An artificial neural network as a troubled-cell indicator

Deep Ray[a,*], Jan S. Hesthaven[a]

[a]*MATHICSE, École Polytechnique Fédérale de Lausanne (EPFL)*
CH-1015 Lausanne, Switzerland

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.

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

1. Introduction

It is well know 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, 3, 4], adaptively choosing the stencil for reconstruction [5, 6, 7], or adding artificial diffusion [8, 9, 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

---

*Corresponding author
Email addresses: `deep.ray@epfl.ch` (Deep Ray), `jan.hesthaven@epfl.ch` (Jan S. Hesthaven)

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 indicator that flags genuine troubled-cells, and is independent of problem-dependent parameters. In this paper, we propose a new type of indicator enjoying both these properties, by constructing an *artificial neural network* (ANN) that serves as a troubled-cell indicator.

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* [14]. 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 identification [15] and speech recognition [16], 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 [17, 18, 19].

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 [20, 21]. The approximation capabilities of simple MLP networks have been studied quite extensively (see [22] 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-cased 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

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

2

where u is the conserved variable with the smooth flux $f(u)$. To approximate the solution of (2.1), the computational domain is discretized using N non-overlapping cells, with cell-interfaces $a = x_{\frac{1}{2}} < x_{\frac{3}{2}} < ... < x_{N+\frac{1}{2}} = b$. 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 $V_r^h = \{v \in L^2([a,b]) : v|_{I_i} \in P^r(I_i)\}$, where $P^r(I_i)$ is the space of polynomials with degree $d \leqslant r$ 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 $v_h \in V_r^h$,*

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

where $v_h(x^{\pm}) = \lim\limits_{\epsilon \downarrow 0} v_h(x \pm \epsilon)$ 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 $\{\phi_{ij}(x), \ j = 0, ..., r\}$ as

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

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

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

where $\mathbf{M}_{jk}^{(i)}$ 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

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

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

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

which is solved using a suitable time-marching scheme, such as the third-order strong stability preserving Runge-Kutta (SSP-RK3) scheme [23].

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

3

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 Figure 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 forward and backward differences,

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

A troubled-cell indicator uses 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:
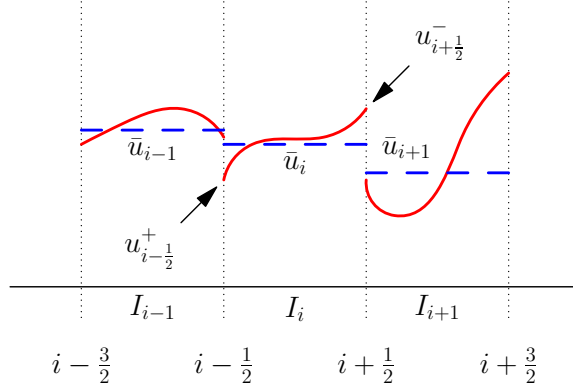

Figure 1: Stencil used by troubled-cell indicators. The polynomial approximation in each cell is denoted by solid line, while the corresponding cell-average values denoted by dashed lines.

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

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

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.

4

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

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

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 [24, 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 Figure 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 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 [25]. 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.
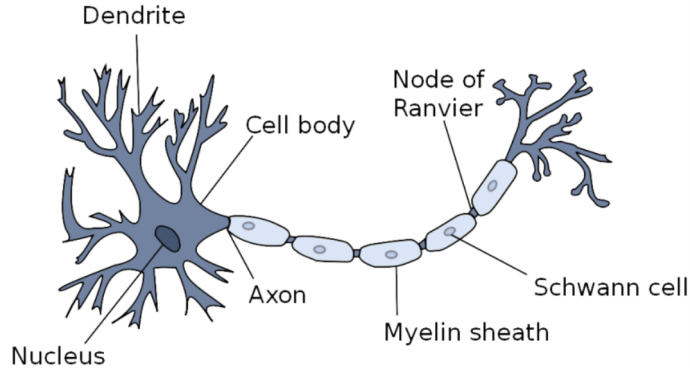

Figure 2: A simplified model of the biological neuron. This image has been taken from [25].

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 Figure 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 Heavyside function

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

which leads to the McCulloch-Pitts neuron model [26]. However, (3.2) is not preferred in practice since its derivative vanishes everywhere (see discussion in Section 3.3).
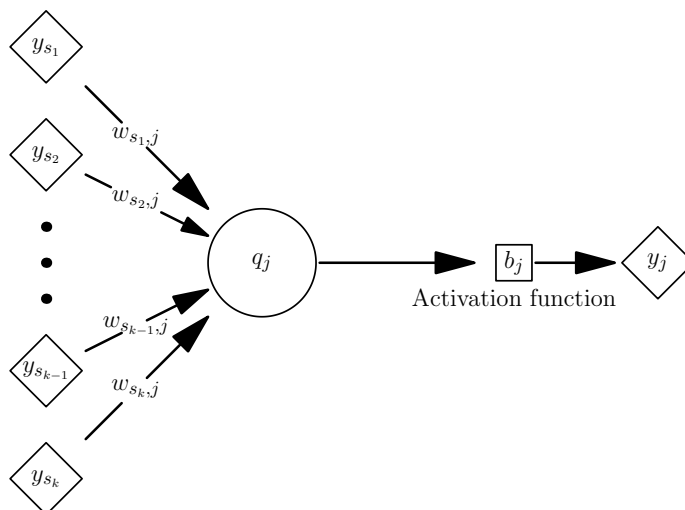
Figure 3: A single artificial neuron receiving signals from k neurons.

3.1. Multi-layer perceptron

Several architectures have been proposed for ANNs [25, 27], 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 Figure 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 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 input and output of the MLP matches the dimensions of the domain and range spaces of the function (3.1) being approximated.
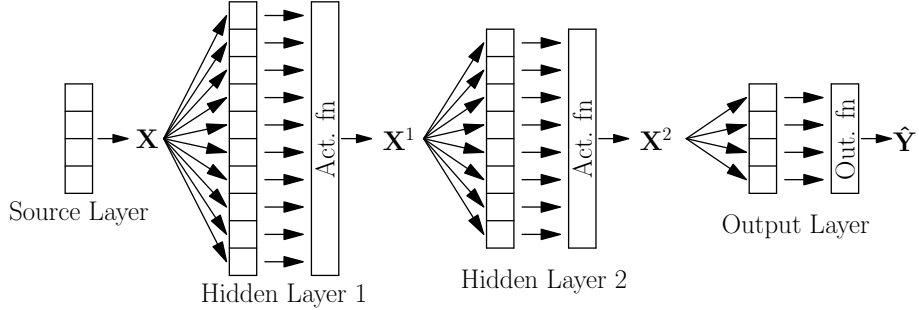

Figure 4: An MLP with 2 hidden layers. The input layer transmits the signal $\mathbf{X}$ to the first hidden layer. The final output of the network is $\hat{\mathbf{Y}}$, which is the prediction to the true output $\mathbf{Y}$ corresponding to $\mathbf{X}$.

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 later 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 [25]. 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 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 Heavyside function (3.2). To understand why this may not be a suitable choice, we note that most

8

optimization algorithms, such as gradient descent, update the weight of the connection between the neuron pair (i, j) iteratively as

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

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 Heavyside function. Thus, the updates in each iteration would be very small, leading to slow convergence of the algorithm. Other non-linear choices for the activation function include the logistic function and the hyperbolic tangent (see Figure 5), which are smooth functions capturing the qualitative features of the Heavyside 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) [28], shown in Figure 5(c). Training with ReLU is often faster, when compared to models using the logistic or hyperbolic tangent activation function [29]. 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 [30], which modifies the original ReLU by adding a small negative slope ν when $x < 0$, as shown in Figure 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-limiters discussed in Section 2, the input for our MLP is the vector $(\bar{u}_{i-1},\ \bar{u}_i,\ \bar{u}_{i+1},\ u_{i-\frac{1}{2}}^+,\ u_{i+\frac{1}{2}}^-)^\top \in \mathbb{R}^5$, 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 $\hat{\mathbf{Y}}$

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

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

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

(b) Hyperbolic tangent: $\frac{e^x - e^{-x}}{e^x + e^{-x}}$

(c) ReLU: $\max(0, x)$

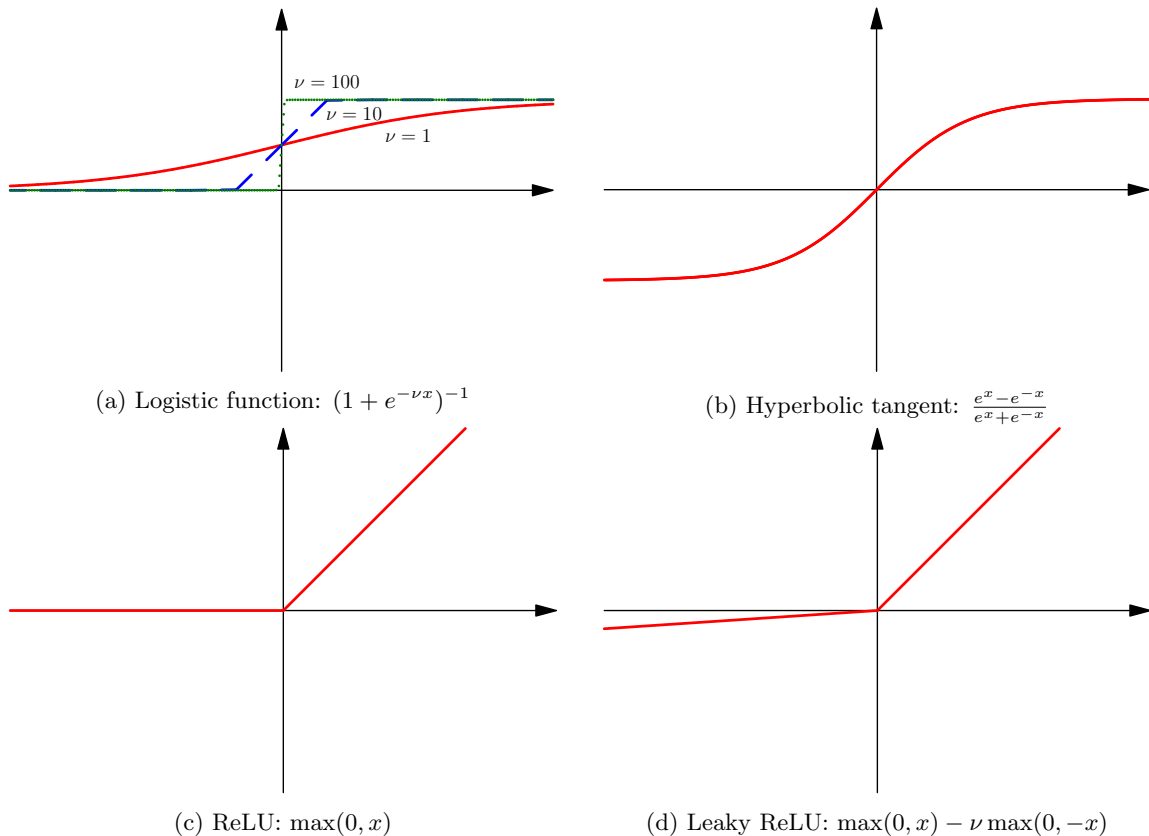(d) Leaky ReLU: $\max(0, x) - \nu \max(0, -x)$

Figure 5: Choices for the activation function used in the MLP network.

values 0 or 1. The cross-entropy function is closely related to the Kullback-Leibler divergence, which measures the discrepancy between two probability distributions [31]. 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 [32].

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 [33] by penalizing the weights $\mathbf{W}$ of the network

$$\widetilde{\mathcal{C}} = \mathcal{C} + \beta \|\mathbf{W}\|_2^2, \quad \beta \geqslant 0, \tag{4.1}$$

10

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 approach commonly used, which makes use of a *validation* data set $\mathbb{V}$ that is independent of the training set $\mathbb{T}$ [33]. 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 training is terminated once the validation accuracy starts decreasing, as it indicates that the MLP is overfitting the data in $\mathbb{T}$. In order to train our MLP, we evaluate the accuracy of the MLP responses over the set $\mathbb{V}$ as

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

where $\hat{Y}^1$ is the probability of the cell being a troubled-cell.

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 $a \leqslant x_i - \frac{3}{2}h < x_i + \frac{3}{2}h \leqslant b$. Thus, we construct a 3-cell stencil centered at x_i, which is contained in the interval $[a, b]$ (see Figure 6).
3. In each cell, project the solution onto the space of polynomials of degree r. We use the Legendre polynomials for the projection.
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 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 the indicator is more robust in terms of capturing the discontinuities.*

5. Numerical results

We now demonstrate the capability of the MLP network when used as a troubled-cell indicator with a RKDG scheme. 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, wherein the training is terminated if the accuracy on the validation set decreases for 10
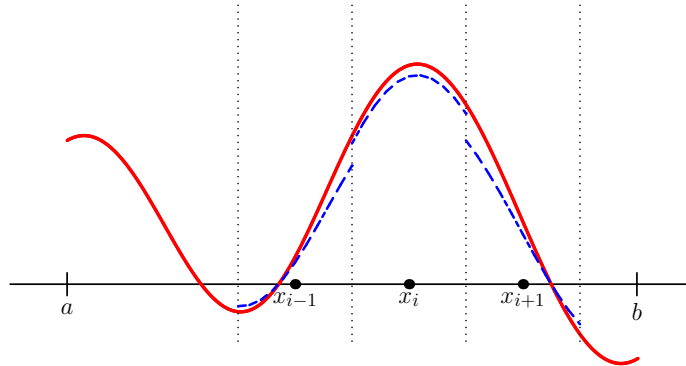
Figure 6: Construction of a training sample for a known function $u(x)$ (solid line) in the interval $[a, b]$. In each cell of the stencil, the function is approximated using a polynomial (dashed line).

consecutive epochs. The momentum based Adam stochastic optimizer [34] is used to minimize the cost function (4.1), with an initial learning rate $\eta = 10^{-3}$. The training and validation datasets are constructed using a number of functions, which are listed in Tables 1(a)-(b). 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.

At the beginning of the training, the weights and biases are randomly initialized using a normal distribution. 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 several times with different initializations of the weights and biases, and select the trained 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 [35].

The DG scheme is evaluated using the local Lax-Friedrichs flux. We compare the performance of the proposed MLP indicator with that of the minmod-limiter and the minmod-type TVB limiter. The notation TVB-1, TVB-2, and TVB-3 refer to the TVB limiter with the TVB constants $M = 10$, $M = 100$, 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.

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 CFL = 0.4.

u(x)	Domain	Additional parameters varied	Good cells	Troubled cells
$\sin(4\pi x)$	$[0,1]$	$-$	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**

(a) Functions used to create $\mathbb{T}$.

u(x)	Domain	Additional parameters varied	Good cells	Troubled cells
$\sum_{p=1}^{5} \sin(p\pi x)$	$[0,2]$	$-$	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**

(b) Functions used to create $\mathbb{V}$.

Table 1: Functions used to create the training and validation datasets. For each function, the mesh size h, the approximating polynomial degree r and additional parameters (if present) are varied. The last two columns 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.

5.1.1. Test 1

This test case corresponds to a smooth initial condition consisting of sine waves with three different frequencies

$$
u_0(x) = \begin{cases} \sin(4\pi x) & \text{if } x \leqslant 0.5 \text{ ,} \\ \sin(8\pi x) & \text{if } 0.5 < x \leqslant 0.75 \text{ ,} \\ \sin(16\pi x) & \text{if } x > 0.75 \text{ ,} \end{cases}
$$

on the domain $[0, 1]$ with periodic boundary conditions. At the end of the final time $T = 1$, the composite wave completes one full revolution and returns to its original position. The results with various troubled-cell indicators on a mesh with $N = 100$ cells are shown in Figure 7. The profiles with the minmod limiter and TVB-1 are almost identical, and lead to the most dissipative results due to truncation near smooth extrema. TVB-2 performs partially better than minmod. On the other hand, the MLP and TVB-3 give equally good results for the two lowest frequency waves, with negligible truncation near the extrema. However, both methods suffer from loss of accuracy near the extrema of the highest frequency wave, with TVB-3 performing marginally better. If we refine the mesh and consider $N = 150$, the results in Figure 8 indicate an improvement for all indicator functions, with the MLP clearly outperforming the other methods. Zooming into the solution profile near the last high-frequency peak (see Figure 9), shows that the MLP indicator is capable of correctly identifying the cells with smooth extrema if the mesh is able to resolve all scales, while TVB-3 may fail to do so.

5.1.2. Test 2

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

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

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 equally dissipative results, as can be seen in Figure 10. TVB-2 performs significantly better, especially near the last two peaks of the solution profile. Note that all TVB schemes and the MLP give rise to overshoots near the second peak, which is the most significant with polynomial degree $r = 1$. However, the overshoots are reduced as the polynomial degree is increased, as shown in Figure 11. Although TVB-3 is better at resolving the discontinuity, it also gives the largest overshoots. The MLP gives milder overshoots, with its resolution capabilities lying between TVB-2 and TVB-3.

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 CFL $= 0.4$.

(a) $r = 1$

(b) $r = 2$

(c) $r = 3$

(d) $r = 4$

Figure 7: Test 1 for linear advection, simulated until $T = 1$ with $N = 100$ cells and polynomial degree r.

15

(a) $r = 1$

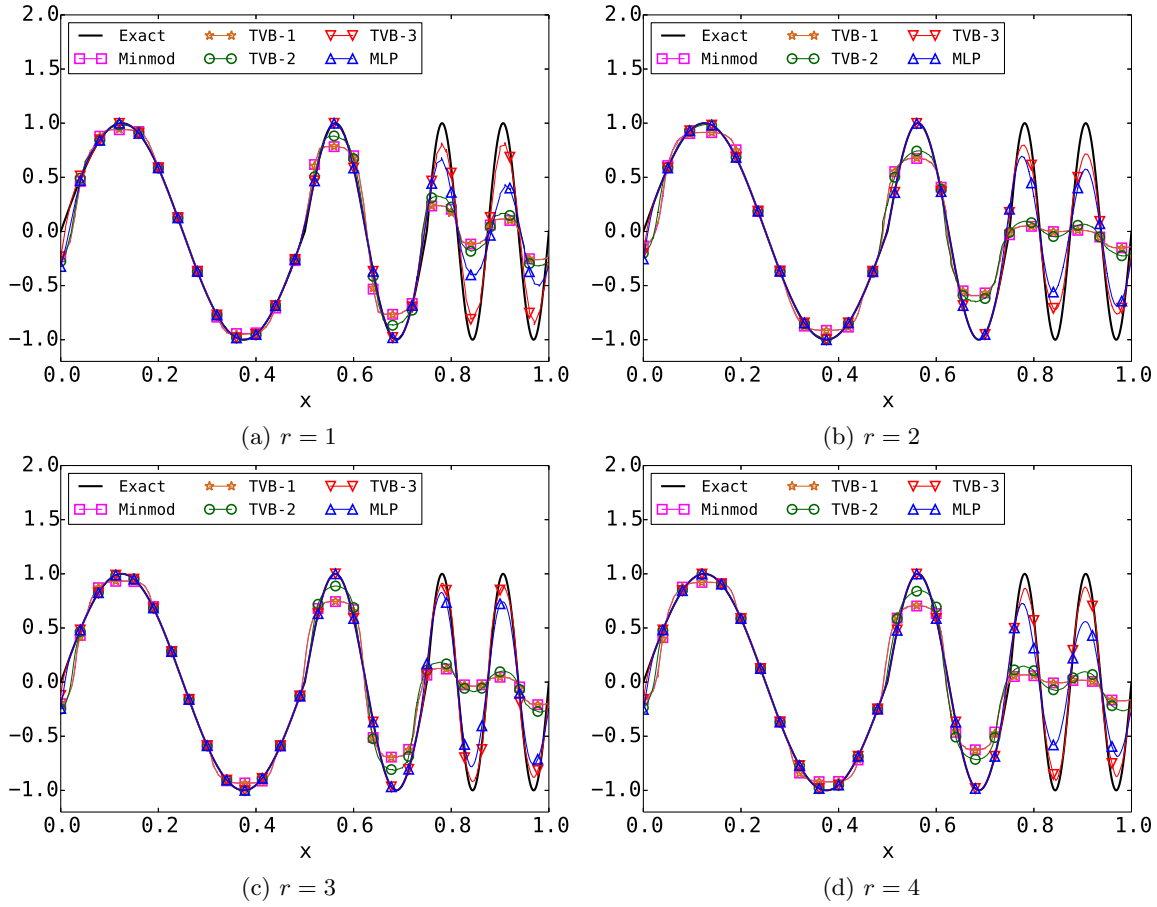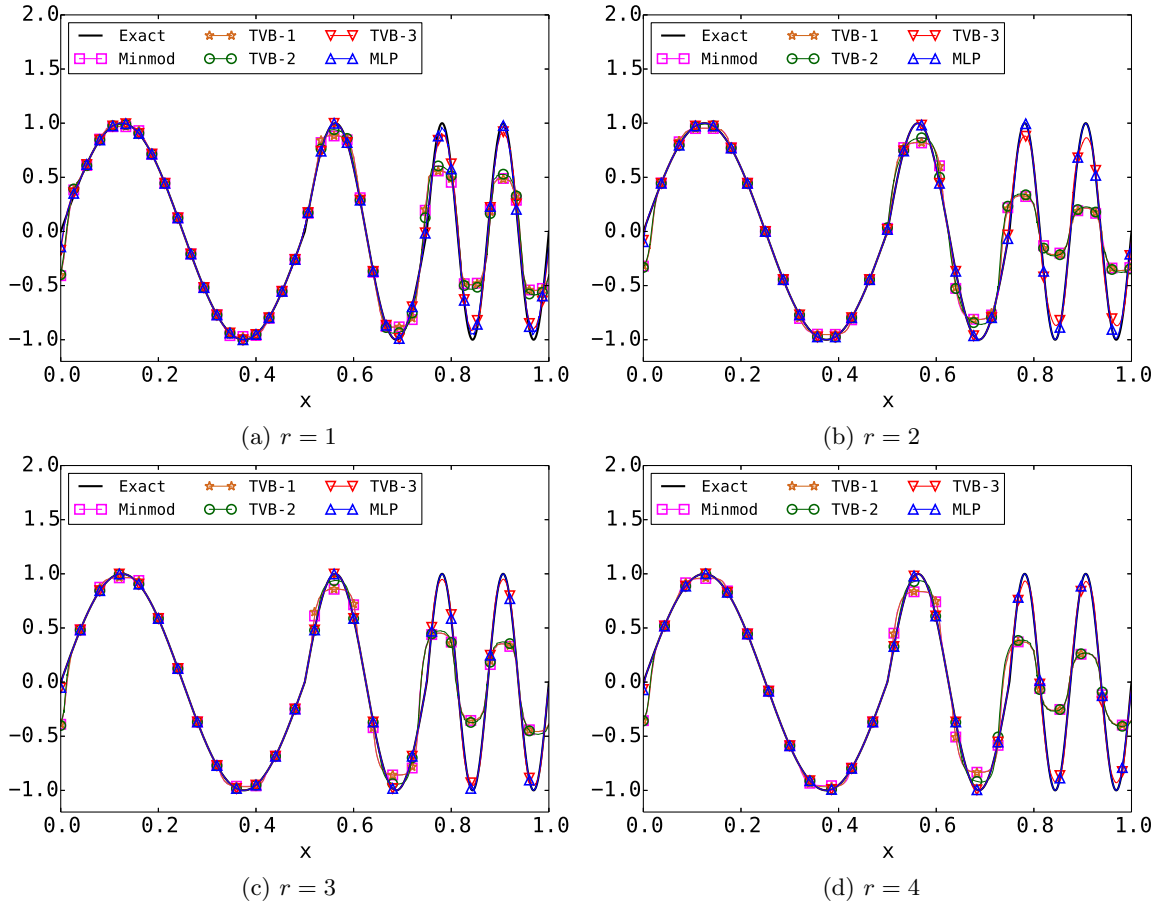(b) $r = 2$

(c) $r = 3$

(d) $r = 4$

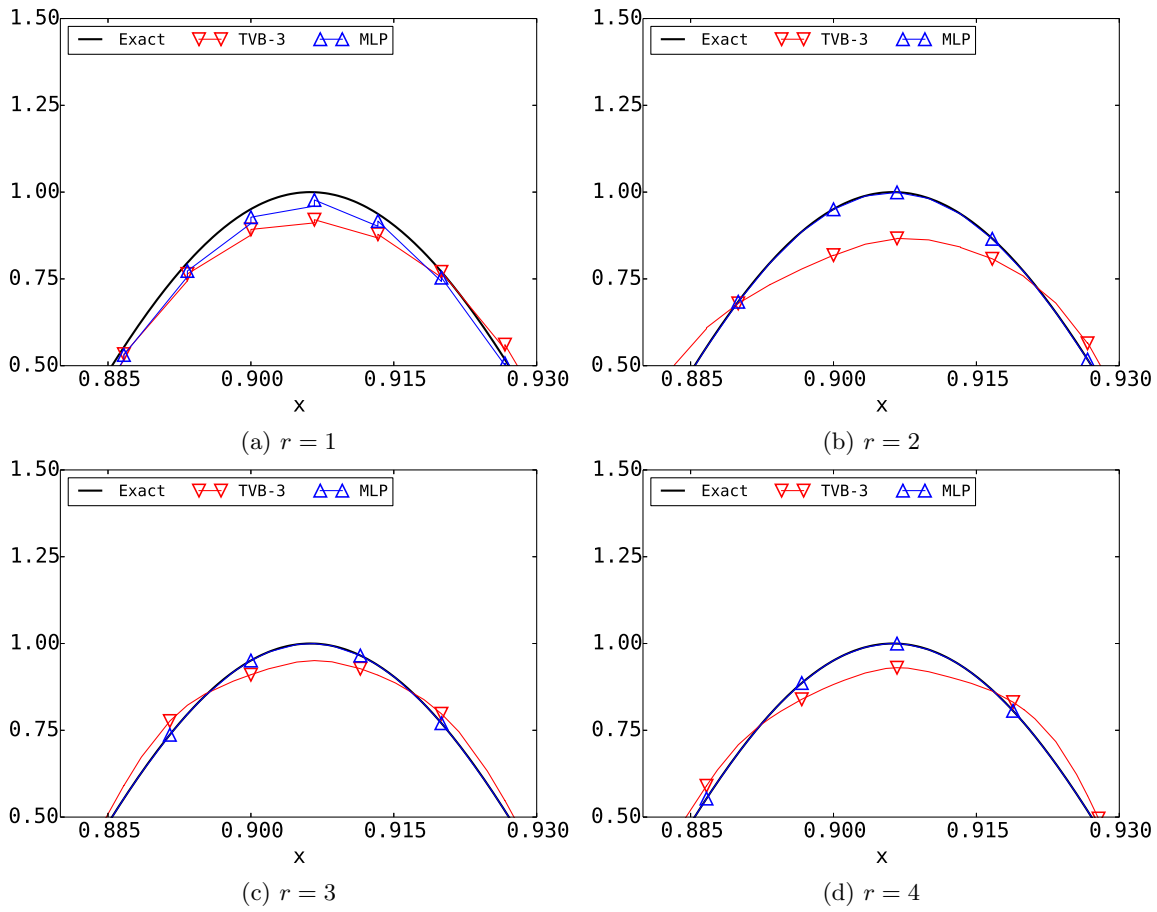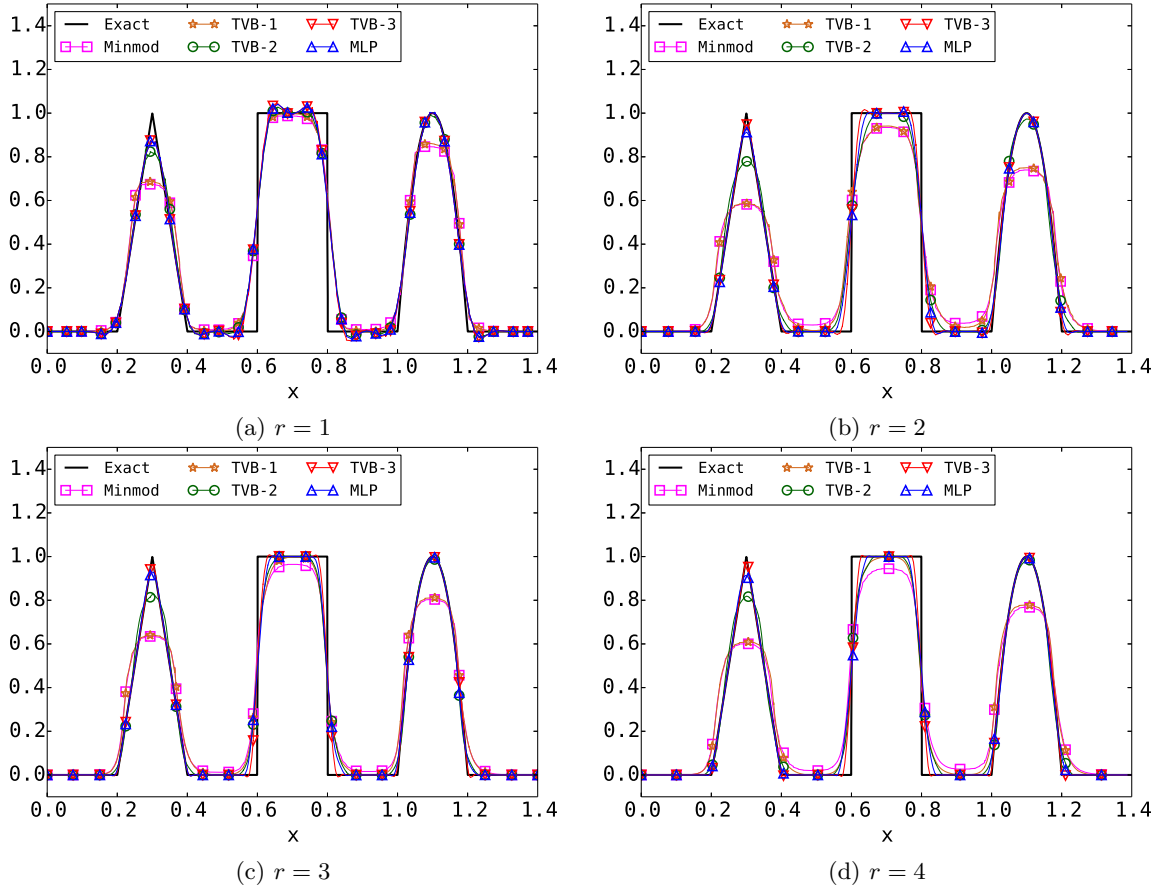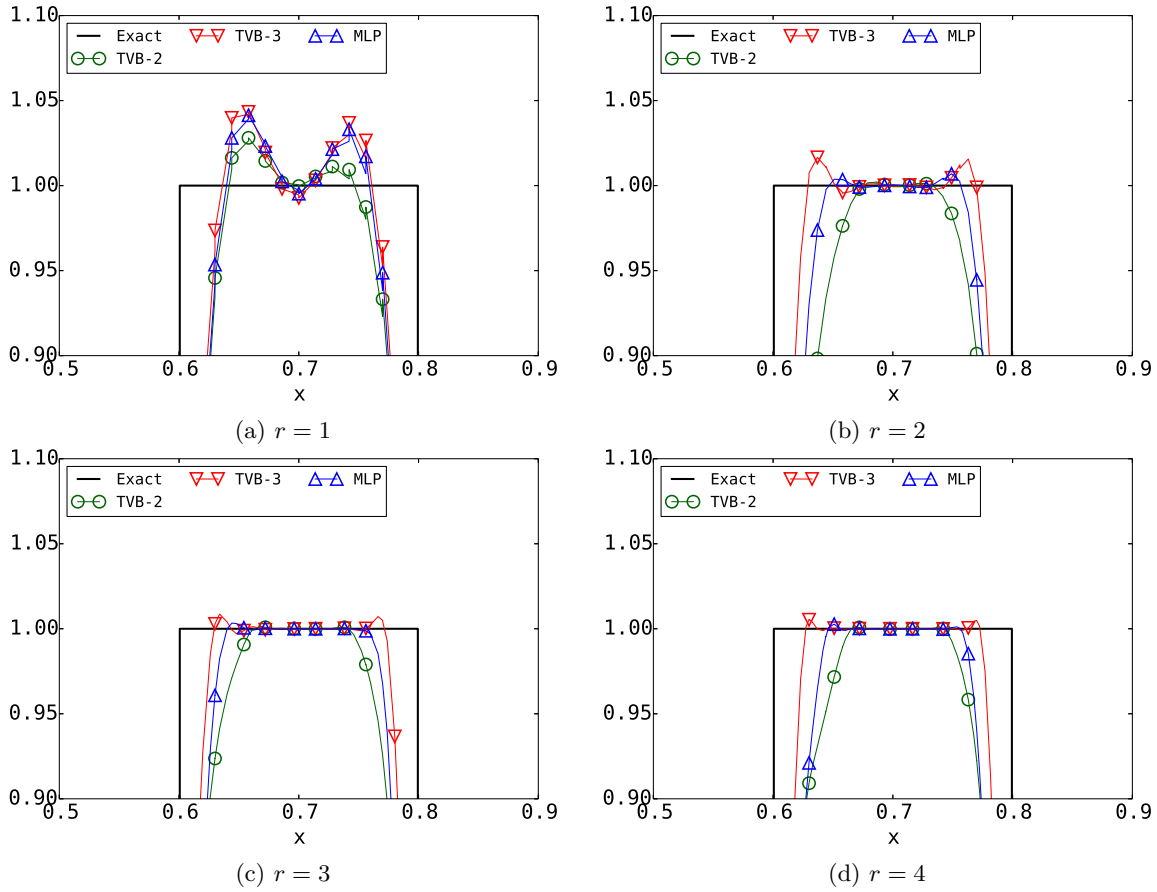Figure 8: Test 1 for linear advection, simulated until $T = 1$ with $N = 150$ cells and polynomial degree r.

Figure 9: Test 1 for linear advection, simulated until $T = 1$ with $N = 100$ cells and polynomial degree r. The plot is zoomed to focus on the last peak of the solution.

Figure 10: Test 2 for linear advection, simulated until $T = 1.4$ with $N = 100$ cells and polynomial degree r.

Figure 11: Test 2 for linear advection, simulated until $T = 1.4$ with $N = 100$ elements and polynomial degree r. The plot is zoomed to focus on the central step wave.

5.2.1. Test 1

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

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

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 are indistinguishable, as shown in Figure 12. To compare and assess the performance of the indicators, we plot the cells flagged on the mesh after each of the three Runge-Kutta steps, as time evolves. We notice from the results in Figure 13 that the minmod limiter marks the most cells. Comparatively, the TVB limiter has thinner zones of marked cells, which decays 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.
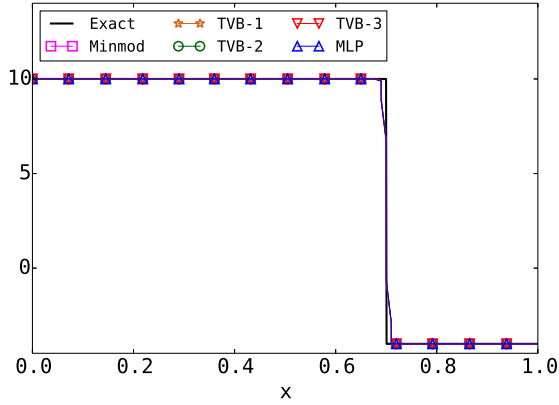

Figure 12: Test 2 for Burgers equation. Solutions at $T = 0.1$ with $N = 100$ cells and polynomial degree $r = 4$.

5.2.2. Test 2

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

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

on the domain $[0, 1]$ with periodic boundary conditions. The initial condition is plotted in Figure 14. As the solution evolves, alternating shock and rarefaction waves begin to develop. In all previous
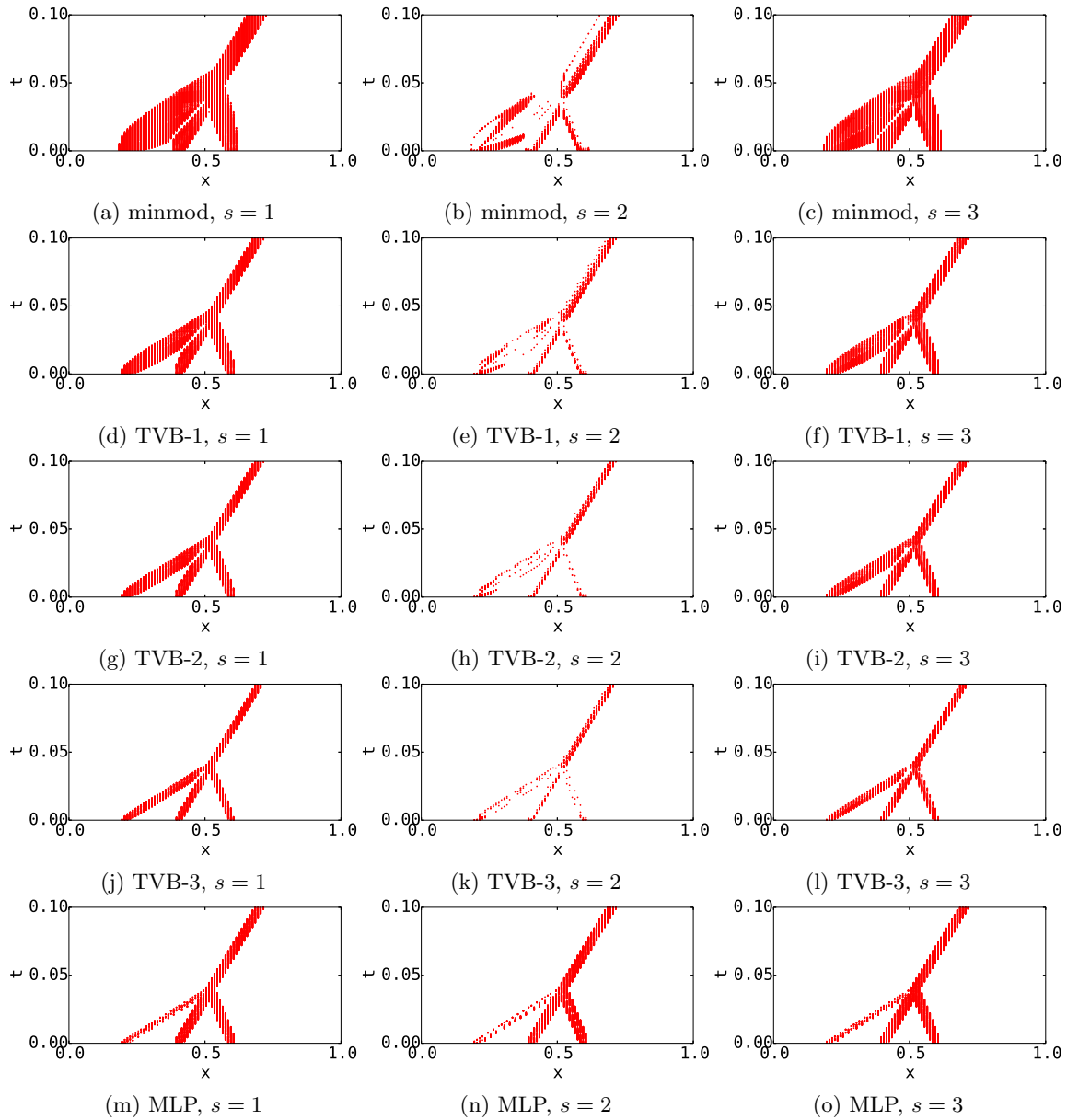
Figure 13: Test 1 for Burgers equation. The figures depict the cells in the mesh, flagged as troubled cells by various indicators as time evolves. The left, middle and right columns represents the cells flagged after the first RK stage (s=1), the second RK stage (s=2) and the third RK stage (s=3), respectively.

21

test problems, the solutions obtained with TVB-3 were superior to those obtained with TVB-1 and TVB-2. However, TVB-3 leads to large Gibbs oscillations near the discontinuities for the current test case, as shown in Figure 15. 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. We track the flagged cells as time evolves in Figure 16, which clearly shows that the minmod limiter marks most cells. The existence of Gibbs oscillations with TVB-3 is further supported by the fact that very few cells are flagged by the indicator. In terms of the number of cells flagged, the MLP lies between TVB-1 and TVB-2.
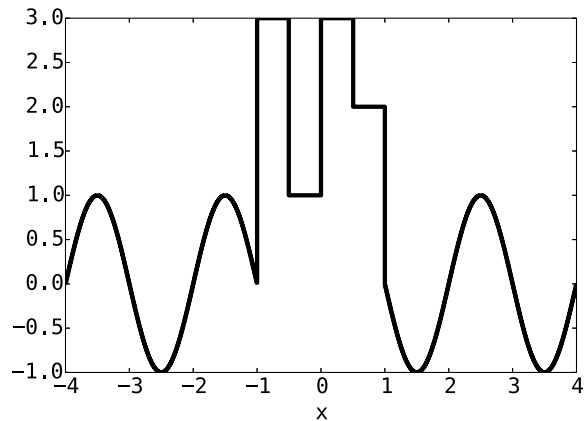

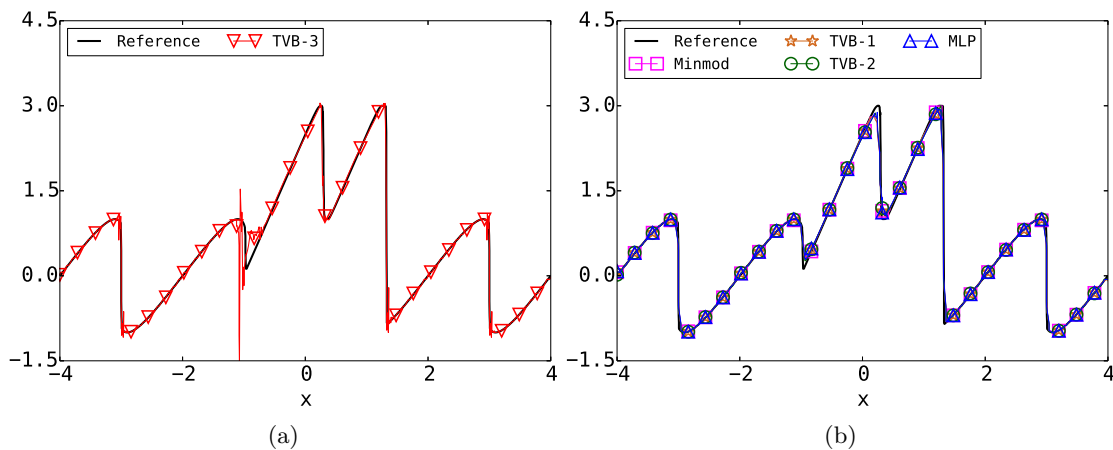Figure 14: Test 2 for Burgers equation, the initial solution profile.

Figure 15: Test 2 for Burgers equation, simulated until $T = 0.4$ with $N = 200$ cells and polynomial degree $r = 4$. The oscillatory solution with TVB-3 is shown in (a), while the solution with the remaining indicator functions is compared in (b).
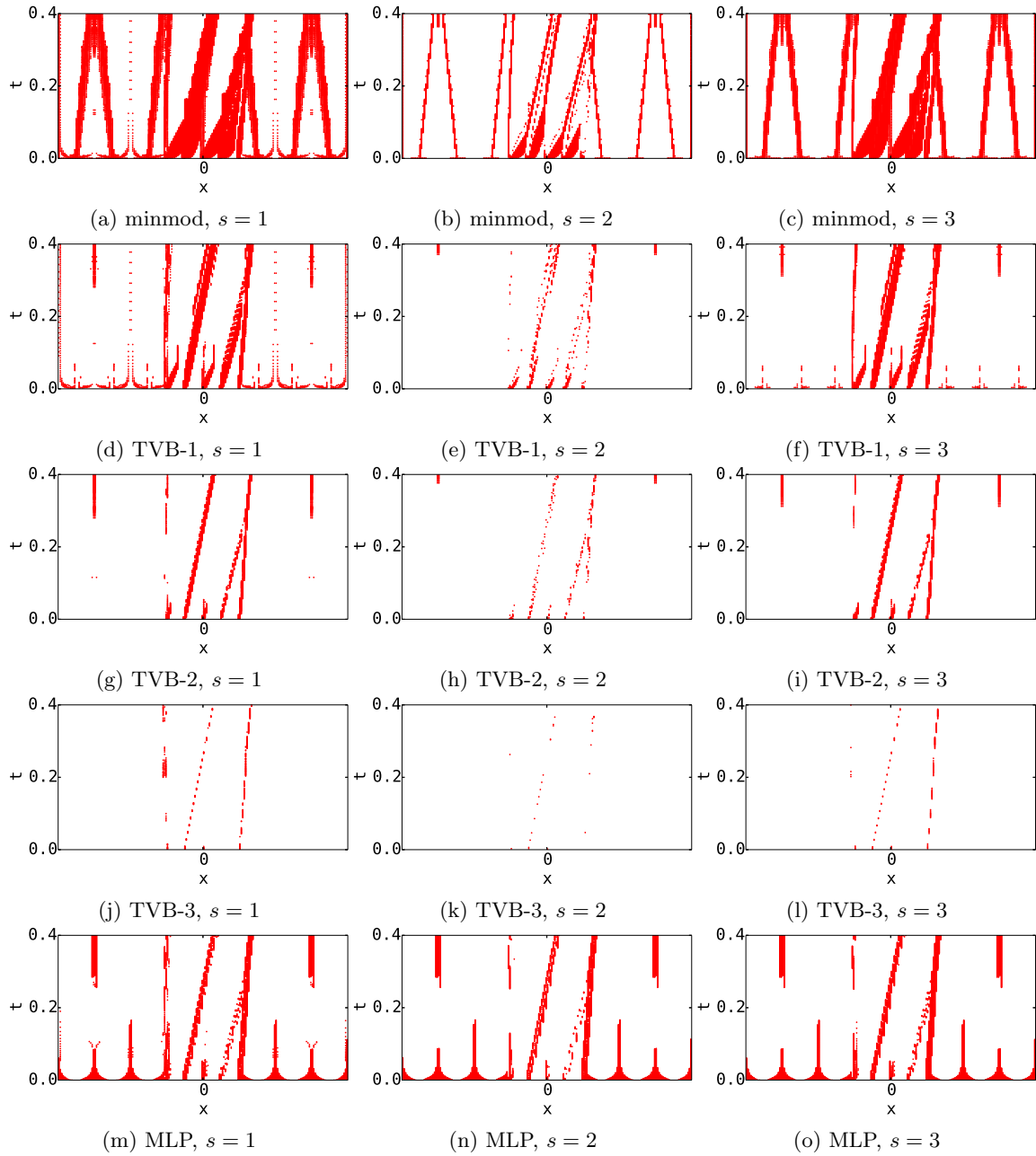
Figure 16: Test 2 for Burgers equation. The figures depict the cells in the mesh, flagged as troubled cells by various indicators as time evolves. The left, middle and right columns represents the cells flagged after the first RK stage (s=1), the second RK stage (s=2) and the third RK stage (s=3), respectively.

5.2.3. Buckley-Leverett

In order to demonstrate the performance of the MLP indicator for a non-convex flux, we consider the Buckley-Leverett equations 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 [36]. We consider the initial condition

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

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 $T = 0.4$ on a mesh with $N = 150$ cells and open boundary conditions. As shown in Figure 17, the solutions with the various indicators are almost 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 Figure 18).
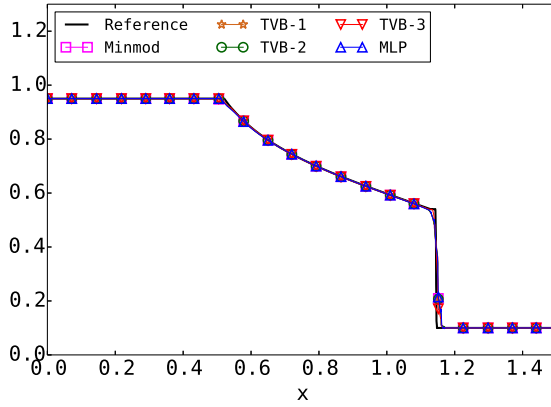

Figure 17: Riemann problem for the Buckley-Leverett equations at $T = 0.4$ with $N = 150$ cells and polynomial degree $r = 4$.

5.3. Shallow-water equations

The proposed MLP indicator can also be used for systems of conservation laws. To demonstrate this, 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. For system of conservation laws, it is important to limit the characteristic variables instead of the primary or conserved variables, to avoid spurious oscillations [37]. We solve the following Riemann problem simulating a dam-break

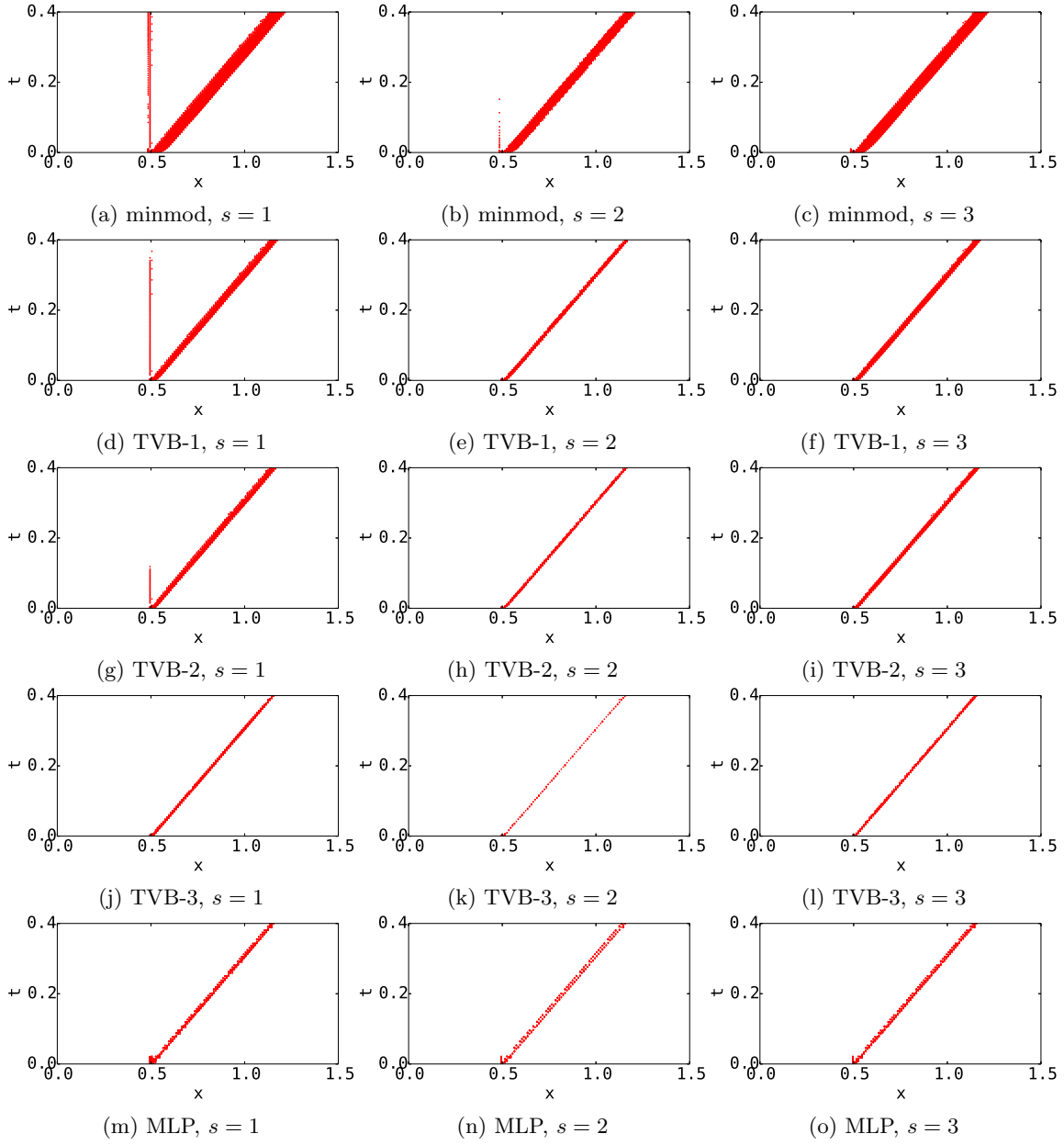$$D_0(x) = \begin{cases} 3 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases}, \qquad u_0(x) = 0,$$

24

Figure 18: Riemann problem for Buckey-Leverett equation. The figures depict the cells in the mesh, flagged as troubled cells by various indicators as time evolves. The left, middle and right columns represents the cells flagged after the first RK stage (s=1), the second RK stage (s=2) and the third RK stage (s=3), respectively.

on a mesh with $N = 100$ cells. The results in Figure 19 show that TVB-2 and TVB-3 give solutions contaminated with low amplitude oscillations, making them unsuitable for the current problem. This once again demonstrates the issue of choosing a TVB parameter M that works for a wide range of problems. The solution with the parameter-independent MLP indicator shows no spurious oscillations, and is comparable to the solutions obtained with the minmod limiter and TVB-1 (see Figure 20).
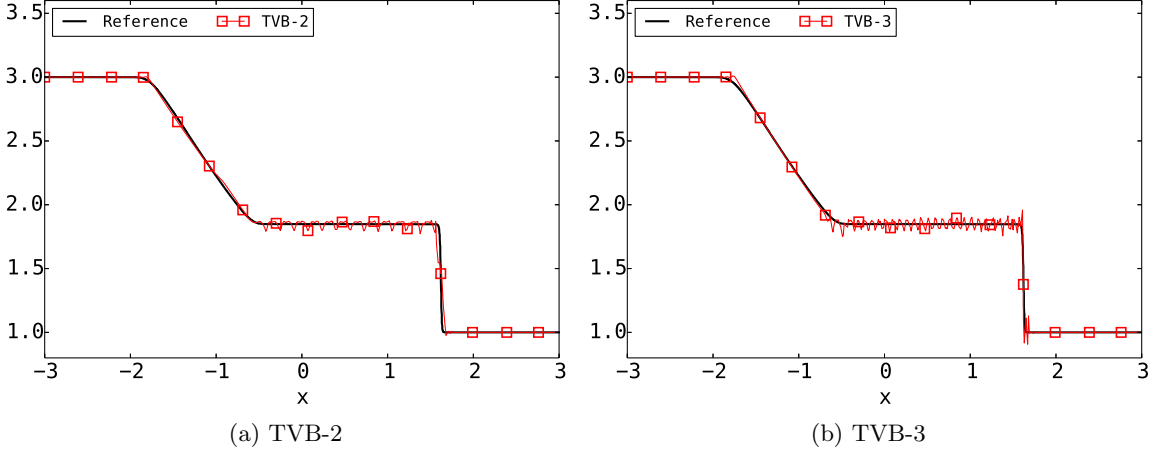

(a) TVB-2   (b) TVB-3

Figure 19: Dam-break problem (depth profile) for the shallow-water equations, evaluated at $T = 1$ with $N = 100$ cells and polynomial degree $r = 4$. TVB-2 and TVB-3 indicators leads to solutions contaminated by low amplitude spurious oscillations.

6. Conclusion

In this work, we propose a new approach to tackle the issue of troubled-cell detection, by designing a multilayer perceptron. 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. Traditional troubled-cell indicators, such as the minmod-type TVB limiter, usually depend on problem-dependent parameters. Thus, a fixed set of values for the parameters does not lead to uniformly satisfactory results as the initial condition, or the underlying conservation law is varied. An advantage of the proposed MLP indicator is that it does not depend on 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.

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 for multi-dimensional systems.
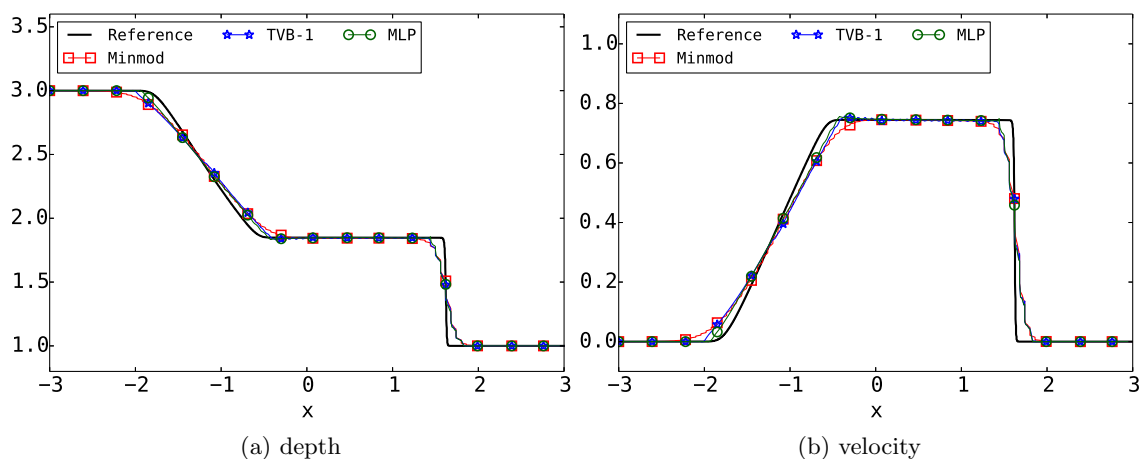
(a) depth   (b) velocity

Figure 20: Dam-break problem for the shallow-water equations, evaluated at $T = 1$ with $N = 100$ cells and polynomial degree $r = 4$. Numerical solutions are obtained using the minmod limiter, TVB-1 and MLP.

References

[1] C. M. Dafermos, Hyperbolic conservation laws in continuum physics, volume 325 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*, third ed., Springer-Verlag, Berlin, 2010.

[2] B. van Leer, Towards the ultimate conservative difference scheme. V - A second-order sequel to Godunov's method, Journal of Computational Physics 32 (1979) 101–136. doi: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, Journal of Computational Physics 84 (1989) 90 – 113. URL: http://www.sciencedirect.com/science/article/pii/0021999189901836. doi:https://doi.org/10.1016/0021-9991(89)90183-6.

[4] B. Cockburn, C.-W. Shu, The runge-kutta discontinuous galerkin method for conservation laws v: Multidimensional systems, Journal of Computational Physics 141 (1998) 199 – 224. URL: http://www.sciencedirect.com/science/article/pii/S0021999198958922. doi:https://doi.org/10.1006/jcph.1998.5892.

[5] A. Harten, B. Engquist, S. Osher, S. R. Chakravarthy, Uniformly high order accurate essentially non-oscillatory schemes, iii, Journal of Computational Physics 131 (1997) 3 – 47. URL: http://www.sciencedirect.com/science/article/pii/S0021999196956326. doi:https://doi.org/10.1006/jcph.1996.5632.

[6] X.-D. Liu, S. Osher, T. Chan, Weighted essentially non-oscillatory schemes, Journal of Computational Physics 115 (1994) 200 – 212. URL: http://www.sciencedirect.com/science/article/pii/S0021999184711879. doi:https://doi.org/10.1006/jcph.1994.1187.

[7] C.-W. Shu, Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws, Springer Berlin Heidelberg, Berlin, Heidelberg, 1998, pp. 325–432. URL: `https://doi.org/10.1007/BFb0096355`. doi:`10.1007/BFb0096355`.

[8] J. S. Hesthaven, T. Warburton, Nodal discontinuous Galerkin methods, volume 54 of *Texts in Applied Mathematics*, Springer, New York, 2008. URL: `http://dx.doi.org/10.1007/978-0-387-72067-8`. doi:`10.1007/978-0-387-72067-8`.

[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, in: Acta Numerica, volume 512, 2004, p. 451.

[11] J. Qiu., C.-W. Shu, Runge–kutta discontinuous galerkin method using weno limiters, SIAM Journal on Scientific Computing 26 (2005) 907–929. URL: `https://doi.org/10.1137/S1064827503425298`. doi:`10.1137/S1064827503425298`. arXiv:`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. URL: `http://dx.doi.org/10.1137/04061372X`. doi:`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. Comp. 52 (1989) 411–435. URL: `http://dx.doi.org/10.2307/2008474`. doi:`10.2307/2008474`.

[14] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998.

[15] P. D. Cristea, Application of Neural Networks In Image Processing and Visualization, Springer Netherlands, Dordrecht, 2009, pp. 59–71. URL: `https://doi.org/10.1007/978-90-481-2899-0_5`. doi:`10.1007/978-90-481-2899-0_5`.

[16] G. Dede, M. H. Sazli, Speech recognition with artificial neural networks, Digital Signal Processing 20 (2010) 763 – 768. URL: `http://www.sciencedirect.com/science/article/pii/S1051200409001821`. doi:`https://doi.org/10.1016/j.dsp.2009.10.004`.

[17] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, Trans. Neur. Netw. 9 (1998) 987–1000. URL: `http://dx.doi.org/10.1109/72.712178`. doi:`10.1109/72.712178`.

[18] S. Golak, A MLP Solver for First and Second Order Partial Differential Equations, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 789–797. URL: `https://doi.org/10.1007/978-3-540-74695-9_81`. doi:`10.1007/978-3-540-74695-9_81`.

[19] K. Rudd, S. Ferrari, A constrained integration (cint) approach to solving partial differential equations using artificial neural networks, Neurocomputing 155 (2015) 277 – 285. URL: `http://www.sciencedirect.com/science/article/pii/S092523121401652X`. doi:`https://doi.org/10.1016/j.neucom.2014.11.058`.

[20] G. Cybenko, Continuous Valued Neural Networks with Two Hidden Layers Are Sufficient, Technical Report, Department of Computer Science, Tufts University, Medford, MA, 1988.

[21] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems 2 (1989) 303–314. URL: https://doi.org/10.1007/BF02551274. doi:10.1007/BF02551274.

[22] 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. URL: https://doi.org/10.1162/NECO_a_00849. doi:10.1162/NECO_a_00849.

[23] S. Gottlieb, C.-W. Shu, E. Tadmor, Strong stability-preserving high-order time discretization methods, SIAM Rev. 43 (2001) 89–112 (electronic). URL: http://dx.doi.org/10.1137/S003614450036757X. doi:10.1137/S003614450036757X.

[24] B. Cockburn, C.-W. Shu, Runge–kutta discontinuous galerkin methods for convection-dominated problems, Journal of Scientific Computing 16 (2001) 173–261. URL: https://doi.org/10.1023/A:1012873910884. doi:10.1023/A:1012873910884.

[25] D. Kriesel, A Brief Introduction to Neural Networks, 2007. URL: http://www.dkriesel.com.

[26] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics 5 (1943) 115–133. URL: https://doi.org/10.1007/BF02478259. doi:10.1007/BF02478259.

[27] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85 – 117. URL: http://www.sciencedirect.com/science/article/pii/S0893608014002135. doi:https://doi.org/10.1016/j.neunet.2014.09.003.

[28] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: J. Frnkranz, T. Joachims (Eds.), Proceedings of the 27th International Conference on Machine Learning (ICML-10), Omnipress, 2010, pp. 807–814. URL: http://www.icml2010.org/papers/432.pdf.

[29] 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 25, Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[30] 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.

[31] S. Kullback, R. A. Leibler, On information and sufficiency, Ann. Math. Statist. 22 (1951) 79–86. URL: https://doi.org/10.1214/aoms/1177729694. doi:10.1214/aoms/1177729694.

[32] Y. Bengio, Practical Recommendations for Gradient-Based Training of Deep Architectures, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 437–478. URL: https://doi.org/10.1007/978-3-642-35289-8_26. doi:10.1007/978-3-642-35289-8_26.

[33] S. J. Nowlan, G. E. Hinton, Simplifying neural networks by soft weight-sharing, Neural Comput. 4 (1992) 473–493. URL: `http://dx.doi.org/10.1162/neco.1992.4.4.473`. doi:`10.1162/neco.1992.4.4.473`.

[34] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, CoRR abs/1412.6980 (2014). URL: `http://arxiv.org/abs/1412.6980`. `arXiv:1412.6980`.

[35] TensorFlow, Large-scale machine learning on heterogeneous systems, 2015. URL: `https://www.tensorflow.org/`.

[36] R. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge Texts in Applied Mathematics, Cambridge University Press, 2002.

[37] J. Qiu, C.-W. Shu, On the construction, comparison, and local characteristic decomposition for high-order central weno schemes, Journal of Computational Physics 183 (2002) 187 – 209. URL: `http://www.sciencedirect.com/science/article/pii/S0021999102971913`. doi:`https://doi.org/10.1006/jcph.2002.7191`.