# The limitation of neural nets for approximation and optimization

T. Giovannelli* O. Sohab† L. N. Vicente‡

October 19, 2024

**Abstract**

We are interested in assessing the use of neural networks as surrogate models to approximate and minimize objective functions in optimization problems. While neural networks are widely used for machine learning tasks such as classification and regression, their application in solving optimization problems has been limited. Our study begins by determining the best activation function for approximating the objective functions of popular nonlinear optimization test problems, and the evidence provided shows that ReLU and SiLU exhibit the best performance on both training and testing data. We then analyze the accuracy of function value, gradient, and Hessian approximations for such objective functions obtained through interpolation/regression models and neural networks. When compared to interpolation/regression models, neural networks can deliver competitive zero- and first-order approximations (at a high training cost) but underperform on second-order approximation. However, it is shown that combining a neural net activation function with the natural basis for quadratic interpolation/regression can waive the necessity of including cross terms in the natural basis, leading to models with fewer parameters to determine. Lastly, we provide evidence that the performance of a state-of-the-art derivative-free optimization algorithm can hardly be improved when the gradient of an objective function is approximated using any of the surrogate models considered, including neural networks.

## 1 Introduction

In recent years, machine learning (ML) models have been used to enhance optimization algorithms with the goal of improving their performance. One popular approach is to use ML models to approximate the objective function being minimized. These models, called surrogate models, can learn from past evaluations of the objective function and predict its value for new inputs (see the review in [71]). Using surrogate models can be particularly useful for optimization problems where each evaluation of the objective function is time-consuming or computationally expensive, like in simulation-based optimization or derivative-free optimization (DFO) [2, 20, 22, 45]. Once an accurate and computationally cheap surrogate model is built, one can evaluate such a model instead of the true objective function to reduce the number of objective function evaluations, thus improving optimization efficiency. Another approach is to use ML models to learn the

*Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015-1582, USA (tog220@lehigh.edu).

†Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015-1582, USA (ous219@lehigh.edu).

‡Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA 18015-1582, USA (lnv@lehigh.edu).

optimization process itself. These models can iteratively learn from past optimization runs and predict the best algorithmic steps and hyperparameters of methods applied to solve a given problem. For example, in [15, 66], the authors train recurrent neural networks to learn the update steps of gradient-based and derivative-free optimization algorithms. Such networks are then used to predict the next point to evaluate without adding to the number of objective function and gradient evaluations.

For the scope of this paper, among all the different types of ML models, we focus on artificial neural networks. Such networks consist of interconnected nodes (also called neurons) that are organized into layers, typically consisting of an input layer, one or more hidden layers, and an output layer [31]. The inputs to the network are fed into the input layer, and then they are passed through the hidden layers, where each neuron receives inputs from the previous layer, computes a weighted sum of inputs, and then outputs the result to the next layer after applying an activation function. The output layer produces the final output of the network. Among all the different types of artificial neural networks, we consider feedforward neural networks, where the inputs are processed in a forward direction only, without any feedback connections.

The ultimate goal of this paper is to assess the use of neural networks as surrogate models for approximation and optimization purposes. Neural networks have gained significant popularity as surrogate models in engineering applications [13, 52, 57, 58, 70]. However, we are not aware of any papers evaluating the accuracy of the approximations produced by neural networks or using neural networks within optimization algorithms to approximate the objective function being minimized. For assessing the performance of neural networks, we choose to compare them against models built through interpolation or regression, including quadratic models [5, 10, 17, 18, 19, 20, 79] and interpolation models based on radial basis functions [14, 35, 51], which have successfully been used for approximation and optimization. To conduct our study, we consider popular nonlinear optimization test problems with different features in terms of linearity, convexity, and separability. It is important to note that our focus in this paper is both on the local and global behavior of the approximations provided by the surrogate models considered. In particular, we are interested in the accuracy of gradient and Hessian approximations for a given point in the domain and in the accuracy of function value approximations across a larger set of points.

The main contributions of this paper can be summarized as follows:

1. Determine the best activation function of neural networks used to approximate the objective functions of popular nonlinear optimization test problems. The evidence provided shows that ReLU and SiLU exhibit the best performance on both training and testing data (see Section 2). Our aim is not to determine an activation function designed for a specific optimization problem. Instead, we aim to find an activation function that performs well across a wide range of problems. Note that optimization problems have not been previously used to assess activation functions, as researchers developing new activation functions typically limit experiments to classification or regression tasks, where neural networks are used to predict labels or responses. For classification tasks, popular datasets such as MNIST and CIFAR are frequently used [11, 16, 41, 50, 78]. For regression tasks, fewer studies have been conducted, and most of them either focus on applications arising in fields like chemistry and biology [64, 78] or consider determining approximate solutions to partial differential equations using physics-informed approaches [39]. Some papers also investigate activation functions in reinforcement learning tasks [25, 78], where neural networks are used to learn an optimal policy.

2. Analyze the accuracy of the function value, gradient, and Hessian approximations for the objective functions in the test problems obtained using interpolation/regression models, including quadratic models and interpolation models based on radial basis functions, and neural networks. The evidence provided shows that the interpolation/regression models provide better accuracy when they are used for second-order approximations. Neural networks are competitive in terms of function evaluations required for zero- and first-order approximations (although at a high training cost) but perform below interpolation/regression models in terms of second-order approximations. Interestingly, the results also show that combining an activation function with the natural basis for quadratic interpolation/regression can waive the necessity of including cross terms in the natural basis, leading to models with fewer parameters to determine (see Section 3).

3. Present evidence that the performance of a state-of-the-art DFO algorithm can hardly be improved when the gradient of the objective functions in the test problems is approximated using any of the surrogate models considered, including neural networks (see Section 4). When doing finite-difference BFGS, it seems that no surrogate modeling helps, including a magical/search step based on surrogate optimization.

In addition to such contributions, we include a discussion in Section 5 to generalize the numerical results obtained in our paper and provide key takeaway messages.

Throughout the paper, we focus on unconstrained optimization problems with a continuously differentiable objective function $f : \mathbb{R}^n \to \mathbb{R}$, namely,

$$\min_{x \in \mathbb{R}^n} f(x). \tag{1.1}$$

The derivatives of $f$ will be computed for assessing the accuracy of the approximations provided by surrogate models. However, they will be assumed unavailable when solving problem (1.1) in Section 4.

All the computational experiments described in this paper were run on a Linux server with 32 GB of RAM and an AMD Opteron 6128 processor running at 2.00 GHz. For our implementation, we used the PyTorch library available in Python [59] and the code is publicly available on GitHub.* Throughout this document, $k$ will be the index used to denote the iterations and $\|\cdot\|$ will be the Euclidean norm. Moreover, given a positive scalar $\Delta$, $B(x;\Delta) = \{y \in \mathbb{R}^n : \|y - x\| \leq \Delta\}$ will denote a closed ball in $\mathbb{R}^n$ of radius $\Delta > 0$.

## 2 Best activation function for approximation

In ML applications, neural networks are widely used for function approximation tasks, such as regression and classification. Activation functions play a critical role in the performance of a neural network model, affecting its ability to approximate functions appearing in complex models. In this section, we provide a review of popular activation functions in ML (see Subsection 2.1) and determine the best activation function for approximating the objective functions of popular nonlinear optimization test problems (see Subsection 2.2).

*`https://github.com/sohaboumaima/BasesNNApproxForOpt.git`

### 2.1 Review of popular activation functions in machine learning

Let us denote the activation function used in a neuron of a neural network as $s$: $\mathbb{R} \to \mathbb{R}$. Although linear activation functions can be considered (i.e., $s(z) = z$), the activation functions used in practice apply a nonlinear transformation to the inputs of the corresponding neurons. ReLU [41], ELU [16], SiLU [25], Sigmoid [64, 65, 67], and Tanh [42] are some of the most commonly used activation functions in neural networks, and their properties are reviewed below.

ReLU is defined by $s(z) = \max\{0, z\}$ and is the recommended activation function for modern neural networks [31, Chapter 6]. Although ReLU is not differentiable at $z = 0$, this is usually not problematic in practice since it is unlikely that the derivative of $g$ needs to be precisely evaluated at $z = 0$ because of numerical errors. Even if such a derivative needs to be evaluated at $z = 0$, software implementations usually provide the left derivative (which is 0) or the right derivative (which is 1) without raising any issues [31, Chapter 6]. Smooth approximations of ReLU include ELU, defined by $s(z) = z$ for $z > 0$ and $s(z) = \alpha(e^z - 1)$ for $z \leq 0$, where $\alpha > 0$, and SiLU, defined by $s(z) = z/(1 + e^{-z})$. Another smooth variant is Softplus [24, 80], which is given by $s(z) = \log(1 + e^z)$. However, although Softplus is differentiable everywhere, its use is discouraged because ReLU is still more likely to lead to better results [29].

The main drawback of ReLU is that it can suffer from the "dying ReLU" problem, which occurs when neurons output 0 because of their negative inputs and become permanently inactive during training [47]. To overcome such an issue, [49] proposed LeakyReLU, which is defined by $s(x) = \max\{0.01z, z\}$. The non-negative slope of LeakyReLU for all $z < 0$ allows neurons to contribute to the network's output even if their inputs are negative. In practice, such an activation function performs comparably to ReLU and, occasionally, it may perform better. An improvement to LeakyReLU is Parametric ReLU [36, 77], where the slope for $z < 0$ is a parameter that needs to be learned while training a neural network. ELU and SiLU are known to be robust to the dying ReLU issue [40].

Sigmoid and Tanh have been used extensively in the past but have become largely replaced by ReLU and its variants. Sigmoid, defined by $s(z) = 1/(1 + e^{-z})$, maps each input to $[0, 1]$ and is often used to approximate probabilities. Tanh, defined by $s(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, is a rescaled version of Sigmoid that maps input values to $[-1, 1]$. By following a common convention [31], we will refer to Sigmoid and Tanh as the sigmoidal activation functions because they are both s-shaped. The main limitation of the sigmoidal activation functions is that they exhibit slow convergence or get stuck at points that are far from optimality. The reason for this behavior is that both Sigmoid and Tanh become flat when $z$ is very positive or very negative, which leads to derivatives that are close to 0, preventing gradient-based algorithms from making good progress at each iteration (in the ML literature, this issue is referred to as the vanishing gradient problem or the saturation problem). While Tanh can perform better than Sigmoid in some cases, it still suffers from the vanishing gradient problem for large (positive or negative) input values. For this reason, the use of Sigmoid and Tanh as activation functions is discouraged [31, Chapter 6]. As opposed to the sigmoidal activation functions, ReLU and its variants have derivatives equal to 1 for $z > 0$, and this allows gradient-based algorithms to make significant progress in the minimization of the objective function at each iteration.

Recently, some papers have explored combinations of activation functions or have proposed new activation functions that try to replicate the best properties of ReLU and its variants, Sigmoid, and Tanh. In [50], two approaches to automatically learn the best linear combination of activation functions during the training phase are proposed. In [78], the authors propose a differ-

entiable universal activation function with 5 trainable parameters that allow it to approximate the most commonly used activation functions in ML. In [11], activation functions are represented as nodes of a computation graph, and evolutionary search and gradient descent are then used to determine the general form of the best activation function and its parameters, respectively. One of the potential drawbacks of using activation functions with trainable parameters is that it can result in overfitting the training data, leading to poor performance on unseen data. Another potential downside is the extra computational cost of training the model. Since the additional parameters need to be learned during training, the computational complexity and time required for training the model can increase [36, 68]. Therefore, in the numerical experiments reported in Subsection 2.2 below, we do not consider activation functions with trainable parameters and we only focus on ReLU, ELU, SiLU, Sigmoid, and Tanh.

### 2.2 Numerical experiments

We conducted experiments to compare the performance of ReLU, ELU, SiLU, Sigmoid, and Tanh in the approximation of test functions from two sets of nonlinear unconstrained optimization problems available in the CUTEst library [32]. A first set is composed of 38 problems with user-defined dimensions (in all the experiments reported in this paper, we considered $n \in \{20, 40, 60\}$), and a second set is composed of 53 functions with dimensions ranging from 2 to 50. The first set includes problems with different features in terms of non-linearity, non-convexity, and partial separability (see Table 1 in Appendix A for the names of such problems). The second set was selected from [34] and contains problems for which negative curvature was detected by running one of the second-order methods reported in [4] (see Table 2 in Appendix A for the names and corresponding dimensions of such problems).

To determine the best activation function for approximation, we used a neural network as a surrogate model for the objective function of each test problem. The weights of the neural network are denoted as $w \in \mathbb{R}^{n_w}$. For each test problem, we trained the neural network to minimize the mean squared error between the objective function $f(x)$ and the surrogate function $f_{\text{NN}}(x; w)$ over the points $x^i$ in a training dataset $\mathcal{D} = \{(x^i, f(x^i)) \mid i \in \{0, \ldots, N\}\}$. Therefore, we solved the following training problem

$$\min_{w \in \mathbb{R}^{n_w}} \mathcal{L}(w; \mathcal{D}) \;=\; \frac{1}{N} \sum_{i=0}^{N} (f(x^i) - f_{\text{NN}}(x^i; w))^2, \tag{2.1}$$

where $\mathcal{L}(w; \mathcal{D})$ denotes the empirical risk function over the dataset $\mathcal{D}$, which is given by the mean squared error. Problem (2.1) was solved five times for each test problem to assess the performance of ReLU, ELU, SiLU, Sigmoid, and Tanh as activation functions in the hidden layers of the neural network used as a surrogate model (in the output layer, a linear activation function was used). In addition to the training dataset $\mathcal{D}$, we also considered a testing dataset to assess the performance of the surrogate model on data that was not used for training.

To aggregate the results over all the test problems, we used performance profiles [23, 53], which are briefly reviewed in this paragraph. Given a set of solvers $\mathcal{S}$ and a set of problems $\mathcal{P}$, let $t_{p,s} > 0$ be the performance measure of the solver $s \in \mathcal{S}$ when applied to solve the problem $p \in \mathcal{P}$. For each solver, we can compute the performance profile $\rho_s(\alpha)$ as follows

$$\rho_s(\alpha) \;=\; \frac{1}{|\mathcal{P}|} \text{size}\{p \in \mathcal{P} \mid r_{p,s} \leq \alpha\},$$

where $r_{p,s}$ is the performance ratio, defined as

$$r_{p,s} \;=\; \frac{t_{p,s}}{\min\{t_{p,s} \mid s \in \mathcal{S}\}}.$$

When a solver $s$ fails to satisfy the convergence test on a problem $p$, we set $r_{p,s} = 2 \max_{p\in\mathcal{P}, s\in\mathcal{S}} r_{p,s}$. One can plot $\rho_s(\alpha)$ as a curve over $\alpha$. The solver associated with the highest curve is the one with the best performance in terms of the metric chosen. In particular, the solver with the highest value of $\rho_s(1)$ is the best in terms of efficiency, while the one with the highest value of $\rho_s(\alpha)$, for large $\alpha$, is the best in terms of robustness.

In this subsection, we assume that a neural network equipped with an activation function in the hidden layers plays the role of a solver. Considering the five activation functions above (i.e., ReLU, ELU, SiLU, Sigmoid, and Tanh) leads to five different neural networks. When using the training dataset, for each test problem $p$, the performance measure $t_{p,s}$ is given by the number of iterations required for the training of each neural network $s$ to achieve a point $w$ that satisfies the following convergence test

$$\mathcal{L}(w^0;\mathcal{D}) - \mathcal{L}(w;\mathcal{D}) \;\geq\; (1-\tau)(\mathcal{L}(w^0;\mathcal{D}) - \mathcal{L}_L(\mathcal{D})), \tag{2.2}$$

where $\tau > 0$ is a convergence tolerance, $w^0$ is the initial vector of weights for each neural network, and $\mathcal{L}_L(\mathcal{D})$ is the lowest value of $\mathcal{L}(w;\mathcal{D})$ obtained by any neural network in $\mathcal{S}$. When using the testing dataset, $t_{p,s}$ represents the number of iterations required to satisfy the convergence test (2.2) with the empirical risk function evaluated over the testing dataset.

The numerical results of our experiments were obtained using the two simplest feedforward neural network architectures that led to the best results among the five considered neural networks. Such architectures are listed as follows

- Varying numbers of nodes: 3 hidden layers with decreasing numbers of nodes given by $64m$ for the first hidden layer, $32m$ for the second hidden layer, and $16m$ for the third hidden layer,[†] where $m = 1$ if $n \in [1, 20]$, $m = 2$ if $n \in (20, 40]$, $m = 3$ if $n \in (40, 60]$, with $n$ denoting the dimension of a test problem.

- Fixed numbers of nodes: 2 hidden layers with $4n$ nodes for each layer,[‡] where $n$ denotes the dimension of a test problem.

For ReLU, ELU, and SiLU, the best architecture was the one with varying numbers of nodes. For Sigmoid and Tanh, the best architecture was the one with fixed numbers of nodes. The reasons why we focused on simple neural network architectures are outlined in Section 5 among the key takeaway messages of our paper. Here, we anticipate that in Section 4, we will use neural networks as surrogate models within derivative-free optimization (DFO) algorithms to approximate the objective function being minimized. Using more complex architectures would require a larger number of training points for the neural network surrogate to achieve satisfactory accuracy, a scenario we aim to avoid to preserve the efficiency of the optimization algorithm.

[†]We also tried configurations with $32m$ nodes in the first hidden layer, $16m$ nodes in the second hidden layer, and $8m$ nodes in the third hidden layer, but they performed worse.

[‡]We also tried configurations with 2 hidden layers and either $n$, $2n$, $3n$, or $4n$ nodes for each layer and with 3 hidden layers and either $n$ or $2n$ nodes for each layer, but they performed worse.

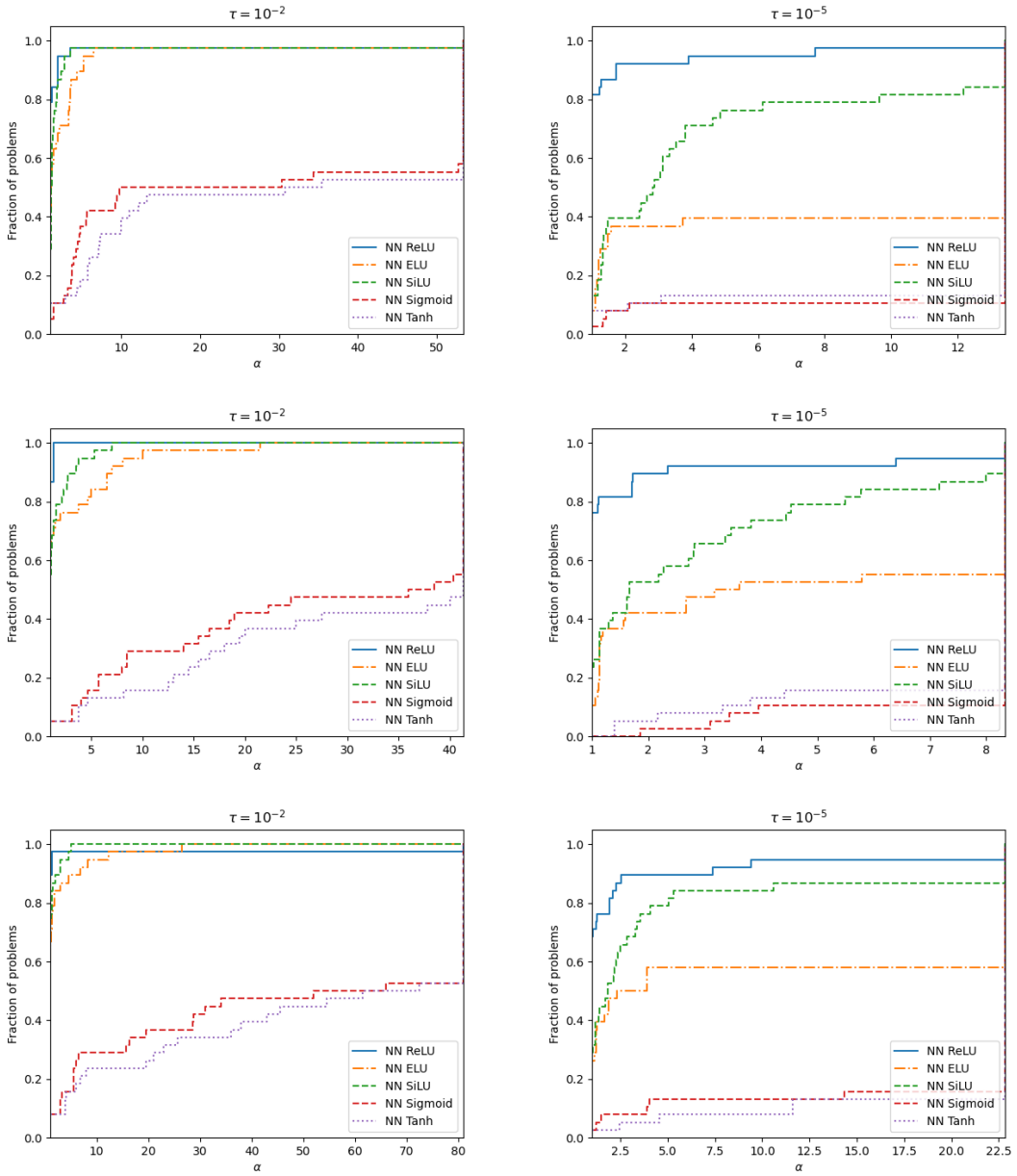


Figure 1: Performance profiles on the training dataset for $n = 20$ (first row), $n = 40$ (second row), and $n = 60$ (third row) with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ for five neural networks with different activation functions on the set of 38 problems from CUTEst.

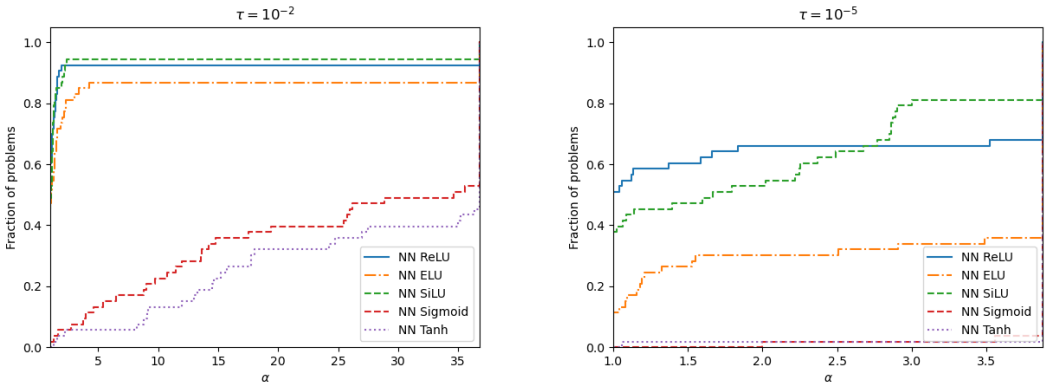


Figure 2: Performance profiles on the training dataset with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ for five neural networks with different activation functions on the set of 53 problems from CUTEst.

We trained the five networks resulting from the five activation functions by using Adam optimizer [43] with 300 epochs. We employed a dynamic learning rate adjustment strategy. Specifically, we used the ReduceLROnPlateau callback, a popular technique in deep learning [1], which led to better performance than using fixed or other decaying stepsize strategies. The learning rate was reduced by a factor of 0.8 when the empirical risk function over the testing dataset ceased to decrease, indicating a potential convergence slowdown. We configured the callback by setting the mode parameter equal to 'min', as we aimed to minimize the testing empirical risk function. Additionally, a patience value of 15 epochs was set, which defines the number of epochs with no improvement on the testing empirical risk function before the learning rate reduction is triggered. The training dataset $\mathcal{D}$ was composed of $N_1 = N + 1 = (n+1)(n+2)/2$ points randomly generated according to a uniform distribution in a ball $B(x^0; 1)$, where $x^0$ is the initial point provided by CUTEst, and we refer to Section 3 (where we will see that $(n+1)(n+2)/2$ is the number of points required for determined quadratic interpolation). The testing dataset was composed of $0.2(n+1)(n+2)/2$ points randomly generated according to a uniform distribution in the same ball $B(x^0; 1)$ used for the training dataset. We have no interest in exploring variations in the training dataset size for the following reasons. As already mentioned, in Section 4, we conduct a comparison between a DFO algorithm using a neural network surrogate and the same algorithm using an interpolation/regression surrogate. Although interpolation/regression models have been extensively studied in the DFO literature due to their strong performance [5, 10, 17, 18, 19, 20, 79], there are no studies investigating the performance of a neural network surrogate trained over the specific number of points required for deterministic quadratic interpolation. Therefore, a key question is whether a neural network can achieve comparable performance to a quadratic interpolation model when the number of training points is approximately equal to the number of points required for deterministic quadratic interpolation.

The minibatch size for training was equal to 16, 32, and 64 for the dimensions 20, 40, and 60, respectively, for the first set of problems. For the second set, we selected the minibatch as the closest power of 2 to 0.05 times the size of the dataset. To ensure practicality and avoid extreme values, we constrained the mini-batch size to a minimum of 2 and a maximum of 64. The value of $w^0$ was initialized using specific weight initialization techniques tailored to the activation function used. For ReLU and its variants, we applied the He initialization [37] method. For

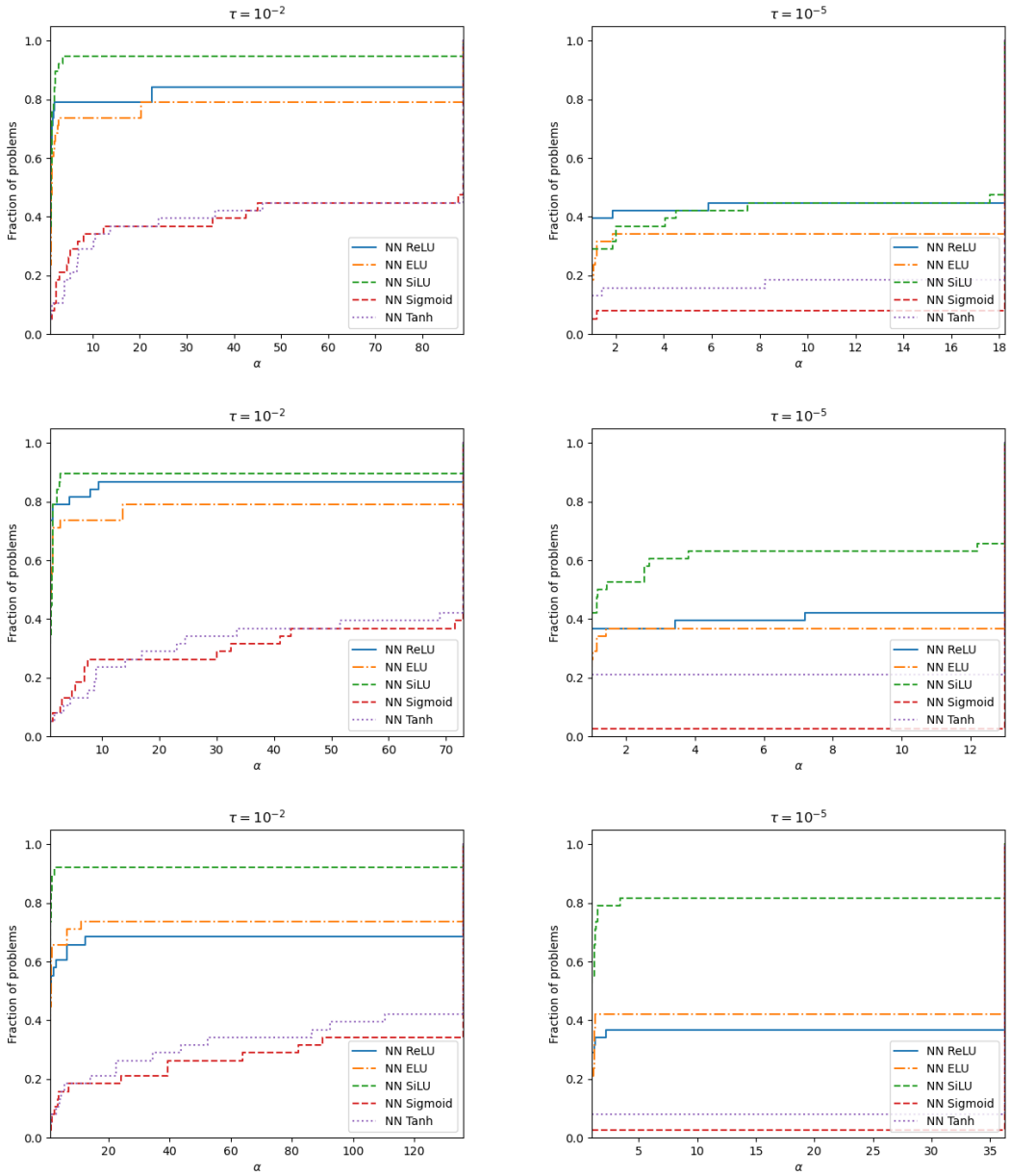


Figure 3: Performance profiles on the testing dataset for $n = 20$ (first row), $n = 40$ (second row), and $n = 60$ (third row) with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ for five neural networks with different activation functions on the set of 38 problems from CUTEst.

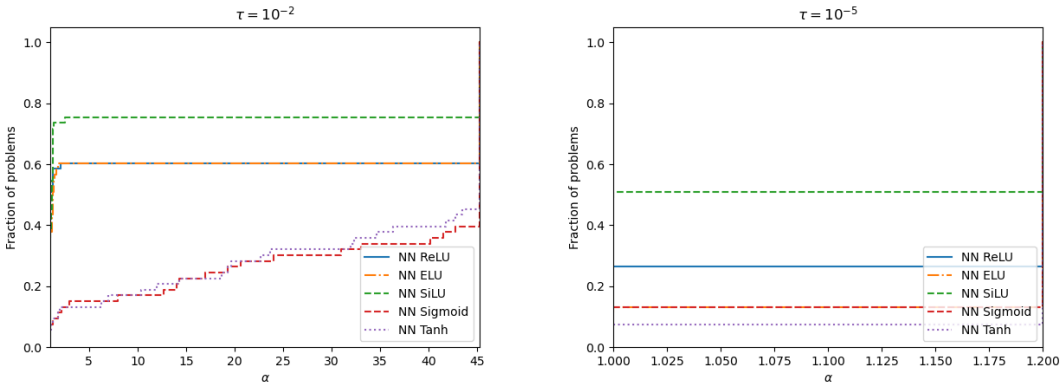


Figure 4: Performance profiles on the testing dataset with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ for five neural networks with different activation functions on the set of 53 problems from CUTEst.

Sigmoid and Tanh, we utilized Xavier/Glorot initialization [30]. These initialization strategies were chosen to address the challenges associated with each activation function and promote stable and efficient training.

For numerical stability and consistency, we normalized both the training dataset $\mathcal{D}$ and the testing dataset. This involved shifting each sample by $-x^0$ and scaling each sample by $\Delta = \max_{1\le i\le N} \|x^i - x^0\|$. As a result, the transformed dataset was contained within a ball of radius one centered at the origin, ensuring that at least one point lies on the boundary of the ball:

$$\left\{0, \frac{x^1 - x^0}{\Delta}, \ldots, \frac{x^N - x^0}{\Delta}\right\} \subset B(0;1). \tag{2.3}$$

Note that normalizing a dataset can help mitigate the impact of the vanishing gradient issue (see [38]). To use the scaling and shifting in (2.3), we solved the training problem (2.1) by replacing the function $f_{\mathrm{NN}}$ in $\mathcal{L}(w;\mathcal{D})$ with

$$\hat{f}_{\mathrm{NN}}(x;w) = f_{\mathrm{NN}}((x - x^0)/\Delta; w). \tag{2.4}$$

Similarly, we used (2.4) when evaluating the empirical risk function at points in the testing dataset.

The numerical results are reported in Figures 1–2 for the training dataset and Figures 3–4 for the testing dataset. The values of $\tau$ were chosen from $\{10^{-2}, 10^{-5}\}$. Regardless of the dimension $n$, ReLU, ELU, and SiLU have superior performance compared to the sigmoidal activation functions on both sets of test problems in terms of both efficiency (see the value of $\rho_s(1)$) and robustness (see the value of $\rho_s(\alpha)$ for large $\alpha$). SiLU achieves the best performance on the testing dataset for both values of $\tau$, followed by ELU and ReLU. On the training data, ReLU performs best on the set of 38 problems but is outperformed by both SiLU and ELU on the set of 53 problems. From these figures, we can also see that the sigmoidal activation functions are not good activation functions for approximation, which is consistent with what is commonly observed in classification tasks, where the performance of ReLU and its variants is often found to be superior [11]. To provide further insights into the accuracy yielded by ReLU and its variants, we include Figures 5–8, which report box plots that illustrate the values of the normalized root mean square error (RMSE) obtained over a set of problems for each activation

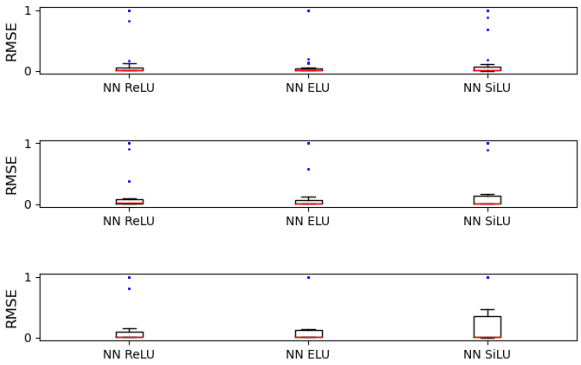


Figure 5: Normalized RMSE for ReLU, ELU, and SiLU on the training dataset for $n = 20$ (first row), $n = 40$ (second row), and $n = 60$ (third row) on the set of 38 problems from CUTEst.

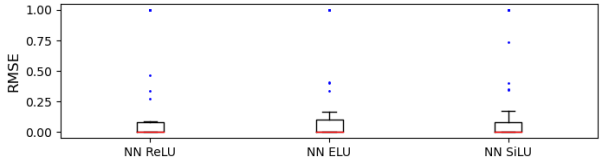


Figure 6: Normalized RMSE for ReLU, ELU, and SiLU on the training dataset on the set of 53 problems from CUTEst.

function. We recall that in a box plot, the horizontal line within the rectangle is the median of the set of values, while the upper and lower lines denote the medians of the upper and lower halves of the same set of values, respectively. The circles represent outliers. The RMSE for a vector of weights $w$ is given by the square root of $\mathcal{L}(w; \mathcal{D})$, as defined in problem (2.1). To generate the box plots in Figures 5–8, we first compute the minimum RMSE value obtained during the training of a neural network for each optimization problem. Then, we limit the range of such RMSE values to $[0, 1000]$ for the first set of problems (for each activation function, 3 to 7 out of 38 problems fall outside this range), and $[0, 5000]$ for the second set of problems (for each activation function, 8 to 9 out of 53 problems fall outside this range), thus excluding extreme outliers. Subsequently, we normalize the obtained values across all problems for each activation function. The plots show that ReLU exhibits slightly better performance than ELU and SiLU on the training and testing datasets on both sets of problems. Such a result does not contradict the observations from the performance profiles in Figures 1–4, where SiLU is shown to outperform ReLU on the testing dataset. The reason is that performance profiles take into account the number of iterations required to achieve a vector of weights that satisfies the convergence test (2.2), while Figures 5–8 simply illustrate the distribution of normalized RMSE values for each activation function.

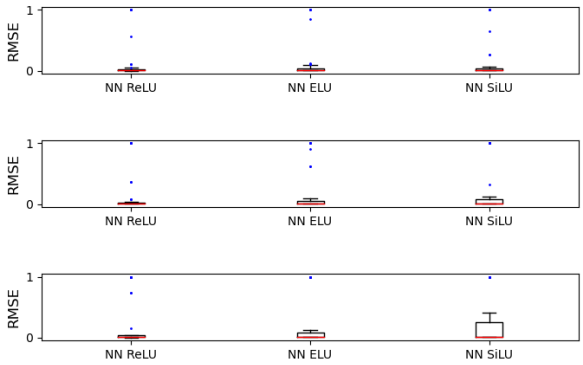


Figure 7: Normalized RMSE for ReLU, ELU, and SiLU on the testing dataset for $n = 20$ (first row), $n = 40$ (second row), and $n = 60$ (third row) on the set of 38 problems from CUTEst.

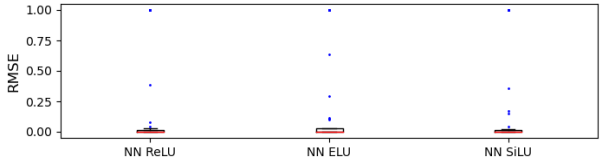


Figure 8: Normalized RMSE for ReLU, ELU, and SiLU on the testing dataset on the set of 53 problems from CUTEst.

# 3 The limitation of neural nets for approximation

We start this section by reviewing the basic concepts of polynomial interpolation and regression in Subsection 3.1. Then, we show that using the composition of an activation function with the natural basis can lead to better approximations of function values, gradients, and Hessians compared to the natural basis (see Subsection 3.2). Lastly, we assess the accuracy of the function value, gradient, and Hessian approximations obtained when using neural networks (see Subsection 3.3).

## 3.1 Review of polynomial interpolation and regression

Polynomial interpolation and regression models are frequently used in DFO methods to approximate computationally expensive objective functions in black-box problems by using polynomials [5, 17, 18, 19, 20, 60, 74]. If the number of sample points used to build an interpolation model is less than its degrees of freedom, the resulting model is underdetermined, and such a situation may arise when functions are too expensive to allow collecting a large number of function values. If there are more sample points than degrees of freedom, one can use a regression model (overdetermined interpolation), which is particularly useful when function values are noisy. If the sample size matches the number of degrees of freedom, the interpolation model is determined. The nonlinear class of models that is often used for both polynomial interpolation and regression are quadratics [5, 17, 61, 62, 75]. As opposed to linear models, quadratic models are able to capture the curvature of the function being approximated.

Let us denote as $\mathcal{P}_n^d$ the space of polynomials of degree less than or equal to $d$ in $\mathbb{R}^n$. Let $q_1 = q + 1$ be the dimension of this space, and let $\phi = \{\phi_0(x), \phi_1(x), \ldots, \phi_q(x)\}$ be a basis for $\mathcal{P}_n^d$ (each $\phi_j(x)$, with $j \in \{0, \ldots, q\}$, is referred to as a basis function). Since $\phi$ is a basis in $\mathcal{P}_n^d$, given real scalars $\alpha_j$, any polynomial $m(x) \in \mathcal{P}_n^d$ can be written as $m(x) = \sum_{j=0}^{q} \alpha_j \phi_j(x)$. The most common polynomial basis is the natural basis $\bar{\phi}$, which is the basis of polynomials in the Taylor expansion

$$\bar{\phi} \;=\; \{1, x_1, x_2, \ldots, x_n, x_1^2/2, x_1 x_2, \ldots, x_{n-1}^{d-1} x_n/(d-1)!, x_n^d/d!\}. \tag{3.1}$$

Throughout this section, we refer to $Y = \{y^0, y^1, \ldots, y^N\} \subset \mathbb{R}^n$ as a sample set, which is a set of $N_1 = N + 1$ points where the objective function is evaluated. In the linear case (i.e., $d = 1$), the dimension of $\mathcal{P}_n^d$ is $q_1 = n + 1$. In the quadratic case (i.e., $d = 2$), the dimension of $\mathcal{P}_n^d$ is $q_1 = (n+1)(n+2)/2$. Underdetermined quadratic interpolation models are based on sample sets of size $N_1$ such that $n + 1 < N_1 < (n+1)(n+2)/2$. Quadratic regression models are based on sample sets of size $N_1 > (n+1)(n+2)/2$. When $N_1 = (n+1)(n+2)/2$, one has determined quadratic interpolation.

Let us now formally describe the optimization problems arising in polynomial interpolation and regression. Denoting the function to approximate as $f : \mathbb{R}^n \to \mathbb{R}$, let $f(Y)$ represent the vector whose elements are $f(y^i)$, with $i \in \{0, \ldots, N\}$. The goal of polynomial interpolation is to determine the coefficients $\alpha_0, \ldots, \alpha_q$ of a polynomial $m(x)$ such that $m(y^i) = \sum_{j=0}^{q} \alpha_j \phi_j(y^i) = f(y^i)$, for all $i \in \{0, \ldots, N\}$, see [19]. To ensure that $m(x)$ interpolates the function $f$ at the points in $Y$, one can solve the following linear system, written in matrix form

$$M(\phi, Y)\alpha_\phi \;=\; f(Y), \tag{3.2}$$

where

$$M(\phi, Y) = \begin{bmatrix} \phi_0\left(y^0\right) & \phi_1\left(y^0\right) & \cdots & \phi_q\left(y^0\right) \\ \phi_0\left(y^1\right) & \phi_1\left(y^1\right) & \cdots & \phi_q\left(y^1\right) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_0\left(y^N\right) & \phi_1\left(y^N\right) & \cdots & \phi_q\left(y^N\right) \end{bmatrix}, \; \alpha_\phi = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_q \end{bmatrix}, \text{ and } f(Y) = \begin{bmatrix} f(y^0) \\ f(y^1) \\ \vdots \\ f(y^N) \end{bmatrix}.$$

The goal of polynomial regression is to determine the coefficients $\alpha_0, \ldots, \alpha_q$ of a polynomial $m(x)$ to solve system (3.2) in the least-squares sense by minimizing $\|M(\phi, Y)\alpha_\phi - f(Y)\|^2$, see [18]. When $N = q$, interpolation and regression yield the same results. Following [20], we say that a sample set $Y$ is poised for polynomial interpolation in $\mathbb{R}^n$ if $M(\phi, Y)$ is square ($N = q$) and non-singular. We say that a sample set $Y$ is poised for polynomial least-squares regression in $\mathbb{R}^n$ if $M(\phi, Y)$ has full-column rank.

For numerical reasons, it is convenient to shift $Y$ by $-y^0$ so that the new set $\{0, y^1 - y^0, \ldots, y^N - y^0\}$ contains the origin. In optimization methods based on polynomial interpolation or regression, the current best iterate is usually selected as such a point $y^0$, see [20]. As in (2.1)–(2.3), one can then scale the resulting sample set by $\Delta = \max_{1 \le i \le N} \|y^i - y^0\|$ so that the new set is contained in a ball of radius one centered at the origin, with at least one point on the boundary of the ball $B(0; 1)$. Throughout this paper, when using interpolation or regression, we will use such a shifting and scaling. Note that this can be achieved by introducing a polynomial $\hat{m}(x)$ such that

$$\hat{m}(x) = m((x - y^0)/\Delta) \tag{3.3}$$

and then performing interpolation or regression on the original sample set to ensure $\hat{m}(y^i) = f(y^i)$, $i \in \{0, \ldots, N\}$, exactly (in the interpolation case) or in the least-squares sense (in the regression case).

### 3.2 The use of an activation function as a basis function

In this subsection, we explore the use of bases obtained from the composition of the basis functions in the natural basis (3.1) with an activation function $s : \mathbb{R} \to \mathbb{R}$. We focus on the quadratic case (i.e., $d = 2$ in (3.1)), which is widely used in practice [5, 17, 61, 62, 75]. The first type of basis[§] we consider is given by

$$\tilde{\phi} \;=\; \{1, x_1, x_2, \ldots, x_n, x_1^2/2, s(x_1x_2), \ldots, s(x_{n-1}x_n), x_n^2/2\}, \tag{3.4}$$

which is obtained from the natural basis (3.1) by applying $s$ to each cross-term $x_ix_j$, with $\{i, j\} \subseteq \{1, \ldots, n\}$ and $i \neq j$. The second type of basis is obtained from (3.4) by removing the cross-terms $x_ix_j$ and adding extra terms $s(x_i)$, for all $i \in \{1, \ldots, n\}$, leading to

$$\hat{\phi} \;=\; \{1, x_1, x_2, \ldots, x_n, s(x_1), s(x_2), \ldots, s(x_n), x_1^2/2, \ldots, x_n^2/2\}. \tag{3.5}$$

We point out that we also considered the bases that can be obtained from (3.4) and (3.5) by applying $s$ to each quadratic term $x_i^2/2$, with $i \in \{1, \ldots, n\}$. However, we have not included them in this paper as they did not show any improvement in performance compared to the natural basis.

In order to have a benchmark to compare bases (3.4) and (3.5) with, we considered a third type of basis consisting of radial basis functions (RBFs) [14, 35, 51]. Such an RBF basis can be written as follows

$$\hat{\hat{\phi}} \;=\; \{h(\|x - y^0\|), h(\|x - y^1\|), \ldots, h(\|x - y^N\|), 1, x_1, \ldots, x_n\}, \tag{3.6}$$

where $h : \mathbb{R} \to \mathbb{R}$ and we are using constant and linear terms as suggested in [20]. Some of the most popular radial basis functions are cubic $h(z) = z^3$, Gaussian $h(z) = e^{-(z^2/\rho^2)}$, multiquadric $h(z) = (z^2 + \rho^2)^{(3/2)}$, and inverse multiquadric $h(z) = (z^2 + \rho^2)^{-(1/2)}$, where $\rho^2$ is a positive constant. When using basis (3.6), given vectors of real scalars denoted as $\lambda = (\lambda_0, \lambda_1, \ldots, \lambda_N)$ and $\gamma = (\gamma_0, \gamma_1, \ldots, \gamma_n)$, the resulting model becomes $m(x) = \sum_{j=0}^{N} \lambda_j h(\|x - y^j\|) + \sum_{j=0}^{n} \gamma_j p_j(x)$, where each $p_j(x)$ represents one of the constant and linear terms in (3.6). To determine the coefficients of such a model, it is common to solve the solve the following linear system [20, 45]

$$\begin{bmatrix} \Phi & P \\ P^\top & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \gamma \end{bmatrix} = \begin{bmatrix} f(Y) \\ 0 \end{bmatrix}, \tag{3.7}$$

where $\Phi_{ij} = h(\|y^i - y^j\|)$, for all $i, j \in \{0, 1, \ldots, N\}$, and $P_{ij} = p_j(y^i)$, for all $i \in \{0, 1, \ldots, N\}$ and $j \in \{0, \ldots, n\}$. The system (3.7) ensures that the interpolation conditions $m(y^i) = f(y^i)$ are satisfied for all $i \in \{0, \ldots, N\}$ and enforces $\sum_{i=0}^{N} \lambda_i p_j(y^i) = 0$, for all $j \in \{0, \ldots, n\}$. In accordance with our notation in (3.3) for interpolation and regression, we will refer to a scaled and shifted model using radial basis functions as $\hat{m}(x)$.

[§]Note that although (3.4) and (3.5) are motivated from the natural basis for the space of polynomials $\mathcal{P}_n^2$, they do not form bases for such a space due to the use of an activation function. Moreover, (3.5) has only $3n + 1$ elements.

Note that the number of degrees of freedom in $m(x)$ when using basis (3.5) is $3n+1$, which is less than the number required for determined quadratic interpolation (i.e., $(n+1)(n+2)/2$). The same remark applies to $\hat{m}(x)$, defined in (3.3). Since we want to test bases (3.4) and (3.5) using a sample set of dimension $(n+1)(n+2)/2$ for both, we need to solve a determined interpolation problem when using (3.4) and a regression problem when using (3.5). Therefore, we will refer to $\hat{m}(x)$ as an interpolation model when using (3.4) and as a regression model when using (3.5). When using the RBF basis (3.6), we will refer to the resulting model $\hat{m}(x)$ as an interpolation model.

To assess the performance of bases (3.4) and (3.5) for interpolation and regression, respectively, and basis (3.6) for interpolation, we used the same sets of test problems considered in Section 2 and listed in Tables 1 and 2 of Appendix A. Here, we are interested in assessing the ability of the proposed basis functions to locally approximate the function $f$ by using an interpolation/regression model $\hat{m}(x)$ as a surrogate model of $f$. To perform this assessment, we compared the function value, gradient, and Hessian approximations obtained using bases (3.4) and (3.5) for different choices of the activation $s$ and basis (3.6) equipped with Gaussian radial basis functions. Similar to the experiments for Section 2, we built the sample set $Y$ by randomly generating $N_1 = (n+1)(n+2)/2$ points according to a uniform distribution in a ball $B(x^0; 1)$, where $x^0$ is the initial point provided by CUTEst.

The results of our numerical experiments are illustrated in Figures 9 and 10, which report box plots that allow us to compare the natural basis (3.1), bases (3.4) and (3.5) when the activation function $s$ is either ReLU, ELU, SiLU, Sigmoid, or Tanh, and basis (3.6). In particular, Figure 9 corresponds to the set of 38 problems (with $n = 20$), while Figure 10 pertains to the set of 53 problems. Recalling the definition of $\hat{m}(x)$ in (3.3), the y-axis of the upper plot in Figures 9 and 10 represents the value of the following metric

$$|\hat{m} - f| \;:=\; \frac{|\hat{m}(x^0) - f(x^0)|}{\max\{|\hat{m}(x^0)|, |f(x^0)|\}}, \tag{3.8}$$

where the max function is used in the denominator to avoid a division by zero when either $\hat{m}(x^0)$ or $f(x^0)$ is equal to zero (in the experiments, such terms are never both equal to zero). Such a metric evaluates the function value approximation error by comparing the value of $f$ at the initial point provided by CUTEst with the value of $\hat{m}$ at the same point. We adopted similar metrics to evaluate the gradient and Hessian approximation errors. Specifically, the y-axes of the middle and lower plots in Figures 9 and 10 represent the values of

$$\|\nabla\hat{m} - \nabla f\| \;:=\; \frac{\|\nabla\hat{m}(x^0) - \nabla f(x^0)\|}{\max\{\|\nabla\hat{m}(x^0)\|, \|\nabla f(x^0)\|\}} \tag{3.9}$$

and

$$\|\nabla^2\hat{m} - \nabla^2 f\| \;:=\; \frac{\|\nabla^2\hat{m}(x^0) - \nabla^2 f(x^0)\|}{\max\{\|\nabla^2\hat{m}(x^0)\|, \|\nabla^2 f(x^0)\|\}}, \tag{3.10}$$

respectively. In all the plots, the x-axis corresponds to the natural basis (3.1), the bases derived from (3.4) and (3.5), and basis (3.6) according to the following notation:

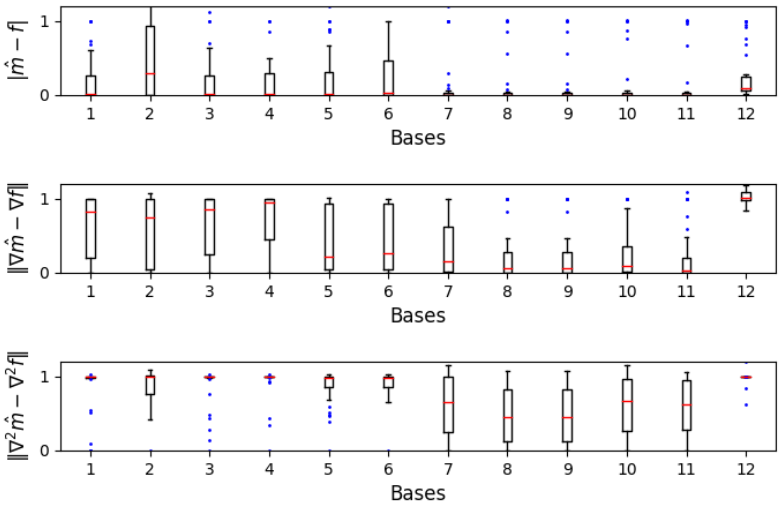


Figure 9: Comparison of $|\hat{m} - f|$, $\|\nabla\hat{m} - \nabla f\|$, and $\|\nabla^2\hat{m} - \nabla^2 f\|$ among different bases for the set of 38 problems listed in Table 1 of Appendix A.

1: (3.1),
2: (3.4) with ReLU,
3: (3.4) with ELU,
4: (3.4) with SiLU,
5: (3.4) with Sigmoid,
6: (3.4) with Tanh,
7: (3.5) with ReLU,
8: (3.5) with ELU,
9: (3.5) with SiLU,
10: (3.5) with Sigmoid,
11: (3.5) with Tanh,
12: (3.6).

Each box plot illustrates the values of a metric from (3.8)–(3.10) obtained over all the problems considered in the corresponding figure. To compute the box plots, we restrict the values of metrics (3.8)–(3.10) to the range $[0, 5]$ in order to exclude extreme outliers. In the figures, we limit the y-axis to the range $[0, 1]$.

Figure 9 shows that the bases derived from (3.5) allow one to obtain the most accurate model $\hat{m}$ in terms of function value, gradient, and Hessian approximations on the set of 38 problems. Basis (3.6) exhibits a similar performance to (3.5) in terms of function value approximation, but it is significantly worse in terms of gradient and Hessian approximations. Note that the values of metrics (3.8)–(3.10) obtained using the bases derived from (3.4) are comparable to those achieved using the natural basis (3.1), and we can observe that both (3.1) and (3.4) lead to quite inaccurate function value, gradient, and Hessian approximations (this is somehow expected given that the radius of the ball is equal to 1, and we observed that the inaccuracy reduces if we take a smaller ball).

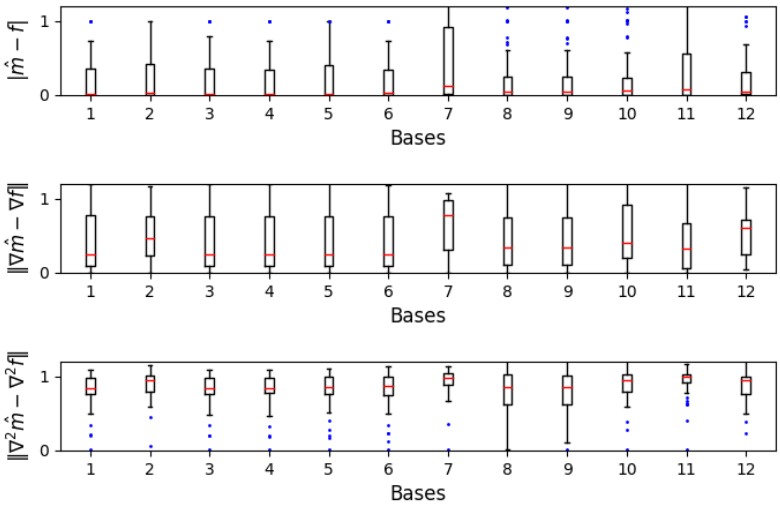


Figure 10: Comparison of $|\hat{m} - f|$, $\|\nabla\hat{m} - \nabla f\|$, $\|\nabla^2\hat{m} - \nabla^2 f\|$ among different bases for the set of 53 problems listed in Table 2 of Appendix A.

When considering the set of 53 problems, Figure 10 shows that (3.1) and (3.4) have similar performance to (3.5) in terms of function value, gradient, and Hessian approximations. Therefore, such results suggest that the cross-terms $x_i x_j$ in (3.1) and (3.4) might not be necessary, regardless of the presence of negative curvature in the functions being approximated. We recall that employing a sample set of size $N_1$ to assess the performance of both (3.4) and (3.5) required us to solve a fully determined interpolation problem when using (3.4) and a regression problem when using (3.5). Similar to (3.1) and (3.4), basis (3.6) performs quite well in terms of function value and gradient approximations, but it is not able to capture curvature information with the same level of accuracy. We point out that we repeated the analysis by replacing $x^0$ with points located between the origin and $x^0$ itself, and this did not change our conclusions.

Our conclusion is that the use of activation functions in quadratic interpolation and regression seems to waive the necessity of including cross terms.

### 3.3 The use of neural networks for approximation

In this subsection, we assess the accuracy of the function value, gradient, and Hessian approximations obtained when using a neural network as a surrogate model of the function $f$. In particular, we considered five neural networks, each equipped with either ReLU, ELU, SiLU, Sigmoid, or Tanh in the hidden layers (in the output layer, a linear activation function is used). To conduct this analysis and ensure a fair comparison with the numerical results obtained for the interpolation/regression model in Subsection 3.2, we constructed the training dataset $\mathcal{D} = \{(x^i, f(x^i)) \mid i \in \{0, \ldots, N\}\}$ using the same approach that we employed to build the sample set $Y$. Specifically, we randomly generated $N_1 = N + 1 = (n+1)(n+2)/2$ points according to a uniform distribution in a ball $B(x^0; 1)$, where $x^0$ is once again the initial point provided by CUTEst. Then, after applying the shifting and scaling described in (2.3), we solved problem (2.1) for each neural network. For the details of the neural network architecture and

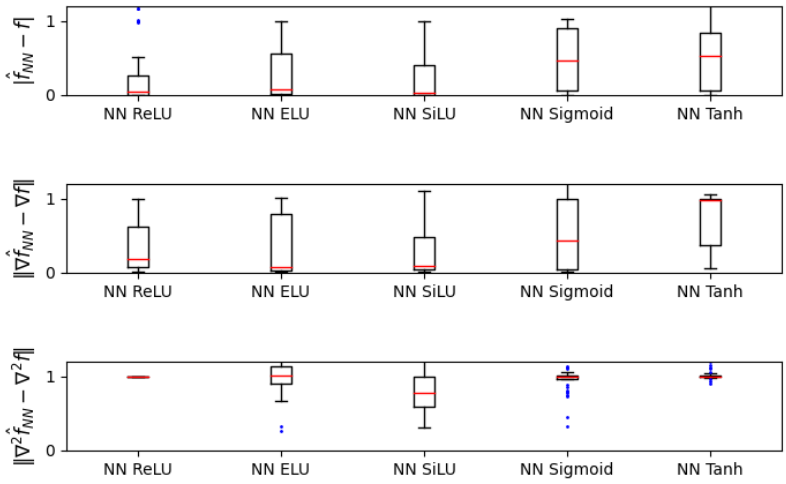


Figure 11: Comparison of $|\hat{f}_{\rm NN} - f|$, $\|\nabla\hat{f}_{\rm NN} - \nabla f\|$, and $\|\nabla^2\hat{f}_{\rm NN} - \nabla^2 f\|$ among different activation functions for the set of 38 problems listed in Table 1 of Appendix A.

the training process, we refer to Subsection 2.2.

Figure 11 corresponds to the set of 38 problems (with $n = 20$), while Figure 12 pertains to the set of 53 problems. To compare the function values, gradients, and Hessians of a neural network surrogate with those of the function $f$, we use similar metrics to (3.8)–(3.10). In particular, recalling the definition of the neural network surrogate $\hat{f}_{\rm NN}$ in (2.4), we consider

$$|\hat{f}_{\rm NN} - f| := \frac{|\hat{f}_{\rm NN}(x^0) - f(x^0)|}{\max\{|\hat{f}_{\rm NN}(x^0)|, |f(x^0)|\}}, \tag{3.11}$$

$$\|\nabla\hat{f}_{\rm NN} - \nabla f\| := \frac{\|\nabla\hat{f}_{\rm NN}(x^0) - \nabla f(x^0)\|}{\max\{\|\nabla\hat{f}_{\rm NN}(x^0)\|, \|\nabla f(x^0)\|\}}, \tag{3.12}$$

$$\|\nabla^2\hat{f}_{\rm NN} - \nabla^2 f\| := \frac{\|\nabla^2\hat{f}_{\rm NN}(x^0) - \nabla^2 f(x^0)\|}{\max\{\|\nabla^2\hat{f}_{\rm NN}(x^0)\|, \|\nabla^2 f(x^0)\|\}}. \tag{3.13}$$

Figure 11 reports box plots that illustrate the comparison of metrics (3.11)–(3.13) among different activation functions for the set of 38 problems. In particular, the upper and middle plots in Figure 11 show that ReLU and its variants are the activation functions that yield the highest accuracy in terms of function value and gradient approximations, while the sigmoidal functions exhibit lower accuracy. Note that the value of metric (3.12) is significantly large when using the sigmoidal activations. Indeed, for most of the problems, the approximate gradients obtained using the sigmoidal activations were observed to be close to the null vector due to the vanishing gradient problem, despite the use of the shifting and scaling in (2.3). The lower plot in Figure 11 shows that the considered neural networks were not able to accurately approximate the curvature of any objective function regardless of the activation function used, with the exception of SiLU, which shows the best performance. The limitation in capturing curvature information was somehow expected for ReLU, which is a piecewise-linear function with no curvature, but it

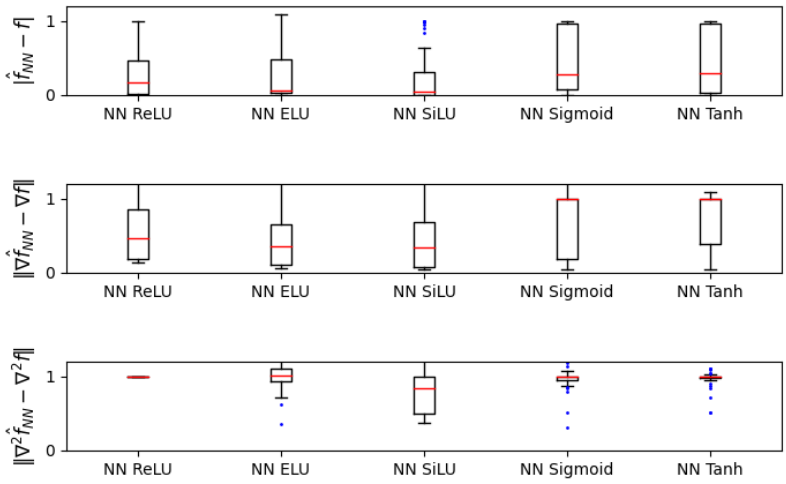


Figure 12: Comparison of $|\hat{f}_{\text{NN}} - f|$, $\|\nabla\hat{f}_{\text{NN}} - \nabla f\|$, and $\|\nabla^2\hat{f}_{\text{NN}} - \nabla^2 f\|$ among different activation functions for the set of 53 problems listed in Table 2 in Appendix A.

is less obvious for the sigmoidal activations. The upper and middle plots in Figure 12 confirm that ReLU and its variants are the best activation functions to approximate the function values and gradients on the set of 53 problems as well. The lower plot in Figure 12 demonstrates that although none of the activation functions provides very accurate Hessian approximations, SiLU exhibits a superior ability to capture curvature information.

# 4 The limitation of neural nets for optimizing without derivatives

The aim of this section is to enhance the performance of a state-of-the-art derivative-free optimization (DFO) algorithm when the gradient of the objective functions of the test problems in Tables 1 and 2 of Appendix A is approximated using the surrogate models considered in Sections 2 and 3, including neural networks. DFO is a well-studied area in the field of mathematical optimization that deals with solving problems where each evaluation of the objective function is time-consuming or computationally expensive [2, 20, 22, 45]. Recent research in DFO has shown that combining a Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton update step with a finite-difference (FD) gradient allows obtaining strong performance in the minimization of a smooth and noiseless objective function by taking advantage of the curvature of such a function [6, 8, 69].

In Subsection 4.1, we will present the BFGS update step with FD gradient, as well as its surrogate-based version, as part of the general class of methods called `Full-Low Evaluation` (FLE), which will be briefly mentioned below and reviewed in Appendix B. FLE methods have been proposed in [9] and can be used to minimize a function $f$ without using its derivatives, which makes this class of algorithms suitable to solve problem (1.1). In Subsection 4.2, we will report the results of our numerical experiments, where we used surrogate models to enhance the

performance of a practical FLE method.

### 4.1 The use of a surrogate model for optimizing without derivatives

We will start this subsection by describing the $k$-th iteration of the BFGS update step with FD gradient in Algorithm 1, and then we will present its surrogate-based version in Algorithm 2 below. The direction explored in Algorithm 1 is given by $p_k = -H_k g_k$, where $g_k$ is the approximation of the gradient $\nabla f(x_k)$ obtained using a forward FD scheme at the $k$-th iterate $x_k$ and $H_k$ is the BFGS approximation of the Hessian inverse $\nabla^2 f(x_k)$. In particular, the $i$-th component of $g_k$ is given by

$$[g_k]_i \;=\; \frac{f(x_k + h_k e_i) - f(x_k)}{h_k}, \quad \forall i \in \{1, \ldots, n\}, \tag{4.1}$$

where $h_k$ is the FD parameter and $e_i \in \mathbb{R}^n$ is the $i$-th canonical vector (see Step 1 for the use of (4.1) in Algorithm 1). To compute $H_k$, one needs first to determine $s_k = x_k - x_{k-1}$ and $y_k = g_k - g_{k-1}$ (see Step 2). Then, if $s_k^\top y_k \geq \varepsilon \|s_k\| \|y_k\|$, with $\varepsilon > 0$, one can set

$$H_k \;=\; \left(I - \frac{s_k y_k^\top}{y_k^\top s_k}\right) H_{k-1} \left(I - \frac{y_k s_k^\top}{y_k^\top s_k}\right) + \frac{s_k s_k^\top}{y_k^\top s_k}, \tag{4.2}$$

and $H_k = H_{k-1}$ otherwise (see Step 3). The condition on the scalar product $s_k^\top y_k$ is necessary to maintain the positive definiteness of $H_k$. At the first iteration, one possible choice is $H_0 = (y_0^\top s_0)/(y_0^\top y_0) I$, which ensures similarity in size between $H_0$ and $\nabla^2 f(x_0)^{-1}$, see [56].

Once the direction $p_k$ has been computed at Step 4, one can obtain the next iterate by performing a line search at Steps 5–6 and setting $x_{k+1} = x_k + \beta_k p_k$ at Step 7, where $\beta_k$ is the value of $\beta > 0$ that satisfies the following sufficient decrease condition by backtracking from a positive initial stepsize $\bar{\beta}$

$$f(x_k + \beta p_k) \;\leq\; f(x_k) + c\beta g_k^\top p_k, \qquad \text{with } c \in (0, 1). \tag{4.3}$$

The sufficient decrease condition (4.3), which is typical in nonlinear optimization, allows one to obtain stepsizes that are neither too large nor too small [56].

**Algorithm 1** FLE (only showing the FD-BFGS step)

**Input:** Iterates $x_{k-1}$ and $x_k$, $\varepsilon > 0$, backtracking parameters $\bar{\beta} > 0$ and $\tau \in (0, 1)$.

1: Compute the FD gradient $g_k$ by using (4.1).
2: Set $s_k = x_k - x_{k-1}$ and $y_k = g_k - g_{k-1}$.
3: **If** $s_k^\top y_k \geq \varepsilon \|s_k\| \|y_k\|$, set $H_k$ according to (4.2), **else** $H_k = H_{k-1}$.
4: Compute the direction $p_k = -H_k g_k$ and set $\beta = \bar{\beta}$ for the backtracking line-search.
5: **While** (4.3) is false **do**
6: Set $\beta = \tau\beta$.
7: Set $\beta_k = \beta$ and $x_{k+1} = x_k + \beta_k p_k$.

**Output:** Iterate $x_{k+1}$.

Note that computing (4.1) requires $n$ additional function evaluations per iteration. Rather than incurring such a cost, we propose to build a surrogate model of the objective function $f$ at each iteration $k$ and use its gradient as the approximate gradient $g_k$ for the BFGS update step. Such a model can be fitted to points in the current dataset $\mathcal{D}_k = \{(x_i, f(x_i)) \mid i < k\}$, consisting of pairs $(x_i, f(x_i))$ associated with function evaluations done at the past iterations. As the algorithm progresses and more points are collected, the accuracy of the model in approximating the objective function improves over the iterations. Since the cost of the FD approximation in (4.1) depends on $n$, the benefit of using the gradient of the model instead of the FD gradient is higher on large-scale optimization problems.

As a surrogate model, we consider either an interpolation or regression model, such as the ones obtained through bases (3.4), (3.5), and (3.6) proposed in Subsection 3.2, or a neural network, like the ones used in Subsection 3.3. When employing an interpolation or regression model, the sample set used at iteration $k$, denoted by $Y_k$, can be obtained as a subset of the points $x_i$ in the current dataset $\mathcal{D}_k$, i.e., $Y_k \subseteq \{x_i \mid (x_i, f(x_i)) \in \mathcal{D}_k\}$. When using a neural network, one can consider the entire $\mathcal{D}_k$ as a training set. The limitation of using an interpolation or regression surrogate in the FD-BFGS step in Algorithm 1 is that the points generated during the backtracking line search in steps 4–7 are aligned along the direction $p_k$ and, therefore, the resulting sample set $Y_k$ might not be well poised. The limitation of using a neural network surrogate is that the points generated during the backtracking line search might not be sufficient to get an accurate approximation of the objective function, and more points need to be collected. To overcome the above limitations, one can take advantage of a direct-search step to collect additional points that are more evenly distributed throughout the space.

In fact, in our experiments, we consider the class of FLE methods, which combines the FD-BFGS step in Algorithm 1 with a direct-search step [9]. FLE methods exhibit state-of-the-art performance by leveraging two types of iterations: on the one hand, `Full-Eval` (FE) iterations are effective in the smooth, non-noisy case but require a large number of function evaluations (this is Algorithm 1); on the other hand, `Low-Eval` (LE) iterations are cheaper in terms of function evaluations and are effective in the noisy and/or non-smooth case. The details of the LE step (direct search) are unnecessary for our discussion as well as the switches from FE to LE and viceversa (see Appendix B).

The version of Algorithm 1 equipped with a surrogate model is given in Algorithm 2 and is denoted as FLE-S, where the 'S' stands for surrogate. FLE-S enables the use of the surrogate when the cardinality of $\mathcal{D}_k$ reaches $\zeta(n+1)(n+2)/2$, where $\zeta > 0$. The dataset $\mathcal{D}_k$ includes all points used in the calculation of the FD gradient up to the current iteration (see Step 1). All the points generated in the LE step (direct search) are also included in $\mathcal{D}_k$ (see Appendix B). When employing an interpolation or regression surrogate, among all the points generated in the line search at Steps 8–9, we only include the first one in $\mathcal{D}_k$ to avoid adding points that are aligned (see Step 11). When using a neural network surrogate, all the points generated in the line search are added to $\mathcal{D}_k$ (again, see Step 11). At Step 2, we generate an extra point according to a uniform distribution defined over a ball centered at the current iterate and with radius 0.1, i.e., $B(x_k; 0.1)$, and we add such a point to $\mathcal{D}_k$. Then, at Step 3, when employing an interpolation or regression surrogate, a model can be built by quadratic interpolation or regression or fitting radial basis functions to the current sample set $Y_k$. When using the neural network surrogate, all the points in $\mathcal{D}_k$ can be used for training the model.

**Algorithm 2** FLE-S (only showing the FD-BFGS step with surrogate)

**Input:** Iterates $x_{k-1}$ and $x_k$, $\varepsilon > 0$, $\zeta > 0$, backtracking parameters $\bar{\beta} > 0$ and $\tau \in (0, 1)$, dataset $\mathcal{D}_k$.

1: **If** $|\mathcal{D}_k| < \zeta(n+1)(n+2)/2$, compute the FD gradient $g_k$ by using (4.1), and add all points used in the calculation of the FD gradient to $\mathcal{D}_k$.

**Else**

2: Add a point uniformly generated on $B(x_k; 0.1)$ to $\mathcal{D}_k$.

3: Build an interpolation or regression surrogate or train a neural network surrogate.

4: Set $g_k$ equal to the gradient of the surrogate.

5: Set $s_k = x_k - x_{k-1}$ and $y_k = g_k - g_{k-1}$.

6: **If** $s_k^\top y_k \geq \varepsilon \|s_k\| \|y_k\|$, set $H_k$ according to (4.2), **else** $H_k = H_{k-1}$.

7: Compute the direction $p_k = -H_k g_k$ and set $\beta = \bar{\beta}$ for the backtracking line-search.

8: **While** (4.3) is false **do**

9: Set $\beta = \tau\beta$.

10: Set $\beta_k = \beta$, $x_{k+1} = x_k + \beta_k p_k$.

11: When using an interpolation or regression surrogate, among all the points generated in the line search at Steps 8–9, add the first one to $\mathcal{D}_k$. When using a neural network surrogate, add all of these points to $\mathcal{D}_k$.

**Output:** Iterate $x_{k+1}$.

### 4.2 Numerical experiments

We used both sets of test problems listed in Tables 1 and 2 of Appendix A to compare the default version of the FLE algorithm against its version equipped with a model. In particular, we compared the results obtained by FLE with those obtained by FLE-S when using the natural basis (3.1), basis (3.5) with Sigmoid activation, the RBF basis (3.6), and neural networks with ReLU and SiLU as surrogates. We did not select basis (3.4) because it has a similar performance to the natural basis (3.1), as discussed in Subsection 3.2. When using basis (3.5), we chose Sigmoid due to its slightly superior performance in the numerical results in Subsection 3.2. When using the neural network surrogates, we chose ReLU and SiLU as activation functions because the numerical results in Subsection 3.3 show that they are effective in approximating gradients, with SiLU being the best at capturing curvature information.

At Step 3 of Algorithm 2, when employing the natural basis (3.1), basis (3.5), or RBF basis (3.6), we built a model on the sample set $Y_k$, which consisted of all the points $x_i$ from $\mathcal{D}_k$ in the ball $B(x_k; 0.1\gamma)$, with $\gamma = 1.1^j$, where $j \in \mathbb{Z}$ is the smallest integer such that $B(x_k; 0.1\gamma)$ contains at least $(n+1)(n+2)/2$ points. Therefore, the resulting sample set was $Y_k = \{x_i \mid (x_i, f(x_i)) \in \mathcal{D}_k \text{ and } x_i \in B(x_k; 0.1\gamma)\}$. When using the neural network surrogate, all the points in $\mathcal{D}_k$ were used for training the model. For the details of the neural network architecture and the algorithm used for training, we refer to Subsection 2.2. We trained the network using a learning rate of $10^{-2}$, which was chosen by performing a grid search over the set $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. Regarding the training process, we implemented the following rule for our experiments. The first time the surrogate model is called, the training consists of 5 epochs. Then, only 1 epoch

is used to avoid an excessive increase in the computational cost. Regarding the value of the parameter $\zeta$ in Step 1 of Algorithm 2, we set $\zeta = 1$ when employing the natural basis (3.1), basis (3.5), or basis (3.6), and $\zeta = 0.2$ when using the neural network surrogate. Note that the choice of $\zeta = 1$ for the interpolation/regression surrogate implies that FLE-S begins constructing the model once there are enough points to perform at least a determined quadratic interpolation.

Figures 13 and 14 show the performance profiles with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ on the sets of 38 and 53 problems, respectively. For each test problem $p \in \mathcal{P}$, the performance measure $t_{p,s}$ is given by the number of function evaluations required by each solver $s \in \mathcal{S}$ to achieve a point $x$ that satisfies the following convergence test

$$f(x_0) - f(x) \;\geq\; (1 - \tau)(f(x_0) - f_L),$$

where $x_0$ is the starting point for the problem $p$ and $f_L$ is the smallest value of $f$ obtained by any solver in $\mathcal{S}$. Although FLE requires $n$ function evaluations more than FLE-S at each iteration when the model is enabled, FLE-S is not able to outperform FLE regardless of the type of surrogate used. Note that on the set of 38 problems, when the size of the problems becomes larger and $\tau = 10^{-2}$, FLE-S with the neural network surrogate tends to be more efficient than FLE, but this is not enough to achieve the same performance as FLE in terms of robustness. Moreover, when using $\tau = 10^{-5}$, the performance of such an FLE-S significantly deteriorates both in terms of efficiency and robustness. One can also observe that on both sets of test problems, FLE-S equipped with the neural network surrogate is outperformed by FLE-S equipped with the interpolation/regression surrogate in terms of robustness. We point out that using a more complex architecture for the neural network surrogate would require more training points to achieve accurate approximations, and so the neural network surrogate would still not be competitive against the default FLE.

# 5 Concluding remarks and takeaway messages

In this paper, we assessed the use of neural networks as surrogate models to approximate and minimize objective functions in optimization problems. To conduct this analysis, we considered two sets of popular nonlinear optimization test problems. The first set consists of 38 problems with different features (non-linearity, non-convexity, partial separability), while the second set consists of 53 problems whose objective functions have been observed to have negative curvature. In Section 2, we compared the performance of neural networks equipped with ReLU, ELU, SiLU, Sigmoid, and Tanh in approximating the objective functions in the two sets of test problems on training and testing datasets. We found that ReLU and SiLU exhibit the best performance on both sets of problems, with ReLU achieving low RMSE values on both training and testing data, and SiLU ensuring superior efficiency and robustness on the testing data, as evidenced by performance profiles.

In Section 3, we compared the function values, gradients, and Hessians of the objective functions in the two sets of test problems with those of surrogate models obtained through interpolation or regression and neural networks. The numerical results of Subsection 3.2 demonstrate that the composition of an activation function with the natural basis can lead to better function value and gradient approximations than both the natural basis and the RBF basis (3.6), by applying quadratic regression with basis (3.5) (which does not contain the cross-terms $x_i x_j$). This suggests that an activation function can waive the necessity of including cross terms in

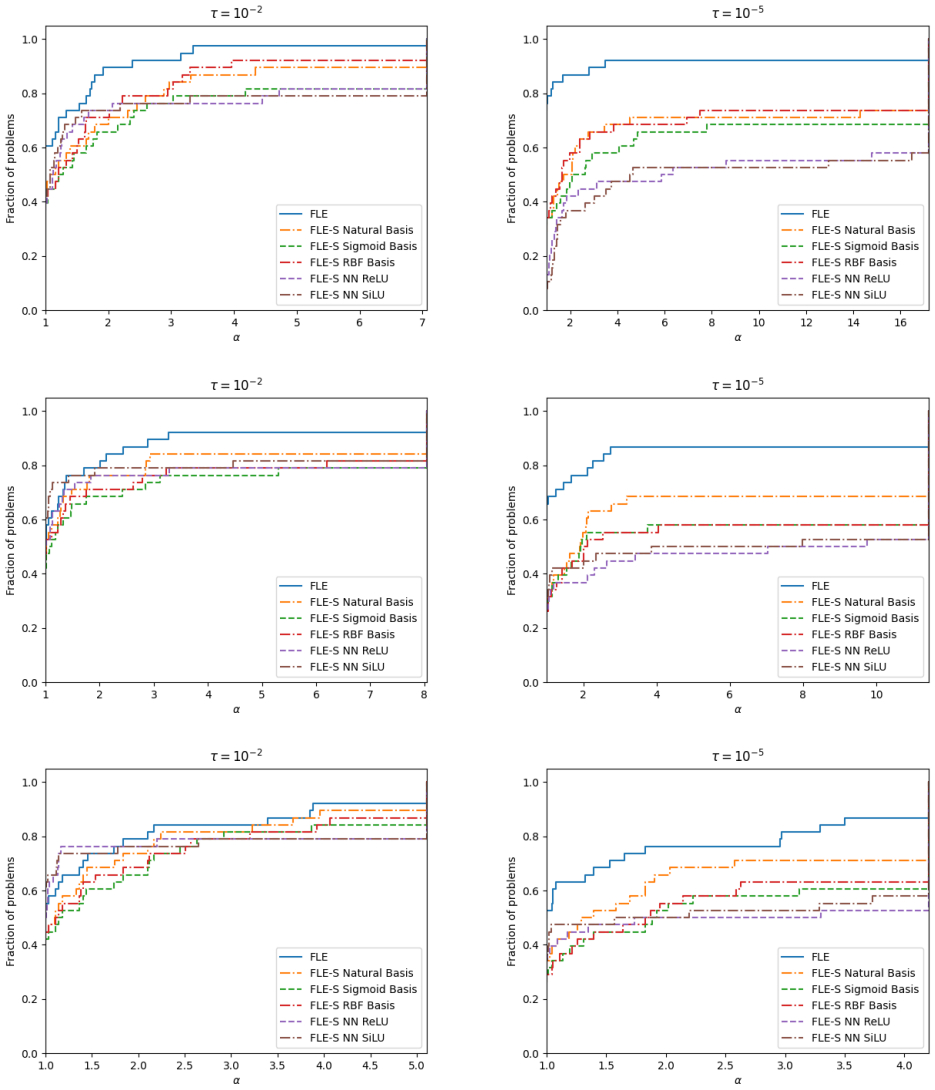


Figure 13: Performance profiles for $n = 20$ (first row), $n = 40$ (second row), and $n = 60$ (third row) with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ on the set of 38 problems listed in Table 1 of Appendix A.

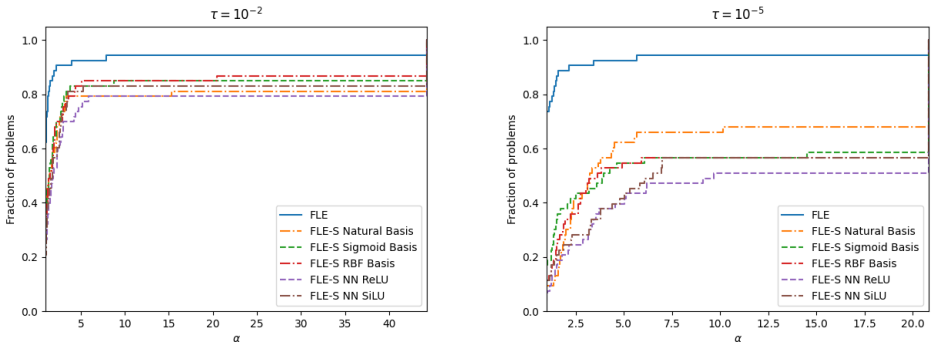


Figure 14: Performance profiles with $\tau = 10^{-2}$ and $\tau = 10^{-5}$ on the set of 53 problems listed in Table 2 of Appendix A.

the natural basis. Quadratic interpolation with basis (3.4) does not lead to any significant improvement compared to the natural basis. On problems with negative curvature, all the bases provide similar performance in terms of function value, gradient, and Hessian approximations. The numerical results of Subsection 3.3 show that a neural network surrogate provides better function value and gradient approximations when ReLU or its variants are used as activation functions, while the Hessian approximations are highly inaccurate regardless of the activation function used, with the exception of SiLU.

In Section 4, we aimed to enhance the performance of the FLE method by using a surrogate model to approximate the objective function gradient in the FD-BFGS update step. This approach allows one to avoid calculating an FD gradient, which requires $n$ additional function evaluations at each iteration. As a surrogate model, we tested the interpolation/regression surrogates obtained from the natural basis (3.1), basis (3.5), and RBF basis (3.6), and a neural network surrogate. None of the resulting algorithms was able to significantly enhance the performance of the default FLE method. The neural network surrogate was observed to be less robust than the interpolation/regression surrogates, although it exhibited higher efficiency when dealing with larger problem sizes. We point out that we also conducted experiments to test the use of the surrogates as replacements for the objective function in the line search at Steps 8–9 of Algorithm 2 and the addition of a search step where the surrogate is minimized to determine a better iterate. However, these experiments did not result in a good performance for the algorithms equipped with surrogate models. In particular, we observed that neural network surrogate models are flat in regions where the network has not seen any data points during training, posing a significant challenge in optimizing such surrogates effectively. Therefore, we decided to omit the corresponding numerical results from the paper.

We conclude this section with some key takeaway messages that can explain the limitations of the algorithms equipped with surrogate models in Section 4. The neural network architectures considered in our paper and described in Subsection 2.2 were intentionally kept simple. Using more complex architectures would require a larger number of training points for the neural network surrogate to achieve satisfactory accuracy. By the time enough points are collected, an algorithm without surrogates would likely have already produced a satisfactory solution, thus nullifying the advantages of using a neural network surrogate (**first key takeaway**), which was already known when using interpolation or regression models. While Subsections 3.2–3.3

demonstrate that interpolation/regression models and neural networks can yield satisfactory approximations of function values and gradients when sample and training points are uniformly distributed on a unit sphere, such an accuracy diminishes when sample and training points collected during optimization are not symmetrically distributed around the current iterate, which is often the case. In such a situation, the quality of the resulting approximations declines (**second key takeaway**). Finally, the numerical results on the set of 38 problems in Subsection 4.2 indicate that the performance of the algorithms equipped with surrogate models does not tend to significantly improve as the dimension increases. This observation is due to the fact that while using a surrogate model reduces the number of function evaluations, higher dimensions require a larger number of training points to achieve satisfactory performance, and this outweighs the benefits gained from the reduction in function evaluations with the surrogate (**third key takeaway**). Such three takeaway messages offer new or consolidated insights into the inherent limitations of surrogate models used within DFO algorithms, which pose challenges that are difficult to overcome. These challenges are particularly significant for neural network surrogates due to their requirement for a large number of training points to achieve satisfactory performance. The conclusions of our analysis may change when addressing problems involving noisy objective function values, and we leave the exploration of such a research avenue for future work.

## Acknowledgments

This work is partially supported by the U.S. Air Force Office of Scientific Research (AFOSR) award FA9550-23-1-0217 and the U.S. Office of Naval Research (ONR) award N000142412656.

## Declarations

**Conflict of interest** The authors have no relevant financial or non-financial interests to disclose.

## Data availability

The authors confirm that the data supporting the findings of this study are available within the article.

## References

[1] A. Al-Kababji, F. Bensaali, and S. Prasad Dakua. Scheduling techniques for liver segmentation: ReduceLROnPlateau Vs OneCycleLR. *arXiv e-prints*, art. arXiv:2202.06373, February 2022.

[2] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer Series in Operations Research and Financial Engineering. Springer, Cham, 2017. With a foreword by John E. Dennis Jr.

[3] C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM J. Optim.*, 17:188–217, 2006.

[4] C. P. Avelino, J. M. Moguerza, A. Olivares, and F. J. Prieto. Combining and scaling descent and negative curvature directions. *Math. Program.*, 128:285–319, 2011.

[5] A. S. Bandeira, K. Scheinberg, and L. N. Vicente. Computation of sparse low degree interpolating polynomials and their application to derivative-free optimization. *Math. Program.*, 134:223–257, 2012.

[6] A. S. Berahas, R. H. Byrd, and J. Nocedal. Derivative-free optimization of noisy functions via quasi-newton methods. *SIAM J. Optim.*, 29:965–993, 2019.

[7] A. S. Berahas, L. Cao, and K. Scheinberg. Global convergence rate analysis of a generic line search algorithm with noise. *SIAM J. Optim.*, 31:1489–1518, 2021.

[8] A. S. Berahas, L. Cao, K. Choromanski, and K. Scheinberg. A theoretical and empirical comparison of gradient approximations in derivative-free optimization. *Found. Comput. Math.*, 22:507–560, apr 2022.

[9] A. S. Berahas, O. Sohab, and L. N. Vicente. Full-low evaluation methods for derivative-free optimization. *Optim. Methods Softw.*, 38:386–411, 2023.

[10] A. Bhaduri, D. Brandyberry, M. D. Shields, P. Geubelle, and L. Graham-Brady. On the usefulness of gradient information in surrogate modeling: Application to uncertainty propagation in composite material models. *Probabilistic Engineering Mechanics*, 60:103024, 2020.

[11] G. Bingham and R. Miikkulainen. Discovering parametric activation functions. *arXiv e-prints*, art. arXiv:2006.03179, June 2020.

[12] A. J. Booker, J. E. Dennis Jr., P. D. Frank, D. B. Serafini, V. Torczon, and M. W. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural and Multidisciplinary Optimization*, 17:1–13, 1998.

[13] M. Boresta, T. Giovannelli, and M. Roma. Managing low–acuity patients in an emergency department through simulation–based multiobjective optimization using a neural network metamodel. *Health Care Management Science*, pages 1–21, 06 2024.

[14] M. D. Buhmann. *Radial basis functions: Theory and implementations.* Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2003.

[15] Y. Chen, M. W. Hoffman, S. Gomez Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas. Learning to learn without gradient descent by gradient descent. *arXiv e-prints*, art. arXiv:1611.03824, November 2016.

[16] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv e-prints*, art. arXiv:1511.07289, November 2015.

[17] A. R. Conn and Ph. L. Toint. An algorithm using quadratic interpolation for unconstrained derivative free optimization. In G. Di Pillo and F. Gianessi, editors, *Nonlinear Optimization and Applications*, pages 27–47. Plenum Publishing, New York, 1996.

[18] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of sample sets in derivative free optimization: Polynomial regression and underdetermined interpolation. *IMA J. Numer. Anal.*, 28:721–748, 2008.

[19] A. R. Conn, K. Scheinberg, and L. N. Vicente. Geometry of interpolation sets in derivative-free optimization. *Math. Program.*, 111:141–172, 2008.

[20] A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to Derivative-Free Optimization.* MPS/SIAM Book Series on Optimization, SIAM, Philadelphia, USA, 2009.

[21] A. R. Conn, K. Scheinberg, and L. N. Vicente. Global convergence of general derivative-free trust-region algorithms to first- and second-order critical points. *SIAM J. Optim.*, 20: 387–415, 2009.

[22] A. L. Custódio, K. Scheinberg, and L. N. Vicente. Chapter 37: Methodologies and software for derivative-free optimization. In *Advances and Trends in Optimization with Engineering Applications*, pages 495–506. SIAM, 2017.

[23] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Math. Program.*, 91, 1 2002.

[24] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia. Incorporating second-order functional knowledge for better option pricing. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[25] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-Weighted linear units for neural network function approximation in reinforcement learning. *arXiv e-prints*, art. arXiv:1702.03118, February 2017.

[26] G. Fasano, J. L. Morales, and J. Nocedal. On the geometry phase in model-based algorithms for derivative-free optimization. *Optim. Methods Softw.*, 24:145–154, 2009.

[27] G. Fasano, G. Liuzzi, S. Lucidi, and F. Rinaldi. A linesearch-based derivative-free approach for nonsmooth constrained optimization. *SIAM J. Optim.*, 24:959–992, 2014.

[28] T. Giovannelli, G. Liuzzi, S. Lucidi, and F. Rinaldi. Derivative-free methods for mixed-integer nonsmooth constrained optimization. *Comput. Optim. Appl.*, 82:293–327, 2022.

[29] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[31] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[32] N. Gould, D. Orban, and P. Toint. CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. and Appl.*, 60:545–557, 2015.

[33] S. Gratton, C. W. Royer, L. N. Vicente, and Z. Zhang. Direct search based on probabilistic descent. *SIAM J. Optim.*, 25:1515–1541, 2015.

[34] S. Gratton, C. Royer, and L. N. Vicente. A decoupled first/second-order steps technique for nonconvex nonlinear unconstrained optimization with improved complexity bounds. *Math. Program.*, 179:1–28, 09 2018.

[35] H.-M. Gutmann. A radial basis function method for global optimization. *J. Global Optim.*, 19:201–227, 2001.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *arXiv e-prints*, art. arXiv:1502.01852, February 2015.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[38] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[39] A. D. Jagtap and G. E. Karniadakis. How important are activation functions in regression and classification? A survey, performance comparison, and future directions. *arXiv e-prints*, art. arXiv:2209.02681, September 2022.

[40] I. Jahan, Md. F. Ahmed, Md. O. Ali, and Y. M. Jang. Self-gated rectified linear unit for performance improvement of deep neural networks. *ICT Express*, 9:320–325, 2023. ISSN 2405-9595.

[41] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, 2009.

[42] B. L. Kalman and S. C. Kwasny. Why tanh: Choosing a sigmoidal function. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 578–581 vol.4, 1992.

[43] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv e-prints*, art. arXiv:1412.6980, December 2014.

[44] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Rev.*, 45:385–482, 2003.

[45] J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numer.*, 28:287–404, 2019.

[46] G. Liuzzi, S. Lucidi, F. Rinaldi, and L. N. Vicente. Trust-region methods for the derivative-free optimization of nonsmooth black-box functions. *SIAM J. Optim.*, 29:3012–3035, 2019.

[47] L. Lu. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28:1671–1706, June 2020.

[48] S. Lucidi and M. Sciandrone. On the global convergence of derivative-free methods for unconstrained optimization. *SIAM J. Optim.*, 13:97–116, 2002.

[49] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[50] F. Manessi and A. Rozza. Learning combinations of activation functions. *arXiv e-prints*, art. arXiv:1801.09403, January 2018.

[51] D. B. McDonald, W. J. Grantham, W. L. Tabor, and M. J. Murphy. Global and local optimization using radial basis function response surface models. *Applied Mathematical Modelling*, 31:2095–2110, 2007.

[52] M. C. Messner. Convolutional neural network surrogate models for the mechanical properties of periodic structures. *Journal of Mechanical Design*, 142, 10 2019.

[53] J. J. Moré and S. M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM J. Optim.*, 20:172–191, 2009.

[54] J. J. Moré and S. M. Wild. Estimating computational noise. *SIAM J. Sci. Comput.*, 33: 1292–1314, 2011.

[55] J. J. Moré and S. M. Wild. Estimating derivatives of noisy simulations. *ACM Trans. Math. Softw.*, 38, apr 2012.

[56] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, Berlin, second edition, 2006.

[57] I. Pan, M. Babaei, A. Korre, and S. Durucan. Artificial neural network based surrogate modelling for multi-objective optimisation of geological CO2 storage operations. *Energy Procedia*, 63:3483–3491, 2014. 12th International Conference on Greenhouse Gas Control Technologies, GHGT-12.

[58] V. Papadopoulos, G. Soimiris, D.G. Giovanis, and M. Papadrakakis. A neural network-based surrogate model for carbon nanotubes with geometric nonlinearities. *Computer Methods in Applied Mechanics and Engineering*, 328:411–430, 2018.

[59] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[60] M. J. D. Powell. On the Lagrange functions of quadratic models that are defined by interpolation. *Optim. Methods Softw.*, 16:289–309, 2001.

[61] M. J. D. Powell. UOBYQA: Unconstrained optimization by quadratic approximation. *Math. Program.*, 92:555–582, 2002.

[62] M. J. D. Powell. Least Frobenius norm updating of quadratic models that satisfy interpolation conditions. *Math. Program.*, 100:183–215, 2004.

[63] M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivatives. Technical Report DAMTP 2004/NA08, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 2004.

[64] F. J. Richards. A flexible growth function for empirical use. *Journal of Experimental Botany*, 10:290–301, 1959.

[65] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[66] Y. Ruan, Y. Xiong, S. Reddi, S. Kumar, and C. Hsieh. Learning to learn by zeroth-order oracle. *arXiv e-prints*, art. arXiv:1910.09464, October 2019.

[67] D. E. Rumelhart, G. E. Hinton, and R. J Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[68] S. Scardapane, M. Scarpiniti, D. Comminiello, and A. Uncini. Learning activation functions from data using cubic spline interpolation. *arXiv e-prints*, art. arXiv:1605.05509, May 2016.

[69] H. J. M. Shi, M. Q. Xuan, F. Oztoprak, and J. Nocedal. On the numerical performance of finite-difference-based methods for derivative-free optimization. *Optim. Methods Softw.*, 38:289–311, 2023.

[70] K. Slimani, M. Zaaf, and T. Balan. Accurate surrogate models for the flat rolling process. *International Journal of Material Forming*, 16, 03 2023.

[71] J. V. Soares do Amaral, J. A. Barra Montevechi, R. de Carvalho Miranda, and W. T. de Sousa Junior. Metamodel-based simulation optimization: A systematic literature review. *Simulation Modelling Practice and Theory*, 114:102403, 2022.

[72] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Optim.*, 7:1–25, 1997.

[73] L. N. Vicente and A. L. Custódio. Analysis of direct searches for discontinuous functions. *Math. Program.*, 133:299–325, 2012.

[74] S. M. Wild, R. G. Regis, and C. A. Shoemaker. ORBIT: Optimization by radial basis function interpolation in trust-regions. *SIAM J. Sci. Comput.*, 30:3197–3219, 2008.

[75] D. Winfield. *Function and Functional Optimization by Interpolation in Data Tables*. PhD thesis, Harvard University, USA, 1969.

[76] D. Winfield. Function minimization by interpolation in a data set. *J. Inst. Math. Appl.*, 12:339–347, 1973.

[77] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv e-prints*, art. arXiv:1505.00853, May 2015.

[78] B. Yuen, M. T. Hoang, X. Dong, and T. Lu. Universal activation function for machine learning. *Scientific Reports*, 11:18757, September 2021.

[79] M. Zaborski and J. Mańdziuk. LQ-R-SHADE: R-SHADE with quadratic surrogate model. In *Artificial Intelligence and Soft Computing: 21st International Conference, ICAISC 2022, Zakopane, Poland, June 19–23, 2022, Proceedings, Part I*, page 265–276, Berlin, Heidelberg, 2023. Springer-Verlag.

[80] H. Zheng, Z. Yang, W. Liu, J. Liang, and Y. Li. Improving deep neural networks using softplus units. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–4, 2015.

# A Appendix: Test Problems from CUTEst

Tables 1 and 2 below list the two sets of nonlinear unconstrained optimization problems from the CUTEst library [32] used throughout the paper and described in Subsection 2.2.

| | | | |
|---|---|---|---|
| ARGLINA | ARGTRIGLS | ARWHEAD | BDEXP |
| BOXPOWER | BROWNAL | COSINE | CURLY10 |
| DQRTIC | DIXON3DQ | ENGVAL1 | EXTROSNB |
| FLETBV3M | FLETCBV3 | FLETCHBV | FLETCHCR |
| FREUROTH | INDEFM | MANCINO | MOREBV |
| NONCVXU2 | NONCVXUN | NONDIA | NONDQUAR |
| PENALTY2 | POWER | QING | QUARTC |
| SENSORS | SINQUAD | SCURLY10 | SCURLY20 |
| SPARSINE | SPARSQUR | SSBRYBND | TRIDIA |
| TRIGON1 | TOINTGSS | | |

Table 1: Names of the 38 CUTEst problems with user-defined dimensions in the first set.

# B Appendix: Full-Low Evaluation Methods

In DFO, two main algorithmic paradigms are employed when designing numerical algorithms, namely, *directional* and *model-based* approaches [20]. In directional algorithms, a set of directions is generated to determine a point that guarantees a (possibly sufficient) decrease condition of the objective function among a set of polling points, which are obtained from the current iterate by considering certain stepsizes along these directions. When only objective function values are used, without approximating the gradients or constructing models, the resulting algorithms are referred to as directional direct-search methods [3, 12, 28, 33, 44, 72]. Such algorithms are able to converge even when applied to problems with a non-smooth objective function [3, 73]. Within this class of methods, one can consider deterministic variants [44, 72] or probabilistic ones [33]. Deterministic variants rely on positive spanning sets of vectors, where at least one

| Name | Dimension | Name | Dimension | Name | Dimension |
|---|---|---|---|---|---|
| ALLINITU | 4 | BARD | 3 | BIGGS6 | 6 |
| BOX3 | 3 | BROYDN7D | 10 | BRYBND | 10 |
| CUBE | 2 | DENSCHND | 3 | DENSCHNE | 3 |
| DIXMAANA1 | 15 | DIXMAANB | 15 | DIXMAANC | 15 |
| DIXMAAND | 15 | DIXMAANE1 | 15 | DIXMAANF | 15 |
| DIXMAANG | 15 | DIXMAANH | 15 | DIXMAANI1 | 15 |
| DIXMAANJ | 15 | DIXMAANK | 15 | DIXMAANL | 15 |
| ENGVAL2 | 3 | ERRINROS | 25 | EXPFIT | 2 |
| FMINSURF | 16 | GROWTHLS | 3 | GULF | 3 |
| HAIRY | 2 | HATFLDD | 3 | HATFLDE | 3 |
| HEART6LS | 6 | HEART8LS | 8 | HELIX | 3 |
| HIELOW | 3 | HIMMELBB | 2 | HIMMELBG | 2 |
| HUMPS | 2 | KOWOSB | 4 | LOGHAIRY | 2 |
| MARATOSB | 2 | MEYER3 | 3 | MSQRTALS | 4 |
| MSQRTBLS | 9 | OSBORNEA | 5 | OSBORNEB | 11 |
| PENALTY3 | 50 | SNAIL | 2 | SPMSRTLS | 28 |
| STRATEC | 10 | VIBRBEAM | 8 | WATSON | 12 |
| WOODS | 4 | YFITU | 3 | | |

Table 2: Names and corresponding dimensions of the 53 CUTEst problems in the second set.

of them is a descent direction. Probabilistic variants use randomly generated directions that are probabilistically descent. To determine the value of the stepsize, DFO algorithms can use line searches along a prespecified set of directions [27, 48] or along directions that are obtained by applying an FD scheme to approximate the gradient of the objective function [6, 7]. In noisy settings, such approximations can be unreliable unless careful estimation of the noise is considered [54, 55].

In model-based algorithms, a model of the objective function is built by using function values at previous iterates or at points randomly sampled [17, 21, 26, 46, 63, 76]. Such methods rely on interpolation or regression techniques and use basis functions such as quadratic polynomials or radial basis functions. The resulting models can be used as a local approximation of the objective function to capture its curvature. A drawback of model-based algorithms is that their performance gets worse as the number of variables increases because of the dense linear algebra of the interpolation and the ill-conditioning of the sample sets [19]. Moreover, noisy and non-smooth settings remain a challenge for such algorithms.

The FLE method introduced in Section 4 combines the benefits of both directional and model-based algorithms and exhibited superior overall performance compared to interpolation-based methods and combinations of these techniques with direct search [9]. In the FLE method, the first iteration is of FE type and is given by Algorithm 1. An FE iteration is no longer deemed successful when the stepsize $\beta_k$ becomes too small, i.e.,

$$\beta \ < \ \lambda\rho(\alpha_k), \tag{B.1}$$

where $\lambda > 0$ and $\rho(\alpha_k)$ is a forcing function depending on the stepsize used in LE iterations, denoted as $\alpha_k$. In such a case, one switches to an LE iteration, which consists of a direct-search step.

The schema of an LE iteration is reported in Algorithm 3. At each iteration, the stepsize $\alpha_k$ is required to satisfy the following sufficient decrease condition

$$f(x_k + \alpha_k d_k) \;\leq\; f(x_k) - \rho(\alpha_k), \tag{B.2}$$

where $\rho(\alpha_k)$ is the same forcing function used in (B.1) and $d_k \in D_k$ is a polling direction. In the practical FLE method proposed in [9], the set of polling directions $D_k$ is given by a direction $d \in \mathbb{R}^n$ uniformly generated on the unit sphere of $\mathbb{R}^n$ and its negative, i.e., $D = [d, -d]$. An LE iteration is deemed successful when a stepsize satisfying (B.2) is found. One can switch from an LE iteration to an FE iteration when the number of unsuccessful consecutive LE iterations becomes greater than the number of line-search backtracks done in the previous FE iteration.

**Algorithm 3** `Low-Eval Iteration`: Direct Search

**Input:** Iterate $x_k$ and stepsize $\alpha_k$. Direct-search parameters $\lambda \geq 1$ and $\theta \in (0, 1)$.

1: Generate a finite set $D_k$ of non-zero polling directions.
2: **If** (B.2) is true for some $d_k \in D_k$, set $x_{k+1} = x_k + \alpha_k d_k$ and $\alpha_{k+1} = \lambda \alpha_k$.
3: **Else** set $x_{k+1} = x_k$ and $\alpha_{k+1} = \theta \alpha_k$.
4: Decide if $t_{k+1} =$ `Low-Eval` or if $t_{k+1} =$ `Full-Eval`.

**Output:** $t_{k+1}$, $x_{k+1}$, and $\alpha_{k+1}$.

For the values of the parameters of the FLE method used in the experiments reported in Section 4, we refer to the practical FLE algorithm in [9].