Numer. Math. Theor. Meth. Appl.
doi: 10.4208/nmtma.OA-2021-0062

Vol. **14**, No. 4, pp. 839-861
November 2021

Nonlinear Level Set Learning for Function Approximation on Sparse Data with Applications to Parametric Differential Equations

Anthony Gruber[1,*], Max Gunzburger[1], Lili Ju[2], Yuankai Teng[2] and Zhu Wang[2]

[1] *Department of Scientific Computing, Florida State University, 400 Dirac Science Library, Tallahassee, FL 32306, USA*
[2] *Department of Mathematics, University of South Carolina, 1523 Greene Street, Columbia, SC 29208, USA*

Received 23 April 2021; Accepted (in revised version) 6 August 2021

Abstract. A dimension reduction method based on the "Nonlinear Level set Learning" (NLL) approach is presented for the pointwise prediction of functions which have been sparsely sampled. Leveraging geometric information provided by the Implicit Function Theorem, the proposed algorithm effectively reduces the input dimension to the theoretical lower bound with minor accuracy loss, providing a one-dimensional representation of the function which can be used for regression and sensitivity analysis. Experiments and applications are presented which compare this modified NLL with the original NLL and the Active Subspaces (AS) method. While accommodating sparse input data, the proposed algorithm is shown to train quickly and provide a much more accurate and informative reduction than either AS or the original NLL on two example functions with high-dimensional domains, as well as two state-dependent quantities depending on the solutions to parametric differential equations.

AMS subject classifications: 65D15, 65D40

Key words: Nonlinear level set learning, function approximation, sparse data, nonlinear dimensionality reduction.

1. Introduction

It is frequently the case that scientists or engineers need to draw conclusions about the output of a function based on limited or incomplete data. Such situations arise, for example, when the output depends on the solution of expensive differential equations,

*Corresponding author. *Email addresses:* agruber@fsu.edu (A. Gruber), mgunzburger@fsu.edu (M. Gunzburger), ju@math.sc.edu (L. Ju), yteng@email.sc.edu (Y. Teng), wangzhu@math.sc.edu (Z. Wang)

or when lack of time and resources precludes the collection of sufficient high-quality samples. When this occurs, it becomes critical to maximize the value of the limited resources at hand, which requires informed algorithms for dimension reduction.

More specifically, let $U \subset \mathbb{R}^n$ be a bounded domain and consider the problem of approximating a continuously differentiable scalar function $f : U \to \mathbb{R}$ based on some predefined samples $\{\mathbf{x}^s, f(\mathbf{x}^s), \nabla f(\mathbf{x}^s)\}_{s\in S}$ of the function and its gradient vector field. Note that f may represent either a scalar quantity or some component of a vector quantity, so that no generality is lost with this consideration. Additionally, let $\rho : \mathbb{R}^n \to \mathbb{R}^+$ be a probability density function supported on U such that f is square-integrable with respect to ρ, i.e.

$$\|f\|_2^2 := \int_U f(\mathbf{x})^2 \rho(\mathbf{x})\, dx < \infty.$$

To generate a pointwise approximation to f, it is reasonable to seek a function $\tilde{f} : U \to \mathbb{R}$ which satisfies the minimization condition

$$\tilde{f}(\mathbf{x}) \in \operatorname*{arg\,min}_{g\in C^1(U)} \big\|f(\mathbf{x}) - g(\mathbf{x})\big\|_2^2. \tag{1.1}$$

However, if the dimension n is large relative to the number $|S|$ of available samples (i.e. sparse data), training a regression model to approximate f directly becomes infeasible. Indeed, unless the training data itself has a hidden low-dimensional structure, unsupervised learning methods such as feed-forward neural networks are prone to overfitting, leading to poor accuracy on new data as a result of inadequate generalizability. Therefore, it is necessary to employ some kind of dimension reduction to increase the density of the sampling data to the point where it is useful for approximating solutions to (1.1).

A prototypical example of this issue arises when studying the numerical solutions of differential equations with limited computational budget. Let I be a multi-index, $\boldsymbol{\beta} \in \mathbb{R}^m$, and consider a k-th order parameterized system of $R \in \mathbb{N}$ partial differential equations (PDE) for the function $\mathbf{u} : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^l$,

$$F^r\left(\boldsymbol{\beta}, \mathbf{x}, \mathbf{u}, \frac{\partial^{|I|}\mathbf{u}}{\partial x^I}\right) = 0, \quad 1 \le r \le R, \quad 1 \le |I| \le k, \tag{1.2}$$

which may depend on some number of initial or boundary conditions. Suppose additionally that the assignment $\boldsymbol{\beta} \mapsto \mathbf{u}(\boldsymbol{\beta}, \mathbf{x})$ is unique, so that solutions to (1.2) are parameterized by the variables $\boldsymbol{\beta}$. For prediction and sensitivity analysis it is often necessary to compute the value of some functional $\mathcal{K}(\mathbf{u})$ on PDE solutions $\mathbf{u} \in C^k(\mathbb{R}^m \times \mathbb{R}^n; \mathbb{R}^l)$ (e.g. temperature or total kinetic energy) which is implicitly a function of the parameters $\boldsymbol{\beta}$, i.e. $\mathcal{K}(\boldsymbol{\beta}) = \mathcal{K}(\mathbf{u}(\boldsymbol{\beta}, \mathbf{x}))$. On the other hand, it is usually not feasible to simulate the (potentially expensive) system (1.2) for every parameter configuration desired, so it is necessary to have a reasonable yet inexpensive approximation to $\mathcal{K}$ which can be computed for any $\boldsymbol{\beta}$ in place of numerically solving (1.2). In the language of before, this means finding $\tilde{\mathcal{K}}$ satisfying

$$\tilde{\mathcal{K}}(\boldsymbol{\beta}) \in \operatorname*{arg\,min}_{\mathcal{G}:\mathbb{R}^m\to\mathbb{R}} \big\|\mathcal{K}(\boldsymbol{\beta}) - \mathcal{G}(\boldsymbol{\beta})\big\|_2^2.$$

This poses difficulty when the set of configurations β for which solutions are available is sparse in $\mathbb{R}^m$, as occurs when solutions to (1.2) require hours (or even days) on a supercomputer to obtain. On the other hand, Section 5.3 shows that dimension reduction can be used to produce certain low-dimensional approximations to functionals like $\mathcal{K}$ which reliably approximate the true values.

The state-of-the-art strategies for dimension reduction can be loosely categorized as either intrinsic or extrinsic based on how they organize the available data. Intrinsic methods look for patterns directly within the input samples, which can be exploited for lower-dimensional classification and clustering once revealed. At the present time, there are several effective dimension-reduction methods which operate intrinsically, including Isomap, Locally Linear Embedding, and others (see e.g. [3, 4, 7, 35] and references therein). Moreover, these methods have demonstrated good performance on low-dimensional encoding problems as well as the recovery of geodesic distances along the data manifold (see e.g. [9, 28, 42]). On the other hand, intrinsic methods are of no use when the data is unstructured, since there is no low-dimensional structure to be found. This motivates the search for extrinsic dimension-reduction methods, which do not assume the given data possesses any structure at all. Instead, these methods take advantage of the structure which is inherited from an external object acting on the input space. For example, consider a surjective C^1 mapping onto a low-dimensional target, which stratifies the source according to its level sets. If this induced level set structure can be learned, then this notion can be used to produce a low-dimensional approximation to the original function. The advantage of such extrinsic methods is that they are applicable to any kind of sampling, including sparsity patterns which are indistinguishable from random noise. Conversely, these techniques are necessarily dependent on the object the source data inherits from, meaning there can be no singular solution which applies to every notion of inherited structure.

Remark 1.1. We use the notation $f'(\mathbf{x})$ to denote the derivative of f, i.e. the linear map induced by f satisfying $f'(\mathbf{x})\mathbf{v} = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle$ for all vectors $\mathbf{v}$. Similarly, if $\{\mathbf{e}_1, \dots, \mathbf{e}_n\}$ denote the standard basis for $\mathbb{R}^n$ then we let $f_i := f'(\mathbf{x})\mathbf{e}_i = \partial f(\mathbf{x})/\partial x^i$ denote the derivative of f with respect to $\mathbf{e}_i$. The extension of this notation to vector-valued functions is straightforward.

Thankfully, in the case of C^1 functions there is a good deal of information provided by classical theory which can be exploited for algorithm design. Given a regular value $y_0 \in \mathbb{R}$, the Implicit Function Theorem (IFT) guarantees that the level set $f^{-1}(y_0) \subset \mathbb{R}^n$ is a differentiable submanifold of $\mathbb{R}^n$, and around any preimage $\mathbf{x}_0 \in f^{-1}(y_0)$ there is the representation

$$T_{\mathbf{x}_0} f^{-1}(y_0) = \ker f'(\mathbf{x}_0),$$

which characterizes the local change in the function f in terms of tangent vectors to the level set $f^{-1}(y_0)$ at the point $\mathbf{x}_0$. In particular, since $T_{\mathbf{x}_0} f^{-1}(y_0) \subset \mathbb{R}^n$ is a linear subspace of (the tangent space to) $\mathbb{R}^n$ and $\ker f'(\mathbf{x}_0)$ has dimension $n - 1$, it follows that the local dimension of f is just one at almost every point in its domain. Moreover,

there is only one direction in U, the direction of the gradient $\nabla f(\mathbf{x}_0)$, in which $\mathbf{x}_0$ can move to produce any change in f whatsoever. This fact shows that the local structure of surjective C^1 functions is actually quite rigid, and there is the potential for constructing a (locally) one-dimensional representation of any such mapping despite the size of its domain.

The present work makes steps toward this idea by building on the "Nonlinear Level set Learning" (NLL) algorithm from [41], which is a neural network procedure for extrinsic dimension reduction in regression applications. In particular, it is shown that the performance of this algorithm can be improved dramatically by incorporating the information provided by the IFT, which leads to less computational expense, faster training, and more accurate regression results. To accomplish this, the NLL algorithm is recast as the minimization of a Dirichlet-type energy functional whose minimizers include mappings $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{g}(\mathbf{x}) = \big(g_1(\mathbf{x}) \dots g_n(\mathbf{x})\big)$ which send the high-dimensional inputs to simple slices $\{\mathbf{g}(\mathbf{x}) \,|\, g_1 = c \in \mathbb{R}\}$ where the function f is constant, and which can be reliably approximated with neural network algorithms. Moreover, once a minimizer $\mathbf{g}$ has been computed, dimension reduction becomes a matter of simple truncation, and a one-dimensional regression can be performed to recover a model $\hat{f} : \mathbb{R} \to \mathbb{R}$ which predicts f at any point in the original data space, i.e. $\hat{f} \circ g_1 \approx f$. Experiments are provided which demonstrate the improved performance of this version of NLL over the original algorithm and over the state-of-the-art linear dimension reduction method Active Subspaces.

The remainder of the work is structured as follows: Section 2 discusses related extrinsic dimension reduction methods and regression techniques, Section 3 details the NLL algorithm and its original formulation, Section 4 discusses the mentioned improvements to NLL and its connection to Dirichlet energy, and Section 5 exhibits numerical experiments which compare the present version of NLL to the original algorithm and Active Subspaces on a variety of test cases. Some concluding remarks are finally drawn in Section 6.

2. Related work

It is well known that the regression problem (1.1) is difficult in the presence of sparse data, see e.g. [20,22] and references. As such, there are a plethora of techniques which have been developed to mitigate the high response variability that is inevitable in this situation. Ranging from optimal sampling (e.g. [16, 36]) to central subspace methods (e.g. [2, 29]), all such works share the common themes of trying to increase predictive power and decrease model overfitting. Extrinsic methods for dimension reduction in pointwise regression applications (e.g. [5, 12, 41]) operate in the same way, although more regularity is usually assumed since a stronger type of convergence is discussed.

The dimension reduction approach most similar to ours is known as approximation by ridge functions [10, 14, 34]. In particular, consider a linear projection $P : \mathbb{R}^n \to \mathbb{R}^k$ where $k \ll n$ and a (not necessarily differentiable) function $\hat{f} : \mathbb{R}^k \to \mathbb{R}$. If the

functions $\hat{f}, P$ satisfy

$$f(\mathbf{x}) = \hat{f}(P\mathbf{x})$$

for all $\mathbf{x} \in U$, then f is called a (generalized) ridge function (c.f. [14]). Clearly, this definition implies that f is constant on the kernel of P.

Using ridge functions for dimension reduction typically involves optimizing over the functions $\hat{f}, P$. In particular, suppose the projection P is given and consider computing a $\hat{f}$ which satisfies

$$\hat{f}(\mathbf{x}) \in \underset{g:\mathbb{R}^k \to \mathbb{R}}{\arg\min} \left\| f(\mathbf{x}) - g(P\mathbf{x}) \right\|_2^2. \tag{2.1}$$

If f is nearly constant on the kernel of P, then $\hat{f}$ will be a reasonable pointwise approximation to the original function. Moreover, the minimization in (2.1) is usually much more feasible than the one in (1.1) when the input data is sparse, as the projection $\mathbf{x}^s \mapsto P\mathbf{x}^s$ naturally increases its density.

Of course, to compute a useful solution to (2.1) it is necessary to have a projection mapping which adequately captures the change in f. This usually involves constructing a low-dimensional "response surface" containing information about the dependence of f on its independent variables (see e.g. [2, 14, 25, 29] and their references), and frequently employs techniques including principal component analysis, projection pursuit regression, kriging, and others [1, 13, 15, 18, 31–33, 39, 40]. One popular algorithm for response surface construction among C^1 functions is known as Active Subspaces (AS) [12]: a procedure for determining the affine subspace of $U \subset \mathbb{R}^n$ where f changes the most on average. In particular, AS computes a Monte Carlo approximation to the covariance matrix $\mathbf{C} := \mathbb{E}[\nabla f (\nabla f)^T]$, so that the eigenvalue decomposition $\mathbf{C} =: \mathbf{W}\mathbf{\Lambda}\mathbf{W}^T$ gives a global linear transformation of the input coordinates in terms of how much they affect the value of f. In the case that only the first k eigenvalues are significant, this yields a suitable low-dimensional projection $P : \mathbb{R}^n \to \mathbb{R}^k$ in terms of the first k columns $\mathbf{W}_A$ of $\mathbf{W}$. More precisely, decomposing $\mathbf{W} = [\mathbf{W}_A \ \mathbf{W}_I]$ in terms of its "active" and "inactive" components, it can be shown that

$$\left\| f(\mathbf{x}) - \hat{f}\left(\mathbf{W}_A^T \mathbf{x}\right) \right\|_2^2 \leq C \left\| \mathbf{W}_I \right\|_2^2,$$

where $\hat{f}$ is a suitably chosen function (see e.g. [14, Theorem 2]) and $C = C(\rho)$ is a Poincaré constant depending on the density ρ. Due to this fact and others, AS projections are known to be quite useful for dimension reduction, regression, and sensitivity analysis (see e.g. [12, 17, 27, 30]).

On the other hand, it is frequently the case that the eigenvalues of the covariance matrix $\mathbf{C}$ decay quite slowly, making a linear AS reduction ineffective for low-dimensional regression (c.f. Section 5.2). This has motivated another line of work into nonlinear methods for extrinsic dimension reduction. One such idea was introduced as the Active Manifolds (AM) algorithm [5], which takes advantage of the local decomposition of f afforded by the IFT. In particular, AM aims to construct an integral curve

$t \mapsto \mathbf{x}(t) \subset \mathbb{R}^n$ of the (normalized) gradient field, i.e. a solution to

$$\dot{\mathbf{x}} = \frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}, \quad \mathbf{x}(0) = \mathbf{x}_0.$$

Once this "active manifold" has been constructed, it is necessarily the case that the entire range of f on the set of level sets intersecting $\mathbf{x}(t)$ is represented simply by the values $f(\mathbf{x}(t))$, i.e. for every $\mathbf{y} \in U$ such that $f(\mathbf{y}) = c$ and $f^{-1}(c) \cap \mathbf{x}(t) \neq \varnothing$ there is a value t_0 such that $c = f(\mathbf{y}) = f(\mathbf{x}(t_0))$. Therefore, to determine the value of f at any suitable point in the input space it is sufficient to know the values of f along the 1-D curve $\mathbf{x}(t)$ as well as a projection map $\pi : \mathbb{R}^n \to \mathbb{R}$, $\pi(\mathbf{y}) = \{t \,|\, f(\mathbf{y}) = f(\mathbf{x}(t))\}$. This algorithm has the benefit of zero intrinsic error (since there is no averaging involved), but comes with the significant challenge of computing the necessary projection map.

However, recent work has also shown that sufficiently deep neural networks have the ability to reproduce arbitrary measurable functions (see e.g. [24]), motivating another line of research into dimension reduction. Most network-based intrinsic methods to date are based on constructing an autoencoder-decoder network (see e.g. [23, 38]) which learns projection and expansion functions that compose to yield an approximation to the identity mapping on the input space. The extrinsic methods which employ neural networks are more various, including the DrLIM method in [21] which computes an invariant nonlinear function mapping the input data evenly to a low-dimensional space, or the method of DIPnets (see [33]) which uses AS in conjunction with projected neural networks for increased generalizability.

3. The NLL algorithm

In contrast to the work previously mentioned, the present approach is based on a neural network algorithm introduced in [41] called "Nonlinear Level set Learning" (NLL). At its core, NLL uses neural network techniques to extend the idea of AS to more general transformations of the input data. In particular, consider computing a diffeomorphism (differentiable bijection with differentiable inverse) $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^n$, $\mathbf{z} = \mathbf{g}(\mathbf{x})$ and $\mathbf{h} \circ \mathbf{g} = \mathbf{I}_{\mathbb{R}^n}$, which separates the domain of the push-forward function $f \circ \mathbf{h}$ into global pairs $\mathbf{z} = (\mathbf{z}_A, \mathbf{z}_I)$ of "active" and "inactive" coordinates. If $\mathbf{g}, \mathbf{h}$ can be constructed such that the sensitivity of $f \circ \mathbf{h}$ to the coordinates $\mathbf{z}_I$ is sufficiently low, then it is reasonable to conclude that for any inactive coordinate $z^i \in \mathbf{z}_I$ the domain of $f \circ \mathbf{h}$ can be restricted to $\mathrm{Span}\{z^i\}^\perp$ with negligible impact on the function value. Provided this condition is satisfied, regression can be applied to obtain a lower-dimensional mapping $\hat{f} : \mathbb{R}^{|A|} \to \mathbb{R}$ such that $f(\mathbf{x}) \approx \hat{f}(\mathbf{z}_A)$. More precisely, given the function $\mathbf{g}$ and writing $\mathbf{z}_A =: \mathbf{g}_A(\mathbf{x})$ to denote its first $|A|$ components, this means computing a generalized ridge function

$$\hat{f}\big(\mathbf{g}_A(\mathbf{x})\big) \in \underset{\varphi:\mathbb{R}^{|A|}\to\mathbb{R}}{\arg\min} \big\|f(\mathbf{x}) - \varphi(\mathbf{g}_A(\mathbf{x}))\big\|_2^2, \tag{3.1}$$

where $\mathbf{g}$ acts as the (nonlinear) projection operator. Note that once $\mathbf{g}$ has been obtained, computing the required $\hat{f}$ in (3.1) is automatically a more feasible regression

problem than (1.1), since almost all of the variation in f is concentrated in the lower dimensional image $\mathbf{z}_A$. Most importantly, the necessary projection from $\mathbf{z}$ to $\mathbf{z}_A$ is simple and canonical: after truncating the domain of $\mathbf{h}$ by the span of the inactive variables $\mathbf{z}_I$, the active variables $\{\mathbf{z}_A\}$ parameterize the low-dimensional inputs by definition.

Remark 3.1. In the original NLL formulation [41], regression on $f \circ \mathbf{h}$ is still performed on the full set of inputs $\mathbf{g}(\mathbf{x})$ without dimension reduction. On the other hand, we find that this is not necessary when the mappings $\mathbf{g}, \mathbf{h}$ are well-trained. Therefore, the algorithm in Section 3 does not require the use of additional inputs beyond $\mathbf{z}_A := z^1$.

Of course, to make use of this idea it is necessary to have an efficient way to compute the diffeomorphism $\mathbf{g}$. The authors of [41] have demonstrated that carefully designed neural networks are well suited to this task, and a minimization procedure can be employed to obtain a mapping which attempts to concentrate the sensitivity of f in some predefined number of active directions. The particular network architecture and appropriate notion of loss which support this approach will now be discussed, as well as the present modifications which improve the overall efficacy of the method.

3.1. Network architecture

In [41] as well as presently, the constrained minimization for $\mathbf{g}$ and its inverse $\mathbf{h}$ is accomplished using a specialized RevNet architecture [8, 19] based on the Verlet discretization of Hamiltonian systems. RevNets are neural networks which are reversible by construction, yielding improved memory efficiency as intermediate layer activations do not have to be stored. The primary benefit of using RevNets inside the NLL algorithm is their connection to Hamiltonian systems, whose solution curves are generated by local diffeomorphisms of the source domain. Because of this, propagating the input data through a RevNet structure (with a sufficiently small step-size) will necessarily give a configuration which is diffeomorphic to the original, removing the need for an explicit constraint during the minimization of the loss functional.

More precisely, let $(\mathbf{u}, \mathbf{v})$ be a channel-wise partition of the inputs, $\boldsymbol{\sigma}$ be an activation function, and let $\mathbf{K}_i$ respectively $\mathbf{b}_i$ denote operator respectively vector valued functions for $i = 1, 2$. It is shown in [8] that the dynamical system

$$\begin{aligned} \dot{\mathbf{u}}(t) &= \mathbf{K}_1^T(t)\boldsymbol{\sigma}\big(\mathbf{K}_1(t)\mathbf{v}(t) + \mathbf{b}_1(t)\big), \\ \dot{\mathbf{v}}(t) &= -\mathbf{K}_2^T(t)\boldsymbol{\sigma}\big(\mathbf{K}_2(t)\mathbf{u}(t) + \mathbf{b}_2(t)\big) \end{aligned} \tag{3.2}$$

is stable and well-posed, hence usable for forward propagation along a neural network. Defining $\mathbf{x} =: (\mathbf{u}_0, \mathbf{v}_0)$ and discretizing (3.2) with layers $1 \le l \le L$ then yields the system

$$\begin{aligned} \mathbf{u}_{l+1} &= \mathbf{u}_l + \tau\, \mathbf{K}_{l,1}^T\, \boldsymbol{\sigma}\big(\mathbf{K}_{l,1}\mathbf{v}_l + \mathbf{b}_{l,1}\big), \\ \mathbf{v}_{l+1} &= \mathbf{v}_l - \tau\, \mathbf{K}_{l,2}^T\, \boldsymbol{\sigma}\big(\mathbf{K}_{l,2}\mathbf{u}_{l+1} + \mathbf{b}_{l,2}\big), \end{aligned}$$

where $\tau \in \mathbb{R}$ is the discrete time step, and $\mathbf{K}_{l,i}$ respectively $\mathbf{b}_{l,i}$ are the weight matrices respectively biases at layer l. It is easily checked that this mapping is invertible at each layer, therefore it is reasonable to define $\mathbf{z} := (\mathbf{u}_L, \mathbf{v}_L)$. In practice, the partition $(\mathbf{u}_0, \mathbf{v}_0)$ for $\mathbf{x}$ is chosen so that $\mathbf{u}_0$ contains the first $\lceil n/2 \rceil$ variables x^i and $\mathbf{v}_0$ contains the remainder. Propagation forward and backward through this scheme then yields mappings $\mathbf{g}, \mathbf{h}$ which satisfy the desired diffeomorphism condition by construction, allowing for an unconstrained minimization of the loss functional. The next goal is to discuss the particular loss criterion which is used to update the $\mathbf{K}_{l,i}, \mathbf{b}_{l,i}$ parameters.

3.2. The loss functional from [41]

It remains to discuss the criterion by which the mappings $\mathbf{g}, \mathbf{h}$ are trained. The key observation to the approach in [41] is that if $z^i \in \mathbf{z}_I$ is an inactive coordinate, then the gradient vector field $\nabla f(\mathbf{x})$ is orthogonal to the derivative of $\mathbf{h}(\mathbf{z})$ with respect to z^i at any point $\mathbf{x} = \mathbf{h}(\mathbf{z})$ in the input space. Said differently, this means that the derivative vector $\mathbf{h}_i(\mathbf{z}) := \mathbf{h}'(\mathbf{z})\mathbf{e}_i$ is tangent to the level set of f at $\mathbf{x}$, hence lies in the kernel of $f'(\mathbf{x})$. Enforcing this condition during neural network training leads the authors of [41] to the minimization problem

$$\underset{\mathbf{h} \in \mathrm{Diff}(\mathbb{R}^n)}{\arg\min}\ \hat{L}(\mathbf{h}),$$

which involves the (regularized) loss functional

$$\hat{L}(\mathbf{h}) = \hat{L}_1 + \lambda \hat{L}_2 = \sum_{s \in S} \sum_{i=1}^{n} \omega_i \big\langle \mathbf{J}_i(\mathbf{z}^s), \nabla f(\mathbf{x}^s) \big\rangle^2 + \lambda \sum_{s \in S} \big(\det \mathbf{J}(\mathbf{z}^s) - 1\big)^2. \tag{3.3}$$

Here $\mathbf{J} = \big(\mathbf{J}_1 \dots \mathbf{J}_n\big) = \big(\mathbf{h}_1 / \|\mathbf{h}_1\| \dots \mathbf{h}_n / \|\mathbf{h}_n\|\big)$ is the column-normalized Jacobian matrix of the transformation, the $\omega_i \in [0, 1]$ are user-defined weights influencing the strength of the constraint in each dimension, and $\lambda \in [0, \infty)$ is a user-defined weight influencing the strength of the regularization term. Note that the choice of column-normalization in $\mathbf{J}$ is introduced in order to simplify the calculation of derivatives in the original implementation, which uses a finite difference scheme. Moreover, the regularization arises for similarly practical considerations, providing extra rigidity which helps direct the minimization toward a reasonable solution. On the other hand, these additional inclusions come at the cost of diminishing the geometric meaning of the procedure, which has a substantial effect on both the rate of training and the overall effectiveness of the algorithm (c.f. Section 5).

Remark 3.2. The informed reader will notice that the publicly available implementation of the NLL algorithm in [41] uses a slightly different loss functional than $\hat{L}$ defined in (3.3). In particular, the loss functional minimized there is

$$\tilde{L}(\mathbf{h}) := \frac{\sqrt{\hat{L}_1}}{|S|} + \lambda \prod_{s \in S} \big(\det \mathbf{J}(\mathbf{z}^s) - 1\big). \tag{3.4}$$

Because the practical behavior of the original algorithm seems to be strongly dependent on this choice, each of the comparisons to "Old NLL" in Section 5 uses whichever loss functional, (3.3) or (3.4), yields the best performance.

While NLL in its original form has shown promising performance in several cases (c.f. [41, Fig. 2]), it requires the specification many parameters $\omega_1, \ldots, \omega_n, \lambda$ whose significance is not clear and whose influence is not easily estimated a priori. Additionally, it is relatively slow-to-train and requires the use of an expensive regularization term in order to guarantee reasonable performance and stability on high-dimensional data. The goal of what follows is to describe theoretically-justified modifications to this algorithm which successfully eliminate these issues while also improving the overall efficacy of the dimension reduction. By reformulating the NLL procedure as the minimization of an appropriate energy functional, a discretization is found which achieves both faster training and more accurate results.

4. A modified NLL algorithm

The present algorithm augments NLL with the geometric knowledge afforded by the IFT. Consider using the same RevNet architecture to construct a bijective mapping $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^n$ such that the level sets of f are parameterized by the images of the inactive variables $\mathbf{z}_I = \{z^2, \ldots, z^n\}$, i.e. for each regular value $y \in \mathbb{R}$ there is a $c \in \mathbb{R}$ such that $f^{-1}(y) = \{\mathbf{h}(\mathbf{z}) \,|\, z^1 = c\}$. Since f is differentiable and scalar-valued, the IFT asserts that this can be done locally away from points where $\nabla f = \mathbf{0}$, but there is no guarantee that a global mapping exists unless $\nabla f \neq \mathbf{0}$ everywhere and each level set in the domain is diffeomorphic to an $(n-1)$-dimensional hyperplane. On the other hand, as with AS it is always reasonable to ask for a function $\mathbf{h}$ which parameterizes these sets "the most on average". Supposing $\mathbf{h}$ does this sufficiently well, the problem of approximating f on the full input space can be reduced to that of approximating $f \circ \mathbf{h}$ on the one-dimensional space spanned by the active variable z^1. In particular, an inexpensive regression approximation $\hat{f} : \mathbb{R} \to \mathbb{R}$ can be trained (e.g. simple neural network or local/global least-squares) using the information that $f(\mathbf{x}) \approx \hat{f}(g_1(\mathbf{x}))$.

Remark 4.1. If $\mathbf{v}_1, \ldots, \mathbf{v}_n$ form a frame of vector fields on an open set $U \in \mathbb{R}^n$, local coordinates $x^1, \ldots, x^n$ such that $\mathbf{v}_i = \partial/\partial x^i$ are called simultaneous flow-box coordinates on U [6]. Such coordinates exist around any point $\mathbf{x} \in U$ provided the vectors $\mathbf{v}_i(\mathbf{x})$ are linearly independent and the Lie bracket identities $[\mathbf{v}_i, \mathbf{v}_j] = 0$ hold on U for all $1 \leq i, j \leq n$. In accordance with this notion, the NLL mapping $\mathbf{h}$ can be understood as providing approximate, best-on-average flow-box coordinates for the level sets of f. Of course, it is easy to check that $[\mathbf{h}_i(\mathbf{z}), \mathbf{h}_j(\mathbf{z})] = \mathbf{h}'(\mathbf{z})[\mathbf{e}_i, \mathbf{e}_j] = \mathbf{0}$ for all $\mathbf{z} \in \mathbb{R}^n$.

4.1. A new loss functional

Computing the mapping $\mathbf{h}$ in this way requires a loss functional which reflects the meaning of the procedure. In practice, it suffices to note that if $f \circ \mathbf{h}$ is insensitive

to perturbations in $z^2, \dots, z^n$ at a point $\mathbf{z} = \mathbf{g}(\mathbf{x})$, then its derivative $(f \circ \mathbf{h})'(\mathbf{z})$ is identically zero on the span of the inactive basis vectors $\{\mathbf{e}_2, \dots, \mathbf{e}_n\}$. Note that this condition is readily expressed coordinate-wise as

$$(f \circ \mathbf{h})'(\mathbf{z})\mathbf{e}_i = \big\langle \nabla f(\mathbf{x}), \mathbf{h}_i(\mathbf{z}) \big\rangle = 0 \quad \text{for all} \quad i \neq 1,$$

yielding precisely the motivating statement for the original NLL algorithm: that the vectors $\mathbf{h}_2(\mathbf{z}), \dots, \mathbf{h}_n(\mathbf{z})$ are orthogonal to $\nabla f(\mathbf{x})$. Conversely, note that the mathematical meaning of $(f \circ \mathbf{h})'(\mathbf{z})\mathbf{e}_i$ as a rate of change is preserved only if there is no normalization of $\mathbf{h}_i(\mathbf{z})$. Therefore, given a set of training samples $\{\mathbf{x}^s, \nabla f(\mathbf{x}^s)\}_{s \in S}$ in the original input space and a RevNet architecture as before, the goal of our modified NLL algorithm is to compute $\mathbf{h} : \mathbb{R}^n \to \mathbb{R}^n$ which minimizes the loss functional

$$L(\mathbf{h}) = \frac{1}{|S|} \sum_{s \in S} \big\| (f \circ \mathbf{h})'(\mathbf{z}^s) \big\|_{\perp}^2, \tag{4.1}$$

where $\|\cdot\|_{\perp}^2$ denotes the (squared) norm on the subspace orthogonal to the active direction $\mathbf{e}_1$ at each point, i.e. the trace of the Euclidean inner product $\langle \cdot, \cdot \rangle$ with respect to the basis $\{\mathbf{e}_2, \dots, \mathbf{e}_n\}$. This encourages the algorithm to find a mapping $\mathbf{h}$ which reduces the composite function $f \circ \mathbf{h}$ to a function of one variable $z^1 = g_1(\mathbf{z})$. Moreover, since $\mathbf{h}$ is naturally constrained to be a diffeomorphism and hence a proper map, it follows that the summand is quadratic and strongly coercive (on $\mathrm{Span}\{\mathbf{e}_1\}^{\perp}$) whenever f is, making gradient descent based on L a feasible strategy. Note the similarity to the method of AS: while AS seeks a linear subspace of the input data $\{\mathbf{x}^s\}$ where the average change in f is maximized, the proposed algorithm seeks a linear subspace $\mathrm{span}\{\mathbf{e}_2, \dots, \mathbf{e}_n\}$ of the transformed data $\{\mathbf{z}^s\}$ where the average change in the push-forward $f \circ \mathbf{h}$ is minimized. It follows that the complementary subspace $\mathrm{span}\{\mathbf{e}_1\}$ maximizes this change, and the original nonlinear submanifold which maximizes the average change in f can be recovered through $\mathbf{x} = \mathbf{h}(\mathbf{z})$.

Remark 4.2. Observe the lack of regularization in (4.1). Experiments show (c.f. Section 5) that this criterion is sufficient to drive the descent to a minimum without additional penalty, suggesting the benefits of using un-normalized derivatives of the network mapping.

4.2. Connection to energy minimization

From a continuous perspective, it is meaningful to note that the L defined in (4.1) is (up to scale) a discretization of the Dirichlet-type energy functional

$$\mathcal{L}(\mathbf{h}) = \int_V \big\| (f \circ \mathbf{h})'(\mathbf{z}) \big\|_{\perp}^2 \, d\mu^n = \int_I \int_{Z_t} \big\| (f \circ \mathbf{h})'(\mathbf{z}) \big\|_{\perp}^2 \, d\mu^{n-1} \, dt, \tag{4.2}$$

where $\mathbf{h}(V) = U$, $I = \pi_1(V)$ is a bounded interval containing the range of z^1 and $Z_t = \{\mathbf{z} \in V \,|\, z^1 = t\}$. To examine the structure of potential minimizers, it is useful to

compute the variational derivative of $\mathcal{L}$. Recall that a variation of $\mathbf{h} : V \to U$ is a one-parameter family of mappings (also denoted $\mathbf{h}$) satisfying $\mathbf{h}(t) = \mathbf{h} + t\boldsymbol{\varphi}$ for all t in a compactly supported interval $t \in (-\varepsilon, \varepsilon)$ and some compactly supported $\boldsymbol{\varphi} \in C^1(V; U)$. The variational derivative of a functional $\mathcal{F}$ depending on $\mathbf{h}$ is then the first-order term in its Taylor expansion around $t = 0$. In particular, there is the notation

$$\delta\mathcal{F}(\mathbf{h})\boldsymbol{\varphi} := \frac{d}{dt}\mathcal{F}(\mathbf{h} + t\boldsymbol{\varphi})\big|_{t=0},$$

which denotes the variational derivative (or simply the variation) of $\mathcal{F}$ at the point $\mathbf{h}$ in the direction of $\boldsymbol{\varphi}$. It is a fundamental fact in the calculus of variations that $\mathcal{F}$ is stationary if and only if $\delta\mathcal{F}(\mathbf{h})\boldsymbol{\varphi} = 0$ for all $\boldsymbol{\varphi}$.

Specifically to the present case, note that the variation in $\mathbf{h}$ induces a variation in $f \circ \mathbf{h}$,

$$(f \circ \mathbf{h})(t) = (f \circ \mathbf{h}) + t(f \circ \boldsymbol{\varphi}),$$

so that the integrand of $\mathcal{L}$ varies as

$$\delta\big\|(f \circ \mathbf{h})'(\mathbf{z})\big\|_\perp^2 = 2\big\langle (f \circ \mathbf{h})'(\mathbf{z}), (f \circ \boldsymbol{\varphi})'(\mathbf{z})\big\rangle_\perp.$$

Moreover, since $\langle \cdot, \cdot \rangle_\perp$ is just the inner product induced from $\mathbb{R}^n$ on the slices Z_t and the variation $\boldsymbol{\varphi}$ vanishes on the boundary of each slice, integration by parts implies the equality

$$\int_{Z_t} \big\langle (f \circ \mathbf{h})'(\mathbf{z}), (f \circ \boldsymbol{\varphi})'(\mathbf{z})\big\rangle_\perp \, d\mu^{n-1} = -\int_{Z_t} (f \circ \boldsymbol{\varphi})(\mathbf{z})\Delta^\perp(f \circ \mathbf{h})(\mathbf{z}) \, d\mu^{n-1}.$$

Putting this together with the fact that spatial and variational derivatives commute in this setting, the variation of $\mathcal{L}$ is expressed as

$$\begin{aligned}\delta\mathcal{L}(\mathbf{h})\boldsymbol{\varphi} &= \int_I \int_{Z_t} \delta \left\|(f \circ \mathbf{h})'(\mathbf{z})\right\|_\perp^2 \, d\mu^{n-1} \, dt \\ &= -2\int_I \int_{Z_t} (f \circ \boldsymbol{\varphi})(\mathbf{z})\Delta^\perp(f \circ \mathbf{h})(\mathbf{z}) \, d\mu^{n-1} \, dt. \end{aligned} \tag{4.3}$$

Since $f \circ \boldsymbol{\varphi}$ is an arbitrary C^1 variation, this quantity (4.3) vanishes identically if and only if $\Delta^\perp(f \circ \mathbf{h}) = 0$ on V, i.e. if and only if $f \circ \mathbf{h}$ is harmonic on the slices Z_t. It is clear that this condition is satisfied when $\mathbf{h}$ parameterizes the level sets of f on U, since $f \circ \mathbf{h}$ is constant on each slice. Moreover, as minimization of Dirichlet-type energies such as $\mathcal{L}$ is known to exhibit good stability and convergence properties (see e.g. [37]), it is reasonable to expect that the minimization of (4.1) will be similarly well-behaved. The next section provides experimental justification for this idea.

5. Numerical examples

This section details examples of the present NLL algorithm "New NLL" and its improvements over both AS and the original NLL algorithm "Old NLL" on sparse data sets.

In particular, the effectiveness of each algorithm is measured using two metrics: the relative sensitivity of the function to the active variable(s), and the predictive ability of the low-dimensional approximation. After some discussion of implementation, two high-dimensional functions from [41] are used for validation of New NLL. Following this, New NLL is applied to the problem of predicting quantities of interest which depend on the solutions to systems of differential equations. In all cases, New NLL is shown to effectively reduce the dimension to one, providing a low-dimensional submanifold which affords accurate approximation of the desired function.

5.1. Implementation details

The relevant algorithms are implemented in Python on an early 2015 MacBook Pro with 2.7GHz Intel i5 processor and 8GB of RAM. The implementation of AS is done in Python 2.7 following P. Constantine's code library [11], while the implementation of the NLL algorithms is done in Python 3.9 using the PyTorch library 1.7. Note that the necessary derivatives of the network mapping $\mathbf{h}$ are computed using the PyTorch version of automatic differentiation, and not through finite differences as in [41]. Moreover, New NLL employs the Adaptive Moment Estimator (ADAM) optimizer during training [26] while Old NLL uses stochastic gradient descent (SGD). In both cases, the RevNet layer step size is fixed at $\tau = 0.25$, the activation function is $\sigma = \tanh$, and 500 samples $\{\mathbf{x}, f(\mathbf{x}), \nabla f(\mathbf{x})\}$ are used for validation as the models train. The amount of training samples is variable and reported in Tables 1 and 2. The weights for Old NLL are chosen as in [41], namely $\lambda = 1$, $\omega_i = 1$ for z^i an inactive variable and $\omega_i = 0$ otherwise. For visualization, the sensitivities of $f \circ \mathbf{h}$ (computed using gradient information) are reported as percentages relative to the sum over all coordinates, and the NLL training and validation losses are reported as percentages relative to their initial values.

After dimension reduction, low-dimensional regression approximations are generated using the original training data projected onto the computed low-dimensional space. To demonstrate that both traditional and modern regression methods can be used effectively after this dimension reduction, experiments on the functions f_5, R_0 below use local or global polynomial least-squares to approximate the function value while the experiments on f_4, K use a simple feed-forward neural network. An additional 10000 uniformly distributed data samples are used for testing the regression in all cases except for K, where 500 additional samples are used. Results of the low-dimensional regressions are visualized through plots of the projected testing data against both the true and approximate function values. The error metrics reported are relative root-mean-square error (RRMSE), relative ℓ_1 error ($R\ell_1$), and relative ℓ_2 error ($R\ell_2$), computed as

$$RRMSE(\hat{f}) = \frac{1}{\sqrt{|S|}} \left(\frac{\|f - \hat{f}\|_2}{\max f - \min f} \right), \quad R\ell_i(\hat{f}) = \frac{\|f - \hat{f}\|_i}{\|f\|_i}.$$

Note finally that in the case of AS the mapping $\mathbf{h}$ should be interpreted as the matrix

$\mathbf{W}^T$ from Section 2. The results for all simulations in this section are summarized in Tables 1 and 2.

5.2. Two high-dimensional examples

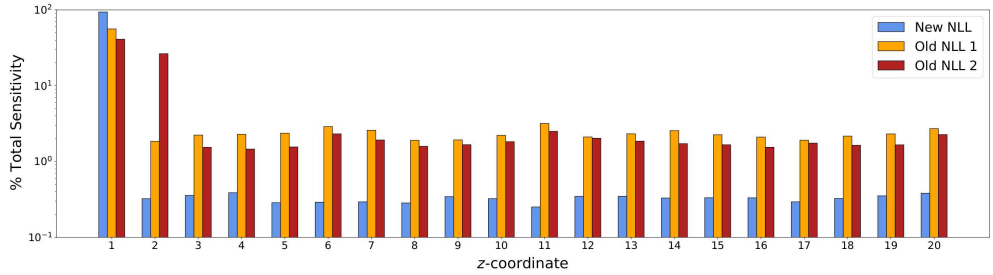
Consider the following functions from [41], numbered consistently with the source:

$$f_4(\mathbf{x}) = \sin\left(\|\mathbf{x}\|^2\right), \quad f_5(\mathbf{x}) = \prod_{i=1}^{20} \frac{1}{1+x_i^2}.$$

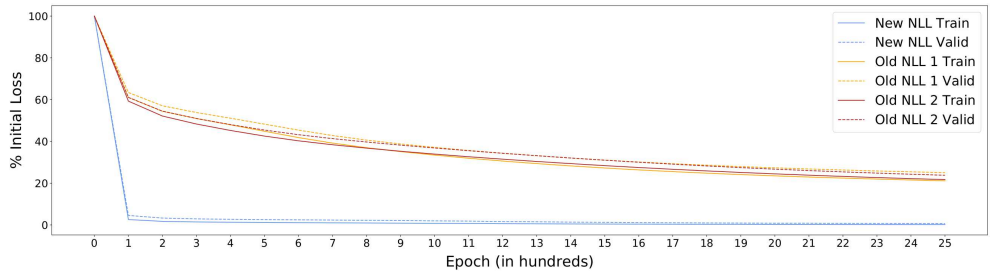
Remark 5.1. Note that the f_5 in [41] is actually $1.2^{40} f_5(1.2\mathbf{x})$ in terms of the present f_5, therefore identical up to a scale factor of 1.2^{40} and a dilation $\mathbf{x} \mapsto 1.2\mathbf{x}$.

First, the experiment from [41] on $f_5 : [0,1]^{20} \to \mathbb{R}$ is repeated, which uses a 30-layer RevNet. Here both Old NLL and New NLL are trained for 5000 epochs (passes forward and backward through the training data) with learning rates of 0.5 respectively 0.003, and regression is performed through local quadratic least-squares using the 10 nearest training neighbors of each test point. The results of New NLL are compared alongside simulations of Old NLL using one respectively two active variables, where Old NLL has been trained using the loss functional $\tilde{L}$ (c.f. Remark 3.2).

Fig. 1 shows that New NLL outperforms Old NLL in training speed (b) and sensitivity concentration (a). In particular, with 500 training data New NLL concentrates 94% of the total sensitivity in $f_5 \circ \mathbf{h}$ in the z^1 direction, while Old NLL can only achieve 56%,



(a)



(b)

Figure 1: (a) Relative sensitivity of $f_5 \circ \mathbf{h}$ to each $\mathbf{z}$-coordinate; (b) Relative value of loss during the first 2500 epochs of NLL training.

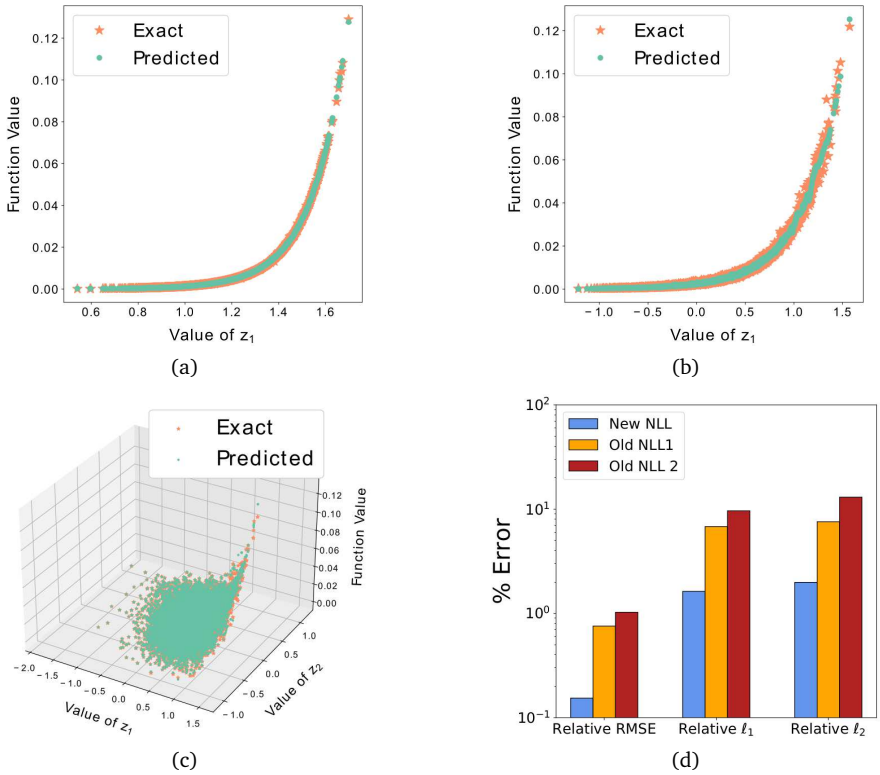


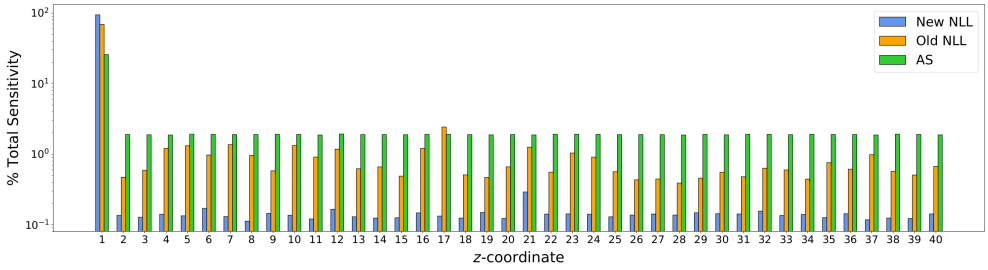
Figure 2: Regression and errors on $f_5 \circ \mathbf{h}$: (a) New NLL; (b) Old NLL 1; (c) Old NLL 2; (d) Errors of the three approaches.

or 67% of the total if two active variables are used. On the other hand, the regression results in Fig. 2 illustrate that low-dimensional approximations built using New NLL are also more accurate, as can be seen in both the regression errors as well as the tightness of the projected testing data around the fit curve. Indeed, when using New NLL 500 training samples is sufficient for relative errors around 3%. Interestingly, the inclusion of two active variables in Old NLL does not appear to improve the training rate nor the regression accuracy, despite concentrating more sensitivity in the active directions.

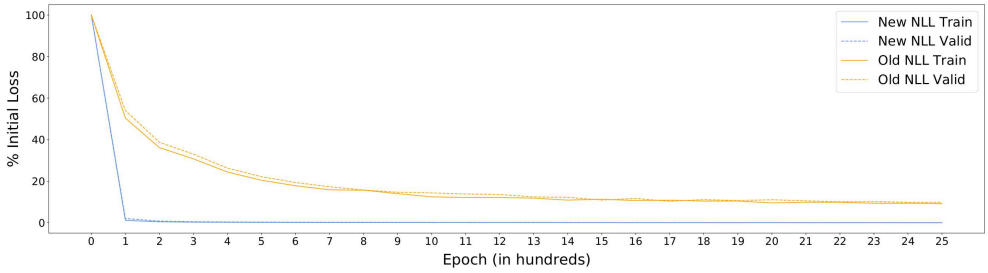
Next, performance is measured on the higher-dimensional function $f_4 : [0,1]^{40} \to \mathbb{R}$. Again, a 30-layer RevNet is used, but now the regression is performed using a small

Table 1: Results from the experiments in Section 5.2, including sensitivity measures and low-dimensional regression errors.

		100 Training Samples				500 Training Samples				2500 Training Samples			
Function	Method	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %
f_4	New NLL	78.7	3.86	8.27	10.9	89.8	1.82	3.52	5.16	94.5	0.827	1.72	2.35
	Old NLL	60.4	6.63	14.5	18.8	65.9	4.58	10.5	13.0	69.2	4.02	9.11	11.4
	AS 1-D	25.8	30.3	75.9	85.9	25.9	21.7	39.5	61.4	25.9	15.9	37.6	44.8
f_5	New NLL	75.1	0.920	5.79	7.92	88.6	0.370	2.78	3.97	93.8	0.154	1.63	1.98
	Old NLL 1	54.6	0.699	7.48	9.40	55.4	0.942	7.26	9.52	56.1	0.784	6.91	8.05
	Old NLL 2	61.8	1.80	12.9	21.1	68.7	1.03	9.22	11.1	67.5	0.894	8.16	9.69



(a)



(b)

Figure 3: (a) Relative sensitivity of $f_4 \circ \mathbf{h}$ to each $\mathbf{z}$-coordinate; (b) Relative value of loss during the first 2500 epochs of NLL training.

feed-forward neural network of two fully-connected layers with 20 neurons each. In each case, the regression is trained for 5000 epochs with a learning rate of 0.05, while the NLL networks Old NLL respectively New NLL are trained for 5000 epochs with learning rates of 0.02 respectively 0.003. Again, the loss functional $\tilde{L}$ is used to train Old NLL.

The results of this experiment are illustrated in Figs. 3 and 4. Again, Fig. 3(b) shows that New NLL reaches roughly 1% of its initial training loss after just 100 epochs, while Old NLL plateaus around 15% regardless of the training length. The sensitivities of $f_4 \circ \mathbf{h}$ also behave as expected, even in the presence of few training samples. Indeed, for 500 samples New NLL concentrates 90% of the total sensitivity in z^1, Old NLL concentrates 66%, and AS concentrates 26%. Note that the number of samples does not affect the quality of the AS reduction, which is both a strength and a weakness of this method. Conversely, all algorithms benefit from increased training samples during the regression, although the approximation built using New NLL remains the most accurate. Observe that 2500 samples is enough for around 2% error with New NLL, while Old NLL still produces errors around 10%. Fig. 4 shows that Active Subspaces is also able to find the general shape of the function, but the one-dimensional approximation using this linear technique still produces around 40% error.

5.3. Application to parametric differential equations

As mentioned in Section 1, one of the primary applications of dimension reduction methods such as NLL and AS lies in predicting quantities of interest which arise from

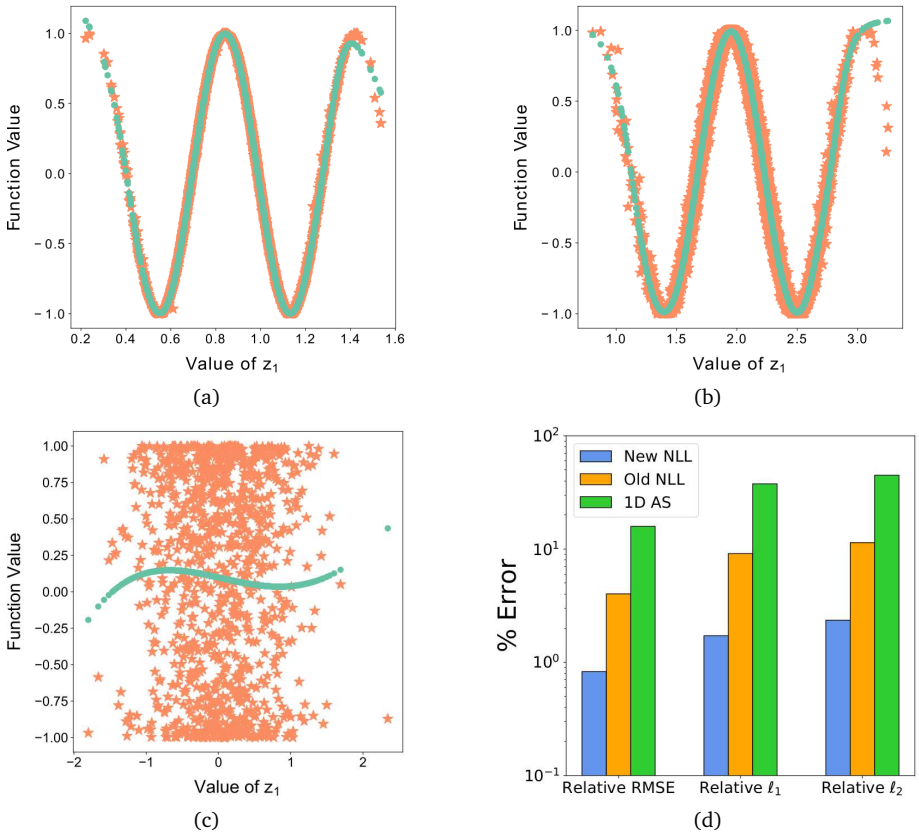


Figure 4: Regression and errors on $f_4 \circ \mathbf{h}$: (a) New NLL; (b) Old NLL; (c) Active Subspaces; (d) Errors of the three approaches.

systems of differential equations. To that end, we now consider two parameterized models for physical phenomena: a modified SIER model for disease spread, and an idealized model for fluid dynamics.

Predicting the basic reproduction number of a disease. Let $\dot{f}$ denote the (total) time derivative of f and consider the following modified SEIR model considered in [17]

Table 2: Results from the experiments in Section 5.3, including sensitivity measures and low-dimensional regression errors.

		20 Training Samples				100 Training Samples				500 Training Samples			
Function	Method	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %	z_A Sens %	RRMSE %	Rℓ_1 %	Rℓ_2 %
R_0	New NLL	75.1	0.435	0.731	1.15	96.4	0.249	0.433	0.605	97.9	0.254	0.469	0.612
	Old NLL	62.0	1.64	2.89	4.59	73.2	1.42	3.09	3.89	72.2	1.24	2.39	3.12
	AS 1-D	55.0	9.18	18.5	22.1	54.4	18.2	22.0	9.14	53.6	18.8	22.7	9.43
	AS 2-D	79.3	4.34	7.87	10.4	79.7	8.01	10.8	4.49	79.6	8.26	10.8	4.50
K	New NLL	97.6	0.425	1.12	1.27	98.3	0.186	0.496	0.555	98.3	0.101	0.502	0.540
	Old NLL	80.1	3.52	9.82	10.5	80.5	3.19	8.99	9.53	80.3	3.25	9.15	9.70
	AS 1-D	64.4	6.64	18.8	19.8	65.1	6.81	19.6	20.3	65.0	6.78	19.4	20.2
	AS 2-D	87.5	3.32	9.54	9.90	88.7	2.64	6.96	7.88	88.7	2.65	7.06	7.91

for the spread of Ebola in West Africa:

$$\begin{aligned}
\dot{S} &= -\beta_1 SI - \beta_2 SR_I - \beta_3 SH, & \dot{R}_I &= \rho_1\gamma_1 I - \omega R_I,\\
\dot{E} &= \beta_1 SI + \beta_2 SR_I + \beta_3 SH - \delta E, & \dot{R}_B &= \omega R_I + \rho_2\gamma_2 H,\\
\dot{I} &= \delta E - \gamma_1 I - \psi I, & \dot{R}_R &= (1-\rho_1)\gamma_1 I + (1-\rho_2)\gamma_2 H,\\
\dot{H} &= \psi I - \gamma_2 H.
\end{aligned}$$

Here S represents the fraction of the population that is susceptible to infection, E represents the (infected but asymptomatic) exposed population, I is the infected fraction, H is the hospitalized fraction, R_I represents the infectious dead (not properly buried), R_B represents the non-infectious dead (properly buried), R_R represents the recovered population, and $\delta = \frac{1}{9}$ is a constant. When studying the spread of disease, it is important to compute the basic reproduction number

$$R_0 = \frac{1}{\gamma_1 + \psi}\left(\beta_1 + \frac{\beta_2\rho_1\gamma_1}{\omega} + \frac{\beta_3}{\gamma_2}\psi\right),$$

which depends on the eight parameters $\beta_1, \beta_2, \beta_3, \rho_1, \gamma_1, \gamma_2, \omega, \psi$ and measures the potential of the disease to transmit throughout the population. In [17], parameter ranges for this model are given around a baseline computed using data provided by the World Health Organization which was collected in the country of Liberia, and AS is used for the prediction of R_0. It is interesting to apply NLL to this problem for comparison with the existing results. In this case, the training of the NLL models is done with a 15-layer RevNet and uniformly distributed samples drawn from the ranges in [17, Table 7]. In particular, Old NLL respectively New NLL are trained for 5000 epochs with learning rates of 0.1 respectively 0.005, and regression is performed using 10-neighbor local quadratic least-squares. Conversely, here global quartic least-squares are used for the AS regression, since global methods outperform local fitting when the spread of function values is wide (c.f. Fig. 6(c)). Note that Old NLL is trained using the loss functional $\hat{L}$, as this leads to better performance.

The results of this comparison show that the benefits of New NLL persist in this situation also. In particular, 90% of the total sensitivity in $R_0 \circ \mathbf{h}$ is concentrated by New NLL in the active direction (for $100+$ training samples), and even a 20 sample training set is sufficient for 1% regression error. Contrast this with Old NLL and either a one-dimensional or two-dimensional AS, which yield significantly more erroneous approximations. Fig. 6 illustrates the low-dimensional regressions, and Fig. 5(a) shows the errors. As expected, Fig. 6(a) shows that the approximation trained on the New NLL reduction produces a much tighter fit than existing methods. Interestingly, note that here both NLL algorithms train relatively well (c.f. Fig. 5(c)), and the Old NLL regression outperforms the 2-D AS regression despite concentrating less sensitivity in the active directions.

Predicting the total kinetic energy. The last experiment in this section considers the parameterized one-dimensional inviscid Burgers' equation, which is a common model

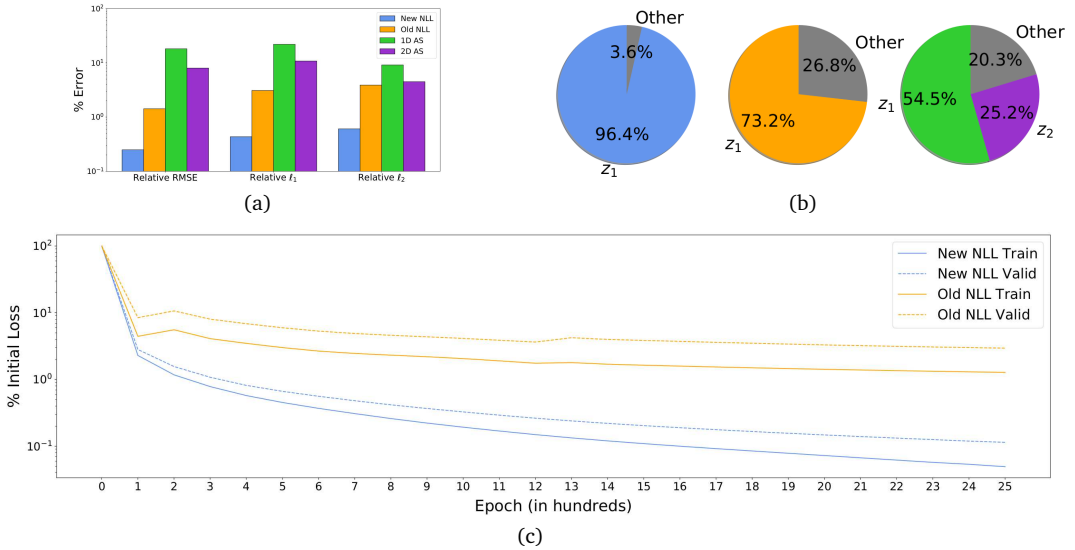


Figure 5: (a) Relative regression errors on $R_0 \circ \mathbf{h}$; (b) Relative sensitivity of $R_0 \circ \mathbf{h}$ to $\mathbf{z}_A$ as a percentage of total; (c) Relative value of loss during the first 2500 epochs of NLL training (log scale).

for fluids whose motion can develop discontinuities. In particular, let $\boldsymbol{\mu} = (\mu_1\, \mu_2\, \mu_3)^\intercal$ and consider the initial value problem,

$$\begin{aligned} &w_t + \frac{1}{2}\left(w^2\right)_x = \mu_3 e^{\mu_2 x}, \\ &w(a,t,\boldsymbol{\mu}) = \mu_1, \\ &w(x,0,\boldsymbol{\mu}) = 1, \end{aligned} \tag{5.1}$$

where $w = w(x,t,\boldsymbol{\mu})$ represents the position of the fluid and $x \in [a,b]$. It is interesting to examine the performance of NLL and AS in predicting the total kinetic energy at time t, given by

$$K(t,\boldsymbol{\mu}) = \frac{1}{2}\int_0^t \int_a^b w(x,\tau,\boldsymbol{\mu})^2\, dx\, d\tau.$$

As K is a function of the PDE solution, this represents a case where prediction is most valuable. Indeed, for more expensive PDE simulations it may not be possible to generate as much data as desired. Therefore, the goal is to use sparsely sampled values of K obtained from simulations of (5.1) to train an approximation to this function at any $(t,\boldsymbol{\mu})$ in parameter space.

To accomplish this using the methods NLL and AS, it is necessary to have access to the gradient ∇K at the sampled points. Noting that $[\partial_t, \partial_x] = [\partial_t, \partial_{\mu_i}] = [\partial_x, \partial_{\mu_i}] = 0$ for $1 \le i \le 3$, simple differentiation yields

$$\nabla K(t,\boldsymbol{\mu}) = (K_t\, K_{\boldsymbol{\mu}})^\intercal = \left(\frac{1}{2}\int_a^b w(x,t,\boldsymbol{\mu})^2\, dx \quad \int_0^t \int_a^b w(x,\tau,\boldsymbol{\mu})\, w_{\boldsymbol{\mu}}(x,\tau,\boldsymbol{\mu})\, dx\, d\tau\right)^\intercal.$$

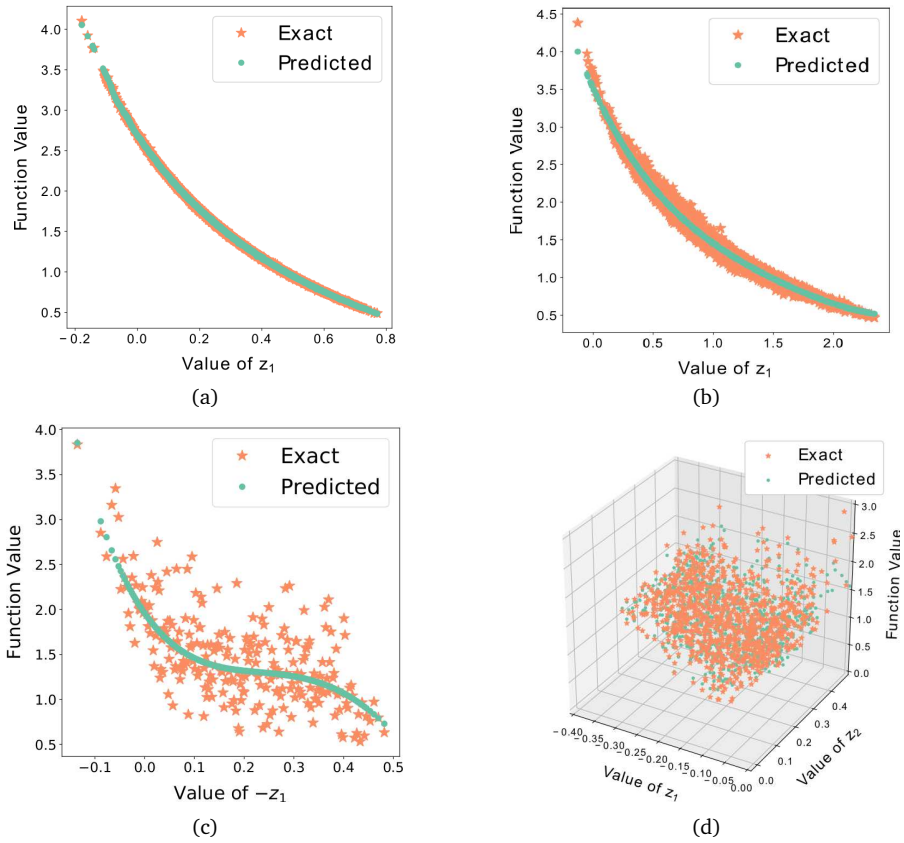


Figure 6: Regression on $R_0 \circ \mathbf{h}$: (a) New NLL; (b) Old NLL; (c) 1-D AS; (d) 2-D AS.

It follows that K_t can be obtained immediately from the solution w, and the components K_μ are further computable by solving sensitivity equations. In particular, differentiating (5.1) with respect to $\boldsymbol{\mu}$ yields the following system of initial value problems in the variable $w_{\boldsymbol{\mu}}$:

$$\begin{aligned} &w_{\boldsymbol{\mu},t} + (w w_{\boldsymbol{\mu}})_x = (0 \;\; x\mu_3 e^{\mu_2 x} \;\; e^{\mu_2 x})^\mathsf{T}, \\ &w_{\boldsymbol{\mu}}(a,t,\boldsymbol{\mu}) = (1 \;\; 0 \;\; 0)^\mathsf{T}, \\ &w_{\boldsymbol{\mu}}(x,0,\boldsymbol{\mu}) = \mathbf{0}. \end{aligned} \tag{5.2}$$

Solving (5.1) and (5.2) provides the necessary samples $\{K(t^s,\boldsymbol{\mu}^s),\, \nabla K(t^s,\boldsymbol{\mu}^s)\}_{s\in S}$ for applying the NLL and AS algorithms.

In the example at hand, the parameters are chosen to take values in $(t,\mu_1,\mu_2,\mu_3) \in [25,30]\times[3,8]\times[0.015,0.06]\times[0,0.05]$ with $x\in[0,100]$. Systems (5.1) and (5.2) are discretized using simple forward Euler with upwinding with increments of $\Delta x = 0.4$ and $\Delta t = 0.025$. Old NLL respectively New NLL are trained for 5000 epochs using a 7-layer RevNet with learning rates of 0.1 respectively 0.005, and regressions are performed using a neural network as in the case of function f_4. Again, Old NLL is trained using the functional $\hat{L}$.

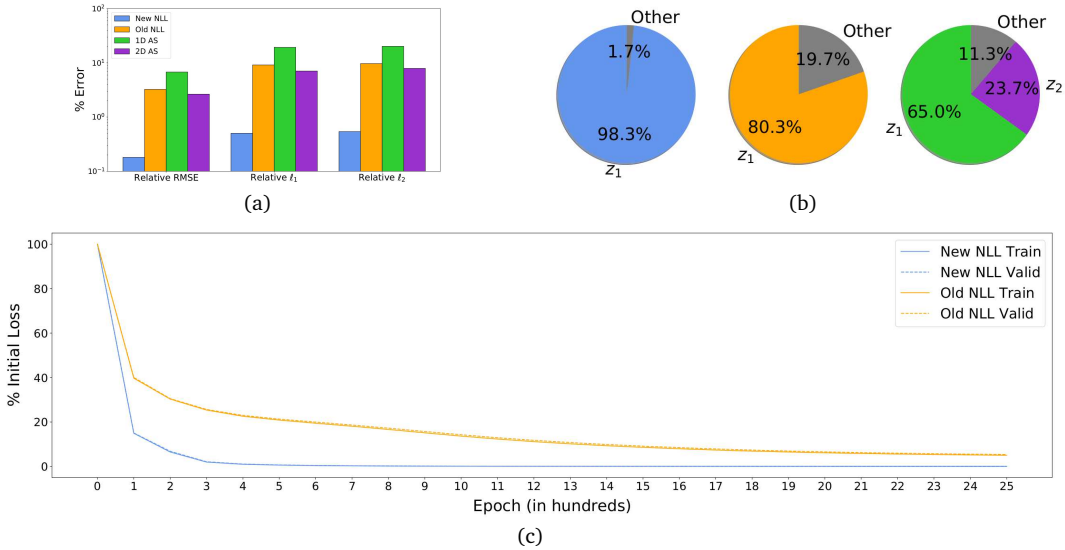


Figure 7: (a) Relative regression errors on $K \circ \mathbf{h}$; (b) Relative sensitivity of $K \circ \mathbf{h}$ to $\mathbf{z}_A$ as a percentage of total; (c) Relative value of loss during the first 2500 epochs of NLL training.

Results are displayed in Figs. 7 and 8. Predictably, both versions of NLL are able to reproduce a reasonable one-dimensional approximation, although the New NLL approximation is much more accurate. Moreover, while a one-dimensional AS approximation can only capture the general trend in the function, a two-dimensional AS approximation (slightly) outperforms the Old NLL approximation. When using New NLL, Figs. 7(a) and (b) show that 100 training samples is sufficient for a sensitivity concentration of 98% and regression errors of around 0.5%.

6. Conclusion

An improved version of the NLL algorithm from [41] has been proposed which reduces the input dimension to one in every case. By reformulating the central learning problem as the minimization of a Dirichlet-type energy functional, good stability and convergence properties are exhibited despite the use of sparse and high-dimensional training data. Through various illustrative examples it has been demonstrated that New NLL has several benefits over the original NLL algorithm and the linear method of Active Subspaces, including faster training than Old NLL and a sharper, more complete dimension reduction. Results show that regression approximations trained after New NLL are also more accurate, leading to confident prediction when approximating functionals of ODE/PDE solutions. Future work includes obtaining rigorous estimates on the data dependence and algorithmic convergence of New NLL, as well as studying the connection between harmonic maps and minimizers of the NLL algorithm. Finally, it remains to investigate the performance of NLL when the level sets of f in U cannot be covered by just one coordinate chart. Does a procedure motivated by the local IFT still

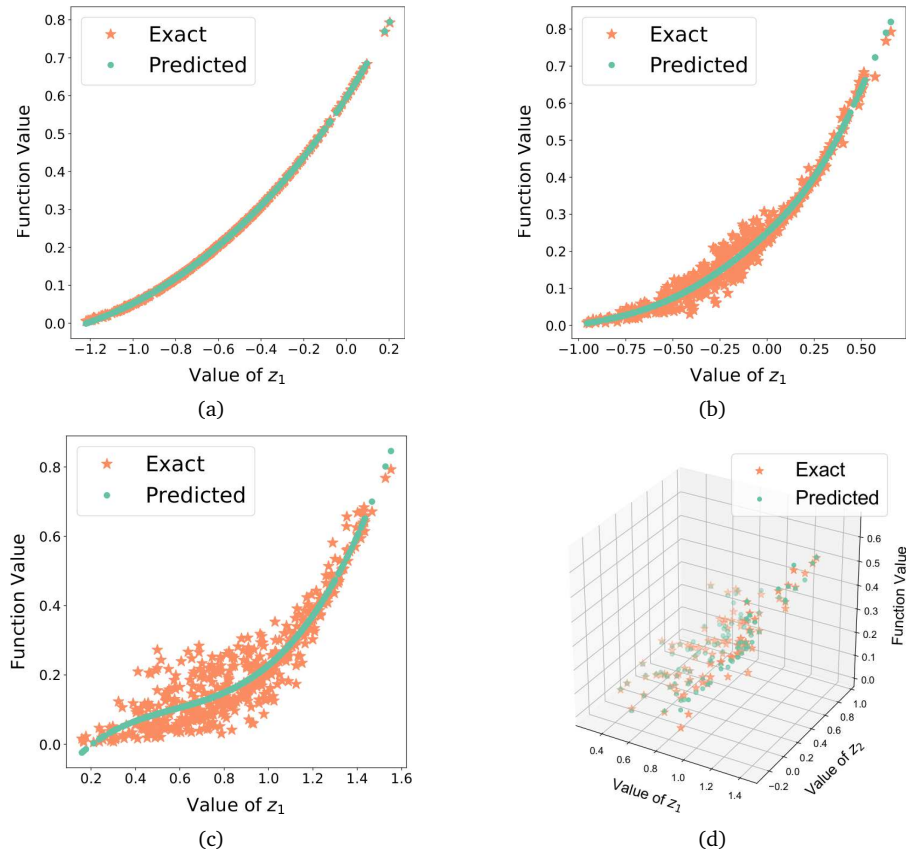


Figure 8: Regression on $K \circ \mathbf{h}$: (a) New NLL; (b) Old NLL; (c) 1-D AS; (d) 2-D AS.

produce a good mapping in this case? If not, is there a straightforward modification which yields better results? Although these concerns can always be addressed in principle by shrinking the domain U, it would be interesting to have an algorithm which is agnostic to such topological considerations.

Acknowledgments

The authors are grateful to the anonymous referee for helpful suggestions regarding the presentation. This work is partially supported by U.S. Department of Energy Scientific Discovery through Advanced Computing (Grants DE-SC0020270, DE-SC0020418).

References

[1] H. Abdi and L. J. Williams, *Principal component analysis*, Wiley Interdiscip. Rev. Comput. Stat. 2 (2010), 433–459.

[2] K. P. Adragni and R. D. Cook, *Sufficient dimension reduction and prediction in regression*, Philos. Trans. A Math. Phys. Eng. Sci. 367 (2009), 4385–4405.

[3] M. BALASUBRAMANIAN, E. L. SCHWARTZ, J. B. TENENBAUM, V. DE SILVA, AND J. C. LANGFORD, *The Isomap algorithm and topological stability*, Science 295 (2002), p. 7.
[4] Y. BENGIO, J.-F. PAIEMENT, P. VINCENT, O. DELALLEAU, N. ROUX, AND M. OUIMET, *Out-of-sample extensions for LLE, Isomap, MDS, eigenmaps, and spectral clustering*, NeurIPS 16 (2003), 177–184.
[5] R. BRIDGES, A. GRUBER, C. FELDER, M. VERMA, AND C. HOFF, *Active manifolds: A non-linear analogue to Active Subspaces*, In: Proceedings of the 36-th International Conference on Machine Learning, K. Chaudhuri and R. Salakhutdinov (Eds.), Vol. 97 of Proceedings of Machine Learning Research, PMLR, 09-15 Jun 2019, 764–772.
[6] R. L. BRYANT, *An introduction to Lie groups and symplectic geometry*, Geometry and Quantum Field Theory (1995), 321–347.
[7] M. BUDNINSKIY, G. YIN, L. FENG, Y. TONG, AND M. DESBRUN, *Parallel transport unfolding: A connection-based manifold learning approach*, SIAGA 3 (2019), 266–291.
[8] B. CHANG, L. MENG, E. HABER, L. RUTHOTTO, D. BEGERT, AND E. HOLTHAM, *Reversible architectures for arbitrarily deep residual neural networks*, In: Proceedings Conference AAAI Artif. Intell., Vol. 32, 2018.
[9] H. CHOI AND S. CHOI, *Robust kernel isomap*, Pattern Recognit. 40 (2007), 853–862.
[10] C. K. CHUI AND X. LI, *Approximation by ridge functions and neural networks with one hidden layer*, J. Approx. Theory 70 (1992), 131–141.
[11] P. CONSTANTINE, R. HOWARD, A. GLAWS, Z. GREY, P. DIAZ, AND L. FLETCHER, *Python active-subspaces utility library*, J. Open Source Softw. 1 (2016), p. 79.
[12] P. G. CONSTANTINE, *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*, SIAM, 2015.
[13] P. G. CONSTANTINE, E. DOW, AND Q. WANG, *Active subspace methods in theory and practice: applications to kriging surfaces*, SIAM J. Sci. Comput. 36 (2014), A1500–A1524.
[14] P. G. CONSTANTINE, A. EFTEKHARI, J. HOKANSON, AND R. A. WARD, *A near-stationary subspace for ridge approximation*, Comput. Method. Appl. M. 326 (2017), 402–421.
[15] R. D. COOK, *Fisher lecture: Dimension reduction in regression*, Statistical Science 22 (2007), 1–26.
[16] A. DASGUPTA, P. DRINEAS, B. HARB, R. KUMAR, AND M. W. MAHONEY, *Sampling algorithms and coresets for ℓ_p regression*, SIAM J. Sci. Comput. 38 (2009), 2060–2078.
[17] P. DIAZ, P. CONSTANTINE, K. KALMBACH, E. JONES, AND S. PANKAVICH, *A modified SEIR model for the spread of Ebola in Western Africa and metrics for resource allocation*, Appl. Math. Comput. 324 (2018), 141–155.
[18] J. H. FRIEDMAN AND W. STUETZLE, *Projection pursuit regression*, J. Am. Stat. Assoc. 76 (1981), 817–823.
[19] A. N. GOMEZ, M. REN, R. URTASUN, AND R. B. GROSSE, *The reversible residual network: backpropagation without storing activations*, in:NeurIPS, 2017.
[20] S. GREENLAND, M. A. MANSOURNIA, AND D. G. ALTMAN, *Sparse data bias: a problem hiding in plain sight*, BMJ, 352 (2016).
[21] R. HADSELL, S. CHOPRA, AND Y. LECUN, *Dimensionality reduction by learning an invariant mapping*, In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), Vol. 2, IEEE, 2006, 1735–1742.
[22] D. M. HAWKINS, *The problem of overfitting*, J. Chem. Inform. Comput. Sci. 44 (2004), 1–12.
[23] G. E. HINTON AND R. R. SALAKHUTDINOV, *Reducing the dimensionality of data with neural networks*, Science 313 (2006), 504–507.
[24] K. HORNIK, M. STINCHCOMBE, AND H. WHITE, *Multilayer feedforward networks are uni-*

versal approximators, Neural Netw. 2 (1989), 359–366.
[25] A. I. KHURI AND S. MUKHOPADHYAY, *Response surface methodology*, Wiley Interdiscip. Rev. Comput. Stat. 2 (2010), 128–149.
[26] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, arXiv:1412.6980, 2014.
[27] R. R. LAM, O. ZAHM, Y. M. MARZOUK, AND K. E. WILLCOX, *Multifidelity dimension reduction via active subspaces*, SIAM J. Sci. Comput. 42 (2020), A929–A956.
[28] J. A. LEE, A. LENDASSE, AND M. VERLEYSEN, *Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis*, Neurocomputing 57 (2004), 49–76.
[29] L. LI, *Sparse sufficient dimension reduction*, Biometrika 94 (2007), 603–613.
[30] T. W. LUKACZYK, P. CONSTANTINE, F. PALACIOS, AND J. J. ALONSO, *Active subspaces for shape optimization*, In: 10-th AIAA Multidisciplinary Design Optimization Conference, 2014, p. 1171.
[31] Y. MA AND L. ZHU, *A review on dimension reduction*, Int. Stat. Rev. 81 (2013), 134–150.
[32] R. H. MYERS, D. C. MONTGOMERY, AND C. M. ANDERSON-COOK, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, John Wiley & Sons, 2016.
[33] T. O'LEARY-ROSEBERRY, U. VILLA, P. CHEN, AND O. GHATTAS, *Derivative-informed projected neural networks for high-dimensional parametric maps governed by PDEs*, arXiv: 2011.15110, 2020.
[34] A. PINKUS, *Approximating by ridge functions*, In:Surface Fitting and Multiresolution Methods, A. Le Mehaute, C. Rabut, L. L. Schumaker (Eds.), Vanderbilt University Press (1997), 279–292.
[35] S. T. ROWEIS AND L. K. SAUL, *Nonlinear dimensionality reduction by locally linear embedding*, Science 290 (2000), 2323–2326.
[36] R. M. ROYALL, *On finite population sampling theory under certain linear regression models*, Biometrika 57 (1970), 377–387.
[37] R. SCHOEN AND K. UHLENBECK, *Boundary regularity and the Dirichlet problem for harmonic maps*, J. Diff. Geom. 18 (1983), 253–268.
[38] Y. WANG, H. YAO, AND S. ZHAO, *Auto-encoder based dimensionality reduction*, Neurocomputing 184 (2016), 232–242.
[39] J. WENG AND D. S. YOUNG, *Some dimension reduction strategies for the analysis of survey data*, J. Big Data 4 (2017), p. 43.
[40] S. WOLD, K. ESBENSEN, AND P. GELADI, *Principal component analysis*, Chemom. Intell. Lab. Syst. 2 (1987), 37–52.
[41] G. ZHANG, J. ZHANG, AND J. HINKLE, *Learning nonlinear level sets for dimensionality reduction in function approximation*, In: Advances in Neural Information Processing Systems, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32, Curran Associates Inc., 2019.
[42] Y. ZHANG, Z. ZHANG, J. QIN, L. ZHANG, B. LI, AND F. LI, *Semi-supervised local multi-manifold isomap by linear embedding for feature extraction*, Pattern Recognit. 76 (2018), 662–678.