Gradient-informed neural networks: Embedding prior beliefs for learning in low-data scenarios

Neural Networks 199 (2026) 108681

Contents lists available at ScienceDirect

Neural Networks

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

Full Length Article

Gradient-informed neural networks: Embedding prior beliefs for learning in low-data scenarios

Filippo Aglietti [a,b,*], Francesco Della Santa [c,d], Andrea Piano [a], Virginia Aglietti [e]

[a] Energy Department, Politecnico di Torino, Corso Duca degli Abruzzi 24, Turin, 10129, Turin, Italy
[b] Dumarey Automotive Italia S.p.A., Corso Castelfidardo 36, Turin, 10129, Italy
[c] Department of Mathematical Sciences, Politecnico di Torino, Corso Duca degli Abruzzi 24, Turin, 10129, Turin, Italy
[d] Gruppo Nazionale per il Calcolo Scientifico, Istituto Nazionale di Alta Matematica, Piazzale Aldo Moro 5, Rome, 00185, Rome, Italy
[e] Independent Researcher, Italy

ARTICLE INFO

2020 MSC:
65D15
68T07

Keywords:
Deep learning
Neural networks
Low-data regime
Prior belief

ABSTRACT

We propose Gradient-Informed Neural Networks (GradINNs), a methodology that can be used to efficiently approximate a wide range of functions in low-data regimes, when only general prior beliefs are available, a condition that is often encountered in complex engineering problems.

GradINNs incorporate prior beliefs about the first-order derivatives of the target function to constrain the behavior of its gradient, thus implicitly shaping it, without requiring explicit access to the target function's derivatives. This is achieved by using two Neural Networks: one modeling the target function and a second, auxiliary network expressing the prior beliefs about the first-order derivatives (e.g., smoothness, oscillations, etc.). A customized loss function enables the training of the first network while enforcing gradient constraints derived from the auxiliary network; at the same time, it allows these constraints to be relaxed in accordance with the training data. Numerical experiments demonstrate the advantages of GradINNs, particularly in low-data regimes, with results showing strong performance compared to standard Neural Networks across the tested scenarios, including synthetic benchmark functions and real-world engineering tasks.

1. Introduction

In the field of computational physics, Neural Networks (NNs) have become an increasingly important tool for modeling complex physical systems that cannot be derived in closed form or for which the traditional empirical models fail to achieve the desired accuracy (Choi et al., 2020; Oreski, 2012). Several studies have shown how NNs are powerful function approximators, able to model a wide variety of large, complex, and highly non-linear systems with unprecedented computational efficiency when a large training dataset is available (Hornik et al., 1989; Ishikawa et al., 2023; Pinkus, 1999; Saha et al., 2023). However, in settings where data is limited or widely dispersed, NNs face considerable difficulties. For instance, in the physical sciences, data is often obtained experimentally and is thus expensive and challenging to collect. In such scenarios, NNs show a decreasing prediction performance and a higher probability to overfit the training data compared to the white/gray physics models. On the other hand, the latter can suffer from lack of flexibility and expressivity or can require high computational effort as Finite Element Method models (Reddy, 2013). In order to address these difficulties, Physics-Informed Neural Networks (PINNs) emerged in recent years (Raissi et al., 2019). These models leverage prior knowledge, often in the form of known differential equations (DE), by embedding it directly into the training process. Specifically, they introduce an additional term into the loss function that represents the residual of the underlying DE evaluated on a set collocation points in the input domain where the physics constraints is enforced. This formulation increases robustness against flawed data, e.g., missing or noisy values, and offers physically consistent predictions, particularly in tasks requiring extrapolation (Karniadakis et al., 2021). Although PINNs have been shown to perform well in a variety of applications (Noguer i Alonso & Maxwell, 2023; Cai et al., 2021a,b; Huang & Wang, 2023; Karniadakis et al., 2021; Mao et al., 2020; Raissi et al., 2019; Yu et al., 2022) they require detailed prior knowledge of the governing DE and are thus not directly applicable when this is not available. Recently, PINNs have been extended to deal with various modified settings: systems characterized by partially unknown underlying DE (Podina et al., 2023; Raissi, 2018), unknown data measurement noise (Pilar & Wahlström, 2023), and learning symplectic gradients (Greydanus et al., 2019; Mattheakis et al., 2022).

Another approach aimed at improving the performance of NNs is the Sobolev Training framework (Czarnecki et al., 2017), which enhances

* Corresponding author.
E-mail address: filippo_aglietti@yahoo.it;filippo.aglietti@dumarey.com (F. Aglietti).

https://doi.org/10.1016/j.neunet.2026.108681
Received 2 May 2025; Received in revised form 3 December 2025; Accepted 1 February 2026
Available online 2 February 2026

learning by introducing gradient-based loss terms that constrain the value of the NN derivatives. In its original formulation, Sobolev Training relies on the explicit knowledge of the true derivatives of the target function, using them as direct supervision during training. This approach has also been combined with PINNs (Son et al., 2021), where the same principle is applied to the residuals of the governing DEs.

However, there exist many physical phenomena where the underlying physics is almost entirely unknown or too complex to be easily represented through DEs, and where the true derivatives of the target function are not accessible. Such conditions make Sobolev training or Physics-Informed formulations inapplicable. In these cases, the only available information that characterizes the target function often consists of sparse measurements (i.e., data) and hypotheses about its general behavior (i.e., the so-called *prior beliefs*). In this scenarios, the typical approach is to perform a regression task to infer the function from data. In particular, when large amounts of data are available but minimal knowledge exists about the behavior of the target function, general-purpose NNs are typically employed (e.g., see Berrone et al. (2021b), Ferrara et al. (2021), Jagannath and Jagannath (2021), Kumar and Dhua (2025)). However, in many engineering and physical applications, the access to data is often limited; for example data collection may require time-consuming experiments or costly high-fidelity simulations. We refer to these situations as low-data regimes, i.e., when the number of training samples can be considered relatively small if compared to the dimensionality of the feature space and/or with respect to the complexity of the problem to be learned. In these cases, the available data are too few to fully capture the system's behavior via NNs, typically resulting in poor predictive performance due to underfitting or overfitting phenomena.

To address these settings, we propose Gradient-Informed Neural Network (GradINN). GradINN is a novel type of NN specifically designed to incorporate prior belief about the system's behavior through an auxiliary NN that constrain the primary NN's gradient. The prior beliefs represent general hypotheses about the behavior of the function's gradients, such as, for instance, smoothness or oscillatory patterns. These hypotheses typically originate from domain knowledge about the phenomenon under investigation. Rather than relying on direct gradient observations, they reflect qualitative properties that the target function is expected to exhibit arising from theoretical understanding, empirical studies, or accumulated experience with similar systems. Such situations are particularly common in industrial applications, where the availability of high-quality data may be limited due to cost, time, or measurement constraints, but significant knowledge about system behavior is often available through domain expertise and prior understanding. Unlike standard NNs, which mainly rely on data, GradINN combines a primary network, designed to reconstruct the target function, with an auxiliary network that encodes prior beliefs and uses a customized loss function to ensure consistency between the reconstructed function and the hypothesized behavior. As a result, GradINN can effectively model complex systems even in scenarios where only limited data are available, by leveraging prior beliefs about function's derivatives, which can often be assumed in real-world applications.

The rest of the paper is organized as follows. In Section 2, we introduce the notation and present a detailed mathematical formulation of GradINN, explaining the underlying mechanism. This section also includes illustrative toy examples that visually highlight the role and impact of prior beliefs during the training process. Section 3 is dedicated to the experimental evaluation of GradINN, where we assess model's performance on synthetic benchmark functions and real-world engineering problems, showcasing GradINN's effectiveness across different scenarios. Finally, Section 4 discusses the main findings of this work, explores its limitations, and outlines potential directions for future research.

2. Gradient-informed neural networks

In this section, we introduce Gradient-Informed Neural Networks (GradINN), a novel deep learning model that integrates two interacting NNs. The primary NN is designed to learn the regression task, while the auxiliary NN encodes prior beliefs about the target function's derivatives - leveraging its intrinsic properties such as differentiability, piece-wise linearity, or initialization - and guides the behavior of the primary NN's derivatives. Using a customized loss function (see Section 2.1 for details), the gradients of the primary network, computed with respect to its inputs at the collocation points, are constrained to align with the outputs of the auxiliary network, ensuring consistency with the hypothesized behavior of the target function's derivatives. Note that, since the collocation points are unlabeled, no additional experiments or gradient measurements are required to impose the constraint. During training, these constraints are progressively relaxed as the primary network adapts to the data, achieving a balance between prior belief and empirical observations. For a detailed explanation, refer to Section 2.2.

2.1. Mathematical formulation

Let $\boldsymbol{F} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$ be the differentiable target function of a regression task and let us denote by $f_1, \dots, f_m$ the scalar functions that define $\boldsymbol{F}$; i.e., $f_j : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}$, for each $j = 1, \dots, m$, and

$$\boldsymbol{F}(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), \dots, f_m(\boldsymbol{x})\big) \in \mathbb{R}^m \tag{1}$$

for each $\boldsymbol{x} \in \Omega$.

Let $\widehat{\boldsymbol{F}}(\cdot\,;\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$ be the characterizing function of a generic NN, suitable for learning $\boldsymbol{F}$, and let $\widehat{\boldsymbol{G}}(\cdot\,;\boldsymbol{\eta}) : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ be the characterizing function of a second NN, where $\boldsymbol{\theta} \in \mathbb{R}^p$ and $\boldsymbol{\eta} \in \mathbb{R}^q$ denote the trainable parameters of the two NNs, respectively. To avoid cluttering the notation, we denote the Neural Networks as $\widehat{\boldsymbol{F}} := \widehat{\boldsymbol{F}}(\cdot;\boldsymbol{\theta})$ and $\widehat{\boldsymbol{G}} := \widehat{\boldsymbol{G}}(\cdot;\boldsymbol{\eta})$, omitting their dependency on the trainable parameters. Moreover, we denote by $\widehat{f}_1, \dots, \widehat{f}_m$ the scalar functions that define $\widehat{\boldsymbol{F}}$, and by $\widehat{\boldsymbol{g}}_1, \dots, \widehat{\boldsymbol{g}}_m$ the vectorial functions that define $\widehat{\boldsymbol{G}}$; i.e., $\widehat{f}_j(\cdot\,;\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}$, $\widehat{\boldsymbol{g}}_j(\cdot\,;\boldsymbol{\eta}) : \mathbb{R}^n \to \mathbb{R}^n$, for each $j = 1, \dots, m$, and

$$\begin{aligned} \widehat{\boldsymbol{F}}(\boldsymbol{x}) &= \Big(\widehat{f}_1(\boldsymbol{x}), \dots, \widehat{f}_m(\boldsymbol{x})\Big) \in \mathbb{R}^m, \\ \widehat{\boldsymbol{G}}(\boldsymbol{x}) &= \begin{bmatrix} \widehat{\boldsymbol{g}}_1^T(\boldsymbol{x}) \\ \vdots \\ \widehat{\boldsymbol{g}}_m^T(\boldsymbol{x}) \end{bmatrix} \in \mathbb{R}^{m\times n}, \end{aligned} \tag{2}$$

for $\boldsymbol{x} \in \mathbb{R}^n$.

Let $\nabla_{\boldsymbol{x}}$ be the differential operator used to compute the gradient of a scalar function with respect to the variables represented by $\boldsymbol{x}$. Similarly, let $\boldsymbol{J}_{\boldsymbol{x}}$ denote the differential operator used to compute the Jacobian of a vector-valued function with respect to $\boldsymbol{x}$. For instance, $\boldsymbol{J}_{\boldsymbol{x}}\widehat{\boldsymbol{F}}$ represents the Jacobian of $\widehat{\boldsymbol{F}}$, obtained by differentiating the Neural Network with respect to the input variables only, excluding the trainable parameters. Note that the derivatives of $\widehat{\boldsymbol{F}}$ can be computed using automatic differentiation (Baydin et al., 2018; Griewank, 1989; Linnainmaa, 1976; Rumelhart et al., 1986) (i.e., the same tool commonly employed to compute gradients of the loss function during NN training) allowing the exact computation of $\nabla_{\boldsymbol{x}}\widehat{f}_j$.

Given the target function $\boldsymbol{F}$ and two NNs, $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$, we can define a GradINN as follows:

Definition 1 (Gradient-Informed Neural Network)**.** A Gradient-Informed Neural Network, defined by $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$ to learn the target function $\boldsymbol{F}$, is a Neural Network model $\mathcal{G} = (\widehat{\boldsymbol{F}}, \widehat{\boldsymbol{G}}) : \mathbb{R}^n \to \mathbb{R}^m \times \mathbb{R}^{m\times n}$ trained according to the following criteria:

1. updates the weights $\boldsymbol{\theta}$ to satisfy $\widehat{\boldsymbol{F}} \approx \boldsymbol{F}$;
2. updates both the weights $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ for obtaining $\nabla_{\boldsymbol{x}}\widehat{f}_j \approx \widehat{\boldsymbol{g}}_j$, for each $j = 1, \dots, m$ (i.e., $\boldsymbol{J}_{\boldsymbol{x}}\widehat{\boldsymbol{F}} \approx \widehat{\boldsymbol{G}}$).

The two criteria illustrated in the definition are achieved by minimizing the sum (or, more generally, a linear combination) of two loss functions, denoted in this work by $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. The first loss function minimizes the approximation error of $\widehat{\boldsymbol{F}}$ with respect to $\boldsymbol{F}$ on a set of labeled data $\mathcal{B} = \{(\boldsymbol{x}_k, \boldsymbol{F}(\boldsymbol{x}_k))\}_{k=1}^{B}$ (e.g., a mini-batch sampled from

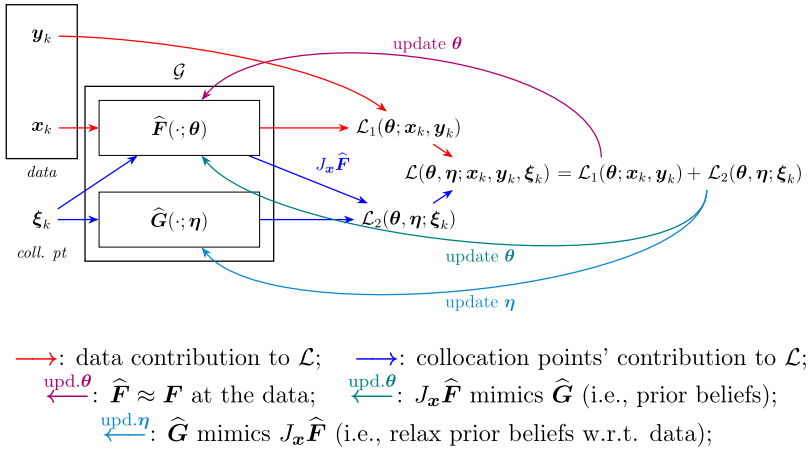

Fig. 1. Training scheme of GradINN. The arrows indicate contributions and parameter updates.

a training set); for instance, this loss function can be the Mean Squared Error (MSE) function:

$$\mathcal{L}_1(\theta;\mathcal{B}) = \frac{1}{B}\sum_{k=1}^{B} \|\widehat{F}(x_k) - F(x_k)\|^2, \tag{3}$$

where $\|\cdot\|$ denotes the euclidean norm. The second loss function minimizes the difference between the input-based derivatives of $\widehat{F}$ and the outputs of $\widehat{G}$ on a batch of *collocation points* $\mathcal{B}_c = \{\xi_k\}_{k=1}^{B_c} \subset \Omega$; i.e., sampled from the domain Ω of F. For instance, this loss function can be:

$$\begin{aligned}\mathcal{L}_2(\theta,\eta;\mathcal{B}_c) :&= \frac{1}{B_c}\sum_{k=1}^{B_c}\sum_{j=1}^{m} \|\nabla_x \widehat{f}_j(\xi_k) - \widehat{g}_j(\xi_k)\|^2 = \\ &= \frac{1}{B_c}\sum_{k=1}^{B_c} \left\| J_x\widehat{F}(\xi_k) - \widehat{G}(\xi_k)\right\|_{\mathrm{F}}^2,\end{aligned} \tag{4}$$

where $\|\cdot\|_{\mathrm{F}}$ denotes the Frobenius norm for matrices. In case of $m = 1$, i.e., $\widehat{F} : \mathbb{R}^n \to \mathbb{R}$ and $\widehat{G} : \mathbb{R}^n \to \mathbb{R}^n$, Eq. (4) reduces to

$$\mathcal{L}_2(\theta,\eta;\mathcal{B}_c) := \frac{1}{B_c}\sum_{k=1}^{B_c} \|\nabla_x \widehat{F}(\xi_k) - \widehat{G}(\xi_k)\|^2 . \tag{5}$$

Given $\mathcal{L}_1$ and $\mathcal{L}_2$, defined as in (3) and (4), a possible loss function for training the GradINN model $\mathcal{G} = (\widehat{F}, \widehat{G})$ is:

$$\begin{aligned}\mathcal{L}(\theta,\eta;\mathcal{B},\mathcal{B}_c) &:= \mathcal{L}_1(\theta;\mathcal{B}) + \mathcal{L}_2(\theta,\eta;\mathcal{B}_c) = \\ &= \frac{1}{B}\sum_{k=1}^{B} \|\widehat{F}(x_k) - F(x_k)\|^2 + \frac{1}{B_c}\sum_{k=1}^{B_c}\sum_{j=1}^{m} \|\nabla_x \widehat{f}_j(\xi_k) - \widehat{g}_j(\xi_k)\|^2,\end{aligned} \tag{6}$$

Which, in case of $m = 1$, reduces to

$$\begin{aligned}\mathcal{L}(\theta,\eta;\mathcal{B},\mathcal{B}_c) = &\frac{1}{B}\sum_{k=1}^{B} \left(\widehat{F}(x_k) - F(x_k)\right)^2 \\ &+ \frac{1}{B_c}\sum_{k=1}^{B_c} \left\|\nabla_x \widehat{F}(\xi_k) - \widehat{G}(\xi_k)\right\|^2 .\end{aligned} \tag{7}$$

The learning process described above is schematically illustrated in Fig. 1.

The mathematical formulation can be extended to incorporate prior beliefs about higher-order derivatives by introducing additional Neural Networks and corresponding loss terms. For additional details, refer to Appendix A.

Remark 1 (GradINN does not require ground-truth gradient information)**.** It is worth highlighting that GradINN operates without requiring any access to the ground-truth gradients of the target function. The only derivative-related information comes from the user-defined prior belief, which is used to design the architecture and initialize the weights of the auxiliary network $\widehat{G}$. Indeed, the collocation points in $\mathcal{B}_c$ are unlabeled and the only supervision stems from the target function values (see Fig. 1).

2.2. Working mechanism

GradINNs extends the common practice of building a NN (by defining architecture, hyper-parameters, etc.) according to the available information of the target function by introducing a second NN designed to incorporate information about its derivatives. Consequently, $\widehat{G}$ can be tailored with an architecture and set of hyperparameters that reflect our *prior beliefs* about F's derivatives, suggesting their behavior at the collocation points (see (4)). Rather than predicting exact derivative values, $\widehat{G}$ acts as a guide for the gradients of the primary network, biasing $\nabla_x \widehat{F}$ toward the assumed behavior and influencing their evolution throughout training in a way that reflects the embedded prior.

For example, consider the case $m = 1$ and assume that $\nabla_x F$ is "almost" or "somehow" piece-wise linear. In this scenario, the *relu* activation function can be used for all layers of $\widehat{G}$, resulting in a network with piece-wise linear outputs. This design guides $\nabla_x \widehat{F}$ to mimic our prior belief about $\nabla_x F$ in the regions of the domain where collocation points are placed. Specifically, $\mathcal{L}_2$ updates the weights θ of $\widehat{F}$ to align $\nabla_x \widehat{F}$ with the outputs of the piece-wise linear network $\widehat{G}$. The same approach can be extended to other cases where $\nabla_x F$ is assumed to be smooth, periodic, strictly positive/negative, or discontinuous, by designing $\widehat{G}$ with activation functions that are d-times continuously differentiable, periodic, strictly positive/negative, or discontinuous.

Note that, since our assumptions about F and its derivatives represent qualitative expectations rather than exact values, the implicit constraints imposed by $\widehat{G}$ are relaxed during training based on available data. Specifically, the loss $\mathcal{L}_2$ updates the weights η of $\widehat{G}$, aligning its outputs with the current values of $\nabla_x \widehat{F}$, which are influenced by data through the loss $\mathcal{L}_1$ (see Eq. (3), Definition 1, and Fig. 1). As a result, during training, $\widehat{G}$ adapts its constraints to better align with the available data.

Remark 2 (Auxiliary network initialization)**.**

The auxiliary network $\widehat{G}$ is randomly initialized and thus provides no theoretical guarantee of immediately generating useful or optimal gradient guidance for the primary network $\widehat{F}$.However, the initialization defines the qualitative nature of the prior belief embedded in $\widehat{G}$ (e.g., smooth or oscillatory), rather than its quantitative accuracy. The effectiveness of the auxiliary network emerges during the joint optimization process, where $\widehat{G}$ progressively adapts its outputs to the data distribution and to the evolving gradients of $\widehat{F}$. This co-training mechanism allows $\widehat{G}$ to act as a data-driven regularizer, adjusting its gradient constraints according to the feasibility induced by the data. In practice, while its

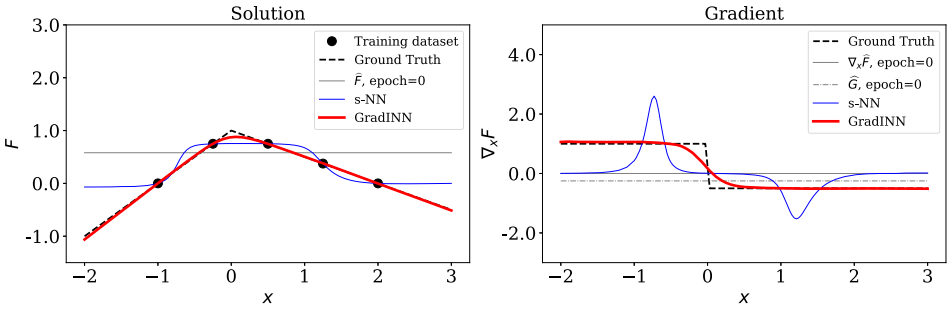

Fig. 2. Piece-wise differentiable function. Ground truth (black dashed lines) and predicted solution (left) and gradient (right) both at initialization (gray lines) and after training for s-NN and GradINN with smooth initialization of both $\widehat{F}$ and $\widehat{G}$.

benefits may be limited under incorrect prior assumptions, our experiments show that when the initialization is coherent with the expected gradient behavior, the model achieves more stable training and higher approximation accuracy (see Section 3).

2.2.1. *Visual examples*

To further clarify the impact of the proposed approach and shed light of GradINN's working mechanism, We show the results obtained when using standard NN (i.e., NN trained solely with Eq. 3, referred to as s-NN hereafter) and GradINN with different prior beliefs on two toy problems.

First, we consider the problem of learning a target piece-wise differentiable function $\boldsymbol{F} : \mathbb{R} \to \mathbb{R}$ defined as:

$$\boldsymbol{F}(x) = \begin{cases} x+1, & \text{if } x \leq 0 \\ -\frac{1}{2}x+1, & \text{otherwise} \end{cases} \quad \text{and} \quad \boldsymbol{F}'(x) = \begin{cases} 1, & \text{if } x < 0 \\ \nexists, & \text{if } x = 0 \\ -\frac{1}{2}, & \text{otherwise} \end{cases}.$$

We train GradINN and a s-NN model considering a training set composed of 5 samples and a set of 100 collocation points uniformly distributed in $[-2, 3]$. We first analyze the effect of applying a smooth prior belief, achieved through the use of smooth activation functions and specific network initialization. Subsequently, we show the results obtained when using the same prior belief but a different initialization for $\widehat{F}$. Finally, we examine the results obtained by varying the initialization for both $\widehat{F}$ and $\widehat{G}$. Across all experiments, the architecture and initialization of the s-NN used for comparison are identical to those of the network $\widehat{F}$ in the GradINN model.

For the GradINN, $\widehat{F}$ was designed as a NN with three hidden layers, each containing 20 neurons. A *sigmoid* activation function was applied across all layers, except for the output layer, which used a linear activation function. The network was initialized with biases set to zero and weights sampled from a Glorot uniform distribution (Glorot & Bengio, 2010). For $\widehat{G}$, we implemented a network with two hidden layers, each comprising 50 neurons. To incorporate a smooth prior belief, *sigmoid* activation functions were applied to the hidden layers, while the output layer employed a linear activation function. Similarly to $\widehat{F}$, the network initialization featured biases set to zero and weights sampled from the Glorot uniform distribution. This initialization, together with the sigmoid activation function, ensures that the network output is approximately constant at initialization, thereby enhancing smoothness in the encoded belief. Note that, even though the target function's derivative exhibits a discontinuity, it remains smooth elsewhere. Given the assumption of limited specific information about the function, using a smooth prior aligns with the general behavior we aim to incorporate in GradINN training.

Fig. 2 shows the ground truth solution (left, black line) and gradient (right, black line) together with the training dataset (black dots) and the predictions both at initialization (gray lines; at initialization the output of $\widehat{F}$ and s-NN is the same) and after training. The s-NN (blue lines) overfits the training set (left plot), exhibiting significant gradient oscillations (right plot), which lead to poor prediction performance. When considering GradINN, the output of $\widehat{G}$ at initialization (i.e., the prior belief) is constant, neither matching the shape of the gradient ground truth nor $\nabla_x\widehat{F}$. During training, $\widehat{G}$ adapts based on information from the training dataset while constraining $\nabla_x\widehat{F}$ to align with the evolving prior belief. By the end of the training (red lines), the outputs of $\widehat{F}$ fit the training set, while $\widehat{G}$ and $\nabla_x\widehat{F}$ converge, ensuring consistency between the updated prior belief and the learned gradients. As a result, GradINN demonstrates superior performance, yielding a smoother function that avoids the large gradient oscillations observed in the s-NN while providing a better fit to both the ground truth function and its gradient.

In order to showcase the performance of GradINN in settings where the output of the primary network at initialization is non-smooth, we repeated the analysis sampling the weights of $\widehat{F}$ from a normal distribution with mean 0 and standard deviation 5 , while setting the biases to 0. For comparison, the same initialization of $\widehat{F}$ was applied to a s-NN. For all the networks, the architecture remains identical to the previous case, with the same number of hidden layers, neurons, and activation functions. As shown in Fig. 3, the s-NN trained with this initialization produces an output that continues to overfit the training set but results in a completely different function, exhibiting an even more irregular gradient. In contrast, when GradINN is used, the smooth prior belief, similar to the previous case, guides the network toward a smoother solution, comparable to the previous result and more consistent with the ground truth. This suggests that the smooth prior belief imposed by $\widehat{G}$ can help mitigate the effects of noisy initializations in $\widehat{F}$. When the same settings are used for $\widehat{F}$ (and consequently for the s-NN), but a less smooth initialization is applied to $\widehat{G}$ (weights sampled from a normal distribution with mean 0 and standard deviation 5 and biases set to zero) the solution reconstructed by GradINN, as shown in Fig. 4, is no longer smooth and exhibits large variations in the gradients. This demonstrates how a prior belief based on a non-smooth initialization fails to help GradINN recover the target function.

Expanding on the previous examples, we now focus on the prediction of an high frequency oscillatory target function: the Rastrigin function. We first examine the effect of a smooth prior belief in reconstructing an oscillatory function and subsequently explore the impact of an oscillatory prior belief, thus highlighting GradINN flexibility in terms of prior belief that can be considered.

The Rastrigin function is defined as:

$$\boldsymbol{F}(x) = A\,n + \sum_{i=1}^{n}\left(x_i^2 - A\,\cos(2\pi x_i)\right), \quad \boldsymbol{F}'(x) = \sum_{i=1}^{n}\left(2x_i + 2\pi A\,\sin(2\pi x_i)\right).$$

where $A = 10$ and $n = 1$ in this example. For the training dataset, we sampled 30 points randomly distributed within the range $[-5.12, 5.12]$. Fig. 5 compares the performance of s-NN and GradINN with a smooth prior belief. Once again, the s-NN overfits the training set, producing large spikes in the predicted gradient. On the other hand, when GradINN is applied with a smooth prior belief (generated by an auxiliary network $\widehat{G}$ with the same architecture and initialization as in Fig. 2 and Fig. 3), the reconstructed function exhibits an oversmoothing effect. This is a consequence of the regularization imposed by the smooth prior belief.

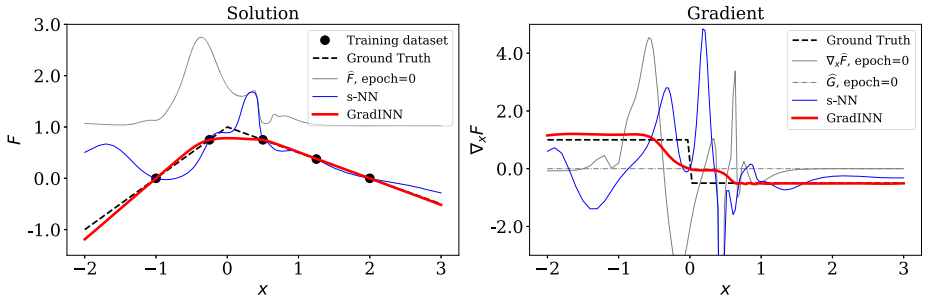

Fig. 3. Piece-wise differentiable function. Ground truth (black dashed lines) and predicted solution (left) and gradient (right) both at initialization (gray lines) and after training for s-NN and GradINN with noisy initialization for $\widehat{F}$ and smooth initialization for $\widehat{G}$.

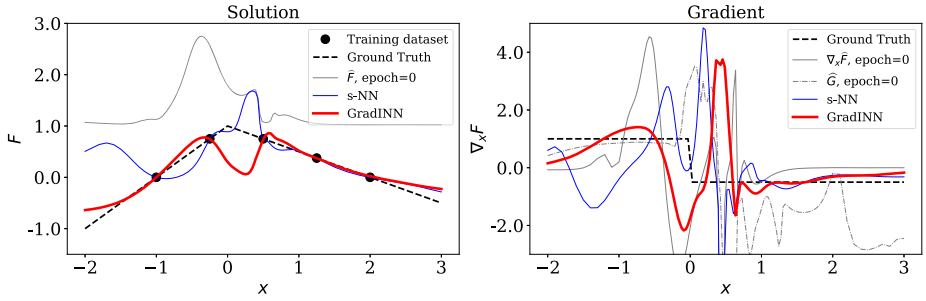

Fig. 4. Piece-wise differentiable function. Ground truth (black dashed lines) and predicted solution (left) and gradient (right) both at initialization (gray lines) and after training for s-NN and GradINN with noisy initialization for both $\widehat{F}$ and $\widehat{G}$.

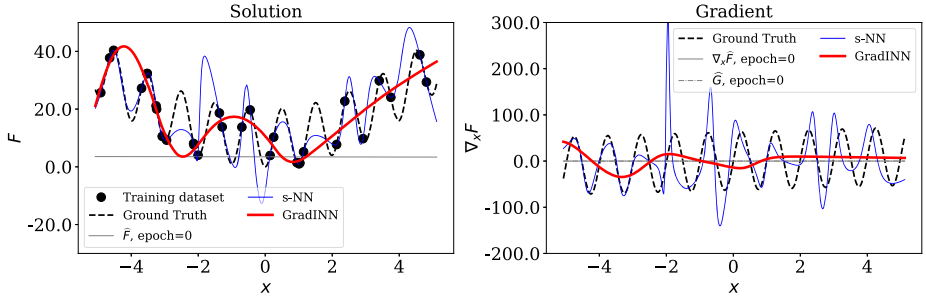

Fig. 5. Oscillatory function. Ground truth (black dashed lines) and predicted solution (left) and gradient (right) both at initialization (gray lines) and after training for s-NN and GradINN with smooth prior belief.

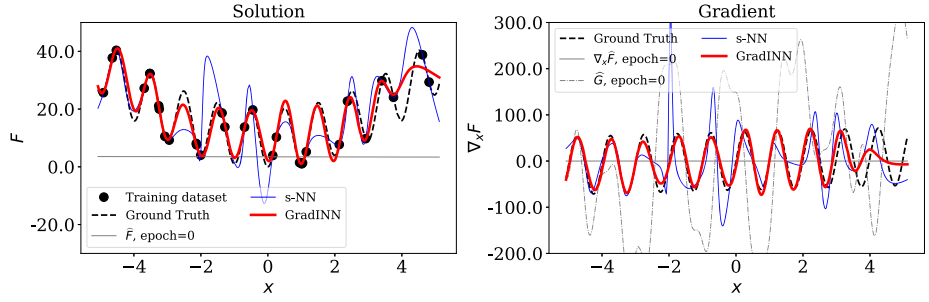

Fig. 6. Oscillatory function. Ground truth (black dashed lines) and predicted solution (left) and gradient (right) both at initialization (gray lines) and after training for s-NN and GradINN with oscillatory prior belief.

Indeed, enforcing gradient smoothness prevents oscillations and leads to underfitting. When a GradINN with an oscillatory prior belief is considered (Fig. 6), using an auxiliary network $\widehat{G}$ with a sufficiently oscillatory initialization (one hidden layer with 10 neurons, biases set to zero, and weights sampled from a normal distribution with mean 0 and standard deviation 20), the reconstructed function matches the target function.

These results, together with those from the previous examples, align with the interpretation of the auxiliary network $\widehat{G}$ as a prior belief that guides the training of $\widehat{F}$ while adapting to the available data. They also show how GradINN can accommodate both simple prior beliefs (e.g., smooth priors) and, when properly designed, more complex ones (e.g., oscillatory priors), thanks to the flexibility provided by the initialization strategy and architecture of $\widehat{G}$.

3. Experimental results

Since many physical and engineering systems exhibit behaviors characterized by smooth transitions and well-behaved gradients, to explore how GradINNs can effectively leverage such general prior information, we first focus our experiments on analyzing the influence of a smooth prior belief on the training process, examining how such priors guide the model and affect the prediction of the target function.

The effect of the smooth prior is assessed on both synthetic benchmark functions (Section 3.2) and real-world engineering datasets from the UCI Machine Learning Repository (Dua & Graff, 2017) (Section 3.3), where smooth gradients are typically expected. In addition, to evaluate the scalability of the framework, we also test GradINN on a high-dimensional regression problem to assess its performance when the dimensionality of both the input feature space and the output space increase (see Section 3.4).

Finally, to further explore the flexibility of the proposed approach, we extend the analysis to functions characterized by oscillatory behaviors, evaluating the role of an oscillatory prior belief in Section 3.5.

Given that GradINN does not rely on structured prior physical knowledge but only on prior beliefs and data, we compare it against an s-NN having the same architecture and hyperparameters as the primary network $\widehat{\boldsymbol{F}}$, so as to isolate the effect introduced by the auxiliary network $\widehat{\boldsymbol{G}}$ and the associated gradient-based regularization.

In the following, we first outline the experimental details shared across all experiments, and then present the results.

3.1. Experimental details

To train and evaluate GradINN model with respect to the function $\boldsymbol{F} : \Omega \subseteq \mathbb{R}^n \to \mathbb{R}^m$, we consider three distinct sets:

- $\mathcal{D}_{\text{TRAIN}}$: training set made of N_{TRAIN} pairs $(\boldsymbol{x}, \boldsymbol{F}(\boldsymbol{x})) \in \Omega \times \mathbb{R}^m$;
- $\mathcal{D}_{\text{TEST}}$: training set made of N_{TEST} pairs $(\boldsymbol{x}, \boldsymbol{F}(\boldsymbol{x})) \in \Omega \times \mathbb{R}^m$;
- $\mathcal{C}$: set of N_C collocation points, such that $\mathcal{C} := \{\boldsymbol{\xi}_k\}_{k=1}^{N_C} \subset \Omega$.

Model's performances are assessed in terms of Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N_{\text{TEST}}} \sum_{k=1}^{N_{\text{TEST}}} (\widehat{\boldsymbol{F}}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_k))^2},$$

$$\text{MAE} = \frac{1}{N_{\text{TEST}}} \sum_{k=1}^{N_{\text{TEST}}} |\widehat{\boldsymbol{F}}(\boldsymbol{x}_k) - \boldsymbol{F}(\boldsymbol{x}_k)|,$$

where $\boldsymbol{F}(\boldsymbol{x}_k)$ and $\widehat{\boldsymbol{F}}(\boldsymbol{x}_k)$ represent the ground truth solution and the predicted output for each $\boldsymbol{x}_k$ of the test set $\mathcal{D}_{\text{TEST}}$.

Regarding the experimental setup, the architectures of the primary network $\widehat{\boldsymbol{F}}$ and the auxiliary network $\widehat{\boldsymbol{G}}$ were defined separately for each type of experiment. In all cases, the baseline s-NN employed the same architecture and weight initialization as the primary network $\widehat{\boldsymbol{F}}$, ensuring that any performance difference arises solely from the presence of the auxiliary network $\widehat{\boldsymbol{G}}$ and the additional gradient-consistency term $\mathcal{L}_2$. Specifically, the architectures adopted are the following:

- *Synthetic smooth functions and UCI datasets (Sections 3.2 and 3.3)*: we set $\widehat{\boldsymbol{F}}$ to be a network with three hidden layers including 20 neurons each. For $\widehat{\boldsymbol{G}}$, we consider two hidden layers comprising 50 neurons each. Both $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$ use sigmoid activation functions across all layers except the final one, which employs a linear activation function.

  Both $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$ are initialized with biases set to zero and weights with Glorot Uniform distribution. This leads to constant network outputs at initialization (for both $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$) which imply, together with the sigmoid activation function, a smooth prior belief. It is important to emphasize that, in GradINN, the smooth prior belief is not influenced by a single initialization instance, but rather by the initialization strategy, i.e., the distribution from which weights are sampled.

  In Appendix B we repeat a subset of experiments by varying the activation function of the hidden layers, substituting the sigmoid with ELU, tanh, GELU, and ReLU activations.
- *High-dimensionality real-world problem (Section 3.4):* both $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$ are implemented as deep networks with nine hidden layers of 128 neurons each with ELU activation function and residual connections inserted every three layers (He et al., 2016). Also in this case the NNs are initialized with zero biases, Glorot Uniform distribution for the weights, since in this problem we do not expect discontinuities or oscillatory behaviors in the target function.
- *Oscillatory functions (Section 3.5):* for the experiments involving highly oscillatory target functions, both the primary and auxiliary networks were modified to encode an oscillatory prior beliefs. Specifically, $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$ were implemented as Sinusoidal Representation Networks (SIREN) (Sitzmann et al., 2020), each comprising two hidden layers with 50 neurons per layer and sinusoidal activation functions. The output layers remained linear, while weights were initialized according to the SIREN initialization strategy with $\omega_0 = 30$ and biases set to zero. This configuration enables $\widehat{\boldsymbol{G}}$ to impose a high-frequency prior belief on the gradients of $\widehat{\boldsymbol{F}}$, promoting oscillatory patterns during training.

To prevent overfitting, we employ early stopping criteria by using 20% of the training dataset as a validation set, monitoring performance during training and restoring the model weights that achieve the best result on the validation dataset.

For GradINN's training, since $\mathcal{D}_{\text{TRAIN}}$ and $\mathcal{C}$ may have different dimensions, we create a combined dataset defined as the Cartesian product $\mathcal{D}_{\text{TRAIN}} \times \mathcal{C}$, where each element takes the form $((\boldsymbol{x}_k, \boldsymbol{F}(\boldsymbol{x}_k)), \boldsymbol{\xi}_h)$ and the resulting dataset has a total of $N_{\text{TRAIN}} \cdot N_C$ elements. This approach enables a unified mini-batch division with a fixed batch size of 64 applied across all experiments. The networks $\widehat{\boldsymbol{F}}$ and $\widehat{\boldsymbol{G}}$ are trained simultaneously using Adam optimizer (Kingma & Ba, 2015) to minimize Eq. 6 with a learning rate starting at 0.1 and adjusted via a Reduce on Plateau schedule, reducing by a factor of 0.5 if the performance on the validation dataset does not improve after 10 epochs of patience.

The s-NN used for comparison employs the same batch size, and is trained using the same early stopping strategy (with a validation dataset that includes the same elements used for GradINN) and a similar Reduce on Plateau schedule with a patience of 100 epochs for the learning rate.

To evaluate the influence of initialization on performance, particularly for GradINN, where the prior depends on the initialization of the auxiliary network $\widehat{\boldsymbol{G}}$, in each experiments, both models were trained using 5 different weight initialization.

3.2. Syntetic function

This section presents the results obtained with a smooth prior belief on standard synthetic benchmark functions, evaluating GradINN's performance on five well-known test cases commonly used in optimization and regression studies: FRIEDMAN, GOLDSTEIN-PRICE, ROSENBROCK, BOOTH, and POWELL.

These functions exhibit diverse properties, including varying input (n) and output (m) dimensions, as well as different characteristics in terms of function shape and gradient smoothness. The detailed description of each function is provided in Appendix C.1.

We construct $\mathcal{D}_{\text{TRAIN}}$, $\mathcal{C}$ and $\mathcal{D}_{\text{TEST}}$ by sampling input points using Latin hypercube sampling and then calculating the corresponding output values for each function. We fix N_{TEST} at 10^4 points for each function and vary the sizes of both $\mathcal{D}_{\text{TRAIN}}$ and $\mathcal{C}$. Specifically, we explore four levels of N_{TRAIN} ($N_{\text{TRAIN}} = 50, 100, 200, 500$) and five levels of N_C ($N_C = 50, 100, 250, 500, 1000$). This experimental setup enabled us to

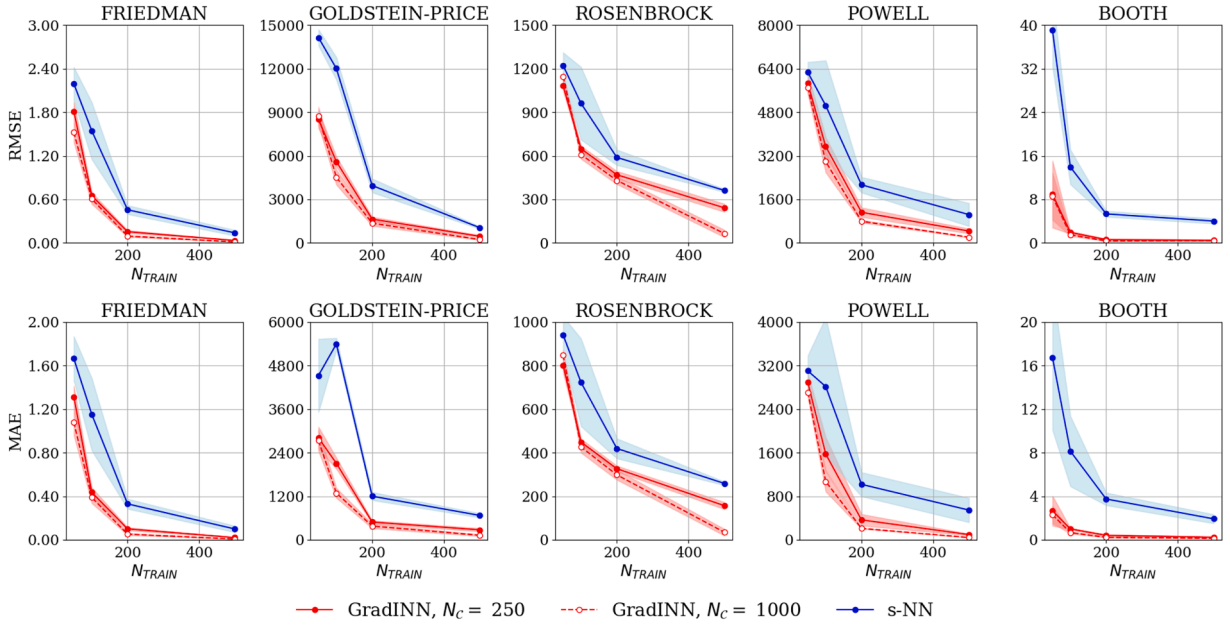

Fig. 7. SYNTHETIC FUNCTIONS. Comparison of RMSE (Top) and MAE (Bottom) for GradINN and s-NN on D_{TEST} for different N_{TRAIN}. For GradINN, results are reported for $N_c = 250$ and $N_c = 1000$. The shaded areas represent the standard deviation across 5 runs with different NNs initialization.

examine how variations in dataset sizes influence the models' ability to approximate the target functions across different scenarios.

Fig. 7 illustrates the RMSE and MAE for all synthetic functions as a function of N_{TRAIN}. The lines represent the average error across the initialization seeds, while the shaded areas show the corresponding standard deviation. The blue curve corresponds to s-NN, while the red curves represent GradINNs. In the figure, we consider only $N_C = 250$ and $N_C = 1000$, while the complete results, including all values of N_C, are provided in Appendix D. As expected, increasing N_{TRAIN} leads to a reduction in both RMSE and MAE for both s-NN and GradINN models. However, GradINN consistently outperforms s-NN across all levels of N_{TRAIN} , with both values of N_C, thanks to the incorporation of a smooth prior belief. This highlights the efficacy of GradINN in leveraging general prior information to guide the training process. Nonetheless, as N_{TRAIN} increases, the characterization of the system through the data becomes more complete, allowing s-NN to approach the performance of GradINN.

Additionally, GradINN exhibits a lower standard deviation for both metrics compared to s-NN, reflecting its robustness across different initialization seeds. This behavior suggests that the design of the smooth prior belief in GradINN is not directly dependent on the specific weight values of the auxiliary network $\widehat{\boldsymbol{G}}$. Instead, it primarily arises from the Glorot distribution used for initializing these weights, making the implementation of the smooth prior belief straightforward and reproducible across different training runs.

Analyzing the influence of N_C, Fig. 8 illustrates the RMSE and MAE for s-NN (blue points) and GradINN (red lines) across varying values of N_C, with results presented for two fixed levels of N_{TRAIN}. As observed, increasing the number of collocation points leads to a consistent reduction in error for GradINN. However, this effect decreases for higher values of N_C, where the domain is already sufficiently covered by the collocation points. This indicates that excessively increasing N_C provides only marginal benefits in error reduction while increasing computational time.

Note that s-NN has been positioned on the graph at $N_C = 0$. In this region, corresponding to low or no collocation points, the performance of GradINN tends to align with that of s-NN. This behavior comes from the relationship between s-NN and GradINN: when N_C is null, Eq. 6 simplifies to Eq. 3, effectively reducing GradINN to s-NN, as its training relies only on minimizing the data-driven error term ($\mathcal{L}_1$). In general, as the number of collocation points decreases, the performance of GradINN approaches to that of s-NN.

See Appendix B for similar results obtained varying the activation functions.

3.3. UCI regression task

To demonstrate the practical applicability of GradINNs, we evaluated their performance on real-world engineering tasks using datasets from the UCI Machine Learning Repository. A summary of the datasets is provided in Table 1, with a more detailed description available in the Appendix C.2.

For each task, the dataset was partitioned into D_{TRAIN} and D_{TEST}. Half of the dataset was fixed as D_{TEST}. The remaining half was used to generate D_{TRAIN}, with training subset sizes varied across five configurations: 50%, 40%, 30%, 20%, and 10% of the full dataset (see Table 1 and Fig. 9). For C, we used the input coordinates of the entire dataset, as sampling the domain for collocation points is often a task-specific process requiring domain expertise. Consequently, the size of C matched the total number of instances in each dataset (e.g., for YACHT, $N_C = 308$; for AIRFOILNOISE, $N_C = 1503$; and so on).

To emphasize the low-data nature of these problems, Table 1 reports the number of samples used in each training set. For reference, we also indicate the approximate number of values per feature that would be required to obtain the same number of samples using a regular grid in the n-dimensional feature space. The resulting values correspond to extremely coarse grids especially given the nonlinear and irregular nature of the target functions.

Leveraging a smooth prior belief enables GradINN to consistently outperform NN across all datasets (Fig. 10), similar to what was previously observed in the synthetic function experiments. Note that we design $\widehat{\boldsymbol{G}}$ with the same architecture and initialization across all tasks, including the synthetic functions, highlighting that the smooth prior belief is neither task-specific nor requires problem-dependent tuning, making it straightforward to apply to diverse target functions.

The performance improvement of GradINN becomes particularly significant in real-world scenarios, where the collection of labeled data is often costly and time-consuming. The ability of GradINN to achieve com-

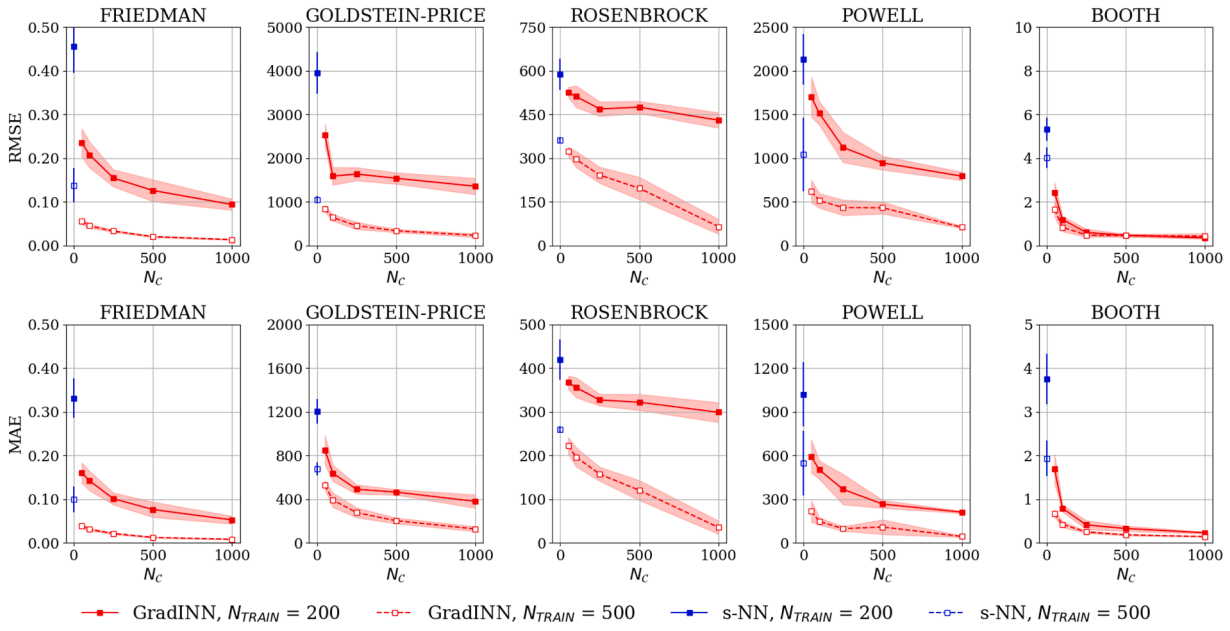

Fig. 8. SYNTHETIC FUNCTIONS. Comparison of RMSE (Top) and MAE (Bottom) for GradINN on D_{TEST} for different values of N_C, considering $N_{\text{TRAIN}} = 200$ and $N_{\text{TRAIN}} = 500$. The shaded areas represent the standard deviation across 5 runs with different NNs initialization.

Table 1
Characteristics of the UCI datasets used in the experiments. n denotes the number of input features and m the number of output variables. In the column "(as grid)", we report the approximate number of equally spaced values per feature that would be required to reach the same number of samples by using a regular grid in the n-dimensional feature space.

	YACHT		AIRFOILNOISE		POWER		ENERGY		NAVAL	
Instances	308		1,503		9,568		768		11,934	
n	6		5		4		8		16	
m	1		1		1		2		2	
D_{TRAIN}	Samples	(as grid)	Samples	(as grid)	Samples	(as grid)	Samples	(as grid)	Samples	(as grid)
50%	154	($\sim 2.3^n$)	752	($\sim 3.8^n$)	4784	($\sim 8.3^n$)	384	($\sim 2.1^n$)	5967	($\sim 1.7^n$)
40%	123	($\sim 2.2^n$)	601	($\sim 3.6^n$)	3827	($\sim 7.9^n$)	307	($\sim 2.1^n$)	4774	($\sim 1.7^n$)
30%	92	($\sim 2.1^n$)	451	($\sim 3.4^n$)	2870	($\sim 7.3^n$)	230	($\sim 2.0^n$)	3580	($\sim 1.7^n$)
20%	62	($\sim 2.0^n$)	301	($\sim 3.1^n$)	1914	($\sim 6.6^n$)	154	($\sim 1.9^n$)	2387	($\sim 1.6^n$)
10%	31	($\sim 1.8^n$)	150	($\sim 2.7^n$)	957	($\sim 5.6^n$)	77	($\sim 1.7^n$)	1193	($\sim 1.6^n$)

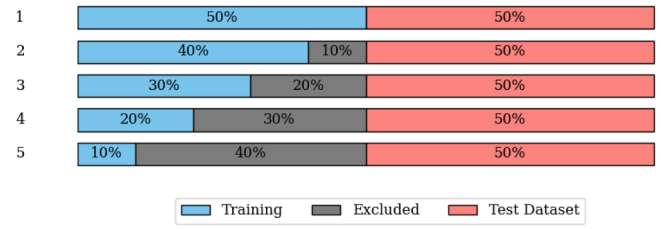

Fig. 9. The five different dataset splits considered for the UCI TASKS.

parable or even superior performance with smaller D_{TRAIN} not only minimizes experimental efforts, but also translates into substantial savings in resources and time, making it a practical and efficient solution for applications with limited data availability or expensive data acquisition processes.

See Appendix B for similar results obtained varying the activation functions.

3.4. Real-world high-dimensional regression task

To further assess the scalability of GradINN to high-dimensional regression problems and to more complex NN architectures, we tested its performance on a real-world problem involving 107 input features and 7 target values. Specifically, this real-world application consists in predicting the flux outflowing from a set of 7 fractures in a rock medium, given the transmissivity properties of the entire set of 107 intersecting fractures in the terrain. Further details about the dataset are reported in Appendix C.3.

In this experiment, we train an s-NN and a GradINN with respect to the problem of predicting the fluxes outflowing form 7 boundary fractures, varying the transmissivities of all 107 fracture. As described in Section 3.1, both the networks in the GradINN are residual NNs composed by 9 layers of 128 ELU units and that the s-NN has an identical structure to $\widehat{\boldsymbol{F}}$. The number of collocation point used for GradINN is $N_C = 1000$. Models performance was evaluated as a function of the number of training samples ($N_{\text{TRAIN}} = 250, 500, 750$), with respect to a test set of $N_{\text{test}} = 5000$ samples.

Looking at the results in Fig. 11, we observe that GradINN outperforms the identically structured s-NN in terms of both RMSE and MAE, across all seven output variables. The results of this experiment confirm that the proposed framework remains effective when scaling to higher input/output dimensionality and more complex architectures, reinforcing the general applicability of GradINN.

From a computational perspective, note that, GradINN scales similarly to PINNs, since both frameworks require differentiating the primary network $\widehat{\boldsymbol{F}}$ with respect to the input variables to compute $\nabla_x \widehat{\boldsymbol{F}}$ at

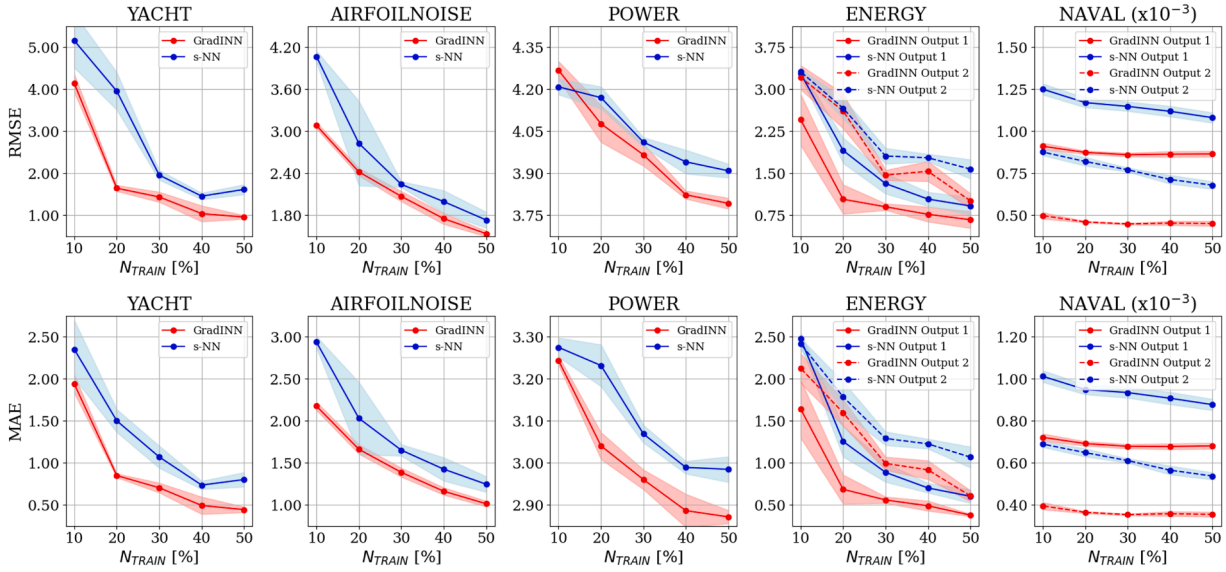

Fig. 10. UCI TASKS. Comparison of RMSE and MAE for GradINN and s-NN on $\mathcal{D}_{\text{TEST}}$ as the size of $\mathcal{D}_{\text{TRAIN}}$ varies. The shaded areas represent the standard deviation across 5 runs with different NNs initialization.

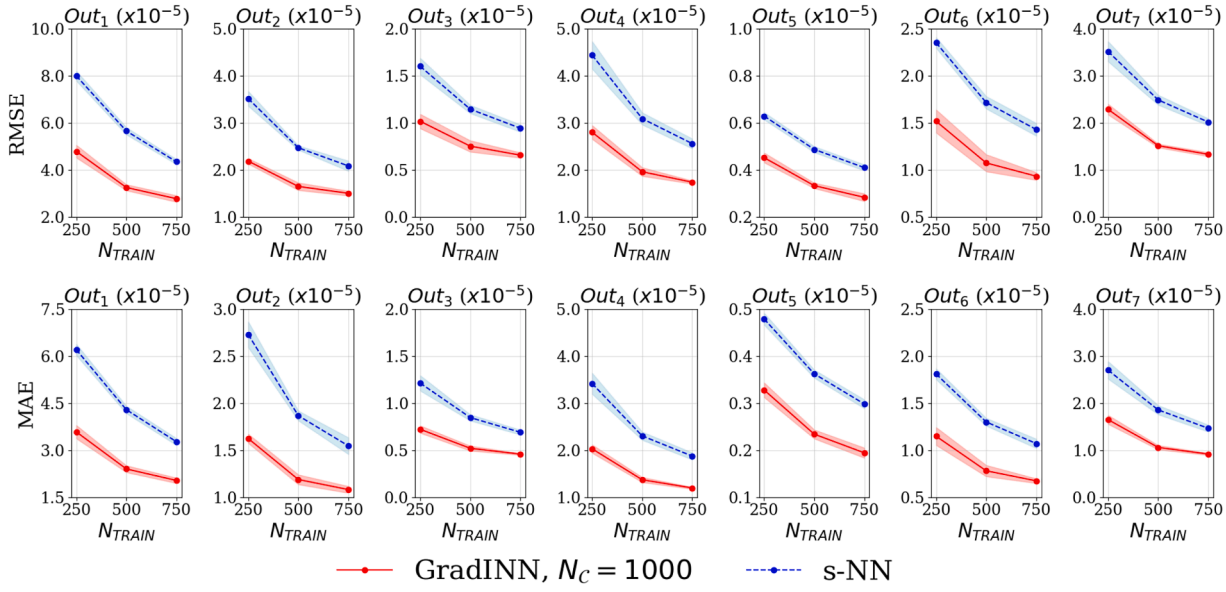

Fig. 11. Comparison between s-NN and GradINN on the high-dimensional regression problem (107 input and 7 output variables). Results are reported in terms of RMSE and MAE as a function of the number of training samples. The shaded areas represent the standard deviation across 5 runs with different NNs initialization.

the collocation points. The presence of the auxiliary network $\hat{G}$ slightly increases the number of trainable parameters, resulting in a minor additional computational cost, which becomes relevant only for very large architectures.

3.5. Oscillatory prior beliefs

After evaluating GradINN under smooth priors on both synthetic and real-world datasets, we now explore its capability to encode more complex qualitative assumptions by extending our analysis to cases characterized by oscillatory behaviors. This additional investigation aims to assess whether an oscillatory prior belief, encoded in the auxiliary network $\hat{G}$, can guide the primary network $\hat{F}$ and its gradients toward high-frequency behaviors, thereby improving the approximation quality.

We considered four well-known benchmark functions exhibiting high-frequency patterns: ACKLEY, RASTRIGIN, EGGHOLDER, and SCHWEFEL (see Appendix C.4). Each function was evaluated over its canonical input domain, with training datasets composed of 500, 750, and 1000 samples generated via Latin Hypercube Sampling, and a test set of 10000 uniformly distributed points. For all experiments, the number of collocation points used was fixed to $N_C = 2000$. Every configuration was repeated over five random seeds to ensure robustness and statistical consistency.

Three configurations were compared in this experiment: an s-NN, a GradINN with smooth priors, and a GradINN with oscillatory priors . All the NNs are SIREN models, as previously described in Section 3.1. The only difference lies in the auxiliary network $\hat{G}$ of the second configuration, where an intentionally smooth prior belief was tested by using sigmoid activation functions and Glorot-uniform initialization.

The performance comparison in terms of RMSE and MAE is reported in Fig. 12.

Across all functions and dataset sizes, the baseline s-NN fits the training data but fails to capture the oscillatory structure of the target func-

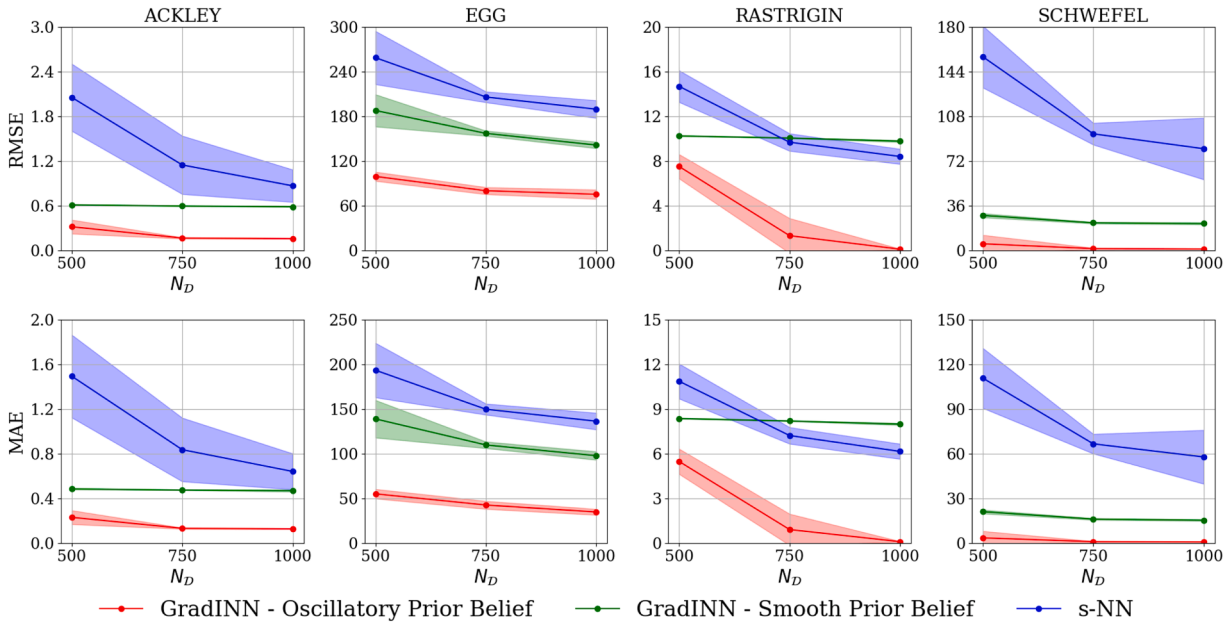

Fig. 12. Comparison of s-NN, GradINN with smooth prior belief, and GradINN with oscillatory prior belief on highly oscillatory benchmark functions. Results are reported in terms of RMSE and MAE as a function of N_D with $N_C = 2000$ for both configuration of GradINN. The shaded areas represent the standard deviation across 5 runs with different NNs initialization.

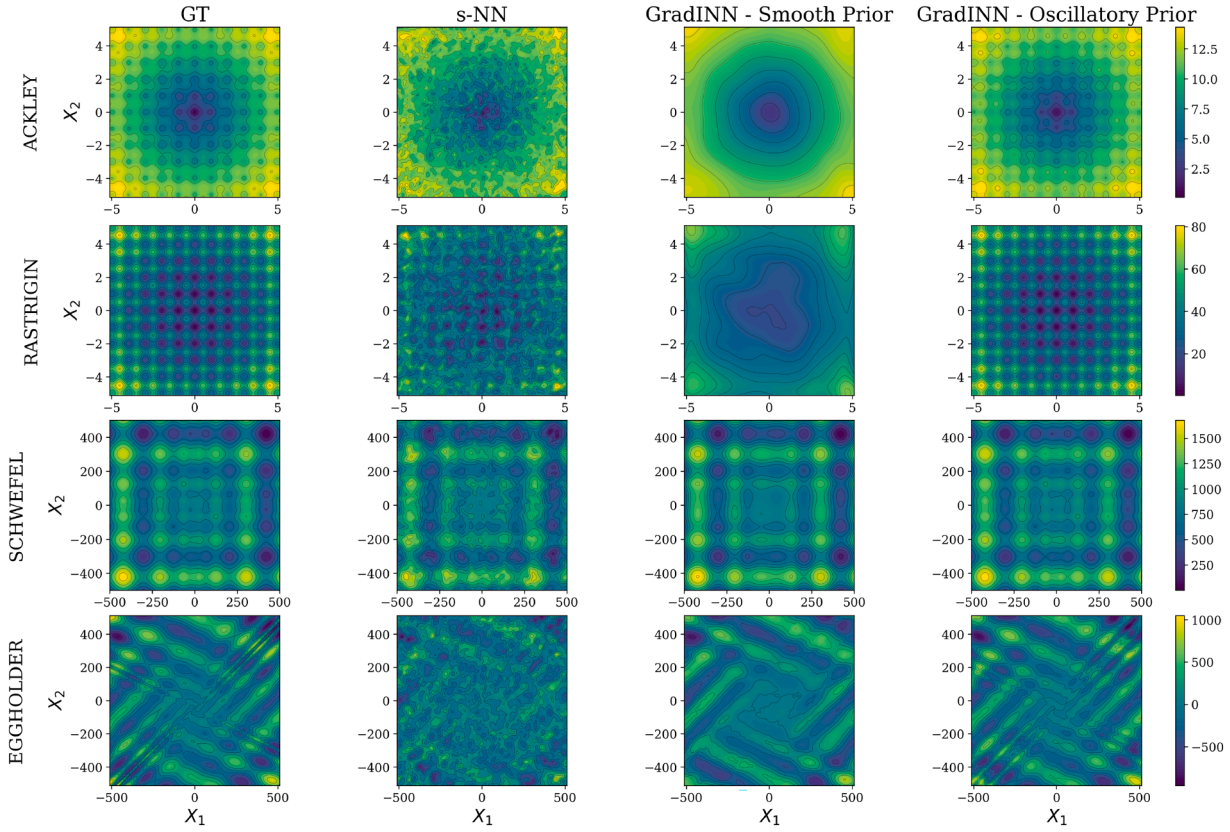

Fig. 13. Visual comparison on oscillatory function. From left to right: Ground truth (GT), baseline s-NN, GradINN with smooth prior belief, and GradINN with oscillatory prior belief ($N_D = 1000$, $N_C = 2000$).

tion, leading to poor generalization on the test domain. When a smooth prior is imposed, the reconstructed surface partially or completely loses its oscillatory nature, tending to average out rapid variations. In contrast, when an oscillatory prior is introduced through the auxiliary network $\widehat{G}$, the learned gradients display high-frequency components that closely follow the ground-truth oscillations, resulting in a more accurate reconstruction of the target function (see Fig. 13).

Note that, as in the smooth-prior experiments, the architecture and initialization of $\widehat{G}$ are identical across all the target functions. In particular, the imposed prior is not tailored to any specific function but

remains the same for all oscillatory benchmarks, reflecting the fact that the true gradients of each function are unknown and only qualitative assumptions are encoded. During training, however, this initial prior is progressively refined through the learning process, allowing the auxiliary network to adapt its gradient representation based on the available data.

These results showcase GradINN flexibility to adapt to different classes of priors, from smooth to oscillatory, without explicit gradient supervision.

4. Conclusion

In this work, we introduced Gradient-Informed Neural Networks (GradINN), a novel NN model that integrates both data and prior beliefs about the target function's derivatives to improve the training and generalization of Neural Networks, particularly in low-data regimes. By combining a primary network with an auxiliary network and employing a customized loss function, GradINN effectively incorporates prior belief by ensuring consistency between the primary network's gradients and the hypothesized behavior encoded in the auxiliary network. The prior belief is encoded in the auxiliary network through its architecture, activation functions, and initialization, which collectively define the qualitative behavior expected for the gradients, without requiring any access to the true derivatives of the target function. We provided a detailed mathematical formulation of GradINN and validated its effectiveness on both synthetic benchmarks and real-world engineering tasks.

This work shows that the GradINN framework can be suitable for different classes of prior beliefs, ranging from smooth to oscillatory gradient behaviors. Such flexibility enables the model to adapt to both regular and highly variable systems through appropriate choices of the auxiliary network's architecture, activation functions, and initialization. Our results further highlight that both smooth and oscillatory priors, implemented with the same auxiliary network architecture and a consistent initialization strategy, provide flexible and robust performance across diverse applications within their respective problem classes. In particular, the same GradINN configuration can be effectively applied to multiple problems that share a similar qualitative gradient behavior.

This work opens up several promising directions for future research. First, a deeper investigation into alternative classes of prior beliefs is of primary interest, including highly oscillatory ones, by considering different variation of the initialization parameters (e.g., different values ω_0), as well as discontinuous or spatially heterogeneous behaviors, where different regions of the input domain may exhibit distinct gradient regimes as, for instance, smooth in some areas and oscillatory in others. Developing strategies to effectively design and encode these complex priors within the auxiliary network would further extend GradINN's applicability to more complex and heterogeneous systems.

Additionally, future work could consider variations of the proposed loss function. For instance, introducing hyperparameters to balance the contributions of the data-driven and prior-belief losses would allow the model to prioritize either training data or prior beliefs depending on the specific problem requirements. Finally, extending the framework to incorporate higher-order derivatives, as discussed in Appendix A, would enable GradINN to leverage more prior information.

CRediT authorship contribution statement

Filippo Aglietti: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Project administration, Methodology, Formal analysis, Data curation, Conceptualization; **Francesco Della Santa:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Data curation; **Andrea Piano:** Writing – review & editing, Supervision; **Virginia Aglietti:** Writing – original draft, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

F.D. acknowledges that this study was carried out within the FAIR-Future Artificial Intelligence Research and received funding fromthe 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).

Appendix A. Higher-order derivatives

GradINN can be easily extended to incorporate also implicit constraints related to higher-order derivatives of the first NN $\widehat{\boldsymbol{F}}$. For example, assuming that $\widehat{\boldsymbol{F}} = (\widehat{f}_1, \dots, \widehat{f}_m)$ is twice differentiable, we can introduce a third network $\widehat{\mathrm{H}} := \widehat{\mathrm{H}}(\cdot\,; \boldsymbol{\zeta}) : \mathbb{R}^n \to \mathbb{R}^{m \times n \times n}$ for guiding the behavior of the hessians $H_{\boldsymbol{x}}\widehat{f}_1, \dots, H_{\boldsymbol{x}}\widehat{f}_m$ by adding a third loss term in (6). This third loss term is defined for minimizing the differences between the second-order derivatives of $\widehat{\boldsymbol{F}}$ and the NN $\widehat{\mathrm{H}}$, at the collocation points, e.g.:

$$\mathcal{L}_3(\boldsymbol{\theta}, \boldsymbol{\zeta}; \mathcal{B}_c) := \frac{1}{B_c} := \frac{1}{B_c} \sum_{k=1}^{B_c} \sum_{j=1}^{m} \left\| H_{\boldsymbol{x}} \widehat{f}_j(\boldsymbol{\xi}_k) - \widehat{\mathrm{H}}_j(\boldsymbol{\xi}_k) \right\|_{\mathrm{F}}^2, \tag{A.1}$$

where $\widehat{\mathrm{H}}_j$ is the j-th n-by-n matrix returned by $\widehat{\mathrm{H}}$, for each $j = 1, \dots, m$, i.e.:

$$\widehat{\mathrm{H}}_j(\boldsymbol{x}) := \left(\widehat{\mathrm{H}}_{j,k,\ell}(\boldsymbol{x}) \right)_{k,\ell=1}^{n} \in \mathbb{R}^{n \times n}, \quad \forall\, j = 1, \dots, m,\ \forall\, \boldsymbol{x} \in \mathbb{R}^n.$$

As a natural consequence, we can consider also a fourth loss term to be added to (6), to enforce the influence of the third new network on the training regularization. This extra loss term is defined for minimizing the differences between the first-order derivatives of $\widehat{\boldsymbol{G}}$ and the NN $\widehat{\mathrm{H}}$, at the collocation points, e.g.:

$$\mathcal{L}_4(\boldsymbol{\eta}, \boldsymbol{\zeta}; \mathcal{B}_c) := \frac{1}{B_c} := \frac{1}{B_c} \sum_{k=1}^{B_c} \sum_{j=1}^{m} \left\| J_{\boldsymbol{x}}\, \widehat{\boldsymbol{g}}_j(\boldsymbol{\xi}_k) - \widehat{\mathrm{H}}_j(\boldsymbol{\xi}_k) \right\|_{\mathrm{F}}^2. \tag{A.2}$$

Given this example, we can generalize Definition 1 to higher-order derivatives, introducing the notion of *k-order Derivative-Informed Neural Network* (*k*DINN).

Definition 2 (*k*-order Derivative-Informed Neural Network)**.** Let $\widehat{\boldsymbol{F}} = \widehat{\boldsymbol{F}}(\cdot\,; \boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}^m$ be a k-times differentiable NN and let $\widehat{\mathrm{F}}^{(\kappa)} := \widehat{\mathrm{F}}^{(\kappa)}(\cdot\,; \boldsymbol{\theta}^{(\kappa)}) : \mathbb{R}^n \to \mathbb{R}^{m \times (\otimes_{h=1}^{\kappa} n)}$ be a $(k-\kappa)$-times differentiable NN, for each $\kappa = 1, \dots, k$. Then, a *k-order Derivative-Informed Neural Network* (*k*DINN) defined by $\widehat{\boldsymbol{F}}$ and $\widehat{\mathrm{F}}^{(1)}, \dots, \widehat{\mathrm{F}}^{(k)}$ for learning the target function $\boldsymbol{F}$ is a NN model $\mathcal{F} = (\widehat{\boldsymbol{F}}, \widehat{\mathrm{F}}^{(1)}, \dots, \widehat{\mathrm{F}}^{(k)}) : \mathbb{R}^n \to \mathbb{R}^m \times \mathbb{R}^{m \times n} \times \cdots \times \mathbb{R}^{m \times (\otimes_{h=1}^{k} n)}$ trained following these criteria:

1. updates the weights $\boldsymbol{\theta}$ for obtaining $\widehat{\boldsymbol{F}} \approx \boldsymbol{F}$;
2. updates all the weights $\boldsymbol{\theta}, \boldsymbol{\theta}^{(1)}, \dots, \boldsymbol{\theta}^{(k)}$ for obtaining

$$\left(\frac{\partial^{\boldsymbol{\alpha}} \widehat{f}_j}{\partial^{\alpha_1} x_1 \cdots \partial^{\alpha_n} x_n} \right)_{\boldsymbol{\alpha} \in M(\kappa)} \approx \widehat{\mathrm{F}}_j^{(\kappa)}, \quad \forall\, j = 1, \dots, m,\ \forall\, \kappa = 1, \dots, k,$$

   where $M(\kappa)$ is the set of all the multi-indices $\boldsymbol{\alpha} \in \mathbb{N}_+^n$ of order κ and $\widehat{\mathrm{F}}_j^{(\kappa)} \in \mathbb{R}^{\otimes_{h=1}^{\kappa} n}$ is the j-th vector/matrix/tensor returned by $\widehat{\mathrm{F}}^{(\kappa)}$;
3. can update the weights $\boldsymbol{\theta}^{(1)}, \dots, \boldsymbol{\theta}^{(k)}$ for obtaining

$$\left(\frac{\partial^{\boldsymbol{\alpha}} \widehat{\mathrm{F}}_j^{(h)}}{\partial^{\alpha_1} x_1 \cdots \partial^{\alpha_n} x_n} \right)_{\boldsymbol{\alpha} \in M(\kappa - h)} \approx \widehat{\mathrm{F}}_j^{(\kappa)}, \quad \forall\, j = 1, \dots, m, \quad \forall\, \kappa = 2, \dots, k, \quad \forall\, h < \kappa\,.$$

Given Definition 2, we have that the NN model $\mathcal{G} = (\widehat{\boldsymbol{F}}, \widehat{\boldsymbol{G}}, \widehat{\mathrm{H}})$ described at the beginning of this subsection is a second-order DINN, if trained with respect to a loss function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\eta}; \mathcal{B}, \mathcal{B}_c) = \mathcal{L}_1(\boldsymbol{\theta}; \mathcal{B}) + \mathcal{L}_2(\boldsymbol{\theta}, \boldsymbol{\eta}; \mathcal{B}_c) + \mathcal{L}_3(\boldsymbol{\theta}, \boldsymbol{\zeta}; \mathcal{B}_c)$$

or a loss function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\eta}; \mathcal{B}, \mathcal{B}_c) = \mathcal{L}_1(\boldsymbol{\theta}; \mathcal{B}) + \mathcal{L}_2(\boldsymbol{\theta}, \boldsymbol{\eta}; \mathcal{B}_c) + \mathcal{L}_3(\boldsymbol{\theta}, \boldsymbol{\zeta}; \mathcal{B}_c) + \mathcal{L}_4(\boldsymbol{\eta}, \boldsymbol{\zeta}; \mathcal{B}_c),$$

where $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$, and $\mathcal{L}_4$ are the losses of Eqs. (3), (4), (A.1), and (A.2), respectively.

While the theoretical framework for integrating higher-order derivatives into GradINNs via additional loss terms has been established, their practical impact on model performance remains to be investigated. Future studies will aim to evaluate the influence of these supplementary loss terms.

Appendix B. Activation functions

This appendix reports additional results obtained by repeating the experiments on the *Friedman* function (Section 3.2) and the *Yacht* dataset (Section 3.3) using different activation functions in the hidden layers of $\widehat{\boldsymbol{F}}$, $\widehat{\boldsymbol{G}}$, and s-NN. In particular, we tested ELU, tanh, GELU, and ReLU activations functions while keeping the same architecture, initialization scheme, and optimization settings as in the main experiments. This analysis aimed to verify whether the performance improvements introduced by GradINN depend on the specific choice of the activation function or remain consistent across different types of smooth and piecewise-linear nonlinearities.

As shown in Figs. B.14 and B.15, GradINN consistently outperformed the baseline NN for all the tested activation functions, confirming that the method's effectiveness is not tied to the sigmoid activation but rather to the incorporation of gradients' priors via the auxiliary NN.

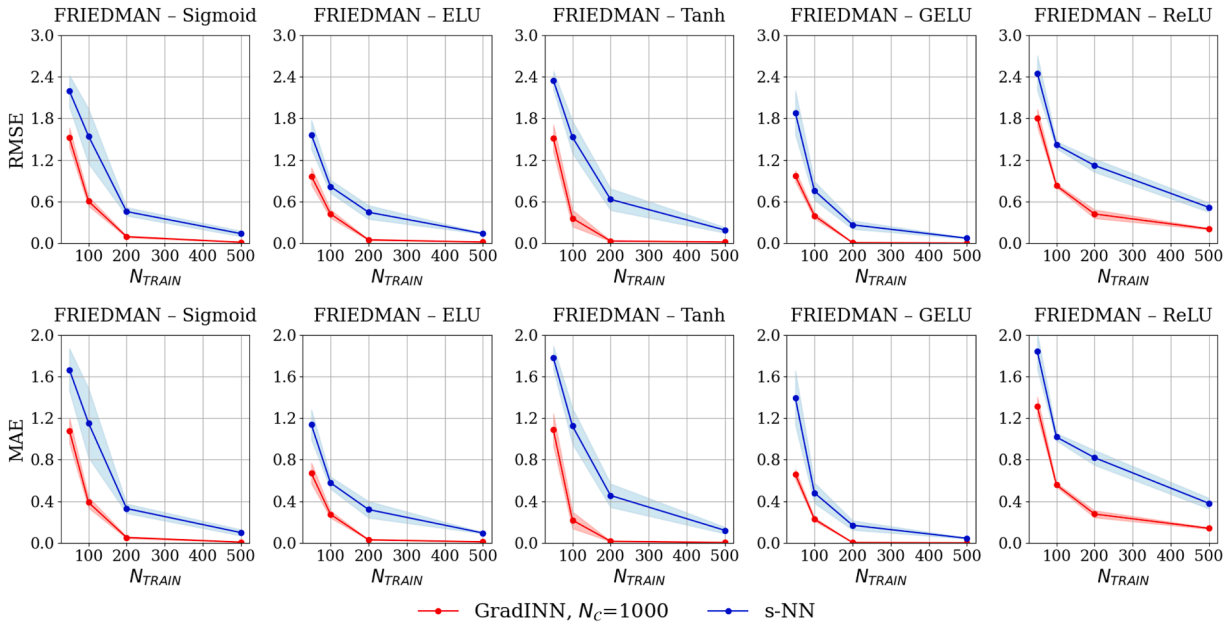

Fig. B.14. FRIEDMAN. Comparison of GradINN and NN across different activation functions. Results are reported in terms of RMSE and MAE as a function of N_{TRAIN}.

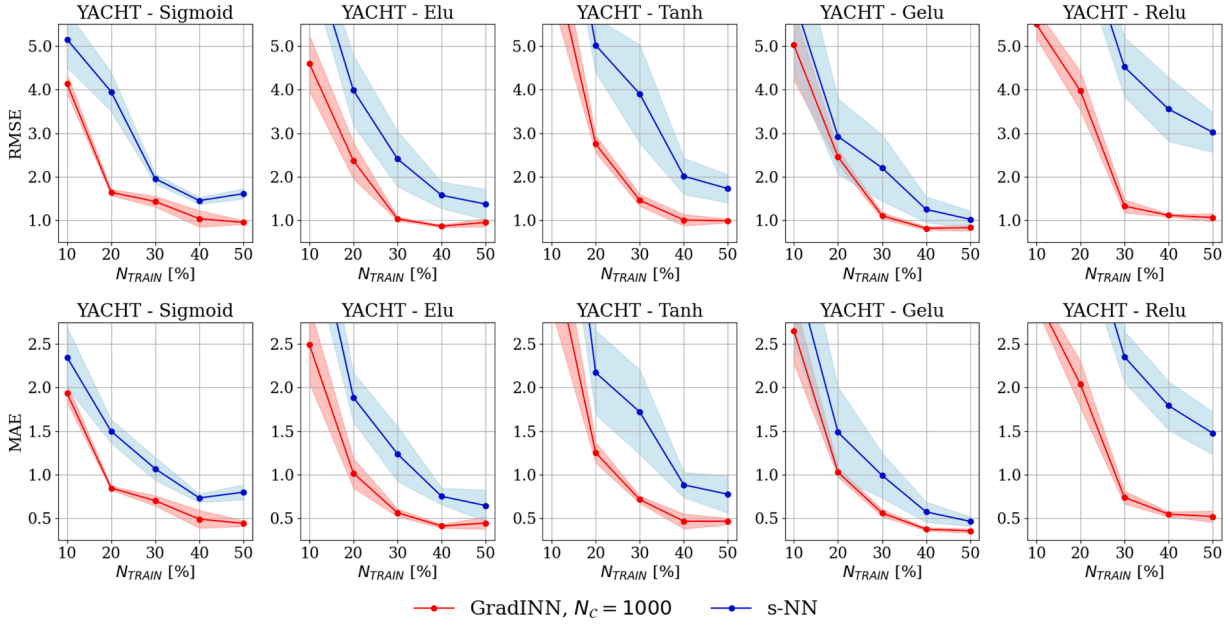

Fig. B.15. YACHT. Comparison of GradINN and NN across different activation functions. Results are reported in terms of RMSE and MAE as a function of N_{TRAIN}.

Appendix C. Functions and datasets

In this appendix, we list the equations characterizing the functions used for the experiments of Section 3.2 and Section 3.5 and describe the datasets of Section 3.3 and Section 3.4.

C.1. Synthetic functions of Section 3.2

- FRIEDMAN:

  $y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$

  for $x_i \in [0, 1]$ with $i = 1, \ldots, 5$.
- GOLDSTEIN-PRICE:

  $y = \left(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\right)$
  $\left(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\right)$

  for $x_1, x_2 \in [-1.5, 1, 5]$.

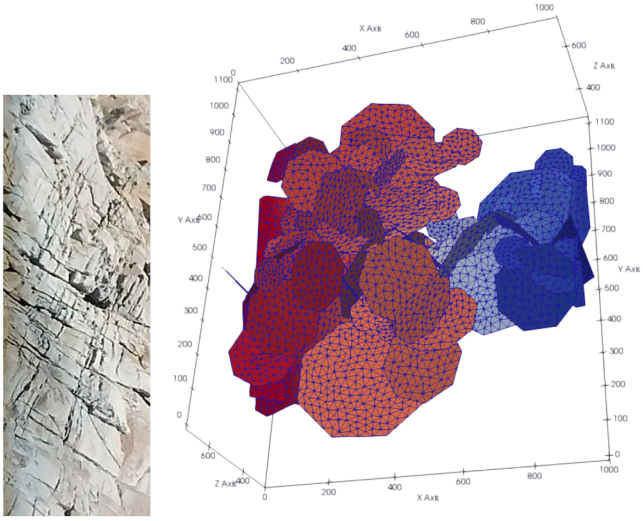

Fig. C.16. External surface of a natural fractured medium (left), a 3D view of the DFN considered in this experiment (right).

- ROSENBROCK:

$$y = \sum_{i=1}^{5} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

for $x_i \in [-2.048, 2.048]$ with $i = 1, \dots, 5$.

- POWELL:

$$y = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

for $x_i \in [-4, 5]$ with $i = 1, \dots, 4$.

- BOOTH:

$$y = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$$

for $x_1, x_2 \in [-10, 10]$.

C.2. UCI datasets of Section 3.3

- The Yacht Hydrodynamics dataset (YACHT), which predicts the hydrodynamic resistance of sailing yachts based on parameters such as velocity and hull dimensions.
- The Airfoil Self-Noise dataset (AIRFOILNOISE), which estimates noise levels generated by airfoils under different operational conditions, such as angle of attack and velocity.
- The Combined Cycle Power Plant dataset (POWER), which predicts the energy output of a power plant based on features such as temperature, pressure, and humidity.
- The Energy Efficiency dataset (ENERGY), which models the heating and cooling load requirements of buildings given architectural and environmental factors.
- The Naval Propulsion Plants dataset (NAVAL), which estimates the condition-based maintenance requirements of naval vessels using sensor measurements.

C.3. High-dimensional dataset of Section 3.4

In this appendix, we briefly introduce the problem and the dataset used in Section 3.4. Summarizing, a Discrete Fracture Network (DFN) model is a discrete representation of an underground network of fractures in a fractured rock medium using a set of intersecting planar polygons (the fractures) in a three-dimensional domain (the rock medium, and the DFN is defined as the union of all the fractures); see Fig. C.16 for a DFN example. Each fracture in a DFN is characterized by a transmissivity parameter that describes its hydrogeological properties. Therefore, flow simulations of the DFN are characterized by these parameters (for each fixed configuration of fractures), that are typically sampled from known probability distributions based on geological properties of the rock medium. For these reasons, NNs are useful to learn specific quantities of interest of the problem (e.g., the fluxes outflowing from a subset of fractures), varying the tramsissivity properties of the DFN's fractures. Further details about the dataset can be found in Berrone and Della Santa (2021), Berrone et al. (2021a,b, 2013a,b, 2014).

C.4. Oscillatory functions of Section 3.5

- ACKLEY:

$$y = -20\exp\left(-0.2\sqrt{0.5(x_1^2 + x_2^2)}\right) - \exp\left(0.5[\cos(2\pi x_1) + \cos(2\pi x_2)]\right) + 20 + e$$

for $x_1, x_2 \in [-5, 5]$.

- RASTRIGIN:

$$y = 20 + x_1^2 + x_2^2 - 10[\cos(2\pi x_1) + \cos(2\pi x_2)]$$

for $x_1, x_2 \in [-5.12, 5.12]$.

- EGGHOLDER:

$$y = -(x_2 + 47)\sin\left(\sqrt{|x_1/2 + x_2 + 47|}\right) - x_1 \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right)$$

for $x_1, x_2 \in [-512, 512]$.

- SCHWEFEL :

$$y = 418.9829 \times 2 - \left[x_1 \sin(\sqrt{|x_1|}) + x_2 \sin(\sqrt{|x_2|})\right]$$

for $x_1, x_2 \in [-500, 500]$.

Appendix D. Complete results for Section 3.2

In this appendix, we report the tables containing all the performances of the GradINNs and the s-NNs with respect to the sythetic functions of Section 3.2. (Tables D.2, D.3, D.4, D.5, D.6, D.7, D.8, D.9, D.10, D.11)

Table D.2
FRIEDMAN. RMSE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_c = 50$	2.008 ± 0.183	0.845 ± 0.077	0.236 ± 0.032	0.056 ± 0.007
	$N_c = 100$	1.956 ± 0.137	0.717 ± 0.080	0.208 ± 0.030	0.045 ± 0.005
	$N_c = 250$	1.813 ± 0.127	0.651 ± 0.045	0.155 ± 0.020	0.033 ± 0.002
	$N_c = 500$	1.727 ± 0.097	0.639 ± 0.030	0.126 ± 0.025	0.021 ± 0.002
	$N_c = 1000$	**1.523 ± 0.141**	**0.604 ± 0.067**	**0.094 ± 0.013**	**0.014 ± 0.001**
s-NN	-	2.193 ± 0.228	1.545 ± 0.393	0.456 ± 0.060	0.138 ± 0.039

Table D.3
FRIEDMAN. MAE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_c = 50$	1.488 ± 0.158	0.593 ± 0.073	0.160 ± 0.023	0.039 ± 0.005
	$N_c = 100$	1.425 ± 0.115	0.496 ± 0.068	0.142 ± 0.022	0.030 ± 0.003
	$N_c = 250$	1.314 ± 0.098	0.437 ± 0.043	0.101 ± 0.014	0.021 ± 0.002
	$N_c = 500$	1.254 ± 0.072	0.424 ± 0.024	0.076 ± 0.017	0.012 ± 0.002
	$N_c = 1000$	**1.080 ± 0.120**	**0.389 ± 0.053**	**0.052 ± 0.009**	**0.008 ± 0.001**
s-NN	-	1.665 ± 0.204	1.153 ± 0.332	0.331 ± 0.045	0.099 ± 0.029

Table D.4
GOLDSTEIN-PRICE. RMSE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_c = 50$	10,828 ± 1692	5901 ± 377	2532 ± 247	842 ± 72
	$N_c = 100$	9661 ± 1066	5402 ± 382	1591 ± 200	642 ± 77
	$N_c = 250$	8508 ± 596	5598 ± 346	1638 ± 150	456 ± 81
	$N_c = 500$	8394 ± 639	**4287 ± 360**	1541 ± 130	339 ± 36
	$N_c = 1000$	**8322 ± 640**	**4319 ± 312**	**1358 ± 185**	**235 ± 40**
s-NN	-	14,144 ± 556	12,012 ± 776	3951 ± 474	1055 ± 85

Table D.5
GOLDSTEIN-PRICE. MAE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	3408 ± 404	2292 ± 104	848 ± 134	526 ± 42
	$N_C = 100$	3549 ± 303	2079 ± 142	637 ± 74	391 ± 66
	$N_C = 250$	2810 ± 302	2109 ± 154	490 ± 42	276 ± 45
	$N_C = 500$	**2614 ± 391**	1365 ± 163	463 ± 25	202 ± 26
	$N_C = 1000$	**2633 ± 287**	**1290 ± 135**	**379 ± 61**	**127 ± 22**
s-NN	-	4529 ± 1009	5399 ± 162	1203 ± 111	674 ± 59

Table D.6
ROSENBROCK. RMSE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	1115 ± 12	690 ± 83	527 ± 17	323 ± 15
	$N_C = 100$	1115 ± 7	639 ± 33	512 ± 37	297 ± 27
	$N_C = 250$	1082 ± 36	649 ± 18	469 ± 25	242 ± 28
	$N_C = 500$	1093 ± 17	610 ± 54	475 ± 21	197 ± 38
	$N_C = 1000$	1126 ± 26	**594 ± 30**	**431 ± 26**	**65 ± 25**
s-NN	-	1222 ± 89	961 ± 250	589 ± 53	362 ± 11

Table D.7
ROSENBROCK. MAE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	825 ± 15	480 ± 55	367 ± 15	223 ± 19
	$N_C = 100$	831 ± 12	441 ± 23	355 ± 23	196 ± 22
	$N_C = 250$	822 ± 34	449 ± 15	327 ± 13	157 ± 16
	$N_C = 500$	**816 ± 14**	403 ± 37	322 ± 18	120 ± 23
	$N_C = 1000$	**818 ± 24**	**407 ± 27**	**299 ± 23**	**35 ± 15**
s-NN	-	941 ± 85	723 ± 201	420 ± 46	259 ± 8

Table D.8
POWELL. RMSE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	6495.82 ± 591.53	3757.64 ± 353.11	1701.78 ± 226.76	619.99 ± 129.08
	$N_C = 100$	6052.32 ± 248.77	3568.62 ± 369.97	1514.93 ± 133.39	514.33 ± 82.18
	$N_C = 250$	5862.02 ± 366.76	3544.80 ± 342.87	1124.94 ± 171.40	434.98 ± 88.04
	$N_C = 500$	5996.96 ± 227.02	3003.57 ± 343.33	946.77 ± 78.84	432.49 ± 69.03
	$N_C = 1000$	**5706.68 ± 202.47**	**2989.64 ± 400.28**	**795.20 ± 48.15**	**210.16 ± 11.36**
s-NN	-	6418.85 ± 444.89	5046.47 ± 1657.54	2133.95 ± 287.40	1040.28 ± 421.44

Table D.9
POWELL. MAE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	3308.32 ± 472.75	1751.33 ± 208.93	592.60 ± 113.48	215.72 ± 72.36
	$N_C = 100$	2930.33 ± 89.82	1646.64 ± 313.89	501.05 ± 63.13	146.88 ± 24.06
	$N_C = 250$	2893.97 ± 164.31	1578.40 ± 304.19	366.89 ± 103.98	96.90 ± 15.54
	$N_C = 500$	2909.31 ± 116.64	1181.43 ± 191.85	265.54 ± 27.29	108.17 ± 50.18
	$N_C = 1000$	**2709.34 ± 81.81**	**1072.99 ± 186.91**	**210.17 ± 7.66**	**43.31 ± 5.18**
s-NN	-	3219.97 ± 341.50	2816.21 ± 1271.74	1019.54 ± 221.12	546.73 ± 223.12

Table D.10
BOOTH. RMSE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	19.78 ± 8.95	5.71 ± 0.65	2.43 ± 0.46	1.65 ± 0.33
	$N_C = 100$	16.99 ± 7.43	2.94 ± 0.76	1.19 ± 0.15	0.84 ± 0.19
	$N_C = 250$	8.95 ± 6.15	1.92 ± 0.25	0.60 ± 0.16	0.48 ± 0.08
	$N_C = 500$	10.06 ± 3.84	1.82 ± 0.53	0.47 ± 0.06	0.46 ± 0.07
	$N_C = 1000$	**8.55 ± 4.30**	**1.48 ± 0.23**	**0.36 ± 0.04**	**0.43 ± 0.12**
s-NN	-	39.06 ± 6.68	13.94 ± 3.19	5.33 ± 0.53	4.02 ± 0.47

Table D.11
BOOTH. MAE comparison. The best results are highlighted in bold.

		N_{TRAIN}			
		50	100	200	500
GradINN	$N_C = 50$	9.06 ± 3.94	2.37 ± 0.44	1.69 ± 0.34	0.67 ± 0.09
	$N_C = 100$	7.45 ± 5.17	1.66 ± 0.61	0.79 ± 0.08	0.42 ± 0.07
	$N_C = 250$	2.67 ± 1.37	1.01 ± 0.07	0.41 ± 0.10	0.25 ± 0.03
	$N_C = 500$	3.72 ± 2.38	0.91 ± 0.23	0.33 ± 0.06	0.18 ± 0.02
	$N_C = 1000$	**2.31 ± 0.79**	**0.67 ± 0.10**	**0.23 ± 0.02**	**0.14 ± 0.01**
s-NN	-	16.76 ± 6.67	8.14 ± 3.23	3.75 ± 0.58	1.93 ± 0.41

References

Noguer i Alonso, M., & Maxwell, D. (2023). Physics-informed neural networks (PINNs) in finance. *Daniel, Physics-Informed Neural Networks (PINNs) in Finance (October 10, 2023)*, .

Baydin, A., Pearlmutter, B., Radul, A., & Siskind, J. (2018). Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, *18*, 1–43.

Berrone, S., & Della Santa, F. (2021). Performance analysis of multi-task deep learning models for flux regression in discrete fracture networks. *Geosciences*, *11*, 131. https://doi.org/10.3390/geosciences11030131

Berrone, S., Della Santa, F., Mastropietro, A., Pieraccini, S., & Vaccarino, F. (2021a). Layer-wise relevance propagation for backbone identification in discrete fracture networks. *Journal of Computational Science*, *55*, 101458. 10.1016/j.jocs.2021.101458. https://doi.org/10.1016/j.jocs.2021.101458

Berrone, S., Della Santa, F., Pieraccini, S., & Vaccarino, F. (2021b). Machine learning for flux regression in discrete fracture networks. *GEM - International Journal on Geomathematics*, *12*(1), 9. https://doi.org/10.1007/s13137-021-00176-0.

Berrone, S., Pieraccini, S., & Scialò, S. (2013a). On simulations of discrete fracture network flows with an optimization-based extended finite element method. *SIAM Journal on Scientific Computing*, *35*(2), A908–A935. https://doi.org/10.1137/120882883.

Berrone, S., Pieraccini, S., & Scialò, S. (2013b). A PDE-constrained optimization formulation for discrete fracture network flows. *SIAM Journal on Scientific Computing*, *35*(2), B487–B510. https://doi.org/10.1137/120865884.

Berrone, S., Pieraccini, S., & Scialò, S. (2014). An optimization approach for large scale simulations of discrete fracture network flows. *Journal of Computational Physics*, *256*, 838–853. https://doi.org/10.1016/j.jcp.2013.09.028

Cai, S., Mao, Z., Wang, Z., Yin, M., & Karniadakis, G. E. (2021a). Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, *37*(12), 1727–1738.

Cai, S., Wang, Z., Wang, S., Perdikaris, P., & Karniadakis, G. E. (2021b). Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, *143*(6), 060801. https://doi.org/10.1115/1.4050542

Choi, D., An, Y., Lee, N., Park, J., & Lee, J. (2020). Comparative study of physics-based modeling and neural network approach to predict cooling in vehicle integrated thermal management system. *Energies*, *13*(20). https://doi.org/10.3390/en13205301

Czarnecki, W. M., Osindero, S., Jaderberg, M., Swirszcz, G., & Pascanu, R. (2017). Sobolev training for neural networks. *CoRR*, *abs/1706.04859*. http://arxiv.org/abs/1706.04859.

Dua, D., & Graff, C. (2017). UCI machine learning repository. http://archive.ics.uci.edu/ml.

Ferrara, M., Della Santa, F., Bilardo, M., De Gregorio, A., Mastropietro, A., Fugacci, U., Vaccarino, F., & Fabrizio, E. (2021). Design optimization of renewable energy systems for NZEBs based on deep residual learning. *Renewable Energy*, . https://doi.org/10.1016/j.renene.2021.05.044

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research*, *9*, 249–256.

Greydanus, S., Dzamba, M., & Yosinski, J. (2019). Hamiltonian neural networks.

Griewank, A. (1989). On automatic differentiation. In *In mathematical programming: recent developments and applications* (pp. 83–108). Kluwer Academic Publishers.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 770–778).

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8

Huang, B., & Wang, J. (2023). Applications of physics-informed neural networks in power systems - a review. *IEEE Transactions on Power Systems*, *38*(1), 572–588. https://doi.org/10.1109/TPWRS.2022.3162473

Ishikawa, I., Teshima, T., Tojo, K., Oono, K., Ikeda, M., & Sugiyama, M. (2023). Universal approximation property of invertible neural networks. *Journal of Machine Learning Research*, *24*(287), 1–68. http://jmlr.org/papers/v24/22-0384.html.

Jagannath, A., & Jagannath, J. (2021). Multi-task learning approach for automatic modulation and wireless signal classification. In *Proc. of IEEE international conference on communications (ICC)*. Montreal, Canada.

Karniadakis, G., Kevrekidis, Y., Lu, L., Perdikaris, P., Wang, S., & Yang, L. (2021). Physics-informed machine learning, . (pp. 1–19). https://doi.org/10.1038/s42254-021-00314-5

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio, & Y. LeCun (Eds.), *3Rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7–9, 2015, conference track proceedings*. http://arxiv.org/abs/1412.6980.

Kumar, R., & Dhua, S. (2025). Dynamic analysis of hashimoto's thyroiditis biomathematical model using artificial neural network. *Mathematics and Computers in Simulation*, *229*, 235–245. https://doi.org/10.1016/j.matcom.2024.10.001

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, *16*(2), 146–160. https://doi.org/10.1007/BF01931367

Mao, Z., Jagtap, A. D., & Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, *360*, 112789. https://doi.org/10.1016/j.cma.2019.112789

Mattheakis, M., Sondak, D., Dogra, A. S., & Protopapas, P. (2022). Hamiltonian neural networks for solving equations of motion. *Physical Review E*, *105*(6). https://doi.org/10.1103/physreve.105.065305

Oreski, S. (2012). Comparison of neural network and empirical models for prediction of second virial coefficients for gases. *Procedia Engineering*, *42*, 303–312. CHISA 2012. https://doi.org/10.1016/j.proeng.2012.07.421

Pilar, P., & Wahlström, N. (2023). Physics-informed neural networks with unknown measurement noise.

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, *8*, 143-195. https://doi.org/10.1017/S0962492900002919

Podina, L., Eastman, B., & Kohandel, M. (2023). Universal physics-informed neural networks: Symbolic differential operator discovery with sparse data. In *Proceedings of the 40th international conference on machine learning* ICML'23. JMLR.org.

Raissi, M. (2018). Deep hidden physics models: Deep learning of nonlinear partial differential equations. *Journal of Machine Learning Research*, *19*.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

Reddy, J. N. (2013). An introduction to the finite element method (vol. 3). McGraw-Hill New York.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Saha, N., Swetapadma, A., & Mondal, M. (2023). A brief review on artificial neural network: Network structures and applications. In *2023 9th international conference on*

advanced computing and communication systems (ICACCS) (pp. 1974–1979). (*vol. 1*). https://doi.org/10.1109/ICACCS57279.2023.10112753

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., & Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 7462–7473). Curran Associates, Inc. (*vol. 33*).

Son, H., Jang, J. W., Han, W. J., & Hwang, H. J. (2021). Sobolev training for physics informed neural networks. arXiv preprint arXiv:2101.08932, .

Yu, J., Lu, L., Meng, X., & Karniadakis, G. E. (2022). Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Computer Methods in Applied Mechanics and Engineering*, *393*, 114823. https://doi.org/10.1016/j.cma.2022.114823