Sobolev Trained Neural Network Surrogate Models for Optimization

Calvin Tsay*

Department of Computing, Imperial College London

South Kensington, London, SW7 2AZ

May 12, 2021

Abstract

Neural network surrogate models are often used to replace complex mathematical models in black-box and grey-box optimization. This strategy essentially uses samples generated from a complex model to fit a data-driven, reduced-order model more amenable for optimization. Neural network models can be trained in Sobolev spaces, i.e., models are trained to match the complex function not only in terms of output values, but also the values of their derivatives to arbitrary degree. This paper examines the direct impacts of Sobolev training on neural network surrogate models embedded in optimization problems, and proposes a systematic strategy for scaling Sobolev-space targets during NN training. In particular, it is shown that Sobolev training results in surrogate models with more accurate derivatives (in addition to more accurately predicting outputs), with direct benefits in gradient-based optimization. Three case studies demonstrate the approach: black-box optimization of the Himmelblau function, and grey-box optimizations of a two-phase flash separator and two flashes in series. The results show that the advantages of Sobolev training are especially significant in cases of low data volume and/or optimal points near the boundary of the training dataset—areas where NN models traditionally struggle.

Keywords: black-box optimization, grey-box optimization, gradient-enhanced surrogate models

1 Introduction

Detailed mathematical models of process systems are often complex and difficult to embed in optimization problems, owing to nonlinearity, large scale, coupled structure, etc. (Biegler, 2017;

*corresponding author: c.tsay@imperial.ac.uk

Mitsos et al., 2018; Tsay et al., 2018). Therefore, surrogate models are commonly used to approximate these complex models during computational process optimization. New techniques and applications for data-driven surrogate modeling and optimization in process systems engineering (PSE) are of very high interest, and have been recently reviewed in Bhosekar and Ierapetritou (2018); McBride and Sundmacher (2019); Tsay and Baldea (2019a). While surrogate models can be obtained via physics-based model reduction, this work focuses in particular on data-driven surrogate models, or those generated by regressing a model to samples obtained from a more detailed model. The former utilize physical knowledge and may be used to make predictions over relatively wide input domains, but purely data-driven models are best suited for making predictions within/near the training data set, or the samples used to obtain a model. At a high level, data-driven surrogate modeling leverages data from expensive simulations to fit a model that is more tractable and/or amenable to optimization tools.

This work particularly focuses on surrogate models that are incorporated into optimization problems. Such problems seek the values of decision variables that minimize/maximize an objective function, while potentially requiring that certain model equation (constraints) are satisfied. A surrogate model that approximates and replaces an entire optimization model is often termed a "black-box" model. Moreover, surrogate models can be incorporated into a larger model alongside explicit mathematical relationships, often termed a "grey-box" model (Boukouvala et al., 2017; Davis and Ierapetritou, 2009). Furthermore, this work focuses on neural network (NN) surrogate models, given their widespread use in PSE, scalability to high-dimensional problems, and accessibility via many open-source software tools. Previous works demonstrated NNs as surrogate models in black-box optimization (Anna et al., 2017), process synthesis (Henao and Maravelias, 2011; Kim and Boukouvala, 2020), flexibility analysis (Dias and Ierapetritou, 2019; Rogers and Ierapetritou, 2015), dynamic optimization (Jin et al., 2015; Schäfer et al., 2019; Tsay and Baldea, 2020), etc. Optimization of a trained NN model—i.e., optimizing over inputs for a fixed set of model parameters—is commonly formulated as a nonlinear program (NLP), but can also be formulated as a mixed-integer linear program (MILP) when piecewise-linear activation functions are chosen (Grimstad and Andersson, 2019). The end-use in optimization can be considered while constructing NN surrogates, e.g., several works (Chen et al., 2019; Yang and Bequette, 2021) formulate NN-based optimization problems as *convex* NLPs by enforcing constraints to impose input-convexity during NN training.

In addition to being trained on function outputs, function gradients can be used to improve

the accuracy of surrogate models, a paradigm referred to as *gradient-enhanced* surrogate modeling (Laurent et al., 2019). For example, a simple strategy exploiting function gradients creates new data "samples" using a first-order Taylor series approximation around an existing sample. Early works (Liu and Batill, 2000; Sellar and Batill, 1996) compared this Taylor-series approach against a more direct strategy of matching the NN gradients during training. For the latter direct approach, the explicit derivatives of the NN gradients with respect to its learned weights were used; however, the authors only considered NNs with a single hidden layer, noting that it would be cumbersome to express the derivatives for larger NNs. Furthermore, the resulting loss function for training involves two terms—one penalizing error in predicting the function output and the other its derivative—and determining appropriate relative weights for these terms is challenging (Liu and Batill, 2000) . Other works explored a similar direct strategy with radial basis function networks, or RBFNs (Kampolis et al., 2004). Previous works employed gradient-enhanced NN/RBFN models in both derivative-free optimization (e.g., Kampolis et al. (2004); Giannakoglou et al. (2006)) and gradient-based optimization (e.g., Bouhlel et al. (2020); Leary et al. (2004); Sellar and Batill (1996)). The above works approximate the objective function of a *black-box* optimization problem with a surrogate model and directly benefit from using a more accurate surrogate model. In a similar theme, there is increasing interest (e.g., Rackauckas et al. (2020), Raissi et al. (2019)) in improving model accuracy and generalizability by incorporating physics-based knowledge during training; here, note that gradient-enhanced training can potentially be performed without any additional knowledge by using automatic/numerical differentiation tools.

More recently, Czarnecki et al. (2017) showed that state-of-the-art NN software tools, e.g., PyTorch (Paszke et al., 2019), enable training NNs in Sobolev spaces, termed Sobolev-trained NNs. The main idea is to quantify the performance of NN models in Sobolev spaces, i.e., the distance between functions is quantified both by differences in output values and by differences in values of their derivatives to arbitrary degree. In contrast to previous works, NNs comprising *multiple* hidden layers are trained directly using function gradients (first-order derivatives) and also *higher-order derivatives*. When there are multiple inputs, the dimensionalities of higher-order function derivatives grow exponentially; target derivatives can be projected on random vectors to reduce their dimensionality to one (known as stochastic Sobolev training). These preliminary results (Czarnecki et al., 2017; Srinivas and Fleuret, 2018) found Sobolev training to improve the accuracy and generalization ability of NN models, and Cocola and Hand (2020)

examined the convergence behavior of Sobolev training. Moreover, Hornik (1991) showed that NNs can be universal approximators in Sobolev spaces, while Gühring et al. (2020) expanded these results for deep NNs. However, these works focus solely on the ability of Sobolev training to improve the prediction accuracy of surrogate models.

For applications in gradient-based optimization, this work proposes that potential improvements in accuracy of the surrogate model derivatives resulting from Sobolev training are similarly important, noting that first- and/or second-order derivatives of a function are directly used by many modern gradient-based optimization algorithms. Since early works found gradient-enhanced training improves prediction accuracy, this work construes Sobolev training as "gradient-enhanced" training of the gradients of an NN; e.g., the gradients of the first-order derivatives themselves, or the second-order partial derivatives, can improve accuracy of the first-order derivatives. In theory, derivatives of overparameterized NN models can be trained to arbitrary accuracy (Cocola and Hand, 2020; Hornik, 1991), but in practice the accuracies of various derivative orders are highly dependent on their relative weighting during training. Therefore this work examines scaling of the surrogate model derivative terms for training and proposes a strategy for systematically weighting the terms during training. Furthermore, the ability of Sobolev training to improve accuracy of optimization-relevant function derivatives is directly examined. Finally, this work investigates the circumstances under which Sobolev training benefits surrogate-model optimization, including the use of Sobolev-trained NNs in grey-box process optimization.

In summary, the novelty of this work therefore comprises:

- An in-depth examination of the optimization-related benefits of Sobolev training and stochastic Sobolev training for NN surrogate models
- A simple strategy for systematically scaling targets in Sobolev training, and its connection to previous heuristic "weighting" approaches for gradient-enhanced NNs
- Computational studies of both single- and multiple-output Sobolev-trained NNs for datasets of varying size in black-box and grey-box optimization

Previous works on Sobolev and gradient-enhanced training focused solely on prediction accuracy, neglecting scaling of derivative terms or scaling them by trial-and-error. A few works employed gradient-enhanced models in optimization, but these were limited to black-box optimization, and the accuracies of model derivatives were not examined. While this work considers (local)

gradient-based optimization, note that deploying surrogate models in the context of deterministic global optimization, e.g., (Mistry et al., 2020; Schweidtmann and Mitsos, 2019), typically relies on gradient-based local solvers to find feasible points (upper bounds) in the form of local optima. Furthermore, surrogate models can be deployed alongside adaptive sampling and/or derivative-free optimization techniques (Boukouvala and Floudas, 2017; Eason and Cremaschi, 2014; Thebelt et al., 2020), but this work focuses on surrogate models generated from a fixed dataset for a direct comparison of conventional vs. Sobolev-trained NNs.

2 Background: Neural Network Surrogate Models

Neural network (NN) surrogate models are trained to approximate a generic function $f(\boldsymbol{x})$. A feed-forward NN (also known as a multilayer perceptron) comprises one or more layers, with each layer defined as:

$$\tilde{\boldsymbol{x}}_l = f_l(\mathbf{W}_l \tilde{\boldsymbol{x}}_{l-1} + \boldsymbol{b}_l) \quad \forall l = 1, ..., L \tag{1}$$

where $f_l : \mathbb{R}^l \to \mathbb{R}^l$ is the activation function of the l^{th} layer, and $\mathbf{W}_l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $\boldsymbol{b}_l \in \mathbb{R}^{N_l}$ are the weights and biases of layer l, respectively. N_l represents the number of nodes in layer l. The input to each layer l is the output vector of the previous layer, $\tilde{\boldsymbol{x}}_{l-1}$. The output vector of the final layer $\tilde{\boldsymbol{x}}_L$ represents the output (prediction) of the full model ($\hat{\boldsymbol{y}} = \tilde{\boldsymbol{x}}_L$), while the input to the first layer $\tilde{\boldsymbol{x}}_0$ represents the input vector $\boldsymbol{x}$. The size and representation ability of an NN model are determined by the number of layers L, as well as the number of nodes in each layer, $N_l,\ l = 1, ..., L$. The results in this work use the popular sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2}$$

Note that the universal approximator theorem is easily extended to function derivatives (i.e., Sobolev spaces), when the activation function f_l is j-times continuously differentiable, where j is the highest-order derivative represented to arbitrary accuracy (Hornik, 1991).

For brevity, we refer to the entire input-output NN model as $\hat{\boldsymbol{y}} = f_{\text{NN}}(\boldsymbol{x})$ herein. Generally, the NN is trained by regressing the parameters ($[\mathbf{W}_l, \boldsymbol{b}_l], l = 1, ..., L$) to minimize a loss function that describes the model accuracy:

$$J = \sum_{n=1}^{N_s} \ell\Big(f(\boldsymbol{x}_n), f_{\text{NN}}(\boldsymbol{x}_n)\Big) \tag{3}$$

where N_s is the number of samples and $[\boldsymbol{x}_n, \boldsymbol{y}_n = f(\boldsymbol{x}_n)]$ is the n^{th} input-output pair. Common

choices for ℓ in regression problems include mean absolute error (MAE) and mean squared error (MSE). Regularization terms that penalize the magnitude of the parameters (e.g., L1- or L2-norm regularization) are sometimes added to the loss function (3), but are not considered herein. Denoting now the combined vector of trainable weights as $\boldsymbol{\theta} \in \mathbb{R}^{N_\theta}$, the regression problem seeks $\boldsymbol{\theta} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J$. This problem is often very large, i.e., $N_\theta >> 1$ and/or $N_s >> 1$, making (stochastic) gradient descent the solution method of choice. Therefore, the derivatives $\partial J/\partial \boldsymbol{\theta}$ should be readily available, and many software tools, e.g., Pytorch (Paszke et al., 2019), include automatic differentiation capabilities that compute derivatives using back-propagation.

3 Sobolev-Trained Neural Networks for Optimization

3.1 Optimization of Trained Neural Networks

This work considers the optimization of some trained NN model(s) formulated as a nonlinear program (NLP); however, note that Sobolev training can also be used for NNs with ReLU activation functions (Czarnecki et al., 2017), which can be formulated as MILPs, e.g., see Grimstad and Andersson (2019).

Consider the general NLP:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \phi(\boldsymbol{x}) \\ s.t. \quad & g(\boldsymbol{x}) \leq 0 \\ & h(\boldsymbol{x}) = 0 \end{aligned} \tag{4}$$

The optimum of (4) is defined by the Karush-Kuhn-Tucker (KKT) conditions:

$$D_{\boldsymbol{x}}\phi(\boldsymbol{x}^*) + \mu D_{\boldsymbol{x}} g(\boldsymbol{x}^*) + \lambda D_{\boldsymbol{x}} h(\boldsymbol{x}^*) = 0 \tag{5}$$

$$g(\boldsymbol{x}^*) \leq 0 \tag{6}$$

$$h(\boldsymbol{x}^*) = 0 \tag{7}$$

$$\mu \geq 0 \tag{8}$$

$$\mu^T g(\boldsymbol{x}^*) = 0 \tag{9}$$

where $\boldsymbol{x}^*$ is the optimal solution and μ and λ are (vectors of) KKT multipliers. NN surrogates can be deployed in *unconstrained black-box* optimization, which seeks the minimum of a function ϕ (equivalent to maximizing ϕ negated). By approximating $\phi(\boldsymbol{x})$ with $f_{\text{NN}}(\boldsymbol{x})$ (with fixed parameters $\boldsymbol{\theta}$), the problem (4) is reduced to $\min_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x})$, and the stationarity criterion (5) becomes $D_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}^*) = 0$, as there no constraints $g(\boldsymbol{x})$ and $h(\boldsymbol{x})$. Therefore, in this con-

text, the accuracy of $D_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x})$ is crucial to identifying the correct optimum $\boldsymbol{x}^*$ that satisfies the optimality criteria for the original function $\phi(\boldsymbol{x})$. Note that while only the accuracy at $D_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}) = 0$ is important at the optimal solution, accuracy at $D_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}) \neq 0$ may be important for convergence to the optimal solution $\boldsymbol{x}^*$ (Biegler et al., 1985; Biegler, 2017; Caballero and Grossmann, 2008). Previous works (Biegler et al., 1985; Biegler, 2017) noted that matching the gradients of the surrogate and true models *at all points* is a sufficient condition for optimization convergence. To this end, trust region methods, e.g., Agarwal and Biegler (2013); Henao and Maravelias (2011), have been shown to improve optimization convergence.

On the other hand, *grey-box* optimization deploys the surrogate model f_{NN} alongside other (first-principles) equations in (4). For example, a surrogate model f_{NN} can replace the objective function ϕ or a constraint in g or h. In this case, the gradient $D_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x})$ is still involved in the stationarity criterion (5) and remains important for identifying the proper solution $\boldsymbol{x}^*$. However, since multiple terms are present in (5), this will not necessarily occur at $D_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}) = 0$. Moreover, accuracy of the prediction itself $f_{\text{NN}}(\boldsymbol{x})$ is crucial if used in g or h owing to its inclusion in the primal feasibility equations (6)–(7) and complementarity slackness (9) (when specifically g involves f_{NN}).

3.2 Sobolev Training

The previous section highlights the importance of function derivatives with respect to the input(s), i.e., $D_{\boldsymbol{x}} f(\boldsymbol{x})$, to surrogate-model optimization. These derivatives may themselves be available in addition to the output values $f(\boldsymbol{x})$ when training a surrogate model, either in symbolic form or via automatic/numerical differentiation tools. In this context, Sobolev training seeks to match the derivatives of the surrogate model with those of the true function. Specifically, training is performed on samples comprising the traditional input-output pairs $[\boldsymbol{x}_n, f(\boldsymbol{x}_n)]$, as well as some j-th order derivatives, denoted as $D^j_{\boldsymbol{x}} f(\boldsymbol{x}_n)$. As noted above, most software implementations for NN models already include differentiation capabilities, and therefore NN surrogates are a natural candidate. The NN derivatives $D^j_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}_n)$ can be obtained using similar techniques to those for finding the derivatives $D_{\boldsymbol{\theta}} f_{\text{NN}}(\boldsymbol{x}_n)$ used in training.

Sobolev training for NN models involves appending the typical loss function (3) with a loss function defined in terms of j-th order derivatives, resulting in:

$$J_{\text{ST}} = \sum_{n=1}^{N_s} \ell\Big(f(\boldsymbol{x}_n), f_{\text{NN}}(\boldsymbol{x}_n)\Big) + \sum_{j \in \mathcal{J}} \sum_{n=1}^{N_s} \omega_j \ell\Big(D^j_{\boldsymbol{x}} f(\boldsymbol{x}_n), D^j_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}_n)\Big) \tag{10}$$

where $\mathcal{J}$ is the set of available/considered derivative orders, and $\omega_j,\ j \in \mathcal{J}$ are weights selected such that the terms have similar magnitude. It can be easily seen that the "direct approach" for gradient-enhanced NNs (Liu and Batill, 2000; Sellar and Batill, 1996) is a specific case of Sobolev training, where $\mathcal{J} = \{1\}$. The rest of this paper refers to this case of $\mathcal{J} = \{1\}$ as *first-order* Sobolev training (denoted as ST-1), and to the case of $\mathcal{J} = \{1, 2\}$ as *second-order* Sobolev training (denoted as ST-2). Figure 1 depicts normal, ST-1, and ST-2 training for NN surrogate models.

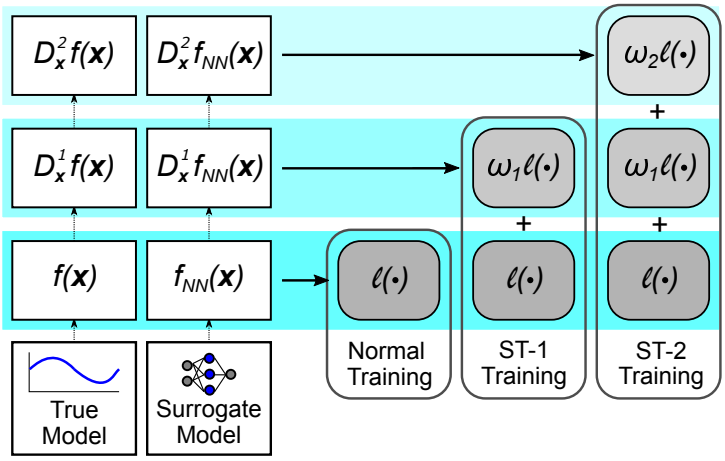

Figure 1: Visual comparison of normal vs first- and second-order Sobolev training for surrogate models.

When the dimensionality of $\boldsymbol{x}$, defined as $\dim(\boldsymbol{x}) = k$, is large, training a NN to minimize (10) can become computationally difficult. Note that for a single output the Jacobian $D^1_{\boldsymbol{x}} f(\boldsymbol{x})$ has dimensionality k, and the first-order Sobolev training term of the loss function (10) becomes:

$$\sum_{n=1}^{N_s} \omega_1 \ell\Big(D^1_{\boldsymbol{x}} f(\boldsymbol{x}_n), D^1_{\boldsymbol{x}} f_{\mathrm{NN}}(\boldsymbol{x}_n)\Big) = \sum_{n=1}^{N_s} \omega_1 \ell\Big([D^1_{x_1} f(\boldsymbol{x}_n), ..., D^1_{x_k} f(\boldsymbol{x}_n)], [D^1_{x_1} f_{\mathrm{NN}}(\boldsymbol{x}_n), ..., D^1_{x_k} f_{\mathrm{NN}}(\boldsymbol{x}_n)]\Big) \quad (11)$$

Common choices are to aggregate or average the loss function ℓ across multiple dimensions. The former makes the weight of each partial derivative residual equal to that of the function output, while the latter makes the total weight of the partial j-th order derivatives equal that of the function output. Continuing past the Jacobian, the Hessian $D^2_{\boldsymbol{x}} f(\boldsymbol{x})$ has dimensionality $k \times k$, and, in general, $D^j_{\boldsymbol{x}} f(\boldsymbol{x})$ has dimensionality k^j.

Because the dimensionality of the Sobolev training function grows exponentially, a stochastic implementation of can approximate the loss function (10) while avoiding the curse of dimen-

sionality (Czarnecki et al., 2017):

$$J_{\text{SST}} = \sum_{n=1}^{N_s} \ell\Big(f(\boldsymbol{x}_n), f_{\text{NN}}(\boldsymbol{x}_n)\Big) + \sum_{j\in\mathcal{J}} \sum_{n=1}^{N_s} \omega_j \ell\Big(\langle D^j_{\boldsymbol{x}} f(\boldsymbol{x}_n), \boldsymbol{v}\rangle, \langle D^j_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}_n), \boldsymbol{v}\rangle\Big) \tag{12}$$

where $\boldsymbol{v}$ is a random vector sampled uniformly from the unit (hyper)sphere. Minimizing the stochastic Sobolev loss function (12) matches the *projections* of derivatives of $f(\boldsymbol{x})$ on random vectors to those of the NN surrogate. Therefore, the approximation approaches (10) with increasing number of sampled vectors $\boldsymbol{v}$; however, dimensionality of the problem is constrained by only using a single sample of $\boldsymbol{v}$ per sample $\boldsymbol{x}_n$ during training. Note that stochastic training reduces the number of derivatives computed, as the j-th order term is approximated as $D_{\boldsymbol{x}}\langle D^{j-1}_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}_n), \boldsymbol{v}\rangle \in \mathbb{R}^k$ rather than $D^j_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x}_n) \in \mathbb{R}^{k^j}$.

Overall, Sobolev training can be beneficial in the data generation and/or the model training phases of surrogate model creation. During data generation, it may be computationally cheaper to generate function derivatives rather than generate more samples. As the case studies later show, Sobolev training on sampled gradients can result in NN models comparable to those trained on orders-of-magnitude more samples of function outputs only. Likewise, assuming a constant batch size, adding samples increases the length of each training epoch during batch gradient descent (i.e., the model is trained on every sample once), but adding derivatives during Sobolev training does not increase the number of samples. Rather, the computational cost of evaluating the augmented loss function is increased, as the derivatives of the NN model must be calculated. Compared to increasing the number of samples, this additional cost could potentially be offset by having shorter training epochs.

3.3 Simple Example: One-Dimensional Function

This section demonstrates the concept of Sobolev training using a simple one-dimensional function. NN surrogates are trained to approximate the simple function using first- and second-order Sobolev training:

$$f(x) = x \times \sin(x) \tag{13}$$

$$D^1_x f(x) = x \times \cos(x) + \sin(x) \tag{14}$$

$$D^2_x f(x) = 2\cos(x) - x \times \sin(x) \tag{15}$$

A training set of five equally spaced interior samples is drawn from the domain $x \in [0, 10]$.

The NN models comprise a single input, a single hidden layer with ten nodes, and a single output node. Sigmoid activation functions are used for the hidden layer, and all outputs are scaled to $[0, 1]$ for training. All NNs are implemented in Pytorch (Paszke et al., 2019) and trained for 500 epochs with mean squared error (MSE) as the loss function using the `Adam` optimizer. Figure 2 shows the predictions of ten randomly initialized NN models (parameters are initialized using $\theta \sim Normal(0, 2)$) trained using standard, ST-1, and ST-2 training.

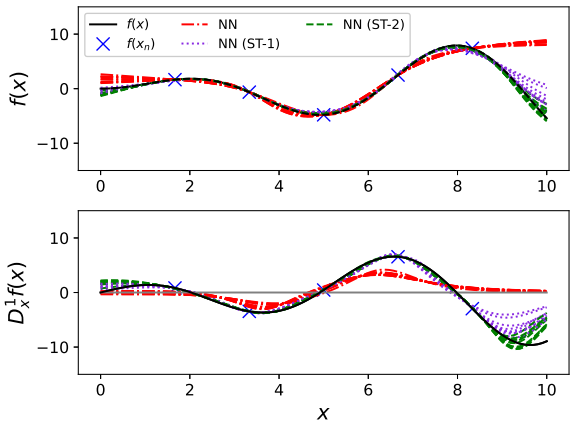

Figure 2: Ten randomly initialized NN models trained on simple example function.

The trained surrogate models interpolate the true function $f(x)$ well, i.e., they accurately represent $f(x)$ within the sampled range. However, as expected of most data-driven models, the standard-trained NNs do not extrapolate accurately outside of the boundary samples. The mean MSEs on $f(x)$ over 100 equally spaced points in the input domain for the models are 10.80, 0.49, and 0.17 for standard, ST-1, and ST-2 training, respectively. Moreover, the derivatives of the standard-trained NNs differ significantly from those of the true function. For example, locating the global maximum of $f(x)$ near $x = 8$ in the context of black-box optimization would require finding $D_x^1 f(x^*) = 0$, but Figure 2 shows that the derivatives $D_x^1 f_{\text{NN}}(x)$ do not cross zero in this region. Therefore, an optimization routine would likely not converge to this true maximum.

On the other hand, the Sobolev-trained NN models are able to accurately extrapolate the true function for a small window outside of the sampled points, as their training included the derivatives, or rate(s) of change, at the boundaries of the sampled area. Furthermore, the derivatives of the Sobolev-trained NN models lie much closer to that of the true model, and the derivatives notably reach a zero near the global optimum of $f(x)$ (Figure 2). ST-2 training further improves accuracy of model derivatives: the mean MSEs on $D_x^1 f(x)$ over 100 equally

spaced points in the input domain for the models are 14.20, 1.72, and 0.70 for standard, ST-1, and ST-2 training, respectively.

3.4 Scaling of Derivatives

While several studies noted the challenge of scaling the terms in gradient-enhanced NNs (selecting ω_1 in this case), specific strategies have only been minimally discussed. In particular, function outputs $\boldsymbol{y}$ are typically scaled linearly (e.g., min-max scaling, normalization) before training a surrogate model. For example, the scaled values $\bar{\boldsymbol{y}}$ can be bounded to $[LB_y, UB_y]$, where LB_y and UB_y are vectors defining the minimum and maximum scaled values for each output. The choice of activation function(s) in the output layer influences these bounds; a common choice is $LB_y = \mathbf{0}$ and $UB_y = \mathbf{1}$. Nevertheless, such scaling does not guarantee consistent scaling of the function derivatives, which can remain of arbitrary orders of magnitude.

Therefore, this work proposes a simple and intuitive method to consistently scale all available derivatives. In particular, each j-th order derivative is treated as a separate output/target and is scaled independently. Consider the case of a single input x and a single output $f(x)$, both scaled such that $\bar{f}(x_n),\ \bar{x}_n \in [0,1]$. The j-th derivative of the scaled output is

$$D^j_{\bar{x}} \bar{f}(\bar{x}_n) = \left.\frac{\partial^j \bar{f}}{\partial \bar{x}^j}\right|_{\bar{x}_n} = \frac{\text{range}(x_n)^j}{\text{range}(f(x_n))} \left.\frac{\partial^j f}{\partial x^j}\right|_{x_n} \tag{16}$$

where $\text{range}(x_n) \coloneqq \max(x_n) - \min(x_n)$. The sampled derivative values can then also be scaled such that $\bar{D}^j_{\bar{x}} \bar{f}(\bar{x}_n) \in [0,1]$:

$$\bar{D}^j_{\bar{x}} \bar{f}(\bar{x}_n) \coloneqq \alpha \frac{\partial^j \bar{f}}{\partial \bar{x}^j}|_{\bar{x}_n} + \beta \tag{17}$$

where $\alpha = \text{range}(\bar{D}^j_{\bar{x}} \bar{f}(\bar{x}_n))$ and $\beta = -\min(\bar{D}^j_{\bar{x}} \bar{f}(\bar{x}_n))$. The scaled derivatives of all orders $j \in \mathcal{J}$ then all lie in the same domain as the scaled function output, and the weights ω_j can be set to unity and omitted from (10) and (12). This scaling enforces that the terms in the objective function, (10) or (12), are of the same order of magnitude, and that NN training is not dominated by matching a particular j-th order derivative. This scaling assigns equal weights to the loss computed from derivatives of all orders considered. While for other machine-learning applications it may be preferable to weight output accuracy more heavily, for the purposes of optimization we also desire accurate function derivatives.

Note that this scaling is not possible for the j-th order derivatives if $\frac{\partial^j f}{\partial x^j}$ is constant; such derivatives can be omitted during training or scaled such that $\bar{D}^j_{\bar{x}} \bar{f}(\bar{x}_n) = \frac{\partial^j \bar{f}}{\partial \bar{x}^j}|_{\bar{x}_n} = \beta \in [0,1]$. Furthermore, this scaling approach assumes that the considered derivatives are defined at all

sampled points, which is a general requirement to evaluate Sobolev-training loss functions (10) or (12) for most choices of $\ell(\cdot)$.

Connection to Objective Function Scaling. For many cases this strategy for scaling j-th order derivatives provides a method for selecting ω_j, a challenge addressed by lengthy trial-and-error by previous works, e.g., Liu and Batill (2000); Sellar and Batill (1996). Specifically, the scaling factor in (16) is independent of sample x_n, and the Sobolev-training terms of the loss function become

$$\ell\Big(\bar{D}^j_{\bar{x}}\bar{f}(\bar{x}_n), \bar{D}^j_{\bar{x}}f_{\mathrm{NN}}(\bar{x}_n)\Big) = \ell\Big(\frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}D^j_{\boldsymbol{x}}f(\boldsymbol{x}_n)+\beta, \frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}D^j_{\boldsymbol{x}}f'_{\mathrm{NN}}(\boldsymbol{x}_n)+\beta\Big) \quad (18)$$

where f'_{NN} represents unscaled predictions from an NN surrogate. The scaling terms on the right-hand side of (18) can be isolated from $\ell(\cdot)$ in many cases, to give an equivalent ω_j. For example:

$$\begin{aligned}\mathrm{MAE}\Big(\bar{D}^j_{\bar{x}}\bar{f}(\bar{x}_n), \bar{D}^j_{\bar{x}}f_{\mathrm{NN}}(\bar{x}_n)\Big) &= \left|\frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}D^j_{\boldsymbol{x}}f(\boldsymbol{x}_n) - \frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}D^j_{\boldsymbol{x}}f'_{\mathrm{NN}}(\boldsymbol{x}_n)\right| \\ &= \frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}\mathrm{MAE}\Big(D^j_{\boldsymbol{x}}f(\boldsymbol{x}_n), D^j_{\boldsymbol{x}}f'_{\mathrm{NN}}(\boldsymbol{x}_n)\Big) \quad (19)\end{aligned}$$

$$\begin{aligned}\mathrm{MSE}\Big(\bar{D}^j_{\bar{x}}\bar{f}(\bar{x}_n), \bar{D}^j_{\bar{x}}f_{\mathrm{NN}}(\bar{x}_n)\Big) &= \left(\frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}D^j_{\boldsymbol{x}}f(\boldsymbol{x}_n) - \frac{\alpha\cdot\mathrm{range}(x_n)^j}{\mathrm{range}(f(x_n))}D^j_{\boldsymbol{x}}f'_{\mathrm{NN}}(\boldsymbol{x}_n)\right)^2 \\ &= \frac{\alpha^2\cdot\mathrm{range}(x_n)^{2j}}{\mathrm{range}(f(x_n))^2}\mathrm{MSE}\Big(D^j_{\boldsymbol{x}}f(\boldsymbol{x}_n), D^j_{\boldsymbol{x}}f'_{\mathrm{NN}}(\boldsymbol{x}_n)\Big) \quad (20)\end{aligned}$$

Therefore, for the MAE loss function $\omega_j = \alpha\cdot\mathrm{range}(x_n)^j/\mathrm{range}(f(x_n))$ and for MSE $\omega_j = \alpha^2\cdot\mathrm{range}(x_n)^{2j}/\mathrm{range}(f(x_n))^2$. When ω_j cannot be separated explicitly (e.g., Huber loss, log-cosh loss), the proposed strategy still enables Sobolev training on suitably scaled derivatives.

4 Case Study 1: Black-Box Optimization of Himmelblau's Test Function

Himmelblau's function (Himmelblau, 2018) is a well-known test function for optimization, defined mathematically as

$$f(\boldsymbol{x}) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \quad (21)$$

Although this function is simple to express, it exemplifies a non-convex and multimodal response surface for surrogate modeling. Being continuous and smooth, the function is furthermore a good choice for approximation via NN models. The global minima, with $f(\boldsymbol{x}) = 0$, exist at $\boldsymbol{x}^* = \{(3,2), (-2.81, 3.28), (-3.78, -3.28), (3.58, -1.85)\}$. The function contours and minima are shown in Figure 3.

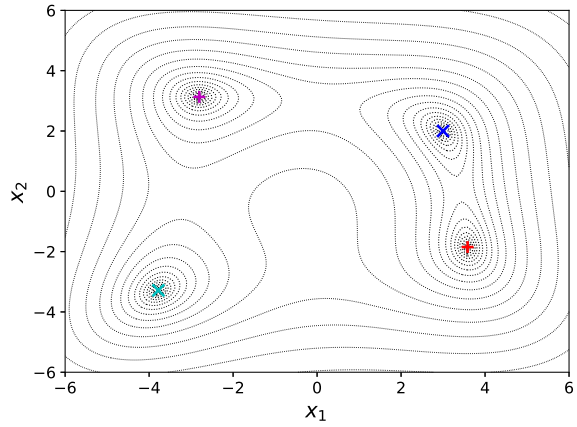

Figure 3: Contours and optima of Himmelblau's test function.

4.1 Surrogate Modeling Results

NN surrogate models were trained to approximate $f(\boldsymbol{x})$ in (21). Quasi-random sampling techniques generally outperform Monte-Carlo and Latin-Hypercube sampling for low-dimensional spaces (Dige and Diwekar, 2018). Therefore, training data were created using standard Sobol sampling over the domain: $x_1, x_2 \in [-6, 6]$. For each training strategy, 100 NN models with two input nodes, three hidden layers with ten nodes each, and a single output node are trained for 2000 epochs using the MSE loss function and `Adam` optimizer in Pytorch. The batch size for training was selected to be $\texttt{floor}(N_s/5)$, which generally improved model accuracy compared to using the entire training set N_s. Sigmoid activation functions are again applied in the hidden layers, and outputs are scaled to $[0, 1]$.

Figure 4 shows the median MSEs and interquartile ranges for each set of 100 models trained with varying N_s, estimated by evaluating both the scaled function $\bar{f}(\bar{\boldsymbol{x}})$ and the surrogate model $f_{\text{NN}}(\bar{\boldsymbol{x}})$ over a meshgrid of 100×100 points. The MSEs of the 1-st order derivatives are also shown, estimated by evaluating $\bar{D}^1_{\bar{\boldsymbol{x}}} \bar{f}(\bar{\boldsymbol{x}})$ and $\bar{D}^1_{\bar{\boldsymbol{x}}} f_{\text{NN}}(\bar{\boldsymbol{x}})$ over the same meshgrid. Outputs and derivatives were scaled using their true min/max (rather than the min/max in the sampled points) while computing MSEs for direct comparison across different numbers of samples. Only the mean MSE of the derivatives with respect to both input variables is shown here for clarity;

element-wise MSEs with respect to the individual derivatives are provided in the Supplementary Information. For each value of N_s, models are trained using standard training, first-order Sobolev training (ST-1), and second-order Sobolev training (ST-2).

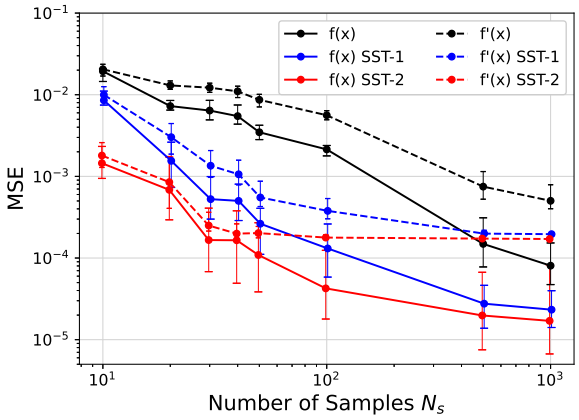

Figure 4: Median squared error (MSE) of NN surrogate models fit to Himmelblau function with Sobolev training. Each point represents the median of the MSEs of 100 trained models, and error bars show interquartile range. MSEs of both function predictions (solid) and model derivatives (dashed) are shown.

NNs trained with first-order Sobolev training (ST-1) predict both the function output and its first derivatives more accurately compared to standard NNs, confirming the approximation benefits of gradient-enhanced NNs. Second-order Sobolev training (ST-2) further improves prediction accuracy for a low number of samples, but, as N_s increases, the accuracies of ST-1 and ST-2 models seem to converge. There are two potential reasons for this: (i) the complicated ST-2 loss function could make training accurate NNs more challenging, especially when N_s is low, as suggested by the larger interquartile ranges, and/or (ii) the accuracies are close to the maximum for NN models of the given size. Note that the cases of low N_s are particularly important/relevant when model samples are expensive and/or the dimensionality of the sampled space is high (i.e., N_s must increase exponentially to maintain the same sample density).

For all cases, the difference in accuracies between function outputs and derivatives seems to increase with increasing N_s; i.e., the quality of derivative approximations improves less than that of the function approximations with increasing number of samples. Nevertheless, ST-2 training results in improved estimates of the function gradients for all tested values of N_s, as suggested by the intuition that incorporating second-order derivatives in training corresponds to "gradient-enhanced" training of both the function and its first derivatives.

As the function has two inputs, the first order derivatives lie in $\mathbb{R}^2$ and the second order derivatives in $\mathbb{R}^{2\times 2}$. The dimensionality of Sobolev training terms can be maintained at one

using the stochastic loss function (12). NN models are again trained to approximate $f(\boldsymbol{x})$, but with stochastic first- and second-order Sobolev training. While the stochastic approach may require more training epochs to improve the estimates of the directional function derivatives, the NNs were trained for the same 2000 epochs for a direct comparison. Figure 5 shows the mean MSEs for each set of 100 models trained via first- (SST-1) and second-order stochastic Sobolev training (SST-2) for the same values of N_s, as computed using the above approach.

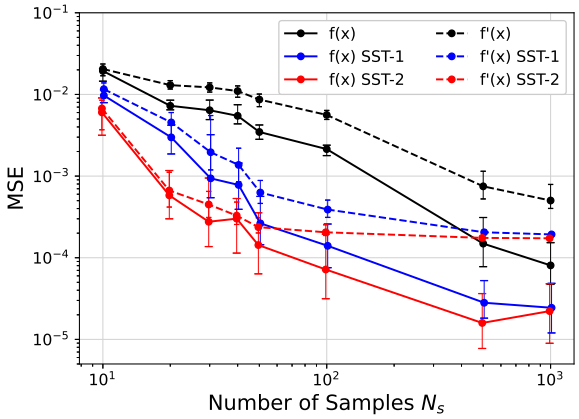

Figure 5: Median squared error (MSE) of NN surrogate models fit to Himmelblau function with stochastic Sobolev training. Each point represents the median of the MSEs of 100 trained models, and error bars show interquartile range. MSEs of both function predictions (solid) and model derivatives (dashed) are shown.

The trends remain largely the same: NNs trained with SST-1 and SST-2 generally are more accurate compared to standard NNs, while the models exhibit increasing difference in accuracies between function outputs and derivatives with increasing N_s. SST-2 training similarly improves estimates of the function gradients for all values of N_s. These results suggest that stochastic Sobolev training approximates standard Sobolev training well: the two strategies result in similar behavior, while SST reduces the dimensionality of the derivative terms during NN training. Most notably, SST does not have a significant impact on the interquartile ranges for model accuracies.

4.2 Optimization Results

The black-box optimization problem $\min_{\boldsymbol{x}} f_{\text{NN}}(\boldsymbol{x})$ was implemented in Pyomo (Hart et al., 2017) and solved using Ipopt version 3.13.3 (Wächter and Biegler, 2006) with default settings. Four optimizations are run for each NN model, starting from the global optima of the true function $\boldsymbol{x}^* = \{(3,2), (-2.81, 3.28), (-3.78, -3.28), (3.58, -1.85)\}$. Note that these initial guesses are not available in practice, but are used here in an attempt to find a local optimum of surrogate

models close to each true optimum.

The distances between the true optima of $\{\boldsymbol{x}_1^*, \boldsymbol{x}_2^*, \boldsymbol{x}_3^*, \boldsymbol{x}_4^*\}$ and the solutions found from the respective optimization problems are shown in Figure 6 for normal, ST-1, ST-2, SST-1, and SST-2 trained NN models. The medians and interquartile ranges of the 100 models obtained using each training procedure are again shown. The 1-norm of distances is shown here for clarity; element-wise optimization results are provided in the Supplementary Information.

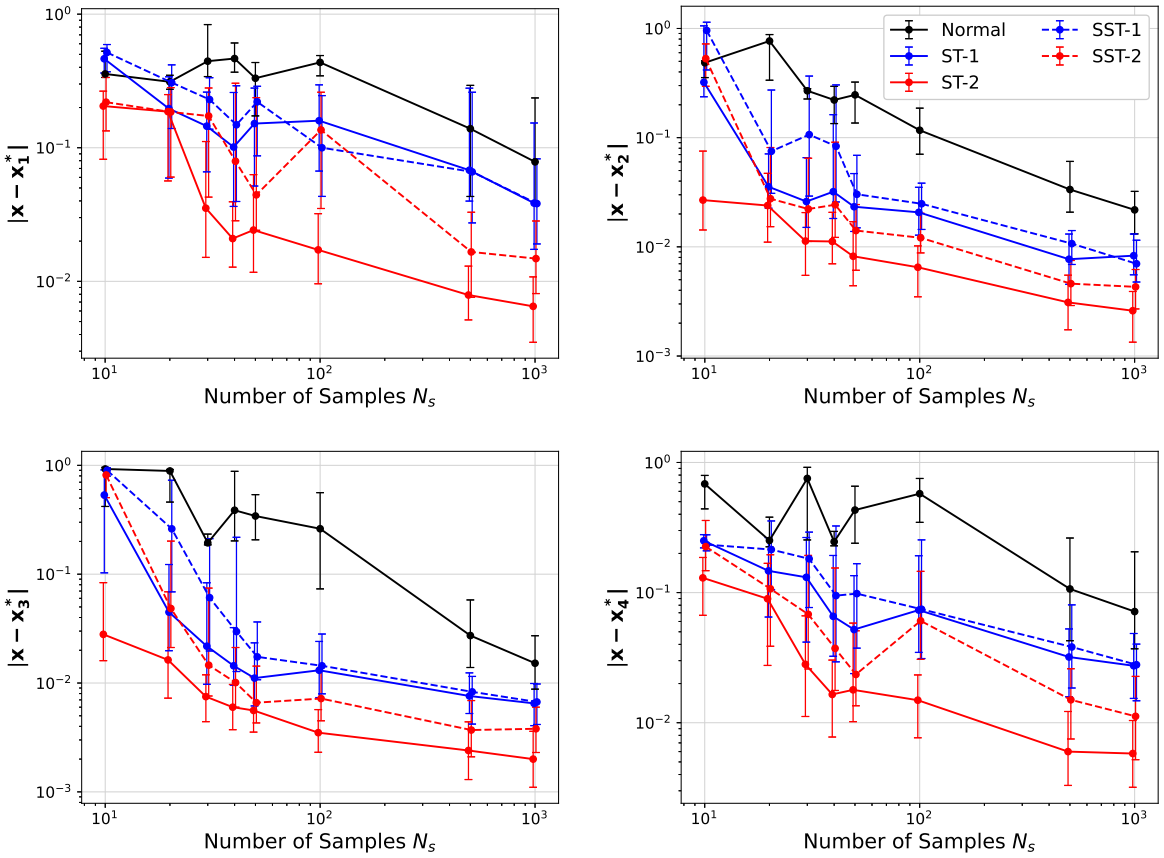

Figure 6: Distances between true optima and solutions found by optimization of NN surrogate models. Each point represents the median found in optimization of 100 trained models, and error bars show interquartile range.

Figure 6 clearly shows the advantages of Sobolev-trained vs normal NN models in terms of black-box optimization accuracy: for almost all cases, ST-1 and ST-2 models result in optima closer to the true function optima. The benefits of Sobolev training appear relatively independent of the number of samples used. For instance, although the ST-1 and ST-2 models seemed to converge in accuracy (Figure 4), ST-2 models consistently outperform ST-1 models in optimization accuracy. The stochastic implementation SST-1 closely approximates ST-1, but, interestingly, the SST-2 models seem to perform slightly worse than ST-2 models, despite having similar accuracies in terms of outputs and first derivatives (Figure 4). This difference

is potentially caused by increased random behavior: SST-2 involves two projection steps, and the second-order derivatives are only computed from projected first-order derivatives. Each epoch has uncertainty from both the batch selection and the derivative projection; as the same number of training epochs was used for all models, the accuracy of the SST-2 approximation is expected to be worse.

To examine the effect of output vs derivative accuracy, the ST-1 models trained with $N_s = 100$ and the ST-2 models trained with $N_s = 40$ provide an interesting comparison. While the ST-1 models predict the function output with slightly higher accuracies (median MSE of 1.31×10^{-4} and 1.65×10^{-4} respectively), their first derivatives are less accurate (median MSE of 3.78×10^{-4} and 1.99×10^{-4} respectively). The contours of the first ten models trained under both above conditions are shown in Figure 7, along with the local optima found by the four optimizations.

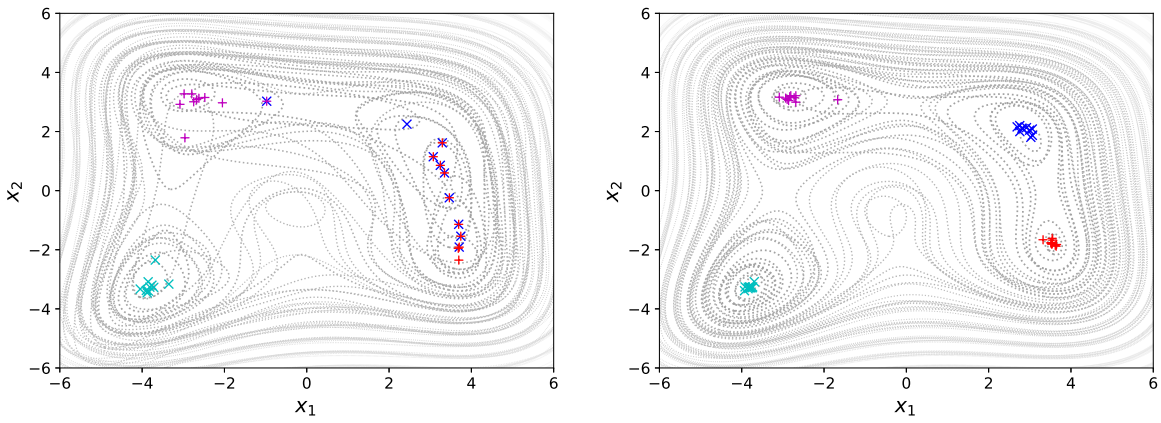

Figure 7: Contours and local optima of the first ten randomly initialized NN models trained using ST-1 with 100 samples (left) and using ST-2 with 40 samples (right).

Figure 7 shows that the contours of both the ST-1 and ST-2 functions generally resemble those of the true function (Figure 3), as expected from the relatively low output MSE of all models. Nevertheless, the ST-2 models predict the function derivatives with much higher accuracy, and the optima are consistently closer to the true function optima (also shown in Figure 6). On the other hand, the ST-1 models result in less consistent optimization results. This is most evident for $x_1 > 2$ in Figure 7 (left). In this region, the gradients of the true function are relatively small (i.e., the function is relatively flat). The less accurate ST-1 models often result in a single optimum in-between the two true optima, while the more accurate ST-2 models correctly identify two separate optima. These results show that derivative accuracy, in addition to the standard measures of function accuracy, are crucial for surrogate models used in

optimization applications, suggesting benchmarking accuracies of both outputs and derivatives.

5 Case Study 2: Grey-Box Optimization of Two-Phase Separator

The two-phase separator is a standard process unit operation commonly found in chemical process flowsheet models. This unit separates a multi-component inlet (feed) stream into liquid- and gas-phase outlet streams, whose compositions and flow rates are modeled using a "flash" (vapor–liquid phase equilibrium) calculation. The flash has two degrees of freedom; this study considers a pressure-temperature flash, where the pressure and temperature of the mixture are assumed to be known. The pressure-temperature flash model can be expressed as:

$$0 = Mz_i - Vy_i - Lx_i \qquad \forall i = 1, ..., N \tag{22}$$

$$0 = y_i - K_i x_i \qquad \forall i = 1, ..., N \tag{23}$$

$$K_i = \frac{\phi_i^L(P, T, \boldsymbol{z})}{\phi_i^V(P, T, \boldsymbol{z})} \qquad \forall i = 1, ..., N \tag{24}$$

$$\sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i = 1 \tag{25}$$

where $i = 1, ..., N$ denotes the components in order of volatility (i.e., from lightest to heaviest). The component-wise liquid and vapor fugacity coefficients, ϕ_i^L and ϕ_i^V, are typically computed from a nonlinear equation of state, as a function of pressure P, temperature T, and overall composition $\boldsymbol{z}$. A case study is selected based on the simulation results in Section 2.1 of Tsay and Baldea (2019b).

The feed stream is a mixture of five alkanes, denoted as z_1 (ethane), z_2 (propane), z_3 (n-butane), z_4 (n-pentane), and z_5 (n-hexane). The cubic Peng-Robinson equation of state is used to compute fugacity coefficients $\phi_i^L(P, T, \boldsymbol{z})$ and $\phi_i^V(P, T, \boldsymbol{z}), i = 1, ..., 5$. The inlet composition is given by $\boldsymbol{z} = [0.3, 0.2, 0.2, 0.25, 0.05]$. At the nominal conditions of $P = 0.5$ MPa and $T = 288$ K, the liquid and vapor stream compositions are, respectively, $\mathbf{x} = [0.12, 0.17, 0.26, 0.38, 0.08]$ and $\mathbf{y} = [0.59, 0.25, 0.11, 0.05, 0.00]$.

5.1 Optimization of Physical Model

The flash model (22)–(25) was implemented in an equation-oriented manner in Pyomo (Hart et al., 2017) and solved with Ipopt v3.13.3 (Wächter and Biegler, 2006). The objective function was selected to maximize the recovery of the heaviest three components in the liquid stream,

subject to an impurity constraint:

$$\begin{aligned} \max_{P,T} \quad & L \sum_{i=3}^{5} x_i \\ s.t. \quad & x_2 \leq 0.10 \\ & (22)\text{-}(25) \\ & T \in [200, 350], P \in [0.1, 1] \end{aligned} \tag{26}$$

The nominal pressure and temperature were used as the initial guess for optimization, with bounds of $P \in [0.1, 1]$ (MPa) and $T \in [200, 350]$ (K). Roots of the cubic equation corresponding to the liquid and vapor phases were identified using the formulation described by Kamath et al. (2010). This approach introduces two copies of the compressibility factor Z, which both must satisfy the cubic equation of state; the signs of the first and second derivatives are constrained to isolate the desired roots.

The resulting optimum is found at $\boldsymbol{x}^* = [0.1 \text{ MPa}, 256.44 \text{ K}]$. The same optimization problem is also solved with an additional constraint that T be greater or equal to 273.15 K, in order to provide an optimal solution away from the boundary of the variable domains and training data. The optimum for the second problem is found at $\boldsymbol{x}^{*\prime} = [0.172 \text{ MPa}, 273.15 \text{ K}]$. The contours of the objective function, inequality constraints, and optima for this flash optimization problem are shown in Figure 8.

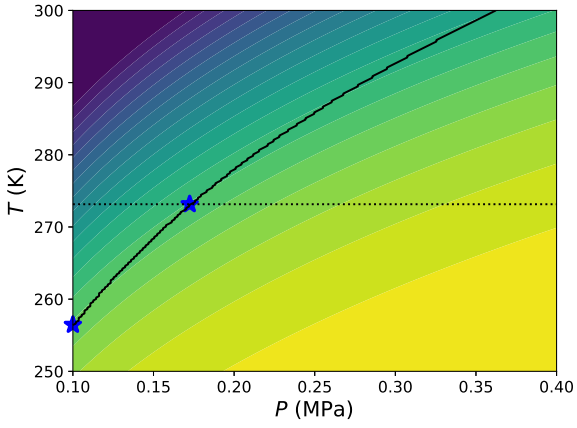

Figure 8: Contours, inequality constraints, and optima of flash optimization problems.

5.2 Hybrid Modeling Results

Based on the approach of Schweidtmann et al. (2019), a neural-network surrogate model replaces the cubic equation of state calculations in the physics-based model (22)–(25). Therefore,

(24) is replaced with $K_i = f_{\text{ANN}}(P, T, \boldsymbol{z})$, giving the following optimization problem:

$$
\begin{aligned}
\max_{P,T} \quad & L \sum_{i=1}^{3} x_i \\
s.t. \quad & x_2 \leq 0.10 \\
& 0 = Mz_i - Vy_i - Lx_i \qquad \forall i = 1, ..., 5 \\
& 0 = y_i - K_i x_i \qquad \forall i = 1, ..., 5 \\
& K_i = f_{\text{ANN}}(P, T, \boldsymbol{z}) \qquad \forall i = 1, ..., 5 \\
& \sum_{i=1}^{5} x_i = \sum_{i=1}^{5} y_i = 1 \\
& T \in [200, 350], P \in [0.1, 1]
\end{aligned}
\tag{27}
$$

As in the physics-based optimization problem, the inlet compositions $\boldsymbol{z}$ are assumed to be fixed, leaving P and T as input variables of the surrogate model. While a separate NN could be used to predict K_i for each component (Schweidtmann et al., 2019), this work employs a single, multiple-output, NN with five outputs, corresponding to $K_i, i = 1, ..., 5$ in order to investigate Sobolev training for a NN with multiple outputs (and associated derivatives). The derivatives of each output with respect to the NN inputs are computed independently via automatic differentiation in Pytorch, enabling simultaneous Sobolev training of all function outputs.

NN surrogate models were trained to approximate K_i computed by the Peng-Robinson equation of state. Training data were again created using standard Sobol sampling over the problem domain: $T \in [200, 350], P \in [0.1, 1]$. Derivatives at the sampled points were obtained using automatic differentiation, via the Autograd package (Maclaurin et al., 2015). For each training strategy, 100 NN models with two input nodes, three hidden layers with ten nodes each, and five output nodes are trained for 2000 epochs using the MSE loss function and `Adam` optimizer in Pytorch. Sigmoid activation functions are again applied in the hidden layers, and outputs are scaled to $[0, 1]$.

Figure 9 shows the median MSEs and interquartile ranges for each set of 100 models trained with varying N_s, estimated by evaluating both the scaled function $\bar{f}(\bar{\boldsymbol{x}})$ and the surrogate model $f_{\text{NN}}(\bar{\boldsymbol{x}})$ at the first 100,000 Sobol sampling locations. Lower values for N_s were used compared to the previous case study (note that each sample corresponds to one measurement for each of the five outputs), as this function seemed easier to approximate. Outputs and derivatives

were scaled using their true min/max (rather than the min/max in the sampled points) while computing MSEs for direct comparison across different numbers of samples. Only the mean MSE of all five outputs, and the mean MSE of the derivatives with respect to both input variables are shown here for clarity; element-wise MSEs with respect to the individual outputs and inputs are provided in the Supplementary Information. For each value of N_s, models are trained using standard training, first-order Sobolev training (ST-1), and second-order Sobolev training (ST-2).

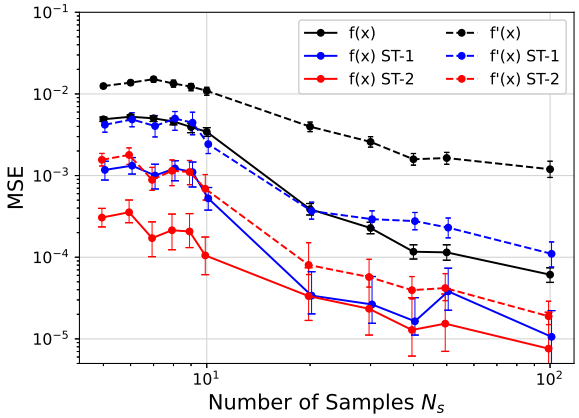

Figure 9: Median squared error (MSE) of NN surrogate models fit to K_i from Peng-Robinson equation with Sobolev training. Each point represents the median of the MSEs of 100 trained models, and error bars show interquartile range. MSEs of both function predictions (solid) and model derivatives (dashed) are shown.

We again see that NNs trained with ST-1 and ST-2 predict both the function output and its first derivatives more accurately compared to standard NNs, with ST-2 generally outperforming ST-1. Furthermore, the accuracies of ST-1 and ST-2 models seem to converge with increasing number of samples N_s, but this trend is only seen in terms of the predicted outputs $f(x)$: the derivatives of of ST-2 models are consistently the most accurate. For each of the five outputs, the first-order derivatives lie in $\mathbb{R}^2$ and the second order derivatives in $\mathbb{R}^{2\times 2}$. Therefore, NN models are once again trained to approximate $f(\boldsymbol{x})$ with stochastic first- and second-order Sobolev training, for 2000 epochs. Figure 10 shows the mean MSEs for each set of 100 models trained via first- (SST-1) and second-order stochastic Sobolev training (SST-2) for the same values of N_s, as computed using the above approach. The stochastic training procedures again approximate Sobolev training well, and the trends in Figures 9 and 10 are similar, with the accuracies of SST-1 and SST-2 converging with increasing number of samples.

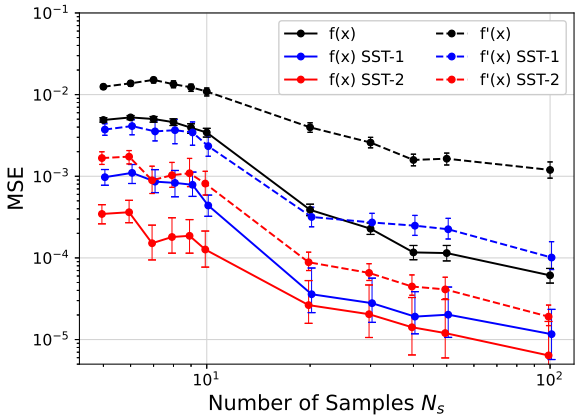

Figure 10: Median squared error (MSE) of NN surrogate models fit to K_i from Peng-Robinson equation with Sobolev training. Each point represents the median of the MSEs of 100 trained models, and error bars show interquartile range. MSEs of both function predictions (solid) and model derivatives (dashed) are shown.

5.3 Hybrid Model Optimization Results

The grey-box optimization problem in (27) was implemented in Pyomo (Hart et al., 2017) and solved using Ipopt version 3.13.3 (Wächter and Biegler, 2006) with default settings. Two optimizations are run for each NN model, with and without the additional constraint that $T \geq 273.15$ K, using the nominal temperature and pressure as the initial guess. The distances between the true optima of $\{\boldsymbol{x}^*, \boldsymbol{x}^{*\prime}\}$ and the solutions found from the respective optimization problems are shown in Figure 11 for normal, ST-1, ST-2, SST-1, and SST-2 trained NN models. The 1-norm of distances is shown here for clarity; element-wise optimization results are provided in the Supplementary Information.

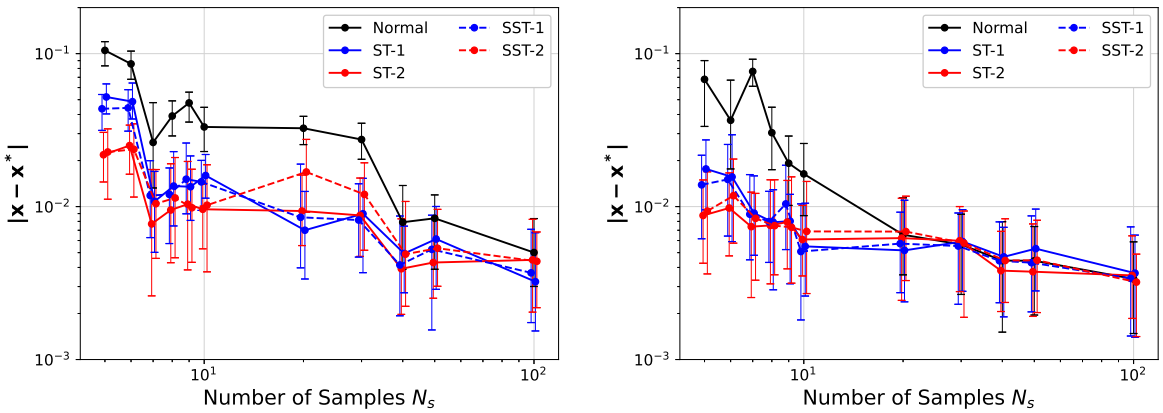

Figure 11: Distances between optima obtained using first-principles model of flash and solutions found by optimization of NN surrogate models. Each point represents the median found in optimization of 100 trained models, and error bars show interquartile range.

Figure 11 shows that the benefits of using Sobolev-trained NN models in grey-box opti-

mization are less straightforward. The ST-1 and ST-2 models result in optima closer to the true function optima compared to normal NNs when the number of samples N_s is very low. Especially when the optimum is near the edge of the sampled area (Figure 11, left), ST-2 and ST-1 models consistently outperform normal NN models in optimization accuracy. ST-2 models remain the best for very small N_s, but ST-2 and ST-1 models seem to quickly converge to the same performance with increasing number of samples. For the optimum in the middle of the sampled dataset (Figure 11, right), ST-2 and ST-1 models outperform normal NN models for small N_s, but all models converge to the same accuracy at approximately $N_s = 20$. A key difference here is that, as noted in Section 3.1, the function accuracy plays a central role in grey-box optimization. The strength of standard-trained NN models is in interpolating functions when a large number of samples is available, explaining their relatively good performance for finding $\boldsymbol{x}^{*\prime}$ when these conditions are met.

While SST-2 models are less consistent compared to ST-2 models (see Figure 11, left), the stochastic implementations SST-1 and SST-2 generally have more similar performance to ST-1 and ST-2, respectively, than in the previous case study. Intuitively, the stochastic methods may perform better, as the improved model MSEs at low numbers of samples suggest that this function $K_i(P, T, \boldsymbol{z})$ is easier to fit than the Himmelblau function. Therefore, since the models are trained for the same 2000 epochs against many fewer samples, the prediction MSE likely converges quickly, allowing relatively more epochs spent training the (stochastic) derivative MSEs.

We further examine the models trained on 20 samples, $N_s = 20$, where ST-1, ST-2, and normal models seem to converge to the same optimization accuracy in terms of identifying $\boldsymbol{x}^{*\prime}$. At this point, the ST-1 and ST-2 models predict the function output with similar accuracy (medians MSE of 3.39×10^{-5} and 3.34×10^{-5} respectively), but the ST-1 models predict function derivatives less accurately (median MSEs of 3.68×10^{-4} and 8.04×10^{-5} respectively). The normal NN models predict both the function outputs (median MSE 3.88×10^{-4}) and derivatives (median MSE 3.97×10^{-3} less accurately than ST-1 and ST-2 models. Nevertheless, Figure 9 shows that the normal models are able to identify the internal optimum $\boldsymbol{x}^{*\prime}$ approximately as well as ST-1 and ST-2 models. The contours of the first ten models trained with each method are shown in Figure 12, along with the local optima found by the two optimizations.

Figure 12 shows that inaccuracy in finding the optima is largely due to inaccuracies in representing the inequality constraint. As we observed in the previous case study, the ST-2 models

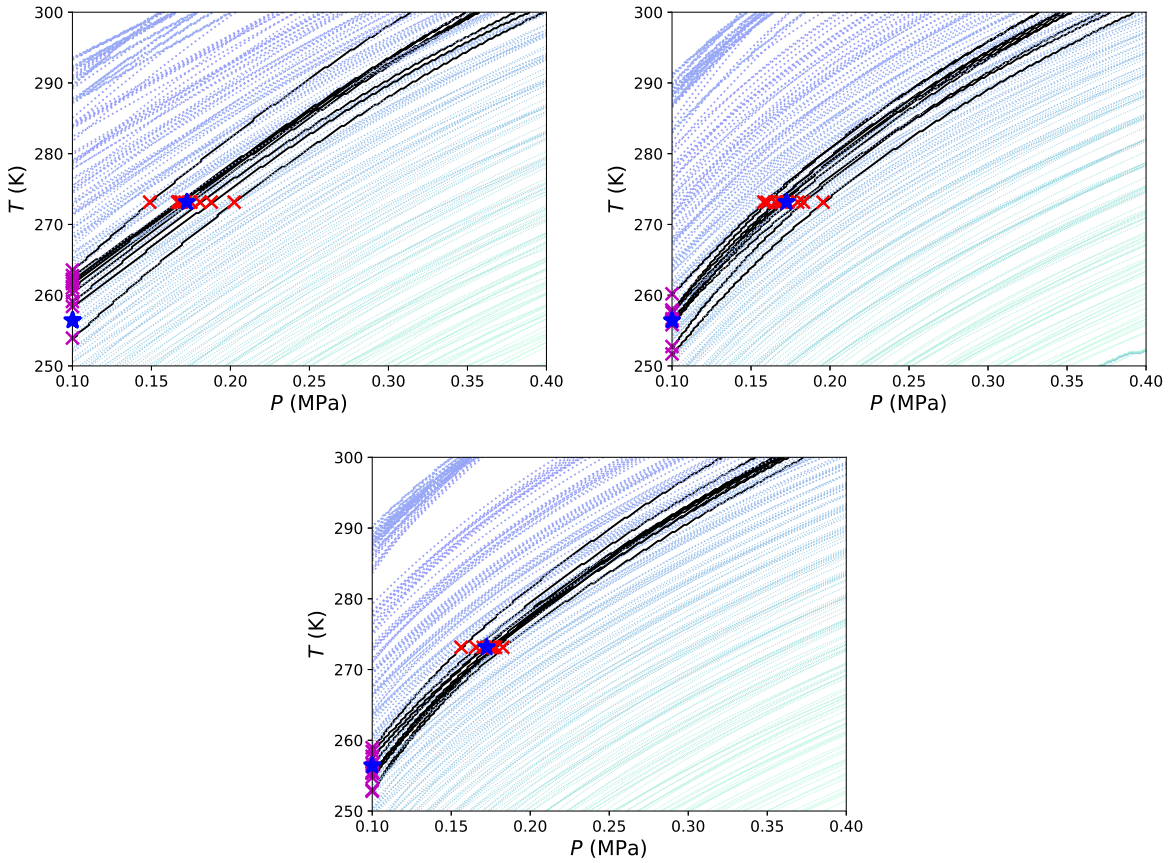

Figure 12: Contours (blue) and local optima of the first ten randomly initialized NN models trained on 20 samples using normal training (top-left), ST-1 (top-right) and ST-2 (bottom).

predict function derivatives more accurately than ST-1 models, and the ST-2 models appear to have more accurate and consistent objective function contours (Figure 12). Nevertheless, the objective function contours for the normal and ST-1 models are still accurate enough to identify that the optima occur at the intersection of a temperature/pressure bound and the impurity constraint. These results, combined with Figure 11 suggest that, at some number of samples, the objective function becomes accurate "enough," that the direction is in general able to locate the proper active inequality constraint(s). Note that the inequality (impurity) constraint exemplifies the incorporation of an NN into a grey-box model (the constraint acts on x_1 rather than the NN output, K_1). The approximate optima found for $\boldsymbol{x}^*$ for the most part are where the impurity constraint intersects with $P = 0.1$ MPa, while those found for $\boldsymbol{x}^{*\prime}$ mostly lie where the same constraint intersects with $T = 273$ K.

In summary, the above results suggest that derivative accuracy are important for surrogate models used in grey-box optimization (the ST models improve accuracy of identifying the first optimum $\boldsymbol{x}^*$), but improvements in accuracy do not necessarily guarantee improve-

ments in optimization performance. For instance, when N_s is large, standard NNs may perform approximately as well (Figure 11, right). Sobolev-training can improve the accuracy NN surrogate models, enabling them to perform better in optimization in circumstances where they traditionally struggle: making predictions with a low volume of training data and/or near the boundary of the training dataset. Such cases are often important in practice, when (i) samples are expensive and/or the dimensionality of the sampled space is large, and (ii) the optima of the true function lie close to the boundary of the feasible space, as is often the case with real-world processes (Tsay et al., 2018). However, with a large enough number of samples, standard NN models can have derivatives accurate "enough" for grey-box optimization, especially when surrogate models are primarily representing (active) constraints.

6 Case Study 3: Grey-Box Optimization of Two Flashes in Series

Process flowsheet models typically comprise multiple interconnected unit operations, which can be individually approximated using surrogate models. For instance, consider the simple process shown in Figure 13. Here, two-phase separators are employed in series in order to separate more components, and each can be modeled using a separate flash calculation. Assuming the same feed stream as above, the first flash is unchanged, with two degrees of freedom (pressure and temperature); however, the inlet stream to the second flash is variable, as both its composition and flow rate depend on the conditions of the first flash.

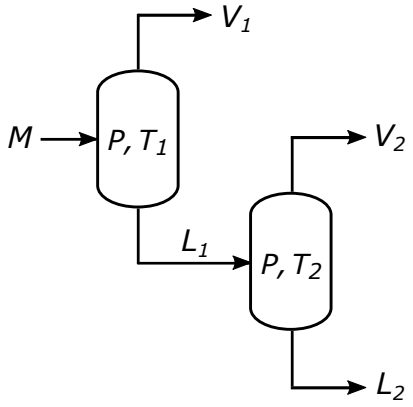

Figure 13: Process diagram of two-phase separators in series.

The model of two pressure-temperature flash in series can be expressed as:

$$0 = Mz_{1,i} - V_1 y_{1,i} - L_1 x_{1,i} \qquad \forall i = 1, ..., N \tag{28}$$

$$0 = L_1 x_{1,i} - V_2 y_{2,i} - L_2 x_{2,i} \qquad \forall i = 1, ..., N \tag{29}$$

$$0 = y_{1,i} - K_{1,i} x_{1,i} \qquad \forall i = 1, ..., N \tag{30}$$

$$0 = y_{2,i} - K_{2,i} x_{2.i} \qquad \forall i = 1, ..., N \tag{31}$$

$$K_{1,i} = \frac{\phi_i^L(P, T_1, \boldsymbol{z})}{\phi_i^V(P, T_1, \boldsymbol{z})} \qquad \forall i = 1, ..., N \tag{32}$$

$$K_{2,i} = \frac{\phi_i^L(P, T_2, \boldsymbol{x}_1)}{\phi_i^V(P, T_2, \boldsymbol{x}_1)} \qquad \forall i = 1, ..., N \tag{33}$$

$$\sum_{i=1}^{N} x_{1,i} = \sum_{i=1}^{N} x_{2,i} = \sum_{i=1}^{N} y_{1,i} = \sum_{i=1}^{N} y_{2,i} = 1 \tag{34}$$

where $i = 1, ..., N$ again indexes the components from lightest to heaviest. The feed composition is again given by $\boldsymbol{z} = [0.3, 0.2, 0.2, 0.25, 0.05]$, and the Peng-Robinson equation of state is used to compute the fugacity coefficients for both flashes. The inlet composition to the second flash is given by $\boldsymbol{x}_1$, the composition of the liquid outlet of the first flash.

6.1 Optimization of Physical Model

The flash model (22)–(25) was implemented in Pyomo (Hart et al., 2017) for both flash tanks together, again in an equation-oriented manner and solved with Ipopt v3.13.3 (Wächter and Biegler, 2006). The objective function was selected to maximize the recovery of the heaviest two components in the liquid outlet of the second flash, plus the recovery of the lightest component in the vapor outlet of the first flash. Both flashes are subject to an impurity constraint:

$$\begin{aligned} \max_{P, T_1, T_2} \quad & V_1 y_{1,1} + L_2 \sum_{i=4}^{5} x_i \\ s.t. \quad & y_2 \leq 0.22 \\ & x_3 \leq 0.20 \\ & (28)\text{-}(34) \\ & T_1 \in [200, 350], T_2 \in [250, 400], P \in [0.2, 1] \end{aligned} \tag{35}$$

The same strategies as above were used to identify roots of the cubic equations corresponding to the liquid and vapor phases for both flashes. Initial guesses for pressure and temperatures were determined by lengthy trial-and-error, as it was difficult to find values that would converge

to a feasible solution; however, all optimizations that converged resulted in the same optimal point. The resulting optimum is found at $\boldsymbol{x}^* = [0.2\text{ MPa}, 249.71\text{ K}, 303.24\text{ K}]$. At this optimal point, both impurity constraints are exactly met, and all flow rates are nonzero, with $V_1 = 0.298$, $L_1 = 0.702$, $V_2 = 0.487$, and $L_2 = 0.215$.

6.2 Hybrid Modeling Results

A grey-box optimization problem is formulated by replacing the cubic equation of state calculations in both flashes using neural-network surrogate models. Replacing the split fractions in (28)–(34) with $K_{1,i} = f_{\text{ANN},1}(P, T_1, \boldsymbol{z})$ and $K_{2,i} = f_{\text{ANN},2}(P, T_2, \boldsymbol{x}_1)$ gives the following grey-box optimization problem:

$$
\begin{aligned}
\max_{P,T_1,T_2} \quad & V_1 y_{1,1} + L_2 \sum_{i=4}^{5} x_i \\
s.t. \quad & y_2 \leq 0.22 \\
& x_3 \leq 0.20 \\
& 0 = M z_{1,i} - V_1 y_{1,i} - L_1 x_{1,i} && \forall i = 1, ..., N \\
& 0 = L x_{1,i} - V_2 y_{2,i} - L_2 x_{2,i} && \forall i = 1, ..., N \\
& 0 = y_{1,i} - f_{\text{ANN},1,\text{i}}(P, T_1, \boldsymbol{z}) x_{1,i} && \forall i = 1, ..., N \\
& 0 = y_{2,i} - f_{\text{ANN},2,\text{i}}(P, T_2, \boldsymbol{x}_1) x_{2.i} && \forall i = 1, ..., N \\
& T_1 \in [200, 350], T_2 \in [250, 400], P \in [0.2, 1]
\end{aligned}
\tag{36}
$$

As mentioned above, the first flash can be modeled identically as in the previous case study, and the NN surrogates from Section 5.2 are used as $f_{\text{ANN},1}(P, T_1, \boldsymbol{z})$, where $\boldsymbol{z}$ is fixed. NN surrogate models were similarly trained to approximate K_i in the second flash; however the input of the $f_{\text{ANN},2}(P, T_2, \boldsymbol{x}_1)$ surrogates must account for variations in $\boldsymbol{x}_1$. Therefore, training data were created by sampling the Peng-Robinson equation of state using Sobol sampling over the domains: $T_2 \in [250, 400]$, $P \in [0.1, 1]$, $x_{1,1} \in [0.05, 0.15]$, $x_{1,2} \in [0.10, 0.20]$, $x_{1,3} \in [0.20, 0.30]$, $x_{1,5} \in [0.05, 0.15]$. Note that the domain $P \in [0.2, 1]$ during optimization results in an optimal point inside the training dataset, to further investigate the optimization properties of Sobolev-trained models under these conditions. For sampling purposes, $x_{1,4}$ was computed using $\sum \boldsymbol{x}_1 = 1$; therefore $x_{1,4}$ is not included as an NN input, as it is linearly dependent with the other inputs. Derivatives at sampled points were again obtained using automatic differentiation via Autograd (Maclaurin et al., 2015). For each training strategy, 100 NN models with six input nodes, three

hidden layers with 20 nodes each, and five output nodes are again trained for 2000 epochs using the MSE loss function and `Adam` optimizer in Pytorch. Sigmoid activation functions are again applied in the hidden layers, and outputs are scaled to $[0, 1]$.

Figure 14 shows the median MSEs (averaged across the five outputs) and interquartile ranges for each set of 100 models trained with varying N_s, estimated by evaluating both the scaled function $\bar{f}(\bar{\boldsymbol{x}})$ and the surrogate model $f_{\text{NN}}(\bar{\boldsymbol{x}})$ at the first 100,000 Sobol sampling locations. The numbers of samples N_s were selected one order of magnitude larger than in the previous case study due to the increase in input dimensionality. Outputs and derivatives were scaled using their true min/max (rather than the min/max in the sampled points) while computing MSEs for direct comparison across different numbers of samples. For each value of N_s, models are trained using standard training, first-order Sobolev training (ST-1), and second-order Sobolev training (ST-2).

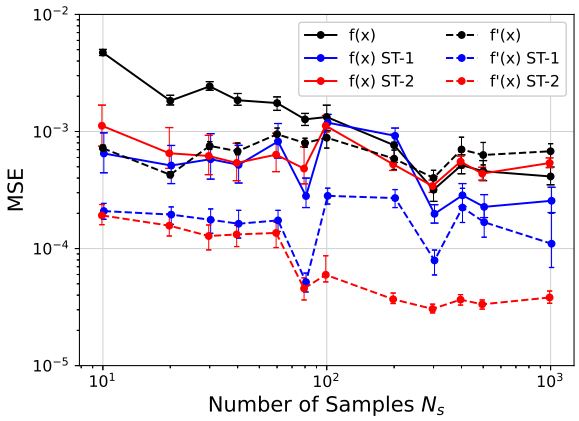

Figure 14: Median squared error (MSE) of NN surrogate models fit to K_i from Peng-Robinson equation including variable inlet composition with Sobolev training. Each point represents the median of the MSEs of 100 trained models, and error bars show interquartile range. MSEs of both function predictions (solid) and model derivatives (dashed) are shown.

For this more complex function, NNs trained with ST-1 and ST-2 predict the function output more accurately than standard NNs do for low N_s, but the accuracies appear similar for N_s greater than approximately 100 samples. Nevertheless, the first derivatives are predicted more accurately by the Sobolev-trained models compared to standard NNs, with ST-2 consistently outperforming ST-1. The accuracies of ST-1 and ST-2 models are comparable, with each being slightly more accurate than the other for NNs trained on certain numbers of samples N_s. This could partially be attributed to overfitting the training data, as the training data are more sparse in the higher-dimensional input space; better training/termination procedures could be investigated in the future to improve model consistency. Furthermore, it could be difficult to

train models on the complicated loss functions: the first-order derivatives for each output lie in $\mathbb{R}^6$ and the second order derivatives in $\mathbb{R}^{6\times6}$. Therefore, NN models are once again trained with stochastic first- and second-order Sobolev training. Figure 15 shows the mean MSEs for each set of 100 models trained via first- (SST-1) and second-order stochastic Sobolev training (SST-2) for the same values of N_s.

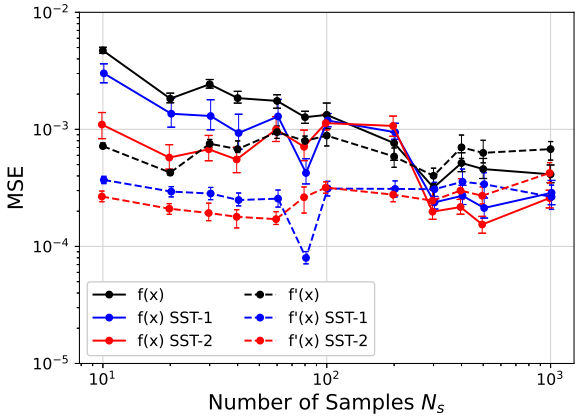

Figure 15: Median squared error (MSE) of NN surrogate models fit to K_i from Peng-Robinson equation including variable input composition with Sobolev training. Each point represents the median of the MSEs of 100 trained models, and error bars show interquartile range. MSEs of both function predictions (solid) and model derivatives (dashed) are shown.

The stochastic training procedures result in more consistent trends. For low numbers of samples, SST-2 models are more accurate than SST-1 models, which are in turn more accurate than standard NNs, both in terms of function outputs and derivatives. The models again converge to similar accuracies for N_s greater than approximately 100 samples. Compared to the previous, lower-dimensional case studies, stochastic Sobolev training seems to have a larger impact here. Intuitively, the stochastic Sobolev training loss function is more different from the non-stochastic loss function when the dimensionality of the input space is higher. In this case, the 6×6 second-order derivatives for each output are replaced with a single projected term in the loss function, greatly reducing the complexity of the loss function.

6.3 Hybrid Model Optimization Results

The combined grey-box optimization problem (36) was again solved using Ipopt version 3.13.3 (Wächter and Biegler, 2006) with default settings. While the physics-based flash model is extremely fast when it converges, a significant downside of the equation-oriented approach is that the optimization model is difficult to converge (or even find a feasible point) for many choices of initial guess values. In contrast, the hybrid flash is significantly simpler and converges

even with no initial guess provided to Ipopt. The 1-norm distances between the optimal inputs to both NNs given by the physics-based optimization problem and the grey-box formulations are shown in Figure 16 for normal, ST-1, ST-2, SST-1, and SST-2 trained NN models. The number of samples N_s in Figure 16 corresponds to the number of samples used to train the surrogate model used in the second flash, $f_{\text{ANN},2}$. To account for the lower-dimensional input space, the surrogate model used in the first flash $f_{\text{ANN},1}$ was chosen as that trained on $\texttt{max}(5, N_s/10)$ samples.

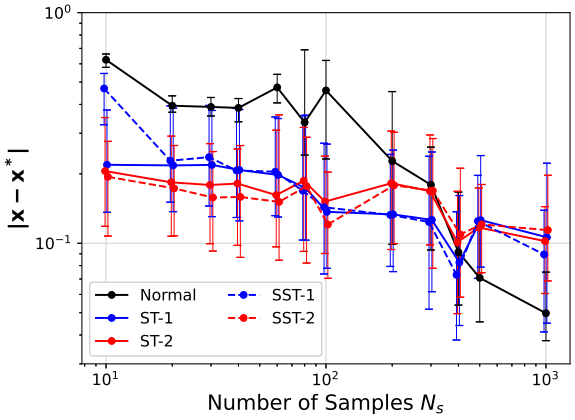

Figure 16: Distances between optima obtained using first-principles model of two flashes and solutions found by optimization of NN surrogate models. Each point represents the median found in optimization of 100 trained models, and error bars show interquartile range.

Figure 16 shows that the Sobolev-trained NN models again result in optima closer to the true function optima compared to standard NNs when the number of samples N_s is low. Similar to the previous case studies, ST-2 models remain the best for very small N_s, but ST-2 and ST-1 models converge to similar performance as N_s increases. Although the error bars are relatively large in this case study, the interquartile ranges found using ST-2 and SST-2 models fall consistently lower than those for ST-1 and SST-1 models for $N_s < 100$. While the Sobolev-trained models consistently have more accurate derivatives than the standard-trained NNs, function output accuracies of the models are similar for large N_s. As hypothesized in the previous case study, the surrogate models here represent active constraints of the model, and therefore the accuracy of surrogate model derivatives seem to play a lesser role in optimization accuracy. All models perform relatively well in terms of finding $\boldsymbol{x}^*$ when the number of training samples used is large (and functions outputs are predicted accurately).

In both grey-box case studies, the stochastic implementations SST-1 and SST-2 generally have similar performance to ST-1 and ST-2, respectively. This again suggests that the equation-

of-state function is easier to fit than the Himmelblau function, allowing relatively many epochs for improving the (stochastic) derivative MSEs after training quickly converges to predict function outputs accurately. In summary, this case study further supports that improvements in the accuracies of surrogate model gradients do not always guarantee improvements in grey-box optimization performance: Figure 16 shows that standard NNs can perform about as well when the number of samples is large. Nevertheless, Sobolev-training again enables the NN surrogate models to perform better in optimization with a low volume of training data. Comparing Figures 11 and 16 shows that "low volume" here is dependent on the dimensionality of the model input space. Although this higher-dimensional problem is more difficult to visualize, it can be hypothesized based on the previous case studies that, for large training data sets, the models all have have derivatives accurate "enough" to identify which constraints are active, and optimization mostly relies on accurate representation of these constraints by the surrogate models.

7 Conclusion

Difficult (e.g., nonlinear, large-scale) mathematical models of process systems are often replaced with data-driven surrogate models during computational process optimization. Such surrogate models are fitted to simulations obtained from a more complex model and are often embedded in black-box and/or grey-box optimization problems. When function derivatives are also available form the complex model, surrogate models can be directly trained to also match such derivatives to arbitrary degree, a technique known as Sobolev training. Sobolev training is particularly relevant for neural network (NN) models, as most software implementations of NNs include capabilities for obtaining derivatives via automatic differentiation.

This work examines the relevance of Sobolev-trained NN surrogate models as pertaining specifically to gradient-based optimization. A strategy was also proposed to systematically scale components of Sobolev-training loss functions such that they are not dominated by some particular term(s). The computational case studies showed the benefits of using Sobolev-trained NNs in both black-box and grey-box optimization. In particular, we found that Sobolev training improves the optimization performance under almost all circumstances for black-box optimization, including when the number of samples is large enough that accuracies of model predictions are similar. In grey-box optimization, Sobolev training can still improve the optimization performance of NN surrogate models, especially when the number of samples is low. The results

furthermore that derivative accuracy, in addition to the standard measures of function accuracy, are crucial metrics for evaluating the performance of surrogate models in optimization applications.

8 Acknowledgments

The research was funded by an Engineering & Physical Sciences Research Council (EPSRC) Research Fellowship (grant EP/T001577/1), with additional support from an Imperial College Research Fellowship.

References

A. Agarwal and L. T. Biegler. A trust-region framework for constrained optimization using reduced order modeling. *Optimization and Engineering*, 14(1):3–35, 2013.

H. R. S. Anna, A. G. Barreto Jr, F. W. Tavares, and M. B. de Souza Jr. Machine learning model and optimization of a psa unit for methane-nitrogen separation. *Computers & Chemical Engineering*, 104:377–391, 2017.

A. Bhosekar and M. Ierapetritou. Advances in surrogate based modeling, feasibility analysis, and optimization: A review. *Computers & Chemical Engineering*, 108:250–267, 2018.

L. T. Biegler. New nonlinear programming paradigms for the future of process optimization. *AIChE Journal*, 63(4):1178–1193, 2017.

L. T. Biegler, I. E. Grossmann, and A. W. Westerberg. A note on approximation techniques used for process optimization. *Computers & Chemical Engineering*, 9(2):201–206, 1985.

M. A. Bouhlel, S. He, and J. R. Martins. Scalable gradient–enhanced artificial neural networks for airfoil shape design in the subsonic and transonic regimes. *Structural and Multidisciplinary Optimization*, pages 1–14, 2020.

F. Boukouvala and C. A. Floudas. ARGONAUT: AlgoRithms for Global Optimization of coNstrAined grey-box compUTational problems. *Optimization Letters*, 11(5):895–913, 2017.

F. Boukouvala, M. F. Hasan, and C. A. Floudas. Global optimization of general constrained grey-box models: new method and its application to constrained PDEs for pressure swing adsorption. *Journal of Global Optimization*, 67(1-2):3–42, 2017.

J. A. Caballero and I. E. Grossmann. An algorithm for the use of surrogate models in modular flowsheet optimization. *AIChE Journal*, 54(10):2633–2650, 2008.

Y. Chen, Y. Shi, and B. Zhang. Optimal control via neural networks: A convex approach. In *International Conference on Learning Representations*, 2019.

J. Cocola and P. Hand. Global convergence of Sobolev training for overparametrized neural networks. *arXiv preprint arXiv:2006.07928*, 2020.

W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu. Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, pages 4278–4287, 2017.

E. Davis and M. Ierapetritou. A kriging based method for the solution of mixed-integer nonlinear programs containing black-box functions. *Journal of Global Optimization*, 43(2-3):191–205, 2009.

L. S. Dias and M. G. Ierapetritou. Data-driven feasibility analysis for the integration of planning and scheduling problems. *Optimization and Engineering*, 20(4):1029–1066, 2019.

N. Dige and U. Diwekar. Efficient sampling algorithm for large-scale optimization under uncertainty problems. *Computers & Chemical Engineering*, 115:431–454, 2018.

J. Eason and S. Cremaschi. Adaptive sequential sampling for surrogate model generation with artificial neural networks. *Computers & Chemical Engineering*, 68:220–232, 2014.

K. Giannakoglou, D. Papadimitriou, and I. Kampolis. Aerodynamic shape design using evolutionary algorithms and new gradient-assisted metamodels. *Computer Methods in Applied Mechanics and Engineering*, 195(44-47):6312–6329, 2006.

B. Grimstad and H. Andersson. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

I. Gühring, G. Kutyniok, and P. Petersen. Error bounds for approximations with deep ReLU neural networks in $W^{s,p}$ norms. *Analysis and Applications*, 18(05):803–859, 2020.

W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, and J. D. Siirola. *Pyomo-Optimization Modeling in Python*, volume 67. Springer, 2017.

C. A. Henao and C. T. Maravelias. Surrogate-based superstructure optimization framework. *AIChE Journal*, 57(5):1216–1232, 2011.

D. M. Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, 2018.

K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4 (2):251–257, 1991.

Y. Jin, J. Li, W. Du, and F. Qian. Multi-objective optimization of pseudo-dynamic operation of naphtha pyrolysis by a surrogate model. *Chemical Engineering & Technology*, 38(5):900–906, 2015.

R. S. Kamath, L. T. Biegler, and I. E. Grossmann. An equation-oriented approach for handling thermodynamics based on cubic equation of state in process optimization. *Computers & Chemical Engineering*, 34(12):2085–2096, 2010.

I. C. Kampolis, E. I. Karangelos, and K. C. Giannakoglou. Gradient-assisted radial basis function networks: theory and applications. *Applied Mathematical Modelling*, 28(2):197–209, 2004.

S. H. Kim and F. Boukouvala. Surrogate-based optimization for mixed-integer nonlinear problems. *Computers & Chemical Engineering*, 140:106847, 2020.

L. Laurent, R. Le Riche, B. Soulier, and P.-A. Boucard. An overview of gradient-enhanced metamodels with applications. *Archives of Computational Methods in Engineering*, 26(1): 61–106, 2019.

S. J. Leary, A. Bhaskar, and A. J. Keane. Global approximation and optimization using adjoint computational fluid dynamics codes. *AIAA Journal*, 42(3):631–641, 2004.

W. Liu and S. Batill. Gradient-enhanced neural network response surface approximations. In *8th Symposium on Multidisciplinary Analysis and Optimization*, page 4923, 2000.

D. Maclaurin, D. Duvenaud, and R. P. Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, volume 238, page 5, 2015.

K. McBride and K. Sundmacher. Overview of surrogate modeling in chemical process engineering. *Chemie Ingenieur Technik*, 91(3):228–239, 2019.

M. Mistry, D. Letsios, G. Krennrich, R. M. Lee, and R. Misener. Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *INFORMS Journal on Computing*, 2020. DOI 10.1287/ijoc.2020.0993.

A. Mitsos, N. Asprion, C. A. Floudas, M. Bortz, M. Baldea, D. Bonvin, A. Caspari, and P. Schäfer. Challenges in process optimization for new feedstocks and energy sources. *Computers & Chemical Engineering*, 113:209–221, 2018.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8026–8037, 2019.

C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

A. Rogers and M. Ierapetritou. Feasibility and flexibility analysis of black-box processes part 2: Surrogate-based flexibility analysis. *Chemical Engineering Science*, 137:1005–1013, 2015.

P. Schäfer, A. Caspari, K. Kleinhans, A. Mhamdi, and A. Mitsos. Reduced dynamic modeling approach for rectification columns based on compartmentalization and artificial neural networks. *AIChE Journal*, 65(5):e16568, 2019.

A. M. Schweidtmann and A. Mitsos. Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948, 2019.

A. M. Schweidtmann, D. Bongartz, W. R. Huster, and A. Mitsos. Deterministic global process optimization: Flash calculations via artificial neural networks. In *Computer Aided Chemical Engineering*, volume 46, pages 937–942. Elsevier, 2019.

R. Sellar and S. Batill. Concurrent subspace optimization using gradient-enhanced neural network approximations. In *6th Symposium on Multidisciplinary Analysis and Optimization*, page 4019, 1996.

S. Srinivas and F. Fleuret. Knowledge transfer with Jacobian matching. *arXiv preprint arXiv:1803.00443*, 2018.

A. Thebelt, J. Kronqvist, M. Mistry, R. M. Lee, N. Sudermann-Merx, and R. Misener. ENTMOOT: a framework for optimization over ensemble tree models. *arXiv preprint arXiv:2003.04774*, 2020.

C. Tsay and M. Baldea. 110th anniversary: using data to bridge the time and length scales of process systems. *Industrial & Engineering Chemistry Research*, 58(36):16696–16708, 2019a.

C. Tsay and M. Baldea. Fast and efficient chemical process flowsheet simulation by pseudo-transient continuation on inertial manifolds. *Computer Methods in Applied Mechanics and Engineering*, 348:935–953, 2019b.

C. Tsay and M. Baldea. Integrating production scheduling and process control using latent variable dynamic models. *Control Engineering Practice*, 94:104201, 2020.

C. Tsay, R. C. Pattison, M. R. Piana, and M. Baldea. A survey of optimal process design capabilities and practices in the chemical and petrochemical industries. *Computers & Chemical Engineering*, 112:180–189, 2018.

A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

S. Yang and B. W. Bequette. Optimization-based control using input convex neural networks. *Computers & Chemical Engineering*, 144:107143, 2021.