Sobolev Training with Approximated Derivatives for Black-Box Function Regression with Neural Networks

Matthias Kissel[1](✉) and Klaus Diepold[1]

[1]Chair for Data Processing, Technical University of Munich
Arcisstr. 21, 80333 Munich, Germany
`matthias.kissel@tum.de`
https://www.ldv.ei.tum.de/

Abstract. With Sobolev Training, neural networks are trained to fit target output values as well as target derivatives with respect to the inputs. This leads to better generalization and fewer required training examples for certain problems. In this paper, we present a training pipeline that enables Sobolev Training for regression problems where target derivatives are not directly available. Thus, we propose to use a least-squares estimate of the target derivatives based on function values of neighboring training samples. We show for a variety of black-box function regression tasks that our training pipeline achieves smaller test errors compared to the traditional training method. Since our method has no additional requirements on the data collection process, it has great potential to improve the results for various regression tasks.

Keywords: Sobolev Training · Neural Networks · Machine Learning.

1 Introduction

Neural networks are used as function approximators for a variety of regression tasks like forecasting problems, policy regression or black-box function approximation (i.e. functions for which the analytical form is unknown). The standard approach of training neural networks is backpropagation, which updates the trainable parameters in the neural network by propagating the output error through the network. A strategy to increase the efficiency of the backpropagation algorithm proposed by several authors [2, 16, 1, 17, 10] is to incorporate information on derivatives of the target function into the training algorithm. For example, terms can be added to the error definition which penalize deviations of the network's partial derivatives to the partial derivatives of the target function. This is based on the idea that the neural network should match the outputs of the target function *and* its partial derivatives at the training points in order to match the desired function accurately. In the remainder of this paper we will use the terms introduced by Czarnecki et al. [1] and Masouka et al. [10] and refer to the standard backpropagation approach for neural network training as

Value Training, and to the modified backpropagation incorporating information on derivatives as *Sobolev Training*.

It has been shown that Sobolev Training outperforms Value Training in terms of validation error and convergence speed for several applications. For example, Witkosie et al. [26] showed that using Sobolev Training to model potential energy surfaces can greatly reduce the density of data needed while still resulting in a better fit. Mitchell et al. [11] showed that Sobolev Training can lead to better generalization even by using fewer data points for training in the robotics and reinforcement learning domain. Besides better training performance, Sobolev Training can also decrease the sensitivity to noise in the training data as shown by Lee and Oh [9]. These publications are consistent with each other in the sense that they all claim that Sobolev Training is advantageous over Value Training in their chosen application. Indeed, Masouka et al. [10] argued that adding derivative information to the training increases the probability for better generalization.

In real world applications and in many toy-examples, however, information on the derivatives of the target function are typically not available. Several publications overcome this problem by rewriting a-priori or expert knowledge as derivatives which can be incorporated into the training process. For example, Lampinen et al. [8] proposed to use numerically inaccurate expert knowledge to design target derivatives which can be used during the training of a neural network. Simard et al. [18] utilized the fact that the derivatives have to be zero if the input data is transformed in specific ways (e.g. for translations or rotations). They claim that by explicitly adding these assumptions into the training process the learning speed is improved. Rifai et al. [17] used regularization terms incorporating the derivative to train an autoencoder for unsupervised feature extraction. By that, the autoencoder is more robust to corruptions in the input data and more relevant information is extracted. Similarly, Varga et al. [25] showed that gradient regularization can increase classification accuracy especially for small training datasets. Another approach is explanation-based learning [11, 10], where knowledge about derivatives is extracted from previously learned tasks and seen examples.

In contrast to the assumptions in the existing approaches, we assume that for our applications no information on derivatives is accessible and no a-priori knowledge or expert knowledge is available. Moreover it is assumed that the analytical structure of the target function is unknown, i.e. we investigate the case of black-box function regression. For this application case, we propose a training pipeline which approximates the partial derivatives of the target function. Derivatives are approximated by a least squares estimate based on the function values of neighboring training samples.

Our goal is to give empirical evidence for the superiority of our training method. Therefore, we evaluate our algorithm by performing experiments with various black-box function regression tasks and different training dataset sizes. Besides comparing our training method with the standard Value Training algorithm, we compare our algorithm with the approach of approximating the target

Algorithm 1 Sobolev Training with Least-Squares approximated Derivatives

```
In-/Output: Input Data $X$, Output Data $Y$
 1: Approximate Target Derivatives
 2: Transform Data
 3: Initialize Neural Network and Optimizer
 4: Initialize Sobolev Weight Factor $\rho$
 5: while Stopping Criterion not met do
 6:     Shuffle the Dataset and create Batches
 7:     for batch in Batches do
 8:         Compute Gradients of the Error for batch w.r.t. the Weights of the Neural
            Network
 9:         Update the Weights of the Neural Network
10:     end for
11:     Update $\rho$
12: end while
```

derivatives using a straightforward finite-difference method. We show that our pipeline has the potential to greatly improve the training results for regression tasks compared to the other methods, which we also validate on multiple real-world regression datasets.

The remainder of this paper is organized as follows. In Section 2 we present our training pipeline for Sobolev Training with approximated derivatives. Results of our experiments with various black-box function regression tasks are presented in the subsequent Section 3. Finally, we summarize our results in Section 4.

2 Sobolev Training with approximated Target Derivatives

Our goal is to enable Sobolev Training for the regression of black-box functions. The difficulty here is that no analytical description of the target function is available, and therefore the required information about the target derivatives are not available in general. We overcome this problem by approximating these derivatives. The training pipeline presented in the following facilitates the approximation of the target derivatives on the basis of the data already collected (i.e. without the need to collect more data). Moreover, our proposed pipeline describes the sequence of steps in which the actual training is embedded. This sequence comprises of steps such as data preprocessing, which need to be tailored to the Sobolev Training.

Algorithm 1 gives an overview over the training pipeline. The inputs to the training pipeline are the training input data X and the corresponding function values Y of the target function. The first step is to approximate the partial derivatives of the target function (details are given in Section 2.1). Subsequently, inputs, outputs and derivatives are transformed while preserving their relative magnitudes (described in Section 2.2). Finally, the neural network is trained using the corresponding error function and the respective sobolev weight factor, as introduced in Section 2.3, until the stopping criterion is met.

2.1 Target Derivative Approximation

We propose to approximate the partial derivatives of the target function with respect to all input dimensions evaluated at each training input. In the following, we describe this approximation for a single input $X_s \in \mathbb{R}^n$, where X_s represents the s^{th} row of the training input matrix $X \in \mathbb{R}^{m \times n}$.

The aim is to approximate the partial derivatives of the unknown target function $f(x)$ at input X_s with $x, d \in \mathbb{R}^n$ such that

$$d_i \approx \left. \frac{\delta f(x)}{\delta x_i} \right|_{x=X_s}. \tag{1}$$

This is done by approximating $f(x)$ with a linear model in the neighborhood of the regarded training input X_s. For this, the function values of the p nearest neighbors $f(n_1), \ldots, f(n_p)$ to X_s (with respect to the euclidean distance $||n_i - X_s||^2$) are used, where each n_i represents a row of the input matrix X (note that the indices of n_i do *not* correspond to the index of the row in the training input matrix X). Approximating the partial derivatives then results in a least squares problem

$$min_d ||W(Ad - b)||^2, \tag{2}$$

where A contains the differences of the inputs

$$A = \begin{pmatrix} (n_1 - X_s) \\ \vdots \\ (n_p - X_s) \end{pmatrix}, \tag{3}$$

and b contains the difference of the output values

$$b = \begin{pmatrix} f(n_1) - f(X_s) \\ \vdots \\ f(n_p) - f(X_s) \end{pmatrix}. \tag{4}$$

The p nearest neighbors are determined using a k-dimensional tree (implementation provided by Pedregosa et al. [14]). W is a diagonal matrix that controls the influence of neighboring training points on the approximation of the derivatives depending on their distance to X_s. The respective diagonal entries are

$$W_{ii} = \frac{1}{\sum_{j=1}^{p} e^{-||n_j - X_s||_2}} e^{-||n_i - X_s||_2}. \tag{5}$$

After solving the minimization problem in Equation 2, d contains the approximated partial derivatives of the target function evaluated at X_s. Thus, solving the respective minimization problem for each input vector in the training dataset determines all target derivatives required for training.

2.2 Data Transformation

Several standard methods for data transformation exist for Value Training, e.g. the *standard scaler* which transforms data to zero mean and unit variance [14]. We propose to adapt this approach for Sobolev Training in order to preserve the relative magnitudes between outputs, derivatives and inputs. The presented method is limited to regression functions with a one-dimensional output. However, the methods can be extended to functions with multidimensional outputs. The adapted transformation comprises three steps.

First, the input values are scaled column-wise to zero mean and unit variance

$$\tilde{x_i} = \alpha_i x_i + \beta_i. \tag{6}$$

Secondly, the output values $y = f(x)$ are shifted such that the mean magnitude of the outputs equals the mean magnitude of the partial derivatives

$$\hat{y} = y + \zeta. \tag{7}$$

This step reduces the magnitude difference between outputs and derivatives. In the third step outputs and derivatives are scaled to

$$\tilde{y} = \gamma \hat{y} \tag{8}$$

and

$$\tilde{d_i} = \frac{\gamma}{\alpha_i} d_i, \tag{9}$$

where d_i are the approximated target derivatives. The factor γ is chosen such that outputs and derivatives have combined unit variance.

2.3 Error Functions

The main distinguishing characteristics between Sobolev Training and Value Training is the function used to compute the error of the network, i.e. the loss which is backpropagated through the network during training. In Value Training, any loss function $l(x, y)$ can be used to compare the output of the network with the desired value, e.g. the mean squared error. The resulting training error e_{VT} for a training input X_s is

$$e_{VT} = l(o, o_d)\Big|_{X_s}, \tag{10}$$

where o is the output of the network for input X_s and o_d is the corresponding desired output.

In Sobolev Training, the neural network is trained to fit the desired outputs *and* the respective partial derivatives of the target function. This is achieved by adding terms to the error function of Value Training. By that, discrepancies between the partial derivatives of the neural network to the partial derivatives of the target function are explicitly penalized. The resulting error function e_{ST} is

$$e_{ST} = l(o, o_d) + \rho \sum_{i=1}^{n} l\left(\frac{\delta net(x)}{\delta x_i}, d_i\right)\Bigg|_{X_s}, \tag{11}$$

where n refers to the dimension of the input space. The derivative of the network with respect to input dimension x_i is given by $\frac{\delta net(x)}{\delta x_i}$ and the approximated partial derivatives of the target function $f(X)$ by d.

The terms added for Sobolev Training are weighted with a factor ρ, which determines the importance of these terms in relation to the Value Training loss function. We propose to decrease this weight factor after each epoch of training to emphasize the importance of accurate output values at the end of the training

$$\rho_{\upsilon+1} = \rho_{\upsilon}\rho_{up}, \tag{12}$$

where ρ_{up} is the corresponding update factor and υ the index of the regarded training epoch.

The theoretical basis for the proposed error function is given by Hornik et al. [5] and Czarnecki et al. [1]. Hornik et al. [5] proved that neural networks with at least one hidden layer are able to arbitrarily well approximate any function and its derivatives if the activation function of the neurons is appropriately smooth. Moreover, Czarnecki et al. [1] showed that Rectified Linear Units (ReLU) can be used as activation function to achieve universal approximation of function values and derivatives up to the first order. Furthermore it should be noted that any gradient-based optimization algorithm such as Adam [7] used for Value Training can be used for Sobolev Training [15].

2.4 Derivative Approximation using Finite-Differences

In order to compare our pipeline with the approach of using a straightforward finite-difference method for derivative approximation, we introduce a slightly modified training pipeline depicted in Algorithm 2. This approach requires additional data collected in a small neighborhood of the given training data. To ensure a fair comparison, we consider this extra effort. Therefore, we assume that the total amount of data which can be collected is limited, i.e. there is a trade-off between collecting data to explore the whole input space versus collecting data for derivative approximation.

For example, if the target function has two-dimensional inputs, the training dataset size passed to the pipeline in Algorithm 2 is only one-third of the size used for the other training approaches. This results from the fact that additional data points required for derivative approximation are collected during the execution of this pipeline. However, the total number of target function evaluations remains the same for all training methods, which allows a fair comparison of the methods.

Derivatives are approximated using a *one-sided-difference* approach [12] defined as:

$$\frac{\delta f(x)}{\delta x_i} \approx \frac{f(x + \epsilon e_i) - f(x)}{\epsilon} \tag{13}$$

Algorithm 2 Sobolev Training with Finite Differences

```
In-/Output: Input Data $X$, Output Data $Y$
 1: Collect Data for Derivative Approximation
 2: Approximate Target Derivatives using Finite-Differences
 3: Transform Data
 4: Initialize Neural Network and Optimizer
 5: Initialize Sobolev Weight Factor $\rho$
 6: while Stopping Criterion not met do
 7:     Shuffle the Dataset and create Batches
 8:     for batch in Batches do
 9:         Compute Gradients of the Error for batch w.r.t. the Weights of the Neural
            Network
10:         Update the Weights of the Neural Network
11:     end for
12:     Update $\rho$
13: end while
```

Where e_i has zero entries except for the i^{th} entry and ϵ is the step-size used for derivative approximation. We use the *one-sided-difference* approach as the small gain in accuracy obtained by using the *two-sided-difference* often does not justify the extra effort for collecting additional data [12].

To make use of all obtained information we propose to add the data collected for derivative approximation to the training data. Moreover, for each of these data points one partial derivative can be added to the training data without extra effort. This is achieved by exchanging e_i with $-e_i$ in Equation 13 to calculate the respective partial derivative at the new data point. After adding the additional data to the training dataset, the number of training points for this method is the same as for the other training methods. However, the data distribution of the training data is different.

The one-sided-difference (Equation 13) results in an error linearly depending on the chosen step-size ϵ, i.e. the error is $\mathcal{O}(\epsilon)$ [12]. Hence, in order to decrease the approximation error, a sufficiently small step size must be chosen. However, note that choosing ϵ too small can lead to underflow in the input as well as the output data depending on the machine precision.

The other steps of the training pipeline remain the same as introduced before. It should be noted that this training pipeline can only be used for regression tasks where training data points can be collected at any desired position in the input space. This is, however, *not* the case for most real world applications.

3 Results

We compare our training pipeline to existing approaches in terms of their performance on approximating specific black-box functions. The black-box functions are chosen to represent functions with different shapes, input-value domains and output-value ranges. Therefore, we chose optimization test functions with

Table 1. Black-Box Functions used in the experiments and their respective category

Function	Category	Definition	Input Range
Sphere	Bowl-Shaped	$f(x) = \sum_{i=1}^{n} x_i^2$	$x_i \in [-5.12; 5.12]$
Sum Diff. Pow.	Bowl-Shaped	$f(x) = \sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$	$x_i \in [-1; 1]$
Sum Squares	Bowl-Shaped	$f(x) = \sum_{i=1}^{n} i x_i^2$	$x_i \in [-10; 10]$
Trid	Bowl-Shaped	$f(x) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1}$	$x_i \in [-4; 4]$
Booth	Plate-Shaped	$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2$	$x_i \in [-10; 10]$
McCormick	Plate-Shaped	$f(x) = sin(x_1 + x_2) + (x_1 - x_2)^2$ $-1.5x_1 + 2.5x_2 + 1$	$x_1 \in [-1.5; 4]$ $x_2 \in [-3; 4]$
Matyas	Plate-Shaped	$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$	$x_i \in [-10; 10]$
Powersum	Plate-Shaped	$f(x) = \sum_{i=1}^{n} [(\sum_{j=1}^{n} x_j^i) - b_i]^2$	$x_i \in [0; 2]$ $b = (8, 18)^T$
Rosenbrock	Valley-Shaped	$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$x_i \in [-5; 10]$
Three Hump Camel	Valley-Shaped	$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$	$x_i \in [-5; 5]$
Six Hump Camel	Valley-Shaped	$f(x) = (4 - 2.1x_1^2 + \frac{x_1^4}{3})x_1^2$ $+x_1x_2 + (-4 + 4x_2^2)x_2^2$	$x_1 \in [-3; 3]$ $x_2 \in [-2; 2]$
Dixon Price	Valley-Shaped	$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{n} i(2x_i^2 - x_{i-1})^2$	$x_i \in [-10; 10]$
Easom	Steep Ridges	$f(x) = -cos(x_1)cos(x_2)exp(-(x_1 - \pi)^2 - (x_2 - \pi)^2)$	$x_i \in [-5; 5]$
Michalewicz	Steep Ridges	$f(x) = -\sum_{i=1}^{n} sin(x_i) sin^{20}(\frac{i x_i^2}{\pi})$	$x_i \in [0; \pi]$
Styblinski Tang	Others	$f(x) = \frac{1}{2}\sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i)$	$x_i \in [-5; 5]$
Beale	Others	$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2$ $+(2.625 - x_1 + x_1x_2^3)^2$	$x_i \in [-4.5; 4.5]$
Branin	Others	$f(x) = (x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2$ $+10(1 - \frac{1}{8\pi})cos(x_1) + 10$	$x_1 \in [-5; 10]$ $x_2 \in [0; 15]$
Golstein Price	Others	$f(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2$ $+6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2$ $\times(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$	$x_i \in [-2; 2]$

two-dimensional input from five different categories proposed by Surjanovic and Bingham [19]: Bowl-shaped functions, plate-shaped functions, functions with steep ridges or drops, valley-shaped functions and special functions grouped in the category named *others*. A list of all functions and their definitions can be found in Table 1.

With our experiments we aim to empirically answer the following questions:

– How does our training pipeline compare to the standard training method (Value Training)?
– Does the size of the training dataset or the shape of the target function have an influence on the performance of our training method?
– How does the performance compare to the approach of approximating derivatives directly with a finite-difference method?

These questions are addressed in the following Sections by interpreting the results of our experiments. Furthermore, at the end of this Section we evaluate our training pipeline on several real-world regression problems. The Hyperparameters of our experiments are listed in Table 2. We would like to emphasize

Table 2. Hyperparameters of the experiments

Hyperparameter	Value
Hidden layers	2
Neurons per Layer	200
Hidden Layer Activation	ReLU
Output Layer Activation	Linear
Loss Function	Mean squared error
Optimizer	Adam
Adam - Learning Rate	$1e-3$
Adam - β_1	0.9
Adam - β_2	0.999
Repetitions per experiment	10
Amount Batches	10
Data Collection Strategy	Uniform-Random
Validation Set Size	4000
Test Set Size	4000
Finite-Difference Step Size ϵ	$1e-4$
Number of Neighbors for Least Squares	5
Initial ρ	1.0
Update Factor ρ_{up}	0.95
Stopping Criterion	Validation Error Convergence
Patience p_c	10

that we did not optimize these hyperparameters. Exemplary code of our training methods can be found on GitHub[1].

3.1 Sobolev Training with approximated Target Derivatives versus Value Training

First, we compare the performance of our training pipeline with the standard method of training neural networks (Value Training) by conducting various experiments. Each experiment consists of training a neural network with the considered training method to approximate one of our regression functions. Experiments are carried out with different training dataset sizes, whereas the collected training data is distributed random-uniformly over the input space. During the experiments, the number of training epochs is not limited. Instead, the neural network is trained until the stopping criterion is met. For our experiments, we chose to stop the training if there is no improvement in the validation error over a period of p_c epochs (p_c is referred to as patience). Each experiment is conducted ten times in order to decrease the influence of statistical effects, where we plot the mean value of the test error in combination with the respective minimum and maximum values. We compare the results of different training methods by means of the mean squared error of the test dataset for the trained network. For this, the network parameters of the epoch with lowest validation set error

[1] https://github.com/MatthiasKi/SobolevTrainingApproxDerivatives

are used to compute the test error (in general, this is not the test error of the last training epoch). To make the results comparable regarding the differences in the output data ranges, we divide this error by the mean squared output of the respective function (the mean squared output is calculated beforehand and the same value is used to normalize all training results of the same target function). All relevant hyper parameters used for training are depicted in Table 2.

The reader can see the training results for one representative function of each category in Figure 1. As expected, Sobolev Training with exact derivative information clearly outperforms the other training methods for most functions. This results from the extra information added to the training algorithm (note that for Sobolev Training with exact derivatives the same training data as for Value Training is used, plus the partial derivatives of the target function for each training data point). Furthermore, Sobolev Training with Least Squares approximated derivatives consistently (except for the function *Michalewicz*) achieves better results than Value Training.

In order to combine the training results of different functions of the same category in one plot, we consider the test error of the respective training method divided by the test error achieved by using Value Training. By that, the magnitude of the output data is canceled out and the results for different functions can be directly compared with each other. The combined performance for functions of the same category is depicted in Figure 2. In the Figure, the lines represent the mean values over the results of all functions of the respective function category (whereas for each target function, training method and training dataset size 10 independent experiments have been conducted). In addition, the minimum and maximum performance of the respective function category are depicted. Values greater than 1.0 indicate that the respective training method achieved higher test errors than Value Training (and therefor has a worse performance), and values less than 1.0 indicate a lower test error, respectively. We observe that Sobolev Training with Least-Squares approximated derivatives achieves consistently lower test errors than Value Training (except for functions with steep ridges or drops).

For some function shapes the effect of our training pipeline is bigger than for others. This is due to the different value of information about derivatives for different function shapes. This can also be seen by looking at the effect of classical Sobolev Training compared to Value Training for the respective function shapes (i.e. Sobolev Training with exact derivatives). For example, Sobolev Training with exact derivatives achieves much lower relative test errors than Value Training for valley shaped functions compared to plate shaped functions (as depicted in Figure 2). In case of our functions with steep ridges or drops, Sobolev Training even tends to worsen the training results. This fits to the expectations as information about target derivatives is of small value for functions of these shapes and, due to the limited accuracy of the learned model, can even be misleading for some functions.

In general, the effect of our training pipeline increases with the number of data points in the training set. This is in line with the expectations, as increasing

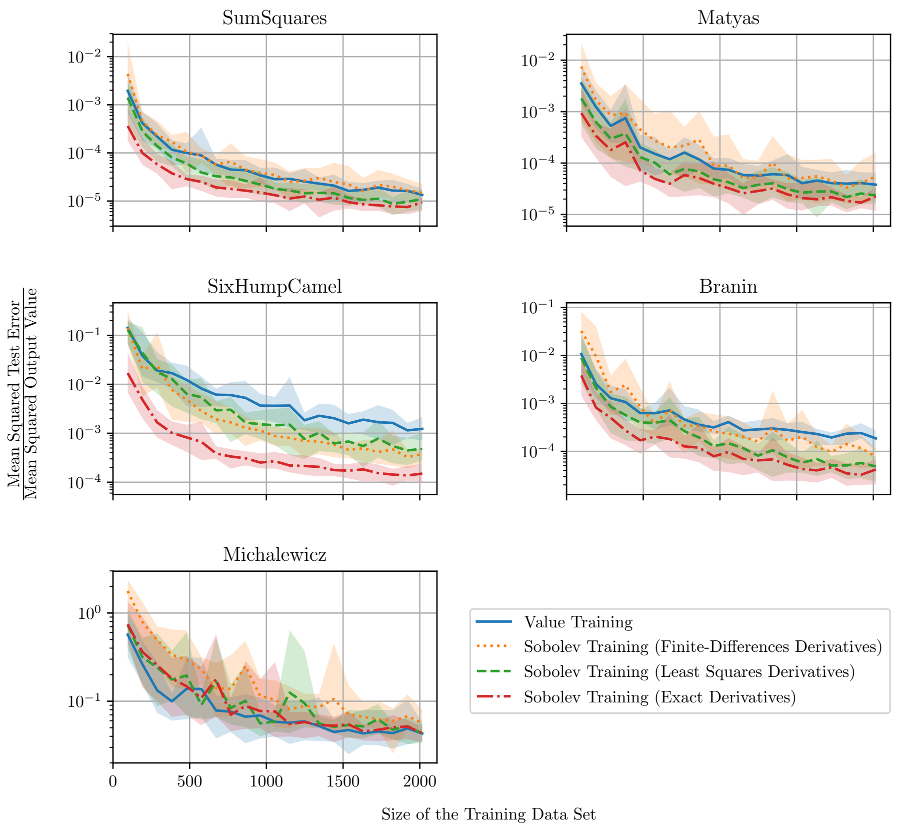

Fig. 1. Comparison of the test errors for different training methods

the dataset size leads to a higher density of the data points distributed in the input space, i.e. neighboring points in the dataset lie closer to each other. This in turn increases the accuracy of the derivative approximation, since as indicated in Section 2.4, the approximation error of the derivatives increases linearly with the distance of neighboring points.

3.2 Sobolev Training with approximated Derivatives based on Finite-Differences

In this Section, we compare the performance of our training pipeline with the approach of approximating derivatives directly using a finite-difference method. As explained in Section 2.4, the size of the training dataset passed to the training pipeline directly using finite-differences is one third of the size of the training dataset used for Value Training. This ensures a fair comparison, as the number of data points which can be collected is limited for most applications.

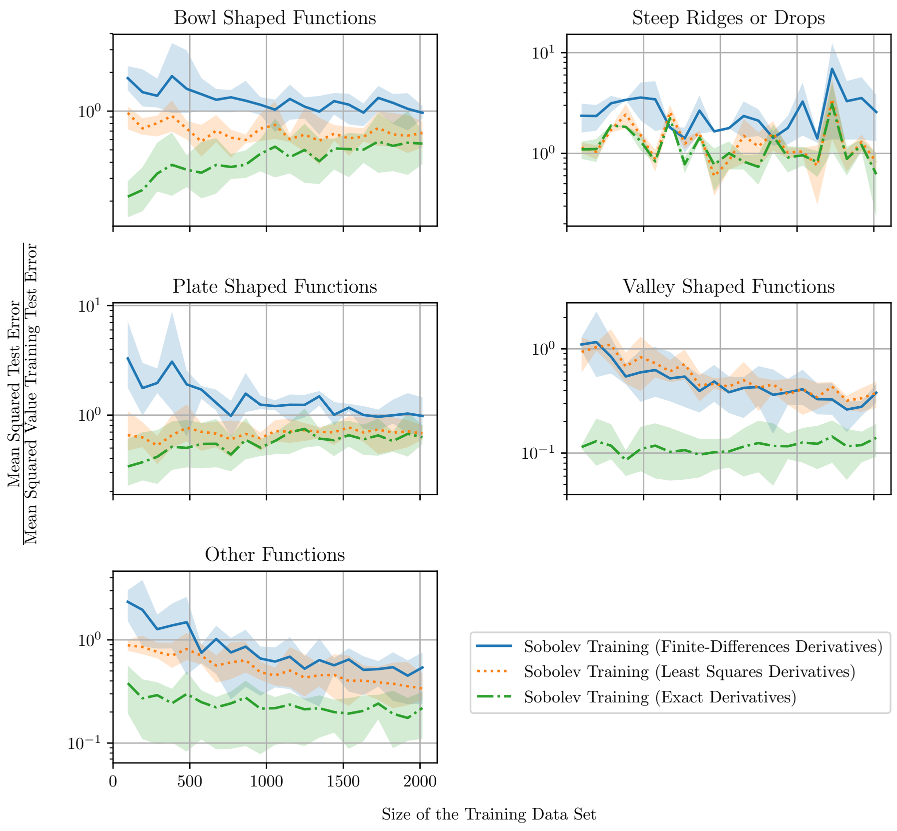

Fig. 2. Comparison of the test error for different training methods in relation to the test error achieved with Value Training

As shown in Figure 2, Value Training outperforms Sobolev Training with Finite-Difference approximated derivatives especially in the low-data regime. This is due to the fact that this method uses some of the samples for derivative approximation, i.e. obtaining more local information instead of exploring unknown regions of the input space. Of course, this effect decreases for larger training datasets.

For all function shapes except for valley shaped functions, our training pipeline clearly outperforms the straightforward approach of using finite-differences for derivative approximation. Our experiments induce that the local additional information in the form of accurate derivatives provide a great deal of additional value for describing our valley shaped functions. This also explains the small magnitude of the relative test error of Sobolev Training with exact derivatives compared to Value Training. Of course, the derivatives approximated with the direct finite-difference method are more accurate than the least-squares approximated ones, which leads to the good performance of this training method for valley-shaped functions.

3.3 Real-World Regression Problems

To show the applicability of our pipeline for problems with high dimensional inputs and noise in the collected data, we evaluate our pipeline with several existing datasets for real-world regression problems. In contrast to the previous Subsections, we do not intend to represent a broad spectrum of different regression problems with our selection. Instead, we aim to show the abilities of our pipeline to be applied to real-world regression tasks using problems selected from the UCI Machine Learning Repository [3], which are presented in the following. Moreover, Table 3 gives an overview over the properties of the different datasets.

– *Combined Cycle Power Plant* [20, 6]: The goal of this regression problem is to predict the net hourly electrical energy output of a combined cycle power plant composed of gas turbines, steam turbines and heat recovery generators. The predictions are based on features such as the ambient pressure and the relative humidity.
– *Communities and Crime* [21–24]: This dataset contains data about different communities such as the age distribution of its population or the number of full time police officers. The aim is to predict the total number of violent crimes per population for each community based on these features. Note that for our experiments, we considered only features which are available for all communities.
– *Concrete Compressive Strength* [27]: This dataset was created to analyze the compressive strength of concrete based on features like the age of the concrete or its components (e.g. the amount of cement).
– *Yacht Hydrodynamics* [4, 13]: The purpose of this dataset is to find a connection between the residuary resistance of sailing yachts and geometric features of the yacht like the hull geometry coefficients.

For our experiments, each dataset is split randomly into training, validation and test set, whereas the validation set and the test set comprise 20% of the total data each. The calculation of the neighboring points of a considered sample was performed using the transformed data matrix as described in Section 2.2. This is necessary, because the entries of the data matrix can have different units or can be of different orders of magnitudes for datasets comprising real-world data, which can distort the calculation of the neighboring points. In addition, we found that the weighted data matrix can be ill-conditioned in some cases. Therefore, we cut-off singular values which are smaller than $0.01\sigma_{max}$ to compute the least-squares solution, where σ_{max} is the largest singular value of A belonging to the least-squares problem $min_x||Ax - b||^2$.

Each experiment was performed 50 times to account for the random initialization of the neural networks and the shuffling of the data before splitting into training, validation and test datasets. We report the mean and standard deviation of the root mean squared errors of both models on the respective test dataset

Table 3. Properties of the datasets for the real-world regression problems (*Considered Neighbors* is the number of neihbors considered for the least-squares approximation of the partial derivatives)

Dataset name	Input Dimensionality	Dataset Size	Considered Neighbors
Combined Cycle Power Plant	4	9568	20
Communities and Crime	101	1994	3
Concrete Compressive Strength	9	1030	10
Yacht Hydrodynamics	7	308	2

Table 4. Mean and standard deviation of the RMSE over 50 independent experiments trained with either Value Training or our proposed training pipeline

Dataset name	LS-Sobolev Training	Value Training
Combined Cycle Power Plant	$\mathbf{3.97 \pm 0.13}$	4.01 ± 0.12
Communities and Crime	$\mathbf{369.95 \pm 26.02}$	375.42 ± 25.93
Concrete Compressive Strength	5.91 ± 0.59	$\mathbf{5.24 \pm 0.5}$
Yacht Hydrodynamics	$\mathbf{1.53 \pm 0.42}$	2.16 ± 0.8

after training in Table 4. The experiments show that our pipeline can have advantages for some real-world regression problems with high-dimensional inputs and noise in the collected data. Note that we did not tune the models, nor performed a hyperparameter optimization (we used the hyperparameters depicted in Table 2 except for the number of neighboring samples used for derivative approximation for which we found suitable numbers by hand). Indeed, a sophisticated parameter optimization is not needed, since our focus lies on the comparison between Value Training and Sobolev Training with Least-Squares approximated derivatives, and we therefore only have to guarantee fair comparison conditions.

4 Conclusion

We introduced a training pipeline for neural networks, which makes it possible to use Sobolev Training for black-box function regression tasks where the target derivatives are not directly accessible. Our pipeline describes the various steps necessary for training, which includes a preprocessing procedure designed for Sobolev Training.

With our experiments we showed empirically that our training pipeline outperformed the standard training approach (i.e. Value Training) for functions with various different shapes. Furthermore, our approach outperforms the straightforward approach of approximating derivatives using finite-differences. In addition to experiments with optimization functions from different categories, we illustrated the practical benefit by evaluating our training pipeline on multiple real world regression problems.

Our pipeline does not require additional training samples and has no special requirements on the data generation process. We observed that our training method leads to improved performance for almost all tested functions, especially if the training dataset is large. Therefore, we believe that the presented training pipeline has the potential to greatly improve the training results for many regression applications.

Our results raise further research questions that lie out of the scope of this paper. For example, it would be interesting to examine the influence of the training data distribution on the learning performance, or to explicitly use information about the data distribution for approximating target derivatives.

References

1. Czarnecki, W.M., Osindero, S., Jaderberg, M., Swirszcz, G., Pascanu, R.: Sobolev training for neural networks. In: Advances in Neural Information Processing Systems. pp. 4278–4287 (2017)
2. Drucker, H., Le Cun, Y.: Double backpropagation increasing generalization performance. In: IJCNN-91-Seattle International Joint Conference on Neural Networks. vol. 2, pp. 145–150. IEEE (1991)
3. Dua, D., Graff, C.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
4. Gerritsma, J., Onnink, R., Versluis, A.: Geometry, resistance and stability of the delft systematic yacht hull series. International shipbuilding progress **28**(328), 276–297 (1981)
5. Hornik, K., Stinchcombe, M., White, H.: Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural networks **3**(5), 551–560 (1990)
6. Kaya, H., Tüfekci, P., Gürgen, F.S.: Local and global learning methods for predicting power of a combined gas & steam turbine. In: Proceedings of the International Conference on Emerging Trends in Computer and Electronics Engineering ICETCEE. pp. 13–18 (2012)
7. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (2014)
8. Lampinen, J., Selonen, A.: Multilayer perceptron training with inaccurate derivative information. In: Proc. 1995 IEEE International Conference on Neural Networks ICNN. vol. 95, pp. 2811–2815. Citeseer (1995)
9. Lee, J.W., Oh, J.H.: Hybrid learning of mapping and its jacobian in multilayer neural networks. Neural computation **9**(5), 937–958 (1997)
10. Masuoka, R., Thrun, S., Mitchell, T.M.: Constraining neural networks to fit target slopes (1993)
11. Mitchell, T.M., Thrun, S.B.: Explanation-based neural network learning for robot control. In: Advances in neural information processing systems. pp. 287–294 (1993)
12. Nocedal, J., Wright, S.J.: Numerical optimization 2nd (2006)
13. Ortigosa, I., Lopez, R., Garcia, J.: A neural networks approach to residuary resistance of sailing yachts prediction. In: Proceedings of the international conference on marine engineering MARINE. vol. 2007, p. 250 (2007)
14. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)

15. Pukrittayakamee, A., Malshe, M., Hagan, M., Raff, L., Narulkar, R., Bukkapatnum, S., Komanduri, R.: Simultaneous fitting of a potential-energy surface and its corresponding force fields using feedforward neural networks. The Journal of chemical physics **130**(13), 134101 (2009)
16. Pukrittayakamee, A., Hagan, M., Raff, L., Bukkapatnam, S.T., Komanduri, R.: Practical training framework for fitting a function and its derivatives. IEEE transactions on neural networks **22**(6), 936–947 (2011)
17. Rifai, S., Mesnil, G., Vincent, P., Muller, X., Bengio, Y., Dauphin, Y., Glorot, X.: Higher order contractive auto-encoder. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 645–660. Springer (2011)
18. Simard, P., Victorri, B., LeCun, Y., Denker, J.: Tangent prop-a formalism for specifying selected invariances in an adaptive network. In: Advances in neural information processing systems. pp. 895–903 (1992)
19. Surjanovic, S., Bingham, D.: Virtual library of simulation experiments: Test functions and datasets. retrieved january 7, 2019, from http://www.sfu.ca/ ssurjano (2013)
20. Tüfekci, P.: Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. International Journal of Electrical Power & Energy Systems **60**, 126–140 (2014)
21. U. S. Department of Commerce, Bureau of the Census, Census Of Population And Housing 1990 United States: Summary tape file 1a & 3a (computer files)
22. U.S. Department Of Commerce, Bureau Of The Census Producer, Washington, DC and Inter-university Consortium for Political and Social Research Ann Arbor, Michigan: (1992)
23. U.S. Department of Justice, Bureau of Justice Statistics, Law Enforcement Management And Administrative Statistics, U.S. Department Of Commerce, Bureau Of The Census Producer, Washington, DC and Inter-university Consortium for Political and Social Research Ann Arbor, Michigan: (computer file) (1992)
24. U.S. Department of Justice, Federal Bureau of Investigation: Crime in the united states (computer file) (1995)
25. Varga, D., Csiszárik, A., Zombori, Z.: Gradient regularization improves accuracy of discriminative models (2017)
26. Witkoskie, J.B., Doren, D.J.: Neural network models of potential energy surfaces: Prototypical examples. Journal of chemical theory and computation **1**(1), 14–23 (2005)
27. Yeh, I.C.: Modeling of strength of high-performance concrete using artificial neural networks. Cement and Concrete research **28**(12), 1797–1808 (1998)