Journal of Computational Physics 367 (2018) 166–191

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

An artificial neural network as a troubled-cell indicator

Deep Ray *, Jan S. Hesthaven

École Polytechnique Fédérale de Lausanne (EPFL), CH-1015 Lausanne, Switzerland

ARTICLE INFO

Article history:
Received 14 November 2017
Received in revised form 1 March 2018
Accepted 13 April 2018
Available online 19 April 2018

Keywords:
Conservation laws
Discontinuous Galerkin
Limiting
Troubled-cell indicator
Artificial neural network

ABSTRACT

High-resolution schemes for conservation laws need to suitably limit the numerical solution near discontinuities, in order to avoid Gibbs oscillations. The solution quality and the computational cost of such schemes strongly depend on their ability to correctly identify troubled-cells, namely, cells where the solution loses regularity. Motivated by the objective to construct a *universal troubled-cell indicator* that can be used for general conservation laws, we propose a new approach to detect discontinuities using artificial neural networks (ANNs). In particular, we construct a multilayer perceptron (MLP), which is trained offline using a supervised learning strategy, and thereafter used as a black-box to identify troubled-cells. The proposed MLP indicator can accurately identify smooth extrema and is independent of problem-dependent parameters, which gives it an advantage over traditional limiter-based indicators. Several numerical results are presented to demonstrate the robustness of the MLP indicator in the framework of Runge–Kutta discontinuous Galerkin schemes, and its performance is compared with the minmod limiter and the minmod-based TVB limiter.

1. Introduction

It is well known that solutions to conservation laws often develop discontinuities in finite time, even for smooth initial data [1]. Thus, numerical methods need to be carefully corrected near discontinuities to avoid spurious Gibbs oscillations. The approach used to handle discontinuities numerically can be broken into two key steps. The first step involves detecting the (mesh) cells where the solution loses regularity. Such cells are termed as *troubled-cells*. In the second step, the solution in these cells are corrected to avoid spurious oscillations. Several correction techniques are available for use in numerical schemes, such as slope limiting [2–4], adaptively choosing the stencil for reconstruction [5–7], or adding artificial diffusion [8–10]. For most numerical methods, the two stages are usually combined into a single step. However, it is useful to consider them separately for Runge–Kutta discontinuous Galerkin (RKDG) schemes [11].

The correction (or limiting) of the numerical solution in troubled-cells can be computationally expensive. To ensure cost-efficiency of numerical schemes, it is essential to use troubled-cell indicators that only flag the *genuine* troubled-cells. In [12], a thorough numerical study was performed to assess the performance of various limiter-based troubled-cell indicators for RKDG schemes. It was observed that the classical minmod limiter flags more cells than necessary, including cells with smooth extrema, which can lead to an unnecessary increase in computational cost. As an alternative, the minmod-type TVB limiter [13] seeks to correctly identify troubled-cells, provided its problem-dependent parameter M is chosen appropriately. In general, however, it is difficult to estimate M a priori. Thus, there is a need to construct a troubled-cell

* Corresponding author.
E-mail addresses: deep.ray@epfl.ch (D. Ray), jan.hesthaven@epfl.ch (J.S. Hesthaven).

https://doi.org/10.1016/j.jcp.2018.04.029
0021-9991/

indicator that flags genuine troubled-cells, and is independent of problem-dependent parameters. Recently, an automatic parameter selection strategy for troubled-cell indicators was proposed in [14], which is based on Tukey's boxplot method of outlier-detection [15]. This approach has also been tested with compact-WENO finite difference methods in [16]. The outlier-detection algorithm requires the input solution vector to be sorted, which in general scales as $\mathcal{O}(N \log N)$ on a mesh with N elements. A new indicator was proposed for RKDG schemes in [17], which is shown to perform well with the Euler equations in one- and two-dimensions, provided an empirically determined parameter is chosen. However, this parameter only seems to depend on the degree of the approximating polynomial. In the present paper, we propose a new type of indicator by constructing an *artificial neural network* (ANN) that serves as a troubled-cell indicator, without requiring the prescription of any problem-dependent parameters.

In principle, ANNs can be seen as function approximators capable of capturing a high-degree of complexity and non-linearity. The design of ANNs is based on the architecture of their biological counterpart, and have the capacity to *learn* [18]. Once suitably trained on a given dataset, ANNs are able to recover key features of the underlying model, and accurately predict the output for data points lying outside the training dataset. Although the training of the network can be computationally-intensive and time consuming, it is done *offline* and only once for a given application. Thereafter, the trained network is used as a black-box, with the involved computations being inexpensive. For this reason, ANNs are popular in applications such as image processing [19] and speech recognition [20], as a substitute for complex rule-based algorithms which are often difficult to program. ANNs have also been used to solve certain classes of ordinary and partial differential equations [21–23].

In this paper, we consider a specific type of ANN, know as a *multilayer perceptron* (MLP), which consists of neurons stacked in a series of layers. Under mild assumptions on the network design, it has been shown that MLPs with one or two *hidden layers*, i.e., layers between the first and the last, can approximate any continuous function [24,25]. The approximation capabilities of simple MLP networks have been studied quite extensively (see [26] and references therein). However, rigorous results for more general and complex networks remain elusive.

We propose a *deep* MLP network, which is trained to inherit the properties of a troubled-cell indicator. The training is performed offline on a robust dataset, and the final network is used within the framework of RKDG schemes. The network is independent of any problem-dependent parameter, thus making it attractive as a universal troubled-cell indicator for general conservation laws.

The rest of the paper is structured as follows. In Section 2 we introduce the RKDG formulation, followed by a brief discussion of a few existing troubled-cell indicators. We motivate the construction of MLPs in Section 3, and give an overview of the key ingredients required to construct and train such networks. In Section 4, we present the construction of an MLP based troubled-cell indicator. Several numerical results are presented in Section 5 to demonstrate the capability of the proposed network, as compared to existing limiter-based troubled-cell indicators. We make a few concluding remarks in Section 6.

2. Discontinuous-Galerkin formulation

We consider the following one-dimensional scalar conservation law

$$\begin{aligned} &\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \qquad \forall\, (x,t) \in [a,b] \times [0,T], \\ &u(x,0) = u_0(x) \qquad \forall\, x \in [a,b], \end{aligned} \tag{2.1}$$

where u is the conserved variable with a smooth flux $f(u)$. To approximate the solution of (2.1), the computational domain is discretized using N non-overlapping cells, with cell-interfaces $a = x_{\frac{1}{2}} < x_{\frac{3}{2}} < \ldots < x_{N+\frac{1}{2}} = b$. The center of cell $I_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ is denoted by $x_i = (x_{i-\frac{1}{2}} + x_{i+\frac{1}{2}})/2$. For the rest of this paper, we assume the discretization to be uniform, with the mesh size denoted by $h = x_{i+\frac{1}{2}} - x_{i-\frac{1}{2}}$. We define the space of broken polynomials $V_r^h = \{v \in L^2([a,b]) : v\big|_{I_i} \in P^r(I_i)\}$, where $P^r(I_i)$ is the space of polynomials with degree $d \leq r$ on the cell I_i. The semi-discrete DG scheme can be formulated as follows.

Definition 2.1. Find $u_h(.,t) \in V_r^h$ such that the following semi-discrete relation is satisfied for all $v_h \in V_r^h$,

$$\int_{I_i} \left[\frac{\partial u_h}{\partial t} v_h - f(u_h) \frac{\mathrm{d}v_h}{\mathrm{d}x} \right] \mathrm{d}x + \hat{f}_{i+\frac{1}{2}}(t) v_h(x^-_{i+\frac{1}{2}}) - \hat{f}_{i-\frac{1}{2}}(t) v_h(x^+_{i-\frac{1}{2}}) = 0, \tag{2.2}$$

where $v_h(x^{\pm}) = \lim_{\epsilon \downarrow 0} v_h(x \pm \epsilon)$ and $\hat{f}_{i+\frac{1}{2}}(t) = \hat{f}\big(u_h(x^-_{i+\frac{1}{2}}, t), u_h(x^+_{i+\frac{1}{2}}, t)\big)$ is a consistent numerical flux.

In practice, the solution in each cell I_i is represented using a suitable local basis $\{\phi_{ij}(x),\ \ j = 0, \ldots, r\}$ as

$$u_h(x,t) = \sum_{j=0}^{r} u_{ij}(t) \phi_{ij}(x) \quad \forall\, x \in I_i,$$

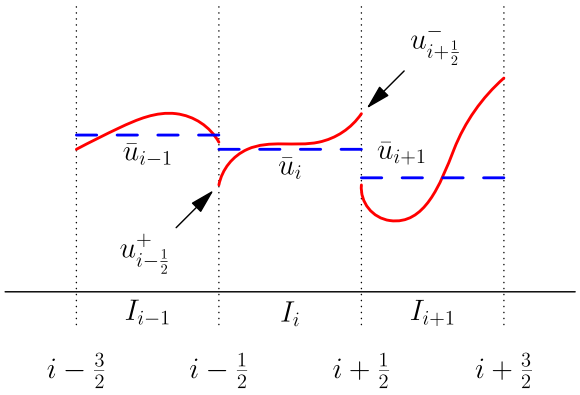


Fig. 1. Stencil used by troubled-cell indicators. The polynomial approximation in each cell is denoted by solid line, while the corresponding cell-average values denoted by dashed lines.

where the coefficients $u_{i0}(t), \ldots u_{ir}(t)$ are the degrees of freedom to be determined using the numerical scheme. Defining the vector $\mathbf{U}_i = (u_{i0}, \ldots, u_{ir})^\top$ and taking $v_h = \phi_{ij}$ gives us

$$\int_{I_i} \frac{\partial u_h}{\partial t}\phi_{ij}\mathrm{d}x = \sum_{k=0}^{r} \frac{\mathrm{d}u_{ik}}{\mathrm{d}t}\int_{I_i} \phi_{ik}\phi_{ij}\mathrm{d}x = \sum_{k=0}^{r} \mathbf{M}^{(i)}_{jk}\frac{\mathrm{d}u_{ik}}{\mathrm{d}t}, \tag{2.3}$$

where $\mathbf{M}^{(i)}_{jk}$ are the elements of the *mass-matrix* $\mathbf{M}^{(i)}$, corresponding to cell I_i. Note that the mass-matrix can be computed using a quadrature rule which is exact for polynomials of degree $2r$. The remaining terms of the (2.2) can be approximated as

$$\begin{aligned}\mathbf{R}^{(i)}(\mathbf{U}(t))_j &= \int_{I_i} f(u_h)\frac{\mathrm{d}\phi_{ij}}{\mathrm{d}x}\mathrm{d}x - \hat{f}_{i+\frac{1}{2}}(t)\phi_{ij}(x^-_{i+\frac{1}{2}}) + \hat{f}_{i-\frac{1}{2}}(t)\phi_{ij}(x^+_{i-\frac{1}{2}}) \\ &\approx \sum_q w_{iq} f\left(u_h(x_{iq},t)\right)\frac{\mathrm{d}\phi_{ij}}{\mathrm{d}x}(x_{iq})\mathrm{d}x - \hat{f}_{i+\frac{1}{2}}(t)\phi_{ij}(x^-_{i+\frac{1}{2}}) + \hat{f}_{i-\frac{1}{2}}(t)\phi_{ij}(x^+_{i-\frac{1}{2}}),\end{aligned} \tag{2.4}$$

where x_{iq} and w_{iq} are the nodes and weights, respectively, for a suitable q-point quadrature. Using (2.3), (2.4) and the fact that the mass-matrix is invertible, we obtain the following system of ordinary differential equations corresponding to cell I_i

$$\frac{\mathrm{d}\mathbf{U}_i}{\mathrm{d}t} = \left(\mathbf{M}^{(i)}\right)^{-1}\mathbf{R}^{(i)}(\mathbf{U}(t)),$$

which is solved using a suitable time-marching scheme, such as the third-order strong stability preserving Runge–Kutta (SSP-RK3) scheme [27].

While the scheme described above is capable of approximating smooth solutions with a high-degree of accuracy, it suffers from Gibbs oscillations near discontinuities. A commonly used technique to mitigate this issue is by correcting the approximating polynomial in troubled-cells after each Runge–Kutta stage. This procedure comprises two steps: i) detection of troubled-cells, and ii) a suitable limited reconstruction of the polynomial solution in the troubled-cells.

In general, troubled-cell indicators analyze the smoothness of the data in the cell I_i based on the information extracted from a 3-cell compact stencil centered at i, as shown in Fig. 1. In each cell, the solution is approximated by a polynomial (solid lines), with corresponding cell-averaged values $\overline{u}_{i-1}, \overline{u}_i, \overline{u}_{i+1}$ (dashed lines). In addition, we require the left and the right cell-interface values of the polynomial in I_i, i.e., $u^+_{i-\frac{1}{2}}$ and $u^-_{i+\frac{1}{2}}$. Using these five quantities, we construct the following forward and backward differences,

$$\Delta^- u_i = \overline{u}_i - \overline{u}_{i-1}, \qquad \Delta^+ u_i = \overline{u}_{i+1} - \overline{u}_i, \qquad \check{u}_i = \overline{u}_i - u^+_{i-\frac{1}{2}}, \qquad \hat{u}_i = u^-_{i+\frac{1}{2}} - \overline{u}_i. \tag{2.5}$$

The class of troubled-cell indicators considered in this paper, use the differences (2.5) to modify the cell-interface values as

$$\widetilde{u}^+_{i-\frac{1}{2}} = \overline{u}_i + \mathcal{M}\left(\check{u}_i, \Delta^- u_i, \Delta^+ u_i; Z\right), \qquad \widetilde{u}^-_{i+\frac{1}{2}} = \overline{u}_i - \mathcal{M}\left(\hat{u}_i, \Delta^- u_i, \Delta^+ u_i; Z\right), \tag{2.6}$$

where $\mathcal{M}$ is a slope-limiter and Z denotes additional parameters that the slope-limiter may depend upon. The cell I_i is marked as a troubled-cell, if the modifications in (2.6) changes either of the two cell-interface values, i.e., if $\widetilde{u}^+_{i-\frac{1}{2}} \neq u^+_{i-\frac{1}{2}}$ or

$\widetilde{u}^-_{i+\frac{1}{2}} \neq u^-_{i+\frac{1}{2}}$. A thorough numerical comparison of various limiter-based indicator functions has been performed in [12]. We consider the following two commonly used limiters:

1. *The minmod limiter:* This slope-limiter modifies the cell-interface values to ensure that the solution is total variation diminishing in the mean (TVDM) [8]. It is given by

$$\mathcal{M}^{mm}(a,b,c) = \begin{cases} \text{sign}(a)\min(|a|,|b|,|c|) & \text{if } \text{sign}(a) = \text{sign}(b) = \text{sign}(c), \\ 0 & \text{otherwise.} \end{cases} \tag{2.7}$$

   A disadvantage with the minmod limiter is that it also flags cells containing a smooth extrema, which can lead to an increased computational cost, and limits local accuracy to first-order.
2. *The minmod-type TVB limiter:* In order to overcome the problems of the minmod limiter, one can relax the TVDM condition by requiring the solution to be total variation bounded (TVB). This can be achieved by the following modified slope-limiter

$$\mathcal{M}^{tvb}(a,b,c;h,M) = \begin{cases} a & \text{if } |a| \leq Mh^2, \\ \mathcal{M}^{mm}(a,b,c) & \text{otherwise,} \end{cases} \tag{2.8}$$

   where the limiter now depends on the local mesh-size h and a problem-dependent parameter M. For scalar conservation laws, M is proportional to the curvature of the initial condition near smooth extrema [13]. However, it is difficult to estimate M for a general system of conservation laws. If M is chosen too small, (2.8) essentially reduces to the minmod limiter (2.7). On the other hand, if M is chosen too large, the indicator does not flag all the troubled-cells, leading to the re-appearance of Gibbs oscillations.

Based on the two limiters discussed above, we seek a troubled-cell indicator which i) does not flag cells with smooth extrema, and ii) is independent of problem specific parameters. To accommodate both these traits, we propose a new approach to the problem of discontinuity detection, via an artificial neural network (see Section 3).

For the reconstruction step, we can replace the polynomial in the troubled-cells by the cell-average or a limited linear polynomial [28,8]. This leads to a loss in accuracy if cells are incorrectly flagged by the indicator. To overcome this issue, one can use WENO limiters to rebuild the polynomial in flagged cells by extending the polynomial in neighboring cells [11]. However, finding the WENO weights is computationally expensive, re-emphasizing the need to not flag more cells than necessary as troubled-cells.

Since the focus of this paper will be on the constructing a suitable troubled-cell indicator, we use the classical MUSCL [2,8] reconstruction procedure for the second step. Specifically, if the unlimited polynomial approximation in a troubled-cell I_i is written as

$$u_h(x) = \overline{u}_i + (x - x_i)s_i + \mathcal{O}\big((x - x_i)^2\big),$$

then the limited linear polynomial is given by

$$\tilde{u}_h(x) = \overline{u}_i + (x - x_i)\mathcal{M}\left(s_i, \frac{\overline{u}_i - \overline{u}_{i-1}}{h}, \frac{\overline{u}_{i+1} - \overline{u}_i}{h}\right),$$

where $\mathcal{M}$ is some slope-limiter.

3. Artificial neural networks

Our goal is to find a suitable approximation to an unknown function

$$\mathbf{G} : \Omega \in \mathbb{R}^{N_I} \mapsto \mathbb{R}^{N_O}, \quad \text{given} \quad \mathbb{T} = \{(\mathbf{X}_p, \mathbf{Y}_p) \mid \mathbf{Y}_p = \mathbf{G}(\mathbf{X}_p)\ \forall\ p \in \Lambda\}. \tag{3.1}$$

For our problem, this corresponds to the underlying *true* indicator function that correctly flags the genuine troubled-cells. Simple approaches such as linear-regression or least-squares polynomial fitting, are unsuitable if $\mathbf{G}$ is highly non-linear. In fact, we have no information about the qualitative features of the troubled-cell indicator. Thus, it is fruitful to look for an approximation which is capable of *learning* these unknown features based on the dataset $\mathbb{T}$. This is precisely what we hope to accomplish using artificial neural networks, inspired by biological networks.

The biological nervous system in vertebrates is responsible for transmitting information through the organism, facilitating the coordination between different body parts. A specialized nerve cell, known as the neuron, forms the fundamental building block of the nervous system. As shown in Fig. 2, a simplified model of the neuron consists of three main components, namely the *dentrites*, the *nucleus* and the *axon*. The dentrites are nerve fibers through which the neuron receives signals from several input neurons, with the connection between two neurons being termed as a *synapsis*. The signal from the input neuron is pre-processed in the synapsis, before being transmitted to the receiving neuron. Thus, the synapsis

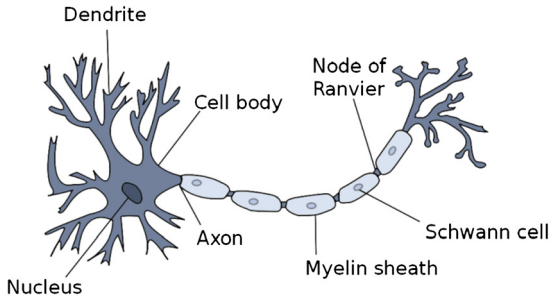


Fig. 2. A simplified model of the biological neuron. This image has been taken from [29].

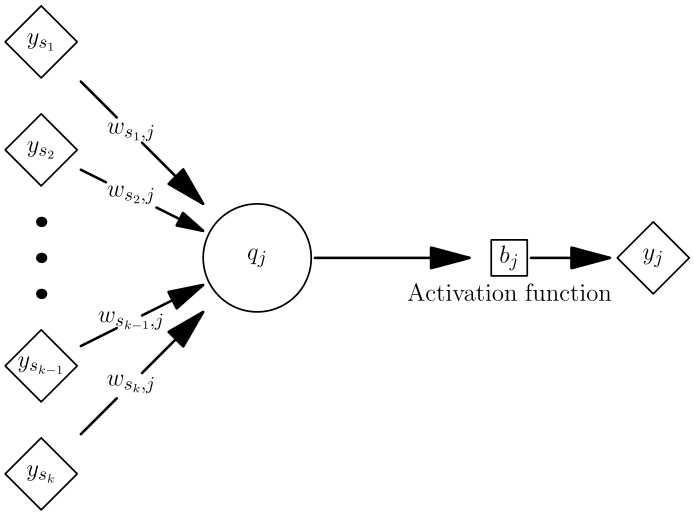


Fig. 3. A single artificial neuron receiving signals from k neurons.

can be seen as a *weighted* connection. The weighted accumulation of the signals received by the neuron is stored in the nucleus. Once the accumulation crosses a certain threshold, the nucleus fires an electrical signal through the axon to other connecting neurons. A complex network of neurons is involved in passing electrical impulses throughout the body [29]. Various studies have shown that *knowledge* is gained through a *training* process, in which synaptic connections are created or modified when exposed to different environmental situations. Based on this learning, the organism is able adapt and react appropriately to more general situations. An artificial neural network (ANN) can be seen as a numerical black-box, designed to mimic the training procedure of the biological network.

Mathematically, an ANN is described by the triplet $(\mathcal{N}, \mathcal{V}, \mathcal{W})$. Here, $\mathcal{N}$ is the set of all artificial neurons in the network, while $\mathcal{V}$ represents the set of all directed connection (i, j), $i, j \in \mathcal{N}$, where i is the sending neuron and j is the receiving neuron. The neural connections are weighted, with $\mathcal{W}$ being the set of weights $w_{i,j}$ for the connections (i, j). A single neuron j, receiving signals $y_{s_1}, ..., y_{s_k}$ from $s_1, ..., s_k$ sending neurons, is depicted in Fig. 3. The weighted accumulation q_j stored in the neuron can be expressed in terms of a *propagation function*

$$q_j = f_{\text{prop}}(y_{s_1}, ..., y_{s_k}, w_{s_1,j}, ..., w_{s_k,j}).$$

In practice, the propagation function is chosen to be linear,

$$f_{\text{prop}}(y_{s_1}, ..., y_{s_k}, w_{s_1,j}, ..., w_{s_k,j}) = \sum_{i=1}^{k} w_{s_i,j} y_{s_i},$$

which is also the choice we adhere to in this paper. The neuron j transmits a *scalar-valued* signal, provided the accumulation crosses a certain threshold or *bias* $-b_j$ (in literature, the bias is often taken as the negative of the threshold value). This aspect is modeled using a non-linear *activation function*

$$y_j = f_{\text{act}}(q_j + b_j).$$

The simplest example of an activation function is the Heaviside function

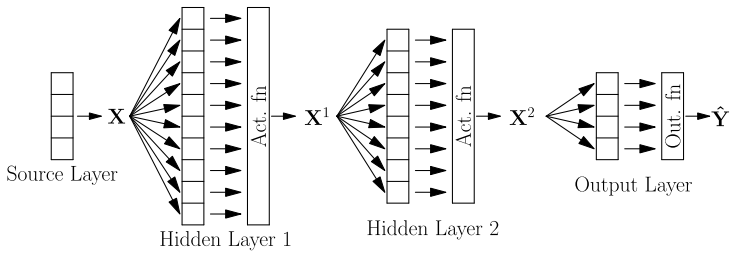


Fig. 4. An MLP with 2 hidden layers. The input layer transmits the signal $\mathbf{X}$ to the first hidden layer. The final output of the network is $\hat{\mathbf{Y}}$, which is the prediction to the true output $\mathbf{Y}$ corresponding to $\mathbf{X}$.

$$h(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}, \tag{3.2}$$

which leads to the McCulloch–Pitts neuron model [30]. However, (3.2) is not preferred in practice since its derivative vanishes everywhere (see discussion in Section 3.3).

3.1. Multi-layer perceptron

Several architectures have been proposed for ANNs [29,31], describing the arrangement and connectivity of neurons in the network. We focus on a specific architecture known as a multi-layer perceptron (MLP), in which the neurons arranged in several layers. The first layer with N_I source neurons is called the *input layer*, while the last layer with N_0 neurons is termed as the *output layer*. The remaining layers lying in between are called the *hidden layers*, with the k-th hidden layer consisting of N_k neurons. The structure of an MLP with two hidden layers is shown in Fig. 4. The neurons in a given layer receive signals from the layer preceding it, and send signals to the succeeding layer. Furthermore, the neurons within a single layer do not communicate with each other. No computations occur inside the input layer, and it simply provides the source signal to the network. The signals from the output layer do not pass through an activation function, but may pass through an *output function* to convert the signals to a meaningful form. For instance, for the classification problem, the output values should be scaled to lie between 0 and 1 to indicate the probability of the input belonging to a particular class. Since the signal transmitted by each neuron is scalar-valued, the widths of the input and output layers of the MLP matches the dimensions of the domain and range spaces of the function (3.1) being approximated.

3.2. Training the network

The appropriate weights and biases of the network are obtained by training the MLP on a given dataset $\mathbb{T}$. The training procedure is essentially an iterative algorithm which tunes the network weights to accurately predict the responses corresponding to the set $\mathbb{T}$. In addition, the trained network must be capable of predicting the responses for data outside $\mathbb{T}$, with a suitable degree of accuracy. This property of the network is termed as *generalization*.

There are several training strategies that one can opt for, a detailed description of which can be found in [29]. For our model, we consider the training paradigm termed as *supervised learning*, where the true responses for the points in the training set are known a priori. The training aims to minimize the error between the prediction and the truth. More precisely, a cost function $\mathcal{C}$ is defined

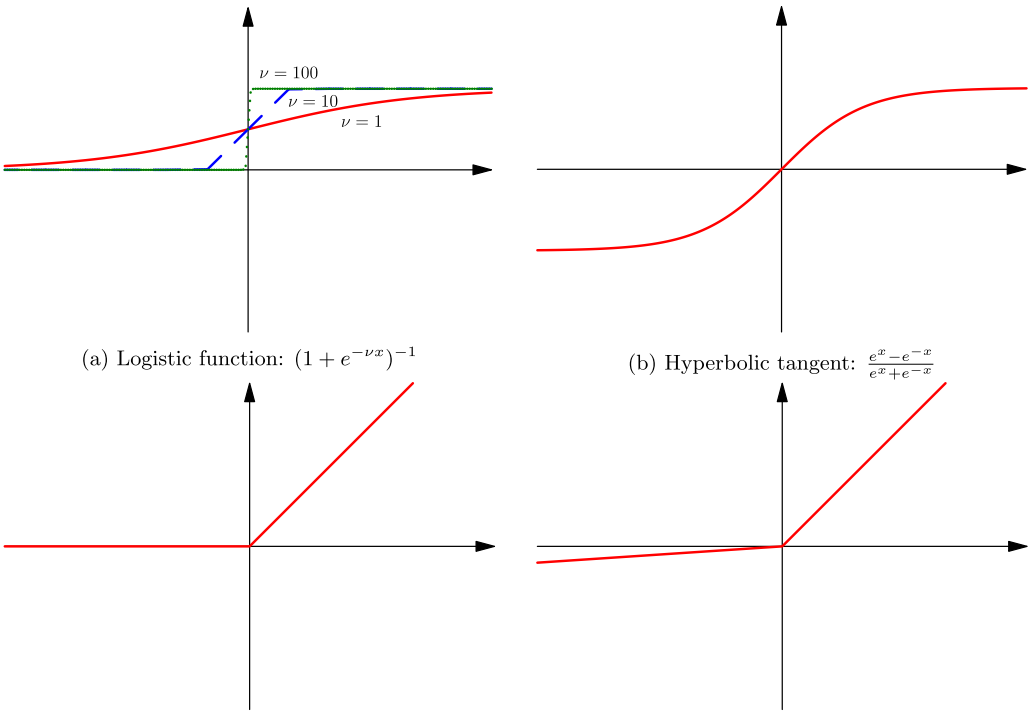
$$\mathcal{C} := \mathcal{C}(\mathbf{Y}, \hat{\mathbf{Y}}), \quad \mathbf{Y} = \mathbf{G}(\mathbf{X}), \quad \hat{\mathbf{Y}} = \hat{\mathbf{G}}(\mathbf{X}), \qquad \forall\, \mathbf{X} \in \mathbb{R}^{N_I},$$

where $\hat{\mathbf{G}}$ represents the approximation of $\mathbf{G}$ by the neural network. The function $\mathcal{C}$ can be seen as a measure of the discrepancy between the predicted response $\hat{\mathbf{Y}}$ and the true response $\mathbf{Y}$. Supervised learning aims to find the optimal values for the weights and biases of the network to minimize $\mathcal{C}$ over the training set $\mathbb{T}$.

Remark 3.1. Even if $\mathcal{C}$ is a convex function of $\mathbf{Y}, \hat{\mathbf{Y}}$, it need not be convex in terms of the weights and biases. Thus, the optimization algorithm may converge to a local minima.

3.3. Activation function

The activation function used in the MLP network is responsible for introducing non-linearity into the model. As mentioned earlier, the most obvious choice for the activation function is the Heaviside function (3.2). To understand why this may not be a suitable choice, we note that most optimization algorithms, such as gradient descent, update the weight of the connection between the neuron pair (i, j) iteratively as



(a) Logistic function: $(1 + e^{-\nu x})^{-1}$ (b) Hyperbolic tangent: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

(c) ReLU: $\max(0, x)$ (d) Leaky ReLU: $\max(0, x) - \nu \max(0, -x)$

Fig. 5. Choices for the activation function used in the MLP network.

$$w_{i,j}^{s+1} = w_{i,j}^{s} + \Delta w_{i,j}, \qquad \Delta w_{i,j} = -\eta \frac{\partial \mathcal{C}}{\partial w_{i,j}},$$

where $\Delta w_{i,j}$ is the update step and $\eta > 0$ is the *learning rate*. The evaluation of the gradient of the cost function involves the gradient of the activation function, which is essentially zero for the Heaviside function. Thus, the updates in each iteration would be very small, leading to a slow convergence of the algorithm. Other non-linear choices for the activation function include the logistic function and the hyperbolic tangent (see Fig. 5), which are smooth functions capturing the qualitative features of the Heaviside function. However, these too suffer from vanishing gradients as we move away from the origin. A popular activation function used by most practitioners, is the *rectified linear unit* (ReLU) [32], shown in Fig. 5(c). Training with ReLU is often faster, when compared to models using the logistic or hyperbolic tangent activation function [33]. This can been justified by noting the linear, non-saturating form of the ReLU function, and the inexpensive evaluation of the function itself. Unfortunately, the ReLU function can suffer from the issue of *dying neurons* during the training process. If an update of weights and the bias for a particular neuron j leads to its deactivation, i.e., $q_j + b_j < 0$, then the neuron may never activate again for the rest of the training. Thus, by the end of the training, most of the neurons are left inactive in the network. A proposed fix to this problem is to consider the *leaky ReLU* function [34], which modifies the original ReLU by adding a small negative slope ν when $x < 0$, as shown in Fig. 5(d).

4. An MLP trouble-cell indicator

We now describe the design of an MLP-based troubled-cell indicator. The function we wish to approximate is the true troubled-cell indicator. As is the case with the slope-limiter based indicators discussed in Section 2, the input for our MLP is the vector $(\overline{u}_{i-1},\ \overline{u}_i,\ \overline{u}_{i+1},\ u^+_{i-\frac{1}{2}},\ u^-_{i+\frac{1}{2}})^\top \in \mathbb{R}^5$, i.e., the input layer has $N_I = 5$ neurons. We use 5 hidden layers, whose widths vary as 256, 128, 64, 32 and 16 as we move from the input layer to the output layer. Based on the discussion in Section 3.3, we choose the leaky ReLU as our activation function. The output layer has a width of 2 neurons, giving the output $\mathbf{X}^O \in \mathbb{R}^2$. Finally, the vector $\mathbf{X}^O$ is put through the *softmax* output function to give the output $\hat{\mathbf{Y}}$

$$\hat{Y}^1 = \frac{e^{X^1}}{e^{X^1} + e^{X^2}}, \quad \hat{Y}^2 = \frac{e^{X^2}}{e^{X^1} + e^{X^2}}, \quad \mathbf{X}^O = (X^1,\ X^2)^\top. \tag{4.1}$$

The softmax transforms the vector of arbitrary real numbers into a vector of real values in $[0, 1]$ which sum up to unity. Thus, the final output can be viewed as the probability that the cell I_i falls into either of two classes: a troubled-cell or a good-cell. Specifically, $\hat{Y}^1$ represents the probability that the cell in question is a troubled-cell.

The cost functional used to train the model is given by the *cross entropy function*

$$\mathcal{C} = -\frac{1}{S}\sum_{k=1}^{S}\left[Y_k^1 \log(\hat{Y}_k^1) + Y_k^2 \log(\hat{Y}_k^2)\right],$$

where S is the number of samples used for training, while $\mathbf{Y}_k = (Y_k^1, Y_k^2)^\top$ is the true troubled-cell probability distribution for a given sample k. Note that Y_k^1, Y_k^2 can only take the discrete values 0 or 1. The cross-entropy function is closely related to the Kullback–Leibler divergence, which measures the discrepancy between two probability distributions [35]. In our case, we aim to measure and minimize the discrepancy between the probability distribution about the type of a given cell predicted by the MLP and the true boolean distribution.

The training is performed using a stochastic optimization algorithm, which uses mini-batches of size S_b from the training set, to take a single optimization step. More specifically, the full training set with S data-points is shuffled, following which mini-batches with $S_b < S$ samples are sequentially extracted to take S/S_b optimization steps. Once the entire training set is exhausted, the training is said to have completed one full *epoch*. The training set is then reshuffled and the process is repeated for several epochs. The shuffling introduces stochasticity in the training data set, and has been observed to lead to faster convergence [36].

The network topology and the choice of cost function can lead to *overfitting* of the network to the training dataset, which can severely effect the ability of the network to generalize. A commonly used method to avoid overfitting involves the *regularization* of the cost functional [37] by penalizing the weights $\mathbf{W}$ of the network

$$\widetilde{\mathcal{C}} = \mathcal{C} + \beta\|\mathbf{W}\|_2^2, \quad \beta \geq 0, \tag{4.2}$$

where $\|\mathbf{W}\|_2^2$ represents the squared sum of all the weights in the MLP. The *early stopping* of the training is yet another commonly used approach, which makes use of a *validation* data set $\mathbb{V}$ that is independent of the training set $\mathbb{T}$ [37]. After each epoch, the responses of the MLP is evaluated over $\mathbb{V}$ and the accuracy of the output is evaluated. As the training evolves, the accuracy on the validation set ideally increases, signifying that the MLP is capable of generalization. However, the validation accuracy starts decreasing after a point due to over-fitting. We evaluate the accuracy of the MLP responses over the set $\mathbb{V}$ as

$$V_{\text{acc}} = \frac{\#\{\mathbf{X} \in \mathbb{V} \mid \hat{\mathbf{Y}} = \hat{\mathbf{G}}(\mathbf{X}),\ \ \hat{Y}^1 \geq 0.5\}}{\#\mathbb{V}} \times 100, \tag{4.3}$$

where $\hat{Y}^1$ is the probability of the cell being a troubled-cell. Based on this evaluation, an early stopping criteria is used, wherein the training is terminated if the accuracy on the validation set decreases for L consecutive epochs.

Since the cost-function is not convex, the choice of initial conditions can strongly influence the local minima to which the optimization converges. Thus, we restart the training process R times with different initializations of the weights and biases, and select the model with the best generalization (measured in terms of the final validation accuracy). The training is performed using TensorFlow, which is an open-source software library for machine learning [38]. The offline training procedure to obtain the optimal wights and biases for the MLP, is outlined in Algorithms 1 and 2. The MLP with the following structure of weights $\mathbf{W}$ and biases $\mathbf{b}$

$$\begin{aligned}
&\mathbf{W} = \{\mathbf{W_1}, \mathbf{W_2}, \mathbf{W_3}, \mathbf{W_4}, \mathbf{W_5}, \mathbf{W_0}\} \quad \text{with} \quad \mathbf{W_1} \in \mathbb{R}^{256\times 5},\ \mathbf{W_2} \in \mathbb{R}^{128\times 256},\\
&\mathbf{W_3} \in \mathbb{R}^{64\times 128},\ \mathbf{W_4} \in \mathbb{R}^{32\times 64},\ \mathbf{W_5} \in \mathbb{R}^{16\times 32},\ \mathbf{W_0} \in \mathbb{R}^{2\times 16},\\
&\mathbf{b} = \left\{\mathbf{b_1} \in \mathbb{R}^{256},\ \mathbf{b_2} \in \mathbb{R}^{128},\ \mathbf{b_3} \in \mathbb{R}^{64},\ \mathbf{b_4} \in \mathbb{R}^{32},\ \mathbf{b_5} \in \mathbb{R}^{16},\ \mathbf{b_0} \in \mathbb{R}^{2}\right\}
\end{aligned} \tag{4.4}$$

is implemented in Algorithm 3. To obtain the gradient of the cost function (4.2), we also need to differentiate the network itself. In Tensorflow, this is achieved by using *automatic differentiation* [39] on the MLP algorithm.

4.1. Generating the training and validation sets

We briefly discuss the generation of the sets $\mathbb{T}$ and $\mathbb{V}$ used to train the MLP network. The methodology can be described via the following steps:

1. Choose a function $u(x)$ such that complete information about its regularity is available in the interval $[a, b]$. For instance we can choose the sine-wave or a step-function.
2. Pick a point $x_i \in [a, b]$ and set a mesh size h ensuring that $a \leq x_i - \frac{3}{2}h < x_i + \frac{3}{2}h \leq b$. Thus, we can construct a 3-cell stencil centered at x_i, which is contained in the interval $[a, b]$ (see Fig. 6).
3. In each cell, project the solution onto the space of polynomials of degree r. We use the Legendre polynomials for the projection.

Algorithm 1: Offline training of the neural network.

Input : Neural network $(\mathcal{N}, \mathcal{V})$, learning rate η, leaky ReLU parameter ν, cost function $\widetilde{\mathcal{C}}$, cost regularization parameter β, training set $\mathbb{T}$, validation set $\mathbb{V}$, maximum number of epochs T, stopping parameter L, mini-batch size S_b, number of restarts R.

Output : Optimal weights $\mathbf{W_{opt}}$ and biases $\mathbf{b_{opt}}$

Function $[\mathbf{W_{opt}}, \mathbf{b_{opt}}] = \texttt{TRAIN\_MLP}(\mathcal{N}, \mathcal{V}, \eta, \nu, \widetilde{\mathcal{C}}, \beta, \mathbb{T}, \mathbb{V}, T, L, S_b, R)$**:**

- $V_{acc}^{opt} = 0$
- **for** $r \leftarrow 1$ **to** R **do**
  - $t = 1,\ V_{acc}^{r} = 0,\ l = 0$
  - Randomly initialize $\mathbf{W}(1)$ and $\mathbf{b}(1)$
  - **while** $t <= T$ ***and*** $l < L$ **do**
    - $[\mathbf{W}(t+1),\ \mathbf{b}(t+1)] = \texttt{MINIBATCH\_OPT}(\mathcal{N},\ \mathcal{V},\ \mathbf{W}(t),\ \mathbf{b}(t),\ \eta,\ \nu,\ \widetilde{\mathcal{C}},\ \beta,\ \mathbb{T},\ S_b)$
    - `// Evaluate the accuracy according to (4.3)`
    - $V_{acc} = \texttt{EVAL\_ACC}(\mathcal{N}, \mathcal{V},\ \mathbf{W}(t+1), \mathbf{b}(t+1),\ \mathbb{V})$
    - **if** $V_{acc} < V_{acc}^{r}$ **then**
      - $l \leftarrow l + 1$
    - **else**
      - $l = 0$
    - **end**
    - $t \leftarrow t + 1$
  - **end**
  - $V_{acc}^{r} = \texttt{EVAL\_ACC}(\mathcal{N}, \mathcal{V},\ \mathbf{W}(t-l), \mathbf{b}(t-l),\ \mathbb{V})$
  - **if** $V_{acc}^{r} > V_{acc}^{opt}$ **then**
    - $\mathbf{W_{opt}} \leftarrow \mathbf{W}(t-l)$
    - $\mathbf{b_{opt}} \leftarrow \mathbf{b}(t-l)$
    - $V_{acc}^{opt} \leftarrow V_{acc}^{r}$
  - **end**
- **end**

return

Algorithm 2: Mini-batch optimization. This is called by the function TRAIN_MLP.

Input : Neural network $(\mathcal{N}, \mathcal{V})$, old weights $\mathbf{W_0}$, old biases $\mathbf{b_0}$, learning rate η, leaky ReLU parameter ν, cost function $\widetilde{\mathcal{C}}$, cost regularization parameter β, training set $\mathbb{T}$, mini-batch size S_b.

Output : Updated weights $\mathbf{W}$ and biases $\mathbf{b}$

Function $[\mathbf{W}, \mathbf{b}] = \texttt{MINIBATCH\_OPT}(\mathcal{N},\ \mathcal{V},\ \mathbf{W_0},\ \mathbf{b_0},\ \eta,\ \nu,\ \widetilde{\mathcal{C}},\ \beta,\ \mathbb{T},\ S_b)$**:**

- Shuffle data set $\mathbb{T}$
- $i = 1$
- $N = \text{size}(\mathbb{T})$
- $\mathbf{W} \leftarrow \mathbf{W_0}$
- $\mathbf{b} \leftarrow \mathbf{b_0}$
- **while** $i <= N$ **do**
  - $Data = \mathbb{T}(i : i + S_b - 1)$
  - `// Evaluate step update using any optimizer`
  - $[\Delta\mathbf{W}, \Delta\mathbf{b}] = \texttt{OPT\_ITERATION}(\mathcal{N},\ \mathcal{V},\ \mathbf{W_0},\ \mathbf{b_0},\ \eta,\ \nu,\ \widetilde{\mathcal{C}},\ \beta,\ Data)$
  - $\mathbf{W} \leftarrow \mathbf{W} + \Delta\mathbf{W}$
  - $\mathbf{b} \leftarrow \mathbf{b} + \Delta\mathbf{b}$
  - $i \leftarrow \min(i + S_b,\ N)$
- **end**

return

Algorithm 3: MLP network used as a troubled-cell indicator.

Input : MLP weights $\mathbf{W}$ and biases $\mathbf{b}$ (see (4.4)), leaky ReLU parameter ν, solution data $\mathbf{X} = (\overline{u}_{i-1},\ \overline{u}_i,\ \overline{u}_{i+1},\ u^{+}_{i-\frac{1}{2}},\ u^{-}_{i+\frac{1}{2}})^{\top} \in \mathbb{R}^5$.

Output : 1 if the data corresponds to a troubled-cell, else 0.

Function $[ind] = \texttt{MLP\_INDICATOR}(\mathbf{W},\ \mathbf{b},\ \nu,\ \mathbf{X})$**:**

- `// Scale data to lie in` $[-1, 1]^5$
- $\mathbf{X} = \mathbf{X} / \max(\text{abs}(\mathbf{X}),\ 1)$
- $ind = 0$
- `//` $i = 6$ `corresponds to the output layer`
- **for** $i \leftarrow 1$ **to** 6 **do**
  - $\mathbf{X} \leftarrow \mathbf{W}\{i\} * \mathbf{X} + \mathbf{b}\{i\}$ `// Propagation`
  - $\mathbf{X} \leftarrow \texttt{LEAKY\_RELU}(\mathbf{X},\ \nu)$ `// Activation: Component-wise leaky ReLU`
- **end**
- $\mathbf{Y} \leftarrow \texttt{SOFTMAX}(\mathbf{X})$ `// See (4.1)`
- **if** $\mathbf{Y}(0) > 0.5$ **then**
  - $ind = 1$
- **end**

return

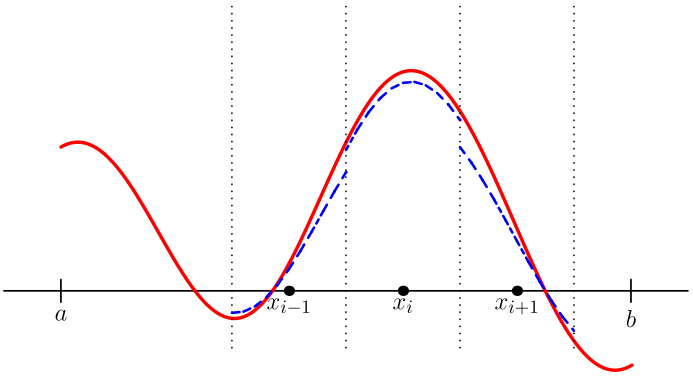


Fig. 6. Construction of a training sample for a known function $u(x)$ (solid line) in the interval $[a, b]$. In each cell of the stencil, the function is approximated using a polynomial (dashed line).

4. Find the cell-averages of the approximating polynomials in each of the three cells to obtain the values $\overline{u}_{i-1}$, $\overline{u}_i$, $\overline{u}_{i+1}$. Also extract $u^+_{i-\frac{1}{2}}$, $u^-_{i+\frac{1}{2}}$ using the polynomial in the cell I_i. Thus, we have generated the input vector $\mathbf{X}$ corresponding to the cell I_i.
5. The true output corresponding to $\mathbf{X}$ depends on the regularity of the solution in the stencil. Ideally, if the solution in cell I_i loses regularity, i.e., has a discontinuity, or is continuous but not differentiable, the cell is flagged as a troubled-cell with the true output $\mathbf{Y} = (1, 0)^\top$. Otherwise, the output is $\mathbf{Y} = (0, 1)^\top$.
6. Repeat steps 2–5 by varying the choice of the point x_i, the mesh size h and the polynomial degree r.
7. Repeat steps 1–6 for different known functions $u(x)$.

Remark 4.1. In practice, we construct true outputs by flagging the cell I_i as a troubled-cell if regularity is lost in the wider zone $[x_i - \frac{3}{2}h, x_i + \frac{3}{2}h]$. This ensures the indicator is more robust in terms of capturing the discontinuities.

4.2. A word on the computational cost

We briefly discuss the computational cost associated with the MLP indicator, which can be broken into two part: the training cost and the cost involved with using the trained MLP. We first elaborate on the latter, by analyzing the key components of Algorithm 3. In each operational layer i of the MLP with $\mathbf{W}\{i\} \in \mathbb{R}^{m\times n}$, the complexities of the propagation and activation steps are $\mathcal{O}(mn)$ and $\mathcal{O}(m)$, respectively. Thus, for our MLP determined by (4.4), the complexity for a single data point $\mathbf{X} \in \mathbb{R}^5$ will be dominated by the computations in the second hidden layer, which is given by $\epsilon = \mathcal{O}(128 \times 256)$. Note that ϵ is independent of the mesh size, or the number of elements. For a mesh with N elements, the total computational cost for using the MLP indicator on the whole mesh is $\mathcal{O}(\epsilon N)$. In other words, the algorithm scales linearly with the N.

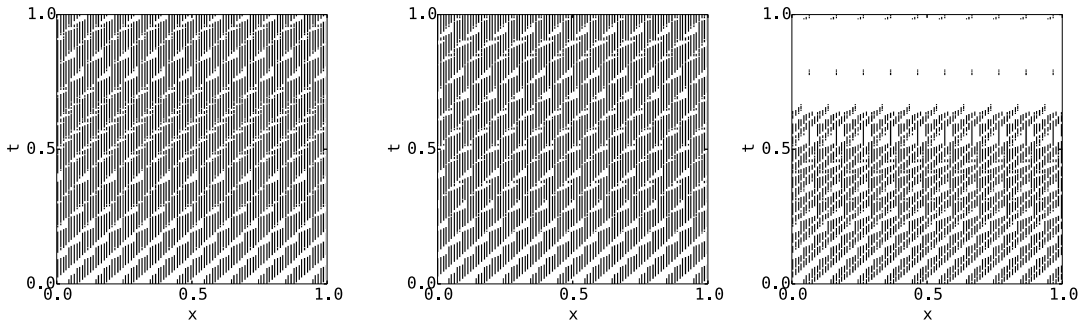
Let us now estimate the training cost by considering Algorithms 1 and 2. The evaluation of the function MINIBATCH_OPT is the most expensive component of the training procedure. Let us denote by ϑ, the complexity of evaluating $\Delta\mathbf{W}$ and $\Delta\mathbf{b}$ for each mini-batch. Note that ϑ depends on the choice of the optimizer, the cost function $\widetilde{\mathcal{C}}$, the size of the network being trained, as well as the mini-batch size S_b. Since this has to be evaluated for each mini-batch, the complexity of Algorithm 2 is $\mathcal{O}(\vartheta N_{\mathbb{T}}/S_b)$, where $N_{\mathbb{T}} = \#\mathbb{T}$. We have an additional cost of $\mathcal{O}(\epsilon N_{\mathbb{V}})$ for evaluating the function EVAL_ACC in each epoch of Algorithm 1, where $N_{\mathbb{V}} = \#\mathbb{V}$. We get the maximal complexity by assuming that the stopping criteria does not come into play for any of the R training cycles. Thus, the complexity of the full training algorithm can be estimated as $\mathcal{O}\left(RT(\vartheta N_{\mathbb{T}}/S_b + \epsilon N_{\mathbb{V}})\right)$. Although the training procedure can be expensive for large data sets $\mathbb{T}$ and $\mathbb{V}$, it needs to be done just once, following which the trained MLP is used as a black-box indicator for all one-dimensional conservation laws.

5. Numerical results

We now demonstrate the capability of the MLP network when used as a troubled-cell indicator in the RKDG framework. The MLP is trained offline by setting the slope parameter for leaky ReLU as $\nu = 10^{-3}$, and the cost regularization parameter as $\beta = 10^{-2}$. We also use an early stopping criteria, by setting $L = 10$. The momentum based Adam stochastic optimizer [40] is used to minimize the cost function (4.2), with an initial learning rate of $\eta = 10^{-3}$. The training and validation datasets are constructed using a number of functions, which are listed in Table 1(a)–(b). For each function, the mesh size h and the approximating polynomial degree r are varied. Some of the functions have additional parameters, which are also varied to generate additional sample points. Furthermore, we use mini-batches of size $S_b = 500$ after reshuffling the training set for each epoch. The training is restarted $R = 10$ times, with the weights and biases randomly initialized using a normal distribution at the beginning of each training. The model with the highest validation accuracy at the end, is chosen as the MLP indicator.

Table 1
Functions used to create the training and validation datasets. The last two columns of each table indicate the number of good cell and troubled-cell sample points extracted from each function. The total number of good cell and troubled-cell sample points present in each dataset are listed in the last row of each table. The total number of samples in each set is obtained by adding the corresponding good and troubled-cells.

$\mathbf{u}(\mathbf{x})$	Domain	Additional parameters varied	Good cells	Troubled cells
	(a) Functions used to create $\mathbb{T}$.			
$\sin(4\pi x)$	$[0, 1]$	–	4470	0
ax	$[-1, 1]$	$a \in \mathbb{R}$	10000	0
$a\|x\|$	$[-1, 1]$	$a \in \mathbb{R}$	800	3200
$ul.(x < x_0) + ur.(x > x_0)$ (only troubled-cells selected)	$[-1, 1]$	$(u_l, u_r) \in [-1, 1]^2$ $x_0 \in [-0.76, 0.76]$	0	19800
			15270	**23000**
	(b) Functions used to create $\mathbb{V}$.			
$\sum_{p=1}^{5} \sin(p\pi x)$	$[0, 2]$	–	3740	0
$\sin(2\pi x)\cos(3\pi x)\sin(4\pi x)$	$[0, 2]$	–	3740	0
$\sin(\pi x) + e^x$	$[-1, 1]$	–	3740	0
$ul.(x < x_0) + ur.(x > x_0)$ (only troubled-cells selected)	$[-1, 1]$	$(u_l, u_r) \in [-20, 20]^2$ $x_0 \in [-0.76, 0.76]$	0	13060
			11220	**13060**



(a) minmod (b) TVB-1 (c) TVB-2

Fig. 7. The time-history of flagged troubled-cells for the linear advection of a sine-wave, simulated until $T = 1$ with $N = 100$ cells and $r = 4$.

The DG scheme is evaluated using the local Lax–Friedrichs flux. We compare the performance of the trained MLP indicator with that of the minmod-limiter and the minmod-type TVB limiter. The notation TVB-1, TVB-2, and TVB-3 are used to refer to the TVB limiter with the parameter $M = 10$, $M = 100$, and $M = 1000$, respectively. The limited reconstruction in troubled-cells is performed using the MUSCL-scheme with the minmod limiter. The semi-discrete DG scheme is integrated in time using SSP-RK3. The computational cost of the scheme depends greatly on the number of cells flagged as troubled-cells. In order to analyze this cost, we also plot the troubled-cells detected by the various indicators at each time-step. All test cases considered in this paper have been simulated for $r = 1, 2, 3$ and 4. However, only the results with $r = 4$ are presented for most problems, due to paucity of space.

In addition to uniform grids, we also compute the results on randomly perturbed meshes obtained by modifying the (interior) cell-interfaces of a uniform mesh as follows

$$x_{i+\frac{1}{2}} \longrightarrow x_{i+\frac{1}{2}} + \theta h \omega_{i+\frac{1}{2}}, \qquad \omega_{i+\frac{1}{2}} \in \mathcal{U}([-0.5, 0.5]), \qquad i = 1, ..., N-1, \tag{5.1}$$

where θ controls the magnitude of perturbation. We choose $\theta = 10\%$ for all simulations.

We also demonstrate the performance of the MLP indicator on systems of conservation laws, by considering the one-dimensional shallow water equations, as well as the Euler equations. In the case of systems, we have the freedom to choose the variables used to detect the troubled-cells, which we refer to as the *indicator-variables*. Additionally, we need to choose the *limited-variables*, i.e., the variables which are limited in the troubled-cells. The importance of limiting the local characteristic variables to avoid spurious oscillations, especially for high-order schemes, has been demonstrated in [41]. Thus, we adhere to this choice of limited-variables for the results of systems of conservation laws.

5.1. Linear advection

We first consider the linear scalar advection equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0,$$

and set the advection speed $c = 1$ for all test cases. The time step is obtained by setting $\text{CFL} = 0.2$.

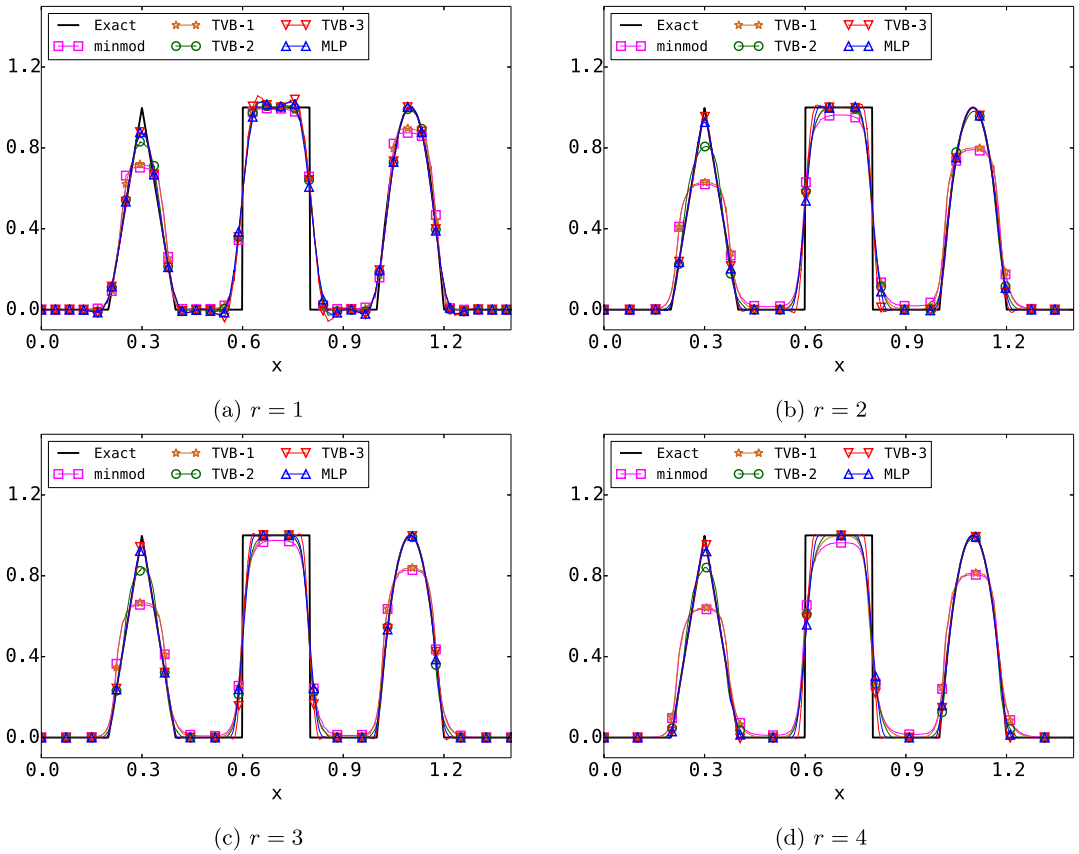


Fig. 8. Solution for the linear advection of the multi-wave, obtained at $T = 1.4$ with $N = 100$ cells and polynomial degree r. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

5.1.1. Sine-wave

This test case describes the advection of a smooth sine wave. The initial condition is given by

$$u_0(x) = \sin(10\pi x), \qquad x \in [0, 1],$$

with periodic boundary conditions. At the end of the final time $T = 1$, the wave completes one full revolution and returns to its original position. The solution is evaluated with various troubled-cell indicators on a mesh with $N = 100$ cells. Ideally, none of the cells should be flagged as troubled-cells, since the solution is smooth. However, the minmod, TVB-1 and TVB-2 indicators incorrectly mark cells with smooth extrema as troubled-cells, as can be seen in Fig. 7. On the other hand, no cells are flagged by the TVB-3 and MLP indicators. Note that, for the given problem, the parameter M corresponding TVB-3 is close to the curvature of the solution near the smooth extrema, which explains the performance of the limiter. Furthermore, the results with the TVB-3 and MLP indicators remain unchanged when the mesh is perturbed in accordance to 5.1.

5.1.2. Multi-wave

This test case corresponds to a solution consisting of waves with different degrees of regularity. The initial condition is given by

$$u_0(x) = \begin{cases} 10(x - 0.2) & \text{if } 0.2 < x \leq 0.3\,, \\ 10(0.4 - x) & \text{if } 0.3 < x \leq 0.4\,, \\ 1 & \text{if } 0.6 < x \leq 0.8\,, \\ 100(x - 1)(1.2 - x) & \text{if } 1 < x \leq 1.2\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

on the domain $[0, 1.4]$ with periodic boundary conditions. The solution completes one full revolution at the end of time $T = 1.4$. The minmod and TVB-1 limiters give very dissipative results, as can be seen in Fig. 8, while TVB-2 performs significantly better. Note that the TVB and MLP indicators give rise to overshoots near the second wave, especially for $r = 1$. However, the overshoots are reduced as the polynomial degree is increased, as shown in Fig. 9. Although TVB-3 is better at

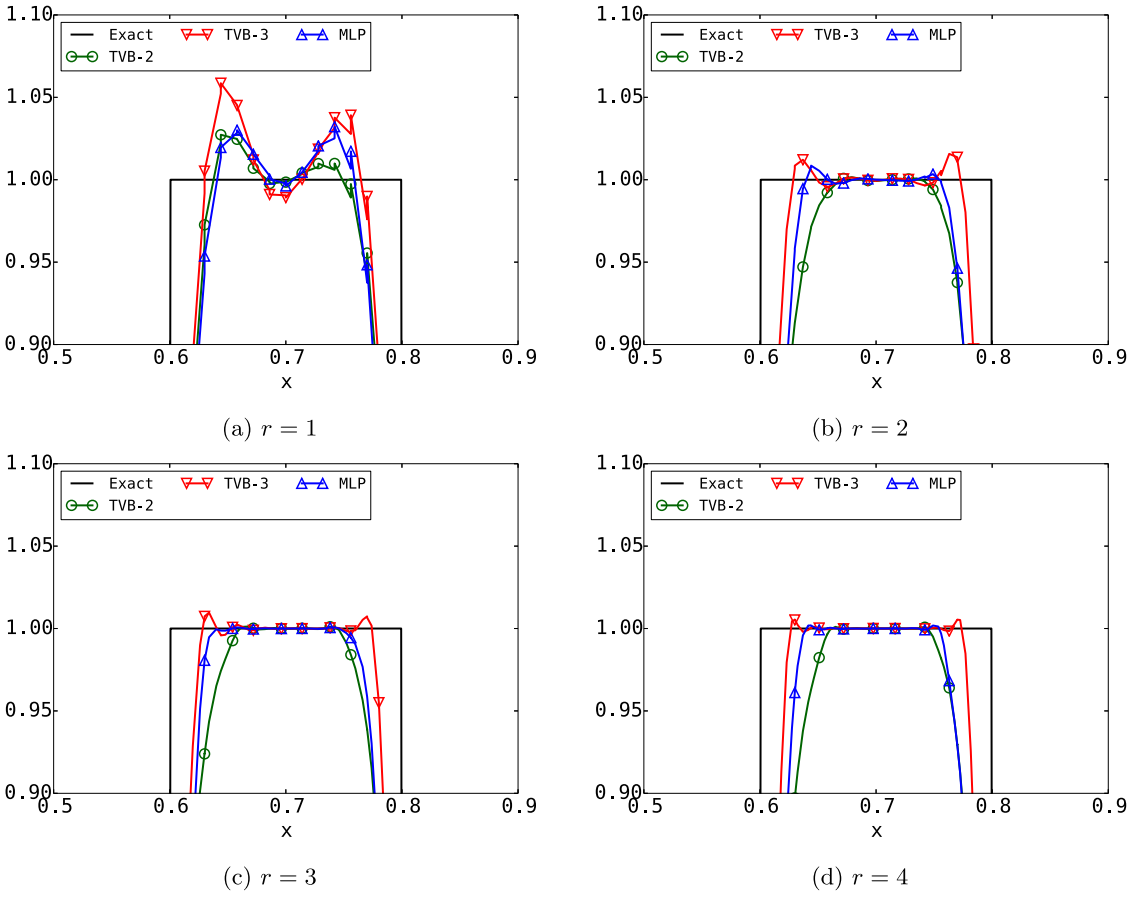


Fig. 9. Solution for the linear advection of the multi-wave, obtained at $T = 1.4$ with $N = 100$ cells and polynomial degree r. The plots are zoomed to focus on the central step wave.

resolving the discontinuity, it also gives the largest overshoots. The MLP indicator gives milder overshoots, with its resolution capabilities lying between TVB-2 and TVB-3.

The time-history of the troubled-cells marked by the various indicators for $r = 4$, is shown in Fig. 10. Since the minmod limiter marks the most number of cells, it leads to the most dissipative solution. The MLP indicator seems to mark the requisite number cells to control the overshoots, without being over-dissipative. Furthermore, the result with the MLP indicator on a randomly perturbed mesh is almost identical to the result obtained on a uniform mesh, as shown in Fig. 11 and 10(f).

5.2. Burgers equation

We compare the performance of the indicator functions for the non-linear Burgers equation

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{u^2}{2}\right) = 0.$$

The time step is obtained by setting $\mathrm{CFL} = 0.2$.

5.2.1. Shock collision

This test describes the collision of three shocks of varying strengths and speeds, which eventually move as a single shock wave to the right. The initial condition is given by

$$u_0(x) = \begin{cases} 10 & \text{if } x \le 0.2\,, \\ 6 & \text{if } 0.2 < x \le 0.4\,, \\ 0 & \text{if } 0.4 < x \le 0.6\,, \\ -4 & \text{if } 0.6 < x\,, \end{cases}$$

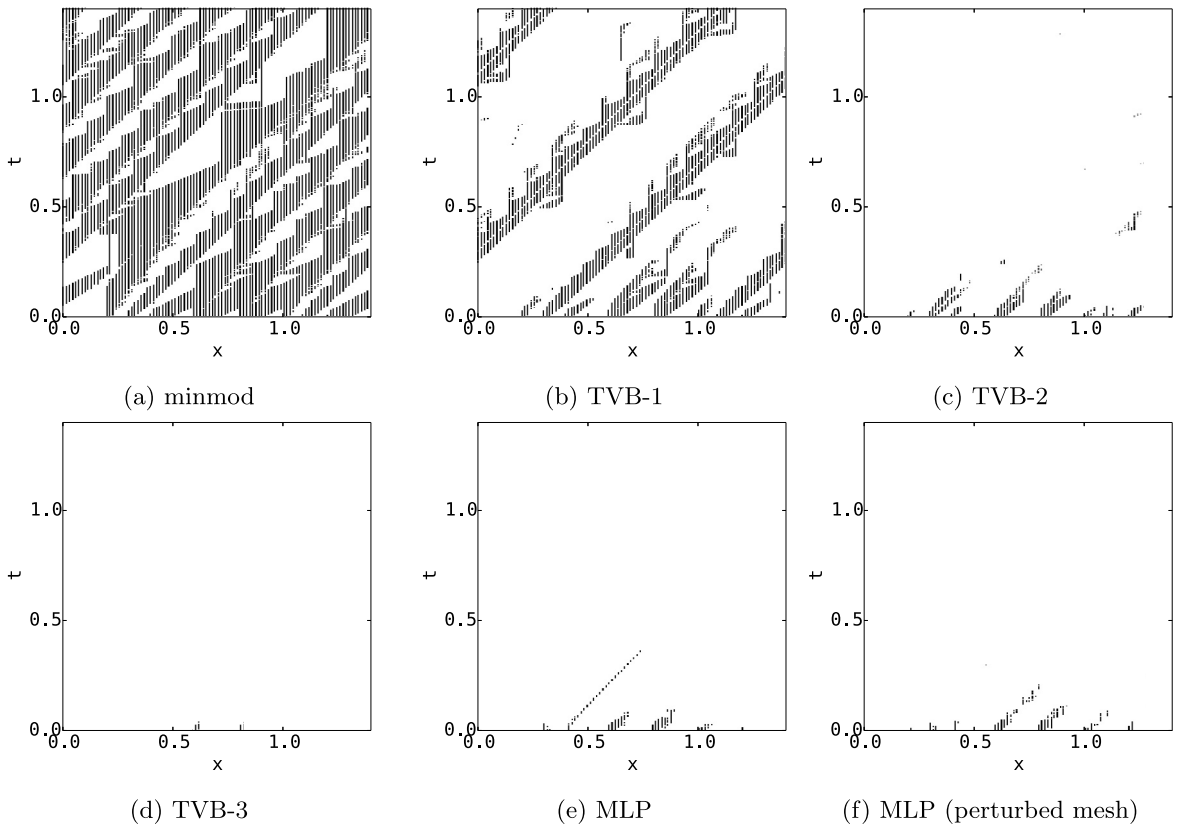


Fig. 10. The time-history of flagged troubled-cells for the linear advection of the multi-wave, simulated until $T = 1.4$ with $N = 100$ cells and $r = 4$. The plots (a)–(e) are obtained on a uniform mesh, while (f) is obtained on a randomly perturbed mesh.

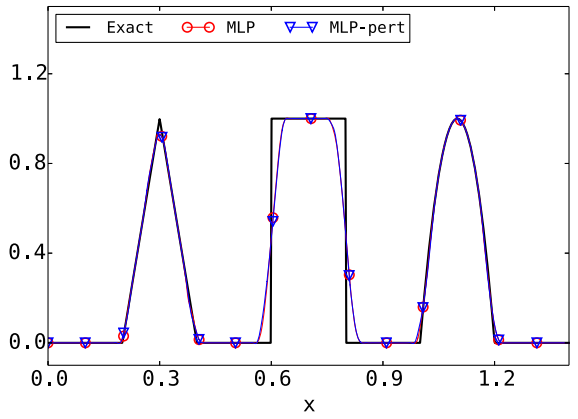


Fig. 11. Comparing the solution for the linear advection of the multi-wave, computed on a uniform mesh and a randomly perturbed mesh with $N = 100$ and $r = 4$.

on the domain $[0, 1]$ with open boundary conditions. The solutions are evaluated at time $T = 0.1$. The results with the various indicators on a mesh with $N = 100$ cells and $r = 4$ are indistinguishable, as shown in Fig. 12(a). To compare and assess the performance of the indicators, we consider the time-history of the troubled-cells. We notice from the results in Fig. 13 that the minmod limiter marks the most cells. Comparatively, the TVB limiter has thinner zones of marked cells, which decay as the TVB parameter M is increased. Thus, TVB-3 ensures the most cost-efficient simulation for this problem, while the MLP lies between TVB-2 and TVB-3. We observe from Fig. 12(b) that the MLP indicator works equally well when the mesh is randomly perturbed, with a similar marking of the troubled-cells as shown in Fig. 13(f).

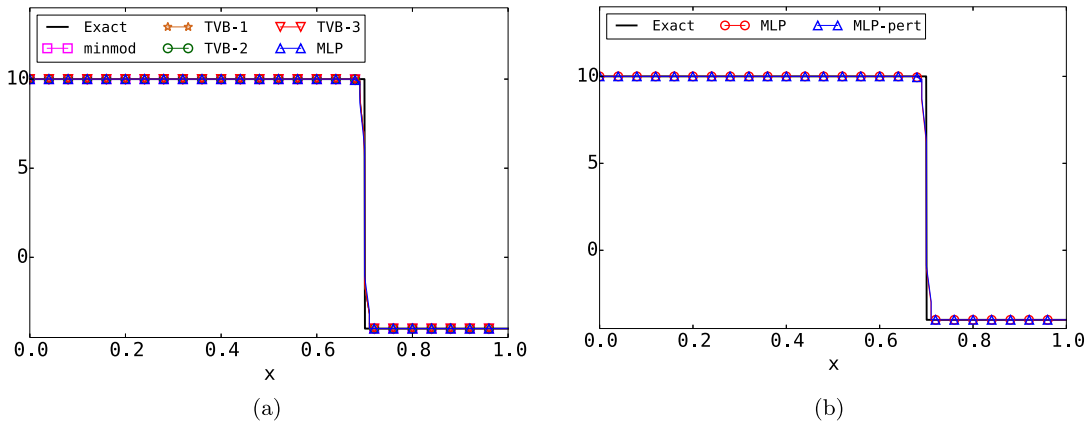


Fig. 12. Solution of the shock collision problem for the Burgers equation, computed at $T = 0.1$ with $N = 100$ cells and $r = 4$. (a) solution evaluated on a uniform mesh; (b) comparison of the solution with the MLP indicator on a uniform mesh and a randomly perturbed mesh.

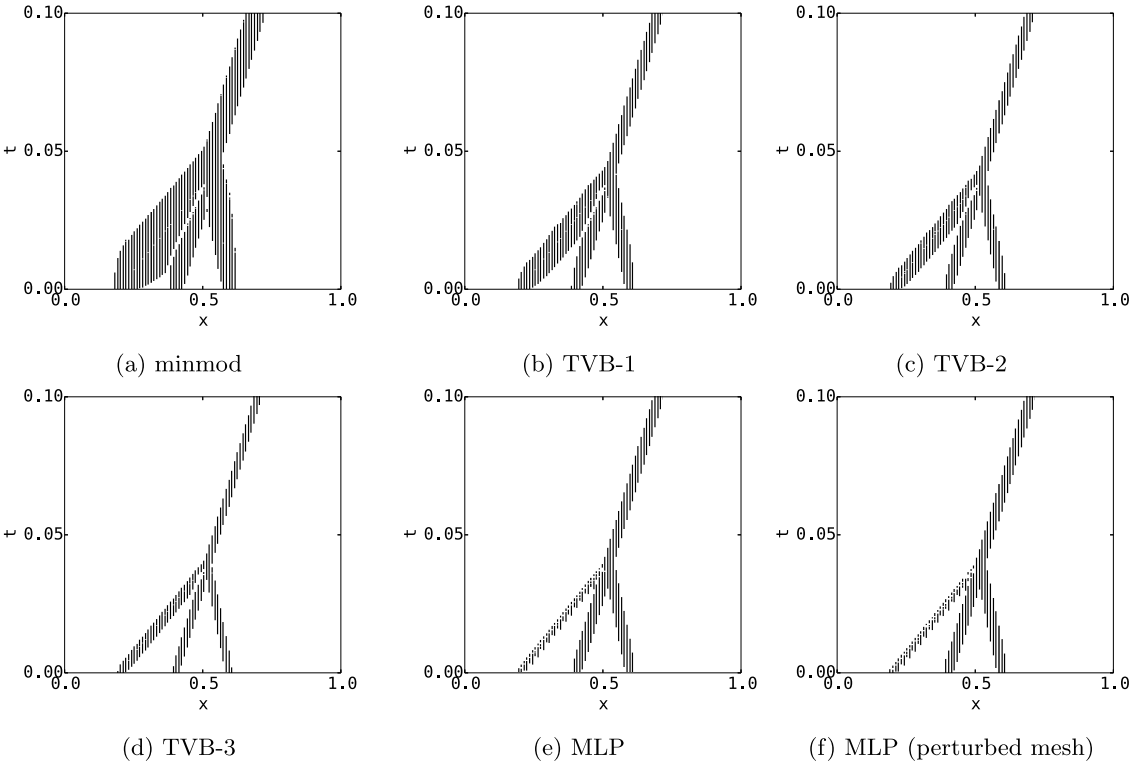


Fig. 13. The time-history of flagged troubled-cells of the shock collision problem for the Burgers equation, simulated until $T = 0.1$ with $N = 100$ cells and $r = 4$. The plots (a)–(e) are obtained on a uniform mesh, while (f) is obtained on a randomly perturbed mesh.

5.2.2. Compound wave

The initial condition for this test is a composition of smooth and discontinuous data, expressed as

$$u_0(x) = \begin{cases} \sin(\pi x) & \text{if } |x| \geq 1\,, \\ 3 & \text{if } -1 < x \leq -0.5\,, \\ 1 & \text{if } -0.5 < x \leq 0\,, \\ 3 & \text{if } 0 < x \leq 0.5\,, \\ 2 & \text{if } 0.5 < x \leq 1 \end{cases}$$

on the domain $[0, 1]$ with periodic boundary conditions. The initial condition is plotted in Fig. 14(a), and the solution is simulated until time $T = 0.4$. As the solution evolves, alternating shock and rarefaction waves begin to develop. In all the

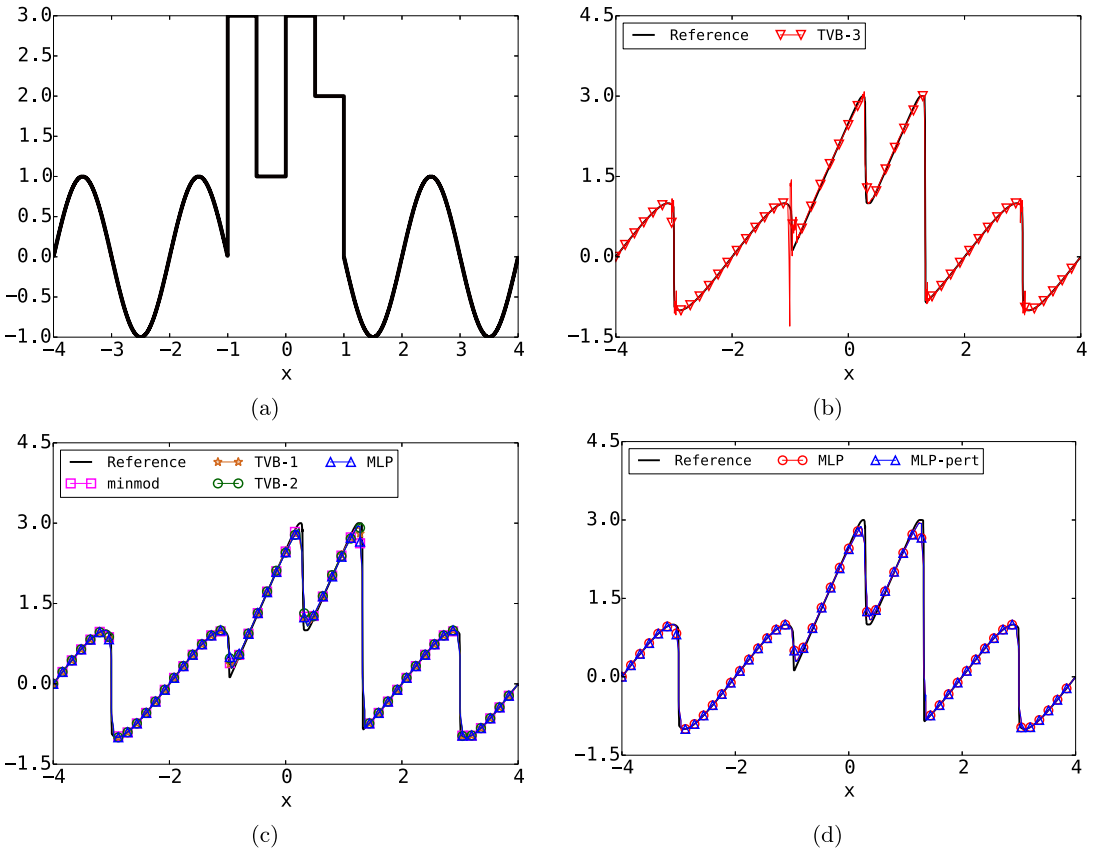


Fig. 14. Solution of the compound wave problem for the Burgers equation, computed at $T = 0.4$ with $N = 200$ cells and $r = 4$. (a) the initial condition; (b) the oscillatory solution obtained with TVB-3; (c) the solution with the remaining indicators; (d) comparison of the solution with the MLP indicator on a uniform mesh and a randomly perturbed mesh.

previous tests, the solutions obtained with TVB-3 were superior to those obtained with TVB-1 and TVB-2, ignoring minor oscillations. However, TVB-3 leads to large Gibbs oscillations near the discontinuities for the current test case, as shown in Fig. 14(b). This demonstrates that the same TVB parameter does not work uniformly for all problems. On the other hand, the results with minmod, TVB-1, TVB-2 and the MLP are equally well resolved. Once again, the minmod limiter marks most number of cells, as shown in Fig. 15, while the count with the MLP indicator lies between TVB-1 and TVB-2. The existence of Gibbs oscillations with TVB-3 is further supported by the fact that a negligible number of cells are flagged by the indicator. The performance of the MLP remains unchanged when the mesh is randomly perturbed, as shown in Figs. 14(d) and 15(f).

5.3. Buckley–Leverett

In order to demonstrate the performance of the MLP indicator for a non-convex flux, we consider the Buckley–Leverett equation with the flux function

$$f(u) = \frac{u^2}{u^2 + 0.5(1-u)^2},$$

where u represents the water saturation in a mixture of oil and water [42]. We consider the initial condition

$$u_0(x) = \begin{cases} 0.95 & \text{if } x \geq 0.5\,, \\ 0.1 & \text{if } x > 0.5\,, \end{cases}$$

on the domain $[0, 1.5]$, which evolves into a compound wave consisting of a shock and a rarefaction. The numerical solutions are evaluated at time $T = 0.4$ with CFL $= 0.4$, on a mesh with $N = 150$ cells and open boundary conditions. As shown in Fig. 16(a), the solutions with the various indicators are indistinguishable. From the point of cost-efficiency, TVB-3 marks the least number of cells, while MLP lies between TVB-2 and TVB-3 (see Fig. 17). Similar to the previous test cases, the MLP performs equally well on a perturbed mesh, as shown in Figs. 16(b) and 17(f).

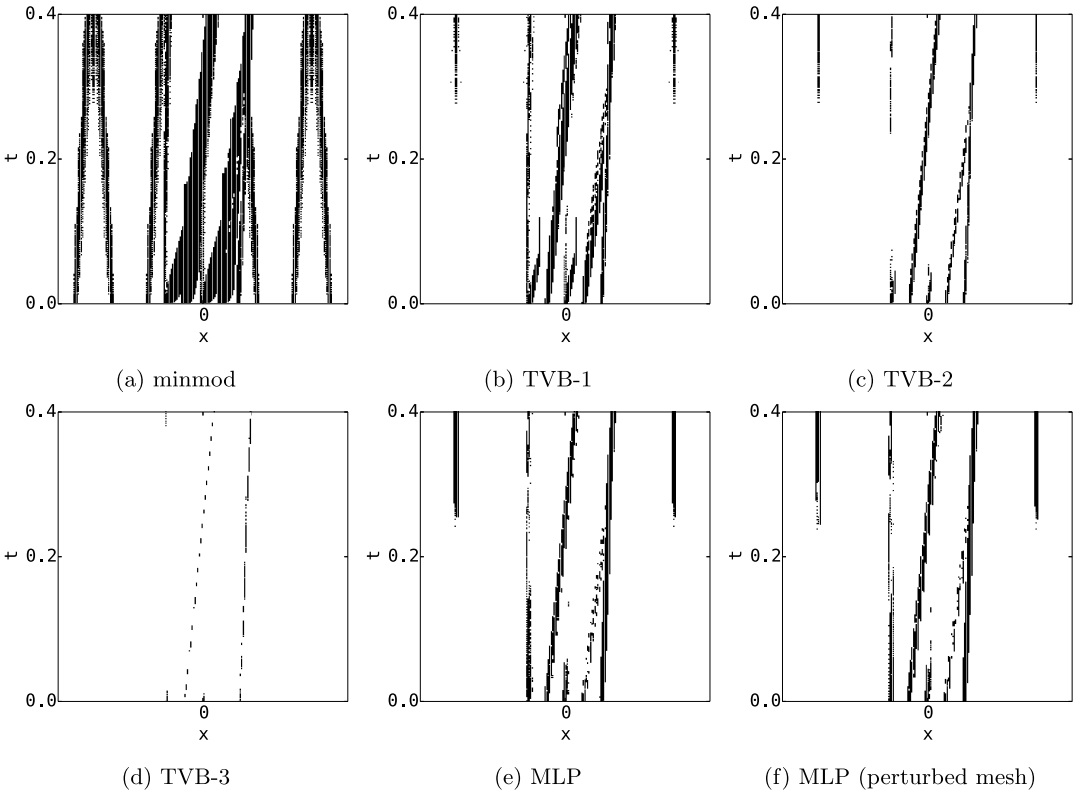


Fig. 15. The time-history of flagged troubled-cells of the compound wave problem for the Burgers equation, simulated until $T = 0.4$ with $N = 200$ cells and $r = 4$. The plots (a)–(e) are obtained on a uniform mesh, while (f) is obtained on a randomly perturbed mesh.

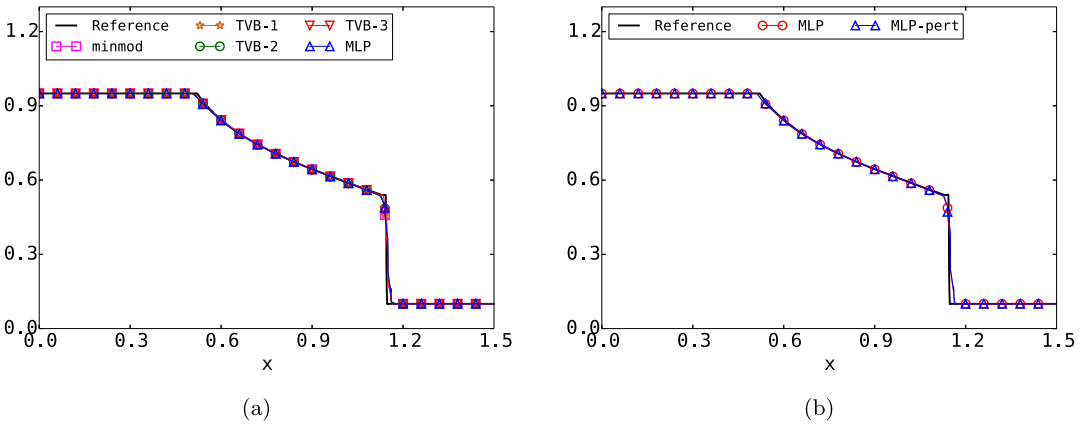


Fig. 16. Solution of the Riemann problem for the Buckley–Leverett equation, computed at $T = 0.4$ with $N = 150$ cells and $r = 4$. (a) solution evaluated on a uniform mesh; (b) comparison of the solution with the MLP indicator on a uniform mesh and a randomly perturbed mesh.

5.4. Shallow-water equations

We consider the one-dimensional shallow-water equations

$$\frac{\partial}{\partial t}\begin{bmatrix} D \\ Du \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} Du \\ Du^2 + \frac{1}{2}gD \end{bmatrix},$$

where D represents the depth of the fluid, u denotes the fluid velocity and g denotes the acceleration due to gravity. We solve the following Riemann problem simulating a dam-break

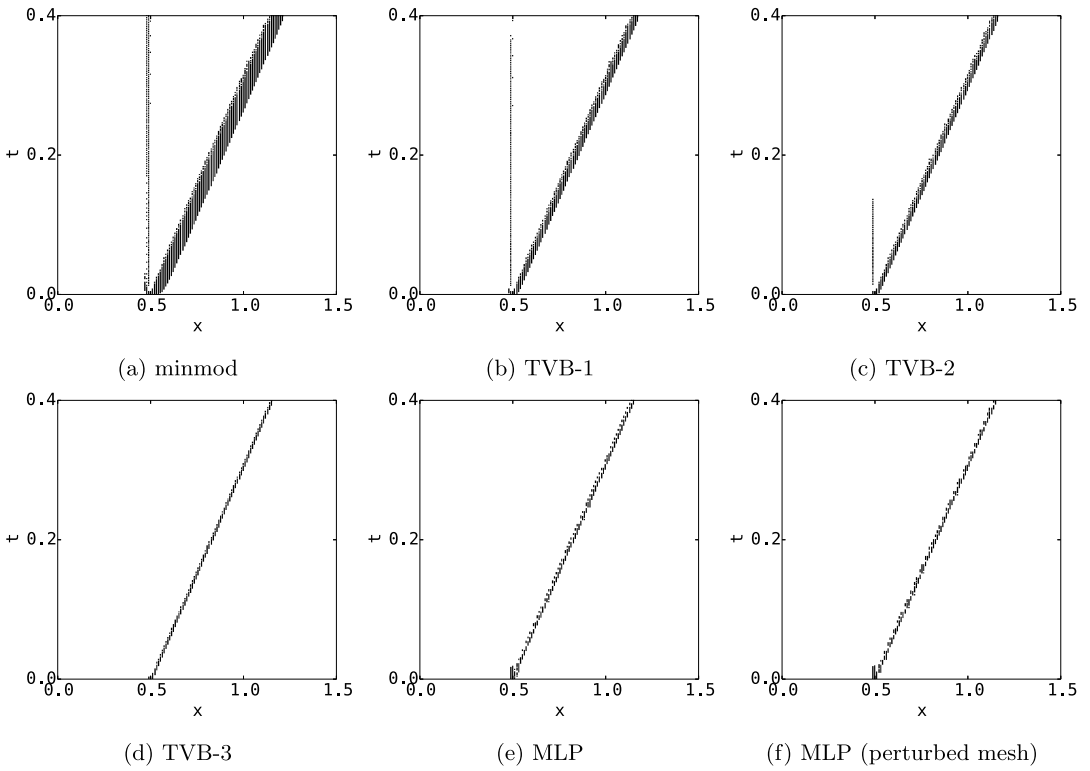


Fig. 17. The time-history of the Riemann problem for the Buckley–Leverett equation, simulated until $T = 0.4$ with $N = 150$ cells and $r = 4$. The plots (a)–(e) are obtained on a uniform mesh, while (f) is obtained on a randomly perturbed mesh.

$$D_0(x) = \begin{cases} 3 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}, \qquad u_0(x) = 0, \qquad g = 1,$$

on a mesh with $N = 100$ cells until the time $T = 1$ with CFL $= 0.4$. The primitive variables D and u are chosen as the indicator-variables. The results in Fig. 18 show that TVB-2 and TVB-3 lead to solutions contaminated with low amplitude oscillations, making them unsuitable for the current problem. While TVB-2 marks an insufficient number of troubled-cells (see 19(c)), TVB-3 does not flag any cell. This once again demonstrates the issue of choosing the TVB parameter M. The solutions with the minmod, TVB-1 and MLP indicators are non-oscillatory and indistinguishable, with the MLP marking the least number of cells among the three (see Fig. 19). Randomly perturbing the mesh does not alter the performance of the MLP indicator, as shown in Figs. 18(d) and 19(e).

5.5. Euler equations

We consider the one-dimensional Euler equations

$$\frac{\partial}{\partial t}\begin{bmatrix} \rho \\ \rho u \\ E \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} \rho u \\ p + \rho u^2 \\ (E + p)u \end{bmatrix},$$

where ρ, u, p represents the fluid density, velocity and pressure, respectively. The quantity E is the total energy per unit volume

$$E = \rho\left(\frac{u^2}{2} + e\right),$$

where e is the specific internal energy given by a caloric equation of state, $e = e(\rho, p)$. We chose the equation of state for ideal gas given by

$$e = \frac{p}{(\gamma - 1)\rho},$$

with $\gamma = c_p/c_v$ denoting the ratio of specific heats. We set $\gamma = 1.4$ for all the test cases below.

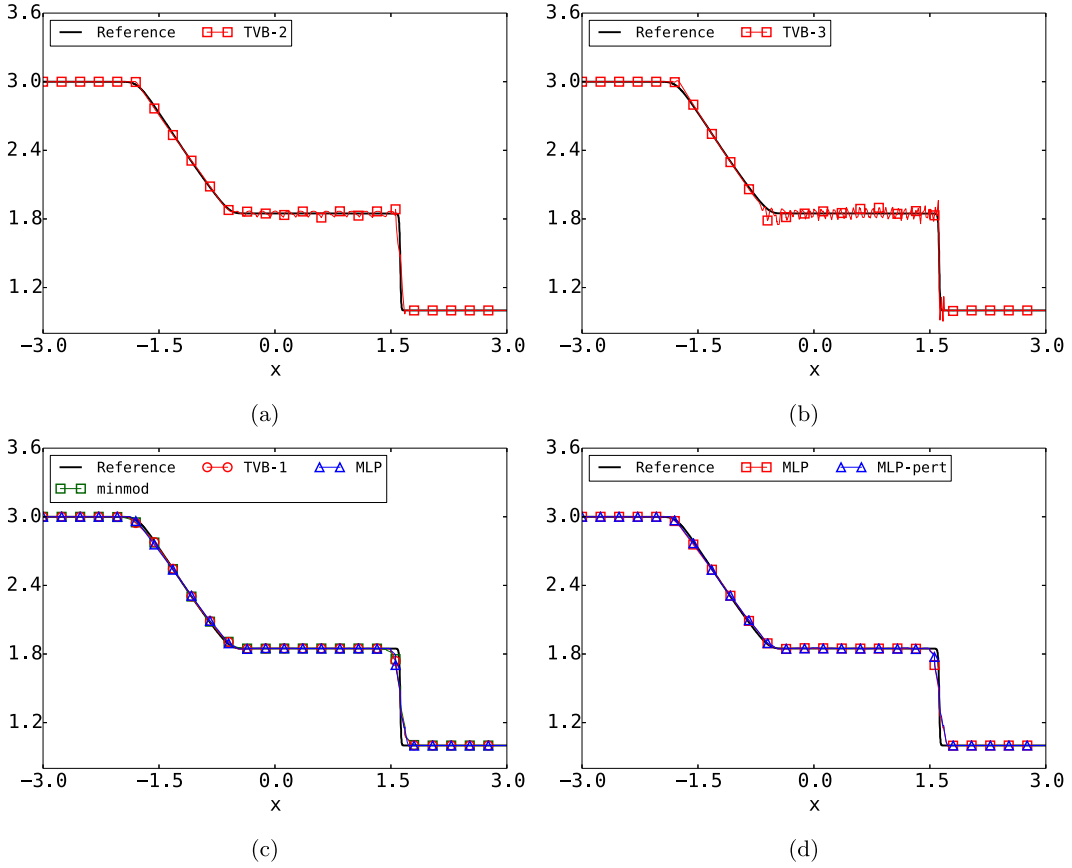


Fig. 18. Solution of the dam-break problem for the shallow water equations, computed at $T = 1$ with $N = 100$ cells and $r = 4$. (a) the oscillatory solution obtained with TVB-2; (b) the oscillatory solution obtained with TVB-3; (c) the solution with the remaining indicators; (d) comparison of the solution with the MLP indicator on a uniform mesh and a randomly perturbed mesh.

Based on the numerical results in Sections 5.1–5.4, we only compare the performance of the TVB and MLP indicators with the Euler equations, since the solutions with the minmod indicator are both diffusive and expensive. The primitive variables ρ, u, p are chosen as the limited-variables. The time-step is evaluated by setting CFL $= 0.4$, except when indicated otherwise.

We have also simulated the various Euler tests with the MLP indicator on a randomly perturbed mesh (using (5.1)) and found them to be very similar to the results obtained on the uniform mesh. Thus, we refrain from presenting them in this paper.

5.5.1. Sod test

This problem describes a mild shock tube test proposed by Sod [43], whose initial condition is given by

$$(\rho,\ u,\ p) = \begin{cases} (1,\ 0,\ 1) & \text{if } x < 0 \\ (0.125,\ 0,\ 0.1) & \text{if } x > 0 \end{cases}, \qquad x \in [-1, 1].$$

The solution is simulated on a mesh with $N = 100$ cells, until the time $T = 2$. The simulations with TVB-2 for $r = 1$ and TVB-3 for $r = 1, 2, 3, 4$ fail due to loss of positivity of density. Furthermore, TVB-2 leads to highly oscillatory results for $r = 2, 3, 4$, with the results for $r = 4$ shown in Fig. 20. This is substantiated by the fact that an insufficient number of cells are flagged by TVB-2 (see Fig. 21). The solutions with TVB-1 and MLP are indistinguishable, with the latter marking a few more cells as compared to the former.

5.5.2. Lax test

We consider the Lax shock tube problem [44], whose initial condition is given by

$$(\rho,\ u,\ p) = \begin{cases} (0.445,\ 0.698,\ 3.528) & \text{if } x < 0 \\ (0.5,\ 0,\ 0.571) & \text{if } x > 0 \end{cases}, \qquad x \in [-5, 5].$$

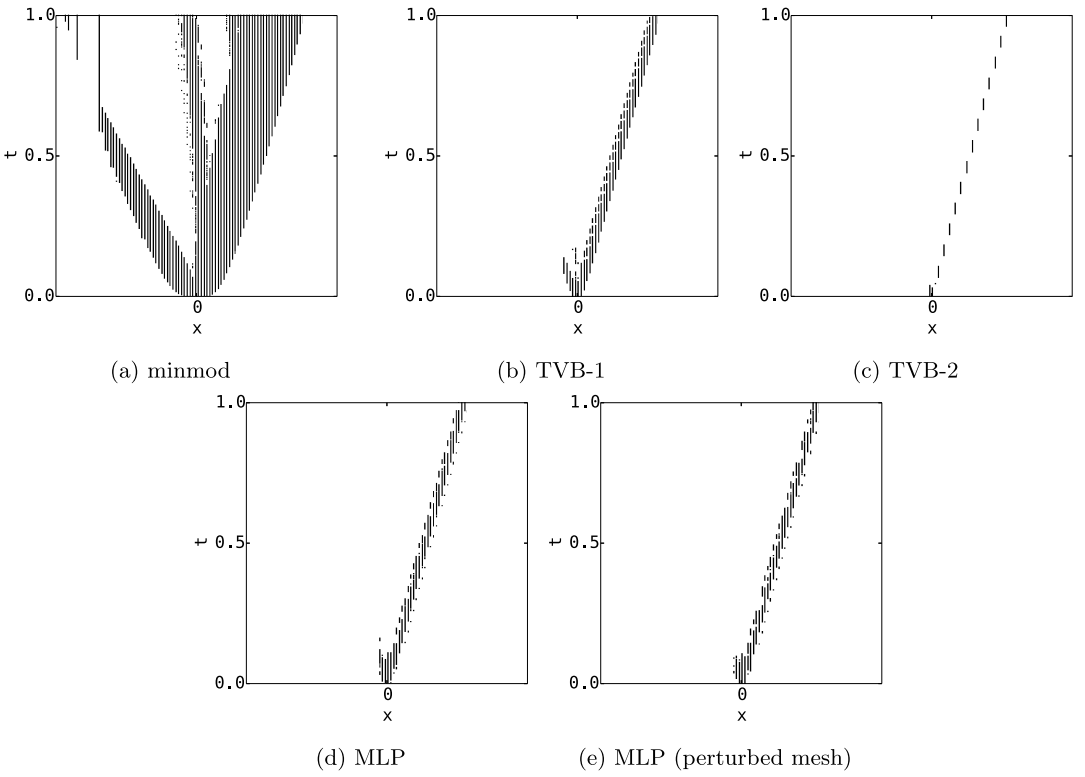


Fig. 19. The time-history of flagged troubled-cells of the dam-break problem for the shallow water equations, simulated until $T = 1$ with $N = 100$ cells and $r = 4$. The plots (a)–(d) are obtained on a uniform mesh, while (e) is obtained on a randomly perturbed mesh.

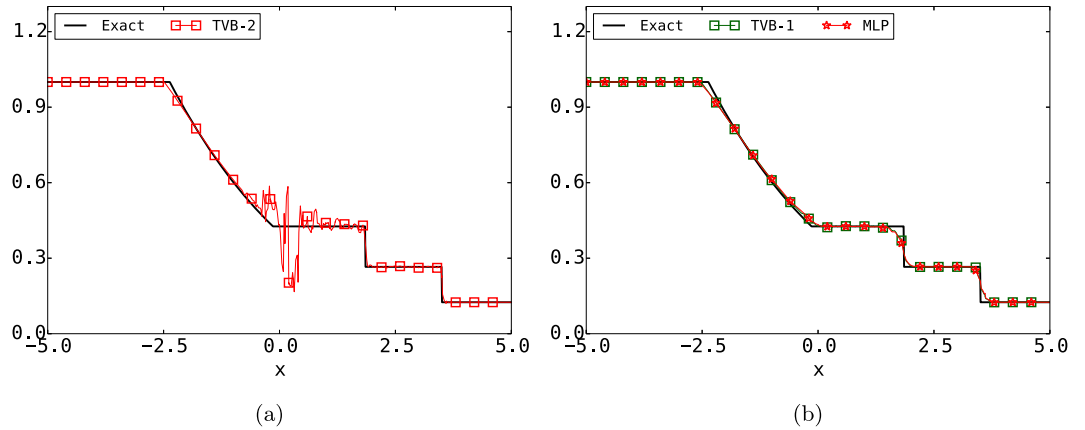


Fig. 20. Solution (density) of the Sod test for the Euler equations, computed at $T = 2$ with $N = 100$ cells and $r = 4$.

The solution is simulated on a mesh with $N = 200$ cells, until the time $T = 1.3$. The simulations with TVB-3 fail due to loss of positivity of density. While TVB-2 leads to the most accurate solution for $r = 4$, it gives the largest oscillation near to contact discontinuity for $r = 1$, as shown in Fig. 22. On the other hand, the MLP indicator gives much milder undershoots for $r = 1$, and lies between TVB-1 and TVB-2 in term of its resolution capabilities and the number of cells flagged as troubled-cells (see Fig. 23).

5.5.3. Double rarefaction

We consider the Riemann problem given by the following initial condition [45]

$$(\rho,\ u,\ p) = \begin{cases} (7,\ -1,\ 0.2) & \text{if } x < 0 \\ (7,\ 1,\ 0.2) & \text{if } x > 0 \end{cases}, \qquad x \in [-1, 1],$$

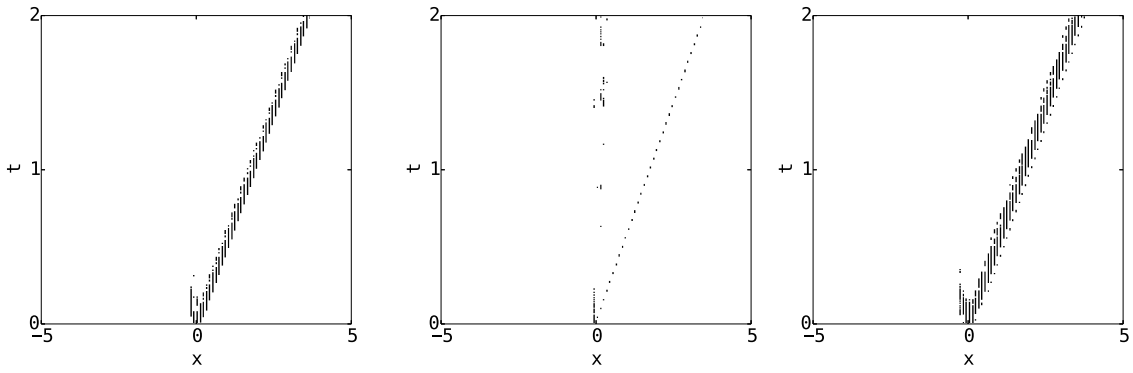


Fig. 21. The time-history of flagged troubled-cells of the Sod test for the Euler equations, computed at $T = 2$ with $N = 100$ cells and $r = 4$.

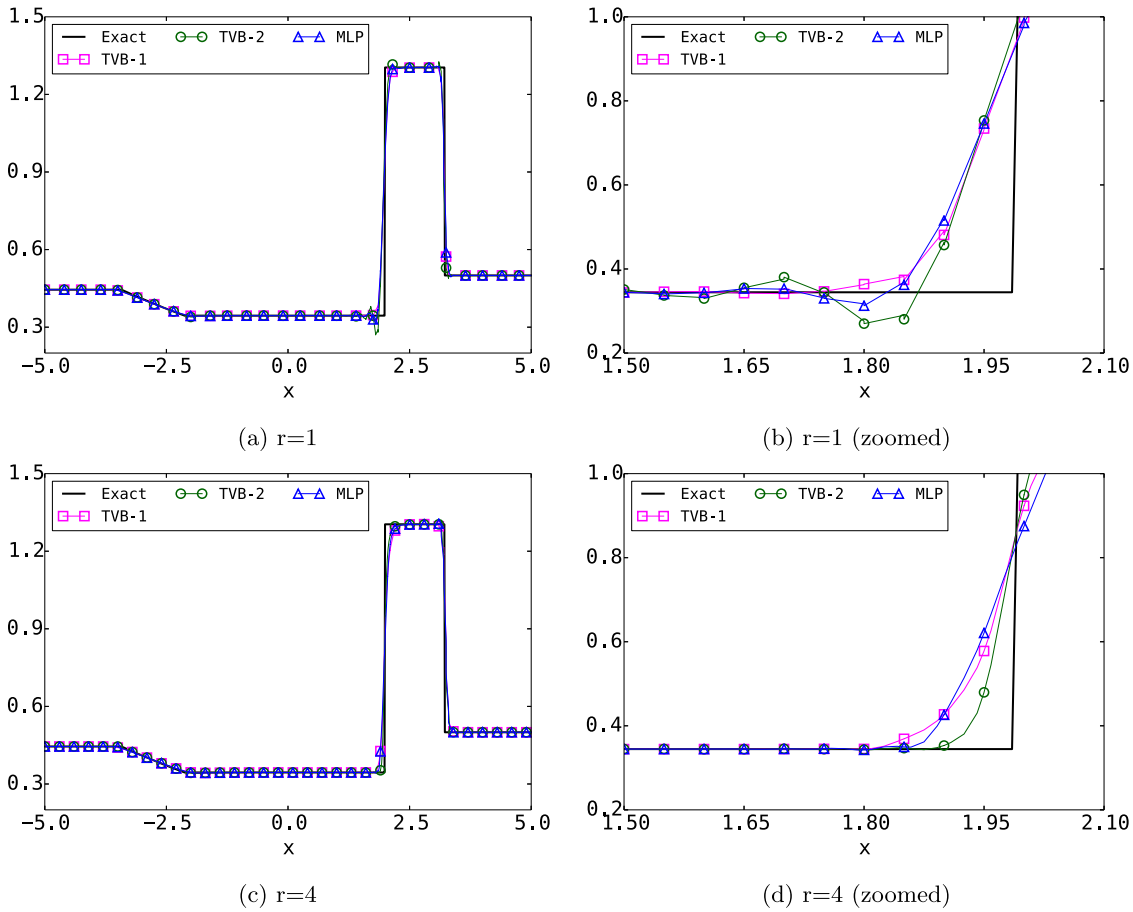
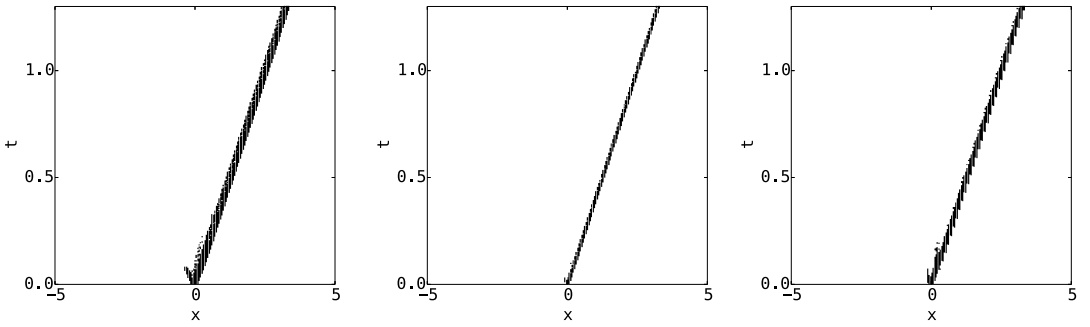


Fig. 22. Solution (density) of the Lax problem for the Euler equations, computed at $T = 1.3$ with $N = 200$ cells. (b) and (d) are zoomed near the contact discontinuity.

simulated on a mesh with $N = 128$ cells, until the time $T = 0.6$. The solution consists of two rarefaction waves pulling away from each other, creating a near-vacuum region at $x = 0$. This test case is used to assess the ability of numerical schemes to preserve the positivity of density and pressure. All three TVB indicators fail for $r = 1$. Additionally, TVB-3 fails for $r = 4$, and is observed to give rise to spurious oscillations near $x = 0$ for $r = 2, 3$ (not shown here). The solution with TVB-1, TVB-2 and MLP are indistinguishable for $r = 4$, as can be seen in Fig. 24. Furthermore, the results in Fig. 25 demonstrate the superiority of the MLP indicator, which marks the least number of cells among the three.



(a) TVB-1 (b) TVB-2 (c) MLP

Fig. 23. The time-history of flagged troubled-cells of the Lax problem for the Euler equations, computed at $T = 1.3$ with $N = 200$ cells and $r = 4$.

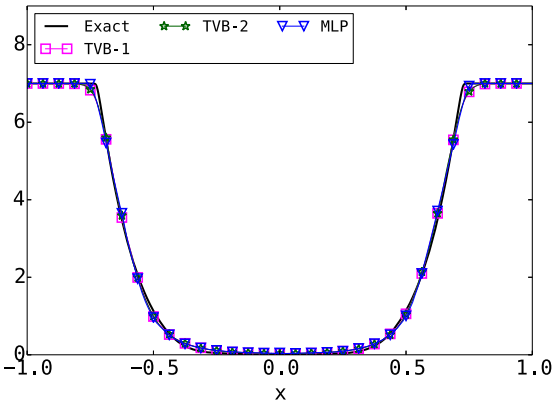
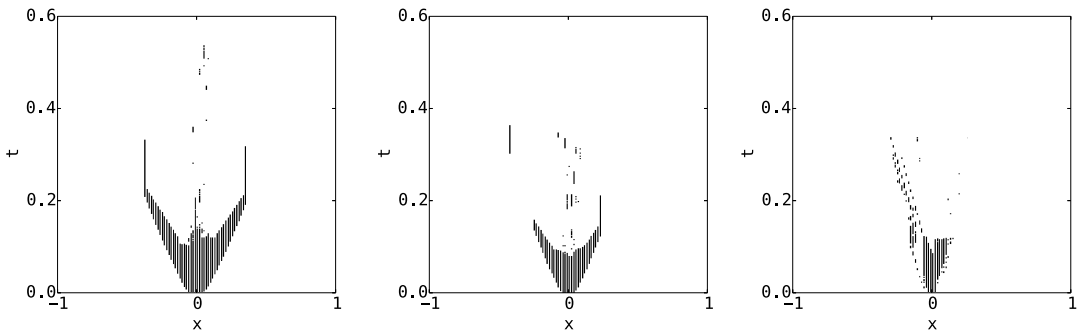


Fig. 24. Solution (density) of the double rarefaction problem for the Euler equations, computed at $T = 0.6$ with $N = 128$ cells and $r = 4$.



(a) TVB-1 (b) TVB-2 (c) MLP

Fig. 25. The time-history of flagged troubled-cells of the double rarefaction problem for the Euler equations, computed at $T = 0.6$ with $N = 128$ cells and $r = 4$.

5.5.4. Shock-entropy test

This test case proposed in [46], describes the interaction of a right moving shock with an oscillatory smooth wave. Its initial condition is given by

$$(\rho,\ u,\ p) = \begin{cases} (3.857143,\ 2.629369,\ 10.33333) & \text{if } x < -4 \\ (1 + 0.2\sin(5x),\ 0,\ 1) & \text{if } x > -4 \end{cases}, \qquad x \in [-5, 5].$$

The solution is simulated on a mesh with $N = 256$ cells, until the time $T = 1.8$ and CFL $= 0.1$. While TVB-3 gives the most accurate results for $r = 4$, especially in the smooth high-frequency region of the solution (see Fig. 26), we note that it fails to preserve positivity of density for $r = 1$. The solution is very dissipative with TVB-1, while the solutions with TVB-2 and MLP are equally well resolved. However, the MLP indicator marks fewer cells than TVB-2, as shown in Fig. 27.

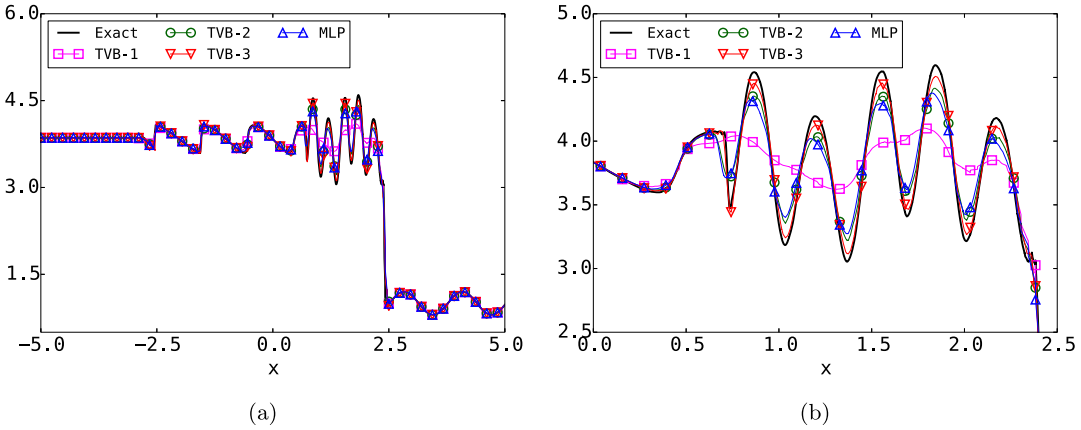


Fig. 26. Solution (density) for the shock-entropy problem, computed at $T = 1.8$ with $N = 256$ cells and $r = 4$. (b) zoomed near the smooth high-frequency region of the solution.

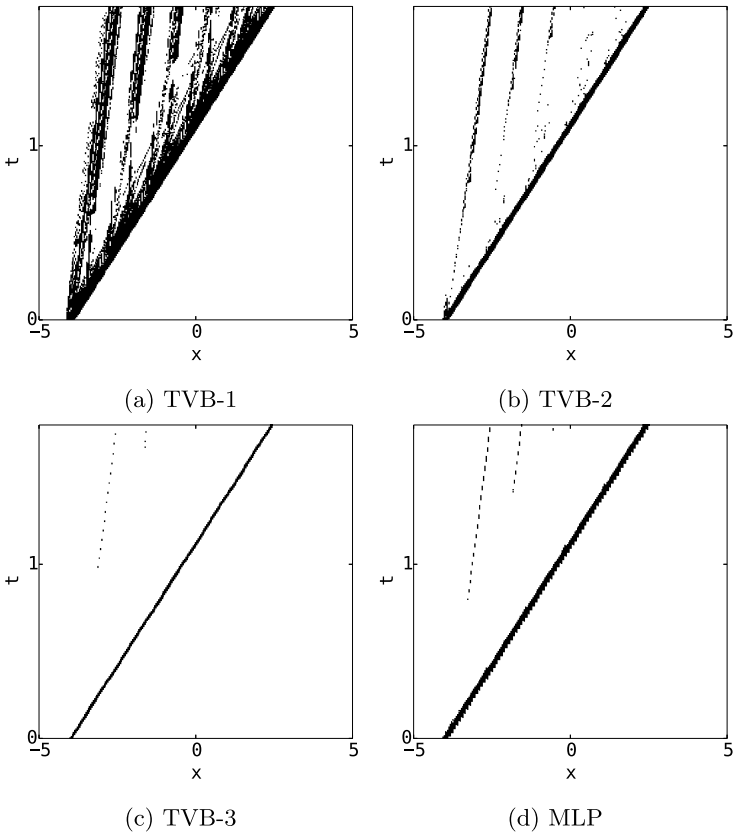


Fig. 27. The time-history of flagged troubled-cells for the shock-entropy problem, computed at $T = 1.8$ with $N = 256$ cells and $r = 4$.

5.5.5. Left half of the blast-wave

We consider a severe test case, describing the left half of the blast-wave problem [47]. The initial condition is given by

$$(\rho,\ u,\ p) = \begin{cases} (1,\ 0,\ 1000) & \text{if } x < 0.5 \\ (1,\ 0,\ 0.01) & \text{if } x > 0.5 \end{cases}, \qquad x \in [0, 1],$$

and is simulated on a mesh with $N = 256$ cells, until the time $T = 0.012$. The solution with TVB-3 fails for $r = 1$. The results for this problem show the most significant improvement with the MLP indicator, which leads to the best approximation of

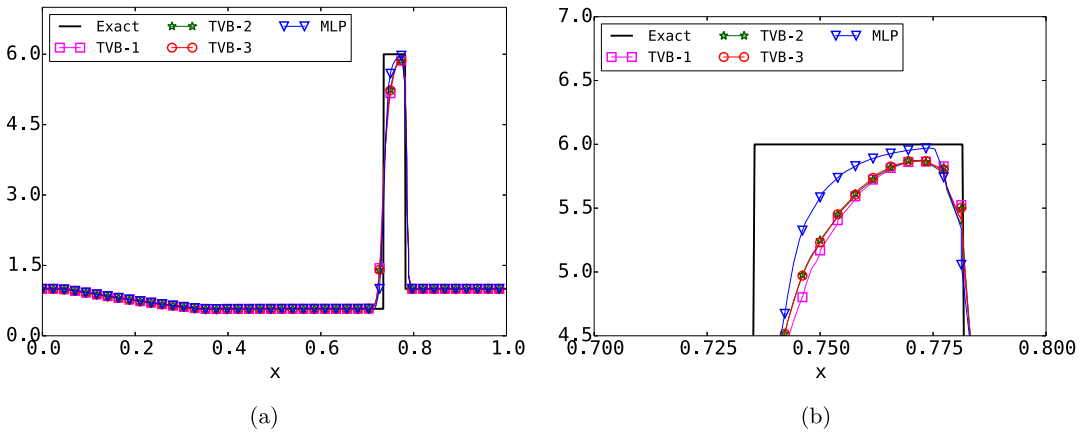


Fig. 28. Solution (density) for the left half of the blast wave problem, computed at $T = 0.012$ with $N = 256$ cells and $r = 4$. (b) the solution zoomed near the wave peak.

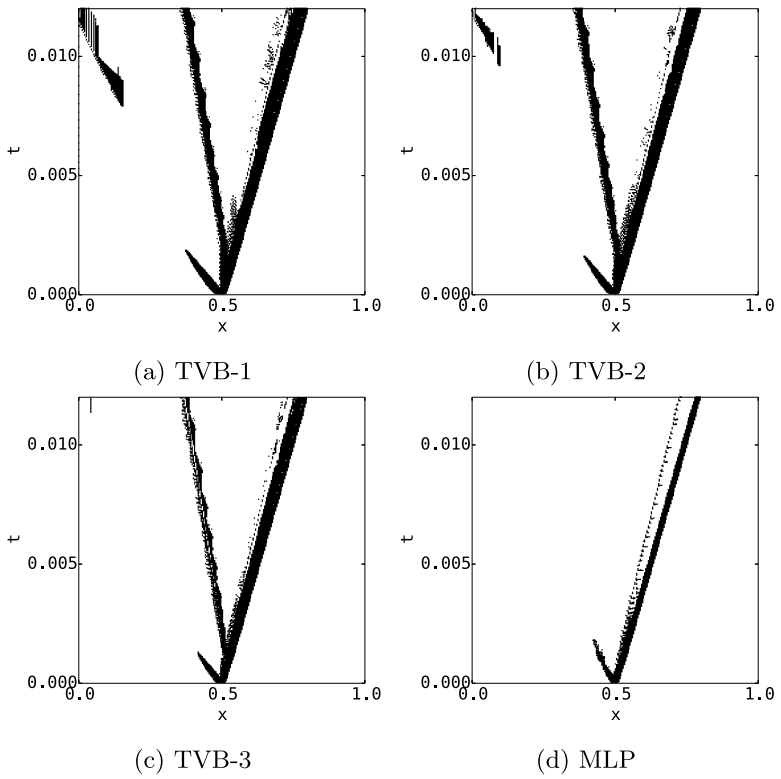


Fig. 29. The time-history of flagged troubled-cells for the left half of the blast wave problem, computed at $T = 0.012$ with $N = 256$ cells and $r = 4$.

the wave peak, as shown in Fig. 28. Furthermore, the results in Fig. 29 show that the MLP marks the fewest number of cells among all the indicators. Unlike the TVB indicators, the MLP only flags the cells close the shock and contact discontinuities (barring the first few time steps), which seems to be sufficient to avoid the appearance of spurious oscillations.

Remark 5.1. The limiting of characteristic variables can lead a loss of positivity of density and/or pressure, which explains why the TVB limiter failed in the above experiments, especially for cases when a sufficient number of cells were flagged. The MLP indicator did not face this issue for any of the Euler test cases considered in this paper. However, it might be useful to add a positivity preserving limiter [48,49] for more complex test cases.

6. Conclusion

In this work, we propose a new approach to tackle the issue of troubled-cell detection, by designing an MLP. The network is trained offline on a robust dataset consisting of canonical samples characterizing the local solution structure of conservation laws. The final trained network is used as a black-box troubled-cell indicator in a RKDG solver. An advantage of the proposed MLP indicator over traditional troubled-cell indicators, is that it does not depend on any tunable problem-dependent parameters.

The proposed indicator has been successfully tested on both scalar and systems of conservations laws in one-dimension. The numerical results presented in this paper demonstrate the ability of the network to try and mimic the behavior of the TVB limiter with an optimally chosen parameter M for a given problem. This ensures that the network correctly classifies cells with smooth extrema as good-cells. Since the network attempts not to flag more troubled-cells than necessary, the computational cost of the simulations can be significantly reduced. We have also demonstrated that despite being trained on uniform grids, the MLP indicator is able to perform well on grids obtained by randomly perturbing uniform meshes.

In the present work, we have restricted our focus to one-dimensional conservation laws. Future work will investigate the performance of similar MLP-based indicators trained for multi-dimensional systems.

References

[1] C.M. Dafermos, Hyperbolic Conservation Laws in Continuum Physics, third ed., Grundlehren der Mathematischen Wissenschaften (Fundamental Principles of Mathematical Sciences), vol. 325, Springer-Verlag, Berlin, 2010.
[2] B. van Leer, Towards the ultimate conservative difference scheme, V: a second-order sequel to Godunov's method, J. Comput. Phys. 32 (1979) 101–136, https://doi.org/10.1016/0021-9991(79)90145-1.
[3] B. Cockburn, S.-Y. Lin, C.-W. Shu, TVB Runge–Kutta local projection discontinuous Galerkin finite element method for conservation laws, III: one-dimensional systems, J. Comput. Phys. 84 (1989) 90–113, https://doi.org/10.1016/0021-9991(89)90183-6, http://www.sciencedirect.com/science/article/pii/0021999189901836.
[4] B. Cockburn, C.-W. Shu, The Runge–Kutta discontinuous Galerkin method for conservation laws, V: multidimensional systems, J. Comput. Phys. 141 (1998) 199–224, https://doi.org/10.1006/jcph.1998.5892, http://www.sciencedirect.com/science/article/pii/S0021999198958922.
[5] A. Harten, B. Engquist, S. Osher, S.R. Chakravarthy, Uniformly high order accurate essentially non-oscillatory schemes, III, J. Comput. Phys. 131 (1997) 3–47, https://doi.org/10.1006/jcph.1996.5632, http://www.sciencedirect.com/science/article/pii/S0021999196956326.
[6] X.-D. Liu, S. Osher, T. Chan, Weighted essentially non-oscillatory schemes, J. Comput. Phys. 115 (1994) 200–212, https://doi.org/10.1006/jcph.1994.1187, http://www.sciencedirect.com/science/article/pii/S0021999184711879.
[7] C.-W. Shu, Essentially Non-Oscillatory and Weighted Essentially Non-Oscillatory Schemes for Hyperbolic Conservation Laws, Springer, Berlin, Heidelberg, 1998, pp. 325–432.
[8] J.S. Hesthaven, T. Warburton, Nodal Discontinuous Galerkin Methods, Texts in Applied Mathematics, vol. 54, Springer, New York, 2008.
[9] A. Jameson, W. Schmidt, E. Turkel, Numerical Solution of the Euler Equations by Finite Volume Methods Using Runge–Kutta Time Stepping Schemes, AIAA Paper 81-1259, 1981.
[10] E. Tadmor, Entropy stability theory for difference approximations of nonlinear conservation laws and related time-dependent problems, Acta Numer. 512 (2004) 451–512.
[11] J. Qiu, C.-W. Shu, Runge–Kutta discontinuous Galerkin method using WENO limiters, SIAM J. Sci. Comput. 26 (2005) 907–929, https://doi.org/10.1137/S1064827503425298.
[12] J. Qiu, C.-W. Shu, A comparison of troubled-cell indicators for Runge–Kutta discontinuous Galerkin methods using weighted essentially nonoscillatory limiters, SIAM J. Sci. Comput. 27 (2005) 995–1013, https://doi.org/10.1137/04061372X.
[13] B. Cockburn, C.-W. Shu, TVB Runge–Kutta local projection discontinuous Galerkin finite element method for conservation laws, II: general framework, Math. Comput. 52 (1989) 411–435, https://doi.org/10.2307/2008474.
[14] M.J. Vuik, J.K. Ryan, Automated parameters for troubled-cell indicators using outlier detection, SIAM J. Sci. Comput. 38 (2016) A84–A104, https://doi.org/10.1137/15M1018393.
[15] J.W. Tukey, Exploratory Data Analysis, Addison–Wesley, 1977.
[16] Z. Gao, X. Wen, W.S. Don, Enhanced robustness of the hybrid compact-WENO finite difference scheme for hyperbolic conservation laws with multi-resolution analysis and Tukey's boxplot method, J. Sci. Comput. 73 (2017) 736–752, https://doi.org/10.1007/s10915-017-0465-0.
[17] G. Fu, C.-W. Shu, A new troubled-cell indicator for discontinuous Galerkin methods for hyperbolic conservation laws, J. Comput. Phys. 347 (2017) 305–327, https://doi.org/10.1016/j.jcp.2017.06.046, http://www.sciencedirect.com/science/article/pii/S0021999117305004.
[18] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.
[19] P.D. Cristea, Application of neural networks, in: Image Processing and Visualization, Springer, Netherlands, Dordrecht, 2009, pp. 59–71.
[20] G. Dede, M.H. Sazli, Speech recognition with artificial neural networks, Digit. Signal Process. 20 (2010) 763–768, https://doi.org/10.1016/j.dsp.2009.10.004, http://www.sciencedirect.com/science/article/pii/S1051200409001821.
[21] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (1998) 987–1000, https://doi.org/10.1109/72.712178.
[22] S. Golak, A MLP Solver for First and Second Order Partial Differential Equations, Springer, Berlin, Heidelberg, 2007, pp. 789–797.
[23] K. Rudd, S. Ferrari, A constrained integration (cint) approach to solving partial differential equations using artificial neural networks, Neurocomputing 155 (2015) 277–285, https://doi.org/10.1016/j.neucom.2014.11.058, http://www.sciencedirect.com/science/article/pii/S092523121401652X.
[24] G. Cybenko, Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient, Technical Report, Department of Computer Science, Tufts University, Medford, MA, 1988.
[25] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314, https://doi.org/10.1007/BF02551274.
[26] N.J. Guliyev, V.E. Ismailov, A single hidden layer feedforward network with only one neuron in the hidden layer can approximate any univariate function, Neural Comput. 28 (2016) 1289–1304, https://doi.org/10.1162/NECO_a_00849.
[27] S. Gottlieb, C.-W. Shu, E. Tadmor, Strong stability-preserving high-order time discretization methods, SIAM Rev. 43 (2001) 89–112, https://doi.org/10.1137/S003614450036757X (electronic).
[28] B. Cockburn, C.-W. Shu, Runge–Kutta discontinuous Galerkin methods for convection-dominated problems, J. Sci. Comput. 16 (2001) 173–261, https://doi.org/10.1023/A:1012873910884.
[29] D. Kriesel, A brief introduction to neural networks, http://www.dkriesel.com, 2007.

[30] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (1943) 115–133, https://doi.org/10.1007/BF02478259.
[31] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117, https://doi.org/10.1016/j.neunet.2014.09.003, http://www.sciencedirect.com/science/article/pii/S0893608014002135.
[32] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: J. Fürnkranz, T. Joachims (Eds.), Proceedings of the 27th International Conference on Machine Learning, ICML-10, Omnipress, 2010, pp. 807–814, http://www.icml2010.org/papers/432.pdf.
[33] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012, pp. 1097–1105, http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf, 2012.
[34] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proceedings of the 30th International Conference on Machine Learning, 2013.
[35] S. Kullback, R.A. Leibler, On information and sufficiency, Ann. Math. Stat. 22 (1951) 79–86, https://doi.org/10.1214/aoms/1177729694.
[36] Y. Bengio, Practical Recommendations for Gradient-Based Training of Deep Architectures, Springer, Berlin, Heidelberg, 2012, pp. 437–478.
[37] S.J. Nowlan, G.E. Hinton, Simplifying neural networks by soft weight-sharing, Neural Comput. 4 (1992) 473–493, https://doi.org/10.1162/neco.1992.4.4.473.
[38] TensorFlow, Large-scale machine learning on heterogeneous systems, https://www.tensorflow.org/, 2015.
[39] M. Bartholomew-Biggs, S. Brown, B. Christianson, L. Dixon, Automatic differentiation of algorithms, J. Comput. Appl. Math. 124 (2000) 171–190, https://doi.org/10.1016/S0377-0427(00)00422-2, http://www.sciencedirect.com/science/article/pii/S0377042700004222.
[40] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, CoRR, arXiv:1412.6980, 2014.
[41] J. Qiu, C.-W. Shu, On the construction, comparison, and local characteristic decomposition for high-order central WENO schemes, J. Comput. Phys. 183 (2002) 187–209, https://doi.org/10.1006/jcph.2002.7191, http://www.sciencedirect.com/science/article/pii/S0021999102971913.
[42] R. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge Texts in Applied Mathematics, Cambridge University Press, 2002.
[43] G.A. Sod, A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws, J. Comput. Phys. 27 (1978) 1–31, https://doi.org/10.1016/0021-9991(78)90023-2, http://www.sciencedirect.com/science/article/pii/0021999178900232.
[44] P.D. Lax, Weak solutions of nonlinear hyperbolic equations and their numerical computation, Commun. Pure Appl. Math. 7 (1954) 159–193, https://doi.org/10.1002/cpa.3160070112.
[45] T. Linde, P. Roe, Robust Euler Codes, AIAA Paper-97-2098, 1987.
[46] C.-W. Shu, S. Osher, Efficient implementation of essentially non-oscillatory shock-capturing schemes, II, J. Comput. Phys. 83 (1989) 32–78, https://doi.org/10.1016/0021-9991(89)90222-2, http://www.sciencedirect.com/science/article/pii/0021999189902222.
[47] P. Woodward, P. Colella, The numerical simulation of two-dimensional fluid flow with strong shocks, J. Comput. Phys. 54 (1984) 115–173, https://doi.org/10.1016/0021-9991(84)90142-6, http://www.sciencedirect.com/science/article/pii/0021999184901426.
[48] X. Zhang, C.-W. Shu, On maximum-principle-satisfying high order schemes for scalar conservation laws, J. Comput. Phys. 229 (2010) 3091–3120, https://doi.org/10.1016/j.jcp.2009.12.030, http://www.sciencedirect.com/science/article/pii/S0021999109007165.
[49] X. Zhang, C.-W. Shu, On positivity-preserving high order discontinuous Galerkin schemes for compressible Euler equations on rectangular meshes, J. Comput. Phys. 229 (2010) 8918–8934, https://doi.org/10.1016/j.jcp.2010.08.016, http://www.sciencedirect.com/science/article/pii/S0021999110004535.