Graph-instructed neural networks for sparse grid-based discontinuity detectors

Applied Mathematics and Computation 519 (2026) 129946

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Full Length Article

Graph-instructed neural networks for sparse grid-based discontinuity detectors

Francesco Della Santa [a,b,*], Sandra Pieraccini [a,b]

[a] Department of Mathematical Sciences, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129, Turin, Italy
[b] Gruppo Nazionale per il Calcolo Scientifico INdAM, Piazzale Aldo Moro 5, 00185, Rome, Italy

ARTICLE INFO

2020 MSC:
68T07
03D32
65D40

Keywords:
Discontinuous functions
Sparse grids
Deep learning
Graph neural networks
Discontinuity interface detection

ABSTRACT

In this paper, we present a novel approach for detecting the discontinuity interfaces of a discontinuous function. This approach leverages Graph-Instructed Neural Networks (GINNs) and sparse grids to address discontinuity detection even in domains of dimension larger than 3. GINNs, trained to identify troubled points on sparse grids, exploit graph structures built on the grids to achieve efficient and accurate discontinuity detection performance. We also introduce a recursive algorithm for general sparse grid-based detectors, characterized by convergence properties and ease of applicability. Numerical experiments on functions with dimensions $n = 2$ and $n = 4$ demonstrate the efficiency and robust generalization properties of GINNs in detecting discontinuity interfaces; test cases with $n = 6$ and $n = 8$ show the applicability of the method when the discontinuity interface presents specific structures. Notably, the trained GINNs offer portability and versatility, allowing integration into various algorithms and sharing among users.

1. Introduction

Detecting discontinuity interfaces of discontinuous functions is a challenging task with significant implications across various scientific and engineering applications. Identifying these interfaces is particularly critical for functions with a high-dimensional domain, as their discontinuities can significantly influence the behavior of numerical methods and simulations; for example, within the realm of uncertainty quantification, where the smoothness of the target function plays a fundamental role in the use of stochastic collocation methods. Specifically, the knowledge of discontinuity interfaces enables the partitioning of the function domain into regions of smoothness, a crucial factor in improving the performance of numerical methods (e.g., see Ref. [1]). Other examples of discontinuity detection applications include signal processing, investigations of phase transitions in physical systems (Ref. [2]), and change-point analyses in geology or biology (Ref. [3]), to name a few.

The central objective of most discontinuity detection methods is to identify the position of discontinuities in the function domain using function evaluations on sets of points. Over the last few decades, progress has been made in discontinuity detection, leading to the development of various algorithms. Notable works, such as Refs. [4–6], have introduced significant contributions in this field. In particular, Ref. [4] introduced a polynomial annihilation edge detection method designed for piecewise-smooth functions with low-dimensional domains ($n \leq 2$). This method identifies discontinuous interfaces by reconstructing jump functions based on a set of function evaluations. Starting from these results, Archibald et al. in [5] extended the approach to higher dimensions, applying the detection method for each input dimension within a generalized polynomial chaos approximation of the target function. However, in

* Corresponding author.
E-mail address: francesco.dellasanta@polito.it (F. Della Santa).

https://doi.org/10.1016/j.amc.2025.129946
Received 17 March 2025; Received in revised form 31 October 2025; Accepted 31 December 2025
Available online 10 January 2026

Ref. [5] the curse of dimensionality restricts considerably the practical applicability of the method. A significant advance in addressing the curse of dimensionality was made by Jakeman et al. in Ref. [6], where sparse grids (see Refs. [7–9]) were utilized to create an adaptive method. This approach increased the method potential for use in higher dimensions.

Besides Ref. [6], several studies have explored adaptive sparse grid methods for detecting discontinuity interfaces. These methods typically employ spatial adaptivity, which focuses on refining the grid in regions where discontinuities are detected, and/or dimensional adaptivity, which aims to reduce refinement along directions that are largely inactive with respect to the discontinuity. To cite a few examples, Ref. [10] addresses the use of sparse grids for solving stochastic differential equations, proposing an adaptive sparse grid collocation method. Hierarchical surplus is employed as an error indicator to automatically identify discontinuity regions in the stochastic space, and an adaptive refinement of the sparse grids is performed in the vicinity of these critical regions. Similarly, Ref. [11] considers discontinuous test functions representing decision functions in classification tasks within the framework of data mining problems. Spatially adaptive sparse grids, obtained with a surplus-based refinement strategy, are proposed as an efficient tool for handling high-dimensional problems. Ref. [12], however, points out that the local mesh refinement inevitably introduced near discontinuity regions causes a notable loss of grid sparsity. To address this limitation, the authors introduce an alternative approach based on approximating hypersurfaces representing the discontinuity interfaces using hyper-spherical coordinates. This method is suitable for large domain dimensions but was primarily designed for detecting a single interface under star-convexity assumptions.

Other techniques addressing the discontinuity detection problem include wavelet-based methods (see Refs. [13,14]), filter-based algorithms (Refs. [15,16]), and troubled cell detection (Refs. [3,17–19]). In the recent work [20], the authors present a method using the so-called null rules, applied and tested on bivariate functions; in particular, this method analyzes the rules asymptotic properties, introducing two indicators for function and gradient discontinuities for the detection of points near discontinuities. Moreover, the method presented in Ref. [20] is integrated with adaptive approximation techniques based on hierarchical spline spaces for reconstructing surfaces with discontinuities.

In recent years, Neural Networks (NNs) have also been successfully applied to discontinuity detection. For example, in Ref. [21] a novel approach emerged, aimed at constructing discontinuous NNs capable of approximating discontinuous functions incorporating trainable discontinuity jump parameters, and enabling the model to simultaneously learn the target function and its discontinuity interface. This method offers several advantages, with few restrictions on its applicability; however, the main limitation is the curse of dimensionality related to the training set cardinality, which can be very large for functions in high-dimensional domains.

Another interesting application of NNs in this context is reported in Ref. [3], which introduces a discontinuity detection method based on specifically tailored Convolutional Neural Network (CNN) models. This study has been motivated by the success of CNNs in the context of image edge detection (see, e.g., Refs. [22–26]); indeed, the problem of edge detection in computer vision is essentially a two-dimensional discontinuity detection problem, with observed data taking the form of intensity values of image pixels. The CNN-based method developed in Ref. [3] consists of a two-level detection procedure. The procedure involves a coarse-to-fine process with two detectors: the first is aimed at preliminary detection over coarsened grids for rapid identification of coarse-troubled cells, and the second is focused on refining and providing detailed detection within the detected coarse-troubled cells, primarily on fine grids. In general, Ref. [3] introduces a cheap and very efficient method for discontinuity detection with respect to functions with two- or three-dimensional domains and, in principle, works also in higher dimensions. Nonetheless, the curse of dimensionality limits its applicability in the latter case, because the method requires regular grids for function evaluations and the implementation of n-dimensional convolutional layers with $n > 3$ (typically, existing Deep Learning frameworks address dimensions $n \leq 3$, such as Ref. [27]).

We believe that Ref. [3] describes a very promising direction; in this work, we modify and extend the methodology of Ref. [3], moving from regular grids (and CNNs) to sparse grids (and NNs based on graphs), in order to tackle the problem in dimensions higher than 3. In particular, we introduce a novel approach that utilizes Graph-Instructed Neural Networks (GINNs), see Refs. [28,29], trained for detecting the so-called troubled points in a sparse grid used for the function evaluations; in particular, each GINN is based on the adjacency matrix of a graph built on the structure of the chosen sparse grid. In addition, partially inspired by Ref. [6], we define a recursive algorithm for general sparse grid-based detectors, characterized by finite termination and convergence properties. Then, running this algorithm using as detector a GINN well-trained in dimension n, we obtain an effective discontinuity detection method suitable for almost any function with n-dimensional domain. Actually, also Multi-Layer Perceptrons (MLPs) can be trained as sparse grid-based detectors but, according to the results in Ref. [28], the experiments show that GINNs take advantage of the graph structure built on the sparse grid, returning better results.

Concerning the experiments, we test our method on a first set of functions with domain of dimension $n = 2$ and $n = 4$, in order to asses its effectiveness; then, we test our method and a dimensionally adaptive variant of it also for dimensions $n = 6$ and $n = 8$. The results obtained are promising and show very good generalization abilities of the GINNs in detecting discontinuity interfaces even for functions different from the ones used in the training sets. Furthermore, the method is extremely flexible and does not rely on specific assumptions about the interfaces to be detected (such as smoothness, mutual non-intersection, or location in specific positions). As a result, the types, the number, and the reciprocal positioning of discontinuity interfaces that can be detected are fairly general. Finally, we point the attention of the reader to the fact that the trained GINNs for discontinuity detection have the advantage of being portable and versatile; i.e., they can be shared among different users and integrated into new algorithms. For this reason, we make available the trained GINNs used in all the experiments for public use and research (see Remark 5.2).

Despite the use of sparse grids for delaying the curse of dimensionality, the creation of synthetic datasets based on deterministic detectors remains computationally expensive, especially in higher dimensions. While we have partially addressed this computational cost by parallelizing the dataset creation process, we remark that this cost is incurred only once; once the model has been trained on a synthetic dataset, it can be applied to any problem with the same dimensionality. Another important bottleneck, particularly in

comparison with adaptive sparse grid approaches, is that the method still requires exploring a large number of points, depending on the complexity of the discontinuity interface, which becomes increasingly problematic as the dimensionality grows. This issue can be partially mitigated by employing dimensional adaptivity when discontinuities are not uniformly distributed across all dimensions. Some preliminary results are shown for $n = 6$ and $n = 8$, but further developments are needed to extend the approach to even higher-dimensional problems.

The work is organized as follows. In Section 2, we recall briefly how to build sparse grids based on equispaced collocation knots and we introduce the concept of sparse grid graph. In Section 3, we define discontinuity detectors and introduce algorithms for identifying the discontinuity interfaces of a discontinuous function. Then, in Section 4, we describe the procedure for building and training an NN-based discontinuity detector (both MLP and GINN); further details are illustrated in Appendix A, at the end of the paper. In Section 5, we show the results obtained with the NN-based discontinuity detectors for two-dimensional functions (including images, for edge-detection applications), four-dimensional functions (including a real-world stochastic setting), and a study for higher-dimensional cases for proving the efficiency and the promising potentialities of the new method, especially if based on GINNs. We end with some conclusions drawn in Section 6.

2. Problem settings and preliminaries

In this work, we consider the discontinuity detection problem for a discontinuous function $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, where the task is performed exploiting sparse grids in order to delay the curse of dimensionality. In this Section, we briefly recall the basic notions about sparse grids, focusing on the case of nested and equispaced sets of knots; then, we introduce new definitions, mainly focused on building graphs with vertices corresponding to the points of a sparse grid. Indeed, the algorithm we introduce later in this work for discontinuity detection (see Section 3.1) is based on such graphs.

For the sake of simplicity, let us consider in the following $\Omega = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n]$. This is not a restricting assumption as any function $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ can be extended, continuously or not, to a hyperrectangular domain $R = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n]$; e.g., setting $g(\boldsymbol{x}) \equiv 0$ for each $\boldsymbol{x} \in R \setminus \Omega$.

2.1. Sparse grids

Sparse grids were originally introduced by Smolyak in Ref. [7] for high-dimensional quadrature and since then also used for polynomial interpolation and stochastic collocation (e.g., see Refs. [30–32]). While referring the reader to Ref. [8] for a comprehensive work on sparse grids, we sketch here the main ideas.

A sparse grid $\mathcal{S}$ in $\mathbb{R}^n$ is obtained by suitably collecting n-dimensional tensor grids, each one obtained as the Cartesian product of univariate sets of knots hierarchically built through a number of levels. More precisely, a sparse grid is built with the following steps.

1. Upon selecting a univariate knot distribution, we consider, on each interval $[\alpha_i, \beta_i]$, $i = 1, \dots, n$, a sequence $\{\mathcal{K}_{i,h}\}_{h \geq 1}$ of grids with nodes taken from the chosen distribution; the index h represents the level of $\mathcal{K}_{i,h}$, and we let $m(h)$ denote the number of nodes of $\mathcal{K}_{i,h}$ (which is independent of the dimension i); we start with $m(1) = 1$ and we further assume that $m(h) \leq m(h+1)$.
2. Chosen a set $\mathcal{I} \subset \mathbb{N}_+^n$ of multi-indices, for each multi-index $\boldsymbol{h} = (h_1, \dots, h_n) \in \mathcal{I}$, we introduce the n-dimensional grid $\mathcal{S}_{\boldsymbol{h}}$ as the tensor product
$$\mathcal{S}_{\boldsymbol{h}} = \bigotimes_{i=1}^{n} \mathcal{K}_{i,h_i}.$$
Standard constructions are proposed in the literature, corresponding to suitable choices of the multi-index set $\mathcal{I}$, see later Eq. (2).
3. The sparse grid $\mathcal{S}$ is the union of all the tensor grids generated at the previous step:
$$\mathcal{S} := \bigcup_{\boldsymbol{h} \in \mathcal{I}} \mathcal{S}_{\boldsymbol{h}}.$$

Several choices are possible for the univariate knot distribution, for the number of nodes $m(h)$ to be introduced on each interval at a given level h, and for the set of multi-indices used to build $\mathcal{S}$; we refer the reader to Ref. [9] for a MATLAB library for the generation of sparse grids.

Herein, we focus on equispaced nodes, essentially doubling the number of nodes at each new level, namely

$$m(h) := \begin{cases} 1, & \text{if } h = 1, \\ 2^{h-1} + 1, & \text{otherwise.} \end{cases} \tag{1}$$

It is worth noting that with this choice, upon increasing the level h, nested set of nodes are obtained.

As far as the choice of the multi-index set is concerned, for a given level $\hat{h} \in \mathbb{N}_+$, the set $\mathcal{I} := \mathcal{I}(\hat{h})$ introduced at step 2 can be described as

$$\mathcal{I}(\hat{h}) = \{\boldsymbol{h} \in \mathbb{N}_+^n \mid r(\boldsymbol{h}) \leq \hat{h}\},$$

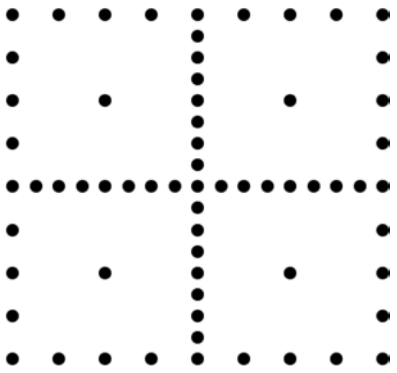


Fig. 1. Example of sparse grid in $\mathbb{R}^2$, function m as in (1), equispaced knots, and multi-indices set $\mathcal{I}_{\text{sum}}(4)$. .

Table 1
Number of nodes of sparse grids in $\mathbb{R}^n$ built upon equispaced univariate knots, function m as in (1), multi-index set $\mathcal{I}_{\text{sum}}(\hat{h})$, for several values of n and $\hat{h}$.

$\hat{h}$	$n=2$	$n=4$	$n=6$	$n=8$	$n=10$
1	5	9	13	17	21
2	13	41	85	145	221
3	29	137	389	849	1 581
4	65	401	1 457	3 937	8 801
5	145	1 105	4 865	15 713	41 265

where $r : \mathbb{N}_+^n \to \mathbb{N}_+$ is suitably chosen. Some examples (see, e.g., Refs. [8,9,33]) are:

$$r_{\text{prod}}(\boldsymbol{h}) = \prod_{i=1}^{n} h_i, \quad r_{\text{sum}}(\boldsymbol{h}) = \sum_{i=1}^{n} (h_i - 1), \quad r_{\max}(\boldsymbol{h}) = \max_{i=1,\dots,n} (h_i - 1)\,. \tag{2}$$

Choice r_{sum} is known as the Total Degree rule, in the framework of polynomial approximation; choice $r_{\max}$ corresponds to a full tensor grid. The multi-index generated by choices in (2) will be denoted by $\mathcal{I}_{\text{prod}}(\hat{h})$, $\mathcal{I}_{\text{sum}}(\hat{h})$, and $\mathcal{I}_{\max}(\hat{h})$, respectively.

In Fig. 1 we show a sparse grid example, built on the square $[\alpha_1, \beta_1] \times [\alpha_2, \beta_2] = [-1, 1] \times [-1, 1]$, using univariate equispaced nodes, function m as in (1), and the multi-index set $\mathcal{I}_{\text{sum}}(4)$. From now on, for simplicity, we will denote as *equispaced sparse grid* a sparse grid characterized by an univariate equispaced nodes distribution and function m as in (1).

We end this introduction by listing in Table 1 the number of nodes of a sparse grid in $\mathbb{R}^n$ built upon equispaced univariate knots, function m as in (1), multi-index set $\mathcal{I}_{\text{sum}}(\hat{h})$, for several values of n and $\hat{h}$, remarking that a full tensor grid corresponding to level $\hat{h}$ and dimension n, would have $(2^{\hat{h}} + 1)^n$ knots.

2.1.1. Sparse grid box and sparse grid similarity

In view of the description of the discontinuity detection algorithm presented in Section 3, we introduce the following definitions related to sparse grids.

Definition 2.1 (Sparse grid box). *Let $\mathcal{S}$ be a sparse grid in $\mathbb{R}^n$. Then, the hyperrectangle*

$$\Omega = [\alpha_1, \beta_1] \times \cdots \times [\alpha_n, \beta_n] \tag{3}$$

upon which $\mathcal{S}$ is built, is called box *of $\mathcal{S}$ and denoted as $\mathbb{B}(\mathcal{S})$*

Definition 2.2 (Sparse grid similarity). *Let $\mathcal{S}'$ and $\mathcal{S}''$ be two sparse grids in $\mathbb{R}^n$, given by the set of points $\mathcal{S}' = \{\boldsymbol{x}'_1, \dots, \boldsymbol{x}'_N\}$ and $\mathcal{S}'' = \{\boldsymbol{x}''_1, \dots, \boldsymbol{x}''_N\}$, respectively. Then, $\mathcal{S}'$ and $\mathcal{S}''$ are similar if and only if there exist $a \in \mathbb{R}$, $a \neq 0$, and $\boldsymbol{b} \in \mathbb{R}^n$ such that*

$$\boldsymbol{x}'_i = a\,\boldsymbol{x}''_i + \boldsymbol{b}, \tag{4}$$

for all $i = 1, \dots, N$. The similarity between two sparse grids is denoted by $\mathcal{S}' \simeq \mathcal{S}''$.

2.2. Sparse grid graphs

The position of the points of a sparse grid is naturally inclined to define a graph. Nonetheless, to the best of the authors' knowledge, there are no formal procedures in the literature to build a graph from a sparse grid. Therefore, we introduce here a formal definition

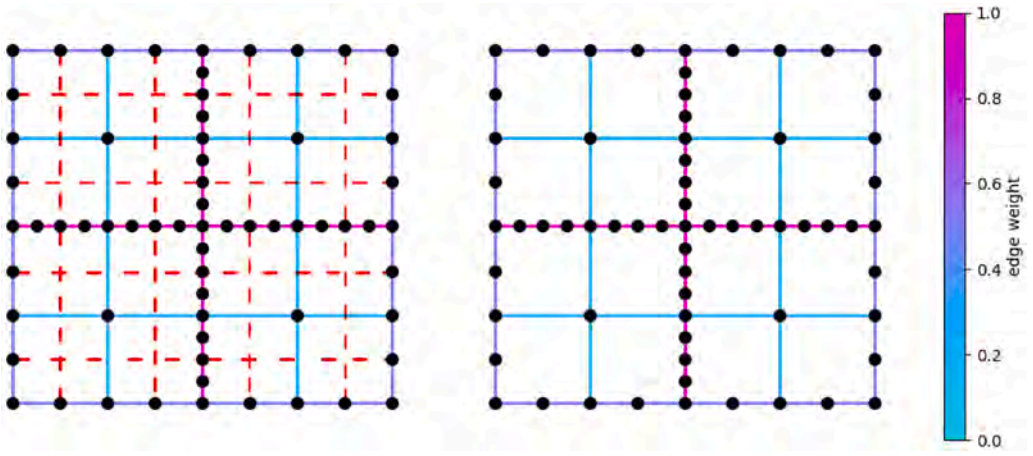


Fig. 2. Sparse grid graph based on the sparse grid S in Fig. 1. *Left.* Raw graph obtained connecting the aligned and consecutive points of the sparse grid. *Right.* Final graph obtained removing the edges that intersect an edge of equal or smaller length. The edge color depends on the edge weight (see Definition 2.4).

for a graph associated to a sparse grid in $\mathbb{R}^n$. The graph representations of sparse grids are aimed at exploiting the connections between the grid points in the framework of a discontinuity detection task (see later Section 3).

Given two points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$, we let $\overline{(\boldsymbol{x}_1, \boldsymbol{x}_2)}$ denote the line segment connecting $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. We define a graph associated to S according to the following definition.

Definition 2.3 (Sparse grid graph). *Let $S \subset \mathbb{R}^n$ be a sparse grid made of N points. We define* sparse grid graph *(SGG) based on S the undirected graph $G = (S, E)$ such that, for each $\boldsymbol{x}_i, \boldsymbol{x}_j \in S$, $\boldsymbol{x}_i \neq \boldsymbol{x}_j$, the set $\{\boldsymbol{x}_i, \boldsymbol{x}_j\}$ is an edge of G if and only if:*

1. *$\boldsymbol{x}_i, \boldsymbol{x}_j$ are aligned along one of the axes in the $\mathbb{R}^n$ space;*
2. *there is no $\boldsymbol{x} \in S$, $\boldsymbol{x} \neq \boldsymbol{x}_i, \boldsymbol{x}_j$, belonging to the segment $\overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)}$;*
3. *it holds*

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\| < \|\boldsymbol{x}_p - \boldsymbol{x}_q\|,$$

for each $\boldsymbol{x}_p, \boldsymbol{x}_q \in S$ such that $\overline{(\boldsymbol{x}_p, \boldsymbol{x}_q)}$ is perpendicular to $\overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)}$ and $\overline{(\boldsymbol{x}_p, \boldsymbol{x}_q)} \cap \overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)} \neq \emptyset$.

In a nutshell, we connect two points of the sparse grid aligned along one of the axes if no other grid points exist between them (item 2, see Fig. 2-left); then, we remove all the edges corresponding to line segments $\overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)}$ that intersect other line segments $\overline{(\boldsymbol{x}_p, \boldsymbol{x}_p)}$ whose length is smaller than or equal to the one of $\overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)}$ (item 3, see Fig. 2-right).

From now on, overloading notation, we identify the line segment $\overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)}$ with the corresponding edge $\{\boldsymbol{x}_i, \boldsymbol{x}_j\}$ in the SGG. Then, the edges of a SGG correspond to line segments in $\mathbb{R}^n$ with possibly different lengths. Therefore, it is natural to exploit the length of line segments to assign weights to edges.

Here, we use the strategy of setting a weight inversely proportional to the distance in $\mathbb{R}^n$ between the points defining the edges. A weighted version of SGGs is useful for improving the training of discontinuity detectors based on Graph-Instructed NNs.

Definition 2.4 (Weighted sparse grid graph). *Let $G = (S, E)$ be a sparse grid graph based on a sparse grid S in $\mathbb{R}^n$; let $\ell \in \mathbb{R}$ be the length of the shortest segment among the ones defined by edges of G, i.e.:*

$$\ell := \min_{\{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E} \|\boldsymbol{x}_i - \boldsymbol{x}_j\| . \tag{5}$$

Then, G is a weighted sparse grid graph *(WSGG) based on S if the edge weights are defined by*

$$\omega_{ij} := \frac{\ell}{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|} \in (0, 1], \tag{6}$$

for each edge $\{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E$.

We remark that the edge weights introduced in (6) are relative quantities, depending on the relative distance of the points in the sparse grid with respect to ℓ, and ℓ is related to the grid level. Therefore, the WSGGs of two similar sparse grids only differ for the set of vertices. This property is summarized in the following proposition.

Proposition 2.1. *Let S' and S'' be two similar sparse grids in $\mathbb{R}^n$ and let $G' = (S', E'), G'' = (S'', E'')$ be the corresponding weighted sparse grid graphs. Then, G' and G'' have the same adjacency matrix.*

Proof. The proof is almost straightforward. Let $A' = (A'_{ij}), A'' = (A''_{ij}) \in \mathbb{R}^{N\times N}$ be the adjacency matrices of G', G'', respectively. Since $\mathcal{S}' \simeq \mathcal{S}''$, A' and A'' have the same sparsity pattern, i.e.:

$$A'_{ij} \neq 0 \iff A''_{ij} \neq 0,\ \forall\, i,j \in \{1,\dots,N\}\,.$$

Moreover, for each $i,j \in \{1,\dots,N\}$ such that $\{\boldsymbol{x}'_i, \boldsymbol{x}'_j\} \in E'$ and $\{\boldsymbol{x}''_i, \boldsymbol{x}''_j\} \in E''$, from Eq. (4) we have that

$$\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\| = \|a\,\boldsymbol{x}''_i + \boldsymbol{b} - a\,\boldsymbol{x}''_j - \boldsymbol{b}\| = |a|\|\boldsymbol{x}''_i - \boldsymbol{x}''_j\|\,;$$

therefore, the corresponding non-zero elements of the adjacency matrices are equal, because

$$A'_{ij} = \omega'_{ij} = \frac{\ell'}{\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|} = \frac{|a|\,\ell''}{|a|\|\boldsymbol{x}''_i - \boldsymbol{x}''_j\|} = \omega''_{ij} = A''_{ij}\,. \tag{7}$$

□

Definition 2.5 (Sparse Grid Graph Similarity). *Let G', G'' be two sparse grid graphs (weighted or not) based on two sparse grids $\mathcal{S}', \mathcal{S}''$, respectively. Then, G' and G'' are similar ($G' \simeq G''$) if $\mathcal{S}' \simeq \mathcal{S}''$.*

3. Discontinuity detection and sparse grids

The discontinuity detection method proposed herein aims to extend the one proposed in Ref. [3], transitioning from uniform to sparse grids. In view of this extension, we seek an NN architecture leveraging sparse grids; thus, we consider the use of Graph-Instructed Neural Networks (GINNs) built with respect to a SGG. We remark that the method proposed herein is designed in such a way that also standard NNs, such as Multi-Layer Perceptrons (MLPs), can be used; however, the numerical experiments show that the method benefits from the use of GINN (see Section 5); this is in line with the results presented in Ref. [28].

In this section, we describe the discontinuity detection procedure, assuming that a so-called detector is available. In Section 4, we provide details on how the procedure can be applied using a surrogate detector built upon GINNs.

For a given function $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, the method proposed relies on the following ingredients:

- the concept of *troubled points* of a sparse grid (i.e, points of the grid that are near a discontinuity interface of g);
- the availability of a so-called *discontinuity detector*, i.e., a function or algorithm that returns the troubled points.

The algorithm defined in this work does not depend on the nature of the discontinuity detector used, and the function g can be rather general; however, the algorithm works better with discontinuous functions characterized by discontinuity interfaces with low codimension (with respect to $\mathbb{R}^n \supseteq \Omega$), see also Remark 3.1. In the following, we formally introduce the notions of troubled point and discontinuity detector based on a sparse grid. The definition of troubled points is inspired by the definition of troubled cells given in Ref. [3].

Definition 3.1 (Troubled points). *Let $\mathcal{S}$ be a sparse grid in $\mathbb{R}^n$ and let $G = (S, E)$ be the (possibly weighted) graph based on $\mathcal{S}$. Let $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ be a given function such that $\mathbb{B}(S) \subseteq \Omega$ and let δ be the set of discontinuity points of g. Then, the node $\boldsymbol{x}_i$ of $\mathcal{S}$ is said to be a troubled point for g if there exist $e_{ij} = \{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E$ such that:*

- $\delta \cap \overline{(\boldsymbol{x}_i, \boldsymbol{x}_j)} \neq \emptyset$;
- *it holds*

$$\|\boldsymbol{x}_i - \boldsymbol{x}_i^\delta\| \leq \|\boldsymbol{x}_i^\delta - \boldsymbol{x}_j\|, \tag{8}$$

where $\boldsymbol{x}_i^\delta$ is the point in $\delta \cap e_{ij}$ nearest to $\boldsymbol{x}_i$.

In other words, $\boldsymbol{x}_i \in \mathcal{S}$ is a troubled point if δ intersects at least one edge $\{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E$ in a point between $\boldsymbol{x}_i$ and the mid-point of the line segment corresponding to the edge.

Definition 3.2 (Discontinuity detector). *Let $\mathcal{S}$ be a sparse grid in $\mathbb{R}^n$ made of N points and let $\mathcal{S}_{/\simeq}$ denote the set of all the sparse grids similar to $\mathcal{S}$. For a fixed $\Omega \subseteq \mathbb{R}^n$, let $\mathcal{F}$ denote the set of functions $g : \Omega \to \mathbb{R}$. Then, a* discontinuity detector *based on $\mathcal{S}$ is a function Δ that, for each $\mathcal{S}' \simeq \mathcal{S}$ and each function $g \in \mathcal{F}$, returns a vector $\boldsymbol{p} = (p_1, \dots, p_N)$, with $0 \leq p_i \leq 1$ for all $i = 1,\dots,N$, where p_i gives an indication of how likely $\boldsymbol{x}'_i \in \mathcal{S}'$ is a troubled point. The closer p_i is to 1, the more likely $\boldsymbol{x}'_i$ is a troubled point.*

According to Definition 3.2, a discontinuity detector provides, for each grid point, a sort of likelihood for being a troubled point. A point $\boldsymbol{x}'_i$ is labeled as troubled if $p_i \geq \tau$, where $\tau \in (0, 1]$ is a given threshold.

The previous definition is intentionally qualitative, as several types of discontinuity detectors can be defined. For example, we can define detectors that evaluate $\boldsymbol{p}$ knowing: *i*) the function g and the coordinates of the points in $\mathcal{S}'$; *ii*) the values of g taken at the points in $\mathcal{S}'$, and the points coordinates; *iii*) the values of g taken at the points in $\mathcal{S}'$, only. In this work, we focus on building NN-based detectors of the latter type; i.e., detectors Δ such that $\Delta : (\mathbb{R} \cup \{\infty\})^N \to [0,1]^N$ and

$$\Delta(\boldsymbol{g}') = \Delta\big(g(\boldsymbol{x}'_1), \dots, g(\boldsymbol{x}'_N)\big) = \boldsymbol{p}\,. \tag{9}$$

If $\boldsymbol{x}'_i \notin \Omega$, it is possible to adopt the convention that $g(\boldsymbol{x}'_i) = \infty$ and $p_i = 0$.

Based on the previous definition, we also introduce the concept of *exact* discontinuity detector based on a sparse grid.

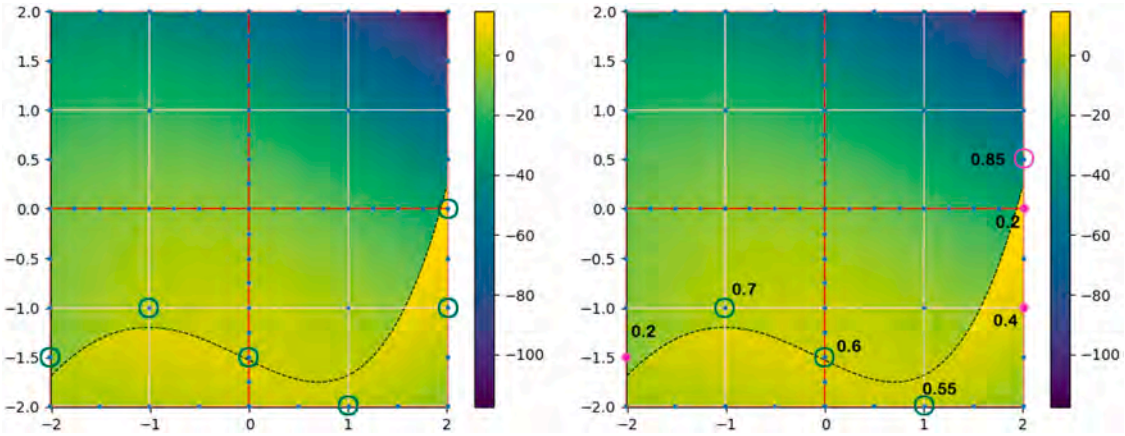


Fig. 3. Example of sparse grid points classification made for a function $g : \mathbb{R}^2 \to \mathbb{R}$ with an exact discontinuity detector Δ^* (left) and an inexact discontinuity detector Δ (right). The black dotted line is the function discontinuity interface. Green circles are true troubled points, magenta circles are false troubled points, magenta dots are false non-troubled points. For the inexact detector, we report also the values p_i corresponding to the highlighted sparse grid points (threshold $\tau = 0.5$).

Definition 3.3 (Exact Discontinuity Detector)**.** *Let S be a sparse grid in $\mathbb{R}^n$ made of N points. An* exact discontinuity detector *based on S is a discontinuity detector Δ^* that for each $S' \simeq S$, and for each given function $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, returns a vector $\boldsymbol{p} \in \{0,1\}^N$ such that, for $i = 1, \dots, N$,*

$$p_i = \begin{cases} 1 & \text{if } \boldsymbol{x}'_i \in S' \text{ is a troubled point for } g \\ 0 & \text{otherwise.} \end{cases}$$

A non-exact discontinuity detector is called an *inexact detector*.

Of course, given a sparse grid, an inexact detector returns a guess for the points to be troubled, whereas the former one detects them exactly. In Fig. 3, the different behaviors of an exact and an inexact discontinuity detector are depicted. In this example, for the inexact detector, a point $\boldsymbol{x}_i$ is labeled as a troubled point if the corresponding value p_i returned by the detector is $p_i \geq \tau = 0.5$. From the example we can see that the inexact detector provides both false-positive troubled points (magenta circles) and false-negative troubled points (magenta dots).

Remark 3.1 (Advantages and disadvantages of exact and inexact detectors)**.** *An exact detector is typically defined through expensive algorithms and in general it is not a detector based on the discontinuous function evaluations only (as in (9)). Therefore, given a sparse grid, our idea consists of building a inexpensive but effective discontinuity detector that is solely based on function evaluations, training an NN model for approximating the exact one, as described in the next Section 4, to be used with a sparse grid-based discontinuity detection algorithm.*

Additionally, a necessary condition for having a troubled point is that at least one edge of the SGG intersects the discontinuity interface δ. The larger the codimension of δ, the lower the probability that δ crosses an edge of the SGG; in the worst-case scenario, δ is a point, and almost always the edges of the SGG will not intersect the point, making the detection of δ almost impossible for an exact discontinuity detector. However, inexact detectors, like the NN-based ones introduced in this work, can identify as troubled point a point that is just close to δ, thus enhancing the possibility to find low dimensional interfaces; see, Section 5.1 for an example (fourth test function).

Remark 3.2 (Detectors independent of point coordinates)**.** *Similarly to what was done in Ref. [3], the introduction of an NN-based detector that is only-evaluations dependent (see (9)) is useful to make the detector universally applicable to any sparse grid $S' \simeq S$ and any discontinuous function with domain dimension equal to n. On the other hand, since this detector is not exact, we can expect that the smaller are the ratios between the function evaluations and the edge lengths (i.e., the slopes $|g(\boldsymbol{x}'_i) - g(\boldsymbol{x}'_j)|/\|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|$), the lower is the reliability of the predicted troubled points.*

3.1. Algorithm for discontinuity detection

In Ref. [3] the discontinuity detection method is characterized by a two-level approach:

1. A first CNN detector is applied on a coarse uniform grid, identifying the coarse troubled cells;
2. A second CNN detector is applied on a fine uniform grid defined on the coarse troubled cells, identifying the fine troubled cells.

We adopt a similar approach for our sparse grid-based method. However, thanks to Proposition 2.1 and Remark 3.2, we can use the same detector Δ for both the coarse step and any refinement step. Therefore, we define an iterative approach that stops when a minimum box size for the sparse grid is reached.

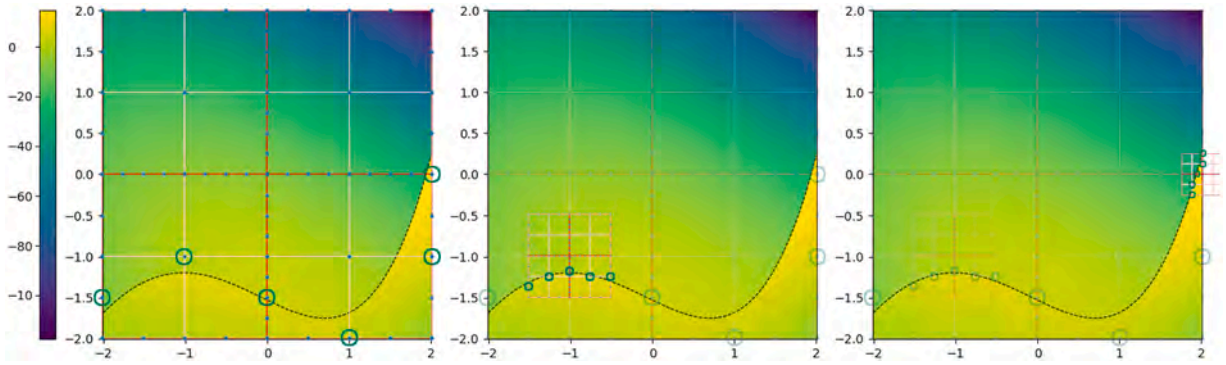


Fig. 4. Example of three steps of Algorithm 1, assuming an exact detector Δ^* and a SGG as in Fig. 2-right. Opaque green circles are the troubled points identified with the current grid. Transparent green circles are the troubled points identified at the previous steps and where the next sparse grids will be centered.

Let us consider a sparse grid S built starting from equispaced knots on a hypercubic box; we recall that, for simplicity, we denote as *equispaced sparse grids* the sparse grids built upon equispaced knots. Given a discontinuity detector Δ based on S, and given a discontinuous function $g : \Omega \to \mathbb{R}$, we sketch the basic version of a sparse grid-based discontinuity detection algorithm as follows (see Fig. 4), while referring for the detailed version to the pseudo-code reported in Algorithm 1:

1. place in Ω one or more initial sparse grids similar to S (equispaced, hypercubic box);
2. using Δ, detect the troubled points of one sparse grid and mark the grid as "checked";

Algorithm 1 Sparse grid-based discontinuity detection (basic).

Require:

$g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, function;
S, sparse grid (equispaced, hypercubic box);
Δ, discontinuity detector based on S;
$L = ((\boldsymbol{x}_1, \lambda_1), \dots, (\boldsymbol{x}_m, \lambda_m))$, sequence of *center* and *edge length* pairs characterizing the sparse grid boxes $\mathbb{B}(S^{(1)}), \dots, \mathbb{B}(S^{(m)}) \subset \Omega$ (typically a queue);
$\Lambda_{\min} \in \mathbb{R}_+$, minimum edge length allowed for sparse grid boxes;
$\tau \in (0, 1]$, detection tolerance.

Ensure:

$T \subset \mathbb{R}^n$, set of points detected as troubled points.

$T \leftarrow \emptyset$
$C \leftarrow \emptyset$ ▷ set of "visited" center-edge pairs
while L is not empty **do**
  $(\boldsymbol{x}, \lambda) \leftarrow$ get element of L ▷ typically the first one, if L is a queue
  $C \leftarrow C \cup \{(\boldsymbol{x}, \lambda)\}$
  $D' \leftarrow [x_1 - \lambda/2, x_1 + \lambda/2] \times \cdots \times [x_n - \lambda/2, x_n + \lambda/2]$
  $G' = (V', E') \leftarrow$ SGG (or WSGG) based on $S' \simeq S$, s.t. $\mathbb{B}(S') = D'$
  $\boldsymbol{p} \leftarrow$ vector returned by Δ, w.r.t. S' and g
  for $i = 1, \dots, N$ **do**
    if $p_i \geq \tau$ **then**
      $\lambda \leftarrow \max_{\{\boldsymbol{x}'_i, \boldsymbol{x}'_j\} \in E'} \|\boldsymbol{x}'_i - \boldsymbol{x}'_j\|$
      if $\lambda \geq \Lambda_{\min}$ **then**
        if $(\boldsymbol{x}'_i, \lambda) \notin C$ **then**
          insert $(\boldsymbol{x}'_i, \lambda)$ in L ▷ typically as last element, if L is a queue
        end if
      else
        $T \leftarrow T \cup \{\boldsymbol{x}'_i\}$
      end if
    end if
  end for
end while

3. for each troubled point $\boldsymbol{x}$, generate a new sparse grid S' similar to S, with box $\mathbb{B}(S')$ centered in $\boldsymbol{x}$ and box edge length equal to the length of the longest graph edge intersecting the troubled point. If this latter length is shorter than a chosen value $\Lambda_{\min} \in \mathbb{R}_+$, do not create the new sparse grid;
4. repeat steps 2 and 3, until all the sparse grids are "checked";
5. the troubled points obtained with the smallest sparse grids are the final troubled points detected by Δ.

Remark 3.3 (Algorithm and non-hypercubic boxes). *In Algorithm 1 the sparse grids are built on hypercubic boxes. This is not a restricting assumption, as the algorithm can be generalized by resorting to sparse grids built on hyperrectangles, changing only the stopping criterion based on the box edge length. For the sake of simplicity we keep the discussion focused on the case of sparse grids built upon hypercubic boxes.*

Remark 3.4 (Equispaced sparse grids for optimized nested structure). *The algorithm exploits equispaced sparse grids for building a nested structure that, in case of an exact discontinuity detector, can guarantee convergence to the discontinuity interfaces of g (see Theorem 1, below); moreover, this structure permits to save computational resources because it increases the probability that some points of a new sparse grid, generated at step 3, are shared with other sparse grids. In order to enhance this characteristic it is important that the boxes of the starting sparse grids are centered properly (i.e., according to the collocation knots criterium), if more than one is given at the first step of the algorithm. Theoretically, the algorithm can be extended also to non-equispaced sparse grids, but the generation of the new sparse grids (step 3) is not straightforward and is postponed to future work.*

Remark 3.5 (Algorithm improvements). *Algorithm 1 represents the backbone of sparse grid-based algorithms for discontinuity detection. Then, depending on the user needs, the algorithm can be changed and/or improved, e.g.:*

- *a stopping criterion based on function evaluations can be introduced (useful for computationally expensive functions g);*
- *the function evaluations at the grid points can be stored and reused;*
- *the troubled points detection, for each box in the list L, can be optimized and/or parallelized.*

We also emphasize that the stopping criterion used here is based on the value of the smallest box size to be considered. As such, the algorithm can detect troubled points with a reciprocal distance as small as $\Lambda_{\min} \cdot 2^{-h_{\max}}$, where $h_{\max}$ is the maximum level of the one-dimensional grids used to construct S. This may result in a large number of visited points and, correspondingly, function evaluations. To balance the two opposing aspects, achieving good coverage of the discontinuity interface while keeping the number of visited (and troubled) points moderate, a different stopping criterion could be considered.

It is worth remarking that any NN-based discontinuity detector is particularly suitable to be used with the algorithm for the troubled points predictions (see the last item in Remark 3.5); indeed, NNs can make predictions for thousands of inputs within seconds, exploiting the inner operations characterizing the NNs. In order to fully exploit the potential of NNs, we propose a modification of Algorithm 1. In particular, given an NN-based detector Δ, we can compute the vectors $\boldsymbol{p}^{(1)}, \dots, \boldsymbol{p}^{(K)} \in \mathbb{R}^N$ corresponding to the points of the sparse grids $S^{(1)}, \dots, S^{(K)}$, respectively, with just one NN batch-evaluation with respect to the vectors $\boldsymbol{g}^{(1)}, \dots, \boldsymbol{g}^{(K)}$; i.e.,

$$P = \Delta(\Gamma), \tag{10}$$

where

$$P := \begin{bmatrix} \boldsymbol{p}^{(1)T} \\ \vdots \\ \boldsymbol{p}^{(K)T} \end{bmatrix} = \left(p_i^{(k)}\right)_{\substack{k=1,\dots,K \\ i=1,\dots,N}} \in \mathbb{R}^{K\times N}, \quad \Gamma := \begin{bmatrix} \boldsymbol{g}^{(1)T} \\ \vdots \\ \boldsymbol{g}^{(K)T} \end{bmatrix} = \left(g(\boldsymbol{x}_i^{(k)})\right)_{\substack{k=1,\dots,K \\ i=1,\dots,N}} \in \mathbb{R}^{K\times N}. \tag{11}$$

Therefore, the efficiency of the algorithm increases, as entire batches of sparse grids can be evaluated at once rather than one at a time (see lines 4–8 in Algorithm 1). This new NN-based method is detailed in Algorithm 2.

We conclude by observing the convergence properties of Algorithm 1 under particular assumptions.

Theorem 1 (Algorithm Convergence with Exact Detector). *Let S be an equispaced sparse grid of $N > 1$ points in $\mathbb{R}^n$. Let Δ^* be an exact discontinuity detector based on S. Let δ be the set of discontinuity points of a discontinuous function $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ and let $S^{(1)}, \dots, S^{(m)}$ be the m starting sparse grids of Algorithm 1.*

Assume that there exists $j \in \{1, \dots, m\}$ such that the intersection of δ with the edges of the SGG based on $S^{(j)}$ is not empty; then, for each $\varepsilon > 0$, we have that Algorithm 1:

1. *terminates in a finite number of steps.*
2. *returns a set of troubled points T such that, for each $\boldsymbol{x}^* \in T$ and $\Lambda_{\min} \leq 2\varepsilon$,*

$$\operatorname{dist}(\boldsymbol{x}^*, \delta) < \varepsilon. \tag{12}$$

Proof. Without loss of generality, we assume to start the algorithm with a unique sparse grid S. Let Λ denote the edge length of the sparse grid box $\mathbb{B}(S)$. Let $e_{ij} = \{\boldsymbol{x}_i, \boldsymbol{x}_j\}$ be an edge of the SGG based on S that intersects δ and, without loss of generality, let us assume that $\boldsymbol{x}_i$ is a troubled point (see Definition 3.1); we denote by $\boldsymbol{x}_\delta$ the point in $\delta \cap e_{ij}$ nearest to $\boldsymbol{x}_i$. Let λ_i be the length of the longest edge with $\boldsymbol{x}_i$ as vertex; then, it holds

$$\|\boldsymbol{x}_i - \boldsymbol{x}_\delta\| \leq \frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{2} \leq \frac{\lambda_i}{2}. \tag{13}$$

Algorithm 2 NN-based discontinuity detection.

Require:

$g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, function;
$\mathcal{S}$, sparse grid (equispaced, hypercubic box);
$\Delta : \mathbb{R}^N \to [0,1]^N$, NN-based discontinuity detector (only-eval. dep.), based on $\mathcal{S}$;
L, set of *center* and *edge length* pairs characterizing the boxes $\mathbb{B}(\mathcal{S}^{(1)}), \dots, \mathbb{B}(\mathcal{S}^{(m)}) \subset \Omega$;
$\Lambda_{\min} \in \mathbb{R}_+$, minimum edge length allowed for sparse grid boxes;
$\tau \in (0,1]$, detection tolerance.

Ensure:

$T \subset \mathbb{R}^n$, set of points detected as troubled points.

$T \leftarrow \emptyset$
$C \leftarrow \emptyset$ ▷ set of "visited" center-edge pairs
while L is not empty **do**
  $\{G^{(1)}, \dots, G^{(K)}\} \leftarrow$ weighted graphs based on the sparse grids with boxes defined by L
  $C \leftarrow C \cup L$
  $L \leftarrow \emptyset$
  $\Gamma \in \mathbb{R}^{K \times N} \leftarrow$ matrix $(g(\boldsymbol{x}_i^{(k)}))$ ▷ "batch of inputs", see (11)
  $P \in [0,1]^{K \times N} \leftarrow \Delta(\Gamma)$ ▷ "batch-predictions" of Δ, see (10) and (11)
  for p_{ki} element of P **do**
    if $p_{ki} \geq \tau$ **then**
      $\lambda \leftarrow \max_{\{\boldsymbol{x}_i^{(k)}, \boldsymbol{x}_j^{(k)}\}} \|\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_j^{(k)}\|$
      if $\lambda \geq \Lambda_{\min}$ **then**
        if $(\boldsymbol{x}_i^{(k)}, \lambda) \notin C$ **then**
          $L \leftarrow L \cup \{(\boldsymbol{x}_i^{(k)}, \lambda)\}$
        end if
      else
        $T \leftarrow T \cup \{\boldsymbol{x}_i^{(k)}\}$
      end if
    end if
  end for
end while

By construction, we certainly have $\lambda_i \leq \Lambda/2$; hence, from (13) we have

$$\|\boldsymbol{x}_i - \boldsymbol{x}_\delta\| \leq \frac{\Lambda}{4}\,. \tag{14}$$

According to Algorithm 1, two scenarios are possible: (*i*) $\lambda_i < \Lambda_{\min}$, then stop and assign $\boldsymbol{x}_i$ to T; (*ii*) $\lambda_i \geq \Lambda_{\min}$, then refine.
If case (*i*) occurs, (12) holds true because

$$\operatorname{dist}(\boldsymbol{x}_i, \delta) \leq \|\boldsymbol{x}_i - \boldsymbol{x}_\delta\| \leq \lambda_i/2 < \Lambda_{\min}/2 \leq \varepsilon$$

If case (*ii*) occurs, consider the sparse grid $\mathcal{S}'$ introduced in the refinement step, centered on $\boldsymbol{x}_i$ and with box edge length $\Lambda' = \lambda_i$. Let $e'_{pq} = \{\boldsymbol{x}'_p, \boldsymbol{x}'_q\}$ be an edge of the SGG based on $\mathcal{S}'$ that intersects δ such that, without loss of generality, $\boldsymbol{x}'_p$ is a troubled point; we denote by $\boldsymbol{x}'_\delta$ the point in $\delta \cap e'_{pq}$ nearest to $\boldsymbol{x}'_p$. Let λ'_p be the length of the longest edge with $\boldsymbol{x}'_p$ as vertex; then, it holds

$$\|\boldsymbol{x}'_p - \boldsymbol{x}'_\delta\| \leq \frac{\lambda'_p}{2} \leq \frac{\Lambda'}{4} \leq \frac{\Lambda}{8} \tag{15}$$

where the third inequality is due to the fact that $\Lambda' \leq \frac{\Lambda}{2}$ because at least one refinement is made, and the second inequality applies because $\lambda'_p \leq \frac{\Lambda'}{2}$, as any edge length is at least half of the sparse grid box.

Repeating the same reasoning, we obtain after k refinement steps, $k \geq 1$, the following inequalities:

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}_\delta^{(k)}\| \leq \frac{\lambda^{(k)}}{2} \leq \frac{\Lambda^{(k)}}{4} \leq \frac{\Lambda}{2^{k+2}} \quad \text{and} \quad \lambda^{(k)} \leq \frac{\Lambda}{2^{k+1}}, \tag{16}$$

where the subscript for $\boldsymbol{x}^{(k)}, \lambda^{(k)}$ is omitted for the ease of notation.

For a fixed $\Lambda_{\min} > 0$, let $\kappa \in \mathbb{N}, \kappa \geq 0$, be the smallest integer such that $\Lambda/(2^{\kappa+1}) < \Lambda_{\min}$; then, $\lambda^{(\kappa)} \leq \frac{\Lambda}{2^{\kappa+1}} < \Lambda_{\min}$ and the algorithm has a finite termination. Moreover, it holds

$$\operatorname{dist}(\boldsymbol{x}^{(\kappa)}, \delta) \leq \|\boldsymbol{x}^{(\kappa)} - \boldsymbol{x}_\delta^{(\kappa)}\| \leq \frac{\lambda^{(\kappa)}}{2} < \frac{\Lambda_{\min}}{2} \leq \varepsilon$$

and thus the thesis follows.

□

We conclude this subsection with an observation related to the potential versatility of the sparse grid-based discontinuity detectors.

Remark 3.6 (Detectors and space dimension). *A discontinuity detector based on a sparse grid S in $\mathbb{R}^n$ can also be used for detecting discontinuities of functions $g' : \Omega' \subseteq \mathbb{R}^{n'} \to \mathbb{R}$, with $n' < n$. Indeed, we can define a function $g : \Omega' \times \mathbb{R}^{n-n'} \subseteq \mathbb{R}^n \to \mathbb{R}$ as*

$$g(x_1, \dots, x_{n'}, x_{n'+1}, \dots, x_n) := g'(x_1, \dots, x_{n'}),$$

for each $\boldsymbol{x} \in \Omega' \times \mathbb{R}^{n-n'}$, and use the detector on g. Moreover, in this case Algorithm 1 can be modified to ignore any detected troubled point with at least one coordinate $x_{n'+1}, \dots, x_n \neq 0$, assuming to start with sparse grids all centered in points with coordinates $x_{n'+1} = \dots = x_n = 0$. The study of such a kind of generalization is postponed to future work.

4. NN-based discontinuity detectors

In this section, we describe the method used to build an NN-based discontinuity detector employed in Algorithm 2. Each detector is associated with a fixed sparse grid S and is designed to work only with similar sparse grids. Accordingly, a dataset generated with respect to S can be used to train only NN-based detectors constructed on the same grid structure.

We emphasize that our goal is not to train the model for specific cases, but rather, for each fixed space dimension n, to perform a single training using fairly general functions. In this way, the resulting NN can be applied in different situations, not necessarily similar to the functions used in the synthetic dataset. The NN is indeed trained to detect the presence or absence of discontinuities based solely on function evaluations at points distributed according to the sparse grid structure, regardless of their position within the domain.

More specifically, given an n-dimensional sparse grid S with N nodes, for any N-dimensional vector of function evaluations at nodes distributed according to S, the NN returns an N-dimensional vector whose entries are either 0 or 1 (1 if a discontinuity is detected at the corresponding point, and 0 otherwise). Using a synthetic dataset built from randomly generated discontinuous functions (as described in Section A.2), the network learns to identify jumps in the portion of the domain where the sparse grid is positioned and the function is evaluated. This design ensures robust generalization capabilities, as confirmed by the results of the numerical tests presented in Section 5.

The entire procedure is sketched in Section 4.1. Then, in the following subsections, we provide details about the loss function and NN models. We refer the reader to Appendix A for further details related to the building of a deterministic approximation of an exact detector (A.1), the creation of synthetic data for the training (A.2), data preprocessing (A.3), and NN architecture archetypes and hyperparameters (A.4 and A.5).

4.1. Building procedure for NN-based detectors

The construction of an NN-based detector is based on the following steps:

1. Choose a sparse grid S in $\mathbb{R}^n$. Let N denote the number of points in S.
2. Generate a set of random piecewise continuous functions $\mathcal{G} = \{g^{(1)}, \dots, g^{(Q)}\}$ with known discontinuities, contained in the zero-level sets $\zeta^{(q)}$ of a function $f^{(q)}$, for each $q = 1, \dots, Q$.
3. For each $g^{(q)} \in \mathcal{G}$, run Algorithm 1 with an exact detector Δ^* (or a deterministic approximation, see A.1 for details); then, at each detection step with respect to a sparse grid $S' \simeq S$, compute:
   - the evaluations of $g^{(q)}$ at the sparse grid points, i.e., the vector $\boldsymbol{g}' = (g^{(q)}(\boldsymbol{x}'_1), \dots, g^{(q)}(\boldsymbol{x}'_N))$;
   - the corresponding value of $\boldsymbol{p} = \Delta^*(\boldsymbol{g}')$.
4. Create a dataset of pairs $(\boldsymbol{g}', \boldsymbol{p})$ collecting the vectors obtained at the previous step.
5. Split the dataset into training, validation, and test set (denoted by $\mathcal{T}$, $\mathcal{V}$, and $\mathcal{P}$, respectively).
6. Given an NN with characterizing function $\widehat{\Delta} : \mathbb{R}^N \to [0,1]^N$, train the model with respect to $\mathcal{T}$ and $\mathcal{V}$ to obtain an approximation of the exact discontinuity detector Δ^*, i.e.:

   $$\widehat{\Delta}(\boldsymbol{g}') \approx \Delta^*(\boldsymbol{g}'),$$

   for each vector $\boldsymbol{g}'$ of function evaluations at the grid points of S'.
7. Measure the performance of the trained NN with respect to $\mathcal{P}$.

The most expensive part of this procedure is item 3, because it involves a deterministic (approximation of) Δ^*; typically, this detector is expensive, especially when we have a high-dimensional space.

4.2. Loss function

Concerning the loss function for training an NN-based detector, several choices are possible. In Ref. [3], the Mean Squared Error is used as loss function for training the convolutional detector. However, since the target vectors $\boldsymbol{p}$ have values in $\{0, 1\}$ (non-troubled point and troubled point, respectively), we prefer to use the loss function

$$\mathcal{L}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{(\boldsymbol{g}', \boldsymbol{p})} \sum_{i=1}^{N} -\mu_1 p_i \log(\widehat{p}_i) - \mu_0 (1 - p_i) \log(1 - \widehat{p}_i), \tag{17}$$

for any batch $\mathcal{B}$ of input-output data, and where $\hat{\boldsymbol{p}} := \hat{\Delta}(\boldsymbol{g}') \in [0,1]^N$; namely, $\mathcal{L}$ is the average sum of the component-wise application of the binary cross-entropy loss (see Refs. [34,35]). The hyperparameters μ_1 and μ_0 are used to suitably weight the identification of troubled points or non-troubled points, respectively.

4.3. NN models: MLPs and GINNs

Looking at step 6 of the building procedure for NN-based detectors (Section 4.1), we observe that the only practical restrictions for the NN model consist of using an input layer of N units and an output layer of N units with values in $[0,1]$. Therefore, any architecture is allowed for the hidden layers.

Since Algorithm 1 is based on SGGs with the same adjacency matrix (see Proposition 2.1), it is reasonable to train an NN model tailored for tasks defined on graph-structured data, where the graph is fixed and only the node features vary. In Ref. [28], Berrone et al. recently observed that a good choice for learning a target function $\boldsymbol{F} : \mathbb{R}^N \to \mathbb{R}^N$, dependent on the adjacency matrix of a fixed graph G, is to use a Graph-Instructed Neural Network (GINN) because they perform better than Multi-Layer Perceptrons (MLPs) and classic Graph Neural Networks (GNNs). The GINN models are part of the wide family of spatial GNNs and have been specifically developed for such a kind of task (see Ref. [28]). Indeed, GNNs are mainly designed for other tasks than learning $\boldsymbol{F}$ (e.g., graph classification, semi-supervised learning on nodes, etc.), while MLPs do not exploit the graph connections (Refs[28,29,36].). We point the reader to the fact that in Ref. [28] GI layers and GINNs are denoted as Graph-Informed layers and Graph-Informed NNs, respectively. In Ref. [37], in a different framework from the one addressed in [28], a homonymous but different model is presented; therefore, starting from the work [29] of Della Santa et al., the authors changed the names of both layers and NNs, to avoid confusion with Ref. [37].

Due to the observations about MLPs and GINNs, we focus here on these architectures for building the NN-based detectors. Moreover, the numerical experiments in Section 5 confirm the observation of Ref. [28]; i.e., we observe that GINN-based detectors perform better than MLP-based detectors.

4.3.1. Graph-instructed neural networks

A GINN is an NN characterized by Graph-Instructed (GI) Layers. These layers are defined by an alternative graph-convolution operation introduced in Ref. [28]. In brief, from a message-passing point of view, this graph-convolution operation is equivalent to having each node v_i of G sending to its neighbors a message equal to the input feature x_i, scaled by a weight w_i; then, each node sum up all the messages received from the neighbors, add the bias, and applies the activation function. We point the attention of the reader to the fact that, only in this subsection, $\boldsymbol{x}$ denotes feature vectors and not the sparse grid points.

In a nutshell, the message-passing interpretation can be summarized by Fig. 5 and the following node-wise equation

$$x_i' = \sum_{j \in \mathrm{N}_{\mathrm{in}}(i) \cup \{i\}} x_j \, w_j + b_i, \tag{18}$$

where x_i' is the layer output feature corresponding to node v_i and $\mathrm{N}_{\mathrm{in}}(i)$ is the set of indices such that $j \in \mathrm{N}_{\mathrm{in}}(i)$ if and only if $e_{ij} = \{v_i, v_j\}$ is an edge of the graph. We dropped the action of the activation function for simplicity.

Starting from (18), we can formally define the GI layers through a compact formulation. Given a graph G (without self-loops) and its adjacency matrix $A \in \mathbb{R}^{N \times N}$, the simplest version of GI layer for G is an NN layer described by a function $\mathcal{L}^{GI} : \mathbb{R}^N \to \mathbb{R}^N$, with one input feature per node and one output feature per node, such that

$$\mathcal{L}^{GI}(\boldsymbol{x}) = \boldsymbol{\psi}\big((\,\mathrm{diag}(\boldsymbol{w})(A + \mathbb{I}_N)\,)^T \boldsymbol{x} + \boldsymbol{b} \big), \tag{19}$$

where $\boldsymbol{x} \in \mathbb{R}^N$ denotes the vector of input features and:

- $\boldsymbol{w} \in \mathbb{R}^N$ is the weight vector, with the component w_i associated to the graph node v_i, $i = 1, \dots, N$;

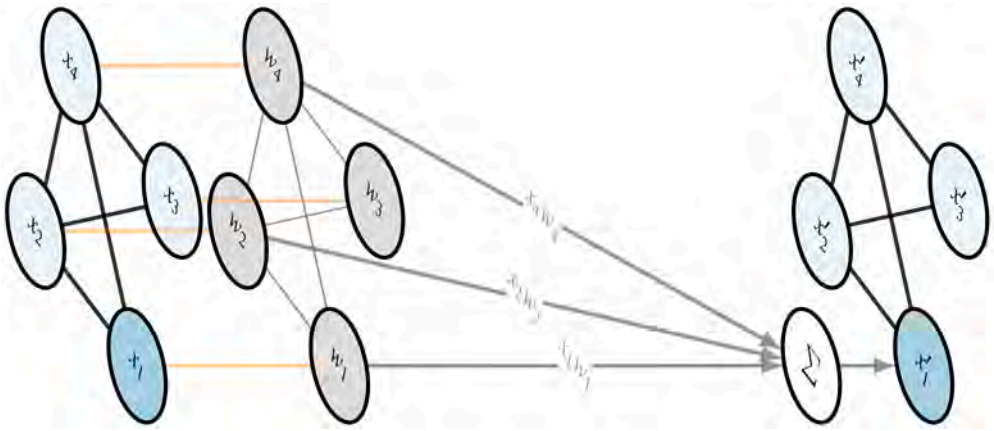
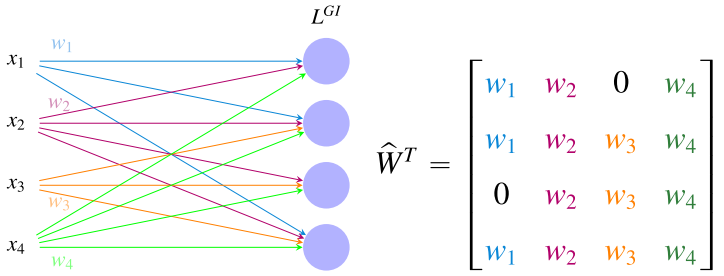


Fig. 5. Visual representation of (18). Example with $n = 4$ nodes (undirected graph), $i = 1$; for simplicity, the bias is not illustrated.



(a) FC representation of GI layer (b) Weight matrix of GI layer

Fig. 6. Visual representation of a GI layer as a "constrained" FC layer (subfigure (*a*)), with weight matrix defined by (20) (subfigure (*b*)). This figure is based on the same graph illustrated in Fig. 5.

- $\mathrm{diag}(\boldsymbol{w}) \in \mathbb{R}^{N\times N}$ is the diagonal matrix with elements of $\boldsymbol{w}$ on the diagonal and $\mathbb{I}_N \in \mathbb{R}^{N\times N}$ is the identity matrix. For future reference, we set $\widehat{W} := \mathrm{diag}(\boldsymbol{w})(A + \mathbb{I}_N)$;
- $\boldsymbol{\psi} : \mathbb{R}^N \to \mathbb{R}^N$ is the element-wise application of the activation function ψ;
- $\boldsymbol{b} \in \mathbb{R}^N$ is the bias vector.

From another point of view, Eq. (19) is equivalent to the action of a "constrained" Fully-Connected (FC) layer where the weights are the same if the connection is outgoing from the same unit, whereas it is zero if two units correspond to graph nodes that are not connected (see Fig. 6); more precisely:

$$\widehat{w}_{ij} = \begin{cases} w_i, & \text{if } a_{ij} \neq 0 \text{ or } i = j \\ 0, & \text{otherwise} \end{cases}, \tag{20}$$

where $a_{ij}, \widehat{w}_{ij}$ denote the (i,j)th element of $A, \widehat{W}$, respectively.

Layers characterized by (19) can be generalized to read any arbitrary number $K \geq 1$ of input features per node and to return any arbitrary number $F \geq 1$ of output features per node. Then, the action of a general GI layer is a function $\mathcal{L}^{GI} : \mathbb{R}^{N\times K} \to \mathbb{R}^{N\times F}$. Additionally, pooling and mask operations can be added. For more details about GI layers, see [28,29,38].

Note that the GI layer definition is very general and does not require the graph to be unweighted. Indeed, one of the main advantages of using a GI layer is that it exploits the zeros of the adjacency matrix A, "pruning" the connections between layer units according to the graph edges; therefore, the zero entries of A are in the same positions, independently of the use of edge weights.

The observation above is important to build GINNs based on the WSGG of a given sparse grid. In particular, we prefer a weighted graph because we want to emphasize the difference between connections according to the distance of the nodes in $\mathbb{R}^n$. Then, using a weighted graph for the GINN, we have that the action of the GINN weights is rescaled with respect to the non-trainable edge weights $\omega_{ij} \in (0, 1]$ (see Definition 2.4).

5. Numerical experiments

In this section, we apply the proposed NN-based discontinuity detection method (see Algorithm 2) on various test functions. In the first experiments, we use both MLP-based and GINN-based detectors (for details on architectures, see A.4); then, we focus on GINN-based detectors, due to their better performance. We recall that our NN-based detectors detect the troubled points reading only the function evaluations at the sparse grid points; then, they can be applied to any sparse grid (similar to the one used for building the NN model) and to any discontinuous function with domain dimension equal to the one of the sparse grid (see Remark 3.2) or, in principle, with smaller domain dimension (see Remark 3.6).

We show results for: *i*) dimension $n = 2$, with application also to the boundary detection problem for images (see Section 5.1); *ii*) dimension $n = 4$, in order to visualize the potential of the method for detecting discontinuities in higher-dimensional spaces (see Section 5.2), and applied to a real-world test case with a complex, and not *a priori* known, discontinuity pattern (see Section 5.3); *iii*) dimensions $n = 6, 8$, to analyze the applicability of the method in higher-dimensional cases and to propose further developments that could take advantage of particular structures in the discontinuity interface, if present (see Section 5.4). After the numerical experiments, we end discussing the potential and the cost of the proposed method in a dedicated subsection (see Section 5.5).

For all the experiments contained in this section, each NN is defined and trained by using the same set of hyperparameters and training configuration, and just one sparse grid. Concerning the training options, the choices have been made after a preliminary exploration of the learning problem and have been guided by the authors prior experience in training NNs for regression and classification problems. Regarding the choice of the sparse grid, it was made by ensuring a balance between the two opposing requirements of having few nodes and achieving a good spatial distribution. The detection process can be initialized with a larger number of points, not necessarily by introducing a sparse grid with a higher number of nodes, but rather by adding multiple SGs of different box sizes, placed in various regions of the domain. The nested structure of the SG (see Remark 3.4) may help in this respect to save on the number of visited point.

Further investigation of the optimal choice of hyperparameters, training options, and sparse grid characteristics is deferred to future work, as it would require a substantially broader analysis.

5.1. Two-dimensional discontinuity detection

We start the numerical experiments testing our method on 2-dimensional test functions.

Following the notation introduced in Section 2.1, let S_2 be the 2-dimensional sparse grid $S_2 := \mathcal{I}_{\text{sum}}(4)$ ($N = 65$ points, see Eq. (2) and Figs. 1 and 2). We randomly generate a set $\mathcal{G}_2$ of $Q = 600$ piecewise continuous functions each one characterized by a linear, spherical, or polynomial discontinuity interface identified by the zero-level set of given functions (i.e., 200 functions for each type, see Appendix A.2 for details); from now on, for simplicity, we refer to the discontinuity interfaces as "*cuts*". Then, by running Algorithm 1 and detecting the zero-level sets containing the discontinuity interfaces of the functions in $\mathcal{G}_2$ (we use the detector $Z^{(150)}$ based on S_2, described in Appendix A.1), we build the synthetic dataset

$$D_2^{\text{tmp}} := \left\{ (\boldsymbol{g}', \boldsymbol{p}) \in \mathbb{R}^{\mathbb{N}} \times \{0,1\}^N \mid \boldsymbol{g}' = (g(\boldsymbol{x}'_1), \dots, g(\boldsymbol{x}'_N)),\ g \in \mathcal{G}_2,\ S' \in S_{/\simeq};\ \boldsymbol{p} = Z^{(150)}(\boldsymbol{g}') \right\}. \tag{21}$$

However, most of the detected/generated pairs $(\boldsymbol{g}', \boldsymbol{p})$ are characterized by null vectors $\boldsymbol{p}$ (i.e., are continuous-region samples); they are in a number D_0 that is much larger than the number D_1 of samples with non-null vector $\boldsymbol{p}$. Therefore, in order to have a more balanced dataset, we randomly keep D'_0 pairs out of D_0 and discard the others, where

$$D'_0 := \max_{i=1,\dots,N} D_1^{(i)},$$

and $D_1^{(i)}$ denotes the number of generated samples where $\boldsymbol{p}$ has exactly i non-null values. We denote by D_2 the final dataset of $(\boldsymbol{g}', \boldsymbol{p})$ samples obtained.

Given D_2, we build an MLP-based detector and a GINN-based detector and we train them with the following training options:

- Dataset split: the test set cardinality $|\mathcal{P}_2|$ is 30% of $|D_2|$, the training set cardinality $|\mathcal{T}_2|$ is 80% of $|D_2| - |\mathcal{P}_2|$, and the validation set cardinality is $|\mathcal{V}_2| = |D_2| - |\mathcal{P}_2| - |\mathcal{T}_2|$;
- Loss function: loss (17), with $\mu_0 = 0.5$ and $\mu_1 = 1.5$;
- Mini-batch size: 64;
- Optimizer: Adam [39] (learning rate $\epsilon = 0.001$, moment decay rates $\beta_1 = 0.9$, $\beta_2 = 0.999$);
- Learning rate decay: reduce on plateau [40] (factor 0.75, 7 epochs of patience);
- Regularization: early-stopping method with best-weights restoration [41] (35 epochs of patience).

Concerning the NN-based detectors, we use a *leaky relu* activation function, see Ref. [42], a *Glorot Normal* initialization for the weights, see Ref. [43], and (for the GINN-based detector) $F = 15$ filters for the hidden layers; both MLPs and GINNs are characterized by layers of $N = |S|$ units and by residual blocks (see Ref. [44]) for exploiting depth in the models, depth that is proportional to the diameter of the SGG. For more details about the model architectures, see Appendix A.4.

At the end of the training, we measure the loss and, for a better understanding, the Mean Absolute Error (MAE) of these models on the test set $\mathcal{P}_2$ (see Table 2). Looking at the values in the table, we see that the GINN prediction performances are slightly better than the MLP ones, at a reasonable cost in terms of trainable parameters. In particular, there are more trainable parameters in the GINN model because the GI hidden layers are endowed with $F = 15$ filters; nonetheless, despite the increased representational capacity of the GINN resulting from its 15 filters, thanks to the sparsity of the adjacency matrix of the SGG, the number of trainable parameters is only marginally more than three times that of the MLP.

We test the discontinuity detection abilities of these NN models used as discontinuity detectors together with the sparse grid-based algorithm proposed (see Algorithm 2). In particular, we test them on four functions defined on a square domain Ω (see Figs. 7 and 8):

1. a function with circular cut, domain $\Omega = [-1,1]^2$;
2. a function with Legendre polynomial cut, domain $\Omega = [-1,1]^2$;
3. a function with one elliptic cut and two bow-shaped cuts, domain $\Omega = [-1,1]^2$;
4. a function with two nonsmooth cuts, one ray cut, and one point belonging to the discontinuity interface, domain $\Omega = [-4,4]^2$.

We emphasize that functions (i) and (ii) are of the same type of functions used to generate the synthetic dataset D_2. On the other hand, functions (iii) and (iv) are functions of a different type with respect to the piecewise polynomial functions, with smooth discontinuity interface of codimension 1, that are in $\mathcal{G}_2$. Then, they are somehow "new" to the NN models, and good performances

Table 2
Two-dimensional case. Performances on $\mathcal{P}_2$ of the MLP-based and GINN-based detectors.

NN model	# trainable params	Loss function value	MAE
MLP	52 910	4.8678	0.0480
GINN	173 880	3.5396	0.0393

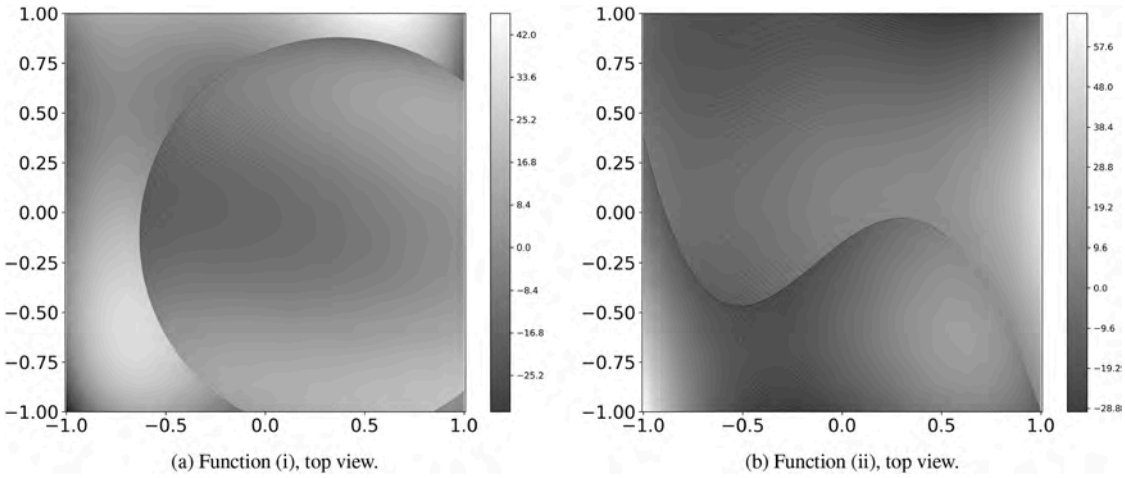


Fig. 7. Test functions (i)-(ii), same type as synthetic dataset functions.

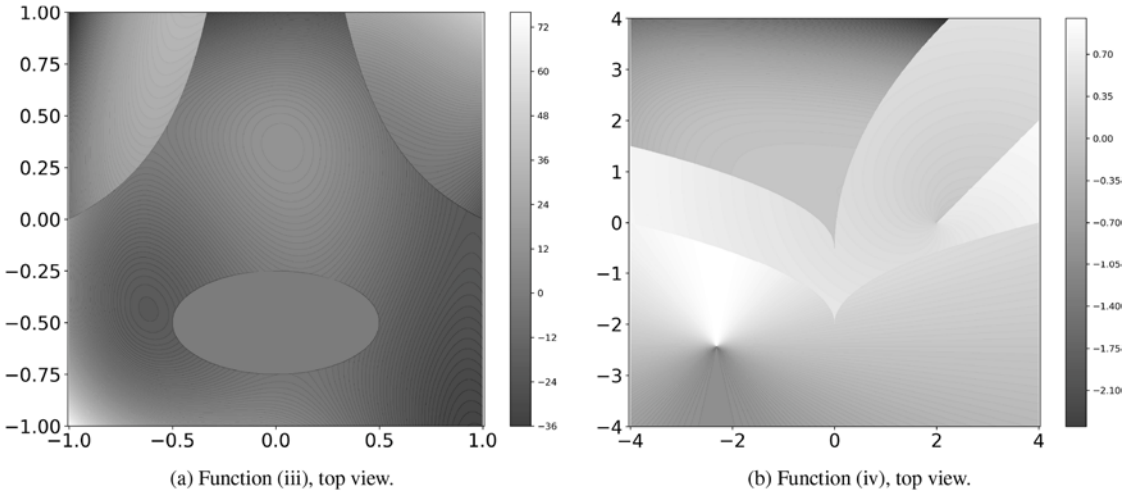


Fig. 8. Test functions (iii)-(iv), different in type from the synthetic dataset functions.

on these functions witness good generalization abilities of the NNs. In particular, function (iv) is fundamental to understand if the NN-based detectors can detect discontinuities that are impossible to be detected by exact detectors (see Remark 3.1); indeed, the discontinuity point of function (iv) has coordinates (-2.31, -2.43), and it never crosses the edges of the SGGs during the detection procedure (by construction).

Additionally, looking at Figs. 7 and 8, we can observe that all the test functions have discontinuity interfaces characterized by parts where the discontinuity jump is approximately zero, i.e., where the interface is not of (evident) discontinuity; then, they are important to understand how much sensitive the NN-based detectors are in identifying discontinuities.

The performances of the NN-based detectors are measured according to a True Positive Rate (TPR). Let T be the set of final troubled points detected by the method and let $\Lambda_{\min}$ be the minimum edge length used as stopping criterion (see Algorithms 1 and 2); then, the TPR is defined as TPR = (*#true troubled points*)/$|T|$, where $\boldsymbol{x} \in T$ is considered as a true troubled point if its distance, in infinity norm, from the discontinuity interface is smaller or equal than $\Lambda_{\min}/2$. If the distance cannot be analytically computed, it is approximated as follows:

1. let ζ be the zero-level set of the function characterizing the discontinuity interface of the test function. Let $\widetilde{S}_2$ be the 2-dimensional sparse grid represented by the blue dots in Fig. 9a;
2. for each $\boldsymbol{x} \in T$, center on $\boldsymbol{x}$ a sparse grid $\widetilde{S}'_2(\boldsymbol{x}) \simeq \widetilde{S}_2$ with box of edge length $\Lambda_{\min}$. We denote by $\widetilde{G}(\boldsymbol{x})$ the SGG of $\widetilde{S}'_2(\boldsymbol{x})$ (see Fig. 9a);
3. $\boldsymbol{x}$ is considered a true troubled point if ζ intersects at least one edge of $\widetilde{G}(\boldsymbol{x})$ (see Fig. 9b).

Remark 5.1 (Focus on TPR)**.** *For measuring the detection quality of our method, we do not consider a True Negative Rate (TNR) because a definition of* true non-troubled point *based on the distance from ζ (like* true troubled points *in TPR) would be misleading, even in the case of exact detectors. In fact, Algorithm 1 labels a point $\boldsymbol{x}_i$ as* non-troubled point *not because it is far from the discontinuity interface,*

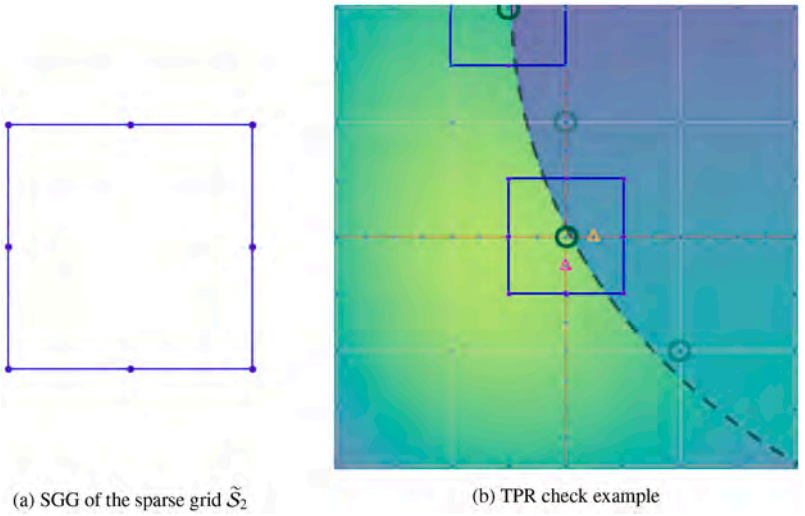


Fig. 9. Left: SGG based on the sparse grid $\widetilde{S}_2$. Right: use of the SGG for computing the TPR. Red segments are edges of the detector's SGG; green circles are troubled points detected at the current step (dark green for final troubled points and light green for points used as grid centers in the next step); in the $\widetilde{S}_2$ grid in the middle, orange and magenta triangles highlight two examples of non-troubled points sufficiently near to ζ (orange triangle if an edge intersects ζ, magenta triangle if no edge intersects ζ, see Remark 5.1).

but according to other criteria. Specifically, $\boldsymbol{x}_i$ *is a* non-troubled point *either because no edge with* $\boldsymbol{x}_i$ *as vertex intersects the discontinuity interface (e.g., see the magenta triangle in Fig. 9b) or because there exists an edge* $e_{ij} = \{\boldsymbol{x}_i, \boldsymbol{x}_j\}$, $\|\boldsymbol{x}_i - \boldsymbol{x}_j\| < \Lambda_{\min}$, *such that* e_{ij} *intersects the discontinuity interface but* $\boldsymbol{x}_j$ *is closer to the intersection than* $\boldsymbol{x}_i$ *(e.g., see orange the triangle in Fig. 9b). In both cases,* $\boldsymbol{x}_i$ *is marked as* non-troubled point *not due to its distance from the discontinuity interface, but because there were better alternatives to it. To better understand these observations, we report in Fig. 9b an example in which two true troubled points are detected; in the figure we highlight two examples of non-troubled points inside the area delimited by the grid* $\widetilde{S}_2$ *in the middle of the picture: the orange triangle marks a non-troubled point that has an edge intersecting* ζ*, and the magenta triangle denotes a non-troubled point without edges that intersect* ζ*. If we would use these points as centers for the grid* $\widetilde{S}_2$ *adopted to check the* ζ*-intersection of the detected troubled points, the grid would intersect* ζ*. For this reason a definition of* true non-troubled point *based on the distance from* ζ *like the* true troubled points *would be misleading.*

We run Algorithm 2 on the four test functions using the NN-based detectors and using as input parameters the same (relative to Ω) starting set L of center and edge length pairs, and the same $\Lambda_{\min} = 2/2^{(h_{\max}+1)} = 2^{-5}$ for the stopping criterion. It is worth noticing how Algorithm 2 performs an exploration of the input space even going partially beyond the boundaries of the domain Ω (see, e.g., Fig. 10, reporting the detection steps performed by the GINN-based detector on the test function (iv)). Moreover, we notice how the detection algorithm actually exploits the NN batch computations, implicitly parallelizing multiple detection steps for multiple sparse grids at the same time, completing the detection procedure in few steps.

In Table 3, we report for each test function the TPRs, the number of troubled points detected, and the total number of points visited by the two NN-based discontinuity detectors. Looking at the results reported in this table, we observe that both the NN-based detectors have extremely good TPRs (at least 98%). According to the test set performances (see Table 2), the GINN-based detector is more precise than the MLP-based one (higher TPRs and TPR equal to 100% in two cases over four); moreover, more troubled points are detected. In particular, looking at the positioning of the detected troubled points (Figs. 11 and 12), we can deduce that the higher sensitiveness of the GINN-based detector depends on its ability to detect troubled points even if the discontinuity jumps are very small.

In general, we can claim that the given GINN-based detector is a very good 2-dimensional detector for finding discontinuity interfaces via Algorithm 2; indeed, it is able to detect troubled points even in presence of very small discontinuity jumps and its TPRs on the test functions are always approximately 100%. On the other hand, the MLP-based detector proves to be a good detector, but it can miss some parts of the discontinuity interfaces due to its lower sensitiveness.

5.1.1. Edge detection problem in images

Black and white images can be modeled as functions $g : \Omega \subset \mathbb{R}^2 \to [0, 1]$, where $g(\boldsymbol{x})$ is the grayscale level of the pixel, for each pixel $\boldsymbol{x} \in \Omega$. Then, two-dimensional detectors can be used for detecting edges in pictures, since edges are discontinuity interfaces of g. Analogously to Wang et al. in Ref. [3], we test our NN-based two-dimensional detectors on the edge detection problem of the Shepp-Logan phantom Ref. [45], varying the image resolution. In particular, let $M = (M_{ij}) \in [0, 1]^{r \times r}$ be the matrix representing the Shepp-Logan phantom image with resolution $r \times r$ pixels; then, we apply Algorithm 2 modeling the image as a function $g_r : \mathbb{R}^2 \to [0, 1]$

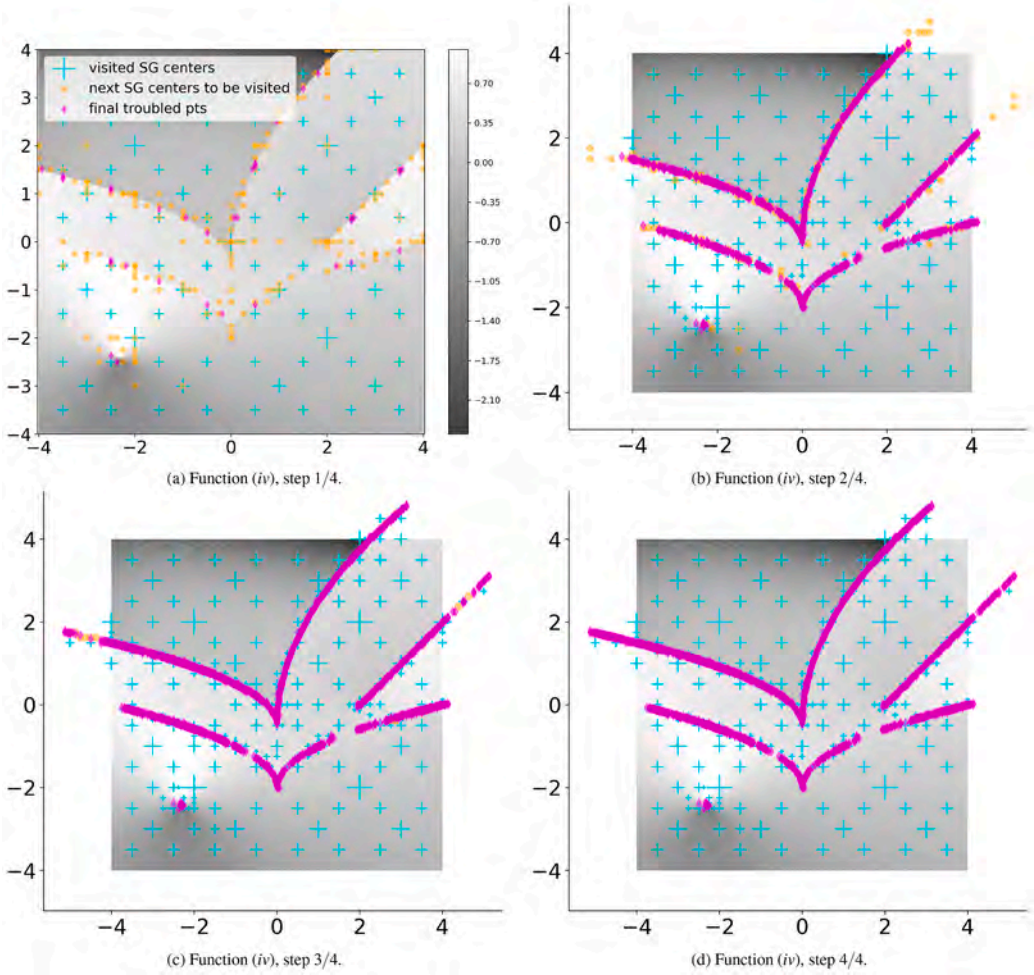


Fig. 10. Detection steps performed by Algorithm 2 with the GINN-based detector on test function (iv). Cyan crosses represent the centers of the visited SGs (marker size proportional to the edge length of the SG Box); orange dots represent the troubled points detected that will be used as new centers for smaller SGs in the next step; magenta diamonds represent the final troubled points.

Table 3
Results of Algorithm 2 with NN-based detectors for test functions (i)-(iv).

NN model	Test function	TPR	Num. of troubled points	Num. of visited points
MLP	(i)	99.74%	759	13 601
GINN	(i)	100%	910	13 466
MLP	(ii)	98.29%	403	9 928
GINN	(ii)	99.86%	740	11 609
MLP	(iii)	99.77%	898	14 883
GINN	(iii)	100%	1 049	14 192
MLP	(iv)	98.03%	798	15 955
GINN	(iv)	99.83%	1 155	16 939

such that:

$$g_r(\boldsymbol{x}) = \begin{cases} M_{[x_1]+1,[x_2]+1}, & \text{if } 0 \le [x_1],[x_2] \le r-1 \\ 0, & \text{otherwise} \end{cases}, \tag{22}$$

where $[x]$ is the rounding to the nearest integer of x, for each $x \in \mathbb{R}$.

In Fig. 13, we report the results obtained on g_r using Algorithm 2 and the two NN-based discontinuity detectors, for $r = 512, 1024$. From the figure, we observe that the edge detection performances of the NN-based method increase with the resolution, similarly to the CNN detectors of [3]; however, we see that the GINN-based detector shows very good performances even in the resolution 512×512 while the MLP-based detector obtains acceptable results (despite some imperfections) only for resolution 1024×1024 (see

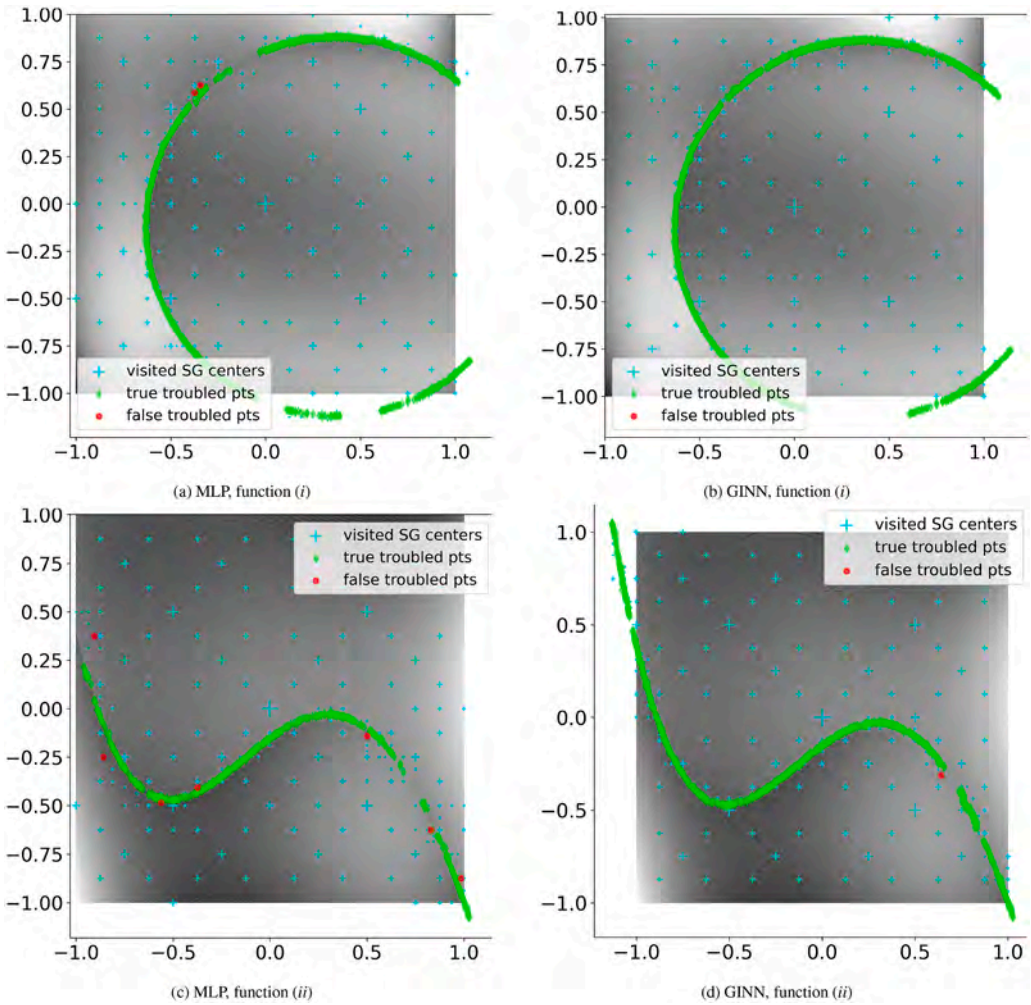


Fig. 11. Final troubled points detected for the test functions (i) and (ii) by Algorithm 2, using the NN-based detectors. Green dots are true troubled points, red circles are false troubled points (only very few are present). Cyan crosses represent the centers of the visited SGs (marker size proportional to the edge length of the SG Box).

Fig. 13). Therefore, we have further experimental evidence of the better detection abilities of the GINN-based detector with respect to the MLP-based one.

A very interesting characteristic of our method applied to the edge detection problem can be observed in the rightmost column of Table 4, which reports the number of troubled points detected and the total number of points visited by the NN-based detectors, for each image resolution. Indeed, we observe that increments in the resolution do not imply a considerably higher number of points visited by the method. This characteristic is very important because it means that increments in the image resolution guarantee better results at a reduced cost; in particular, we have that the function evaluations done by the algorithm are less than 250^2 for resolution 512, and approximately 350^2 or less for resolution 1024; in both cases, the function evaluations (i.e., number of pixels visited) is much smaller than the number of pixels available in the image (512^2 and 1024^2, respectively).

In the end, we remark that the Shepp-Logan phantom problem represents an unconventional test cases for our trained NNs. Indeed, as already mentioned, both the MLP-based detector and the GINN-based detectors have been trained on piecewise continuous synthetic polynomial functions, each one with a single random linear, spherical, or polynomial cut. Therefore, function (22) represents an unexplored case for our method, as it exhibits features that do not correspond to those of the functions in the synthetic dataset (no interfaces consisting of overlapping or nested ellipses of different sizes were included in the training set).

5.2. Four-dimensional discontinuity detection

Following the considerations made before, in this subsection we focus on a GINN-based detector, as in general GINNs perform better than MLPs. The sparse grid we consider is $S_4 := \mathcal{I}_{\text{sum}}(4)$ in $\mathbb{R}^4$ ($N = 401$ points). Then, we consider a randomly generated set $\mathcal{G}_4$ of 750 functions, again with linear, spherical, or polynomial cut (250 each). We use a larger set of functions (with respect to Section 5.1)

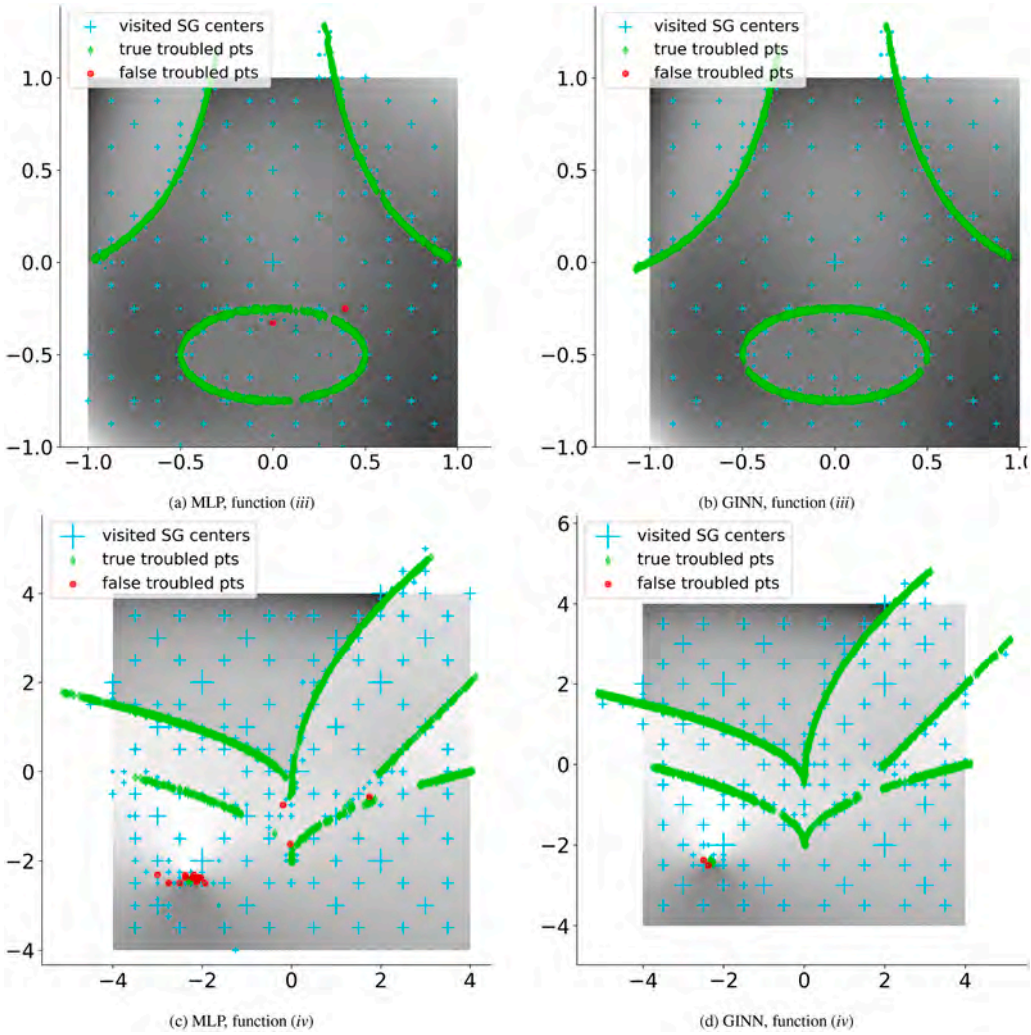


Fig. 12. Final troubled points detected for the test functions (iii) and (iv) by Algorithm 2, using the NN-based detectors. Green dots are true troubled points, red circles are false troubled points (only very few are present). Cyan crosses represent the centers of the visited SGs (marker size proportional to the edge length of the SG Box).

Table 4
Results obtained by Algorithm 2, using NN-based detectors, on the Shepp-Logan phantom problem at different resolutions.

NN model	Image Resolution	Num. of troubled points	Num. of visited points
MLP	512×512	3 463	46 501 ($\sim 216^2$)
GINN	512×512	6 144	61 045 ($\sim 247^2$)
MLP	1024×1024	5 820	78 721 ($\sim 280^2$)
GINN	1024×1024	11 314	123 224 ($\sim 351^2$)

because we use a less precise approximation of the exact detector for the creation of the synthetic dataset ($Z^{(3)} = Z^{(t+1)}\big|_{t=2}$, see Appendix A.1 for details); indeed, we prefer a less precise detector to make the dataset creation process less expensive, even if at the cost of a less detailed dataset.

Given the sparse grid and $\mathcal{G}_4$, we proceed analogously to Section 5.1: we build a GINN-based detector with respect to the SGG of S_4, we generate a synthetic dataset D_4, and we train the GINN-based detector with the same set of hyperparameters and training options defined for the two-dimensional case (see Section 5.1 and Appendix A.4). The performances of this detector on the test set P_4 are reported in Table 5.

Given the trained GINN-based four-dimensional detector, we test its abilities running Algorithm 2 ($\Lambda_{\min} = 2/2^{(h_{\max})} = 2^{-4}$) on a piecewise continuous four-dimensional test function. Specifically, we use a test function $g_\vartheta : \Omega \subseteq \mathbb{R}^4 \to \mathbb{R}$ with domain $\Omega = [-1, 1]^4$,

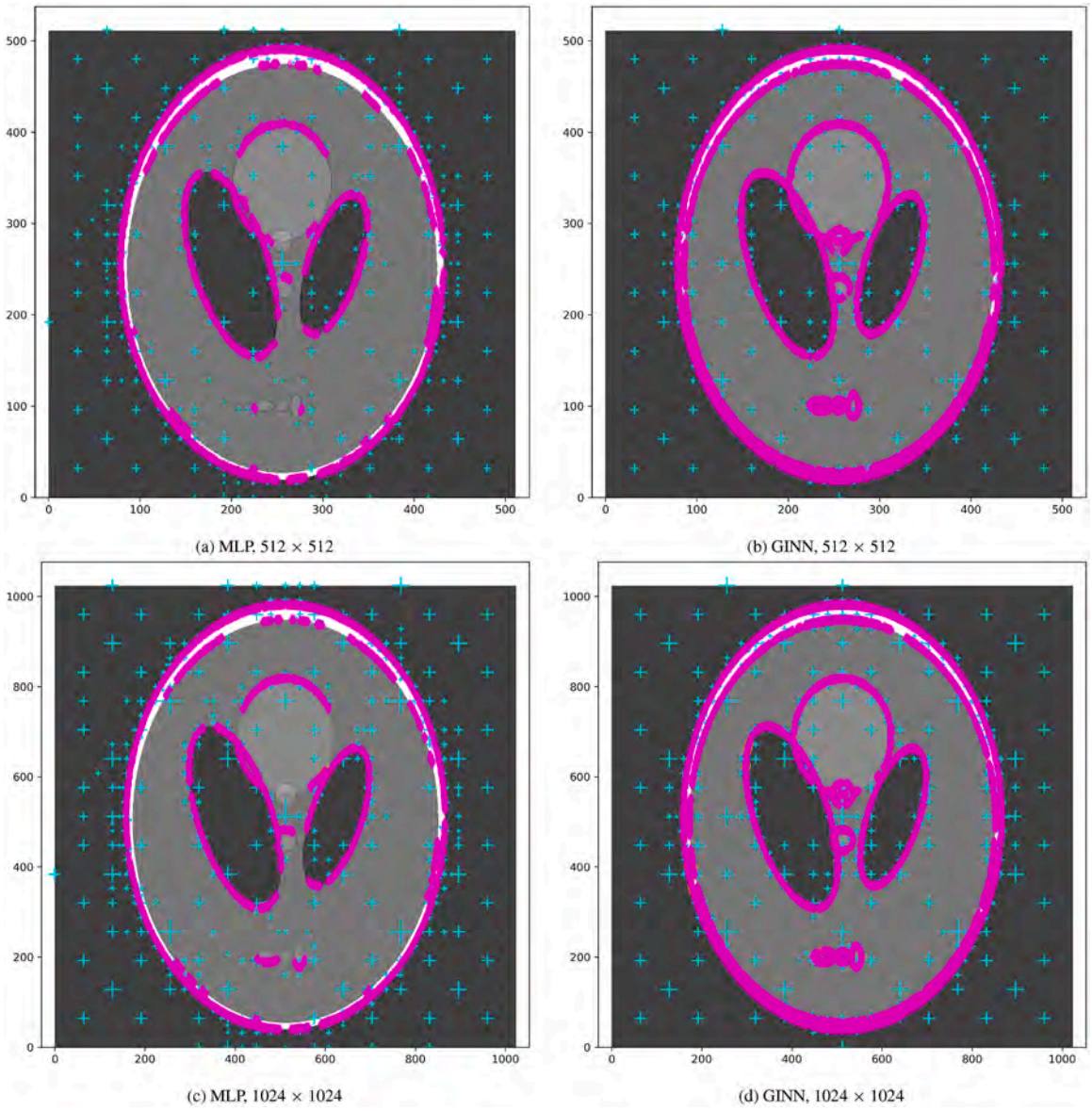


Fig. 13. Troubled points detected for the Shepp-Logan phantom problem at different resolutions by Algorithm 2, using the NN-based detectors. Magenta dots are the troubled points. Cyan crosses are the centers of the visited SGs (marker size proportional to the edge length of the SG Box).

Table 5
Four-dimensional case. Performances on $\mathcal{P}_4$ of the GINN-based detector.

NN model	# trainable param.s	Loss function value	MAE
GINN	1 263 540	37.5907	0.0585

continuous pieces defined by four-dimensional Legendre polynomials (see equation (A.3), Appendix A.2), and as discontinuity interface a torus with size depending on x_4; i.e., the discontinuity interface of g_ϑ is characterized by the zero-level set of the function

$$\vartheta(\boldsymbol{x}) = \left(|x_4| - \sqrt{x_1^2 + x_2^2}\right)^2 + x_3^2 - (|x_4|/4)^2 \,. \tag{23}$$

In Table 6, we report the TPR, the number of troubled points detected, and the total number of points visited by the GINN-based detector; then, looking at this table, we see a TPR of 99.14% for more than two millions of troubled points. Moreover, the total number

Table 6
Results of Algorithm 2 on test function g_ϑ, using the four-dimensional GINN-based detector.

NN model	TPR	Num. of troubled points	Num. of visited points
GINN	99.14%	2 607 122	28 805 451 ($\sim 73^4$)

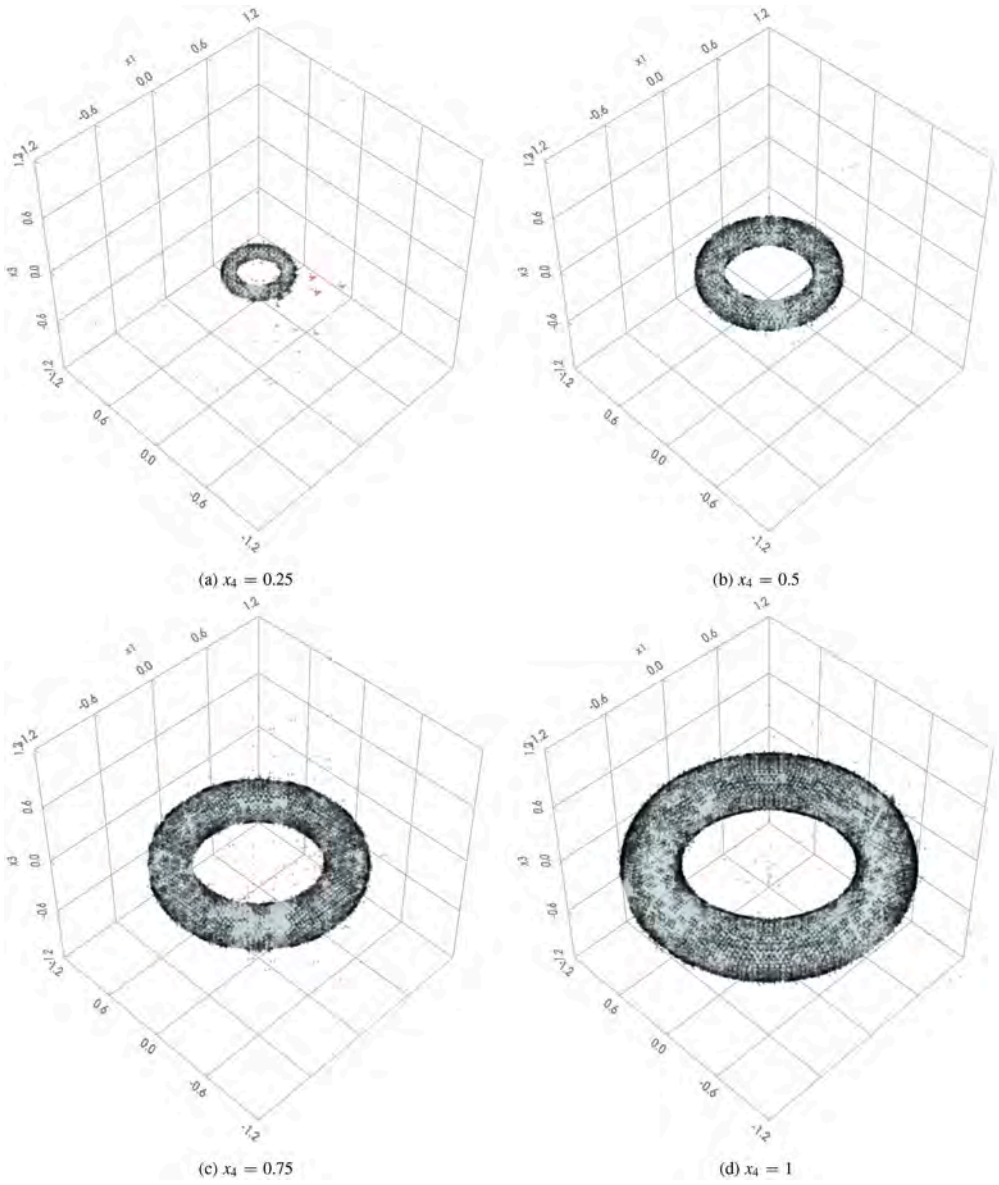


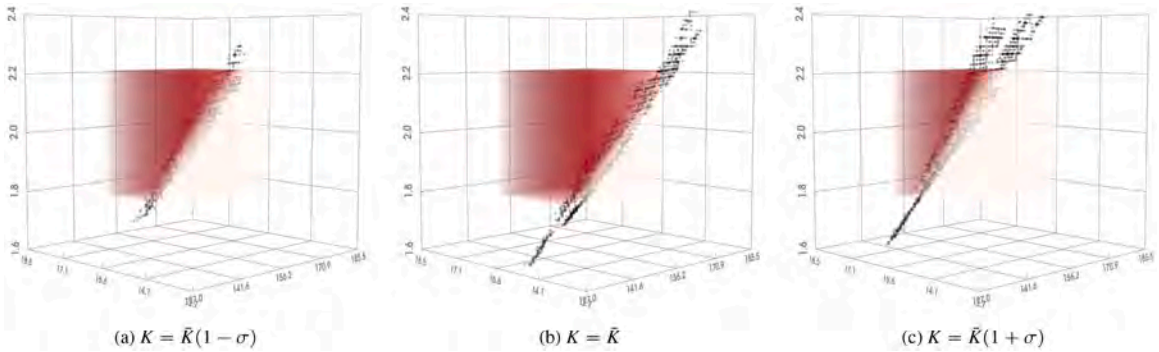
Fig. 14. Visual examples of the troubled points detected by the GINN-based detector for g_ϑ. Each picture is a visualization in $\mathbb{R}^3$ of g_ϑ for a fixed value of x_4. The surface is the size-changing torus $\vartheta(\boldsymbol{x}) = 0$ (see (23)). Black dots are true troubled points, red dots are false troubled points.

of points evaluated during the algorithm execution is approximately equal to the number of points of a coarse regular grid of 73 knots per dimension but the point distribution is more dense near the discontinuity interface $\vartheta(\boldsymbol{x}) = 0$, instead of being equally distributed in Ω.

In Fig. 14, we report some examples of the troubled points identified by the GINN-based detector for the function g_ϑ. In particular, for fixed values of x_4, we observe that the detector almost perfectly identifies the discontinuity interface, with very few false troubled points.

Table 7
Genetic toggle switch problem: costs (visited points, computational time) and performance (troubled points detected) of Algorithm 2 with $\Lambda_{\min} = 2^{-2}$.

n	Grid pts	$\Lambda_{\min}$	Max Err. (TP)	Troubled pts	Visited pts	Total Time (hh:mm:ss)
4	401	0.5	0.25	50 461	456 523	0:18:15.5372



(a) $K = \bar{K}(1-\sigma)$ (b) $K = \bar{K}$ (c) $K = \bar{K}(1+\sigma)$

Fig. 15. Genetic toggle switch problem: troubled points detected (black dots). In the picture we represent with a colorbar the steady state value v_s as a function of $(\alpha_1, \alpha_2, \eta)$ for three values of K. The red color correspond to $v_s \simeq 0.2$ and the white color to $v_S \simeq 16.6$.

5.3. A realistic 4D problem in a stochastic setting

In this section we address a problem related to a stochastic differential system of equations, introduced in Ref. [46], modeling a genetic toggle switch in Escherichia Coli. The system of equations describe the evolution of the concentration u and v of two repressors; the system depends on some parameters whose values are given in Ref. [46]:

$$\begin{cases} u' = \frac{\alpha_1}{1+v^\beta} - u \\ v' = \frac{\alpha_2}{1+w^\gamma} - v \\ w = \frac{u}{(1+[IPTG]/K)^\eta} \end{cases} \tag{24}$$

In Refs. [5,6,47] these parameters are considered as stochastic parameters, and in the two latter works the problem has been addressed in the framework of discontinuity detection. Indeed, depending on the concentration of an inducer, represented by the parameter [IPTG], the solution v reaches a steady state value v_s with a switch behavior, as the precise value depend in an unclear and nonsmooth way on the parameters. This test problem is therefore useful to test the behavior of the method on a complex, and not *a priori* known, discontinuity pattern.

Refs. [5,6] focus on the four parameters $(\alpha_1, \alpha_2, \eta, K)$, as β and γ do not affect the switching behavior. Accordingly, we applied our method to the steady state v_s of v in a 4D setting, corresponding to the stochastic modelization of parameters $(\alpha_1, \alpha_2, \eta, K)$; for each parameter, a uniform distribution in the interval $[\mu(1-\sigma), \mu(1+\sigma)]$ is considered, where μ is the value originally proposed in Ref. [46] and σ is set to 0.1. We used $[IGPT] = 2 \cdot 10^{-5}$ and mean values $(\bar{\alpha}_1, \bar{\alpha}_2, \bar{\eta}, \bar{K}) = (156.25, 15.6, 2.0015, 2.9618 \cdot 10^{-5})$; the initial concentration was set to $u_0 = 1$ and $v_0 = 1$.

For detecting the discontinuity interface of the steady state v_s with respect to the parameters $(\alpha_1, \alpha_2, \eta, K)$, we rescale the parameter ranges for fitting the 4-dimensional hypercube $[-1, 1]^4$; then we run Algorithm 2 using the 4-dimensional GINN-based detector introduced in the previous subsection and setting $\Lambda_{\min} = 2^{-2}$. The detection starts from one SG centered in the origin and with SG Box with edge length equal to 2, in order to cover the entire hyper-cube. We remark that the discontinuity detection is performed directly on the steady state values, without applying any *a priori* thresholding for obtaining a 4-dimensional step-function (e.g., in Refs. [5,6] the detection is performed with respect to an "on/off" behavior of v_s); therefore, the GINN-based detector reads the v_s values, which are solutions of a differential problem.

Unlike the experiments discussed in the previous subsections, the cost of function evaluation in this case is not negligible, as it involves solving the differential problem (24) for each combination of parameters $(\alpha_1, \alpha_2, \eta, K)$ visited by the GINN-based detector during the detection procedure. In these circumstances, the user can exploit parallelization to accelerate the function evaluations. Nonetheless, in this work we consider a worst-case scenario, where no parallelization is applied and the values of v_s are computed sequentially for each SG and each node. The detection task has been performed on a node of a remote cluster; specifically, the node is endowed with up to 180GB of RAM, 10CPUs used (Intel Xeon E5-2640 v3, 2.60GHz), 1GPU used (GTX980 6G).

The results obtained for the discontinuity detection tasks are illustrated in Table 7 and in Fig. 15. We observe that the computational costs are not negligible (approximately 450 thousands visited points and 18 minutes of total time); nonetheless, the detection algorithm explores successfully the 4-dimensional domain, returning more than 50 thousands of troubled point with good accuracy (see Fig. 15).

5.4. Higher-dimensional cases

In this section, we present an analysis of the proposed GINN-based detection method applied to two n-dimensional test functions, while varying the domain dimension among the values $n = 2, 4, 6, 8$. For $n = 2, 4$ we use the GINN-based detectors already trained for the experiments analysed in the previous section; two additional GINN-based detectors have been trained for $n = 6, 8$ with new synthetic datasets of dimension 6 and 8, respectively, and with the SGG built on the sparse grid $\mathcal{S}_n = \mathcal{I}_{\text{sum}}(4)$. Specifically, $\mathcal{S}_6$ has 1457 nodes and $\mathcal{S}_8$ has 3937 nodes (see Table 1). For training these new detectors, we followed the same strategy and used the same hyperparameters as in the 2-dimensional and 4-dimensional cases, except for a few differences listed in A.5.

The scope of the study conducted in this section is to analyse the behavior of the NN-based detection method as the space dimension n increases, and to propose a heuristic-based strategy to overcome scalability issues. One of the main features of our detection algorithm, is its ability to combine an exhaustive exploration of the domain (see the results on 2-dimensional and 4-dimensional experiments) and a prescribed precision in detecting the discontinuity interfaces (see Theorem 1). This approach can be suitable for a small or moderate dimension, but it becomes prohibitive with larger dimensions, especially if due to the space dimension and to the level $\widehat{h}$ chosen, the underlying sparse grid has a large number of nodes (see Table 1). Indeed, for each current troubled point detected at a fixed detection step, Algorithm 2 explores along all the n dimensions near that point by using the SG made of N nodes, increasing exponentially the number of visited points of the method, where the function is evaluated; obviously, this behavior is mitigated or emphasized depending on the detection precision required $\Lambda_{\min}$. For quantifying these aspects, in analogy with Refs. [6,12], we consider a test problem with a spherical discontinuity interface. for several values of n:

$$g_\Sigma(\boldsymbol{x}) = \begin{cases} 1 & \|\boldsymbol{x}\| \leq 1 \\ 0 & \text{otherwise} \end{cases}, \quad \forall\, \boldsymbol{x} \in \mathbb{R}^n. \tag{25}$$

Computations for these test problems have been performed on nodes of a cluster, each node endowed with up to 180GB of RAM, 10CPUs used (Intel Xeon Silver 4114, 2.20GHz), 1GPU used (Quadro RTX5000 16G). Looking at the values in Table 8 and at Fig. 16a, we clearly observe that the number of visited points increases exponentially with the space dimension n (and the number N of SG nodes); this behavior clearly worsens as $\Lambda_{\min}$ decreases (see Tables 9 and 10 and Fig. 16b). Nonetheless, it is worth noticing that the resources spent by the detection algorithm are exploited to concretely increase the number of troubled points detected (see Fig. 16), proving that the computational effort is well spent on finding new points; the exhaustive detection abilities of this method are also confirmed by the high TPRs, see Tables 8–10.

From this preliminary study, it is clear that the applicability of the method may be limited in high-dimensional settings. Nonetheless, our method is characterized by a scalability (visited points) that is comparable to other strategies available in the literature. Indeed, it is known that if the discontinuity interface exists along all the axial directions, the exponential growth of the number of points necessary to efficiently detect the interface cannot be broken. Such an effect was observed for example by Jakeman et al. in Ref. [6], where they developed a heuristic strategy able to delay the curse of dimensionality when the discontinuity interface solely depends on a small subset of variables. Inspired by this approach, in the next subsection we present a similar strategy based on our detection algorithm and GINN-based detectors.

Table 8
Test function g_Σ: costs (visited points, computational time) and performance (troubled points detected, TPR) of Algorithm 2 with $\Lambda_{\min} = 2^{-1}$ and several values for n.

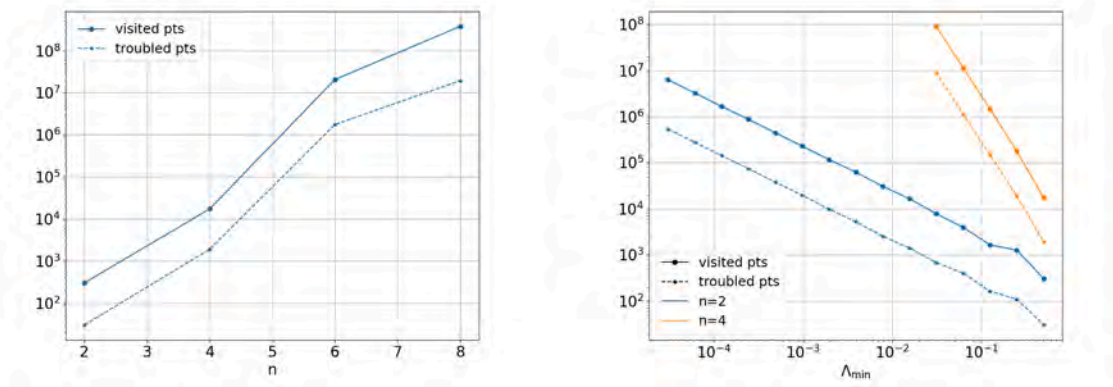
n	Grid pts	$\Lambda_{\min}$	Max Err. (TP)	Troubled pts	TPR	Visited pts	Total Time (hh:mm:ss)
2	65	0.5	0.25	30	100%	301	0:00:00.8517
4	401	0.5	0.25	1 899	98.84%	17 257	0:00:01.5976
6	1457	0.5	0.25	1 750 630	93.02%	20 519 672	0:06:17.9698
8	3937	0.5	0.25	19 293 501	93.53%	378 177 631	4:28:70.2153

Table 9
Test function g_Σ: costs (visited points, computational time) and performance (troubled points detected, TPR) of Algorithm 2 with $n = 2$ and several values for $\Lambda_{\min}$.

$\Lambda_{\min}$	Troubled pts	TPR	Visited pts	Total Time (hh:mm:ss)
0.5	30	100%	301	0:00:00.8517
0.25	109	100%	1 261	0:00:00.8255
0.125	162	100%	1 639	0:00:00.7374
6.25e-02	399	100%	3 941	0:00:00.7975
3.125e-02	679	100%	7 796	0:00:00.8456
7.8125e-03	2 523	100%	30 732	0:00:00.7951
1.953125e-03	9 810	100%	116 588	0:00:00.9588
4.8828125e-04	37 600	100%	441 365	0:00:03.5652
1.220703125e-04	141 386	100%	1 667 475	0:00:39.6897
3.0517578125e-05	531 253	100%	6 263 857	0:24:00.2044

Table 10
Test function g_Σ: costs (visited points, computational time) and performance (troubled points detected, TPR) of Algorithm 2 with $n = 4$ and several values for $\Lambda_{\min}$.

$\Lambda_{\min}$	Troubled pts	TPR	Visited pts	Total Time (hh:mm:ss)
0.5	1 899	98.84%	17 257	0:00:01.5976
0.25	18 835	99.26%	176 779	0:00:01.7794
0.125	145 590	99.39%	1 462 418	0:00:02.9860
6.25e-02	1 115 399	99.37%	11 281 556	0:00:24.5329
3.125e-02	8 932 423	99.46%	90 349 673	0:16:52.7142



(a) Number of visited points and number of troubled points detected versus n, with $\Lambda_{\min} = 2^{-1}$.

(b) Number of visited points and number of troubled points detected versus $\Lambda_{\min}$, with $n = 2, 4$.

Fig. 16. Test function g_Σ: number of points visited by Algorithm 2, varying the domain dimension n and the parameter $\Lambda_{\min}$.

5.4.1. A dimensionally adaptive approach

In cases where the discontinuity interface depends only on a small subset of the domain variables, a heuristic strategy can be defined to reduce the number of visited points visited. The main idea is to identify the m less important dimensions for the discontinuity interface detection in $\mathbb{R}^n$; then, assuming for simplicity that the first dimensions are the more important ones, we consider a suitable number $\kappa \in \mathbb{N}$ of $(n-m)$-dimensional affine subspaces $\Sigma^{(1)}, \dots, \Sigma^{(\kappa)}$, defined as follows: given a fixed $\boldsymbol{x}^{(k)} \in \mathbb{R}^m$, we set

$$\Sigma^{(k)} := \{\boldsymbol{x} \in \mathbb{R}^n \mid (\boldsymbol{x}', \boldsymbol{x}^{(k)}), \boldsymbol{x}' \in \mathbb{R}^{n-m}\}.$$

Once the κ affine subspaces are identified, we perform κ discontinuity detection procedures, one for each $\Sigma^{(k)}$, by using Algorithm 2 with an $(n-m)$-dimensional detector.

In order to identify the m less important dimensions and the κ subspaces of dimension $(n-m)$, we propose here a preliminary heuristic. We perform a single detection step with an n-dimensional detector and a SG covering the entire domain. Let D be the number of troubled points detected at this step; for all the D troubled points, we inspect the number of unique values for each coordinate x_i, $i = 1, \dots, n$; by comparing the number of unique values, we can have a guess about the axial directions that the detector have explored more, and we will identify as the less important dimensions the ones corresponding to the coordinates exhibiting less unique values. After the identification of these dimensions, the subspaces $\Sigma^{(k)}$ are easily recovered by defining the vectors $\boldsymbol{x}^{(k)}$ as the subvectors of the troubled points corresponding to the less important dimensions (thus, $\kappa \leq D$).

We test this heuristic on the discontinuous function

$$g_c(\boldsymbol{x}) = \begin{cases} 1 & \text{if } x_1^2 + x_2^2 \leq 1, \\ 0 & \text{otherwise} \end{cases}, \quad \forall\, \boldsymbol{x} \in \mathbb{R}^n, n \geq 2\,. \tag{26}$$

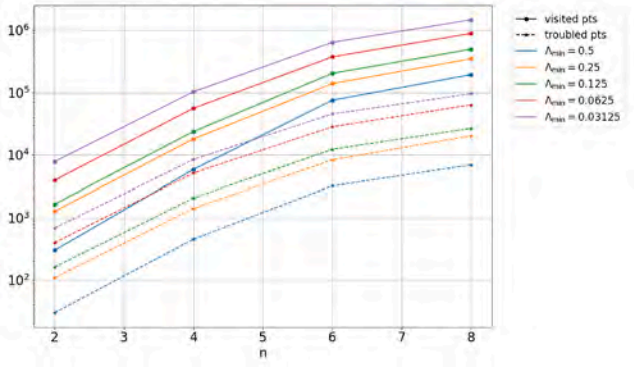
The function g_c describes a circular discontinuity in $\mathbb{R}^2$, that evolves in $\mathbb{R}^n$, $n > 2$, into a cylinder and its corresponding higher-dimensional representations. We defined such a kind of test function in analogy to Ref. [6], where a 3D sphere is used, instead of a 2D circle.

Applying our heuristic to (26), with $n = 2, 4, 6, 8$, we observe the follwing:

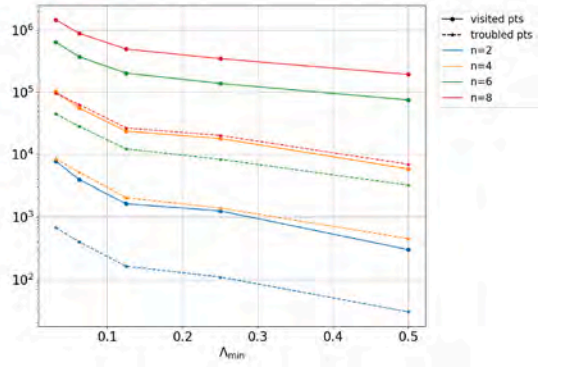
- $n = 2$: the heuristic is obviously not applied and only a 2-dimensional detection is performed (i.e., the problem is equivalent to g_Σ defined in (5.4));
- $n = 4$: the 4-dimensional detector identifies x_1 and x_2 as most important dimensions, by counting 7 unique values for them, and 5 unique values for all the other dimensions. Then, we obtain $\kappa = 13$ subspaces for performing the 2-dimensional detections;
- $n = 6$: the 6-dimensional detector identifies x_1 and x_2 as most important dimensions, by counting 10 and 11 unique values for them, respectively, and 9 unique values for all the other dimensions. Then, we obtain $\kappa = 137$ subspaces for performing the 2-dimensional detections;

Table 11
Test function g_c: costs (visited points, computational time) and performance (troubled points detected, TPR) of Algorithm 2, by varying the domain dimension n and $\Lambda_{\min}$.

n	Grid pts	$\Lambda_{\min}$	Max Err. (TP)	Troubled pts	TPR	Visited pts	Total Time (s)
2	65	0.5	0.25	30	100%	301	0.4233
4	401	0.5	0.25	452	100%	5 852	0.4639
6	1457	0.5	0.25	3 199	100%	75 089	2.2762
8	3937	0.5	0.25	6 939	100%	193 682	6.3637
2	65	0.25	0.125	109	100%	1 261	0.4229
4	401	0.25	0.125	1 405	100%	18 010	0.5553
6	1457	0.25	0.125	8 341	100%	138 759	2.3871
8	3937	0.25	0.125	20 165	100%	347 437	6.4727
2	65	0.125	0.0625	162	100%	1 639	0.4752
4	401	0.125	0.0625	2 052	100%	23 660	0.5523
6	1457	0.125	0.0625	12 276	100%	202 728	2.9826
8	3937	0.125	0.0625	26 622	100%	495 232	7.0214
2	65	0.0625	0.03125	399	100%	3 941	0.4751
4	401	0.0625	0.03125	5 137	100%	55 869	0.5727
6	1457	0.0625	0.03125	28 218	100%	372 773	2.7067
8	3937	0.0625	0.03125	63 340	100%	887 430	7.9578
2	65	0.03125	0.015625	679	100%	7 796	0.5399
4	401	0.03125	0.015625	8 479	100%	102 837	0.6247
6	1457	0.03125	0.015625	45 202	100%	639 204	4.1028
8	3937	0.03125	0.015625	95 452	100%	1 465 391	10.0513



(a) Number of visited points and number of troubled points detected versus n for several $\Lambda_{\min}$ values.



(b) Number of visited points and number of troubled points detected versus $\Lambda_{\min}$ for several n values.

Fig. 17. Test function g_c: number of points visited by Algorithm 2, varying the domain dimension n and the parameter $\Lambda_{\min}$.

- $n = 8$: the 8-dimensional detector identifies x_1 and x_2 as most important dimensions, by counting 11 unique values for them, and 9 unique values for all the other dimensions. Then, we obtain $\kappa = 389$ subspaces for performing the 2-dimensional detections;

The results obtained performing the detections varying both n and the value of $\Lambda_{\min}$, from 2^{-1} to 2^{-5}, are summarized in Table 11 and Fig. 17. From the results, we can observe significant improvements in the behavior of the number of visited points.

The most immediate observation is that, for the same dimensionality and value of $\Lambda_{\min}$ as in the previous case, the number of visited points is smaller, with a reduction of slightly less than one order of magnitude for $n = 4$, which increases to three orders of magnitude for $n = 8$. This is consistent with the further observation that the growth with respect to n is slower, since in Fig. 17a, presented on a logarithmic scale, the curves exhibit a sublinear trend.

Furthermore, looking at Fig. 17b, we notice that the behavior of the number of visited points with respect to $\Lambda_{\min}$ is characterized, for all dimensions considered, by a slope (in the logarithmic scale) very similar to the one of case $n = 2$ for the function g_Σ (see Fig. 16b); this is a direct consequence of the fact that, since two dominant dimensions were preliminarily identified in the structure of the discontinuity, most of the work was carried out by the repeated application of the 2D detector.

A comparison of the computational cost (in terms of visited points) of our approach with that of other methods based on similar concepts in the literature reveals that our method still needs to be improved; for example, the adaptive method presented in Ref. [6] scales linearly with the domain dimension. Nonetheless, our method ensures an exhaustive coverage of the discontinuity interface, thanks to the exploration performed during the initial n-dimensional detection step, which also identifies an appropriate value of κ.

We also note that the number of troubled points has a trend of the same type as the number of visited points, with respect to variations both in n and in $\Lambda_{\min}$; moreover, we observe that in all our tests the TPR is 100% (see Table 11).

It is worth noticing that this dimensionally adaptive version of our approach was able to run for an 8-dimensional detection problem on g_c with $\Lambda_{\min} = 2^{-5}$ within seconds on a Personal Computer (Lenovo ThinkPad P14s Gen 4 laptop, 13th Gen Intel Core i7-1360P, 32GB RAM, NVIDIA RTX A500 4GB VRAM), whereas the 8-dimensional detection on g_Σ with $\Lambda_{\min} = 2^{-1}$ required more than 4 hours on a remote cluster.

5.5. Costs and potentialities

As observed in the experiments above, the proposed NN-based detectors have good potential. In particular, a well-trained NN-based n-dimensional detector can identify with high precision the discontinuity interfaces of generic discontinuous functions with domain in $\mathbb{R}^n$. We recall that the NN (better if a GINN) is trained only once on a synthetic dataset, interestingly showing very good generalization abilities. Moreover, such a kind of models have the advantage of being portable and versatile (see Remark 5.2 below); i.e., they can be shared among different users and integrated also into algorithms different from the ones proposed in this paper. Furthermore, the experiments show that these detectors and algorithms are suitable for the discontinuity detection problem in dimensions higher than 3.

Nevertheless, some drawbacks still exist. The main one is the synthetic dataset creation, based on deterministic detectors; indeed, increasing the problem dimensionality, the cost of this operation becomes particularly expensive. One solution is to parallelize the operation; we adopted this approach for the creation of D_n with $n = 4, 6, 8$, reducing the cost from days to hours. However, we recall that the cost of the dataset creation is paid only once; hence, it is a price worth paying if it returns a good NN-based detector applicable to any discontinuity detection problem with the same domain dimension. Moreover, if a trained detector is already available and publicly shared, any user can use it without the need of any training .

Other limitations of the method concern its applicability to high-dimensional settings and/or very small values of the tolerance parameter $\Lambda_{\min}$. Indeed, for generic discontinuity interfaces extending along all coordinate directions, the cost in terms of visited points increases exponentially, similarly to other approaches reported in the literature. Therefore, following a strategy analogous to those in existing works, we have developed a dimensionality adaptive heuristic that can mitigate the curse of dimensionality when the discontinuity interface depends only on a small subset of variables. Under these circumstances, the proposed method proves to be highly efficient in terms of both computational cost and detection performance.

Remark 5.2 (GINN-based detectors availability)**.** *The trained NNs used in the experiments of Section 5 (dimension $n = 2, 4, 6, 8$), an implementation of Algorithm 2, and practical toy examples are available at https://github.com/Fra0013To/NNbasedDiscDetectors. Users can modify the practical examples available in the repository to detect discontinuities in a custom target function. No training is required for using the NN-based detectors.*

6. Conclusion

Our work presents a novel approach leveraging Neural Networks, and in particular Graph-Instructed Neural Networks (GINNs), to effectively detect discontinuity interfaces of functions with domains of dimension greater than or equal to 2. The proposed method utilizes GINNs trained to detect troubled points within a sparse grid used for function evaluations, where the GINNs are based on the adjacency matrix of a graph constructed on the grid structure. We have introduced a recursive algorithm for general sparse grid-based detectors, characterized by finite termination and convergence properties. Running this algorithm with a well-trained GINN as the detector results in a cost-effective and rapid discontinuity detection method suitable for functions with n-dimensional domains. Our experiments, conducted on functions with dimensions $n = 2, 4, 6, 8$, demonstrate the efficiency and notable generalization abilities of the GINNs in detecting discontinuity interfaces, even for functions different from those used for building the training sets. The portable and versatile nature of the trained GINNs allows them to be shared among users and integrated into new algorithms. While our proposed NN-based detectors exhibit a good potential, some limitations persist. Creating synthetic datasets based on deterministic detectors can be costly, particularly as problem dimensionality increases. We have addressed this by parallelizing the dataset creation process.

In summary, our method offers a powerful tool for discontinuity detection, demonstrating its potential, especially for dimensions higher than 3. Further research in dataset creation efficiency, optimal NN hyperparameters search, and algorithm improvements will permit us to develop even more efficient software for discontinuity detection techniques, expanding the applicability of this approach to a wider range of practical problems and, in particular, to even higher dimensions.

Code availability

The code for loading the trained NN-based detectors illustrated in this paper, for an implementation of Algorithm 2, and for running practical toy examples is available at https://github.com/Fra0013To/NNbasedDiscDetectors. See also Remark 5.2.

Data availability

We have shared the link to code in the manuscript.

Acknowledgements

The authors acknowledge that this study was carried out within the FAIR-Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR)-MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3—D.D. 1555 11/10/2022, PE00000013). This manuscript reflects only the authors' views and opinions; neither the European Union nor the European Commission can be considered responsible for them. Sandra Pieraccini acknowledges the support of the FaReX project ("Full and Reduced order modelling of coupled systems: focus on non-matching methods and automatic learning"), PRIN 2022 program (CUP: E53D23005510006) funded by European Union - Next Generation EU, Missione 4 Componente 1.

Appendix A. Practical details for building NN-based detectors

In this appendix, we report the details of the procedure used for building the NN-based detectors. We start illustrating how to build a deterministic approximation of an exact detector Δ^* for discontinuous functions with discontinuities contained in the zero-level set of known continuous functions (A.1). Then, we introduce the random piecewise continuous functions used for generating the synthetic data (A.2), the preprocessing function for NN inputs (A.3), the architecture archetype used for the NN models (A.4), and some details about the detectors trained for $n = 6$ and $n = 8$.

A.1. Zero-level set detection and approximation of exact detectors

Algorithm 1 can be easily extended to other tasks if we use detectors focused on identifying other kind of troubled points. For example, we can consider detectors built for detecting points that are near to the zero-level set of a continuous function. In these cases, Definitions 3.1–3.3 can be reformulated considering a continuous function $f : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ and its zero-level set $\zeta := \{\boldsymbol{x} \in \Omega \mid f(\boldsymbol{x}) = 0\}$, in place of a discontinuous function g and its set of discontinuity points δ, respectively. From now on, we let Z denote the zero-level set detectors for such a kind of extension of Algorithm 1, and Z^* denotes the exact ones.

Let $\mathcal{C}$ denote the set of continuous functions $f : \Omega \to \mathbb{R}$, for any $\Omega \subseteq \mathbb{R}^n$. For a fixed sparse grid $\mathcal{S}$, we recall from Definition 3.2 that $\mathcal{S}_{/\simeq}$ denotes the set of all the sparse grids similar to $\mathcal{S}$. We can define a deterministic zero-level detector $Z^{(t+1)} : \mathcal{C} \times \mathcal{S}_{/\simeq} \to \{0,1\}^N$, fixed $t \in \mathbb{N}$, $t \geq 2$, described by the following algorithm:

1. Let G' be the SGG based on $\mathcal{S}' \simeq \mathcal{S}$, $\mathbb{B}(\mathcal{S}') \subset \Omega$. Then, for each edge $\{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E'$, we compute the equispaced knots $\boldsymbol{x}_{ij}^{(\tau)} := \boldsymbol{x}_i + \frac{\tau}{t}(\boldsymbol{x}_j - \boldsymbol{x}_i)$, $\tau = 0, \dots, t$;
2. For each edge $\{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E'$, we compute $s(\boldsymbol{x}_{ij}^{(\tau)}) := \text{sign}(f(\boldsymbol{x}_{ij}^{(\tau)}))$, $\tau = 0, \dots, t$;
3. For each edge $\{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E'$, $\boldsymbol{x}_i$ is detected as troubled point ($p_i = 1$) if $s(\boldsymbol{x}_i) = 0$ or exist $\tau \in \{1, \dots, \lceil t/2 \rceil\}$ such that $s(\boldsymbol{x}_i) \neq s(\boldsymbol{x}_{ij}^{(\tau)})$. Analogously, $\boldsymbol{x}_j$ is detected as troubled point if $s(\boldsymbol{x}_j) = 0$ or exist $\tau \in \{\lfloor t/2 \rfloor, \dots t-1\}$ such that $s(\boldsymbol{x}_j) \neq s(\boldsymbol{x}_{ij}^{(\tau)})$.

Proposition A.1 ($Z^{(t+1)}$ approximates Z^*)**.** *The detector $Z^{(t+1)}$ can be considered an approximation of the exact detector Z^*, i.e.:*

$$\lim_{t \to +\infty} Z^{(t+1)}(\boldsymbol{x}) = Z^*(\boldsymbol{x}), \tag{A.1}$$

for each $\boldsymbol{x} \in \mathcal{S}'$ and each $\mathcal{S}' \simeq \mathcal{S}$.

Proof. Let $e_{ij} = \{\boldsymbol{x}_i, \boldsymbol{x}_j\} \in E'$ be an edge intersecting ζ and let $\boldsymbol{x}_i^\zeta \in \zeta \cap e_{ij}$ be the nearest point to $\boldsymbol{x}_i$ such that $\|\boldsymbol{x}_i - \boldsymbol{x}_i^\zeta\| \leq \|\boldsymbol{x}_i^\zeta - \boldsymbol{x}_j\|$; i.e., by definition, $\boldsymbol{x}_i$ is a troubled point with respect to the zero-level set detection problem (generalization of Definition 3.1).

Then, there exists $\tau^\zeta \in [0, \|\boldsymbol{x}_i - \boldsymbol{x}_j\|/2]$ such that

$$\boldsymbol{x}_i^\zeta = \boldsymbol{x}_i + \tau^\zeta \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|}.$$

Now, let $\hat{\boldsymbol{x}}_i$ be such that

$$\hat{\boldsymbol{x}}_i = \boldsymbol{x}_i + \frac{\hat{\tau}}{t}(\boldsymbol{x}_j - \boldsymbol{x}_i),$$

with $t \in \mathbb{N}$ fixed, $\hat{\tau} \in \{1, \dots, \lceil t/2 \rceil\}$, and

$$\hat{\tau} = arg\, min_{\tau \geq \tau^\zeta} \tau - \tau^\zeta .$$

Since f is continuous, for each t sufficiently large, we have that this $\hat{\tau}$ exists such that $\hat{\boldsymbol{x}}_i$ is the first vector of the sequence $\{\boldsymbol{x}_{ij}^{(\tau)}\}_{\tau=1,\dots,t}$ where the sign of f is not equal to $\text{sign}(f(\boldsymbol{x}_i))$; i.e., $\boldsymbol{x}_i$ is a troubled point for $Z^{(t+1)}$, detected thanks to $\hat{\boldsymbol{x}}_i$.

Let $h = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|/t$; then, by construction, we also have that $\|\boldsymbol{x}_i^\zeta - \hat{\boldsymbol{x}}_i\| < h$. Therefore, it holds

$$\lim_{t \to +\infty} \|\boldsymbol{x}_i^\zeta - \hat{\boldsymbol{x}}_i\| = 0,$$

and the troubled points detected by $Z^{(t+1)}$ tend to coincide with the true troubled points detected by Z^*, when $t \to +\infty$.

□

The zero-level set extension of the algorithm is particularly useful for building a good approximation of the troubled points returned by an exact discontinuity detector Δ^* for discontinuous functions with discontinuity points contained in the zero-level set of a continuous function; e.g., for a function $g : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$ such that

$$g(\boldsymbol{x}) = \begin{cases} g_1(\boldsymbol{x}), & \text{if } f(\boldsymbol{x}) \geq 0 \\ g_2(\boldsymbol{x}), & \text{otherwise} \end{cases}, \tag{A.2}$$

where g_1, g_2, and f are continuous functions in Ω.

Indeed, for a function as in (A.2), we have that $\delta \subseteq \zeta$; then, assuming f is known, it holds that the set of troubled points detected by an exact discontinuity detector Δ^* is contained in the set of troubled points detected by an exact zero-level set detector Z^*. Therefore, we can use a detector $Z^{(t+1)}$ to approximate Z^* and, as a consequence, to approximate the troubled points returned by Δ^*.

A.2. Synthetic dataset creation

At item 2 of the procedure in Section 4.1, we generate a set of random piecewise continuous functions $g^{(q)} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, $q = 1, \dots, Q$. In our numerical experiments (Section 5), we generate a dataset from Legendre polynomial piecewise continuous functions characterized by one linear, spherical (see [3]), or polynomial discontinuity interface (see [21]). Indeed, similarly to the convolutional detectors in [3], we observe that our NN-based detectors show good generalization abilities in detecting discontinuities of different nature, even if trained only on these three limited cases (see Section 5).

Let P_n denote the Legendre polynomial of degree n; then, the random discontinuous functions in $\mathcal{G}$ we consider are piecewise smooth functions $g : [-1, 1]^n \to \mathbb{R}$ defined as (A.2), where:

- the functions $g_1, g_2 : [-1, 1]^n \to \mathbb{R}$ are defined as

$$g_j(\boldsymbol{x}) = \sum_{\boldsymbol{h} \in \mathcal{I}_{\text{sum}}(4)} \left(c_{\boldsymbol{h}} \prod_{i=1}^{n} P_{h_i}\left(\frac{x_i + 1}{2}\right)\right), \quad \forall\, j = 1, 2, \tag{A.3}$$

  with $\boldsymbol{h} = (h_1, \dots, h_n) \in \mathbb{N}^n$ multi-index and $c_{\boldsymbol{h}}$ random coefficients sampled from the normal distribution $\mathcal{N}(0, 10)$;
- the function f characterizing the discontinuity interface of g is also generated randomly. We consider three types of interfaces, labeled with specific symbols.
  (a) *Linear cut:*

$$\eta(\boldsymbol{x}) = \boldsymbol{x}^T \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} + b,$$

  where $w_i \in \mathcal{N}(0, 1)$ and $b \in \mathcal{U}(-1, 1)$.
  (b) *Spherical cut:*

$$\sigma(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{c}\| - r,$$

  where $c_i \in \mathcal{U}(-1, 1)$ and $r = \min\{0.2, \rho\}$, with $\rho \in \mathcal{U}(0, \sqrt{n})$.
  (c) *Polynomial cut:*

$$\pi(\boldsymbol{x}) = C \frac{\Pi(\boldsymbol{\xi})}{\max_{\boldsymbol{y} \in [-1,1]^{n-1}} |\Pi(\boldsymbol{y})|} - x_\iota,$$

  where $C \in \mathcal{U}(0.75, 1.15)$, $\iota \in \mathcal{U}(\{1, \dots, n\})$, $\boldsymbol{\xi} := (x_1, \dots, x_{\iota-1}, x_{\iota+1}, \dots, x_n) \in \mathbb{R}^{n-1}$, and

$$\Pi(\boldsymbol{\xi}) = \sum_{\boldsymbol{h} \in \mathcal{I}_{\text{sum}}(4)} \left(c_{\boldsymbol{h}} \prod_{i=1, i \neq \iota}^{n} P_{h_i}\left(\frac{x_i + 1}{2}\right)\right),$$

  with $\boldsymbol{h} = (h_1, \dots, h_{\iota-1}, h_{\iota+1}, \dots, h_n) \in \mathbb{N}^{n-1}$ multi-index and $c_{\boldsymbol{h}} \in \mathcal{N}(0, 10)$.

  Namely, π is the polynomial Π rescaled to a maximum absolute value equal to C, minus x_ι. See Fig. 4 and Section 5 for 2D examples of such a kind of discontinuity interface.

A.3. Preprocessing of input data

At step 6 of the procedure in Section 4.1, we train the NN on the synthetic data generated. We observe that the inputs $\boldsymbol{g}'$ have a range of values that depends on the random generation of the discontinuous functions in $\mathcal{G}$ (on which we have a limited control); then, a preprocessing of the input data is necessary in order to make the NN-based detector applicable to any discontinuous function g (independently on the values it assumes on the sparse grid points).

Typically, NN inputs are standardized (i.e., centered in zero and with normalized standard deviation) but, for our task, we use a different preprocessing function. Indeed the standardization must be performed with respect to the mean and standard deviation vectors of the training data; therefore, it is not suitable for discontinuous functions particularly different from the ones randomly generated at items 2–4.

Let $\gamma : \mathbb{R}^n \to \mathbb{R}^n$ denote the function representing the preprocessing operation of the input data. Naive but efficient preprocessing functions for our needs are the rescaling functions, because they permit to "feed" the NN with vectors having all the values into a chosen range $[\alpha, \beta]$. In particular, we use the following function:

$$\gamma(\boldsymbol{g}') = \begin{cases} \boldsymbol{e}, & \text{if } g'_{\max} = g'_{\min} = 0 \\ \boldsymbol{g}' / \max\{|g'_{\max}|, |g'_{\min}|\}), & \text{otherwise} \end{cases}, \tag{A.4}$$

where $\boldsymbol{e} = (1, \dots, 1) \in \mathbb{R}^N$, such that the elements of $\gamma(\boldsymbol{g}')$ are always between -1 and 1, and where $g'_{\min} := \min\{g(\boldsymbol{x}') \,|\, \boldsymbol{x}' \in S'\})$, $g'_{\max} := \max\{g(\boldsymbol{x}') \,|\, \boldsymbol{x}' \in S'\})$. We prefer (A.4) than a simplest rescaling function with values between 0 and 1, defined by

$$\gamma(\boldsymbol{g}') = \begin{cases} \boldsymbol{0}, & \text{if } g'_{\max} = g'_{\min} \\ (\boldsymbol{g}' - g'_{\min}\, \boldsymbol{e})/(g'_{\max} - g'_{\min}), & \text{otherwise} \end{cases}, \tag{A.5}$$

because (A.5) always transforms almost-constant vectors into vectors that are not almost-constant. Some preliminary investigations confirmed the effectiveness of our choice, since NNs trained using (A.4) showed better performances than NNs trained using (A.5) (in particular, on piecewise constant functions).

A.4. Architecture archetypes

For both MLPs and GINNs, we use a fixed architecture archetype, with characteristics depending on the SGG. We postpone to future work experiments based on different architectures and/or hyperparameters.

The architecture archetype we use is based on residual blocks [44] and exploits also the batch-normalization [48]. The number of units per layer is N (i.e., the number of sparse grid nodes) while the depth is characterized by the diameter of the SGG; this depth characterization has been chosen mainly thinking to GINN models (see [28, Proposition 1]), but it is a good choice also for MLPs, since residual blocks permit to better exploit the model depth (see [44]).

Let S be a sparse grid in $\mathbb{R}^n$ of N points and let A be the adjacency matrix of G, the WSGG based on S. The architecture archetype we consider (see Fig. A.18) is defined by:

- One input layer I of N units;
- One hidden GI/FC layer L_1 of N units and activation function ψ. If it is a GI layer, it is based on A and it is characterized by $F \geq 1$ output features per node;
- One Batch-normalization layer B_1;
- $\lfloor \text{diam}(G)/2 \rfloor$ residual block. In particular, for $h = 1, \dots, \lfloor \text{diam}(G)/2 \rfloor$, the residual block R_{h+1} is given by:
  - One hidden GI/FC layer L'_{h+1} of N units and activation function ψ. If it is a GI layer, it is based on A and it is characterized by $F \geq 1$ input and output features per node;
  - One Batch-normalization layer B'_{h+1};
  - One hidden GI/FC layer L''_{h+1} of N units and *linear* activation function. If it is a GI layer, it is based on A and it is characterized by $F \geq 1$ input and output features per node;
  - One layer Σ_{h+1} that returns the sum of the outputs of L''_{h+1} and Σ_h ($\Sigma_1 \equiv L_1$ by convention), applying to it the activation function ψ.
  - One Batch-normalization layer B''_{h+1};
- One GI/FC layer L_{fin} of N units and *sigmoid* activation function. If it is a GI layer, it is based on A and it is characterized by $F \geq 1$ output features per node; moreover, it is also endowed with a pooling operation, that aggregates the F output features per node into one (with values still in $[0, 1]$).

A.5. GINN models for high dimensions

Given $\mathcal{D}_n$, $n = 6, 8$, we build two GINN-based detectors and we train them with the following differences in the training options (with respect to the cases $n = 2, 4$):

- Dataset cardinality: The datasets $\mathcal{G}_n$ are obtained by running the zero-level detection algorithm with respect to $Q = 450$ randomly generated piecewise continuous functions (linear, spherical, or polynomial cuts, 150 each), $Z^{(3)}$ detector; the detection procedure have been distributed among multiple nodes of a remote cluster, in order to parallelize and fasten the dataset creation. Contrary to $n = 2, 4$, the number of troubled points detected for each SG is highly variable (also due to the larger size of the grid) and the quantity of SGs without troubled points detected is still large but not the predominant one. For this reason, in order to have a more balanced dataset $\mathcal{D}_n$, we reorganized the samples in subsets $\mathcal{D}_n^{(i)}$, each one containing the samples for which exactly i troubled points have been detected, for $i = 0, \dots, N$. Then, we randomly picked $s \in \mathbb{N}$ samples from each $\mathcal{D}_n^{(i)}$, with $s = 32, 30$ for $n = 6, 8$, respectively (if $|\mathcal{D}_n^{(i)}| < s$, we kept all the elements of $\mathcal{D}_n^{(i)}$). This process allowed to build a dataset $\mathcal{D}_n$ with cardinality in the order of the tens of thousands, but not large enough to run out of memory during the training.
- Dataset split: the test set cardinality $|\mathcal{P}_n|$ is 30% of $|\mathcal{D}_n|$, the validation set cardinality $|\mathcal{V}_n|$ is 10% of $|\mathcal{D}_n|$, and the training set cardinality is $|\mathcal{T}_n| = |\mathcal{D}_n| - |\mathcal{P}_n| - |\mathcal{V}_n|$;
- Mini-batch size: 16;

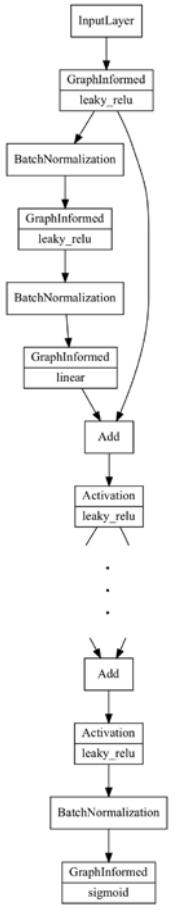


Fig. A.18. Example of architecture archetype described in Appendix A.4 for a GINN model, written in Keras (Tensorflow) [27,49]. In this example, the activation function used is the *leaky relu* function.

- Learning rate decay: reduce on plateau [40] (factor 0.75, 8 epochs of patience);
- Regularization: early-stopping method with best-weights restoration [41] (32 epochs of patience).

Concerning the architecture, the number of filters is $F = 8$.

References

[1] J.D. Jakeman, A. Narayan, D. Xiu, Minimal multi-element stochastic collocation for uncertainty quantification of discontinuous functions, J. Comput. Phys. 242 (2013) 790–808. https://doi.org/https://doi.org/10.1016/j.jcp.2013.02.035
[2] G. Jaeger, The ehrenfest classification of phase transitions: introduction and evolution, Arch. Hist. Exact Sci. 53 (1998) 51–81. https://doi.org/10.1007/s004070050021
[3] S. Wang, Z. Zhou, L.-B. Chang, D. Xiu, Construction of discontinuity detectors using convolutional neural networks, J. Sci. Comput. 91 (2) (2022) 40. https://doi.org/10.1007/s10915-022-01804-z
[4] R. Archibald, A. Gelb, J. Yoon, Polynomial fitting for edge detection in irregularly sampled signals and images, SIAM J. Numer. Anal. 43 (1) (2005) 259–279. https://doi.org/10.1137/S0036142903435259
[5] R. Archibald, A. Gelb, R. Saxena, D. Xiu, Discontinuity detection in multivariate space for stochastic simulations, J. Comput. Phys. 228 (7) (2009) 2676–2689. https://doi.org/10.1016/j.jcp.2009.01.001
[6] J.D. Jakeman, R. Archibald, D. Xiu, Characterization of discontinuities in high-dimensional stochastic problems on adaptive sparse grids, J. Comput. Phys. 230 (10) (2011) 3977–3997. https://doi.org/10.1016/j.jcp.2011.02.022
[7] S.A. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions„ Dokl. Akad. Nauk 4 (1963) 240–243.
[8] H.-J. Bungartz, M. Griebel, Sparse grids, Acta Numerica 13 (2004) 147-269. https://doi.org/10.1017/S0962492904000182
[9] C. Piazzola, L. Tamellini, Algorithm 1040: the sparse grids matlab kit - a matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification, ACM Trans. Math. Software 50 (2024) 1–22. https://doi.org/10.1145/3630023
[10] X. Ma, N. Zabaras, An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations, J. Comput. Phys. 228 (8) (2009) 3084–3113. https://www.sciencedirect.com/science/article/pii/S002199910900028X. https://doi.org/https://doi.org/10.1016/j.jcp.2009.01.006
[11] D. Pflüger, B. Peherstorfer, H.J. Bungartz, Spatially adaptive sparse grids for high-dimensional data-driven problems, in: J. Complexity, 26, Academic Press Inc., 2010, pp. 508–522. https://doi.org/10.1016/j.jco.2010.04.001
[12] G. Zhang, C.G. Webster, M. Gunzbuerger, J. Burkardt, Hyperspherical sparse approximation techniques for high-dimensional discontinuity detection, SIAM Rev. 58 (3) (2016) 517–551.
[13] M. Bozzini, F. De Tisi, M. Rossini, Irregularity Detection from Noisy Data with Wavelets, AK Peters/CRC Press, 1994, pp. 75–82. .

[14] V. Suresh, S.K. Rao, G. Thiagarajan, R.P. Das, Denoising and detecting discontinuities using wavelets, Indian J. Sci. Technol. 9 (2016). https://doi.org/10.17485/ijst/2016/v9i19/85440
[15] R. Jain, R. Kasturi, B.G. Schunck, Machine Vision, McGraw-Hill, 1995.
[16] M. Wei, A.R. De Pierro, J. Yin, Iterative methods based on polynomial interpolation filters to detect discontinuities and recover point values from Fourier data, IEEE Trans. Signal Process. 53 (2005) 136–146. https://doi.org/10.1109/TSP.2004.838936
[17] Z. Gao, X. Wen, W.S. Don, Enhanced robustness of the hybrid compact-WENO finite difference scheme for hyperbolic conservation laws with multi-resolution analysis and Tukey's boxplot method, J. Sci. Comput. 73 (2017) 736–752. https://doi.org/10.1007/s10915-017-0465-0
[18] M.J. Vuik, J.K. Ryan, Automated parameters for troubled-cell indicators using outlier detection, SIAM J. Scient. Comput. 38 (2016) A84–A104. https://doi.org/10.1137/15M1018393
[19] D. Ray, J.S. Hesthaven, An artificial neural network as a troubled-cell indicator, J. Comput. Phys. 367 (2018) 166–191. https://doi.org/10.1016/j.jcp.2018.04.029
[20] C. Bracco, F. Calabrò, C. Giannelli, Discontinuity detection by null rules for adaptive surface reconstruction, J. Sci. Comput. 97 (2023). https://doi.org/10.1007/s10915-023-02348-6
[21] F. Della Santa, S. Pieraccini, Discontinuous neural networks and discontinuity learning, J. Comput. Appl. Math. 419 (2023) 114678. https://www.sciencedirect.com/science/article/pii/S0377042722003430. https://doi.org/https://doi.org/10.1016/j.cam.2022.114678
[22] M.A. El-Sayed, M.A. Khafagy, Automated edge detection using convolutional neural network, IJACSA-Int. J. Adv. Comput. Sci. Appl. 4 (2013).
[23] Y. Liu, M.-M. Cheng, X. Hu, J.-W. Bian, L. Zhang, X. Bai, J. Tang, Richer convolutional features for edge detection, IEEE Trans. Pattern Anal. Mach. Intell. 41 (8) (2019) 1939–1946. https://doi.org/10.1109/TPAMI.2018.2878849
[24] R. Wang, Edge Detection Using Convolutional Neural Network, Springer, Cham, 2016, pp. 12–20. .
[25] C. Wen, P. Liu, W. Ma, Z. Jian, C. Lv, J. Hong, X. Shi, Edge detection with feature re-extraction deep convolutional neural network, J. Vis. Commun. Image Represent. 57 (2018) 84–90. https://doi.org/10.1016/j.jvcir.2018.10.017
[26] C. Xue, J. Zhang, J. Xing, Y. Lei, Y. Sun, Research on edge detection operator of a convolutional neural network, IEEE, 2019, pp. 49–53. https://doi.org/10.1109/ITAIC.2019.8785855
[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015, Software available from tensorflow.org, https://www.tensorflow.org/.
[28] S. Berrone, F. Della Santa, A. Mastropietro, S. Pieraccini, F. Vaccarino, Graph-informed neural networks for regressions on graph-structured data, Mathematics 10 (5) (2022) 786. https://doi.org/10.3390/math10050786
[29] F. Della Santa, A. Mastropietro, S. Pieraccini, F. Vaccarino, Edge-wise graph-instructed neural networks, J. Comput. Sci. 85 (2025) 102518. https://linkinghub.elsevier.com/retrieve/pii/S1877750324003119. https://doi.org/10.1016/j.jocs.2024.102518
[30] V. Barthelmann, E. Novak, K. Ritter, High dimensional polynomial interpolation on sparse grids, Adv. Comput. Math. 12 (2000) 273–288. https://doi.org/10.1023/A:1018977404843
[31] I. Babuska, F. Nobile, R. Tempone, A stochastic collocation method for elliptic partial differential equations with random input data, SIAM J. Numer. Anal. 45 (3) (2007) 1005–1034. https://doi.org/10.1137/050645142.
[32] F. Nobile, R. Tempone, C.G. Webster, A sparse grid stochastic collocation method for partial differential equations with random input data, SIAM J. Numer. Anal. 46 (5) (2008) 2309–2345. https://doi.org/10.1137/060663660. https://doi.org/10.1137/060663660https://doi.org/10.1137/060663660
[33] J. Bäck, F. Nobile, L. Tamellini, R. Tempone, Stochastic spectral galerkin and collocation methods for PDEs with random coefficients: a numerical comparison, in: J.S. Hesthaven, E.M. Rønquist (Eds.), Spectral and High Order Methods for Partial Differential Equations, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 43–62.
[34] TensorFlow, Binary Cross-Entropy Loss - TensorFlow Losses, 2024, (Accessed on October 2024), https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy.
[35] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. https://www.deeplearningbook.org.
[36] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P.S. Yu, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. 32 (1) (2021) 4–24. https://doi.org/10.1109/TNNLS.2020.2978386
[37] E.J. Hall, S. Taverniers, M.A. Katsoulakis, D.M. Tartakovsky, GINNs: graph-informed neural networks for multiscale physics, J. Comput. Phys. 433 (2021) 110192. https://www.sciencedirect.com/science/article/pii/S0021999121000875. https://doi.org/https://doi.org/10.1016/j.jcp.2021.110192
[38] F.D. Santa, Sparse Implementation of Versatile Graph-Informed Layers, 2024, https://arxiv.org/abs/2403.13781.
[39] D.P. Kingma, J.L. Ba, Adam: a method for stochastic optimization, 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings (2015) 1–15. 1412.6980
[40] TensorFlow, Reduce Learning Rate on Plateau - TensorFlow Callbacks, 2023, (Accessed on October 2023), https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau.
[41] TensorFlow, Early Stopping - TensorFlow Callbacks, 2023, (Accessed on October 2023), https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping.
[42] A. Apicella, F. Donnarumma, F. Isgrò, R. Prevete, A survey on modern trainable activation functions, Neural Networks 138 (2021) 14–32. 2005.00817https://doi.org/10.1016/j.neunet.2021.01.026
[43] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, J. Mach. Learn. Res. 9 (2010) 249–256.
[44] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Patt. Recogn. 2016-Decem (2016) 770–778. 1512.03385https://doi.org/10.1109/CVPR.2016.90
[45] L.A. Shepp, B.F. Logan, The Fourier reconstruction of a head section, IEEE Trans. Nucl. Sci. 21 (3) (1974) 21–43. https://doi.org/10.1109/TNS.1974.6499235
[46] T.S. Gardner, C.R. Cantor, J.J. Collins, Construction of a genetic toggle switch in Escherichia coli, Nature 403 (2000) 339–342. https://doi.org/10.1038/35002131
[47] D. Xiu, Efficient collocational approach for parametric uncertainty analysis, Commun. Comput. Phys. 2 (2007) 293–309. http://www.global-sci.com/.
[48] S. Ioffe, C. Szegedy, Batch normalization: accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, p. 448-456.
[49] F. Chollet, et al., Keras, 2015, (https://keras.io).