Journal of Scientific Computing (2023) 97:36
https://doi.org/10.1007/s10915-023-02335-x

Using Deep Neural Networks for Detecting Spurious Oscillations in Discontinuous Galerkin Solutions of Convection-Dominated Convection–Diffusion Equations

Derk Frerichs-Mihov[1] · Linus Henning[2] · Volker John[1,2]

Received: 4 March 2023 / Revised: 17 June 2023 / Accepted: 24 August 2023 /
Published online: 25 September 2023

Abstract
Standard discontinuous Galerkin finite element solutions to convection-dominated convection–diffusion equations usually possess sharp layers but also exhibit large spurious oscillations. Slope limiters are known as a post-processing technique to reduce these unphysical values. This paper studies the application of deep neural networks for detecting mesh cells on which slope limiters should be applied. The networks are trained with data obtained from simulations of a standard benchmark problem with linear finite elements. It is investigated how they perform when applied to discrete solutions obtained with higher order finite elements and to solutions for a different benchmark problem.

Keywords Convection–diffusion equations · Discontinuous Galerkin methods · Spurious oscillations · Deep neural networks · Slope limiter

Mathematics Subject Classification 65N30 · 68T07

1 Introduction

Convection–diffusion equations are a basic model to describe the distribution of a scalar quantity in fluids. Besides modeling the heat distribution in a room (energy balance), they can describe the concentration of drugs in blood and the propagation of chemical substances

✉ Derk Frerichs-Mihov
frerichs-mihov@wias-berlin.de

Linus Henning
linus.henning@fu-berlin.de

Volker John
john@wias-berlin.de

[1] Weierstrass Institute for Applied Analysis and Stochastics (WIAS), Mohrenstr. 39, 10117 Berlin, Germany

[2] Department of Mathematics and Computer Science, Freie Universität Berlin, Arnimallee 6, 14195 Berlin, Germany

in water (mass balance) to name just a few. Mathematically speaking, they are given in a bounded domain $\Omega \subset \mathbb{R}^d, d \in \{2, 3\}$, with polyhedral Lipschitz boundary $\Gamma = \Gamma_D \cup \Gamma_N$ with $\Gamma_D \cap \Gamma_N = \emptyset$. The steady-state convection–diffusion–reaction problem with homogeneous Neumann boundary conditions on Γ_N then reads as follows: Find a sufficiently smooth function u such that

$$\begin{aligned} -\varepsilon \Delta u + \boldsymbol{b} \cdot \nabla u + cu &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \Gamma_D, \\ \varepsilon \nabla u \cdot \boldsymbol{n} &= 0 \quad \text{on } \Gamma_N, \end{aligned} \tag{1}$$

where $\varepsilon > 0$ is the diffusion coefficient, the convection field is denoted by $\boldsymbol{b} \in [W^{1,\infty}(\Omega)]^d$, $c \in L^\infty(\Omega)$ describes the reaction coefficient, and $f \in L^2(\Omega)$ models sources. On the Dirichlet boundary Γ_D Dirichlet conditions g are prescribed and the outer unit normal vector on the boundary of Ω is denoted by $\boldsymbol{n}$. At the inflow boundary $\Gamma_- = \{\boldsymbol{x} \in \Gamma : \boldsymbol{b}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x}) < 0\}$, Dirichlet boundary conditions have to be prescribed, i.e., $\Gamma_- \subset \Gamma_D$.

In many applications the convective transport dominates the diffusive one. Then, the characteristic feature of solutions of Eq. (1) are layers, which are thin regions with a very large gradient. The thickness of layers is usually so small that they cannot be resolved on feasible grids. This situation, which is mathematically expressed by $\varepsilon \ll h\|\boldsymbol{b}\|_{L^\infty(\Omega)}$, where h is the (local) mesh size, is called the convection-dominated regime. It is a typical property of multiscale problems that very important features of the solution cannot be resolved. Consequently, convection–diffusion–reaction problems are usually multiscale problems, with the layers being the subgrid scales. It is well known that in this case the discrete solution to Eq. (1) obtained by classical numerical schemes, like the central finite difference method and Galerkin finite element method, exhibits huge so-called spurious oscillations, i.e., unphysical values such as negative concentrations or an unreasonable amount of energy, e.g., see [1–4].

There have been many contributions concerning discontinuous Galerkin (DG) methods for discretizing second order elliptic boundary value problems in the last decades, e.g., see the monographs [5–7], even though they were already invented in 1973 in [8]. One advantage is that, compared with conforming finite elements, hp-refinement on both simplicial and also polygonal and polyhedral meshes can be performed very easily, e.g., see [9, 10]. With respect to convection–diffusion equations, DG methods with a standard upwind flux are stable discretizations in the convection-dominated regime. It was shown in [6, 11–13] that they even control the streamline derivative without needing an additional term, as it is the case for conforming finite elements. Furthermore, DG methods are known to produce very sharp layers in the convection-dominated regime. But on the other hand these methods also have the flaw of producing large over- and undershoots [4, 14, 15].

A computational cheap way to significantly reduce spurious oscillations in a post-processing step are so-called slope limiters. In a first step, they identify so-called troubled-cells where over- and undershoots occur and, in a second step, replace the solution locally by a polynomial of lower degree. The solution is usually replaced by a constant approximation [16, 17] or a (at most) linear one [5, 18]. This approach is typical for appropriate numerical methods for multiscale problems in the sense that different schemes are applied for the different scales. Using low order finite elements in a vicinity of layers is fine since error bounds for higher order elements contain the norm of the solution in a higher order Sobolev space and these norms scale (locally at layers) with inverse powers of the diffusion coefficient. The power increases with the order of the Sobolev space and finally there is a huge constant in the error bound such that it cannot be expected to obtain a better accuracy in a neighborhood of layers when using higher order elements there. In [14, 15] several of these methods have been numerically investigated for convection-dominated convection–diffusion

equations. These methods share the advantage that they are computationally cheap, keep the higher order approximation away from layers, and most important that most of the methods also reduce the oscillations significantly, but not completely [14, 15].

Within the last decades the interest in deep neural networks as a form of deep learning has risen sharply. Thanks to their ability to be universal function approximators [19–21, Chapter 6.4.1] and classifiers [22], they have also been used to detect troubled-cells. In 2018, Ray and Hesthaven have trained a multilayer perceptron (MLP) to identify troubled-cells for one-dimensional scalar and systems of conservation laws [23]. They have observed that their MLP detector can mimic a classical limiter but without the need of fine-tuning a parameter. Their results have been extended by the authors to two-dimensional problems in [24]. Liu et al. have trained a convolutional neural network (CNN) based shock detector for Euler's gas equations and saw that their detector was significantly faster than classical ones [25]. In 2018 and 2020, Han Veiga and Abgrall have trained a MLP detector based on data from a Runge–Kutta DG scheme and tested how well it can then be transferred to a residual distribution (RD) scheme without retraining it [26, 27]. Again, they have used scalar transport equations and Euler's gas equations. Morgan et al. have trained and tested a MLP detector with a Lagrangian RD method to detect troubled-cells in the two-dimensional Taylor–Green vortex and Triple-point vortex [28]. Beck et al. have trained a CNN based limiter in 2020 for a DG spectral element method to approximate the solution of Euler's gas equations [29]. Their limiter is able to both detect cells and also locate the position of the shock inside the cell. As it can be seen, all these results are for different numerical schemes for the discretization of scalar or hyperbolic systems of equations, mainly Euler's gas equations. Neural networks have been applied also with respect to other aspects of the numerical solution of partial differential equations, e.g., see [30–34].

The goal of this paper consists in performing a first step in systematically exploring the behavior of MLPs for classifying mesh cells for the numerical solution of convection–diffusion–reaction problems. In contrast to the above mentioned papers, this manuscript studies an elliptic boundary value problem. On the one hand, slope limiting processes in the hyperbolic and the elliptic case are quite similar: Based on local features of the numerical solution, slope limiting algorithms try to detect the cells where spurious oscillations are present. Indeed, many slope limiting techniques that were proposed for elliptic problems are either initially constructed for hyperbolic ones or are modified versions of methods for such problems. Therefore, in general it could be expected that similar architectures of the neural networks can be employed in the elliptic case as in the hyperbolic counterparts. This is also the motivation why in the present work multilayer perceptron models are used as in [23, 24, 26–28]. Note that with the same argument it should also be possible to use CNN-based architectures as done in [25, 29], but for the sake of brevity, this approach is postponed to future work. Moreover, due to the close relationship between slope limiters for hyperbolic and elliptic problems, it is not surprising that many features that are used as the input to the MLPs in this work are also considered in [23, 24, 26–28], e.g., the integral mean of the cell and its neighbors, and values at interface midpoints.

On the other hand, numerical methods for hyperbolic (or convection-dominated parabolic) problems apply often small time steps. Then, the starting solution, which is (nearly) free of spurious oscillations, can be utilized to reduce such oscillations in the solution of the next time step, since it can be expected that both solutions are in some sense not that much different. This strategy is used, e.g., in the construction of limiters for flux corrected transport schemes, e.g., see [35] for a description. However, there is no such starting solution for elliptic problems. Slope limiting for elliptic problems has to be applied to a solution that was computed with some (stabilized) discretization and which possesses usually notable spurious oscillations.

As already mentioned, the present work focuses on the elliptic boundary value problem given in Eq. (1). It is well known that for computing an accurate solution of Eq. (1) without or with only small (acceptable) spurious oscillations, one has to treat the subgrid scale layer regions differently than the large scale regions, see the recent survey paper [35]. This first step aims to figure out whether or not there are MLPs that work well for a situation where the result is already principally known, just to have a kind of benchmark situation to compare with. This means, this first step should be considered as a proof of concept. The considered situation is the above described reduction of spurious oscillations in DG methods using limiters. To this end, a limiter is going to be trained with data that is obtained by applying classical limiters to the lowest order discrete solution of a standard benchmark problem defined in [36]. Several architectures are tested and it will be investigated how well the resulting limiter can be applied to higher order solutions and to another benchmark problem, the so-called Hemker example from [37]. Since standard limiters are already quite efficient, there is no need to replace them by the MLP-based limiter for increasing the efficiency. However, using the MLP-based limiter might be appealing in practice, since it combines good properties of various traditional limiters and it is not necessary that the user chooses explicitly a concrete limiter, together with the necessary parameters. But one should be aware that the MLP-based limiter depends implicitly on the parameters of traditional limiters that were used in the training process. Our principal intention consists in using the results and experience obtained in this proof of concept study in situations that require to perform different algorithms for different mesh cells and where efficient methods are not known. Such a situation is the choice of local parameters in stabilized discretizations of convection–diffusion–reaction problems. Currently available approaches, based on solving non-convex optimization problems, e.g., [38, 39], are rather time-consuming.

The remainder of the paper is structured as follows: in Sect. 2, both the standard DG method for discretizing equation (1) and relevant classical slope limiters are introduced. Section 3 follows with explaining the multilayer perceptron model and how the data is created with which the MLP limiter is trained. Several architectures are trained in Sect. 4 and are tested numerically. The paper concludes with a short summary and outlook. All data and code created and used for this work can be found at www.doi.org/10.20347/40vd-f944 [40].

In what follows the usual notation is used for Lebesgue and Sobolev spaces and their respective norms. The inner product in $L^2(\Omega)$ is denoted by $(\cdot,\cdot)$, a norm of a space X is denoted by $\|\cdot\|_X$ and a seminorm by $|\cdot|_X$.

2 Discontinuous Galerkin Methods and Slope Limiter for Convection–Diffusion Equations

2.1 Discontinuous Galerkin Methods

Equation (1) can be transformed to its weak formulation in a standard way which then reads as follows: Find $u \in H^1(\Omega)$ such that $u = g$ on Γ_D and

$$(\varepsilon \nabla u, \nabla v) + (\boldsymbol{b} \cdot \nabla u + cu, v) = (f, v) \qquad \forall v \in H^1_{D,0}(\Omega), \tag{2}$$

where $H^1_{D,0}(\Omega) := \{v \in H^1(\Omega) \ : \ v = 0 \text{ on } \Gamma_D\}$. Under the assumptions that

$$c - \frac{1}{2}\nabla \cdot \boldsymbol{b} \geq 0, \qquad \Gamma_D \neq \emptyset, \qquad \boldsymbol{b} \cdot \boldsymbol{n} \geq 0 \text{ on } \Gamma_N,$$

by applying the Lax–Milgram Lemma it can be proven that problem (2) possesses a unique weak solution, e.g., see [1, Section III.1.1].

To introduce the DG discretization of (2), some notation needs to be fixed. Let $\{\mathcal{T}_h\}, 0 < h$, be a quasi-uniform family of decompositions of $\overline{\Omega}$ into simplicial or quadrilateral/hexahedral meshes such that for any h it holds $\overline{\Omega} = \cup_{K\in\mathcal{T}_h} K$ and the cells have pairwise disjoint interiors. As usual the triangulations should be admissible, see [41, p. 38, p. 51], i.e., among other properties, each facet of a mesh cell that lies on Γ is either contained in Γ_D or Γ_N. The set of all facets is denoted by $\mathcal{E}_h := \cup_{K\in\mathcal{T}_h} \mathcal{E}_h(K)$, where $\mathcal{E}_h(K)$ is the set of all facets $E \subset \partial K$ of a cell K. Furthermore, this set can be decomposed into the set of all interior facets $\mathcal{E}_h^\mathrm{I}$ and boundary facets $\partial\mathcal{E}_h := \mathcal{E}_h \cap \partial\Omega$. The inflow boundary facets are called $\mathcal{E}_h^- := \Gamma_- \cap \mathcal{E}_h$, the set of Dirichlet boundary facets is denoted by $\mathcal{E}_h^\mathrm{D} := \partial\mathcal{E}_h \cap \Gamma_\mathrm{D}$ and the notation $\mathcal{E}_h^\mathrm{ID} := \mathcal{E}_h^\mathrm{I} \cup \mathcal{E}_h^\mathrm{D}$ is set. In addition to that, $|K|$ denotes the d-volume and h_K the diameter of a cell $K \in \mathcal{T}_h$ and $h := \max_{K\in\mathcal{T}_h} h_K$ is defined. Due to the regularity of the family of triangulations there exists a constant $C > 0$ such that for all $\mathcal{T}_h$ and $K \in \mathcal{T}_h$ it holds $h_E \le h_K \le C h_E$, where h_E is the diameter of a facet $E \in \mathcal{E}_h(K)$.

If there exists a facet $E \in \mathcal{E}_h(K_i) \cap \mathcal{E}_h(K_j)$ that is shared by the cells $K_i, K_j \in \mathcal{T}_h$, then the cells are called neighbors. Under the assumption that there is a fixed numbering of the mesh cells $K_0, K_1, \ldots \in \mathcal{T}_h$, the unit normal vector $\boldsymbol{n}_E$ on a facet $E \in \mathcal{E}_h$ is defined by

$$\boldsymbol{n}_E := \begin{cases} \boldsymbol{n}_K, & \text{if } E \in \partial\mathcal{E}_h \cap \mathcal{E}_h(K) \text{ for a } K \in \mathcal{T}_h, \\ \boldsymbol{n}_{K_i}, & \text{if } K_i \text{ and } K_j \text{ are neighbors along facet } E \text{ and } i < j, \end{cases}$$

where $\boldsymbol{n}_K$ denotes the outward unit normal vector of the cell $K \in \mathcal{T}_h$.

The below defined DG space is a subspace of the broken Sobolev space

$$H^k(\mathcal{T}_h) = \{v \in L^2(\Omega) \,:\, v|_K \in H^k(K) \text{ for any } K \in \mathcal{T}_h\} \supset H^k(\Omega), \qquad k \in \mathbb{N},$$

that is equipped with its piecewise-defined norm and semi-norm

$$\|v\|^2_{H^k(\mathcal{T}_h)} := \sum_{K\in\mathcal{T}_h} \|v\|^2_{H^k(K)}, \quad |v|^2_{H^k(\mathcal{T}_h)} := \sum_{K\in\mathcal{T}_h} |v|^2_{H^k(K)}.$$

Given a fixed $p \in \mathbb{N}$, the finite element space is defined by

$$V_{h,p} := \left\{v_h \in L^2(\Omega) \,:\, v_h|_K \in \mathcal{R}_p(K) \text{ for any } K \in \mathcal{T}_h\right\} \subset H^k(\mathcal{T}_h),$$

where $\mathcal{R}_p(K) := P_p(K)$ is the space of polynomials of at most degree p on simplicial mesh cells and $\mathcal{R}_p(K) := Q_p(K)$ is the tensor product space of polynomials of at most degree p on quadrilateral/hexahedral cells.

Both $H^k(\mathcal{T}_h)$ and $V_{h,p}$ contain functions that are discontinuous along interior facets $E \in \mathcal{E}_h$. Hence, a given function $v \in V_{h,p}$ itself is not well-defined on E but its jump $[\![v]\!]$ and average $\langle v\rangle$ can be defined for any $\boldsymbol{x} \in E$ by

$$[\![v]\!](\boldsymbol{x}) := \begin{cases} v|_{K_i}(\boldsymbol{x}) - v|_{K_j}(\boldsymbol{x}), & \text{if } K_i \text{ and } K_j \text{ are neighbors along facet } E \text{ and } \\ & i < j, \\ v|_K(\boldsymbol{x}), & \text{if } E \in \partial\mathcal{E}_h \cap \mathcal{E}_h(K) \text{ for a } K \in \mathcal{T}_h, \end{cases}$$

and

$$\langle v\rangle(\boldsymbol{x}) := \begin{cases} \frac{1}{2}(v|_{K_i}(\boldsymbol{x}) + v|_{K_j}(\boldsymbol{x})), & \text{if } K_i \text{ and } K_j \text{ are neighbors along facet } E \\ & \text{and } i \neq j, \\ v|_K(\boldsymbol{x}), & \text{if } E \in \partial\mathcal{E}_h \cap \mathcal{E}_h(K) \text{ for a } K \in \mathcal{T}_h. \end{cases}$$

Finally, the DG discretization of (1) reads as follows: Find $u_h \in V_{h,p}$ such that

$$a_{\mathrm{DG}}(u_h, v_h) = f_{\mathrm{DG}}(v_h) \quad \forall\, v_h \in V_{h,p}, \tag{3}$$

where the bilinear form $a_{\mathrm{DG}} : H^1(\mathcal{T}_h) \times H^1(\mathcal{T}_h) \to \mathbb{R}$ is defined as $a_{\mathrm{DG}}(v, w) := a_\varepsilon(v, w) + a_{bc}(v, w)$, where $v, w \in H^1(\mathcal{T}_h)$, with

$$\begin{aligned} a_\varepsilon(v, w) = &\sum_{K \in \mathcal{T}_h} \int_K \varepsilon \nabla v \cdot \nabla w \, \mathrm{d}\boldsymbol{x} \\ &- \sum_{E \in \mathcal{E}_h^{\mathrm{ID}}} \varepsilon \int_E \Big(\langle \nabla v \cdot \boldsymbol{n}_E \rangle [\![w]\!] + \kappa \langle \nabla w \cdot \boldsymbol{n}_E \rangle [\![v]\!] \Big) \, \mathrm{d}s \\ &+ \sum_{E \in \mathcal{E}_h^{\mathrm{I}}} \frac{\sigma}{h_E} \int_E [\![v]\!][\![w]\!] \, \mathrm{d}s + \sum_{E \in \mathcal{E}_h^{\mathrm{D}}} \frac{2\sigma}{h_E} \int_E vw \, \mathrm{d}s \end{aligned} \tag{4}$$

and

$$\begin{aligned} a_{bc}(v, w) = &\sum_{K \in \mathcal{T}_h} \int_K \big(\boldsymbol{b} \cdot \nabla v w + c v w \big) \, \mathrm{d}\boldsymbol{x} - \sum_{E \in \mathcal{E}_h^{\mathrm{I}}} \int_E \boldsymbol{b} \cdot \boldsymbol{n}_E [\![v]\!] \langle w \rangle \, \mathrm{d}s \\ &+ \sum_{E \in \mathcal{E}_h^{\mathrm{I}}} \int_E \frac{\eta}{2} |\boldsymbol{b} \cdot \boldsymbol{n}_E| [\![v]\!][\![w]\!] \, \mathrm{d}s - \sum_{E \in \mathcal{E}_h^-} \int_E \boldsymbol{b} \cdot \boldsymbol{n}_E v w \, \mathrm{d}s. \end{aligned} \tag{5}$$

The discrete right-hand side $f_{\mathrm{DG}} : H^1(\mathcal{T}_h) \to \mathbb{R}$ of (3) is defined by

$$\begin{aligned} f_{\mathrm{DG}}(w) = &\sum_{K \in \mathcal{T}_h} \int_K f w \, \mathrm{d}\boldsymbol{x} - \sum_{E \in \mathcal{E}_h^-} \int_E \boldsymbol{b} \cdot \boldsymbol{n}_E g w \, \mathrm{d}s \\ &- \sum_{E \in \mathcal{E}_h^{\mathrm{D}}} \varepsilon \kappa \int_E \nabla w \cdot \boldsymbol{n}_E g \, \mathrm{d}s + \sum_{E \in \mathcal{E}_h^{\mathrm{D}}} \frac{2\sigma}{h_E} \int_E g w \, \mathrm{d}s. \end{aligned} \tag{6}$$

The discrete scheme (3) contains three user-chosen parameters. The parameter κ controls the symmetry of (4) where $\kappa = 1$ corresponds to the symmetric interior penalty Galerkin (SIPG), $\kappa = 0$ to the incomplete interior penalty Galerkin (IIPG), and $\kappa = -1$ to the non-symmetric (NIPG) discretization of the Laplacian. The stability parameter σ, also called penalty parameter, in (4) and (6) that is incorporated as in [13, Section 2.2] influences the coercivity of a_ε: The bilinear form for the SIPG and IIPG method is coercive if σ is sufficiently large, where σ is proportional to ε, and for the NIPG method it is coercive for any $\sigma > 0$, e.g., see [5, Chapter 2.7.1]. Last but not least, the stabilization parameter $\eta \geq 0$ appearing in (5) has to be chosen by the user. The choice $\eta = 0$ corresponds to a centered flux and $\eta = 1$ to an upwind flux discretization across the facet E, e.g., see [6, p. 55, p. 65]. It can be proven that DG methods converge asymptotically with an optimal rate in the DG norm, with an optimal rate in the L^2-norm only for the SIPG variant and suboptimally for the IIPG and NIPG method, e.g., see [5–7, 13] and the references therein.

2.2 Slope Limiters

Slope limiters are a cheap post-processing technique to reduce spurious oscillations in the discrete solution. After the solution u_h of (3) is computed, the solution is adapted as follows:

1. *Identify* and *mark* cells in which the function might show spurious oscillations by
   (a) computing (cell wise) a set of *features* of the solution and
   (b) based on these features deciding whether to mark a cell.
2. *Approximate* the solution locally on the marked cells by a polynomial of lower degree.

These steps can be translated into mathematics by introducing some mappings. Let $\mathcal{F}_l : V_{h,p} \times \mathcal{T}_h \to \mathbb{R}^{n_l}$ be a function that maps locally a discrete function to $n_l \in \mathbb{N}$ features, and $\mathcal{M}_l : \mathbb{R}^{n_l} \to \{0, 1\}$ be a decision maker function. The post-processing techniques can then be seen as mappings $l : V_{h,p} \to V_{h,p}$ defined cell wise on a cell $K \in \mathcal{T}_h$ for $u_h \in V_{h,p}$ by

$$l(u_h)|_K := \begin{cases} u_h|_K, & \text{if } \mathcal{M}_l(\mathcal{F}_l(u_h, K)) = 0, \\ \Pi_{l,K}(u_h), & \text{else,} \end{cases}$$

where $\Pi_{l,K} : V_{h,p}|_K \to \mathcal{R}_p(K)$ reconstructs the solution in marked cells.

Different methods differ only in their respective functions $\mathcal{F}_l$, $\mathcal{M}_l$, $\Pi_{l,K}$. Several of these methods are described in detail and were tested numerically in [14, 15] and they will be briefly recalled here. Since for what comes later only $\mathcal{M}_l$ and $\mathcal{F}_l$ are important, $\Pi_{l,K}$ is only mentioned in passing. For the sake of presentation, the methods are described in two dimensions on triangles, but they can be easily extended to three dimensions or to quadrilateral/hexahedral meshes.

It is worth to emphasize that for all the methods presented below, the mappings $\mathcal{F}_l$ and $\mathcal{M}_l$ act locally, i.e., they are defined cell wise. Especially $\mathcal{F}_l$ can be computed using only information of the discrete solution on a cell itself and possibly its direct neighbors, and globally defined quantities like a tolerance or reference values. The mapping $\mathcal{M}_l$ then takes *cell wise features* and returns locally the decision whether to mark a cell or not.

Since the numerical studies consider only two-dimensional problems, the individual slope limiters will be presented, for simplicity, only to this situation. As already noted in [14], their extension to three dimensions is usually straightforward.

LinTriaReco

This method was proposed in [18] and described again in [5, pp. 103–104] and [14].

Let $K \in \mathcal{T}_h$ be a simplicial cell with facets $E_i \in \mathcal{E}_h(K)$, $i = 0, 1, 2$, and neighbors $K_i \in \mathcal{T}_h$ along these edges. Using the notation m_i, $i = 0, 1, 2$, for the edge mid points and $\overline{u}_{h,K} := \int_K u_h \, \mathrm{d}\boldsymbol{x}/|K|$ for the integral mean of u_h in K, the feature mapping of *LinTriaReco* is defined by

$$\mathcal{F}_{\mathrm{LTR}}(u_h, K) := \{\overline{u}_{h,K_0},\ u_h|_K(m_0),\ \overline{u}_{h,K_1},\ u_h|_K(m_1),\ \overline{u}_{h,K_2},\ u_h|_K(m_2),\ \overline{u}_{h,K},\ tol\}, \tag{7}$$

where $tol \in \mathbb{R}$, $tol \ll 1$ is a fixed positive tolerance. Hence, the number of features n_{LTR} of *LinTriaReco* is 8.

Let $\lfloor a, b \rfloor := \min\{a, b\}$ and $\lceil a, b \rceil := \max\{a, b\}$ for $a, b \in \mathbb{R}$. The decision maker $\mathcal{M}_{\mathrm{LTR}}$ is given by

$$\mathcal{M}_{\mathrm{LTR}}(\mathcal{F}_{\mathrm{LTR}}(u_h, K)) := \begin{cases} 1, & \text{if } \mathcal{E}_h(K) \cap \partial\mathcal{E}_h = \emptyset \wedge \\ & \big(u_h|_K(m_0) \notin \big[\lfloor \overline{u}_{h,K_0}, \overline{u}_{h,K} \rfloor - tol, \lceil \overline{u}_{h,K_0}, \overline{u}_{h,K} \rceil + tol\big] \vee \\ & u_h|_K(m_1) \notin \big[\lfloor \overline{u}_{h,K_1}, \overline{u}_{h,K} \rfloor - tol, \lceil \overline{u}_{h,K_1}, \overline{u}_{h,K} \rceil + tol\big] \vee \\ & u_h|_K(m_2) \notin \big[\lfloor \overline{u}_{h,K_2}, \overline{u}_{h,K} \rfloor - tol, \lceil \overline{u}_{h,K_2}, \overline{u}_{h,K} \rceil + tol\big]\big) \\ 0, & \text{else.} \end{cases} \tag{8}$$

The tolerance tol is introduced to prevent marking of cells due to numerical round-off errors. Hence roughly speaking, *LinTriaReco* marks an interior cell K if for at least one edge the value of the solution at the edge midpoint is not between the cell averages of the function in the cell and the corresponding neighbor.

For the reconstruction $\Pi_{\mathrm{LTR,K}}$, three affine functions are constructed based on the cell averages of the discrete solution in the cell and its neighbors of which one is chosen, e.g., see [5, 14, p. 104].

ConstTriaReco

This method was proposed in [14] and is a modification of *LinTriaReco*. Instead of evaluating the function at the edge midpoint the integral mean $\overline{u}^{E}_{h,K} := \int_E u_h|_K \,\mathrm{d}s / h_E$ is used. Furthermore, for boundary edges $E \in \mathcal{E}_h \cap \partial\mathcal{E}_h$, i.e., edges along which the cell has no neighbor, a virtual neighbor is constructed by mirroring the opposite vertex along the edge E. Then, on this virtual neighbor the discrete function is defined to be the continuation of $u_h|_K$ which exists and is well-defined since $u_h|_K$ is a polynomial of degree at most p. In this way every triangle has three neighbors and a cell average in each neighbor can be computed.

The feature mapping is then given by

$$\mathcal{F}_{\mathrm{CTR}}(u_h, K) := \{\overline{u}_{h,K_0},\ \overline{u}^{E_0}_{h,K},\ \overline{u}_{h,K_1},\ \overline{u}^{E_1}_{h,K},\ \overline{u}_{h,K_2},\ \overline{u}^{E_2}_{h,K},\ \overline{u}_{h,K},\ tol\}, \tag{9}$$

and hence $n_{\mathrm{CTR}} = 8$.

ConstTriaReco's decision maker is then analogously defined by

$$\mathcal{M}_{\mathrm{CTR}}(\mathcal{F}_{\mathrm{CTR}}(u_h, K)) := \begin{cases} 1, & \text{if } \overline{u}^{E_0}_{h,K} \notin \left[\lfloor \overline{u}_{h,K_0}, \overline{u}_{h,K} \rfloor - tol,\ \lceil \overline{u}_{h,K_0}, \overline{u}_{h,K} \rceil + tol\right] \vee \\ & \overline{u}^{E_1}_{h,K} \notin \left[\lfloor \overline{u}_{h,K_1}, \overline{u}_{h,K} \rfloor - tol,\ \lceil \overline{u}_{h,K_1}, \overline{u}_{h,K} \rceil + tol\right] \vee \\ & \overline{u}^{E_2}_{h,K} \notin \left[\lfloor \overline{u}_{h,K_2}, \overline{u}_{h,K} \rfloor - tol,\ \lceil \overline{u}_{h,K_2}, \overline{u}_{h,K} \rceil + tol\right] \\ 0, & \text{else.} \end{cases} \tag{10}$$

To reconstruct the solution, $\Pi_{\mathrm{CTR},K}(u_h) := \overline{u}_{h,K}$ is used, which led often to good results in the numerical studies of [14].

ConstJumpMod

A different approach is taken by *ConstJumpMod* that was proposed in [14] and improved in [15]. Based on the marking criterion of [16, 17] *ConstJumpMod* tries to approximate the local order of convergence along each edge and marks a cell if this order is smaller than some reference value.

Let $0 < C_0 \in \mathbb{R}$ be a positive constant, L be a characteristic length scale of the problem and u_0 a characteristic scale of the solution. For each edge $E \in \mathcal{E}_h$ the quantity

$$\alpha_E := \begin{cases} \ln\left(\frac{1}{C_0 L u_0^2} \int_E [\![u_h]\!]^2 : \mathrm{d}s\right) \Big/ \ln\left(\frac{h_E}{L}\right), & \text{if } E \in \mathcal{E}_h^{\mathrm{I}}, \\ \alpha_{\mathrm{ref}}, & \text{else,} \end{cases}$$

can be computed, where $\alpha_{\mathrm{ref}} \in \mathbb{R}$ is a fixed positive reference value. These values are used to define the feature mapping that is given by

$$\mathcal{F}_{\mathrm{CJM}}(u_h, K) := \{\alpha_{E_0}, \alpha_{E_1}, \alpha_{E_2}, \alpha_{\mathrm{ref}}\} \tag{11}$$

and it follows that $n_{\mathrm{CJM}} = 4$.

To mark a cell K, the decision maker

$$\mathcal{M}_{\mathrm{CJM}}(\mathcal{F}_{\mathrm{CJM}}(u_h, K)) := \begin{cases} 1, & \text{if } \min_{i=0,1,2} \alpha_{E_i} < \alpha_{\mathrm{ref}}, \\ 0, & \text{else,} \end{cases} \tag{12}$$

is used. Note, to prevent having infinite values in the feature set before computing $\mathcal{M}_{\mathrm{CJM}}$, these values can be replaced by α_{ref} without changing the result of $\mathcal{M}_{\mathrm{CJM}}$, which might be beneficial for the implementation.

The solution in the marked cells is again replaced by the cell integral mean, i.e., $\Pi_{\mathrm{CJM},K}(u_h) := \overline{u}_{h,K}$.

ConstJumpNorm

Based on the previous approach, the method *ConstJumpNorm* was introduced in [15] that depends on the mean $L^\infty(E)$-norm of the jump of the function u_h. The L^1- and L^2-norms have been investigated as well but significant differences could not be observed [15]. If this jump is larger than a fixed positive reference value $0 < \beta_{\mathrm{ref}} \in \mathbb{R}$ the cell is marked.

To be precise, for each edge $E \in \mathcal{E}_h$ the quantity

$$\beta_E := \begin{cases} \left\| [\![u_h]\!] \right\|_{L^\infty(E)}, & \text{if } E \in \mathcal{E}_h^{\mathrm{I}}, \\ 0, & \text{else,} \end{cases}$$

can be defined. Based on this quantity, the feature mapping

$$\mathcal{F}_{\mathrm{CJN}}(u_h, K) := \{\beta_{E_0}, \beta_{E_1}, \beta_{E_2}, \beta_{\mathrm{ref}}\} \tag{13}$$

can be computed, so that $n_{\mathrm{CJN}} = 4$.

The decision maker function of *ConstJumpNorm* is given by

$$\mathcal{M}_{\mathrm{CJN}}(\mathcal{F}_{\mathrm{CJN}}(u_h, K)) := \begin{cases} 1, & \text{if } \max_{i=0,1,2} \beta_{E_i} \geq \beta_{\mathrm{ref}}, \\ 0, & \text{else,} \end{cases} \tag{14}$$

and the solution is approximated by $\Pi_{\mathrm{CJN},K}(u_h) := \overline{u}_{h,K}$.

3 Deep Neural Networks as Spurious Oscillations Detector

Deep learning techniques such as deep (neural) networks are a subpart of machine learning which try to approximate a possibly unknown function by learning it from data [21, p. 1–8]. In the following, multilayer perceptrons (MLPs) also known as feed forward neural networks are briefly introduced; see also [21, 42, Chapter 6] for detailed information.

Mathematically speaking, MLPs can be seen as functions that map an input domain $\mathcal{X}$ to some output domain $\mathcal{Y}$ by composing a sequence of functions $g_1, g_2, \ldots, g_\ell$, i.e.,

$$x \mapsto g_\ell(g_{\ell-1}(\ldots g_1(x))\ldots) \in \mathcal{Y} \qquad (x \in \mathcal{X}).$$

Here each g_i, $i = 1, 2, \ldots, \ell$, also called ith layer has the form $g_i(\bullet) = \sigma_i(W_i \bullet + b_i)$, where W_i is a rectangular matrix called weight matrix, b_i is a vector called bias vector, σ_i is a component wise defined nonlinear function called activation function. The first layer is called input layer, the last layer is called output layer and the layers in between hidden layers. In other words, starting with x as value(s) of the input layer each following layer takes as input all the output of the previous layer, also called nodes, performs an affine transformation and applies component wise an activation function. MLPs can be therefore characterized or rather parameterized by their corresponding weights, biases and activation functions, which is why they are often referred to as *parameters*. Different activation functions can be used, but what they all have in common is that they are nonlinear, which is crucial to approximate nonlinear functions [21, p. 168]. Possible choices are the sigmoid function $\sigma(x) = 1/\left(1 + e^{-x}\right)$, the rectified linear unit (ReLU) function $\sigma(x) = \max\{0, x\}$ or the

hyperbolic tangent $\sigma(x) = \tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$. To reach the goal that a MLP approximates a given function, the parameters need to be chosen accordingly. They are chosen in an optimization process that is called *training*.

Let $F : \mathcal{X} \to \mathcal{Y}$ be the function that will be approximated by a MLP. During the training the parameters are optimized to minimize a given loss function $\mathcal{L}$ over a given finite data set $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ which consists of pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ of features x_i and labels $y_i = F(x_i)$.[1]

In this work, different approaches are investigated to approximate (combinations of) the decision maker functions $\mathcal{M}_{\mathrm{LTR}}$, $\mathcal{M}_{\mathrm{CTR}}$, $\mathcal{M}_{\mathrm{CJM}}$, and $\mathcal{M}_{\mathrm{CJN}}$ by MLPs, see below. The concrete choice $F = \mathcal{M}_{\mathrm{LTR}}$, $\mathcal{X} = \mathrm{Im}(\mathcal{F}_{\mathrm{LTR}}) \subset \mathbb{R}^8$, $\mathcal{Y} = \{0, 1\}$,

$$\mathcal{L}(D) := -\frac{1}{N} \sum_{i=1}^{N} y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i), \tag{15}$$

where N is the number of training data in $\mathcal{D}$ and $\hat{y}_i$ is the prediction of the MLP, may serve as a simple example and is used in Sect. 4.2. This loss function is usually called *binary cross entropy loss*.

During the training the parameters p are updated by

$$p \to p - \eta \nabla_p \mathcal{L}(p),$$

where $0 < \eta \in \mathbb{R}$ is a positive step width, also called learning rate, and ∇_p denotes the partial derivatives with respect to the parameters. In this work the minibatch stochastic gradient descent [21, 42, Chapter 8.1.3] is used together with the Adam algorithm [43] to adapt the step width automatically.

3.1 Generating the Data Set

To enable the MLP to approximate a given function training data is needed. As stated above, decision maker functions are approximated that take as input a feature vector of the solution on a single cell and return either 1 (mark the cell) or 0 (do not mark a cell). To generate training data the following problem is fixed.

Example 1 [HMM example] Let $\Omega = (0, 1)^2$ be the unit square and $\boldsymbol{b} = (\cos(-\pi/3), \sin(-\pi/3))^T$, $c = f = 0$. On the whole boundary Dirichlet boundary conditions are prescribed, i.e., $\Gamma_{\mathrm{D}} = \partial\Omega$, by choosing

$$g = \begin{cases} 1 & (y = 1 \wedge x > 0) \text{ or } (x = 0 \wedge y > 0.75), \\ 0 & \text{else.} \end{cases}$$

This example is a modification of a classical benchmark problem stated in [36] in which the discontinuity point of the Dirichlet boundary conditions is chosen slightly different to match the requirements of applying a DG method on a uniform grid.

For small diffusion coefficients ε, the solution possesses two boundary layers at the outflow boundary and an interior layer in the direction of the convection, see Fig. 1 for a sketch of the solution.

To generate training data, the discrete problem (3) can be solved on a series of uniformly refined meshes starting with the initial meshes depicted in Fig. 2. On each level, the discrete solution is calculated and afterwards on each cell the features of *LinTriaReco*, *ConstTriaReco*,

[1] This is so-called supervised learning. See [21, p. 103–105] for unsupervised and reinforcement learning.

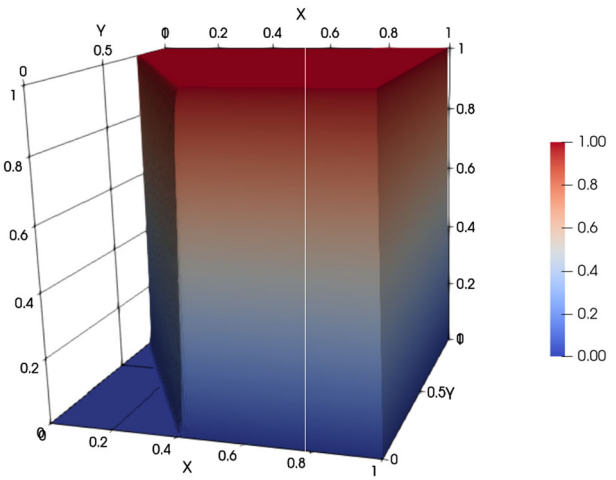

Fig. 1 Sketch of the solution to Example 1 for $\varepsilon = 10^{-8}$ obtained with a nonlinear algebraic flux-corrected (AFC) finite element method with Kuzmin limiter, see [44]

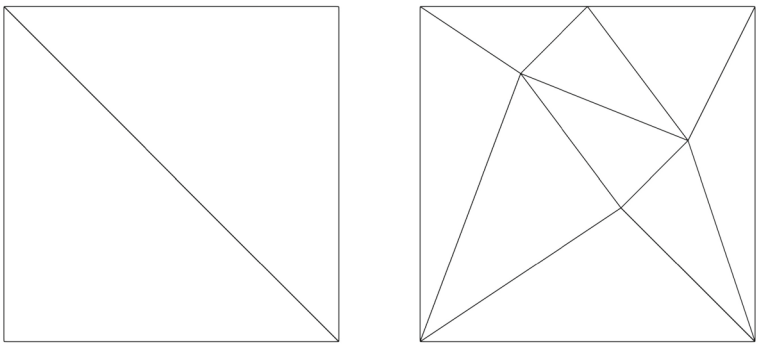

Fig. 2 Initial meshes for Example 1. The grid on the left-hand side is referred to as regular and the grid on the right-hand side as irregular grid

ConstJumpMod, *ConstJumpNorm* and the corresponding labels are computed and stored, see Eqs. (7)–(13). Since a data point for each *cell* is created, a lot of data can be generated easily since the number of cells scales quadratically when the grid is refined. Here it comes in handy that the decision maker functions act locally.

The data is generated using discontinuous piecewise linear finite elements P_1 for the above mentioned problem with a diffusion coefficient $\varepsilon = 10^{-8}$. The SIPG discretization is chosen with upwind stabilization, i.e., $\kappa = 1$ and $\eta = 1$ in Eqs. (4) to (6). Let n_0 be the number of vertices in each cell, i.e., $n_0 = 3$ on the triangular grids depicted in Fig. 2. Guided by [5, Chapter 2.7.1], the penalty parameter $\sigma = 2\varepsilon n_0(p+1)(p+2)/2 = 18\varepsilon$ is used. All simulations are performed with ParMooN [45, 46] and the direct solver UMFPACK [47] is used to solve the linear systems of equations. In *LinTriaReco* and *ConstTriaReco* the tolerance tol is set to 10^{-11}. For *ConstJumpMod* the parameters $C_0 = 1$, $L = 1.5$ and $u_0 = 1$, and $\alpha_{\text{ref}} = 4$ are chosen. Lastly, the reference value β_{ref} is set to be the arithmetic mean of all β_E in *ConstJumpNorm*.

3.1.1 Rotation Invariance of the Data

Unfortunately, the features above described and defined in Eqs. (7), (9), (11) and (13) depend on the numbering of the edges and hence, so does the data. To overcome this problem, or in other words to introduce some sort of rotation invariance, each data point in the data set is stored three times, one for each particular counter clockwise numbering of the edges.

3.1.2 Magnitude Invariance

In [27] the authors have decided to scale the features to introduce some form of magnitude invariance. In contrast to this, here the features are *not* scaled. The reason for this is that all decision maker functions essentially compare the magnitude of a feature with other features. If features are scaled feature-wise as in [27], the ratio of the magnitude of features can be changed. In this way inconsistent data can be created, i.e., the label does not fit to the data anymore. To prevent this situation, a scaling of the features is therefore not applied.

3.2 Restricting the Data Set

Following the above describe procedure a lot of data can be generated, e.g., refining the regular grid nine times and the irregular grid eight times leads to 4.456.437 data points. Unfortunately, a lot of duplicates exist in the data, e.g., due to the fact that the solution of the problem is piecewise constant in huge parts of the domain and hence, the features can be equal. This can be the case for an individual limiter but also for any combination of limiters, e.g., also for all limiters at the same time. Our approach consists in removing the duplicates to prevent the MLP from learning a pattern specific to the duplicates and to prevent overfitting to the duplicates and hence, ending up in a MLP that does not generalize well to unseen data. That is, either the duplicates in the data of a single limiter are removed if a single limiter is learned, or duplicates of the data of all limiters if all limiters are learned at the same time, see also the numerical examples below.

After having removed the duplicates it can further be noted that there are for each limiter individually significantly less marked cells than unmarked cells, e.g., for *LinTriaReco* after removing the duplicates there are around 77% cells that are not marked and 23% marked cells. When inspecting the whole data set it can be seen that cells that are not marked by any limiter are more common (ca. 93.6%) than cells that are marked at least by one limiter (ca. 6.4%). It is well known that care has to be taken when it comes to such unbalanced data sets [48, Chapter 11.2]. To have a better balance between marked cells and unmarked cells, resp., in the distribution of the label combinations, the data set is further restricted to have either as many marked cells as cells that are not marked in the case that a single limiter is approximated or the amount of the combination where no limiter marks a cell is reduced to equal the amount of the second most occurring label combination in the case all limiters are approximated at a single time. The rows that are removed are chosen randomly using a fixed random seed to guarantee reproducibility.

3.3 Splitting the Data Set

Even after deleting duplicates and decreasing the amount of cells that are not marked, resp., the amount of the most occurring label combination, a lot of training data remains, e.g., 379.539 when all limiters are learned at the same time, and 260.436 if only *ConstJumpNorm*

is considered. On the one hand, the more data exists the more likely it is that the network approximates the function that lies behind the data, but on the other hand, more training data increases the optimization time when the network is trained. Hence, it is recommended to have less data of higher quality, i.e., showing relevant features of the function, than more data with lower quality. In this data set it might be difficult to choose "good" data points a priori but it might be still useful to choose only a subset of the data points for performance reasons. Therefore, a subset of only 7500 data points is randomly chosen to be the training data for the networks.

Furthermore, to prevent overfitting of the data, another 1875 are chosen to be the validation data set, see also [21, Chapter 5.3] for an introduction to overfitting and validation sets. During the training the validation set is evaluated as well to see how well the network generalizes to unseen data. At some point the networks might only fit better the training data but they become worse on the validation data set, which is why the optimization can be stopped at this point to prevent overfitting.

Last but not least, a third set is introduced with which the trained networks are rated how well they work. The so-called test set consists of the validation set and all remaining data. After the training has finished the networks are applied on the test set to measure the overall performance of the networks.

3.4 Measuring the Performance of the Networks

To measure the performance of the trained networks the measures

$$\begin{aligned} \text{accuracy} \quad & \text{acc} := \frac{t_p + t_n}{t_p + f_p + t_n + f_n}, \\ \text{precision} \quad & \text{prc} := \frac{t_p}{t_p + f_p}, \\ \text{recall} \quad & \text{rec} := \frac{t_p}{t_p + f_n}, \end{aligned}$$

are used, where t_p denotes the true positive, t_n the true negative, f_p the false positive and f_n the false negative classifications. The measure accuracy is the ratio of correct classified data to all data, i.e., it measures how good the networks performs overall. While the second measure gives information about the proportion of positive classifications that was actually correct, recall answers what proportion of actual positives was identified as such. Since for reducing spurious oscillations it is worse to not detect a cell that should be marked than to mark a cell that should not, recall might be considered more important than precision. Therefore, the total rating r_{tot} of the limiters is set to be a weighted combination of the measures, namely

$$r_{\text{tot}} := \frac{2}{5}\text{acc} + \frac{1}{5}\text{prc} + \frac{2}{5}\text{rec},$$

where acc, prc and rec are computed based on the test set.

4 Numerical Studies

For the implementation of the MLP networks the open source library TensorFlow is used [49, 50]. As stated above, the finite element computations are performed with ParMooN [45, 46]

Table 1 Hyperparameters that are tested resulting in 648 different combinations

Hidden layers	[256, 128, 64], [128, 128, 128], [256, 128, 64, 32]
	[100, 100, 100, 100], [256, 128, 64, 32, 16], [90, 90, 90, 90, 90]
Learning rate	0.005, 0.001, 0.0005, 0.0001, 5×10^{-5}, 1×10^{-5}
Batch size	32, 64, 128
Activation	ReLU, tanh
Initialization seed	40, 41, 42

and CppFlow [51] is used to open and deploy stored TensorFlow models in ParMooN. Note that the data and most parts of the code that are used in this section are publicly available at www.doi.org/10.20347/40vd-f944 [40].

4.1 Architecture of the MLPs

Given a specific mapping that should be approximated by a MLP, it is in general almost impossible to come up a priori with the optimal architecture of the MLP, i.e., the number of layers, activation functions, number of nodes per layer. To find a suitable architecture, in this work, different architectures are tested. Six different combinations of number of hidden layers and number of nodes which corresponds to the number of columns in the weight matrices W_i, two different activation functions, three different batch sizes, six different learning rates and three different initializations of the parameters are used which are coded by different seeds, resulting in 648 different architectures that are investigated, see Table 1. Each combination therefore can also be identified by a number between one and 648 which is done below. The size of the input and the output layer are determined by the task to solve. While for all hidden layers the same activation function is used, i.e., one of the functions given in Table 1, the activation function for the output layer depends on the experiment and is therefore stated in the experiments below. The parameters of the layers are initialized using the Glorot normalized initialization [52] with different seeds for each layer based on the seeds given in Table 1. Also the loss function $\mathcal{L}$ depends on the experiment and hence is given below.

The networks are trained for at most 10000 epochs and the training is stopped earlier, if the loss of the validation set does not decrease for 100 epochs. The model with the best accuracy is then saved as trained model.

4.2 Learning Single Limiters

The first experiment figures out whether the individual feature mappings (8), (10), (12), and (14) can by approximated by a MLP. Since for all functions $\mathcal{Y} = \{0, 1\}$, the output layer consists of a single node and uses the sigmoid activation function. The size of the input layer is defined by the input of the decision maker functions, i.e., eight for *LinTriaReco* and *ConstTriaReco*, and four for *ConstJumpMod* and *ConstJumpNorm*. As loss the binary cross entropy loss $\mathcal{L}(D)$ given in (15) is applied. The data is prepared as described in Sects. 3.1 to 3.3 and the measures defined in Sect. 3.4 are used to measure the performance of the networks.

The total rating r_{tot} for the networks for each limiter is plotted in Fig. 3, where *MLP* (*lim*) denotes the *MLP* networks that approximate the decision maker function of lim. In general

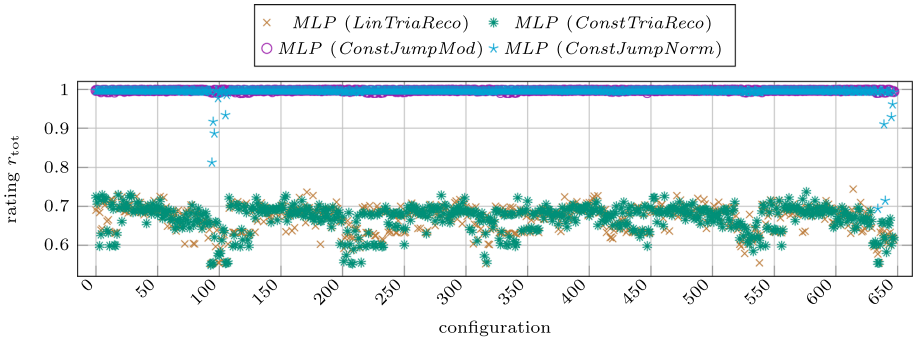

Fig. 3 Rating for all architectures for all limiters for Sect. 4.2

Table 2 Statistics about the total rating of the trained networks of Sect. 4.2

	r_{tot} *LinTriaReco*	r_{tot} *ConstTriaReco*	r_{tot} *ConstJumpMod*	r_{tot} *ConstJumpNorm*
Max	0.744	0.737	0.999	0.997
Mean	0.669	0.670	0.997	0.994
Std	0.034	0.035	0.001	0.019

Standard deviation is abbreviated by std

Table 3 Pearson correlation coefficients between the hyperparameters and the total ratings from Sect. 4.2

	r_{tot} *LinTriaReco*	r_{tot} *ConstTriaReco*
Hidden layers	− 0.018	− 0.004
Learning rate	− 0.521	− 0.413
Batch size	− 0.057	− 0.068
Activation	− 0.372	− 0.394
Initialization seed	0.027	0.029

it can be seen that the results of *MLP* (*LinTriaReco*) and *MLP* (*ConstTriaReco*) look similar as well as the results of *MLP* (*ConstJumpMod*) and *MLP* (*ConstJumpNorm*). On the one hand, all architectures are able to approximate *ConstJumpMod* very well and *ConstJumpNorm* can be approximated by almost all architectures. On the other hand, *LinTriaReco* and *ConstTriaReco* cannot be approximated that well with the investigated architectures. The best total rating for *MLP* (*LinTriaReco*) and *MLP* (*ConstTriaReco*) is still around 0.253 worse than the mean of *MLP* (*ConstJumpMod*) and *MLP* (*ConstJumpNorm*), see also Table 2. As indicated by the standard deviation, the quality of the approximation of *MLP* (*LinTriaReco*) and *MLP* (*ConstTriaReco*) depends more on the choice of the architecture than of *MLP* (*ConstJumpNorm*), which in turn is more dependent than the approximation of *MLP* (*ConstJumpMod*). It can further be noted that in Fig. 3 there is a pattern indicating which architectures work worse for *MLP* (*LinTriaReco*) and *MLP* (*ConstTriaReco*). The Pearson correlation coefficients between the hyperparameters and the ratings of *MLP* (*LinTriaReco*) and *MLP* (*ConstTriaReco*) are given in Table 3. The correlation coefficients for *MLP* (*ConstJumpMod*) and *MLP* (*ConstJumpNorm*) are not investigated since almost all architectures lead to good results. It can be seen that the learning rate and the activation function have the largest impact. From the obtained results it can be deduced that the learning rate should

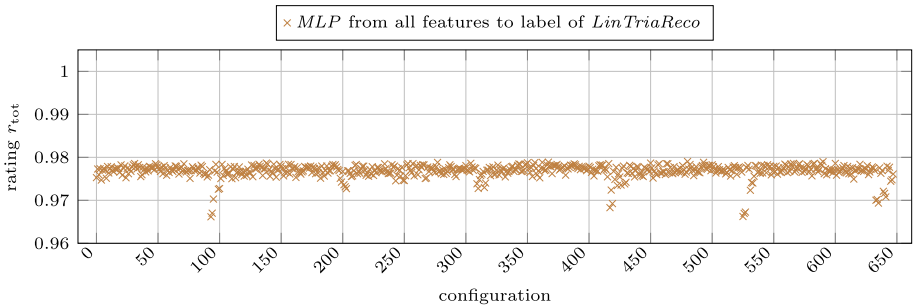

Fig. 4 Results for all architectures from Sect. 4.3

Table 4 Statistics about the total rating of the trained networks of Sect. 4.3

	r_{tot} *LinTriaReco*
Max	0.979
Mean	0.977
Std	0.002

Standard deviation is abbreviated by std

not be chosen too small and after looking into the data it can be observed that the ReLU activation function works better than tanh. This might be a reason for the pattern in Fig. 3.

4.3 Overcoming the Difficulties When Learning *LinTriaReco* and *ConstTriaReco*

As seen in the previous experiment, the decision maker functions of *LinTriaReco* and *ConstTriaReco* could not be approximated well and *ConstJumpMod* and *ConstJumpNorm* could be approximated by the chosen architectures if only the features of the respective limiter are used. This experiment investigates if enriching the feature set enables the architectures to predict the outcome of the decision maker function of *LinTriaReco*. The hope is that there is an implicit dependency between this enriched feature space and the outcome of *LinTriaReco* and that the MLPs can approximate this mapping better than $\mathcal{M}_{\text{LTR}}$ itself. To this end, the output layer consists again of a single node and the sigmoid activation function is used. The idea in this experiment is to use all features of *LinTriaReco*, *ConstTriaReco*, *ConstJumpMod*, and *ConstJumpNorm*. Hence, the input layer, in contrast to the previous experiment, is now larger and consists of $n_{\text{LTR}} + n_{\text{CTR}} + n_{\text{CJM}} + n_{\text{CJN}} = 24$ nodes. As a consequence, this experiment allows us to investigate whether there is a hidden dependency between the features of all these limiters and the label of *LinTriaReco*. Again the binary cross entropy loss (15) is applied to train the networks. The data is loaded, restricted and split as before and the measure r_{tot} is used to rate the trained MLPs.

Figure 4 shows the result for the tested architectures. All tested architectures have a similar good rating, i.e., it seems that there is a mapping from all features to the label of *LinTriaReco* that can be approximated with the used architectures. The best rating (0.979) is slightly worse than the best results for *ConstJumpMod* (0.999) and *ConstJumpNorm* (0.997) of the previous experiment but could increase the rating of *LinTriaReco* by around 0.235. Also all architectures are stable in the sense of producing similar good results as shown by the mean that is close to the best rating and the small standard deviation, see Table 4. Pearson correlation coefficients are not shown since all architectures are stable.

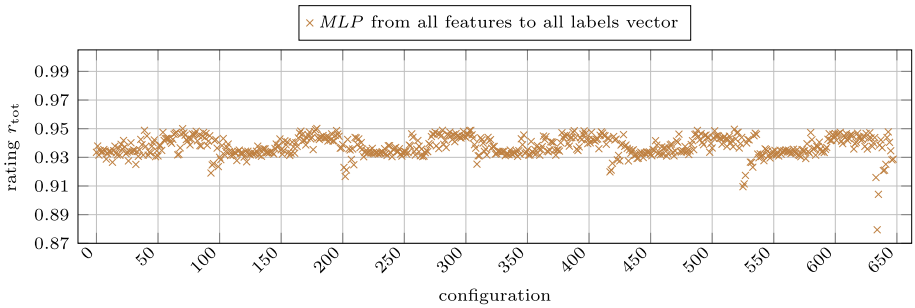

Fig. 5 Results for all architectures from Sect. 4.4

Altogether, the results of this and the former experiment seem to indicate that the features of *LinTriaReco* and *ConstTriaReco* might not be suited best for deciding whether to mark a cell or not. In contrast to this, the features of *ConstJumpMod* and *ConstJumpNorm* seem to provide better information and hence are more suited, since also the label of *LinTriaReco* can be approximated if these features are part of the input to the MLPs.

4.4 Learning All Limiters Simultaneously Based on Vectors

The previous experiment raises the questions whether it is possible to learn all decision maker functions at once. The idea is to train a network that approximates the map from all features to all labels simultaneously, i.e., the size of the input layer is $n_{\text{LTR}} + n_{\text{CTR}} + n_{\text{CJM}} + n_{\text{CJN}} = 24$ and the output size is four, since we want to approximate four decision maker functions at once. In this sense the problem is a multi-label classification task. As in Sect. 4.2, the activation function of the output layer is set to be the sigmoid function such that the output of the network is in $[0, 1]^4$. This means that by construction the networks return a vector of four predicted labels at once. Furthermore, the loss

$$\mathcal{L}(D) := \frac{1}{4} \sum_{j=1}^{4} \left(-\frac{1}{N} \sum_{i=1}^{N} y_{i,j} \ln(\hat{y}_{i,j}) + (1 - y_{i,j}) \ln(1 - \hat{y}_{i,j}) \right)$$

is used, where N is again the number of training data in $\mathcal{D}$, $y_{i,j}$ is the jth component of the ith training data point and $\hat{y}_{i,j}$ is the jth component of the prediction $\hat{y}_i$ of the MLP. Comparing with Eq. (15), this loss is the average of the binary cross entropy loss of the four decision maker functions that are learned at once. The data is prepared following the procedure described in Sects. 3.1 to 3.3 and the total measure r_{tot} from Sect. 3.4 is used to rate the trained MLPs. The measures are evaluated element-wise and not vector-wise for the output of the MLPs.

The results for all configurations are depicted in Fig. 5. It can be seen that almost all configurations are able to approximate all decision maker functions at once. As indicated also by Table 5 the best network achieves a total rating of around 0.95 which is slightly lower than the best rating in Sect. 4.2 but can still considered to be a good result. Both the mean of 0.938 that is close to the maximal total rating and the small standard deviation (0.007) indicate that there are only few configurations that work worse than the best one. For the sake of brevity and due to the small standard deviation, Pearson correlation coefficients are

Table 5 Statistics about the total rating of the trained networks of Sect. 4.4. Standard deviation is abbreviated by std

	r_{tot}
Max	0.950
Mean	0.938
Std	0.007

not shown. However, the learning rate has the largest impact with a coefficient of -0.116, supporting the result from Sect. 4.2.

4.5 Learning All Limiters Simultaneously Based on Classes

The multi-label problem of the previous section can be transformed to a multi-class problem. After preparing the data as before, every unique label combination is assigned to a unique number, e.g.,

$$[0,0,0,0] \mapsto 0, \qquad [0,0,0,1] \mapsto 1, \qquad [0,0,1,0] \mapsto 2$$

and so on. This number j is then assigned to a probability vector, i.e., the components are non-negative and sum to 1, by mapping j to the vector $[0,\ldots,0,1,0,\ldots,0]$ that has the 1 at the jth component. Every entry in this vector gives the probability that the input belongs to the respective class. The input layer size is again 24 and the output layer size is the number of classes which are in this experiment 12 since not all possible label combinations occur in the dataset. After splitting the data into training, validation, and test set it can be observed that not all label combinations occur in the training set since some combinations are very rare. The activation function of the output layer is chosen to be the softmax function $\sigma(x)_i = \exp(x_i)/\sum_{j=1}^{12}\exp(x_j)$, $i = 1,\ldots,12$, such that the output is a probability vector. Hence, in contrast to the previous section, the output gives probabilities that the input belongs to the possible label combinations and does not return a specific combination. The usual loss for multi-class problems is used, namely the categorical cross entropy

$$\mathcal{L}(D) := -\sum_{i=1}^{N}\sum_{j=1}^{12} y_{i,j}\ln(\hat{y}_{i,j}),$$

where N is the number of training data points, $y_{i,j}$ is the jth component of the ith training data vector and $\hat{y}_{i,j}$ the prediction of the network. Note that $y_{i,j}$ is 0 for all except one entry where it is 1. To measure the performance of the networks the outputs of the networks are mapped back to label vectors by multiplying the probability of the classes with the corresponding label vectors of the classes and summing up the results. In other words, a weighted sum of all label vectors is computed where the weights correspond to the predicted probabilities. This procedure gives a vector of four labels that can be compared with the corresponding true labels in the test set, to identify the true positives, true negatives, false positives and false negatives. Afterwards the measures given in Sect. 3.4 can be computed.

In Fig. 6 the total rating of the trained networks is plotted. The results are similar to the results of Sect. 4.4 as also indicated by the values in Table 6. The best and the mean are negligibly smaller than the values obtained in the previous experiment. Hence, it does not matter whether to deal with the problem as a multi-label or a multi-class problem, at least in this particular setting with the used training set and the measures.

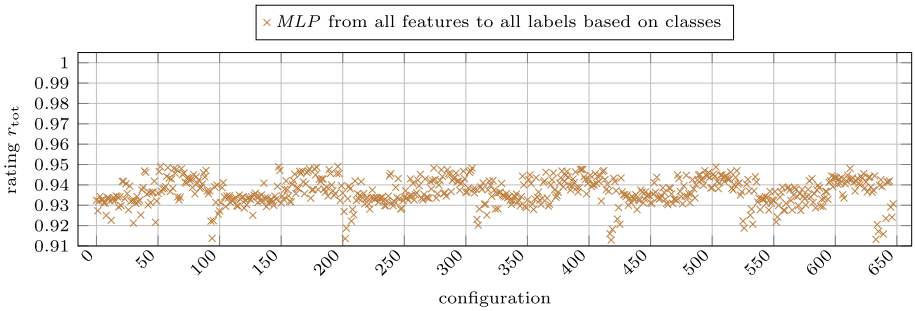

Fig. 6 Results for all architectures from Sect. 4.5

Table 6 Statistics about the total rating of the trained networks of Sect. 4.5

	r_{tot}
Max	0.949
Mean	0.937
Std	0.007

Standard deviation is abbreviated by std

4.6 Applying a MLP Limiter to Higher Polynomial Degrees

Until here the experiments have investigated how good the MLPs can approximate the data but they have not been applied to the DG solution of (other) convection-dominated convection–diffusion equations. To this end, the MLP from Sect. 4.4 with the best total rating is used, which has four hidden layers of 100 nodes and the hyperbolic tangent as activation function, and is trained with a learning rate of 0.0005, a batch size of 128, and is initialized with seed 42. After the DG solution of a convection–diffusion problem is solved, all features of the conventional limiters are calculated and the MLP is asked to predict the label combination given these features. A cell is finally marked if at least $n \in [1, 2, 3, 4]$ of the four predicted labels are true, i.e., larger than 0.5. If a cell is marked then the solution is locally replaced by its integral mean, i.e., $\Pi_{\text{MLP},K}(u_h) := \overline{u}_{h,K}$ since this choice has produced the best results in [14] and [15]. This limiter is below called *MLP* limiter.

4.6.1 Determining the Minimum Number of Predicted Marks *n*

Since it is not a priori clear which value of n to choose, this is determined in a first step. The smaller n, the more cells are marked, which on the one hand hopefully leads to less spurious oscillations but on the other hand, marking too many cells might reduce the order of accuracy and leads to unnecessary computational overhead. Therefore, n should be chosen in such a way that enough but not too many cells are marked. To find the optimal n, Example 1 is used with exactly the same setting as in Sect. 3.1, i.e., the setting with which the data is created. Since the limiter is trained with this data it can be expected that it predicts the labels of the traditional limiter correctly in most of the cases. Since for Example 1 an analytical solution is not known, the discrete solution u_h cannot be compared against the exact solution. As in [14, 15], to assess the quality of the limited discrete solution therefore the measures

$$\text{osc}_{\max}(u_h) = \max_{(x,y)\in\overline{\Omega}} u_h(x, y) - u_{\max} + u_{\min} - \min_{(x,y)\in\overline{\Omega}} u_h(x, y),$$

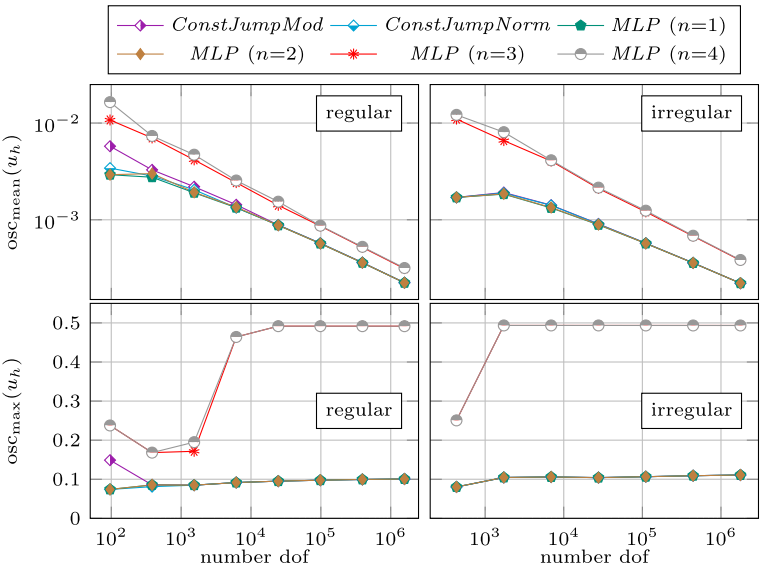

Fig. 7 Results of $\mathrm{osc}_{\mathrm{mean}}$ and $\mathrm{osc}_{\mathrm{max}}$ for Example 1 on the regular and irregular grid depicted in Fig. 2 for P_1 finite elements with two classical limiters and various versions of the *MLP* limiter

$$\mathrm{osc}_{\mathrm{mean}}(u_h) = \frac{1}{|\mathcal{T}_h|} \sum_{K \in \mathcal{T}_h} \Big[\max\{0, \max_{(x,y)\in K} u_h(x,y) - u_{\max}\} + \max\{0, u_{\min} - \min_{(x,y)\in K} u_h(x,y)\} \Big],$$

are used to measure the maximal size and a mean value of spurious oscillations, where $u_{\max}$ and $u_{\min}$ are the largest and smallest value of the weak solution, resp., and $|\mathcal{T}_h|$ denotes the number of cells in the triangulation. In Example 1 it is $u_{\min} = 0$ and $u_{\max} = 1$. To compute the desired quantities, u_h is evaluated at certain points, which are the points of the nodal functionals defining continuous P_p finite elements of the same order.

The results of the *MLP* limiter with $n = 1, 2, 3, 4$ on both the regular and the irregular grid are shown in Fig. 7 and compared with *ConstJumpMod* and *ConstJumpNorm*, which are the classical limiters that work best for this problem [15].

It can be seen that, on the one hand, the *MLP* limiter with $n = 1$ behaves similarly to the one with $n = 2$, and, on the other hand, the limiters where $n = 3$ and $n = 4$ show almost no difference. While the former ones are as good as the classical *ConstJumpMod* and *ConstJumpNorm* limiters, the latter ones behave much worse, meaning they lead to larger mean and maximal oscillations. As a consequence, in what follows, the *MLP* limiter with $n = 2$ is used since it produces better results than the ones with $n = 3, 4$ and should by definition mark less or the same amount of cells than the one with $n = 1$. Of course, it is not guaranteed that $n = 2$ is the optimal choice also in other scenarios, however this experiment indicates that it is a reasonable choice and the experiments below confirm the choice.

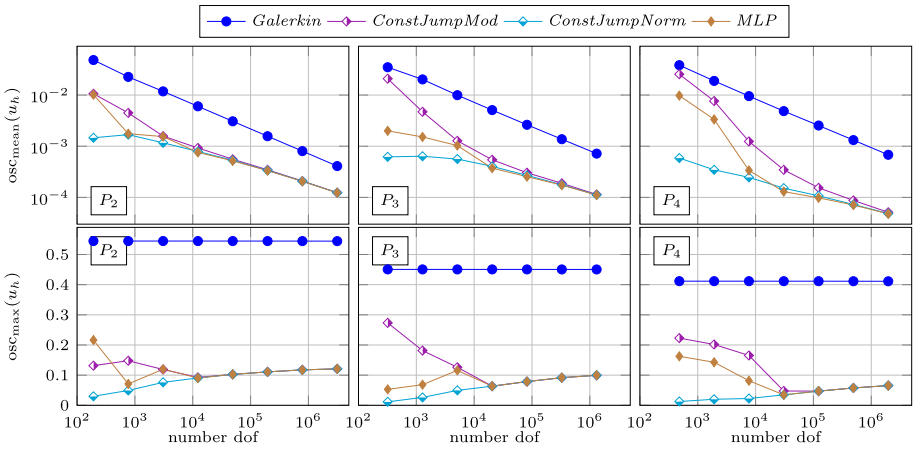

Fig. 8 Results for measures for various limiters and various polynomial degrees obtained for Example 1 on the regular grid from Fig. 2

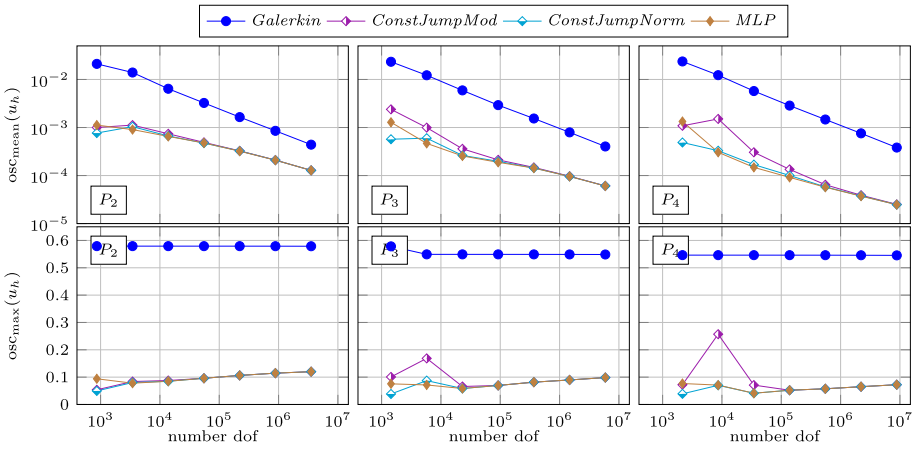

Fig. 9 Results for measures for various limiters and various polynomial degrees obtained for Example 1 on the irregular grid from Fig. 2

4.6.2 Higher Polynomial Degrees

Since the *MLP* limiter is fixed it can be applied to other problems. To start, again Example 1 is used but the limiter is applied to the discrete solution obtained with finite elements with higher polynomial degrees, namely P_2, P_3, and P_4 finite elements. The rest of the problem is not varied, i.e., $\varepsilon = 10^{-8}$, $\kappa = 1$, $\eta = 1$. As in Sect. 3.1 the penalty parameter is chosen to be $\sigma = 2\varepsilon n_0(p+1)(p+2)/2$, where again n_0 denotes the number of vertices a cell has. Also the parameters used in the classical limiters are kept the same. The problems are solved on the series of uniformly refined grids starting as above with the initial meshes depicted in Fig. 2. In what follows *Galerkin* denotes the DG solution from Eq. (3) without being limited.

The results for the measures $\mathrm{osc}_{\mathrm{mean}}$ and $\mathrm{osc}_{\mathrm{max}}$ for the best conventional limiter as well as the *MLP* limiter on both types of meshes are shown in Figs. 8 and 9. It can be seen that the *MLP* limiter reduces both the mean and the maximal oscillations significantly compared to

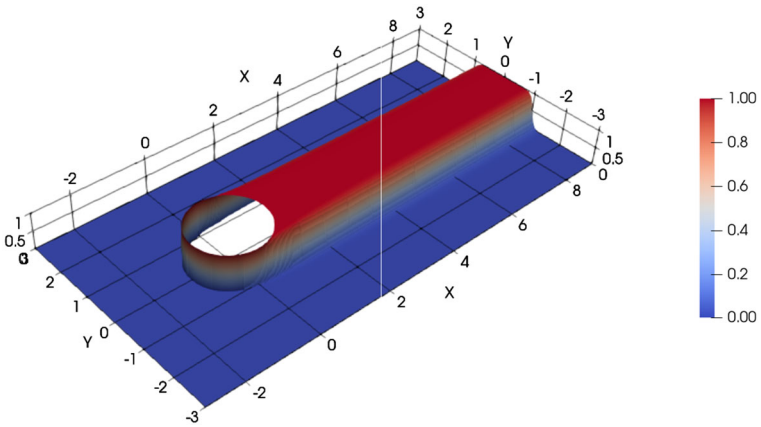

Fig. 10 Sketch of the solution to Example 2 for $\varepsilon = 10^{-8}$ obtained with a nonlinear algebraic flux-corrected (AFC) finite element method with Kuzmin limiter, see [44]

Galerkin. While on coarser grids it acts worse than *ConstJumpNorm* but better than *ConstJumpMod*, on finer grids all limiters almost coincide. A reason for this could be the fact that way more training data obtained on finer grids is available compared to data from coarser grids, since the number of available data scales exponentially with the number of the refinement.

4.7 Applying a MLP Limiter to the Hemker Problem

Finally, in this section the *MLP* limiter is applied to a different example, namely the Hemker benchmark problem. It was proposed in [37] and it is a very popular benchmark problem for convection-dominated convection–diffusion equations. It models the transport of energy from a body through a channel and shows many features of problems that are also relevant in applications. The structure of the solution is similar to the solution of the HMM example, e.g., it is constant in most regions, and hence there is hope that the *MLP* limiter is able to limit the solution in a reasonable way.

Example 2 (Hemker example) The problem is stated in $\Omega = \{(-3, 9) \times (-3, 3)\} \backslash \{(x, y) : x^2 + y^2 \leq 1\}$, and has the coefficients $\boldsymbol{b} = (1, 0)^T$, $c = f = 0$. If $x = -3$ and at the circular boundary, Dirichlet boundary conditions are prescribed by setting

$$g = 0 \text{ if } x = -3, \qquad g = 1 \text{ at the circle.}$$

Everywhere else homogeneous Neumann conditions are applied. The solution is sketched in Fig. 10 and takes values in $[0, 1]$.

As before, the diffusion coefficient is set to $\varepsilon = 10^{-8}$ and $\kappa = 1$, $\eta = 1$, and $\sigma = \varepsilon n_0(p + 1)(p + 2)$ are used as parameters in the DG method. The problem is solved on a series of grids starting from the one depicted in Fig. 11. The characteristic length scale for this problem is $L = 13.5$ and the remaining parameters of the limiter stay the same.

In Fig. 12 the results for both measures for the limited solution and the original solution for various polynomial degrees are shown. As before *ConstJumpMod*, *ConstJumpNorm*, and the *MLP* limiter are able to reduce the oscillations significantly compared to *Galerkin*. For

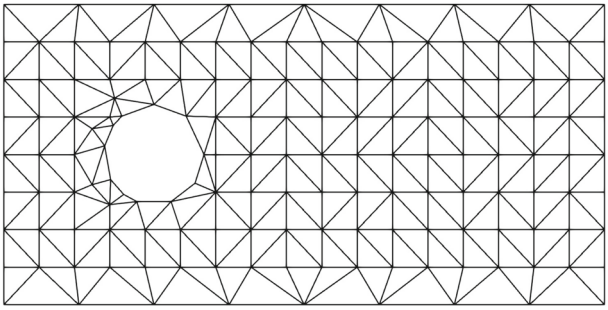

Fig. 11 Initial mesh for Example 2 used in Sect. 4.7

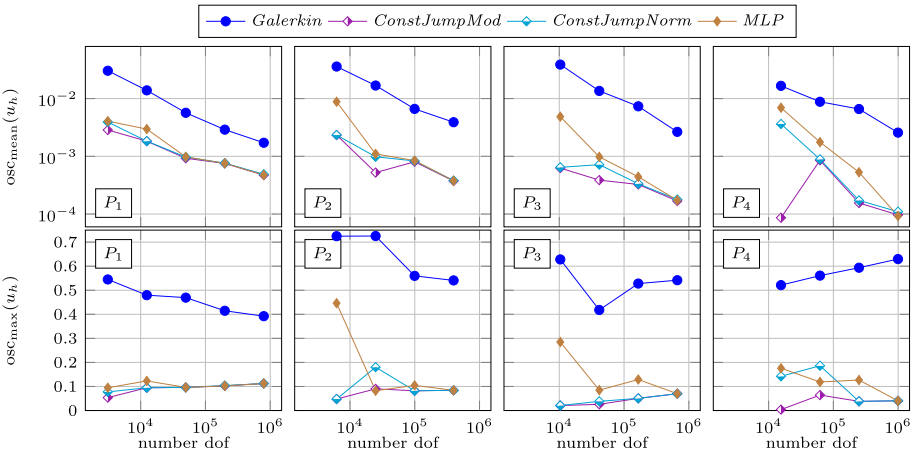

Fig. 12 Results for measures for various limiters and various polynomial degrees obtained for Example 2

P_1 and P_2 the *MLP* limiter is slightly worse than the traditional ones on coarser grids but on finer grids it behaves equally well. For P_3 and P_4 it is worse than the classical companions, except for the finest grid and except for P_4 for $\mathrm{osc}_{\mathrm{max}}$ on one coarser level. We observed that the *MLP* limiter is always better than *LinTriaReco* in both measures, better than *ConstJumpNorm* for $\mathrm{osc}_{\mathrm{max}}$ for all polynomial degree, and for $\mathrm{osc}_{\mathrm{mean}}$ for P_1 and for P_2 to P_4 on the two finer levels, but these results are not presented for the sake of brevity. A visualization of the limited P_4 solution with *ConstJumpNorm* and *MLP* on the second finest grid is shown in Fig. 13. It can be observed that the *MLP* limiter limits most of the cells correctly but forgets to mark some cells with undershoots. This is also is the reason why it has worse $\mathrm{osc}_{\mathrm{mean}}$ and $\mathrm{osc}_{\mathrm{max}}$ values compared to *ConstJumpNorm*.

5 Summary and Outlook

This paper is a contribution to deeper understanding how neural network based slope limiters can be created and applied. In contrast to previous papers, it was focused on constructing a multilayer perceptron model for limiting the discrete solution of an elliptic problem, namely convection–diffusion equations in the convection-dominated regime. It was shown how data

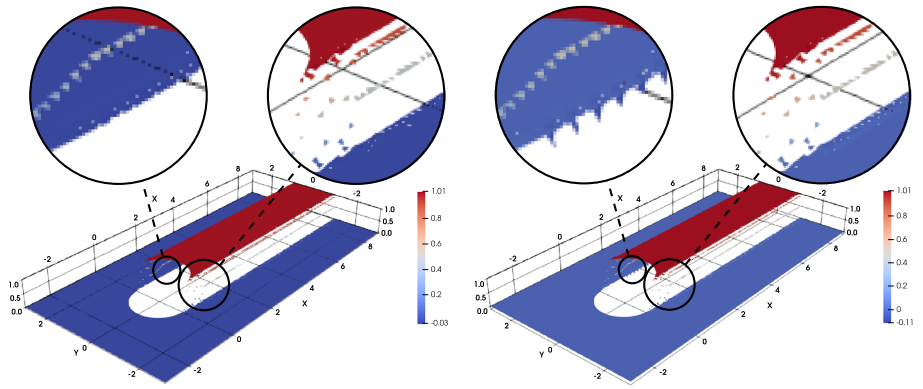

Fig. 13 Discrete P_4 solution to Example 2 limited by the *ConstJumpNorm* limiter (left) and the *MLP* limiter (right)

from a lowest order discretization can be used to train a limiter that then can be applied to the discrete solution of higher order methods. The results have indicated that the limiter works almost equally well as classical methods for higher order methods for the same problem but somewhat worse than these methods when applied to the solution of a different problem.

These results are also in agreement with the findings of previous works that treat hyperbolic problems. In [24] the authors report that their MLP-based limiter works for some problems equally well and for some better than the classical reference approaches. The limiter constructed in [27] behaves equally well or slightly worse than traditional limiters. The authors furthermore discuss the lack of theoretical guarantees which indeed also cannot be provided in the present work. Finally, in [28] it is reported that the MLP-based limiter produces high-quality results even though it is not compared to traditional limiters.

In our opinion, there are four main conclusions to be drawn from the presented studies. First, it could be shown that it is possible to construct MLP-based limiters also for elliptic problems. Second, care has to be taken which features are chosen, since it was observed that features based on jumps are better suited than others. The other two conclusions are of importance for our future work and to reach such insight was actually a main motivation for performing the presented studies. First, it can be concluded that it is possible to classify mesh cells corresponding to subgrid scales and large scales on the basis of MLPs for the class of problems we are interested in. And finally, the concrete architecture of the MLP played only a minor role as long as no extreme values for the hyperparameters were chosen. Hence, the results were robust with respect to the choice of the MLP and pursuing a sophisticated approach for finding an appropriate MLP is not necessary.

As already mentioned in the introduction, we consider the presented study as a proof of concept to show that it is possible to construct MLPs which are capable of appropriately detecting subregions with subgrid scales for numerical solutions of convection–diffusion–reaction equations. Based on their prediction, spurious oscillations in the discrete solution could be reduced significantly in this study. We plan to use the obtained insights for developing MLP-based algorithms for choosing local parameters in stabilized discretizations of convection–diffusion–reaction problems. Among others, this process requires a classification of mesh cells into cells in subgrid scale regions and away of these regions. Since the choice of user-defined parameters is an issue in many numerical methods for many problems, a

medium-range goal consists of extending successful MLP-based techniques to other kinds of boundary value or initial-boundary value problems.

Acknowledgements The authors express their gratitude to Dr. Ulrich Wilbrandt for many time consuming but always valuable discussions.

Funding Open Access funding enabled and organized by Projekt DEAL. No funding was received for conducting this study.

Data Availability Enquiries about data availability should be directed to the authors.

Declarations

Conflict of interest The authors have no financial or proprietary interests in any material discussed in this article.

References

1. Roos, H.-G., Stynes, M., Tobiska, L.: Robust Numerical Methods for Singularly Perturbed Differential Equations: Convection–Diffusion–Reaction and Flow Problems, 2nd Ed., Vol. 24 of Springer Series in Computational Mathematics. Springer, Berlin (2008). https://doi.org/10.1007/978-3-540-34467-4
2. John, V., Knobloch, P.: On spurious oscillations at layers diminishing (SOLD) methods for convection-diffusion equations: Part I - A review. Comput. Methods Appl. Mech. Eng. **196**(17–20), 2197–2215 (2007). https://doi.org/10.1016/j.cma.2006.11.013
3. John, V., Knobloch, P.: On Discontinuity-Capturing Methods for Convection-Diffusion Equations. In: de Castro, A.B, Gómez, D., Quintela, P., Salgado, P. (Eds.), Numerical Mathematics and Advanced Applications, pp. 336–344. Springer, Berlin (2006). https://doi.org/10.1007/978-3-540-34288-5_27
4. Augustin, M., Caiazzo, A., Fiebach, A., Fuhrmann, J., John, V., Linke, A., Umla, R.: An assessment of discretizations for convection-dominated convection–diffusion equations. Comput. Methods Appl. Mech. Eng. **200**(47), 3395–3409 (2011). https://doi.org/10.1016/j.cma.2011.08.012
5. Rivière, B.: Discontinuous Galerkin Methods for Solving Elliptic and Parabolic Equations: Theory and Implementation, Vol. 35 of Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia (2008). https://doi.org/10.1137/1.9780898717440
6. Di Pietro, D.A., Ern, A.: Mathematical Aspects of Discontinuous Galerkin Methods, 1st edn, Vol. 69 of Mathématiques et Applications. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-22980-0
7. Dolejší, V., Feistauer, M.: Discontinuous Galerkin Method: Analysis and Applications to Compressible Flow, 1st edn, Vol. 48 of Springer Series in Computational Mathematics, Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19267-3
8. Reed, W., Hill, T.: Triangular mesh methods for the neutron Transport Equation, Technical Report LA-UR-73-479. Los Alamos Scientific Laboratory, Los Alamos (1973)
9. Dolejší, V., Solin, P.: hp-discontinuous Galerkin method based on local higher order reconstruction. Appl. Math. Comput. **279**, 219–235 (2016). https://doi.org/10.1016/j.amc.2016.01.024
10. Cangiani, A., Dong, Z., Georgoulis, E.H., Houston, P.: hp-Version Discontinuous Galerkin Methods on Polygonal and Polyhedral Meshes. Springer Briefs in Mathematics, Springer, Cham (2017)
11. Gopalakrishnan, J., Kanschat, G.: A multilevel discontinuous Galerkin method. Numer. Math. **95**(3), 527–550 (2003). https://doi.org/10.1007/s002110200392
12. Ayuso, B., Marini, L.D.: Discontinuous Galerkin methods for advection–diffusion–reaction problems. SIAM J. Numer. Anal. **47**(2), 1391–1420 (2009). https://doi.org/10.1137/080719583

13. Kanschat, G.: Discontinuous Galerkin Methods for Viscous Incompressible Flow, 1st Edn, Advances in Numerical Mathematics. Teubner Research, Dt. Univ.-Verl, Wiesbaden (2007). http://d-nb.info/985773979
14. Frerichs, D., John, V.: On reducing spurious oscillations in discontinuous Galerkin (DG) methods for steady-state convection–diffusion equations. J. Comput. Appl. Math. **393**, 113487 (2021). https://doi.org/10.1016/j.cam.2021.113487
15. Frerichs-Mihov, D., John, V.: On a technique for reducing spurious oscillations in DG solutions of convection–diffusion equations. Appl. Math. Lett. **129**, 107969 (2022). https://doi.org/10.1016/j.aml.2022.107969
16. Dolejší, V., Feistauer, M., Schwab, C.: On discontinuous Galerkin methods for nonlinear convection–diffusion problems and compressible flow. In: Proceedings of EQUADIFF 10, Vol. 127, pp. 163–179. Prague (2002). https://doi.org/10.21136/MB.2002.134171
17. Dolejší, V., Feistauer, M., Schwab, C.: On some aspects of the discontinuous Galerkin finite element method for conservation laws. Math. Comput. Simul. **61**(3–6), 333–346 (2003). https://doi.org/10.1016/S0378-4754(02)00087-3
18. Cockburn, B., Shu, C.-W.: The Runge–Kutta discontinuous Galerkin method for conservation laws V: multidimensional systems. J. Comput. Phys. **141**(2), 199–224 (1998). https://doi.org/10.1006/jcph.1998.5892
19. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8
20. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274
21. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
22. Rojas, R.: Networks of width one are universal classifiers. In: Proceedings of the International Joint Conference on Neural Networks, Vol. 4, 2003, pp. 3124–3127. https://doi.org/10.1109/IJCNN.2003.1224071
23. Ray, D., Hesthaven, J.S.: An artificial neural network as a troubled-cell indicator. J. Comput. Phys. **367**, 166–191 (2018). https://doi.org/10.1016/j.jcp.2018.04.029
24. Ray, D., Hesthaven, J.S.: Detecting troubled-cells on two-dimensional unstructured grids using a neural network. J. Comput. Phys. **397**, 108845 (2019). https://doi.org/10.1016/j.jcp.2019.07.043
25. Liu, Y., Lu, Y., Wang, Y., Sun, D., Deng, L., Wang, F., Lei, Y.: A CNN-based shock detection method in flow visualization. Comput. Fluids **184**, 1–9 (2019). https://doi.org/10.1016/j.compfluid.2019.03.022
26. Veiga, M.H., Abgrall, R.: Towards a general stabilisation method for conservation laws using a multilayer perceptron neural network: 1d scalar and system of equations. In: European Conference on Computational Mechanics and VII European Conference on Computational Fluid Dynamics, no. 1, ECCM, 2018, pp. 2525–2550. https://doi.org/10.5167/uzh-168538
27. Abgrall, R., Han Veiga, M.: Neural Network-Based Limiter with Transfer Learning, Communications on Applied Mathematics and Computation (2020). https://doi.org/10.1007/s42967-020-00087-1
28. Morgan, N.R., Tokareva, S., Liu, X., Morgan, A.: A machine learning approach for detecting shocks with high-order hydrodynamic methods. In: AIAA Scitech 2020 Forum (2020). https://doi.org/10.2514/6.2020-2024
29. Beck, A.D., Zeifang, J., Schwarz, A., Flad, D.G.: A neural network based shock detection and localization approach for discontinuous Galerkin methods. J. Comput. Phys. **423**, 109–824 (2020). https://doi.org/10.1016/j.jcp.2020.109824
30. Joshi, S.M., Anandh, T., Teja, B., Ganesan, S.: On the choice of hyper-parameters of artificial neural networks for stabilized finite element schemes. Int. J. Adv. Eng. Sci. Appl. Math. **13**, 278–297 (2020). https://doi.org/10.1007/s12572-021-00306-9
31. Margenberg, N., Lessig, C., Richter, T.: Structure preservation for the deep neural network multigrid solver. Electron. Trans. Numer. Anal. **56**, 86–101 (2021). https://doi.org/10.1553/etna_vol56s86
32. von Wahl, H., Richter, T.: Using a deep neural network to predict the motion of underresolved triangular rigid bodies in an incompressible flow. Int. J. Numer. Methods Fluids **93**(12), 3364–3383 (2021). https://doi.org/10.1002/fld.5037
33. Montalvão Silva, R., Coutinho, A.: PINNs for parametric incompressible newtonian flows. In: Proceedings of the XLII Ibero-Latin-American Congress on Computational Methods in Engineering and III Pan-American Congress on Computational Mechanics, ABMEC-IACM (2021). https://cilamce.com.br/anais/arearestrita/apresentacoes/252/9345.pdf
34. Beck, A., Flad, D., Munz, C.-D.: Deep neural networks for data-driven les closure models. J. Comput. Phys. **398**, 108910 (2019). https://doi.org/10.1016/j.jcp.2019.108910

35. Barrenechea, G.R., John, V., Knobloch, P.: Finite element methods respecting the discrete maximum principle for convection–diffusion equations, Tech. rep., arXiv, accepted for publication in SIAM Review (2023). https://doi.org/10.48550/ARXIV.2204.07480
36. Hughes, T.J.R., Mallet, M., Mizukami, A.: A new finite element formulation for computational fluid dynamics. II. Beyond SUPG. Comput. Methods Appl. Mech. Eng. **54**(3), 341–355 (1986)
37. Hemker, P.W.: A singularly perturbed model problem for numerical computation. J. Comput. Appl. Math. **76**(1–2), 277–285 (1996)
38. John, V., Knobloch, P., Savescu, S.B.: A posteriori optimization of parameters in stabilized methods for convection-diffusion problems—part I. Comput. Methods Appl. Mech. Eng. **200**(41–44), 2916–2929 (2011). https://doi.org/10.1016/j.cma.2011.04.016
39. John, V., Knobloch, P., Wilbrandt, U.: A posteriori optimization of parameters in stabilized methods for convection–diffusion problems—part II. J. Comput. Appl. Math. **428**, Article 115167 (2023)
40. Frerichs-Mihov, D., Wilbrandt, U., Henning, L., John, V.: Data and code for using deep neural networks for detecting spurious oscillations in discontinuous Galerkin solutions of convection-dominated convection–diffusion equations, this work is licensed under CC BY 4.0 (2022). https://doi.org/10.20347/40vd-f944
41. Ciarlet, P.G.: The Finite Element Method for Elliptic Problems, Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia (2002). https://doi.org/10.1137/1.9780898719208
42. Higham, C.F., Higham, D.J.: Deep learning: an introduction for applied mathematicians. SIAM Rev. **61**(4), 860–891 (2019). https://doi.org/10.1137/18M1165748
43. Kingma, D.P., Ba, J.L.: Adam: a method for stochastic optimization. In: ICLR 2015, arXiv, p. 13 (2014). https://doi.org/10.48550/ARXIV.1412.6980
44. Barrenechea, G.R., John, V., Knobloch, P., Rankin, R.: A unified analysis of algebraic flux correction schemes for convection–diffusion equations. SeMA J. **75**(4), 655–685 (2018). https://doi.org/10.1007/s40324-018-0160-6
45. Ganesan, S., John, V., Matthies, G., Meesala, R., Abdus, S., Wilbrandt, U.: An object oriented parallel finite element scheme for computing PDEs: design and implementation. In: IEEE 23rd International Conference on High Performance Computing Workshops (HiPCW) Hyderabad, pp. 106–115. IEEE (2016)
46. Wilbrandt, U., Bartsch, C., Ahmed, N., Alia, N., Anker, F., Blank, L., Caiazzo, A., Ganesan, S., Giere, S., Matthies, G., Meesala, R., Shamim, A., Venkatesan, J., John, V.: ParMooN—a modernized program package based on mapped finite elements. Comput. Math. Appl. **74**(1), 74–88 (2017). https://doi.org/10.1016/j.camwa.2016.12.020
47. Davis, T.A.: Algorithm 832: UMFPACK V4.3—an unsymmetric-pattern multifrontal method. ACM Trans. Math. Softw. **30**(2), 196–199 (2004)
48. Kubat, M.: An Introduction to Machine Learning, 3rd edn. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81935-4
49. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M. ,Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, software available from https://www.tensorflow.org/ (2015)
50. Developers, T.: TensorFlow, v2.9.1. https://doi.org/10.5281/zenodo.6574233 (2022)
51. Izquierdo, S.: CppFlow, v2.0.0. https://github.com/serizba/cppflow, https://serizba.github.io/cppflow/ (2022)
52. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Vol. 9 of Proceedings of Machine Learning Research, PMLR, Chia Laguna Resort, Sardinia, Italy, pp. 249–256 (2010). https://proceedings.mlr.press/v9/glorot10a.html