f(x) = \max_{1 \leq k \leq M} \{\min_{\mathrm{i} \in s_k} l_{\mathrm{i}}\}. \tag{3.37}$$

Based on the theorem, the authors in [41], [42] showed that every continuous piece-wise linear function in $\mathbb{R}^d$ can be represented by a ReLU neural network model with at most $\lceil \log_2^{(d+1)} \rceil$ hidden layers. Then we denote $\{C_n\}$ as a set of regions of u_n that determines a domain partition and u_n is expressed by a piece-wise constant function in each region $C \in C_n$. Therefore, u_n can be represented by (3.37) and a three-layer ReLU network is able to express such function. Note that the lattice representation theorem does not require the region to be convex, it is applicable to both convex and non-convex $\{C_n\}$.

Together with the universal approximation property, the results of this section as well as conclusions of section 3.3 imply that a two- or three-layer ReLU neural network is sufficient to accurately approximate the solution of the linear advection-reaction problem without oscillation.

3.8 Discussion

For the linear advection-reaction problem, we proposed the least-squares neural network (LSNN) method. The least-squares formulation, based on a direct application of the least-squares principle to the underlying problem, does not require additional smoothness of the solution if $f \in L^2(\Omega)$. In the V_β norm, the LSNN approximation is proved to be quasi-optimal, i.e., the error of the LSNN approximation is bounded above by the approximation error of the network.

The main challenge in numerical simulation of the hyperbolic partial differential equation is the discontinuity of their solution. In this chapter, we theoretically and numerically show that the LSNN method using a two- or three-layer network is capable of approximating the

discontinuous solution accurately without oscillation. In particular, the piece-wise constant solution can be approximated well by a ReLU network with a small number of neurons. From this perspective, the LSNN method outperforms the traditional mesh-based method.

In addition, numerical test in section 3.5.2 indicates that properly choosing the scheme to approximate the differential operator in the least-squares functional is important for the success of the LSNN method. Discretizing the differential operator along each coordinate direction would result in the failure of resolving the discontinuous interface. More discussions will be presented in the subsequent chapter for the nonlinear case.

Furthermore, we also proposed a method of model continuation motivated by the test problem in section 3.5.6. By employing the method of model continuation, numerical results show that the strategy is effective for reducing the number of the parameters of the network.

In the subsequent chapter, we will extend the LSNN method to solve scalar nonlinear hyperbolic conservation laws.

4. LEAST-SQUARES NEURAL NETWORK METHOD FOR SCALAR NONLINEAR HYPERBOLIC CONSERVATION LAW

A version of this chapter has been submitted for publication [43].

In the previous chapter, we introduced the least-squares ReLU neural network method for solving linear advection-reaction problems with discontinuous solutions and showed that the method outperforms mesh-based numerical methods in terms of the number of degrees of freedom. In this chapter, we focus on the application of the least-squares ReLU neural network method to scalar nonlinear hyperbolic conservation laws.

4.1 Introduction

Let Ω be a bounded domain in $\mathbb{R}^d$ ($d = 1$, 2, or 3) with Lipschitz boundary, consider the scalar nonlinear hyperbolic conservation law

$$\begin{cases} u_t(\mathbf{x},t) + \nabla_{\mathbf{x}} \cdot \mathbf{f}(u) &= 0, & \text{in } \Omega \times I, \\ u &= g, & \text{on } \Gamma_-, \\ u(\mathbf{x},0) &= u_0(\mathbf{x}), & \text{in } \Omega, \end{cases} \tag{4.1}$$

where u_t is the partial derivative of u with respect to temporal variable t; $\nabla_{\mathbf{x}}\cdot$ is a divergence operator with respect to spatial variable $\mathbf{x}$; $\mathbf{f}(u) = (f_1(u), ..., f_d(u))$ is the spatial flux vector field; $I = (0, T)$ is temporal interval; Γ_- is the part of the boundary $\partial\Omega \times I$ where the characteristic curves enter the domain $\Omega \times I$; and the boundary data g and the initial data u_0 are given scalar-valued functions.

Numerical methods for the scalar nonlinear hyperbolic conservation law (4.1) have been intensively studied during the past several decades by many researchers and many numerical schemes have been developed. A major difficulty in numerical simulation is that the solution of a scalar hyperbolic conservation law is often discontinuous due to the discontinuous initial condition or shock formation; in addition, there is no a priori knowledge of the location and shape of the discontinuity interface. Given the success of using LSNN method to approxi-

mate the discontinuous solutions for the linear advection-reaction problems, it is desired to replicate such success for scalar nonlinear hyperbolic conservation laws.

For the nonlinear case, the partial differential equation is not generally sufficient to determine the solution. An additional constraint the so-called Rankine-Hugoniot (RH) jump condition [14], [44], [45], is needed at where the solution is not continuous. To enforce this condition weakly, [35] introduced an independent variable, the spatial-temporal flux, for the inviscid Burgers equation and applied the least-squares principle to the resulting equivalent system. A variant of this method was also studied in [35], [46] by using the Helmholtz decomposition of the flux.

Due to the training difficulty of the least-squares method of [35], we employ the naive least-squares method used for the linear advection-reaction problems, i.e., a direct application of least-squares principle to the PDE, initial and inflow boundary conditions. For the nonlinear hyperbolic conservation law, to ensure that the numerical solution does not violate the RH jump condition condition, we approximate the differential operator by following ideas of the conservative schemes such as Roe's scheme, ENO, etc.

The numerical results show that it is difficult to train the space-time LSNN method when the computational domain $\Omega \times I$ is relatively large, even though the NN model we used is relatively small for approximating the solution of the underlying problem well. This experience motivates us to propose the modified LSNN method for the nonlinear problem, i.e., the block space-time LSNN method. The basic idea is to partition the computational domain into a number of blocks based on the "inflow" boundary and initial conditions, then the LSNN method solves problems on these blocks sequentially. The trained parameters of the NN model for the previous block is used as an initial for the current block, which guarantees a close enough first approximation as the network initialization.

This chapter is organized as follows. The space-time LSNN method for scalar nonlinear hyperbolic conservation law is introduced in section 4.2. Conservative finite difference operators are described in section 4.3. The block LSNN method is proposed in section 4.4. Finally, implementation and numerical results for various benchmark one dimensional problems are presented in section 4.5.

4.2 Space-Time Least-Squares Neural Network Method for Scalar Nonlinear Hyperbolic Conservation Law

In this section, we will apply the LSNN method to solve the scalar nonlinear hyperbolic conservation laws on a space-time computational domain.

As described in chapter 2, we employ the deep neural network (DNN) to define the following scalar-valued function of the spatial and temporal variables:

$$\mathcal{N} : \mathbf{z} = (\mathbf{x}, t) \in \mathbb{R}^{d+1} \longrightarrow \mathcal{N}(\mathbf{z}) \in \mathbb{R}.$$

Then denote the set of such DNN functions by

$$\mathcal{M}(\boldsymbol{\theta}, L) = \Big\{\mathcal{N}(\mathbf{z}) = \boldsymbol{\omega}^{(L)} \Big(N^{(L-1)} \circ \cdots N^{(2)} \circ N^{(1)}(\mathbf{z})\Big) - b^{(L)} : \boldsymbol{\omega}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}},\ \mathbf{b}^{(l)} \in \mathbb{R}^{n_l}\Big\},$$

where $N^{(l)}(\mathbf{z}^{(l-1)})$ is defined in (2.2) and $\boldsymbol{\theta}$ denotes all parameters: $\boldsymbol{\omega}^{(l)}$ and $\mathbf{b}^{(l)}$ for $l = 1, ..., L$. It is easy to see that $\mathcal{M}(\boldsymbol{\theta}, L)$ is a set, but not a linear space.

Applying the least-squares principle directly to the problem in (4.1), we have the following least-squares (LS) functional

$$\mathcal{L}(v; g) = \|v_t + \nabla_{\mathbf{x}} \cdot \mathbf{f}(v)\|^2_{0,\Omega\times I} + \|v - g\|^2_{0,\Gamma_-} + \|v(\mathbf{x}, 0) - u_0(\mathbf{x})\|^2_{0,\Omega}. \tag{4.2}$$

Then the least-squares approximation is to find $u_{_N}(\mathbf{x}, t; \boldsymbol{\theta}^*) \in \mathcal{M}(\boldsymbol{\theta}, L)$ such that

$$\mathcal{L}\Big(u_{_N}(\cdot; \boldsymbol{\theta}^*);\, \mathbf{f}\Big) = \min_{v\in\mathcal{M}(\boldsymbol{\theta},L)} \mathcal{L}\Big(v(\cdot; \boldsymbol{\theta});\, g\Big) = \min_{\boldsymbol{\theta}\in\mathbb{R}^N} \mathcal{L}\Big(v(\cdot; \boldsymbol{\theta});\, g\Big), \tag{4.3}$$

where N is the total number of parameters in $\mathcal{M}(\boldsymbol{\theta}, L)$ given by

$$N = M_d(L) = \sum_{l=1}^{L} n_l \times (n_{l-1} + 1).$$

Similar to the LSNN method for the linear advection-reaction equations and [2], the integral in the LS functional is evaluated by numerical integration. To do so, let

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

be a partition of the domain Ω. Then

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

are partitions of the boundary Γ_- and $\Omega \times \{0\}$, respectively. Let $\mathbf{z}_K = (\mathbf{x}_K, t_K)$ and $\mathbf{z}_E = (\mathbf{x}_E, t_E)$ be the centroids of $K \in \mathcal{T}$ and E in $\mathcal{E}_-$ or $\mathcal{E}_0$, respectively. Define the discrete LS functional as follows:

$$\mathcal{L}_\tau\Big(v(\cdot;\boldsymbol{\theta}); g\Big) = \sum_{K\in\mathcal{T}} \Big(\delta_\tau v + \nabla_{\mathbf{h}} \cdot \mathbf{f}(v)\Big)^2(\mathbf{z}_K;\boldsymbol{\theta})\,|K| + \sum_{E\in\mathcal{E}_-} \Big(v-g\Big)^2(\mathbf{z}_E;\boldsymbol{\theta})|E| + \sum_{E\in\mathcal{E}_0} \Big(v-u_0\Big)^2(\mathbf{z}_E;\boldsymbol{\theta})|E|, \tag{4.4}$$

where $|K|$ and $|E|$ are the d and $d-1$ dimensional measures of K and E, respectively; δ_τ and $\nabla_{\mathbf{h}}$ are finite difference operators to be defined in the subsequent section. Then the discrete least-squares approximation is to find $u_\tau(\mathbf{z}, \boldsymbol{\theta}^*) \in \mathcal{M}(\boldsymbol{\theta}, L)$ such that

$$\mathcal{L}_\tau\Big(u_\tau(\cdot,\boldsymbol{\theta}^*); g\Big) = \min_{v\in\mathcal{M}(\boldsymbol{\theta},L)} \mathcal{L}_\tau\Big(v(\cdot;\boldsymbol{\theta});\ g\Big) = \min_{\boldsymbol{\theta}\in\mathbb{R}^N} \mathcal{L}_\tau\Big(v(\cdot;\boldsymbol{\theta}); g\Big). \tag{4.5}$$

4.3 Conservative Finite Difference Operator

How to discretize the differential operator is fairly critical for the success of the LSNN method. For the linear advection-reaction problems, we observe that using finite difference quotient along coordinate directions to approximate the differential operator fails to resolve the discontinuity (as shown in section 3.5.2). For the nonlinear problems, we employ conservative finite volume schemes to evaluate the derivatives in the least-squares functional in (4.4). There are many well-developed conservative schemes such as Roe's scheme, ENO, and WENO, etc. (see, e.g., [15], [47], [48]). For simplicity, we briefly describe the discrete finite

difference operator using the idea of either Roe's scheme or the second-order accurate ENO scheme in this section.

For any $K \in \mathcal{T}$, let $(\mathbf{x}_{_K}, t_{_K})$ be the centroid of K. Let $(\mathbf{h}, \tau) = (h_1, ..., h_d, \tau)$ be sufficiently small step size such that $(\mathbf{x}_{_K} \pm \mathbf{h}, t_{_K} \pm \tau) \in K$. For $\mathrm{i} = 1, ..., d$, let $\mathbf{h}_{\mathrm{i}} = h_{\mathrm{i}}\mathbf{e}_{\mathrm{i}}$, where $\mathbf{e}_{\mathrm{i}}$ is the unit vector in the x_{i}-coordinate direction. Then the finite difference operator of the least-squares functional in (4.4) at the point $(\mathbf{x}_{_K}, t_{_K})$ is given by

$$\begin{aligned} &\Big(\delta_\tau v + \nabla_{\mathbf{h}} \cdot \mathbf{f}(v)\Big)(\mathbf{x}_{_K}, t_{_K}) \\ = \;& \frac{v\Big(\mathbf{x}_{_K}, t_{_K}\Big) - v\Big(\mathbf{x}_{_K}, t_{_K} - \tau\Big)}{\tau} + \sum_{\mathrm{i}=1}^{d} \frac{\hat{f}_{\mathrm{i}}\Big(v(\mathbf{x}_{_K} + \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K})\Big) - \hat{f}_{\mathrm{i}}(v(\mathbf{x}_{_K} - \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K}))}{h_{\mathrm{i}}}, \end{aligned} \tag{4.6}$$

where $\hat{f}_{\mathrm{i}}\Big(v(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K})\Big)$ are the i^{th} component of the numerical flux at $(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K})$. Various conservative schemes are more or less on how to reconstruct proper numerical flux.

Below, we describe how to reconstruct numerical fluxes by either Roe's scheme or ENO. To this end, we introduce the Roe speed at point $(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K})$ in the $\mathbf{e}_{\mathrm{i}}$ direction

$$a_{\mathrm{i}}\left(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K}\right) = \begin{cases} \dfrac{f_{\mathrm{i}}\Big(v(\mathbf{x}_{_K} \pm \mathbf{h}_{\mathrm{i}}, t_{_K})\Big) - f_{\mathrm{i}}(v(\mathbf{x}_{_K}, t_{_K}))}{v\Big(\mathbf{x}_{_K} \pm \mathbf{h}_{\mathrm{i}}, t_{_K}\Big) - v(\mathbf{x}_{_K}, t_{_K})}, & \text{if } v\Big(\mathbf{x}_{_K} \pm \mathbf{h}_{\mathrm{i}}, t_{_K}\Big) \neq v(\mathbf{x}_{_K}, t_{_K}), \\ f_{\mathrm{i}}'\Big(v(\mathbf{x}_{_K}, t_{_K})\Big), & \text{if } v\Big(\mathbf{x}_{_K} \pm \mathbf{h}_{\mathrm{i}}, t_{_K}\Big) = v(\mathbf{x}_{_K}, t_{_K}). \end{cases} \tag{4.7}$$

Then the i^{th} components of the Roe numerical flux at $(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K})$ are given by

$$\begin{aligned} &\hat{f}_{\mathrm{i}}\left(v\Big(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K}\Big)\right) \\ = \;& \frac{f_{\mathrm{i}}\Big(v(\mathbf{x}_{_K}, t_{_K})\Big) + f_{\mathrm{i}}\Big(v(\mathbf{x}_{_K} \pm \mathbf{h}_{\mathrm{i}}, t_{_K})\Big)}{2} \mp \left|a_{\mathrm{i}}\left(\mathbf{x}_{_K} \pm \frac{1}{2}\mathbf{h}_{\mathrm{i}}, t_{_K}\right)\right| \frac{v\Big(\mathbf{x}_{_K} \pm \mathbf{h}_{\mathrm{i}}, t_{_K}\Big) - v(\mathbf{x}_{_K}, t_{_K})}{2}. \end{aligned}$$

The key idea of the Roe's scheme is to enforce the RH condition by using the finite volume approximation and use only grid points, if possible, on one side of the interface for constructing a finite difference scheme. This is done through the signs of the Roe speed a_{i} at midpoints. This idea was further explored for developing higher order schemes, e.g., the ENO schemes introduced in [49] (see also [47], [48]), by employing extra grid points.

To make sure all used grid points locate on one side of the interface, it requires additional decisions and, hence, the ENO schemes are generally sophisticated.

For simplicity, we describe the second order ENO numerical flux here. The ENO uses the sign of the Roe speed to build up upwind scheme. Specifically,

$$\hat{f}_{\mathrm{i}}\left(v(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K})\right)=\begin{cases}\hat{f}_{\mathrm{i}}^{-}\left(v(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K})\right), & \text{if } a_{\mathrm{i}}\left(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K}\right)\geq 0,\\ \hat{f}_{\mathrm{i}}^{+}\left(v(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K})\right), & \text{if } a_{\mathrm{i}}\left(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K}\right)<0.\end{cases} \tag{4.8}$$

Additionally, ENO uses the magnitudes of the finite difference quotient of the i^{th} component of the flux with respect to x_{i} over the neighboring intervals to determine which side of grid points are used. In this way, again ENO tries to use grid points on one side of the discontinuity if possible. More precisely, let

$$\begin{aligned} f_{\mathrm{i}}(\mathbf{x}_{_K},t_{_K};\mathbf{h}_{\mathrm{i}}) &= \frac{f_{\mathrm{i}}\big(v(\mathbf{x}_{_K}+\mathbf{h}_{\mathrm{i}},t_{_K}))-f_{\mathrm{i}}(v(\mathbf{x}_{_K},t_{_K}))}{h_{\mathrm{i}}}\\ \text{and}\quad f_{\mathrm{i}}(\mathbf{x}_{_K},t_{_K};-\mathbf{h}_{\mathrm{i}}) &= \frac{f_{\mathrm{i}}\big(v(\mathbf{x}_{_K},t_{_K})\big)-f_{\mathrm{i}}\big(v(\mathbf{x}_{_K}-\mathbf{h}_{\mathrm{i}},t_{_K}))}{h_{\mathrm{i}}}. \end{aligned}$$

In the case that $a_{\mathrm{i}}\left(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K}\right)\geq 0$, combining with (4.8), the ENO numerical flux is given by

$$\begin{aligned} &\hat{f}_{\mathrm{i}}\left(v(\mathbf{x}_{_K}+\frac{1}{2}\mathbf{h}_{\mathrm{i}},t_{_K})\right)\\ =\ &\begin{cases} -\frac{1}{2}f_{\mathrm{i}}\big(v(\mathbf{x}_{_K}-\mathbf{h}_{\mathrm{i}},t_{_K}))+\frac{3}{2}f_{\mathrm{i}}(v(\mathbf{x}_{_K},t_{_K})), & \text{if } \big|f_{\mathrm{i}}(\mathbf{x}_{_K},t_{_K};-\mathbf{h}_{\mathrm{i}})\big|<|f_{\mathrm{i}}(\mathbf{x}_{_K},t_{_K};\mathbf{h}_{\mathrm{i}})|,\\ \frac{1}{2}f_{\mathrm{i}}\big(v(\mathbf{x}_{_K},t_{_K})\big)+\frac{1}{2}f_{\mathrm{i}}\big(v(\mathbf{x}_{_K}+\mathbf{h}_{\mathrm{i}},t_{_K}))\big), & \text{if } \big|f_{\mathrm{i}}(\mathbf{x}_{_K},t_{_K};-\mathbf{h}_{\mathrm{i}})\big|\geq|f_{\mathrm{i}}(\mathbf{x}_{_K},t_{_K};\mathbf{h}_{\mathrm{i}})|. \end{cases} \end{aligned} \tag{4.9}$$

If $a_{\rm i}\left(\mathbf{x}_{K}+\frac{1}{2}\mathbf{h}_{\rm i},t_{K}\right)<0$, then the numerical flux is reconstructed by

$$\hat{f}_{\rm i}\left(v(\mathbf{x}_{K}+\frac{1}{2}\mathbf{h}_{\rm i},t_{K})\right)$$

$$=\begin{cases}\frac{1}{2}f_{\rm i}\Big(v(\mathbf{x}_{K},t_{K})\Big)+\frac{1}{2}f_{\rm i}\Big(v(\mathbf{x}_{K}+\mathbf{h}_{\rm i},t_{K})\Big), & \text{if } \Big|f_{\rm i}(\mathbf{x}_{K}+\mathbf{h}_{\rm i},t_{K};-\mathbf{h}_{\rm i})\Big|<|f_{\rm i}(\mathbf{x}_{K}+\mathbf{h}_{\rm i},t_{K};\mathbf{h}_{\rm i})|,\\ \frac{3}{2}f_{\rm i}\Big(v(\mathbf{x}_{K}+\mathbf{h}_{\rm i},t_{K})\Big)-\frac{1}{2}f_{\rm i}(v(\mathbf{x}_{K}+2\mathbf{h}_{\rm i},t_{K})), & \text{if } \Big|f_{\rm i}(\mathbf{x}_{K}+\mathbf{h}_{\rm i},t_{K};-\mathbf{h}_{\rm i})\Big|\geq|f_{\rm i}(\mathbf{x}_{K}+\mathbf{h}_{\rm i},t_{K};\mathbf{h}_{\rm i})|.\end{cases}$$

In a similar fashion, $\hat{f}_{\rm i}\left(v(\mathbf{x}_{K}-\frac{1}{2}\mathbf{h}_{\rm i},t_{K})\right)$ may be defined accordingly and we finish describing the numerical flux reconstruction by using ENO.

Similar idea can be extended to construct higher-order accurate ENO scheme, more details can be found in [49].

4.4 Block Space-Time Least-Squares Neural Network Method

For the linear advection-reaction equations, we directly implement the LSNN method on the entire computational domain. However, for the nonlinear hyperbolic conservation law, our numerical results show that it is difficult to train the LSNN method when the computational domain Ω is relatively large, even though the NN model we used is relatively small for approximating the solution of the underlying problem well. Since we have limited understanding about the optimization (training) process, this numerical experience motivates us to propose the block space-time least-squares neural network method to compensate.

For clarity of exposition, let us consider one-dimensional problem defined on $\Omega=(a,b)\times(0,T)$. Without loss of generality, assume that $\tilde{\Gamma}_{-}=\{(a,t)|\,t\in(0,T)\}$ is the part of the boundary where the characteristic curves enter the domain Ω. Hence,

$$\Gamma_{-}=\tilde{\Gamma}_{-}\cup\{(x,0)|\,x\in(0,T)\}$$

is the "inflow" boundary of Ω. Let m_0 be a positive integer and let

$$\Omega_1=\left(a,a+\frac{b-a}{m_0}\right)\times\left(0,\frac{T}{m_0}\right)\quad\text{and}\quad\Omega_{\rm i}=\left(a,a+{\rm i}\frac{(b-a)}{m_0}\right)\times\left(0,{\rm i}\frac{T}{m_0}\right)\setminus\Omega_{\rm i-1}$$

for $\mathrm{i} = 2, ..., m_0 - 1$. It is clear that $\{\Omega_\mathrm{i}\}_{\mathrm{i}=1}^{m_0}$ forms a partition of the domain Ω. Denote by $u_\mathrm{i} = u|_{\Omega_\mathrm{i}}$ the restriction of the solution u of (4.1) on Ω_i, then u_i is the solution of the following problem:

$$\begin{cases} (u_\mathrm{i})_t + \nabla_{\mathbf{x}} \cdot \mathbf{f}(u_\mathrm{i}) = 0, & \text{in } \Omega_\mathrm{i} \in \mathbb{R}^2, \\ u_\mathrm{i} = g, & \text{on } \Gamma_-^\mathrm{i} = \Gamma_- \cap \partial\Omega_\mathrm{i}, \\ u_\mathrm{i} = u_{\mathrm{i}-1}, & \text{on } \Gamma_{\mathrm{i}-1,\mathrm{i}} = \partial\Omega_{\mathrm{i}-1} \cap \partial\Omega_\mathrm{i} \end{cases} \tag{4.10}$$

for $\mathrm{i} = 1, ..., m_0$, where $\partial\Omega_0 = \emptyset$.

Define the least-squares functional for problem (4.10) by

$$\mathcal{L}^\mathrm{i}\Big(v; u_{\mathrm{i}-1}, g\Big) = \|v_t(\mathbf{x}, t) + \nabla_{\mathbf{x}} \cdot \mathbf{f}(v)\|_{0,\Omega_\mathrm{i}}^2 + \|v - g\|_{0,\Gamma_-^\mathrm{i}}^2 + \|v - u_{\mathrm{i}-1}\|_{0,\Gamma_{\mathrm{i}-1,\mathrm{i}}}^2.$$

Then the corresponding discrete least-squares functional $\mathcal{L}_\mathcal{T}^\mathrm{i}\Big(v(\cdot;\boldsymbol{\theta}); u_{\mathrm{i}-1}, g\Big)$ over the subdomain Ω_i may be defined in a similar fashion as in (4.4). Now, we can introduce the block space-time LSNN method accordingly, i.e., the method is to find $u_\mathcal{T}^\mathrm{i}(\mathbf{z}, \boldsymbol{\theta}_\mathrm{i}^*) \in \mathcal{M}(\boldsymbol{\theta}, L)$ such that

$$\mathcal{L}_\mathcal{T}^\mathrm{i}\Big(u_\mathcal{T}^\mathrm{i}(\cdot, \boldsymbol{\theta}_\mathrm{i}^*); g\Big) = \min_{v \in \mathcal{M}(\boldsymbol{\theta}, L)} \mathcal{L}_\mathcal{T}^\mathrm{i}\Big(v(\cdot;\boldsymbol{\theta});\, g\Big) = \min_{\boldsymbol{\theta} \in \mathbb{R}^N} \mathcal{L}_\mathcal{T}^\mathrm{i}\Big(v(\cdot;\boldsymbol{\theta}); g\Big) \tag{4.11}$$

for $\mathrm{i} = 1, ..., m_0$. In a similar fashion, the block space-time LSNN method can be extended to multi-dimensional problems.

Remark 4.4.1. *To start from a close enough first approximation in the training process, the NN model $\mathcal{M}(\boldsymbol{\theta}, L)$ is determined by the first subdomain and will be used for all subdomains. The trained parameter $\boldsymbol{\theta}_\mathrm{i}^*$ from the i^{th}-subdomain is a good approximation to the parameters of the $(\mathrm{i}+1)^{th}$-subdomain and, hence, may be used as an initial. This is because the solution in the current block is the evolution of the solution in the previous block.*

The block space-time LSNN method is based on a proper partition of the domain Ω depending on the "inflow" boundary of the domain. For example, in one dimension case, if

$$\Gamma_- = \{(x, t) \in [a, b] \times [0, T] |\, x = a,\ x = b,\ \text{or },\, t = 0\},$$

then the domain Ω may be partitioned by time blocks as

$$\Omega_{\mathrm{i}} = (a,b) \times \Big((\mathrm{i}-1)T/m_0, \mathrm{i}T/m_0 \Big) \tag{4.12}$$

for $\mathrm{i} = 1, ..., m_0$. Then the block space-time LSNN method may be defined accordingly.

4.5 Implementation and Numerical Experiments

In this section, we present numerical results for one dimensional benchmark test problems. Test problems include scalar nonlinear hyperbolic conservation law: (1) inviscid Burgers equation, i.e., $\mathbf{f}(u) = \frac{1}{2}u^2$ (section 4.5.1-4.5.4) and (2) $\mathbf{f}(u) = \frac{1}{4}u^4$ (section 4.5.5).

The domain $\Omega = (a,b) \times (0,T)$ is partitioned into time blocks as (4.12) and m_0 is the number of blocks, which is empirically chosen for different test problems. Integration mesh $\mathcal{T}$ is obtained by uniformly partitioning all subdomains Ω_{i} into identical squares with the mesh size $h = 0.01$ for $\mathrm{i} = 1, \cdots, m_0$. The block space-time LSNN method is implemented, and the minimization problem in (4.11) is numerically solved using the Adams version of gradient descent [39] with a fixed or an adaptive learning rate. To ensure the conservation, the discretization sizes τ and h_{i} in both Roe (4.6) and ENO (4.9) schemes should be the same as the quadrature size.

As presented in the previous chapter, a three-layer NN is sufficient for approximating the discontinuous solutions of linear advection-reaction problems. This is the reason for us to choose three-layer neural networks for all test problems since there is not a priori knowledge of the location and shape of the shock. The same architecture of three-layer NN models are used for all blocks. As suggested in Remark 4.4.1, the parameters of the network for the current block are initialized by the values of the parameters of the network in the previous block. For the first block, the parameters of the second hidden layer are initialized randomly; and those of the first hidden layer are initialized using the strategy introduced in section 3.4. Again, we briefly describe here for the convenience of reading: Let $\boldsymbol{\omega}_{\mathrm{i}} \in \mathcal{S}^1$ and $b_{\mathrm{i}} \in \mathbb{R}$ be the weights and bias of the i^{th} neuron of the first hidden layer of the first block NN model, respectively, where $\mathcal{S}^1$ is the unit circle in $\mathbb{R}^2$. Initial of $\{(\boldsymbol{\omega}_{\mathrm{i}}, b_{\mathrm{i}})\}_{\mathrm{i}=1}^{n_1}$ is chosen so that the hyper-planes $\{\boldsymbol{\omega}_{\mathrm{i}} \cdot (x,t) = b_{\mathrm{i}})\}_{\mathrm{i}=1}^{n_1}$ form a uniform partition of the first block Ω_1. In addition,

without an effective training strategy, we observe from the experiment that adding a weight α to the L^2 loss of the initial condition in (4.11) is helpful for the training. Then the following least-squares functional is used in the implementation

$$\mathcal{L}^{\mathrm{i}}\Big(v; u_{\mathrm{i}-1}, g\Big) = \|v_t(\mathbf{x},t) + \nabla_{\mathbf{x}}\cdot\,\mathbf{f}(v)\|^2_{0,\Omega_{\mathrm{i}}} + \|v-g\|^2_{0,\Gamma^{\mathrm{i}}_{-}} + \alpha\|v-u_{\mathrm{i}-1}\|^2_{0,\Gamma_{\mathrm{i}-1,\mathrm{i}}} \tag{4.13}$$

for $\mathrm{i} = 1, \cdots, m_0$.

Let u_{i} be the solution of the problem in (4.10) and $u_{\mathrm{i},\mathcal{T}}$ be the DNN approximation. The relative error in the L^2 norm for each block is reported in Tables 4.1-4.5. The network structure is expressed as 2-n_1-n_2-1 for a three-layer network with n_1 and n_2 neurons in the respective first and second layers. The traces of the exact solution and the numerical approximation are depicted in Figures 4.1-4.6 on a plane perpendicular to the space-time plane. Those traces exhibit the capability of the numerical approximation in resolving the shock/rarefaction. Since those planes are generally not perpendicular to the discontinuous interface, the errors shown in those traces are larger than the actual error.

4.5.1 Riemann problem for the inviscid Burgers equation–Shock formation

For the one dimensional inviscid Burgers equation, $\mathbf{f}(u) = f(u) = \frac{1}{2}u^2$, we report numerical results for the corresponding Riemann problem where the initial condition with a single discontinuity is given by:

$$u_0(x) = \begin{cases} u_{_L}, & \text{if } x \le 0, \\ u_{_R}, & \text{if } x > 0. \end{cases} \tag{4.14}$$

When $u_{_L} > u_{_R}$, the characteristic lines intersect and a shock forms immediately for $t > 0$. The weak solution is given by

$$u(x,t) = \begin{cases} u_{_L}, & \text{if } x \le st, \\ u_{_R}, & \text{if } x > st, \end{cases} \tag{4.15}$$

with the shock speed determined by the RH condition

$$s = \frac{f(u_{_L}) - f(u_{_R})}{u_{_L} - u_{_R}} = \frac{f(1) - f(0)}{1 - 0} = 1/2.$$

The first test problem is corresponding to the case

$$u_{_L} = 1 > 0 = u_{_R}$$

with a computational domain $\Omega = (-1, 2) \times (0, 0.6)$. The inflow boundary is

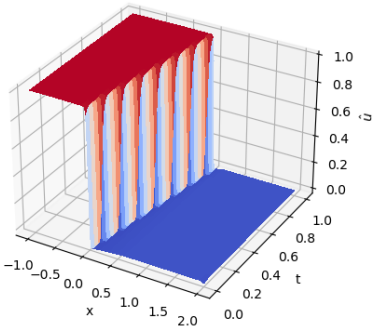
$$\Gamma_- = \Gamma_-^L \cup \Gamma_-^R \equiv \{(-1, t) : t \in [0, 0.6]\} \cup \{(2, t) : t \in [0, 0.6]\}$$

with the boundary conditions: $g = u_{_L}$ on Γ_-^L and $g = u_{_R}$ on Γ_-^R. The block space-time LSNN method is employed with $m_0 = 3$ blocks, a fixed learning rate 0.003, and 30000 iterations for each block.

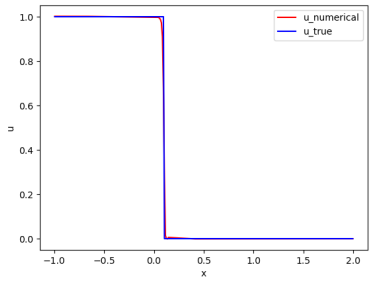
The set of experiments is done by using the numerical fluxes of Roe (4.6) and the second order ENO (4.9). Numerical results of both schemes are reported in Table 4.1 by choosing $\alpha = 20$ in (4.13). Since the results of the ENO flux are similar, only the traces of the exact and numerical solution generated by the Roe flux are depicted on the planes $t = \mathrm{i}T/m_0$ for $\mathrm{i} = 1, \cdots, m_0$ in Figure 4.1 (b)-(f). Clearly, the block space-time LSNN method with a conservative scheme is able to resolve the shock and accurately approximate the solution without the Gibbs phenomena. For this simple Riemann problem, Roe and ENO schemes generate similar results and we do not observe the advantage of using higher-order scheme for flux reconstruction.

Table 4.1. Relative errors of Riemann problem (shock) for Burgers equation using Roe and ENO fluxes

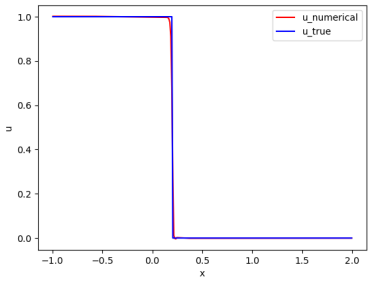
Network structure	Block	Roe flux $\frac{\|u_{\mathrm{i}} - u_{\mathrm{i},\mathcal{T}}\|_0}{\|u_{\mathrm{i}}\|_0}$	ENO flux $\frac{\|u_{\mathrm{i}} - u_{\mathrm{i},\mathcal{T}}\|_0}{\|u_{\mathrm{i}}\|_0}$
2-10-10-1	Ω_1	0.049553	0.050115
2-10-10-1	Ω_2	0.046321	0.051211
2-10-10-1	Ω_3	0.044123	0.049623



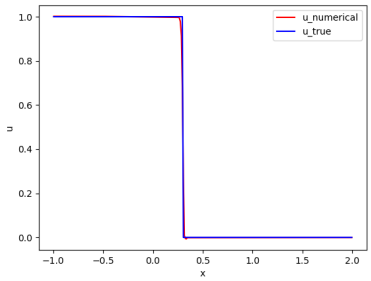
(a) Network approximation $u_{\mathcal{T}}$ on Ω



(b) Traces of exact and numerical solutions $u_{1,\mathcal{T}}$ on the plane $t = 0.2$



(c) Traces of exact and numerical solutions $u_{2,\mathcal{T}}$ on the plane $t = 0.4$



(d) Traces of exact and numerical solutions $u_{3,\mathcal{T}}$ on the plane $t = 0.6$

Figure 4.1. Approximation results of Riemann problem (shock) for Burgers equation using Roe flux

4.5.2 Riemann problem for the inviscid Burgers equation–Rarefaction waves

When $u_{_L} < u_{_R}$ in (4.14), the range of influence of all points in $\mathbb{R}$ is a proper subset of $\mathbb{R} \times [0, \infty)$. This fact implies that the week solution of the scalar hyperbolic conservation law is not unique. To ensure the underlying Cauchy problem having a unique solution over the whole domain $\mathbb{R} \times [0, \infty)$, the so-called vanishing viscosity weak solution is introduced (see, e.g., [14], [44], [45]) and defined as follows:

$$u(x,t) = \begin{cases} u_{_L}, & \text{if} \quad x < u_{_L} t, \\ x/t, & \text{if} \quad u_{_L} t \le x \le u_{_R} t, \\ u_{_R}, & \text{if} \quad x > u_{_R} t. \end{cases}$$

The second test problem is corresponding to the case

$$u_{_L} = 0 < 1 = u_{_R}$$

with a computational domain $\Omega \times I = (-1, 2) \times (0, 0.4)$. The inflow boundary is

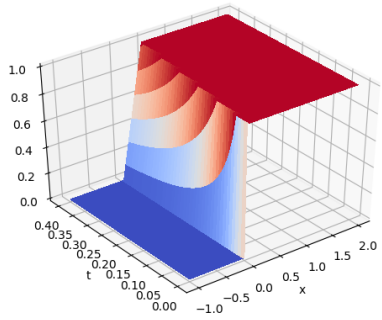
$$\Gamma_- = \Gamma_-^L \cup \Gamma_-^R \equiv \{(-1, t) : t \in [0, 0.4]\} \cup \{(2, t) : t \in [0, 0.4]\}$$

with the boundary conditions: $g = u_{_L}$ on Γ_-^L and $g = u_{_R}$ on Γ_-^R. The block space-time LSNN method is employed with $m_0 = 4$ blocks, a fixed learning rate 0.003, and 20000 iterations for each block.

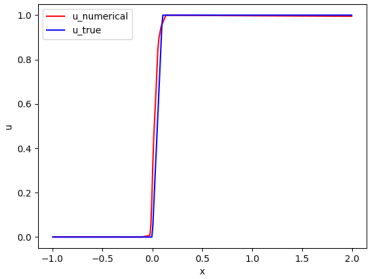
Numerical results of a 2-10-10-1 network using the Roe flux (4.6) are reported in Table 4.2. The traces of the exact and numerical solutions in Fig. 4.2 indicate that Roe's scheme fails to resolve the rarefaction. This is because the scheme approximates the numerical flux depending the sign of the speed a_{i} (4.7) at midpoints. If the sign differs on two sides and $u(x,t)$ travels slower on the left, Roe's scheme may not be able to capture such behavior. From this perspective, we do observe certain limitations of using such conservative scheme in the block space-time LSNN method. In the traditional mesh-based method, "entropy fix" is proposed to address such issue [14], [50].

Table 4.2. Relative errors of Riemann problem (rarefaction) for Burgers equation using Roe flux

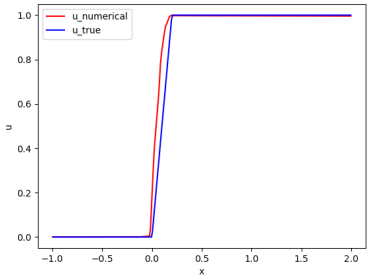
Network structure	Time block	$\frac{\|u_{\mathrm{i}}-u_{\mathrm{i},\mathcal{T}}\|_0}{\|u_{\mathrm{i}}\|_0}$
2-10-10-1	Ω_1	0.047435
2-10-10-1	Ω_2	0.074521
2-10-10-1	Ω_3	0.098679
2-10-10-1	Ω_4	0.122921



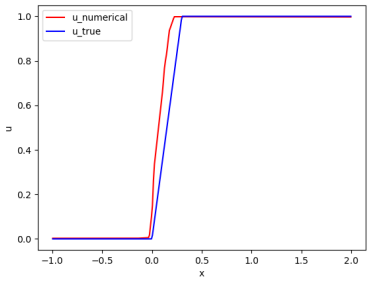
(a) Exact solution u on Ω



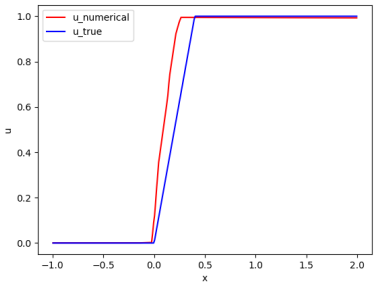
(b) Traces of exact and numerical solutions $u_{1,\mathcal{T}}$ on the plane $t = 0.1$



(c) Traces of exact and numerical solutions $u_{2,\mathcal{T}}$ on the plane $t = 0.2$



(d) Traces of exact and numerical solutions $u_{3,\mathcal{T}}$ on the plane $t = 0.3$



(e) Traces of exact and numerical solutions $u_{4,\mathcal{T}}$ on the plane $t = 0.4$

Figure 4.2. Approximation results of Riemann problem (rarefaction) for Burgers equation using Roe flux

4.5.3 Inviscid Burgers equation with piece-wise linear initial condition

The third test problem is the inviscid Burgers equation defined on the computational domain $\Omega \times I = (-1, 2) \times (0, 1.2)$ with a continuous piece-wise linear initial condition

$$u_0(x) = \begin{cases} 1, & \text{if } x < 0, \\ 1 - x, & \text{if } 0 \le x \le 1, \\ 0, & \text{if } x > 1. \end{cases}$$

The inflow boundary is

$$\Gamma_- = \Gamma_-^L \cup \Gamma_-^R \equiv \{(-1, t) : t \in [0, 1.2]\} \cup \{(2, t) : t \in [0, 1.2]\},$$

with the boundary conditions: $g = 1$ on Γ_-^L and $g = 0$ on Γ_-^R. Even though the initial value u_0 is continuous, the shock will appear at some point since $u(x, t)$ travels faster on the left-hand side than on the right-hand side. Specifically, when $t < 1$, the solution is continuous and it is determined by the characteristic lines as well as the initial conditions:

$$u(x, t) = \begin{cases} 1, & \text{if } x < t < 1, \\ \dfrac{1 - x}{1 - t}, & \text{if } t \le x \le 1, \\ 0, & \text{if } x > 1. \end{cases}$$

When $t > 1$, the shock appears and the desired weak solution satisfying RH condition is given by the following

$$u(x, t) = \begin{cases} 1, & \text{if } x < (t + 1)/2, \\ 0, & \text{if } x \ge (t + 1)/2. \end{cases}$$

The block space-time LSNN using a 2-10-10-1 network is implemented with $m_0 = 6$ blocks, a learning rate which starts with 0.003 and decreased by 0.001 for every 10000 iterations. The total number of iterations for each block is 30000. Additionally, we empirically choose $\alpha = 5$ in (4.13) during the training.

Using the same initialization and quadrature size, numerical results for both the Roe (4.6) and the second order ENO (4.9) fluxes are reported in Table 4.3. For $\mathrm{i} = 1, \cdots, m_0$, the traces of the exact and numerical solutions on the plane $t = \mathrm{i}T/m_0$ for both schemes are depicted in Fig. 4.3 and 4.4, respectively. Clearly, the block space-time LSNN method with a conservative finite difference operator can accurately approximate the solution and resolve the shock. A careful examination of two NN approximations, we observe that the ENO flux performs slightly better than the Roe flux near the left side of the transition layer as well as the relative error in the L^2 norm.

Table 4.3. Relative errors of Burgers equation with a piece-wise linear initial condition

Network structure	Block	Roe flux $\frac{\|u_{\mathrm{i}}-u_{\mathrm{i},\mathcal{T}}\|_0}{\|u_{\mathrm{i}}\|_0}$	ENO flux $\frac{\|u_{\mathrm{i}}-u_{\mathrm{i},\mathcal{T}}\|_0}{\|u_{\mathrm{i}}\|_0}$
2-10-10-1	Ω_1	0.009801	0.007791
2-10-10-1	Ω_2	0.012359	0.012027
2-10-10-1	Ω_3	0.014977	0.012418
2-10-10-1	Ω_4	0.021336	0.013142
2-10-10-1	Ω_5	0.044073	0.047582
2-10-10-1	Ω_6	0.065152	0.066512

4.5.4 Inviscid Burgers equation with smooth initial condition

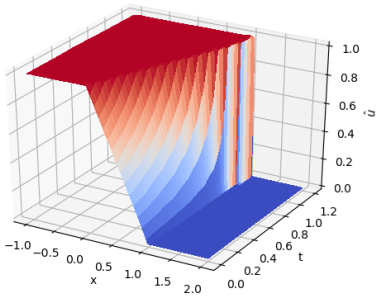
The fourth test problem is again the inviscid Burgers equation defined on the computational domain $\Omega \times I = (0, 2) \times (0, 0.4)$ with the inflow boundary

$$\Gamma_- = \Gamma_-^L \cup \Gamma_-^R \equiv \{(0, t) : t \in [0, 0.4]\} \cup \{(2, t) : t \in [0, 0.4]\}$$

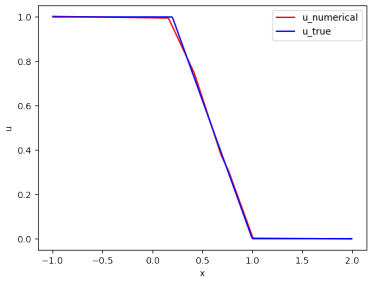
and a sinusoidal initial condition

$$u_0(x) = 0.5 + \sin(\boldsymbol{\pi} x).$$

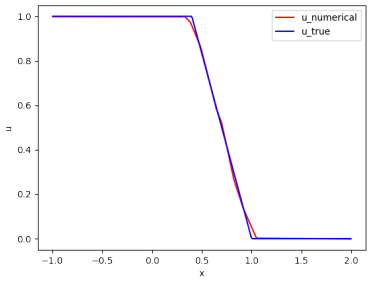
The shock forms at $t = 1/\boldsymbol{\pi} \approx 0.318$. Since the exact solution of the test problem is defined implicitly, in order to measure the quality of the NN approximation, we generate a benchmark reference solution $\hat{u}$ using the traditional mesh-based approach. Specifically,



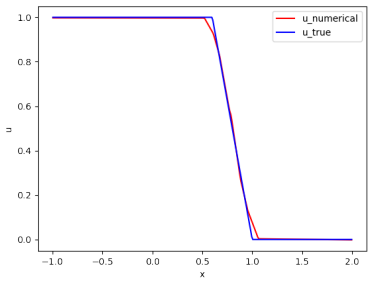
(a) Exact solution u on Ω



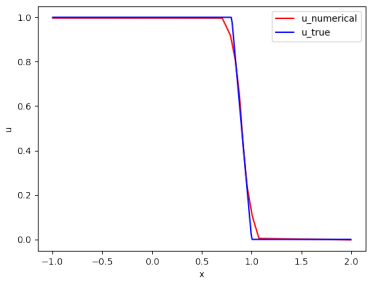
(b) Traces of exact and numerical solutions $u_{1,\mathcal{T}}$ on the plane $t = 0.2$



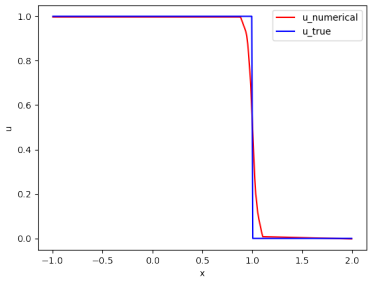
(c) Traces of exact and numerical solutions $u_{2,\mathcal{T}}$ on the plane $t = 0.4$



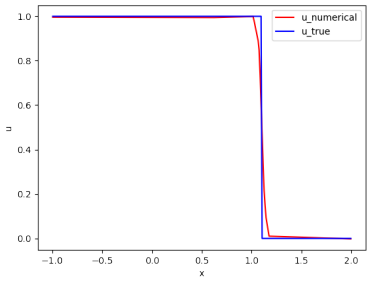
(d) Traces of exact and numerical solutions $u_{3,\mathcal{T}}$ on the plane $t = 0.6$



(e) Traces of exact and numerical solutions $u_{4,\mathcal{T}}$ on the plane $t = 0.8$

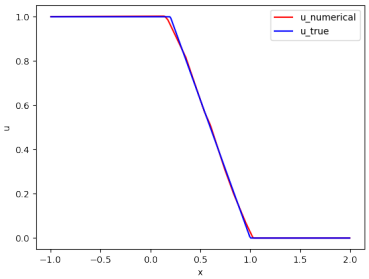


(f) Traces of exact and numerical solutions $u_{5,\mathcal{T}}$ on the plane $t = 1.0$

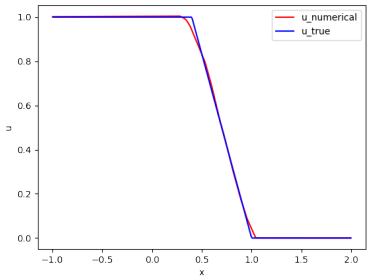


(g) Traces of exact and numerical solutions $u_{6,\mathcal{T}}$ on the plane $t = 1.2$

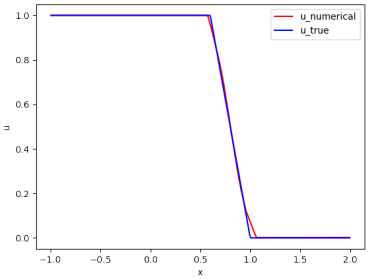
Figure 4.3. Approximation results of Burgers equation with a piece-wise linear initial using Roe flux



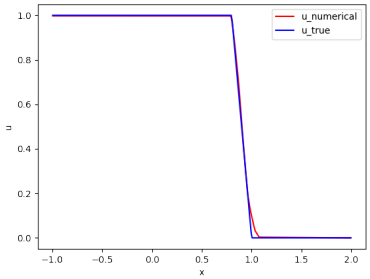
(a) Traces of exact and numerical solutions $u_{1,\mathcal{T}}$ on the plane $t = 0.2$



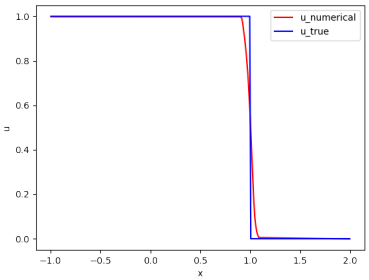
(b) Traces of exact and numerical solutions $u_{2,\mathcal{T}}$ on the plane $t = 0.4$



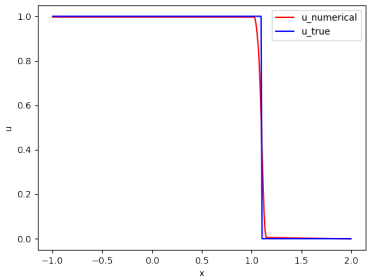
(c) Traces of exact and numerical solutions $u_{3,\mathcal{T}}$ on the plane $t = 0.6$



(d) Traces of exact and numerical solutions $u_{4,\mathcal{T}}$ on the plane $t = 0.8$



(e) Traces of exact and numerical solutions $u_{5,\mathcal{T}}$ on the plane $t = 1.0$



(f) Traces of exact and numerical solutions $u_{6,\mathcal{T}}$ on the plane $t = 1.2$

Figure 4.4. Approximation results of Burgers equation with a piece-wise linear initial using ENO flux

the third order accurate WENO scheme [47] is employed for the spatial discretization with a fine grid ($\Delta x = 0.001$ and $\Delta t = 0.0002$) on the computational domain Ω; and the fourth order Runge-Kutta method is used for the temporal discretization [51]. The block space-time LSNN method is implemented with $m_0 = 8$ blocks and an adaptive learning rate which starts at 0.005 and decays by half for every 25000 iterations. The learning rate decay strategy is employed with 50000 iterations on each time block.

Since the initial condition of the test problem is a smooth function, it is expected that a network with additional neurons is needed for approximation. Choosing $\alpha = 5$ in (4.13), numerical results of a 2-30-30-1 network using the ENO flux are reported in Table 4.4. Figure 4.5 depicts the traces of the reference solution and numerical approximation on the plane $t = \mathrm{i}T/m_0$ for $\mathrm{i} = 1, \cdots, m_0$. We observe some error accumulation when block evolves and the block space-time LSNN method can accurately approximate the solution as well as resolve the shock.

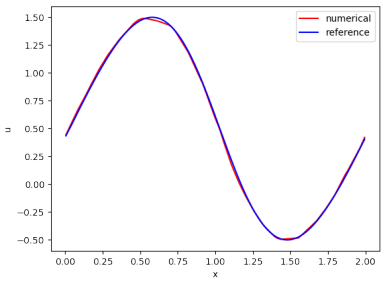
Table 4.4. Relative errors of Burgers equation with a sinusoidal initial condition using ENO flux

Network structure	Time block	$\frac{\|\hat{u}_{\mathrm{i}} - u_{\mathrm{i},\mathcal{T}}\|_0}{\|\hat{u}_{\mathrm{i}}\|_0}$
2-30-30-1	Ω_1	0.010461
2-30-30-1	Ω_2	0.012517
2-30-30-1	Ω_3	0.019772
2-30-30-1	Ω_4	0.022574
2-30-30-1	Ω_5	0.029011
2-30-30-1	Ω_6	0.038852
2-30-30-1	Ω_7	0.075888
2-30-30-1	Ω_8	0.078581

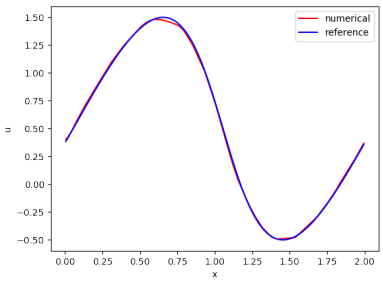
4.5.5 Riemann problem with a convex flux

The last numerical experiment is the Riemann shock problem with a convex flux $\mathbf{f}(u) = f(u) = \frac{1}{4}u^4$. We choose the initial condition

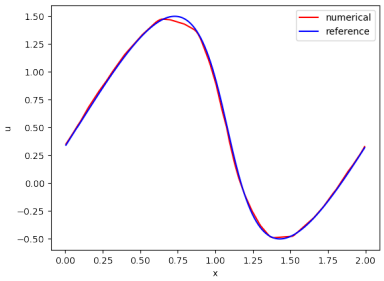
$$u_{_L} = 1 > 0 = u_{_R}$$



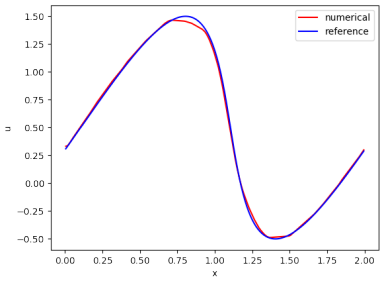
(a) Traces on the plane $t = 0.05$



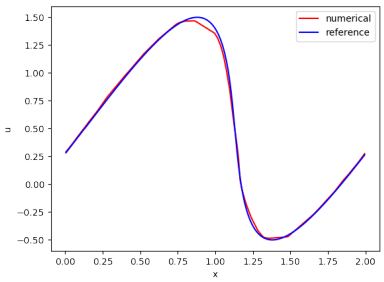
(b) Traces on the plane $t = 0.1$



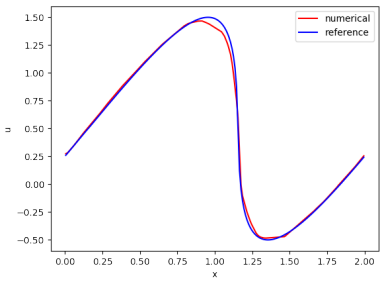
(c) Traces on the plane $t = 0.15$



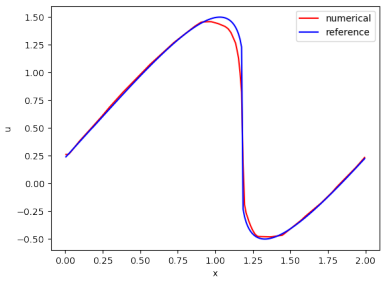
(d) Traces on the plane $t = 0.2$



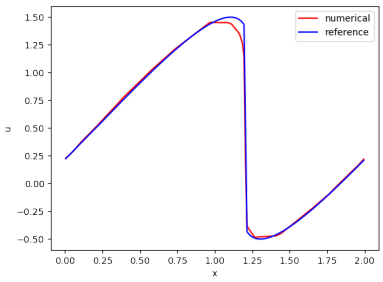
(e) Traces on the plane $t = 0.25$



(f) Traces on the plane $t = 0.3$



(g) Traces on the plane $t = 0.35$



(h) Traces on the plane $t = 0.4$

Figure 4.5. Approximation results of Burgers equation with a sinusoidal initial using ENO flux

in (4.14), then the weak solution is given by (4.15) with the speed $s = 1/4$. The computational domain of the problem is given by $\Omega \times I = (-1, 1) \times (0, 0.6)$ and the inflow boundary is

$$\Gamma_- = \Gamma_-^L \cup \Gamma_-^R \equiv \{(-1, t) : t \in [0, 0.6]\} \cup \{(1, t) : t \in [0, 0.6]\}$$

with the boundary conditions: $g = 1$ on Γ_-^L and $g = 0$ on Γ_-^R.

Empirically, we choose $\alpha = 20$ in (4.13) in the training. Employing the block space-time LSNN method with $m_0 = 3$ blocks, a fixed learning rate 0.003 and 30000 iterations for each block, we report the numerical results of a 2-10-10-1 network in Table 4.5 and Fig. 4.6 using Roe flux. Again, the results imply that the discontinuous interface can be accurately captured as the NN approximation is almost overlapped with the exact solution.

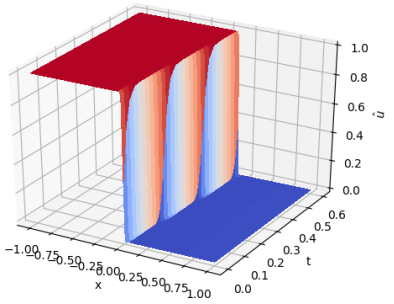
Table 4.5. Relative errors of Riemann problem (shock) with a convex flux using Roe flux

Network structure	Block	$\frac{\|u_{\mathrm{i}} - u_{\mathrm{i},\mathcal{T}}\|_0}{\|u_{\mathrm{i}}\|_0}$
2-10-10-1	Ω_1	0.035034
2-10-10-1	Ω_2	0.036645
2-10-10-1	Ω_3	0.036798

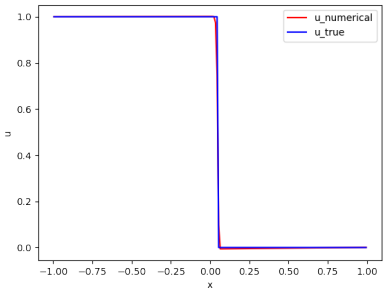
4.6 Discussion

In this chapter, we proposed the block space-time LSNN method for solving scalar nonlinear hyperbolic conservation laws. Similar to the linear advection-reaction problem, the least-squares formulation is a direct application of the least-squares principle to the underlying problem: the equation, the inflow boundary condition, and the initial condition. The block version of the LSNN method is introduced to compensate with some uncertainty of the not well-understood non-convex optimization procedure.

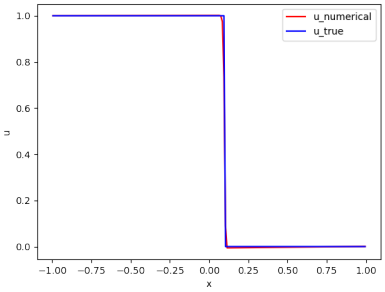
How to approximate the differential operator in the least-squares functional is critical for the success of the block space-time LSNN method. Employing conservative finite difference operators, for the benchmark one dimensional problems, we show numerically that the block space-time LSNN method is capable of accurately approximating the solution and resolving



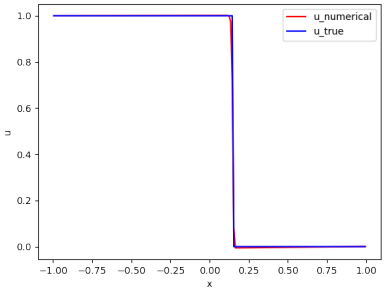
(a) Network approximation $u_{\mathcal{T}}$ on Ω



(b) Traces of exact and numerical solutions $u_{1,\mathcal{T}}$ on the plane $t = 0.2$



(c) Traces of exact and numerical solutions $u_{2,\mathcal{T}}$ on the plane $t = 0.4$



(d) Traces of exact and numerical solutions $u_{3,\mathcal{T}}$ on the plane $t = 0.6$

Figure 4.6. Approximation results of Riemann problem (shock) with a convex flux using Roe flux

the shock without oscillation. However, we do observe that Roe's scheme has a limitation for the rarefaction as indicated by the numerical experiments in section 4.5.2.

5. CONCLUSIONS, LIMITATIONS AND FUTURE WORK

In this thesis, we proposed the least-squares neural network (LSNN) method for solving scalar hyperbolic conservation laws. The least-squares formulation is based on a direct application of the least-squares principle to the underlying problem.

As pointed out earlier, the major challenge in numerical simulation of hyperbolic conservation law lies in the discontinuity of the solution. The numerical results suggest that the LSNN method is capable of resolving the discontinuity by using a small number of neurons. Given these facts and the comparisons presented in the thesis (see section 3.5.1 and section 3.5.2), the LSNN method outperforms the traditional mesh-based method in terms of the number of degree of freedoms. Moreover, we also theoretically show that a three-layer ReLU neural network has the capability to accurately approximate the discontinuous solution without the common Gibbs phenomena, and the theory is confirmed by the numerical results in section 3.5. In addition, as emphasized in the thesis that how to discretize the differential operators in the least-squares functional is fairly critical for the success of the LSNN method. For the linear problems, section 3.5.2 presents the results produced by two discretization approaches and only the one using the directional derivative is able to resolve the discontinuous interface. For the nonlinear problems, Roe's scheme has a limitation to resolve the rarefaction (see section 4.5.2).

However, on the other hand, the powerful approximation property of DNN comes with a price. Now, determining the DNN parameters values through a iterative process becomes a non-convex optimization even though the underlying PDE is linear. Since we have limited understanding about this field, we proposed the initialization strategy for two-layer network, a method of model continuation and block space-time LSNN method as compensations for training difficulties. Nevertheless, this is still a challenging problem in the implementations since the learning rate of the methods of the gradient type is difficult to tune. A reasonably good learning rate can only be discovered through the method of trial and error. Without a appropriate learning rate, the iterative solver may be trapped at a local minimum which does not produce a good approximation. This may explain why an "over-parametrized" network

is often used in practice. Study on the non-convex optimization algorithm could be a future research direction.

REFERENCES

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

[2] Z. Cai, J. Chen, M. Liu, and X. Liu, "Deep least-squares methods: An unsupervised learning-based numerical method for solving elliptic PDEs," *Journal of Computational Physics*, vol. 420, p. 109 707, 2020.

[3] T. Dockhorn, "A discussion on solving partial differential equations using neural networks," *CoRR*, vol. abs/1904.07200, 2019. arXiv: 1904.07200. [Online]. Available: http://arxiv.org/abs/1904.07200.

[4] W. E and B. Yu, "The deep ritz method: A deep learning-based numerical algorithm for solving variational problems," *Communications in Mathematics and Statistics*, vol. 6, no. 1, Mar. 2018.

[5] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1139–1364, 2018.

[6] M. W. M. G. Dissanayake and N. Phan-Thien, "Neural network based approximations for solving partial differential equations," *Communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 1994.

[7] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[8] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[9] K. S. McFall and J. R. Mahan, "Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1221–1233, 2009.

[10] M. Liu, Z. Cai, and J. Chen, "Adaptive two-layer ReLU neural network," *submitted*, 2020.

[11] J. S. Hesthaven and T. Warburton, *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[12] D. Gottlieb and C.-W. Shu, "On the gibbs phenomenon and its resolution," *SIAM review*, vol. 39, no. 4, pp. 644–668, 1997.

[13] J. S. Hesthaven, *Numerical methods for conservation laws: From analysis to algorithms*. SIAM, 2017.

[14] R. J. LeVeque and R. J. Leveque, *Numerical methods for conservation laws*. Springer, 1992, vol. 3.

[15] P. L. Roe, "Approximate riemann solvers, parameter vectors, and difference schemes," *Journal of computational physics*, vol. 43, no. 2, pp. 357–372, 1981.

[16] F. Brezzi, L. D. Marini, and E. Süli, "Discontinuous galerkin methods for first-order hyperbolic problems," *Mathematical models and methods in applied sciences*, vol. 14, no. 12, pp. 1893–1903, 2004.

[17] W. Dahmen, C. Huang, C. Schwab, and G. Welper, "Adaptive petrov–galerkin methods for first order transport equations," *SIAM journal on numerical analysis*, vol. 50, no. 5, pp. 2420–2445, 2012.

[18] L. Demkowicz and J. Gopalakrishnan, "A class of discontinuous petrov–galerkin methods. part i: The transport equation," *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 23-24, pp. 1558–1572, 2010.

[19] E. Burman, "A posteriori error estimation for interior penalty finite element approximations of the advection-reaction equation," *SIAM journal on numerical analysis*, vol. 47, no. 5, pp. 3584–3607, 2009.

[20] P. Houston, J. A. Mackenzie, E. Süli, and G. Warnecke, "A posteriori error analysis for numerical approximations of friedrichs systems," *Numerische Mathematik*, vol. 82, no. 3, pp. 433–470, 1999.

[21] P. Houston, R. Rannacher, and E. Süli, "A posteriori error analysis for stabilised finite element approximations of transport problems," *Computer methods in applied mechanics and engineering*, vol. 190, no. 11-12, pp. 1483–1508, 2000.

[22] Q. Liu and S. Zhang, "Adaptive least-squares finite element methods for linear transport equations based on an H(div) flux reformulation," *Computer Methods in Applied Mechanics and Engineering*, vol. 366, p. 113 041, 2020.

[23] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," in *International Conference on Representation Learning, Vancouver, BC, Canada*, 2018.

[24] J. Tarela and M. Martinez, "Region configurations for realizability of lattice piecewise-linear models," *Mathematical and Computer Modelling*, vol. 30, no. 11-12, pp. 17–27, 1999.

[25] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, "Nonlinear approximation and (deep) ReLU networks," *arXiv preprint arXiv:1905.02199*, 2019.

[26] D. Yarotsky, "Error bounds for approximations with deep ReLU networks," *Neural Networks*, vol. 94, pp. 103–114, 2017.

[27] D. Yarotsky, "Optimal approximation of continuous functions by very deep ReLU networks," in *Conference on Learning Theory*, PMLR, 2018, pp. 639–649.

[28] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[29] Z. Cai, J. Chen, and M. Liu, "Least-squares ReLU neural network method for linear advection-reaction equation," *submitted*, 2021.

[30] P. Bochev and J. Choi, "Improved least-squares error estimates for scalar hyperbolic problems," *Computational Methods in Applied Mathematics*, vol. 1, no. 2, pp. 115–124, 2001.

[31] P. Bochev and M. Gunzburger, "Least-squares methods for hyperbolic problems," in *Handbook of Numerical Analysis*, vol. 17, Elsevier, 2016, pp. 289–317.

[32] P. B. Bochev and J. Choi, "A comparative study of least-squares, supg and galerkin methods for convection problems," *International Journal of Computational Fluid Dynamics*, vol. 15, no. 2, pp. 127–146, 2001.

[33] G. F. Carey and B. Jianng, "Least-squares finite elements for first-order hyperbolic systems," *International journal for numerical methods in engineering*, vol. 26, no. 1, pp. 81–93, 1988.

[34] 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*, vol. 26, no. 1, pp. 31–54, 2004.

[35] 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 Journal on Scientific Computing*, vol. 26, no. 5, pp. 1573–1597, 2005.

[36] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta numerica*, vol. 8, no. 1, pp. 143–195, 1999.

[37] Z. Shen, H. Yang, and S. Zhang, "Deep network approximation characterized by number of neurons," *arXiv preprint arXiv:1906.05497*, 2019.

[38] J. W. Siegel and J. Xu, "Approximation rates for neural networks with general activation functions," *Neural Networks*, 2020.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Representation Learning, San Diego*, 2015.

[40] L. Mu and X. Ye, "A simple finite element method for linear hyperbolic problems," *Journal of Computational and Applied Mathematics*, vol. 330, pp. 330–339, 2018.

[41] J. He, L. Li, J. Xu, and C. Zheng, "ReLU deep neural networks and linear finite elements," *arXiv preprint arXiv:1807.03973*, 2018.

[42] S. Wang and X. Sun, "Generalization of hinging hyperplanes," *IEEE Transactions on Information Theory*, vol. 51, no. 12, pp. 4425–4431, 2005.

[43] Z. Cai, J. Chen, and M. Liu, "Least-squares neural network method for scalar nonlinear hyperbolic conservation law," *submitted*, 2021.

[44] E. Godlewski and P.-A. Raviart, *Numerical approximation of hyperbolic systems of conservation laws*. Springer Science & Business Media, 2013, vol. 118.

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

[46] D. Z. Kalchev and T. A. Manteuffel, "A least-squares finite element method based on the helmholtz decomposition for hyperbolic balance laws," *arXiv preprint arXiv:1911.05831v2*, 2020.

[47] C.-W. Shu, "Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws," in *Advanced numerical approximation of nonlinear hyperbolic equations*, Springer, 1998, pp. 325–432.

[48] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of computational physics*, vol. 77, no. 2, pp. 439–471, 1988.

[49] A. Harten, B. Engquist, S. Osher, and S. R. Chakravarthy, "Uniformly high order accurate essentially non-oscillatory schemes, iii," in *Upwind and high-resolution schemes*, Springer, 1987, pp. 218–290.

[50] A. Harten, "High resolution schemes for hyperbolic conservation laws," *Journal of computational physics*, vol. 135, no. 2, pp. 260–278, 1997.

[51] Y. Wang, Z. Shen, Z. Long, and B. Dong, "Learning to discretize: Solving 1d scalar conservation laws via deep reinforcement learning," *arXiv preprint arXiv:1905.11079*, 2019.

VITA

Jingshuang Chen is a Ph.D candidate in the Department of Mathematics, Purdue University. She obtained her Bachelor's degree in Applied Mathematics from Huazhong University of Science and Technology in June 2015 and became a graduate student of Purdue University in Aug 2015. During her Ph.D, she did an internship in Disney research in 2019 summer and focused on cloth simulation project. In 2020 summer, she did a remote internship in Microsoft as a data scientist and used statistical tool to analyze the impact of security features.