Data Driven Derivative-based Regularization for Regression

Enrico Lopedoto*, Kizito Salako*, Maksim Shekhunov†, Vitaly Aksenov*, Tillman Weyde*
*City St George's University of London, UK, †ITMO University, St Petersburg, Russia

***Abstract*—In this work, we introduce a novel approach to regularization in multivariable regression problems. Our regularizer, called *DLoss*, penalizes differences between the model's derivatives and derivatives of the data generating function as estimated from the training data. We call these estimated derivatives *data derivatives*. The goal of our method is to align the model to the data, not only in terms of target values but also in terms of the derivatives involved. To estimate data derivatives, we select (from the training data) 2-tuples of input-value pairs, using either nearest neighbor or random selection. We evaluate the effectiveness of *DLoss* on synthetic and real datasets with different weights, to the standard mean squared error loss. The experimental results show that with *DLoss* (using nearest neighbor selection) we obtain, on average, the best rank with respect to MSE on validation data sets, compared to no regularization, L_2 regularization, and Dropout. Our implementation code is available on Github.**

***Index Terms*—Machine Learning, Neural Networks, Regularization, Regression, DLoss, Data Derivative**

I. Introduction

Regularization is used by many different methods in statistics, inverse problems, and machine learning, to prevent overfitting and numeric instability when a model is fitted to data [Kukacka et al., 2017]. Traditional regularization methods use an additional term in the loss function used during model training, where this additional term is based on the model parameters. Most frequently used are the L_1 [Tibshirani, 1996] and L_2 [Tikhonov, 1943] norms of model parameters (i.e., norms of the coefficients of a linear model or the weights of a neural network). Another widely used regularization method (but only for neural networks) is *DropOut*, where there is no additional loss term but neurons are randomly switched off during training [Hinton et al., 2012], [Srivastava et al., 2014].

The distinguishing feature of all these regularizers is that they do not explicitly use characteristics of the unknown target function, where these characteristics may be inferred from training data. In this paper, we propose a novel regularization approach for regression, called *DLoss*, that (a) is derived from training data, and (b) is applicable to any type of model where one can specify a loss function. *DLoss* is formulated as a term in the loss function that penalizes the difference between a model derivative and a derivative estimated from data points.

Our contributions are:

1) We propose *DLoss*, a new regularization method for regression models;
2) We propose two variants of *DLoss* using nearest neighbor or random selection of data points;
3) We provide a publicly available implementation in PyTorch, available at https://github.com/EnricoLope/DLoss;
4) We evaluate our approach on different datasets and compare it to training using no regularization, L_2 regularization, and Dropout.

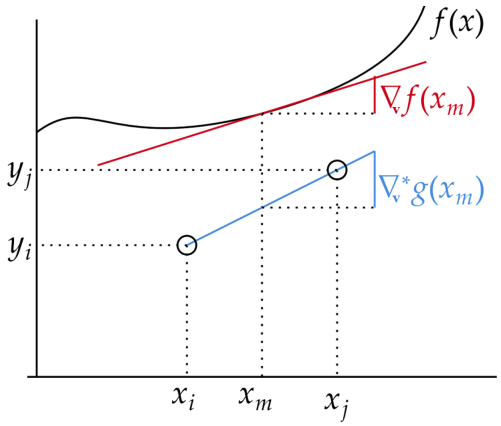

Fig. 1: *DLoss* approach in the one-dimensional case for a tuple of pairs $((\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j))$. On the horizontal axis we show the inputs x, which are scalars in this example. On the vertical axis, we show the scalar regression targets. In general, for a pair of points $\mathbf{x}_i$ and $\mathbf{x}_j$, we calculate the *midpoint* $\mathbf{x}_m = (\mathbf{x}_i + \mathbf{x}_j)/2$ and the difference vector $\mathbf{v} = \mathbf{x}_j - \mathbf{x}_i$. The red line shows the model derivative $\nabla_{\mathbf{v}} f(\mathbf{x}_m)$ calculated at $\mathbf{x}_m$. The blue line shows the estimated derivative $\nabla^*_{\mathbf{v}} g(\mathbf{x}_m)$ calculated at $\mathbf{x}_m$. With *DLoss*, we aim to make the blue and the red lines parallel.

The results show that *DLoss* leads on average to better generalization results at a moderate increase of the computational cost.

The paper is structured as follows. In section II, we discuss related work on regularization and differential equations. In section III, we introduce our method. In section IV, we describe the setup of our experiments. In section V, we present and discuss the experimental results, and in section VI we draw conclusions and discuss possible future work.

II. Related Work

Our method uses information derived from the data and is not specific to any model type as opposed to common regularization approaches. Since we are using the derivatives in our approach we also address the related work on Neural Differential Equations.

A. Regularization

Regularization aims to improve generalization [Kukacka et al., 2017]. The most common regularization methods, L_1 and L_2 loss on the parameters, are also called *weight decay* when applied to neural networks. In generalized linear models and neural networks, lower weights lead to smaller gradients of the model function with respect to the inputs. Intuitively speaking, preferring smaller gradients means that flat and smooth functions are preferred. This preference is independent of the training data.

DropOut is another popular regularization technique and is specific to neural networks. It works by switching off neurons at random during training [Srivastava et al., 2014], [Baldi and Sadowski, 2013].

B. Neural Networks, Derivatives and Differential Equations

Our model adds differential terms to the loss function. While we believe that this our approach is novel, there are various types of models in the literature that integrate differential equations with neural networks in other ways.

Approximating derivatives of known functions by neural networks was already studied by [Hornik et al., 1990] who showed that neural networks can approximate functions and their derivatives. This approach assumes that the derivatives of the approximated function are known, and a modification of the network architecture is required in order to achieve the approximation. A more recent approach to approximating a known function with neural networks has been presented in [Avrutskiy, 2017], which operates similar to our approach and does not require a change to network architectures. It extended to higher-order derivatives in [Avrutskiy, 2021].

Integrating prior knowledge by using derivatives was, to the best of our knowledge, first introduced in [Simard et al., 1991] as *tangent prop*. Tangent prop integrates prior knowledge about invariances into the network: the network is trained using desired directional derivatives of a target function with respect to the changes in the inputs. This is used in the context of image classification to encourage small derivatives in directions where changes of the inputs should not affect the output. For example, for handwritten digits, rotations by a small amount should not affect the class output. Tangent prop results in a fitted neural network that approximates the desired differential behavior.

This approach has, in recent years, been re-kindled in the more general framework of physics-informed neural networks (PINNs) [Raissi et al., 2019], [Cuomo et al., 2022], which injects knowledge from physics into neural networks, in the form of differential equations.

Numeric solutions to differential equations by neural networks have been used since the 1990s. Early work by [Lee and Kang, 1990] already introduced this approach, more recent examples are [Parisi et al., 2003] and [Malek and Beidokhti, 2006], and an overview can be found in [Beck et al., 2023].

Neural differential equations (NDE) are a more recent approach to unify neural networks with differential equations for an overview, see [Kidger, 2022]. A neural ordinary differential equation (NODE) [Chen et al., 2018] estimates a continuous-time function that defines an ordinary differential equation (ODE), where this ODE approximates the discrete-time behavior of a recursive process (such as an RNN). In essence, the recursive mapping of inputs to outputs is approximated by motion along the solution to the ODE — i.e., motion along a vector field. Neural stochastic differential equations (NSDE) [Tzen and Raginsky, 2019] define diffusion processes, rather than vector fields. They are also used to learn continuous-time models, but of *stochastic* recurrent processes. In contrast, for our work, we assume the target function is continuously differentiable and its differential characteristics can be estimated from data.

III. Derivative-based Regularization Method

In this study, we introduce a regularization term that aims to align model derivatives with estimated derivatives of the target function. This approach is data driven, and does not impose a prior on the model parameters. Notwithstanding the increasing use of differential equations in combination with neural networks, this form of regularization is, to the best of our knowledge, proposed here for the first time.

A. Intuition

Our regularizer aligns model derivatives with estimated derivatives of the unknown, true, target function that generated the data. By considering derivatives of the target (estimated from training data), in addition to the target values, our approach seems intuitively more promising than other regularization approaches that rely solely on target values, and more generally applicable than approaches that rely on prior knowledge of the target's differential characteristics.

For our regularizer, the following information is needed. Firstly, it needs derivatives of the model with respect to the input. These derivatives may be calculated analytically, or can be approximated by a finite difference approach given the model. Secondly, it needs estimates of target function derivatives with respect to the input, which we estimate with a simple finite-difference approach. We want to minimize the difference between the model derivative and the estimated target derivative; i.e., we want to minimize the value of a *DLoss* function, which we will define below.

B. Regression

We focus on *regression problems*, that is, the approximation of an unknown, continuous, real-valued target function with k arguments, $g(\cdot) : \mathbb{R}^k \to \mathbb{R}$, by a model, $f(\cdot, \boldsymbol{\beta}) : \mathbb{R}^k \to \mathbb{R}$, parameterized by a vector $\boldsymbol{\beta} \in \mathbb{R}^m$. Using training data consisting of input-output pairs $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, n$, where n is the number of training data points, regression modeling aims to optimize the parameter vector $\boldsymbol{\beta}$ of this function f, so that for general input-output pairs $(\mathbf{x}, y)$,

$$f(\mathbf{x}, \boldsymbol{\beta}) = \hat{y} \tag{1}$$

approximates the unknown function $g(\mathbf{x}) = y$ sufficiently well. The variable $\mathbf{x} \in \mathbb{R}^k$ is called an *independent variable vector* in statistics, and called *feature* or *input vectors* in machine learning. The corresponding $y \in \mathbb{R}$, that satisfies $g(\mathbf{x}) = y$, is the *dependent variable*, *response variable* or *target*. In this paper, we refer to an input vector $\mathbf{x}$ as a point, to $(\mathbf{x}, y)$ as a pair, and to two pairs $((\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j))$ as a tuple.

For a given parameter vector $\boldsymbol{\beta}$, the model makes a prediction, $\hat{y}_i = f(\mathbf{x}_i, \boldsymbol{\beta})$, for the ith training input $\mathbf{x}_i$. In general, it is expected that $\hat{y}_i$ will differ from the target value y_i. The task is to determine a preferred $\boldsymbol{\beta}$, via training using $(\mathbf{x}_i, y_i)$ pairs that minimizes a suitable loss function; i.e., minimizes a suitable measure of the error between the predicted values $\hat{y}_i$ and target values y_i. One has "fitted the model to the data" when a preferred $\boldsymbol{\beta}$ has been determined.

The standard loss function to optimize is the *mean squared error* (MSE) (i.e., the mean of the squared residuals):

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i, \boldsymbol{\beta}))^2 . \tag{2}$$

In standard multivariable linear modeling, we assume $y = g(\mathbf{x}) + \epsilon$, where ϵ is a normally distributed random variable representing noise, in which case the solution that optimizes the MSE (the least squares solution) also maximizes the likelihood of the data see, e.g., [Hastie et al., 2001]. We optimize $\boldsymbol{\beta}$ using only the training data — a separate *validation dataset* is used to assess how well the fitted model performs on data (specifically, model inputs) it was not trained on. It is common to find that the value of $\boldsymbol{\beta}$ that gives the best model performance on the training set does not give good model performance on the validation set, which is called overfitting.

C. Regularization

Traditional regularization discourages overfitting as follows: during model training, an additional term to the loss function helps find a parameter vector, $\boldsymbol{\beta}$, that ensures $f(\cdot, \boldsymbol{\beta})$ approximates the unknown target function well on unseen data from the same data distribution see, e.g., [Kukacka et al., 2017]. The most common additional terms are the L_1 norm and the squared L_2 norm of the parameter vector (denoted $\|\boldsymbol{\beta}\|_1$ and $\|\boldsymbol{\beta}\|_2^2$). These terms are typically applied with a coefficient $\theta \in \mathbb{R}$, that regulates how much large weights (i.e., large model parameters) are penalized, so that the total loss L is

$$L = \theta \|\boldsymbol{\beta}\|_i^i + MSE , \quad \text{for } i \in \{1, 2\}. \tag{3}$$

Both L_1 and L_2 norms encourage the weights to be as small as possible. The L_1 norm, $\|\boldsymbol{\beta}\|_1 = \sum_{j=1}^{m} |\boldsymbol{\beta}_j|$, corresponds to a Laplacian prior with 0 mean [Williams, 1995], [Plaut et al., 1986], [Tibshirani, 1996]. The L_2 norm, $\|\boldsymbol{\beta}\|_2 = \sqrt{\sum_{j=1}^{m} \boldsymbol{\beta}_j^2}$, corresponds to a Gaussian prior with 0 mean and standard deviation θ^{-1} [Rennie, 2003], [Bishop, 1995].

D. Derivative Error

We propose a new regularization term, the *DLoss*, that is defined on the difference between the model's derivative (analytically or numerically calculated) and the target's derivative (estimated from the training data). We use the finite-difference method (FDM) to estimate both derivatives [Paszyński et al., 2021], [Samaniego et al., 2020]. Intuitively, the main idea of *DLoss* is to align the slope of our model $f(\mathbf{x}, \boldsymbol{\beta})$ and the slope of the unknown target function $g(\mathbf{x})$, as illustrated in Figure 1. Since we are dealing with functions of multi-dimensional data, we use *directional derivatives*.

1) Data Derivative: In order to calculate the derivative error, we need to estimate the derivative of the unknown target function from the training data. For the one-dimensional case, derivatives can be estimated based on neighboring points [Kis, 2021]. We expand this approach to multi-dimensional feature spaces with the *data derivative* — an estimate of the directional derivative of g in the direction of vector $\mathbf{v}$, denoted $\nabla_{\mathbf{v}}^* g$. When calculating $\nabla_{\mathbf{v}}^* g$ we select a tuple, $((\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j))$, from the training data set. We calculate the *midpoint* $\mathbf{x}_m = (\mathbf{x}_i + \mathbf{x}_j)/2$ between the points $\mathbf{x}_i$ and $\mathbf{x}_j$, and we obtain the difference vector $\mathbf{v} = \mathbf{x}_j - \mathbf{x}_i$. Then, the data derivative at $\mathbf{x}_m$ along the direction $\mathbf{v}$ is

$$\nabla_{\mathbf{v}}^* g(\mathbf{x}_m) = \frac{y_j - y_i}{\|\mathbf{v}\|_2} . \tag{4}$$

In general, one cannot give any guarantees as to how far the estimated derivative $\nabla_{\mathbf{v}}^* g$ may be from the true derivative $\nabla_{\mathbf{v}} g$. However, $\nabla_{\mathbf{v}}^* g(\mathbf{x}_m)$ is the value of the directional derivative at *some* point between $\mathbf{x}_i$ and $\mathbf{x}_j$. More formally, consider all points $\mathbf{x}_\tau$ of the form

$$\mathbf{x}_\tau = \tau \mathbf{x}_i + (1 - \tau) \mathbf{x}_j \tag{5}$$

with $0 \leq \tau \leq 1$. By the mean value theorem e.g., [Rohde et al., 2012], if g is continuous over $\{\mathbf{x}_\tau | 0 \leq \tau \leq 1\}$ and differentiable over $\{\mathbf{x}_\tau | 0 < \tau < 1\}$, there exists some τ such that

$$\nabla_{\mathbf{v}} g(\mathbf{x}_\tau) = \nabla_{\mathbf{v}}^* g(\mathbf{x}_m) . \tag{6}$$

2) Model Derivative: Consider the function $f(\mathbf{x}, \boldsymbol{\beta})$, viewed as a function of only $\mathbf{x}$ with fixed $\boldsymbol{\beta}$, and denote this $f(\mathbf{x})$. We can obtain the value of $\nabla_{\mathbf{v}} f(\mathbf{x})$—i.e., the directional derivative of f at point $\mathbf{x}$ in the direction of $\mathbf{v}$—either analytically or numerically. As with target derivatives, we use

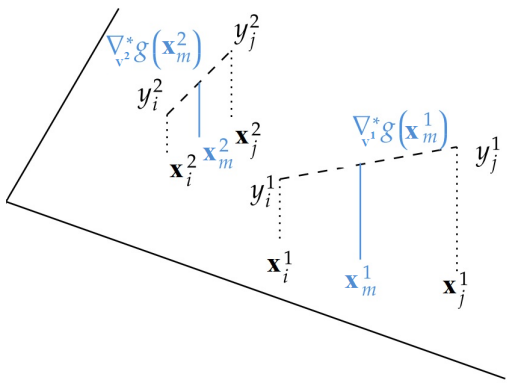

Fig. 2: Illustration of data derivatives calculated from two tuples of pairs. For each tuple $((\mathbf{x}^s_i, y^s_i), (\mathbf{x}^s_j, y^s_j))$, where $s \in \{1, 2\}$, we calculate the midpoint $\mathbf{x}^s_m$, difference vector $\mathbf{v}^s$, and the corresponding data derivative $\nabla^*_{\mathbf{v}^s} g(\mathbf{x}^s_m)$ over a 2-dimensional feature space ($\mathbf{x} \in \mathbb{R}^2$).

an FDM approach to approximate $\nabla_{\mathbf{v}} f(\mathbf{x})$ with $\nabla^{\diamondsuit}_{\mathbf{v}} f(\mathbf{x})$, by calculating

$$\nabla^{\diamondsuit}_{\mathbf{v}} f(\mathbf{x}) = \frac{f(\mathbf{x} + \varepsilon\tilde{\mathbf{v}}) - f(\mathbf{x} - \varepsilon\tilde{\mathbf{v}})}{2\varepsilon} \tag{7}$$

where $\tilde{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|_2}$ is normalized to unit length and ε is a small constant scalar. This approach is more universal, in the sense that one does not need to determine the directional derivative analytically and (using automatic differentiation) it is easy to implement as a loss function.

3) Tuple Selection: We used two alternative algorithms for selecting tuples. Each algorithm, for each training pair $(\mathbf{x}_i, y_i)$, selects l other pairs and, using these pairs, constructs l tuples of the form $((\mathbf{x}_i, y_i), (*, *))$. These tuples are used to estimate derivatives in section III-D4. In Figure 2 we sketch an example of multiple tuples in a multi-dimensional space. For training, l is a hyper-parameter.

Nearest neighbor selection For each training pair $(\mathbf{x}_i, y_i)$, we select its l nearest neighbors. The pair $(\mathbf{x}_j, y_j)$, where $j \neq i$, is a nearest neighbor if the distance $\|\mathbf{x}_j - \mathbf{x}_i\|_2$ is minimal. We determine the l nearest neighbors using the efficient KD-tree algorithm [Bentley, 1975], [Taunk et al., 2019].

This approach approximates the derivative based only on local information. However, if there is noise in the data (see section III-B) that creates a deviation of ϵ from the true value of $g(\mathbf{x}_j) - g(\mathbf{x}_i)$, this will lead to $\epsilon/\|\mathbf{x}_j - \mathbf{x}_i\|_2$ deviation of the estimate $\nabla^*_{\mathbf{v}} g(\mathbf{x}_m)$, so that small distances between points can amplify noise.

Random selection For each training pair $(\mathbf{x}_i, y_i)$, we randomly choose l other pairs from the training set. The idea here is that, for noisy data, there is less influence on $\nabla^*_{\mathbf{v}} g(\mathbf{x}_m)$ since the distances between points are greater. The trade-off is that greater distances between tuples lead to lower accuracy of the estimated derivatives.

The random selection has $\mathcal{O}(nl)$ time complexity while the nearest neighbors point selection with the KD-tree has $\mathcal{O}(nl + n\log(n))$, which may make random selection preferable for very large datasets.

4) Optimization: The *derivative difference* $dd(\mathbf{x})$ is the amount of of the misalignment between estimated model and target derivatives:

$$dd(\mathbf{x}) = \nabla^{\diamondsuit}_{\mathbf{v}} f(\mathbf{x}) - \nabla^*_{\mathbf{v}} g(\mathbf{x})\,. \tag{8}$$

Let $\mathcal{S}$ be an index set for all tuples generated by one of the tuple selection approaches above. For the tuple with index $s \in \mathcal{S}$, we calculate the midpoint, $\mathbf{x}^s_m$, and the derivatives' estimates, $\nabla^{\diamondsuit}_{\mathbf{v}^s} f(\mathbf{x}^s_m)$ and $\nabla^*_{\mathbf{v}^s} g(\mathbf{x}^s_m)$. The *DLoss* aggregates the dd values as the mean squared derivative-difference:

$$DLoss = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} dd(\mathbf{x}^s_m)^2 \tag{9}$$

where $|\mathcal{S}|$ is the cardinality of $\mathcal{S}$. The total loss L for training the model is

$$L = \theta_D DLoss + MSE \tag{10}$$

where θ_D is a weighting coefficient. This loss is minimized with gradient descent.

IV. Experiments

Setup In our experiments, the regression model is a simple feed-forward neural network with a single hidden layer of L_H neurons using ReLU activation. We use ReLU activation because of its popularity in modern neural networks [Goodfellow et al., 2016]. The network calculates the function $\hat{y} = f(\mathbf{x}, \boldsymbol{\beta}) = \sum_{h=1}^{L_H} w_{o,h} r(\mathbf{w}_h \cdot \mathbf{x} + b_h) + b_o$, where $\mathbf{w}_h$ and b_h are the weight vector and bias value of hidden neuron h, respectively, and $\cdot$ denotes the scalar product. The weight between hidden neuron h and the output neuron is $w_{o,h}$ and b_o is the bias of the output neuron. All of these weights and biases make up the vector $\boldsymbol{\beta}$. The rectified linear function (ReLU) $r : \mathbb{R} \to \mathbb{R}$, defined as $r(x) = max(x, 0)$, was to our knowledge first introduced for neural networks by [Fukushima, 1975].

In our experiments, we use the Adam optimizer [Kingma and Ba, 2015]. We compare the effects of using $DLoss$ to L_2 and Dropout (DO) and no regularization (STD). We also compare the different variants of tuple selection - Random (DL_{RND}) and Nearest neighbors (DL_{NN}).

Model parameters Our models have an input layer with a number of neurons determined by the dimensionality of the datasets, a single hidden layer with $L_H = 64$ hidden neurons, and a single linear output neuron. We set ε to 0.001 for calculating the model derivative.

Data In order to observe the effect of our approach in different contexts, we conduct experiments using real data and synthetic, noiseless data. Five commonly available real datasets have been selected: Wine, Cancer, Modechoice, Anes96, Diabetes; further information is given in Table I.

For noiseless data, we created synthetic datasets using data generation functions available from the Scikit-Learn library, where we have selected F1, Regression1, Regression10,

NAME	TYPE	INPUTS	ROWS	LIBRARY
ANES96	REAL	5	944	STM
CANCER	REAL	1	301	STM
DIABETES	REAL	10	442	SCIKIT
MODECHOICE	REAL	6	840	STM
WINE	REAL	11	1599	OML
F1	SYNTH	10	2500	SCIKIT
REGRESSION1	SYNTH	1	2500	SCIKIT
REGRESSION10	SYNTH	10	2500	SCIKIT
SPARSE UNCORR	SYNTH	10	2500	SCIKIT
SWISS ROLL	SYNTH	3	2500	SCIKIT

TABLE I: Datasets used in our experiments. All datasets have a single real-valued output per row. The number of input values is listed in column INPUTS and ROWS is the number of input-output pairs in the dataset. LIBRARY describes the Python package from which the dataset is obtained: *STM* is for statsmodels (https://www.statsmodels.org/), *Scikit* for scikit-learn (https://scikit-learn.org/stable/) and *OML* for OpenML (https://www.openml.org/). The source for "modechoice" dataset is [Greene and Hensher, 2011], "anes96" dataset [The American National Election Studies, 1996], "diabetes" dataset [Efron et al., 2004], "cancer" dataset [Rice, 2007] and "wine" dataset [Cortez et al., 2009]. The synthetic datasets (TYPE: SYNTH) were generated with the Python library *Scikit-Learn*, version 1.4.1, using the methods from the package $sklearn.datasets$. In the generation of the datasets, no noise has been added. Further details of the data generating functions can be found in the library documentation available at https://scikit-learn.org/stable/datasets/.

Sparse-uncorrelated, and Swiss Roll. For these, the training data range of the input $\mathbf{x}$ is in the range $[0, 1]$ with a fixed 2500 uniformly distributed points and 1, 3, or 10 features. Table I shows the main features of the datasets. We use 5-fold cross validation.

Hyper-parameters We ran a grid search over the following hyper-parameters: learning rate $\lambda = 0.03, 0.01, 0.003, 0.001$, L_2 weight $\theta = 10^{[-3,-4,-5,-6,-7]}$, *DLoss* weight $\theta_D = 10^{[-3,-4,-5,-6,-7]}$, and Dropout probability $p = [0.05, 0.1, 0.2, 0.4, 0.8]$. Regularizations L_2, DO, and *DLoss*, are not combined in our experiments. We train our models for 250 epochs and use full batch learning.

Metrics We report the first epoch at which the best accuracy result is achieved (ep) and the time (t) for the full training (250 epochs). The metrics reported are calculated over the 5 cross-validation folds with mean and standard deviation:

- MSE_{train}: the average of the best MSE value per fold on the training set over all epochs;
- $\sigma_{MSE_{train}}$: standard deviation of the best MSE value per fold on the training set over all epochs;
- MSE_{val}: average of the best MSE value per fold on the validation set over all epochs;
- $\sigma_{MSE_{val}}$: standard deviation of the best MSE value per fold on the validation set over all epochs;
- ep: average of the epoch at which the minimum MSE_{val} is reached per epoch;
- t: average time in seconds to complete a single training per single fold.

All experiments were run on a PC with Intel I5 processor and 16GB of RAM running Ubuntu Linux.

DATASET	STD	L_2	DO	DL_{RND}	DL_{NN}
ANES96	5	3	4	2	1
CANCER	5	1	4	3	2
DIABETES	5	3	4	2	1
MODECHOICE	5	3	2	4	1
WINE	5	2	4	1	3
F1	5	1	3	4	2
REGRESSION1	4	2	5	3	1
REGRESSION10	5	1	2	4	3
SPARSE UNCORR	4	5	1	3	2
SWISS ROLL	4	2	5	3	1
Avg	4.7	2.3	3.4	2.9	1.7

TABLE II: The ranks of methods STD, L_2, DO, DL_{RND} and DL_{NN} according to MSE_{val} by $Dataset$ and their averages.

METHOD	MRR	SIGNIFICANCE OF DIFFERENCES			
		L_2	DL_{RND}	DO	STD
DL_{NN}	0.716	-	*	*	**
L_2	0.570		-	-	**
DL_{RND}	0.408			-	**
DO	0.373				*
STD	0.215				

TABLE III: The Mean Reciprocal Rank (MRR) values of the different models and their mutual comparisons (higher is better). '*' or '**' indicate a significant difference, according to a Wilcoxon signed rank test over the model ranks per dataset as shown in Table II, at the 5% or 1% level, respectively.

V. RESULTS

Table V shows the results of our DL_{RND} and DL_{NN} methods as well as STD, L_2 and DO training. DL_{NN} achieves the best generalization errors MSE_{val} for 5 of the 10 datasets.

Table III shows the Mean Reciprocal Ranks (MRR) of the different methods with respect to their validation MSE on the datasets.

We observe that DL_{NN} has the best MRR, but the difference to L_2 is not significant. This is not unexpected, considering the small number of datasets. However, DL_{NN} errors are significantly lower than DL_{RND}, DO and STD, and all regularized models are significantly better than STD, as expected.

A. Discussion

In our experiments the DL_{NN} method showed the best results as measured by average and mean reciprocal rank

(MRR). While the rank difference between DL_{NN} and L_2 is not significant, DL_{NN} is significantly better than the DL_{RND} and DO, while L_2 is not, thus indicating a statistical advantage for DL_{NN} (Table III). We also analyzed the performance differences in terms of the differences of the validation MSE values, where the differences between DO and DL_{NN} not significant (see Table IV), but we see this as less reliable because the error sizes were quite variable between datasets. Overall, there seems to be an advantage to using DL_{NN} that may be worth the additional computation, depending on the application.

It is worth mentioning that the weight θ_D associated with the *DLoss* is on average higher for synthetic data than for real dataset. We assume that both of these observations relate to the fact that the synthetic datasets are noise-free and the *DLoss* therefore more effective.

The current finite-difference calculation of the model derivative requires two additional forward-passes per tuple when calculating the derivative, one for $f(\mathbf{x} + \varepsilon\tilde{\mathbf{v}})$ and one for $f(\mathbf{x} - \varepsilon\tilde{\mathbf{v}})$. Correspondingly, the computation time for *DLoss* is generally higher than the STD models (see t values in Table V). However, the computation times show that all regularization methods require additional time on average, and that the amount is quite variable. It may be possible to reduce the computational cost by not determining the model derivative at the midpoint but reusing data points instead or by using analytical gradient calculations, which would be model dependent.

Overall, both the performance and the computation time warrants further experimentation, including different machines, different models and combinations of different regularization methods.

VI. Conclusion

We have proposed a new regularization method, *DLoss*, which aims to reduce the difference between model derivatives and target function derivatives by using differential information inferred from training data. The *DLoss* method is tested on ten regression datasets. We propose two versions of the method with different tuple-selection algorithms — random and nearest neighbor selection. We experiment on a neural network with a single hidden layer, multiple inputs, and a single output.

The metric we use is the MSE on validation sets to compare the performance of different regularization methods. We observe an overall generalization improvement using *DLoss* with nearest neighbor selection (DL_{NN}). With this method, the generalization as measured by mean reciprocal rank is better than the established L_2 and Dropout regularization.

The improved generalization requires additional computation, roughly, but the results in practice are quite variable between the different regularization methods and algorithmic optimizations may help to reduce the cost. Future work on this approach will include tests with more models and more datasets and exploration of combined regularization methods as well as the formulation of *DLoss* for classification problems. Higher dimensional datasets with more data points and more model architectures will be explored in order to determine how far the benefits of using the *DLoss* method generalize in these situations.

Group	$Median_\Delta$	Wilcoxon p
Real STD DL_{RND}	1.3×10^{-2}	0.035^{*}
Real STD DL_{NN}	4.2×10^{-2}	0.078
Real L_2 DL_{RND}	-8.6×10^{-3}	0.53
Real L_2 DL_{NN}	-4.1×10^{-4}	0.32
Real DO DL_{RND}	1.8×10^{-2}	0.19
Real DO DL_{NN}	4.5×10^{-3}	0.25
Synthetic STD DL_{RND}	3.9×10^{-5}	0.005^{**}
Synthetic STD DL_{NN}	1.8×10^{-5}	0.011^{*}
Synthetic L_2 DL_{RND}	-2.9×10^{-5}	0.85
Synthetic L_2 DL_{NN}	-2.7×10^{-5}	0.77
Synthetic DO DL_{RND}	6.9×10^{-6}	0.42
Synthetic DO DL_{NN}	1.2×10^{-5}	0.26

Group	Shapiro-Wilk p	t-test p
Real STD DL_{RND}	0.046^{*}	0.057
Real STD DL_{NN}	0.37	0.044^{*}
Real L_2 DL_{RND}	0.75	0.48
Real L_2 DL_{NN}	0.022^{*}	0.28
Real DO DL_{RND}	0.017^{*}	0.38
Real DO DL_{NN}	0.56	0.21
Synthetic STD DL_{RND}	$3.6\times10^{-6***}$	0.157
Synthetic STD DL_{NN}	$2.3\times10^{-5***}$	0.121
Synthetic L_2 DL_{RND}	$6.7\times10^{-6***}$	0.241
Synthetic L_2 DL_{NN}	$5.6\times10^{-7***}$	0.144
Synthetic DO DL_{RND}	$7.3\times10^{-6***}$	0.86
Synthetic DO DL_{NN}	$2.2\times10^{-6***}$	0.49

TABLE IV: The results of significance tests for differences of MSE_{val} between STD and *DLoss* methods. We run the tests on the difference of the standard learning method group - STD group - and the group with *DLoss* - DL group. For STD and DL training we have 2 groups, therefore we run all combinations, giving us 6 results for each, real and synthetic data. We compare each of the 5-fold in the cross validation for each of the 5 datasets, thus 25 samples per test. $Median_\Delta$ is the median of pairwise difference between the MSE_{val} values of the groups. We applied the Wilcoxon Signed Ranked Test, a non-parametric test for significance of differences between the medians. The low p-values of the Wilcoxon test indicate that the STD groups' MSE_{val} are significantly greater than the DL groups' (indicated with an asterisk *). The *,**,*** indicate the significance at 5%, 1% and 0.1% levels, respectively. We also applied a paired t-test. However, the t-test is only strictly valid for normally distributed data, and the Shapiro-Wilk test indicates that the distribution is significantly different from a normal distributions in most cases here.

REFERENCES

[Avrutskiy, 2017] Avrutskiy, V. I. (2017). Backpropagation generalized for output derivatives. *CoRR*, abs/1712.04185.

[Avrutskiy, 2021] Avrutskiy, V. I. (2021). Enhancing function approximation abilities of neural networks by training derivatives. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):916–924.

[Baldi and Sadowski, 2013] Baldi, P. and Sadowski, P. J. (2013). Understanding dropout. *Advances in neural information processing systems*, 26.

[Beck et al., 2023] Beck, C., Hutzenthaler, M., Jentzen, A., and Kuckuck, B. (2023). An overview on deep learning-based approximation methods for partial differential equations. *Discrete & Continuous Dynamical Systems-Series B*, 28(6).

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

[Bishop, 1995] Bishop, C. M. (1995). *Neural networks for pattern recognition*. Clarendon Press, Oxford.

[Chen et al., 2018] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[Cortez et al., 2009] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47(4):547–553.

[Cuomo et al., 2022] Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. (2022). Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88.

[Efron et al., 2004] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of Statistics*, 32(2):407–499.

[Fukushima, 1975] Fukushima, K. (1975). Cognitron: A self-organizing multilayer neural network. *Biological Cybernetics*, 20:121–136.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[Greene and Hensher, 2011] Greene, W. H. and Hensher, D. (2011). Multinomial logit and discrete choice models. In *Econometric Analysis*. Prentice Hall, 7 edition. Data table F18-2 from online complements.

[Hastie et al., 2001] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

[Hornik et al., 1990] Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551–560.

[Kidger, 2022] Kidger, P. (2022). On neural differential equations. *CoRR*, abs/2202.02435.

[Kingma and Ba, 2015] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

[Kis, 2021] Kis, T. (2021). Numerical differentiation of data (derivative). Technical report, Mathworks. https://github.com/tamaskis/derivative-MATLAB/blob/main/Basic_Numerical_Calculus.pdf, retrieved 1st May 2024.

[Kukacka et al., 2017] Kukacka, J., Golkov, V., and Cremers, D. (2017). Regularization for deep learning: A taxonomy. *CoRR*, abs/1710.10686.

[Lee and Kang, 1990] Lee, H. and Kang, I. S. (1990). Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131.

[Malek and Beidokhti, 2006] Malek, A. and Beidokhti, R. S. (2006). Numerical solution for high order differential equations using a hybrid neural network—optimization method. *Applied Mathematics and Computation*, 183(1):260–271.

[Parisi et al., 2003] Parisi, D. R., Mariani, M. C., and Laborde, M. A. (2003). Solving differential equations with unsupervised neural networks. *Chemical Engineering and Processing: Process Intensification*, 42(8-9):715–721.

[Paszyński et al., 2021] Paszyński, M., Grzeszczuk, R., Pardo, D., and Demkowicz, L. (2021). Deep learning driven self-adaptive hp finite element method. In *International Conference on Computational Science*, pages 114–121. Springer.

[Plaut et al., 1986] Plaut, D. C., Nowlan, S. J., and Hinton, G. E. (1986). Experiments on learning by back propagation. Technical report, Carnegie-Mellon University.

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

[Rennie, 2003] Rennie, J. (2003). On l2-norm regularization and the Gaussian prior. Technical report, MIT.

[Rice, 2007] Rice, J. A. (2007). *Mathematical Statistics and Data Analysis*. Breast cancer data used in Owen's empirical likelihood. Available from: http://www.cengage.com/statistics/discipline_content/dataLibrary.html.

[Rohde et al., 2012] Rohde, U. L., Jain, G. C., Poddar, A. K., and Ghosh, A. K. (2012). *Introduction to Differential Calculus*. Wiley.

[Samaniego et al., 2020] Samaniego, E., Anitescu, C., Goswami, S., Nguyen-Thanh, V. M., Guo, H., Hamdia, K., Zhuang, X., and Rabczuk, T. (2020). An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790.

[Simard et al., 1991] Simard, P. Y., Victorri, B., LeCun, Y., and Denker, J. S. (1991). Tangent prop - A formalism for specifying selected invariances in an adaptive network. In Moody, J. E., Hanson, S. J., and Lippmann, R., editors, *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 895–903. Morgan Kaufmann.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[Taunk et al., 2019] Taunk, K., De, S., Verma, S., and Swetapadma, A. (2019). A brief review of nearest neighbor algorithm for learning and classification. In *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pages 1255–1260.

[The American National Election Studies, 1996] The American National Election Studies (1996). Anes 1996 time series study. Technical report, University of Michigan, Center for Political Studies, Ann Arbor, MI.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288.

[Tikhonov, 1943] Tikhonov, A. N. (1943). On the stability of inverse problems. *Proceedings of the USSR Academy of Sciences*, 39:195–198.

[Tzen and Raginsky, 2019] Tzen, B. and Raginsky, M. (2019). Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*.

[Williams, 1995] Williams, P. M. (1995). Bayesian regularization and pruning using a laplace prior. *Neural computation*, 7(1):117–143.

DATASET	METHOD	MSE_{train}	σ_{train}	MSE_{val}	σ_{val}	ep	t
ANES96	STD	0.177 33	0.019 52	0.600 09	0.112 39	15	2.6
	L_2	0.307 81	0.023 52	0.574 75	0.092 36	41	7.7
	DO	0.492 99	0.032 60	0.581 50	0.138 98	53	2.6
	DL_{RND}	0.198 96	0.015 40	0.563 51	0.119 46	12	5.5
	DL_{NN}	0.429 54	0.039 32	**0.563 24**	0.094 68	53	11.8
CANCER	STD	0.292 84	0.013 98	0.247 64	0.244 36	17	2.2
	L_2	0.307 78	0.031 25	**0.216 62**	0.212 21	98	2.2
	DO	0.294 50	0.016 44	0.233 69	0.208 39	105	7.8
	DL_{RND}	0.292 08	0.021 32	0.224 74	0.183 97	84	3.9
	DL_{NN}	0.309 94	0.031 89	0.218 58	0.168 65	77	3.7
DIABETES	STD	0.336 35	0.050 41	0.506 46	0.179 58	113	2.3
	L_2	0.143 09	0.043 30	0.457 52	0.116 61	43	5.3
	DO	0.384 24	0.015 92	0.476 32	0.131 76	133	13.7
	DL_{RND}	0.117 45	0.024 34	0.456 37	0.176 95	32	4.3
	DL_{NN}	0.008 93	0.005 29	**0.455 37**	0.231 25	23	5.9
MODECHOICE	STD	0.117 95	0.018 04	0.639 44	0.144 80	131	2.6
	L_2	0.149 03	0.037 07	0.588 50	0.055 71	179	7.5
	DO	0.441 51	0.017 33	0.538 32	0.071 32	186	15.4
	DL_{RND}	0.130 25	0.013 08	0.594 05	0.086 26	99	9.1
	DL_{NN}	0.141 56	0.014 16	**0.536 03**	0.168 74	147	8.8
WINE	STD	0.097 47	0.010 40	0.595 75	0.118 23	17	3.1
	L_2	0.087 25	0.005 77	0.554 72	0.139 84	31	43.3
	DO	0.441 90	0.015 09	0.591 67	0.169 77	117	93.7
	DL_{RND}	0.477 00	0.029 36	**0.549 62**	0.233 04	167	15.1
	DL_{NN}	0.115 77	0.009 38	0.568 62	0.135 15	36	16.3
F1	STD	0.001 44	0.000 23	0.008 54	0.086 97	241	3.7
	L_2	0.001 37	0.000 13	**0.006 30**	0.081 55	248	45.8
	DO	0.011 65	0.000 39	0.006 73	0.084 39	242	3.5
	DL_{RND}	0.001 39	0.000 33	0.006 88	0.056 19	245	10.2
	DL_{NN}	0.001 37	0.000 14	0.006 56	0.000 46	250	23.2
REGRESSION1	STD	$7.140\,00 \cdot 10^{-6}$	$4.030\,00 \cdot 10^{-6}$	$7.640\,00 \cdot 10^{-6}$	0.022 37	250	3.6
	L_2	$4.930\,00 \cdot 10^{-6}$	$2.300\,00 \cdot 10^{-6}$	$4.840\,00 \cdot 10^{-6}$	0.074 53	250	44.6
	DO	0.002 75	0.000 15	$1.459\,00 \cdot 10^{-5}$	0.034 45	217	3.5
	DL_{RND}	$5.260\,00 \cdot 10^{-6}$	$2.930\,00 \cdot 10^{-6}$	$5.860\,00 \cdot 10^{-6}$	0.031 73	249	10.1
	DL_{NN}	$4.150\,00 \cdot 10^{-6}$	$2.660\,00 \cdot 10^{-6}$	$\mathbf{4.190\,00 \cdot 10^{-6}}$	0.084 70	250	23.8
REGRESSION10	STD	0.000 13	$3.658\,00 \cdot 10^{-5}$	0.000 43	0.133 15	250	3.7
	L_2	$2.228\,00 \cdot 10^{-5}$	$8.800\,00 \cdot 10^{-7}$	$\mathbf{2.652\,00 \cdot 10^{-5}}$	0.078 47	250	57.7
	DO	0.004 52	0.000 22	0.000 18	0.137 55	235	3.5
	DL_{RND}	0.000 14	$2.240\,00 \cdot 10^{-5}$	0.000 39	0.087 68	249	24.4
	DL_{NN}	0.000 13	$2.350\,00 \cdot 10^{-5}$	0.000 38	$9.917\,00 \cdot 10^{-5}$	250	9.7
SPARSE UNCORR	STD	0.055 14	0.003 02	0.085 19	0.061 31	174	3.7
	L_2	0.024 00	0.001 72	0.086 20	0.099 19	42	3.8
	DO	0.171 09	0.004 05	**0.079 39**	0.049 38	197	3.5
	DL_{RND}	0.038 10	0.002 56	0.082 94	0.149 12	45	10.0
	DL_{NN}	0.024 00	0.002 06	0.082 09	0.075 35	44	23.8
SWISS ROLL	STD	0.000 17	$3.736\,00 \cdot 10^{-5}$	0.000 23	0.107 06	249	3.7
	L_2	0.000 13	$4.676\,00 \cdot 10^{-5}$	0.000 17	0.051 57	248	39.8
	DO	0.012 37	0.000 52	0.001 09	0.039 87	210	3.5
	DL_{RND}	0.000 13	$2.252\,00 \cdot 10^{-5}$	0.000 20	0.034 57	245	23.8
	DL_{NN}	0.000 12	$2.436\,00 \cdot 10^{-5}$	**0.000 17**	$2.355\,00 \cdot 10^{-5}$	239	9.5

TABLE V: Experimental training and validation results. All values are the best in the grid search, averaged over 5-fold cross validation with 250 epochs of training. The neural networks have multiple inputs, matching the dataset and a single output. Networks have a single hidden layer with 64 neurons and use ReLU activation function in all neurons. We compare STD (no regularization), and L_2 and DO regularization with DL_{RND} ($DLoss$ with random pair selection) and DL_{NN} ($DLoss$ with nearest neighbor pair selection). The best results per datasets are highlighted in bold.