IOP Publishing Inverse Problems

Inverse Problems **34** (2018) 014004 (22pp) https://doi.org/10.1088/1361-6420/aa9a90

Stable architectures for deep neural networks

Eldad Haber[1,3] **and Lars Ruthotto**[2,3]

[1] Department of Earth and Ocean Science, The University of British Columbia, Vancouver, BC, Canada
[2] Department of Mathematics and Computer Science, Emory University, Atlanta, GA, United States of America
[3] Xtract Technologies Inc., Vancouver, Canada

E-mail: haber@math.ubc.ca, lruthotto@emory.edu and info@xtract.tech

Received 30 June 2017, revised 14 September 2017
Accepted for publication 14 November 2017
Published 13 December 2017

Abstract

Deep neural networks have become invaluable tools for supervised machine learning, e.g. classification of text or images. While often offering superior results over traditional techniques and successfully expressing complicated patterns in data, deep architectures are known to be challenging to design and train such that they generalize well to new data. Critical issues with deep architectures are numerical instabilities in derivative-based learning algorithms commonly called exploding or vanishing gradients. In this paper, we propose new forward propagation techniques inspired by systems of ordinary differential equations (ODE) that overcome this challenge and lead to well-posed learning problems for arbitrarily deep networks.

The backbone of our approach is our interpretation of deep learning as a parameter estimation problem of nonlinear dynamical systems. Given this formulation, we analyze stability and well-posedness of deep learning and use this new understanding to develop new network architectures. We relate the exploding and vanishing gradient phenomenon to the stability of the discrete ODE and present several strategies for stabilizing deep learning for very deep networks. While our new architectures restrict the solution space, several numerical experiments show their competitiveness with state-of-the-art networks.

Keywords: machine learning, deep neural networks, dynamic inverse problems, PDE-constrained optimization, parameter estimation, image classification

(Some figures may appear in colour only in the online journal)

1. Introduction

In this work, we propose new architectures for deep neural networks (DNNs) and exemplarily show their effectiveness for solving supervised machine learning (ML) problems; for a general overview about DNNs and ML see, e.g. [1, 21, 22, 40] and reference therein. We consider the following classification problem: Assume we are given *training data* consisting of s feature vectors, $\mathbf{y}_1, \ldots, \mathbf{y}_s \in \mathbb{R}^n$, and label vectors, $\mathbf{c}_1, \ldots, \mathbf{c}_s \in \mathbb{R}^m$, whose kth components represent the likelihood of an example belonging to class k. The goal is to *learn* a function that approximates the data-label relation on the training data and generalizes well to similar unlabeled data. Our goals in this work are to highlight the relation of the learning problem to dynamic inverse problems, analyze its stability and ill-posedness for commonly used architectures, and derive new architectures that alleviate some of these difficulties for arbitrarily deep architectures.

We are particularly interested in deep learning, i.e. ML using neural networks with many hidden layers. DNNs have been successful in supervised learning, particularly when the relationship between the data and the labels is highly nonlinear; see, e.g. [5, 30, 34, 35] and reference therein. Their depths (i.e. their number of layers) allow DNNs to express complex data-label relationships since each layer nonlinearly transforms the features and therefore effectively filters the information content.

Given the training data $(\mathbf{y}_1, \mathbf{c}_1), \ldots, (\mathbf{y}_s, \mathbf{c}_s)$, an inverse problem needs to be solved in order to *train* a given network architecture. This problem, also called the *learning problem*, aims at finding a parameterization of the DNN that explains the data-label relation and generalizes well to new unlabeled data. Clearly, using deeper network architectures increases the capacity of the network but also the dimensionality, and thus the computational complexity, of the parameter estimation problem. Additionally, more labeled data is required to calibrate very deep networks reliably. Therefore, despite the fact that neural networks have been used since the early 70s, deep learning has only recently revolutionized many applications fueled by advances in computational hardware and availability of massive data sets.

Well-known sources of difficulty in deep learning are the dimensionality and non-convexity of the associated optimization problem. Traditionally, stochastic gradient descent methods have been used [43]. It has been observed that the performance of the DNN can be highly dependent on the choice of optimization algorithm and sample size [11, 14]. Furthermore, it has been noted that some optimization algorithms yield DNNs that generalize poorly to new unlabeled data [33].

Additional difficulties in deep learning stem from instabilities of the underlying forward model, most importantly, the propagation of features through the DNN. As has been shown in [11], the output of some networks can be unstable with respect to small perturbations in the original features. A related problem is the observation of vanishing or exploding gradients [6]. These results are unsettling since predictions made by networks with unstable forward propagation are very sensitive to small perturbations of the input features (as is common, e.g. in adversarial attacks), which may render the network useless in practice.

The main goal of this work is to gain new insight into the stability of the forward propagation and the well-posedness of the learning problem summarized in the following two questions:

(i) Given a network architecture and parameters obtained by some optimization process, is the forward propagation problem well-posed?
(ii) Is the learning problem well-posed? In other words, given sufficient training are there parameters such that the DNN generalizes well or can generalization be improved by adding appropriate regularization?

The first question is important because, while it may be possible to fit the training data even for unstable forward propagation models, the trained network is unlikely to generalize. In other words, small deviations in the data, e.g. due to noise, may be drastically amplified by the forward propagation, resulting in incorrect labels. We show that the forward problem can be thought of as a discretization of an ordinary differential equation (ODE). Therefore, the stability of the network corresponds to the stability of its underlying ODE. Based on this observation we develop stability criteria for a simplified version of the commonly used residual network (ResNet) architecture [27] and develop *new network architectures* that are ensured to be stable and lead to well-posed learning problems.

The paper is organized as follows. In section 2 we give a brief mathematical derivation of the deep learning problem illustrated using the ResNet architecture. In section 3 we analyze the stability of the simplified ResNet forward propagation and the well-posedness of the resulting learning problem. Our analysis and examples suggest stability criteria, and as a result, in section 4, we propose three new stable architectures. In section 5 we propose regularization functions favoring smoothness of parameters and multi-level ideas to initialize deep networks. In section 6 we demonstrate the effectiveness of our new architectures on a number of small-scale model problems and explore their effectiveness for solving image classification problems. Finally, in section 7 we summarize the paper.

2. Mathematical formulation of the deep learning problem

In this section we briefly describe three main ingredients of deep learning relevant to our work; for a comprehensive introduction see, e.g. [1, 21, 22, 40]. First, we outline *forward propagation* techniques, which transforms the input features in a nonlinear way to filter their information. Second, we describe *classification*, which predicts the class label probabilities using the features at the output layer (i.e. the output of the forward propagation). Finally, we formulate the learning problem, which aims at estimating parameters of the forward propagation and classification that approximate the data-label relation.

For notational convenience we stack the training features and labels row-wise into matrices $\mathbf{Y}_0 = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_s]^\top \in \mathbb{R}^{s\times n}$ and $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_s]^\top \in \mathbb{R}^{s\times m}$.

To exemplify our discussion of forward propagation we consider a simplified version of the residual neural network (ResNet) [27] model that has been very successful in classifying images using deep network architectures; see [22] for other options. In ResNets, the forward propagation of the input values, $\mathbf{Y}_0 \in \mathbb{R}^{s\times n}$, through a network consisting of N layers is given by

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{Y}_j\mathbf{K}_j + b_j) \quad \text{for} \quad j = 0, \ldots, N-1. \tag{2.1}$$

The propagation in equation (2.1) is parametrized by the nonlinear activation function $\sigma : \mathbb{R}^{s\times n} \to \mathbb{R}^{s\times n}$ and affine transformations represented by their weights, $\mathbf{K}_0, \mathbf{K}_1, \ldots, \mathbf{K}_{N-1} \in \mathbb{R}^{n\times n}$, and biases, $b_0, b_1, \ldots, b_{N-1} \in \mathbb{R}$. We augmented the original formulation in [27] by the parameter $h > 0$ in order to increase the stability of the forward propagation and allow for a continuous interpretation of the process; see also section 3. The values $\mathbf{Y}_1, \ldots, \mathbf{Y}_{N-1}$ are also called *hidden* layers and $\mathbf{Y}_N$ is called the *output* layer. The activation function is applied element-wise and is typically (piecewise) smooth and monotonically non-decreasing. As two commonly used examples, we consider the hyperbolic tangent and the rectified linear unit (ReLU) activations

$$\sigma_{\text{ht}}(\mathbf{Y}) = \tanh(\mathbf{Y}) \quad \text{and} \quad \sigma_{\text{ReLU}}(\mathbf{Y}) = \max(0, \mathbf{Y}).$$

The class label probabilities are predicted using the values at the output layers, $\mathbf{Y}_N$, a *hypothesis function* $\mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top)$, and its associated weights, $\mathbf{W} \in \mathbb{R}^{n\times m}$, and bias, $\mu \in \mathbb{R}^m$. Here $\mathbf{e}_k \in \mathbb{R}^k$ denotes the k-dimensional vector of all ones. For Bernoulli variables (i.e. $\mathbf{C} \in \{0,1\}^{s\times m}$) it is natural to consider the logistic regression function

$$\mathbf{h}(\mathbf{x}) = \exp(\mathbf{x})./(1 + \exp(\mathbf{x})), \tag{2.2}$$

where the exponential and the division are applied element-wise. For multinomial distributions we use the softmax function

$$\mathbf{h}(\mathbf{X}) = \exp(\mathbf{X})./(\exp(\mathbf{X})\mathbf{e}_m). \tag{2.3}$$

The learning problem aims at estimating the parameters of the forward propagation (i.e. $\mathbf{K}_0, \mathbf{K}_1, \ldots, \mathbf{K}_{N-1}$ and $b_0, b_1, \ldots, b_{N-1}$) and the classifier ($\mathbf{W}$ and μ) so that the DNN accurately approximates the data-label relation for the training data *and* generalizes to new unlabeled data. As we show below, the learning problem can be cast as a dynamic inverse problem, which provides new opportunities for applying theoretical and computational techniques from parameter estimation to deep learning problems. We phrase learning as an optimization problem

$$\begin{aligned} &\min \frac{1}{s} S(\mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top), \mathbf{C}) + \alpha R(\mathbf{W}, \mu, \mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1}) \\ &\text{s.t.} \mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{Y}_j\mathbf{K}_j + b_j), \quad j = 0, 1, \ldots, N-1, \end{aligned} \tag{2.4}$$

where the loss function S is convex in its first argument and measures the quality of the predicted class label probabilities, the convex regularizer R penalizes undesirable (e.g. highly oscillatory) parameters, and the parameter $\alpha > 0$ balances between minimizing the data fit and regularity of the parameters. A simple example for a loss function is the sum-of-squared difference function $S(\mathbf{C}_{\text{pred}}, \mathbf{C}) = \frac{1}{2}\|\mathbf{C}_{\text{pred}} - \mathbf{C}\|_F^2$. Since our numerical experiments deal with classification we use cross entropy loss functions. Choosing an "optimal" regularizer, R, and regularization parameter, α, is both crucial and nontrivial. Commonly Tikhonov regularization [19, 26, 48], also referred to as weight decay, has been used [23], although, other possibilities enforcing sparsity or other structure have been proposed [41]. We introduce novel regularization functions in section 5. For simplicity, we assume that a suitable value of $\alpha > 0$ is chosen by the user or that it is done dynamically as suggested in [12].

There are numerous approaches to solving the learning problem. In this work we use a simple block coordinate descent method to demonstrate the properties of the forward propagation. Our method alternates between updating the parameters of the classifier $(\mathbf{W}, \mu)$ fixing the current value of the propagated features $\mathbf{Y}_N$ and then updating the parameters of the forward propagation while keeping the updated weights of the classifier fixed. The first problem is typically convex and the latter problem is generally non-convex due to the forward propagation process. Both steps are based on subsampling the training data. To this end note that most common loss functions can be written as a sum over all examples, i.e.

$$\begin{aligned} \frac{1}{s} S(\mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top), \mathbf{C}) &= \frac{1}{s}\sum_{i=1}^{s} S(\mathbf{h}(\mathbf{y}_i^\top\mathbf{W} + \mu^\top), \mathbf{c}_i^\top) \\ &\approx \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}} S(\mathbf{h}(\mathbf{y}_i^\top\mathbf{W} + \mu^\top), \mathbf{c}_i^\top), \end{aligned}$$

where $\mathcal{T} \subset \{1, 2, \ldots, s\}$ is a randomly chosen set updated in each iteration of the block coordinate descent method. The size of the batches is a parameter in our algorithm whose choice depends on the size and complexity of the problem and resource considerations.

In the following, we assume the sample size is constant; for adaptive selection of sample size see, e.g. [14]. In each iteration of the block coordinate descent scheme, our algorithm approximately solves the resulting classification problem using a Newton-PCG method, i.e. an inexact Newton method that uses a preconditioned conjugate gradient (PCG) method to determine a search direction (see, e.g. [29, 42, 47]). Subsequently, the weights of the forward propagation are updated using a Gauss–Newton-PCG method. Note that gradient and approximated Hessian computations require matrix-vector products with the derivative matrices of the values of the output layer, $\mathbf{Y}_N$, with respect to $\mathbf{K}_{0,\ldots,N-1}$ and $b_{0,\ldots,N-1}$. The matrix-vector products with the derivative matrix can be computed without its explicit construction through forward and backward propagation, respectively; see also [38]. However, this requires storing (or recomputing) the values at the output layers; see, for instance, section 4.4 for derivative computation. Therefore, as also suggested in [13], we subsample $\mathcal{T}$ further to reduce the cost of the Hessian matrix-vector products in our PCG scheme.

Our implementation includes computing the validation error in each iteration of the block coordinate descent method. The final output of our algorithm are the parameters that achieve the lowest validation error.

3. Stability and well-posedness of the forward propagation

In this section we analyze the stability of the ResNet forward problem 2.1 and illustrate why some choices of transformation weights may generate instabilities or prohibit effective learning altogether.

It is well-known that any parameter estimation problem requires a well-posed forward problem, i.e. a problem whose output is continuous with respect to its input. For example, practical image classification algorithms need to be robust against noisy or slightly shifted input images. Ill-posedness of the forward problem implies that even if the estimated parameters lead to a small training error they will probably fail or will do poorly on a perturbation of that data. In other words a network whose forward propagation is ill-posed will generalize poorly. Thus, well-posedness of the forward propagation is a necessary condition to obtain DNNs that generalize well.

The following discussion also gives new perspectives on two well-known phenomena in the deep learning community: Vanishing and exploding gradients; see, e.g. [6]. These phenomena refer to the gradient of the objective function in equation (2.4) and pose a severe challenge for very deep architectures. Note that the gradient represents the sensitivity of the output with respect to a perturbation in the input. Thus, an exploding gradient implies that the output is unstable with respect to the input. Similarly a vanishing gradient implies that the output is insensitive with respect to the input. Clearly both cases prohibit effective training, but more importantly, may not provide DNNs that generalize well.

To understand the phenomena we consider a simplified version of the forward propagation in ResNets given in equation (2.1). As pointed out in [25] the forward propagation can be seen as an explicit Euler discretization of the nonlinear ordinary differential equation (ODE)

$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}^\top(t)\mathbf{y}(t) + b(t)), \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0, \tag{3.5}$$

over a time interval $t = [0, T]$. The final time $T > 0$ and the magnitude of $\mathbf{K}(t)$ control the depth of the network. The ODE is stable if $\mathbf{K}(t)$ is changing sufficiently slow and

$$\max_{i=1,2,\ldots,n} \mathrm{Re}(\lambda_i(\mathbf{J}(t))) \leqslant 0, \quad \forall t \in [0, T], \tag{3.6}$$

where $\mathrm{Re}(\cdot)$ denotes the real part, $\lambda_i(\mathbf{J}(t))$ is the ith eigenvalue of the Jacobian of the right hand side in equation (3.5), denoted by $\mathbf{J}(t) \in \mathbb{R}^{n\times n}$. A more accurate statement that uses kinematic eigenvalues of the Jacobian can be found in [3]. Here, the Jacobian is

$$\begin{aligned}\mathbf{J}(t) &= \left(\nabla_{\mathbf{y}}\left(\sigma(\mathbf{K}(t)^\top \mathbf{y} + b(t))\right)\right)^\top \\ &= \mathrm{diag}\left(\sigma'(\mathbf{K}(t)^\top \mathbf{y} + b(t))\right)\mathbf{K}(t)^\top.\end{aligned} \tag{3.7}$$

Since the activation function σ is typically monotonically non-decreasing, i.e. $\sigma'(\cdot) \geqslant 0$, equation (3.6) is satisfied if $\mathbf{K}$ changes sufficiently slowly and

$$\max_{i=1,2,\ldots,n} \mathrm{Re}(\lambda_i(\mathbf{K}(t))) \leqslant 0, \quad \forall t \in [0,T]. \tag{3.8}$$

Controlling the smoothness of $\mathbf{K}$ can be done by regularization as described in section 5. To ensure the stability of the overall discrete forward propagation, we also require the discrete version of the ODE to have a sufficiently small h as summarized in the following well-known lemma.

Lemma 1 (Stability of forward Euler method). *The forward propagation in equation (2.1) is stable if*

$$\max_{i=1,2,\ldots,n} |1 + h\lambda_i(\mathbf{J}_j)| \leqslant 1, \quad \forall j = 0, 1, \ldots, N-1. \tag{3.9}$$

Proof. See, e.g. [2, 4] for the proof of stability criteria for the forward Euler method. □

The above discussion suggests that the stability of the continuous forward propagation in equation (3.8) and its discrete analog equation (3.9) need to be added to the optimization problem 2.4 as constraints. Otherwise, one may obtain some transformation weights, $\mathbf{K}_0, \mathbf{K}_1, \ldots, \mathbf{K}_{N-1}$, that may fit the training data but generate an unstable process. As discussed above these solutions cannot be expected to generalize well for other data.

We illustrate the stability issues in ResNet using a simple example.

Example 1 (Stability of ResNet). For $s = 3$ and $n = 2$ we consider the forward propagation through a ResNet as given by equation (2.1). We consider three networks consisting of $N = 10$ identical layers, i.e. on each layer we use the activation $\sigma_{\mathrm{ht}} = \tanh$, $h = 0.1$, $b = 0$, and a constant weight matrix. To illustrate the impact of the eigenvalues of the weight matrix on the propagation, we consider three ResNets parameterized by

$$\mathbf{K}_+ = \begin{pmatrix} 2 & -2 \\ 0 & 2 \end{pmatrix}, \ \mathbf{K}_- = \begin{pmatrix} -2 & 0 \\ 2 & -2 \end{pmatrix}, \ \mathbf{K}_0 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \tag{3.10}$$

where $\lambda_1(\mathbf{K}_+) = \lambda_2(\mathbf{K}_+) = 2$, $\lambda_1(\mathbf{K}_-) = \lambda_2(\mathbf{K}_-) = -2$ and $\lambda_1(\mathbf{K}_0) = \imath$, $\lambda_2(\mathbf{K}_0) = -\imath$. We consider the feature vectors $\mathbf{y}_1 = [0.1, 0.1]^\top$, $\mathbf{y}_2 = -\mathbf{y}_1$, $\mathbf{y}_3 = [0, 0.5]^\top$. After propagating the features through the layers, we illustrate the different propagations in figure 1. We represent the values at the hidden layers as colored lines in the 2D plane where each color is associated with one feature vector. To highlight the differences in the dynamics in all three cases, we also depict the force field using black arrows in the background. This plot is often referred to as the *phase plane diagram*.

As can be seen in left subplot, the features diverge away from the origin and each other using $\mathbf{K}_+$. Note that $\mathbf{y}_1$ and $\mathbf{y}_2$, which are close together initially, depart into opposite directions. This clearly suggests an unstable forward propagation that cannot be expected to generalize well. In contrast to that, the center subplot shows that $\mathbf{K}_-$ yields an accumulation point at the

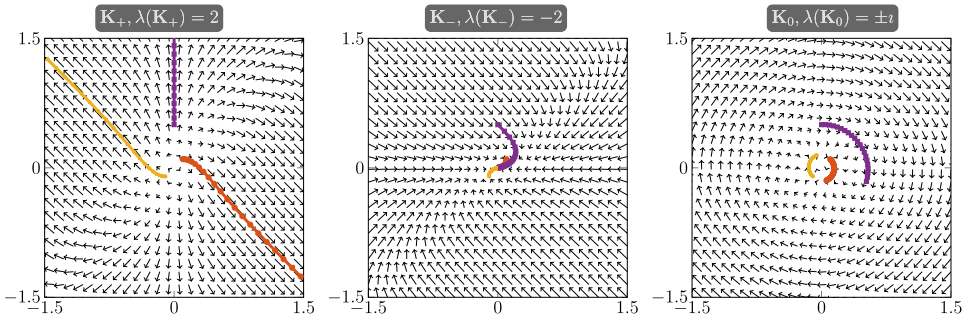

Figure 1. Phase plane diagrams for ResNets with $N = 10$ identical layers parameterized by the weight matrices in equation (3.10) starting from three different input features. The values at hidden layers are indicated by colored lines and arrows depict the force field. Left: due to its positive eigenvalues, the features diverge using $\mathbf{K}_+$ leading to an unstable forward propagation. Center: $\mathbf{K}_-$ yields a contraction that annihilates differences in the features and renders the learning problem ill-posed. Right: the antisymmetric matrix $\mathbf{K}_0$ leads to rotations, preserves distances between the features, and yields well-posed forward propagation and learning.

origin. While the forward propagation satisfies equation (3.8) and is thus stable, the learning problem, which requires inversion of this process, is ill-posed. Finally, the antisymmetric matrix $\mathbf{K}_0$ yields a rotation in the feature space, which preserves the distances between the features and leads to a stable forward propagation and a well-posed learning problem.

Clearly the effects depicted here are more pronounced in deeper networks with more layers and/or larger values for h.

While the first case in example 1 indicates that equation (3.8) is a necessary condition for successful learning, the second case suggests that it is not sufficient. In general, when $\mathrm{Re}(\lambda(\mathbf{K}(t))) < 0$ for most times and the network is deep (e.g. long time integration) differences in the initial feature vectors decay. In other words, for all initial conditions $\mathbf{y}_0$ we have that $\mathbf{y}(t) \to 0$ as $t \to \infty$; compare with center plot in figure 1. Hence, even though the forward problem is stable the inverse problem is highly ill-posed as it comparable to an inverse heat equation. In these situations, the gradients of the objective function in equation (2.4) will vanish since small changes in the kernels will have no noticeable impact on the values of the outputs.

If we are inspired from the propagation of signals through neurons, then a ResNet as in equation (2.4) with $\max_i \mathrm{Re}\,(\lambda_i(\mathbf{K})) > 0$ for all i consists of neurons that amplify the signal with no upper bound (which is not biological) and a ResNet with $\max_i \mathrm{Re}\,(\lambda_i(\mathbf{K})) \ll 0$ can be seen as a lossy network. A moderately lossy network may be advantageous when the input is noisy, since it tends to decay high order oscillations. However, having too much signal loss is clearly harmful as it also annihilates relevant differences in the input features.

In summary, our discussion of ResNets illustrates that stable forward propagation and well-posed learning problems can be obtained for deep networks when

$$\mathrm{Re}\,(\lambda_i(\mathbf{K}(t))) \approx 0, \quad \forall i = 1, 2, \ldots, n, \ t \in [0, T]. \tag{3.11}$$

In this case, the forward propagation causes only moderate amplification or loss and thus even deep networks preserve features in the input data and allow for effective learning.

4. Stable forward propagation for DNNs

Motivated by the discussion in the previous section, we introduce three new forward propagation methods that are stable for arbitrarily DNNs and lead to well-posed learning problems. Our approaches, presented in sections 4.1 and 4.2, use different means to enforce Jacobians whose eigenvalues have very small real part; see equation (3.11). The methods in section 4.2 are inspired by Hamiltonian systems and we propose leapfrog and Verlet integration techniques for forward propagation in section 4.3. Finally, we compute the derivative of the Verlet method using back propagation [46] and discuss the relation between back propagation and the older and more general adjoint method [32] in section 4.4.

4.1. Antisymmetric weight matrices

Perhaps the simplest way to obtain a stable forward propagation is to construct force fields whose Jacobians are antisymmetric. For example, consider the forward propagation

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma\left(\frac{1}{2}\mathbf{Y}_j\left(\mathbf{K}_j - \mathbf{K}_j^\top - \gamma\mathbf{I}\right) + b_j\right),\ j = 0,\ldots,N-1, \tag{4.12}$$

where $\gamma \geqslant 0$ is a small constant and $\mathbf{I}$ is an identity matrix. The resulting forward propagation can be seen as a forward Euler discretization of the ODE

$$\dot{\mathbf{y}}(t) = \sigma\left(\frac{1}{2}(\mathbf{K}(t) - \mathbf{K}(t)^\top - \gamma\mathbf{I})\mathbf{y}(t) + b(t)\right), \quad \forall t \in [0,T].$$

Since $\mathbf{K}(t) - \mathbf{K}(t)^\top$ is anti-symmetric its eigenvalues are imaginary, which also holds for the Jacobian in equation (3.7). Thus, the continuous ResNet parameterized by the antisymmetric weight matrix is stable and preserves information given an appropriate integration technique and sufficiently small time steps.

To stabilize the discrete forward propagation, which is based on the forward Euler method, we have added diffusion to the system in equation (4.12). The amount of diffusion depends on the parameter $\gamma \geqslant 0$. While small values of γ might improve the robustness against noise in the feature vectors, too large values can render the learning problem ill-posed. Alternatively, more advanced time integration methods can be used with $\gamma = 0$, i.e. without adding diffusion.

4.2. Hamiltonian inspired neural networks

Restricting the parameter space to antisymmetric kernels is only one way to obtain a stable forward propagation. Alternatively we can recast forward propagation as a Hamiltonian system, which has the structure

$$\dot{\mathbf{y}}(t) = -\nabla_{\mathbf{z}} H(\mathbf{y},\mathbf{z},t) \quad \text{and} \quad \dot{\mathbf{z}}(t) = \nabla_{\mathbf{y}} H(\mathbf{y},\mathbf{z},t), \quad \forall t \in [0,T].$$

The function $H : \mathbb{R}^n \times \mathbb{R}^n \times [0,T] \to \mathbb{R}$ is the Hamiltonian. In our application Hamiltonian systems for forward propagation are attractive due to their property to conserve rather than increase or dissipate the energy of the system; see [2, chapter 6] for a general introduction.

The Hamiltonian function H measures the energy of the system, which in the autonomous case gets conserved. A simple approach can be derived from the Hamiltonian

$$H(\mathbf{y},\mathbf{z}) = \frac{1}{2}\mathbf{z}^\top\mathbf{z} + f(\mathbf{y}),$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is at least twice continuously differentiable. This leads to the ODE system

$$\dot{\mathbf{z}}(t) = -\nabla_{\mathbf{y}} f(\mathbf{y}(t)), \quad \dot{\mathbf{y}}(t) = \mathbf{z}(t), \quad \forall t \in [0, T].$$

Eliminating $\mathbf{z}$ we obtain a second order system

$$\ddot{\mathbf{y}}(t) = \nabla_{\mathbf{y}} f(\mathbf{y}(t)), \quad \forall t \in [0, T].$$

Inspired by the continuous version of the ResNet forward propagation given in equation (3.5) we propose to use the following second order ODE

$$\ddot{\mathbf{y}}(t) = \sigma(\mathbf{K}^\top(t)\mathbf{y}(t) + b(t)), \quad \mathbf{y}(0) = \mathbf{y}_0, \quad \dot{\mathbf{y}}(0) = \dot{\mathbf{y}}_0, \tag{4.13}$$

where in the following we assume that $\dot{\mathbf{y}}_0 = 0$. The dynamical system in equation (4.13) is stable for all weight matrices with non-positive real eigenvalues, i.e. that satisfy equation (3.8). This constraint can either be added to the optimization problem 2.4 or, similar to the previous section, be enforced by design. The latter approach can be obtained, e.g. using the parametrization

$$\mathbf{K}(\mathbf{C}) = -\mathbf{C}^\top\mathbf{C}, \quad \text{for some} \quad \mathbf{C} \in \mathbb{R}^{n\times n}. \tag{4.14}$$

Note that in contrast to the antisymmetric model in equation (4.12), the negative definite model is based on a nonlinear parametrization which might lead to a more challenging optimization problem. Alternatively equation (3.8) can be added as a constraint to the optimization problem 2.4 requiring expensive eigenvalue computations.

The forward propagations equations (2.1), (4.12), or (4.13) require additional constraints on $\mathbf{K}$ and are limited to square weight matrices. We therefore propose a network that is *intrinsically* stable and supports non-square weight matrices, i.e. a network whose forward propagation is stable independent of the choice of the weights. To this end, we introduce symmetry into the Hamiltonian system by defining

$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}(t)\mathbf{z}(t) + b(t)) \;\text{ and }\; \dot{\mathbf{z}}(t) = -\sigma(\mathbf{K}(t)^\top\mathbf{y}(t) + b(t)). \tag{4.15}$$

Note that equation (4.15) can be seen as a special case of ResNet with an augmented variable $\mathbf{z} \in \mathbb{R}^m$ and the associated ODE

$$\frac{\partial}{\partial t}\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}(t) = \sigma\left(\begin{pmatrix}0 & \mathbf{K}(t)\\ -\mathbf{K}(t)^\top & 0\end{pmatrix}\begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}(t) + b(t)\right), \quad \begin{pmatrix}\mathbf{y}\\ \mathbf{z}\end{pmatrix}(0) = \begin{pmatrix}\mathbf{y}_0\\ 0\end{pmatrix}.$$

This system is stable regardless of the spectrum or size of the matrices $\mathbf{K}(t)$ since the overall matrix in the linear transformation is antisymmetric. To ensure stability of the discrete version the time step size needs to be sufficiently small; see lemma 3.9.

4.3. Symplectic forward propagation

In this section we use symplectic integration techniques for solving the discrete versions of the Hamiltonian-inspired networks in equations (4.13) and (4.15). Symplectic methods have been shown to capture the long time features of Hamiltonian systems and thus provide a slightly different approach as compared to the forward Euler method used in ResNet (2.4); we refer to [2, chapter 6] for a detailed discussion. For the second-order ODE in equation (4.13) we recommend the conservative leapfrog discretization

$$\mathbf{y}_{j+1} = \begin{cases} 2\mathbf{y}_j + h^2\sigma(\mathbf{K}_j\mathbf{y}_j + b_j), & j = 0 \\ 2\mathbf{y}_j - \mathbf{y}_{j-1} + h^2\sigma(\mathbf{K}_j\mathbf{y}_j + b_j), & j = 1, 2, \ldots, N-1 \end{cases}. \tag{4.16}$$

To discretize the augmented network in equation (4.15) we use a Verlet integration where for $j = 0, 1, \ldots, N-1$ we have

$$\mathbf{z}_{j+\frac{1}{2}} = \mathbf{z}_{j-\frac{1}{2}} - h\sigma(\mathbf{K}_j^\top \mathbf{y}_j + b_j) \text{ and } \mathbf{y}_{j+1} = \mathbf{y}_j + h\sigma(\mathbf{K}_j\mathbf{z}_{j+\frac{1}{2}} + b_j). \tag{4.17}$$

Since both discretizations are *symplectic*, the respective Hamiltonians are preserved if the transformation weights are time invariant and the step size is sufficiently small.

We demonstrate the long time dynamics of the two new Hamiltonian-inspired forward propagation methods using a simple example with two-dimensional features and identical layers.

Example 2. Let $s = 2$ and $n = 2$ and consider the features $\mathbf{y}_1 = [0.1, 0.1]^\top$ and $\mathbf{y}_2 = [-0.1, -0.1]^\top$. We consider networks with identical layers featuring $b = 0$ and hyperbolic tangent as activation function. For the leapfrog integration we use the matrix $\mathbf{K} = \mathbf{K}_-$ defined in equation (3.10) (recall that $\lambda_1(\mathbf{K}) = \lambda_2(\mathbf{K}) = -2$) and a step size of $h = 1$. To illustrate the Verlet integration (which can handle non-square kernels) we use a time step size of $h = 0.1$ and

$$\mathbf{K}_v = \begin{pmatrix} 2 & 1 \\ -1 & 2 \\ 0 & 1 \end{pmatrix}.$$

To expose the short term behavior we use $N = 500$ layers and to demonstrate the non-trivial long term characteristics we use $N = 5000$ layers. We depict the phase plane diagrams for both networks and both depths in figure 2. Even though we use identical layers with constant kernels the features neither explode nor vanish asymptotically.

We proposed three new ways of forward propagation to ensure stability for arbitrary numbers of layers. Among the three approaches, the Verlet method is the most flexible as it can handle non-square weighting matrices and does not require additional constraints on $\mathbf{K}$.

4.4. Derivatives of the Verlet method

We now discuss the computation of derivatives for the Verlet method. The derivatives of the ResNet are standard and can be found, e.g. in [28]. Given the ResNet derivatives the anti-symmetric model can be differentiated using chain rule and the time stepping. The leapfrog method can be treated in a similar way to usual ResNets and is therefore omitted.

Our presentation follows the standard approach used in ML known as *back propagation* [28]. However, we note that back propagation is a special case of the *adjoint method* [7] used in time-dependent optimal control; see, e.g. [9, 16]. The adjoint method is more general than back propagation as it can also be used compute the gradient of less 'standard' time stepping methods. Thus, we discuss the back propagation method in the context of the adjoint method as applied to the Verlet method. We start by the usual sensitivity equation and then discuss how to efficiently compute matrix-vector products.

Exemplarily we show how to differentiate the values at the output layer in equation (4.17) with respect to the weight matrix $\mathbf{K}_k$ at an arbitrary layer k. For simplicity we assume that $\mathbf{K}$ is square. Derivatives for non-square weight matrices and with respect to the bias, b_k, can be

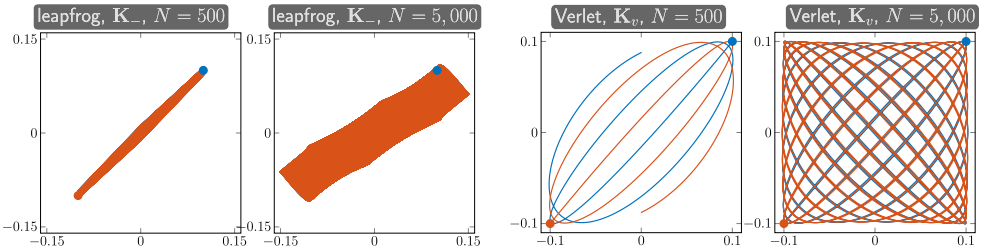

Figure 2. Phase space diagrams for the forward propagation using the leapfrog and Verlet methods starting from $\mathbf{y}_1 = [0.1, 0.1]^\top$ (blue) and $\mathbf{y}_2 = [-0.1, -0.1]^\top$ (red). For each network we show the short term ($N = 500$) and long term ($N = 5000$) behavior for identical layers. In both cases non-trivial behavior can be observed even for constant weight matrices (This figure is optimized for screen use as provided in the online version of this journal.).

computed along the same lines. Clearly, the values of $\mathbf{z}_j$ and $\mathbf{y}_j$ at the hidden layers $j \leqslant k$ are independent of $\mathbf{K}_k$. Applying the chain rule to equation (4.17) we see that

$$\frac{\partial \mathbf{z}_{k+\frac{1}{2}}}{\partial \mathbf{K}_k} = h\text{diag}\left(\sigma'(\mathbf{K}_k\mathbf{y}_k + b_k)\right)(\mathbf{y}_k \otimes \mathbf{I}) =: \mathbf{C}_1 \tag{4.18}$$

$$\begin{aligned}\frac{\partial \mathbf{y}_{k+1}}{\partial \mathbf{K}_k} &= -h\text{diag}\left(\sigma'(-\mathbf{K}_k^\top \mathbf{z}_{k+\frac{1}{2}} + b_k)\right)(\mathbf{I} \otimes \mathbf{Z}_{k+\frac{1}{2}} + \mathbf{K}_k^\top \otimes \mathbf{I}) \\ &=: \mathbf{C}_2,\end{aligned} \tag{4.19}$$

where $\otimes$ denotes the Kronecker product and $\mathbf{I}$ is the $n \times n$ identity matrix. We can now differentiate layer by layer, where for $k < j < N - 1$ we obtain

$$\frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k} = \frac{\partial \mathbf{z}_{j-\frac{1}{2}}}{\partial \mathbf{K}_k} + h\text{diag}\left(\sigma'(\mathbf{K}_j\mathbf{y}_j + b_j)\right)\mathbf{K}_j \frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k} \tag{4.20}$$

$$\frac{\partial \mathbf{y}_{j+1}}{\partial \mathbf{K}_k} = \frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k} - h\text{diag}\left(\sigma'(-\mathbf{K}_j^\top \mathbf{z}_{j+\frac{1}{2}} + b_j)\right)\mathbf{K}_j^\top \frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k}. \tag{4.21}$$

Combining equations (4.18)–(4.20) the derivatives can be written compactly as a block linear system

$$\begin{pmatrix} \mathbf{I} & & & & & & \\ & \mathbf{I} & & & & & \\ -\mathbf{I} & \mathbf{B}_{k+1} & \mathbf{I} & & & & \\ & -\mathbf{I} & \mathbf{A}_{k+1} & \mathbf{I} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -\mathbf{I} & \mathbf{A}_N & \mathbf{I} & \\ & & & & -\mathbf{I} & \mathbf{B}_N & \mathbf{I} \end{pmatrix} \begin{pmatrix} \frac{\partial \mathbf{z}_{k+\frac{1}{2}}}{\partial \mathbf{K}_k} \\ \frac{\partial \mathbf{y}_{k+1}}{\partial \mathbf{K}_k} \\ \vdots \\ \frac{\partial \mathbf{z}_{N+\frac{1}{2}}}{\partial \mathbf{K}_k} \\ \frac{\partial \mathbf{y}_{N+1}}{\partial \mathbf{K}_k} \end{pmatrix} = \begin{pmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{4.22}$$

where

$$\begin{aligned}\mathbf{B}_j &= -h\text{diag}\left(\sigma'(\mathbf{K}_j\mathbf{y}_j + b_j)\right)\mathbf{K}_j, \\ \mathbf{A}_j &= h\text{diag}\left(\sigma'(-\mathbf{K}_j^\top \mathbf{z}_{j+\frac{1}{2}} + b_j)\right)\mathbf{K}_j^\top.\end{aligned}$$

The linear system in equation (4.22) is block triangular with identity matrices on its diagonal and thus can be solved explicitly using forward substitution. Such systems commonly arise in optimal control of time-dependent PDEs and in dynamic inverse problems. For more details involving notes on implementation see, e.g. [45].

In the optimal control literature the matrices $\frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k}$ and $\frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k}$ are often referred to as the sensitivity matrices. While the matrices can be dense and large, computing matrix-vector products can be done efficiently by forward propagation. To this end, the right hand side of equation (4.22) is multiplied by the vector and the linear system is then solved with a vector right hand side. The multiplication with the transpose, also known as the adjoint method, is done by solving the equation backwards. It is important to note that the back-propagation algorithm [46] is nothing but a particular implementation of the adjoint method discussed much earlier in [32].

5. Regularization

In this section we present derivative-based regularization functions and a multi-level learning approach to ensure smoothness of parameters in deep learning. By biasing the learning process towards smooth time dynamics we aim at improving the stability of the forward propagation (e.g. to ensure that $\mathbf{K}(t)$ changes sufficiently slow as assumed in section 3) and ultimately the generalization. Intuitively, we cannot expect the network to generalize well in the absence of smoothness. While the presented regularization techniques are commonly employed in other areas of inverse problems [26, 48], e.g. imaging their application to deep learning is, to the best of our knowledge, rather novel.

5.1. Regularizing the forward propagation

The perhaps most common regularization strategy used in deep learning is referred to as *weight decay*, which is equivalent to standard Tikhonov regularization of the form

$$R(\mathbf{K}) = \frac{1}{2}\|\mathbf{K}\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm; see, e.g. [22]. While this regularization reduces the magnitude of the weights, it is not sensitive to rapidly changing weights between adjacent layers. To illustrate why this may be problematic, consider removing a single layer from a deep network. Since the network is deep, we should not expect large changes in the values of the output layer and thus similar classification errors. However, if this layer performs, e.g. a 90 degree rotation of the features while the adjacent layers keep the features unchanged, the effect will be dramatic.

Our interpretation of forward propagation as a system of non-autonomous ODEs and the stability analysis in section 3 motivate that $\mathbf{K}$ should be smooth, or at least piecewise smooth in time. To this end we propose the following new regularization for the transformation weights

$$R(\mathbf{K}) = \frac{1}{2h}\sum \|\mathbf{K}_j - \mathbf{K}_{j-1}\|_F^2 \text{ and } R(b) = \frac{1}{2h}\sum (b_j - b_{j-1})^2. \tag{5.23}$$

This regularization favors weights that vary smoothly between adjacent layers. Furthermore, as we see in our numerical experiments, the regularization adds robustness to the process. We can easily add or subtract steps without significantly changing the final result, thus adding more generalizing power to our network.

5.2. Regularizing the classification weights

We propose using smoothness regularization on the classification weights for image classification problems, e.g. in convolution neural networks (CNN); see; e.g. [22, chapter 9]. For motivation, let the examples in $\mathbf{Y}_0$ represent vectorized $n_1 \times n_2 \times n_3$ images. The network propagates the images in time and generates perturbed images of the same size. The hypothesis function predicts the class label probabilities based on affinely transformed output images, i.e.

$$\mathbf{C}^{\text{pred}} = \mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top).$$

The number of rows in $\mathbf{W}$ is $n = n_1 \cdot n_2 \cdot n_3$ and the operation $\mathbf{Y}_N(j,:)^\top\mathbf{W}(:,k)$ is a dot product between the jth output image and the classification weights for the kth class. Noting that the rows of $\mathbf{Y}_N$ and the columns of $\mathbf{W}$ can be interpreted as discrete images sets the stage for developing regularization approaches commonly used in image processing; see, e.g. [17].

To further motivate the importance of regularization, note that if the number of examples is much larger than the number of features (pixels in the image) and if there is no significant redundancy, finding the optimal $\mathbf{W}$ given $\mathbf{Y}_N$ and $\mathbf{C}$ is an over-determined problem. Otherwise the problem is ill-posed and there may be infinitely many weights that yield the same classification on the observed data. The risk in both cases is *overfitting* since the optimal classification weights can be highly irregular and generalize poorly. Thus, one way to enforce uniqueness and improve generalization is to regularize the weights.

A standard approach in ML also known as *pooling* aims at achieving this goal; see, e.g. [22]. The main idea of pooling is to coarsen the output images and thereby to decrease the dimensionality of the classification problem. The simplest approach is skipping, i.e. subsampling the image $\mathbf{Y}_N$, e.g. at every other pixel. Other common options are average pooling and max-pooling, where image patches are represented by their average and maximum value, respectively. From an inverse problems perspective, pooling can be thought of as a subspace regularization [26]. For example, average pooling is similar to requiring $\mathbf{W}$ to be constant over each image patch.

An alternative approach to regularization is to interpret the jth feature, $\mathbf{y}_j \in \mathbb{R}^n$, and the kth classification weight, $\mathbf{w}_k \in \mathbb{R}^n$, of the CNN as discretizations of image functions $y_j : \Omega \to \mathbb{R}$ and $w_k : \Omega \to \mathbb{R}$, respectively, where $\Omega \subset \mathbb{R}^2$ is the image domain. Assuming y_j is sufficiently regular (which is to be enforced by the regularization) we can see that the probability of the jth example belonging to the kth class can be viewed as

$$\mathbf{h}(\mathbf{y}_j^\top\mathbf{w}_k + \mu_k) \approx \mathbf{h}\left(\text{vol}(\Omega)\int_\Omega y(x)w(x)\,\mathrm{d}x + \mu_k\right), \tag{5.24}$$

where $\text{vol}(\Omega)$ denotes the volume of the image domain. To obtain weights that are insensitive to small displacements of the images it is reasonable to favor spatially smooth parameters by using regularization of the form

$$R(\mathbf{w}_k) = \frac{1}{2}\|\mathbf{L}\mathbf{w}_k\|^2, \tag{5.25}$$

where $\mathbf{L}$ is a discretized differential operator. This regularization also embeds the optimal w_k into a suitable function space. For example, using an image gradient ensures that w_k is in the Sobolev space $H^1(\Omega, \mathbb{R})$ [20]. Most importantly for our application at hand, derivative-based regularization yields smooth classification weights that, as we see next, can be interpreted by visual inspection.

5.3. Multi-level learning

As another means of regularizing the problem, we exploit a multi-level learning strategy that gradually increases the number of layers in the network. Our idea is based on the continuous interpretation of the forward propagation in which the number of layers in the network corresponds to the number of discretization points. Our idea is closely related to cascadic multigrid methods [8] and ideas in image processing where multi-level strategies are commonly used to decrease the risk of being trapped in local minima; see, e.g. [39]. More details about multi-level methods in learning can be found in [25].

The basic idea is to first solve the learning problem using a network with only a few layers and then prolongate the estimated weights of the forward propagation to a network with twice as many layers. The prolongated weights are then used to initialize the optimization problem on the finer level. We repeat this process until a user-specified maximum number of layers is reached.

Besides realizing some obvious computational savings arising from the reduced size of the networks, the main motivation behind our approach is to obtain good starting guesses for the next level. This is key since, while deeper architectures offer more flexibility to model complicated data-label relation, they are in our experience more difficult to initialize. Additionally, the Gauss–Newton method used to estimate the parameters of the forward propagation benefits in general from good starting guesses.

6. Numerical examples

In this section we present numerical examples for classification problems of varying levels of difficulty. We begin with three examples aiming at learning classification functions in two variables, which allow us to easily assess and illustrate the performance of the DNNs. In section 6.4 we show results for the MNIST data set [36, 37], which is a common benchmark problem in image classification.

6.1. Concentric ellipses

As a first test we consider a small-scale test problem in two dimensions. The test data consists of 1200 points that are evenly divided into two groups that form concentric ellipsoids; see left subplot in figure 4. The original data is randomly divided into 1000 training examples and 200 examples used for validation.

We train the original ResNet, the antisymmetric ResNet, and the Hamiltonian network with Verlet propagation using the block coordinate descent method and a multi-level strategy with $4, 8, 16, 32, 64, 128, 256, 1024$ layers. Each block coordinate descent iteration consists of a classification step ($\leqslant$2 iterations of Newton-PCG with $\leqslant$ 2 PCG iterations) and a Gauss–Newton-PCG step to update the propagation weights ($\leqslant$20 PCG iterations preconditioned by regularization operator). In all examples we use the logistic regression hypothesis function in equation (2.2) and $\tanh$ activation function. The final time is $T = 20$ and the width of the network is $n = 2$. To enforce smoothness of the propagation weights in time we employ the regularizer in equation (5.23) weighted by a factor of $\alpha = 10^{-3}$. No regularization is used in the classification.

We show the performance of the multi-level scheme in the left subplot of figure 3. For this simple data set, all forward propagation methods achieve an optimal validation accuracy of 100% at some level. As to be expected, the validation accuracy increases with increasing

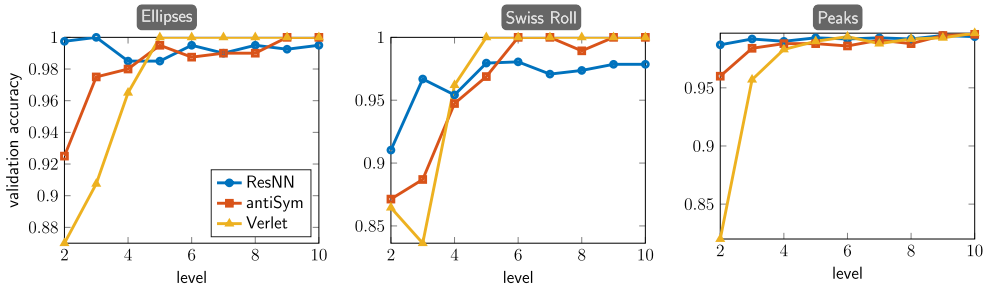

Figure 3. Multi-level convergence for the two-dimensional examples in sections 6.1–6.3. For each level (level ℓ corresponds to DNN with 2^ℓ layers) we show the best validation accuracy (the optimal value is 1, which corresponds to a validation error of 0%).

depth of the network, which also results in more degrees of freedom. The results for the Verlet method at the final level are shown in figure 4. The two steps of deep learning (propagation and classification) can be seen in the center plot. The propagation transforms the feature such that they can be linearly separated. The result of the learning process is a network that predicts the class for all points in the 2D plane, which is illustrated in the right subplot of figure 4.

6.2. Swiss roll

We consider another small-scale test problem that is inspired by the swiss roll example. The data is obtained by sampling the vector functions

$$f_1(r,\theta) = r\begin{pmatrix}\cos(\theta)\\ \sin(\theta)\end{pmatrix}, \quad \text{and} \quad f_2(r,\theta) = (r+0.2)\begin{pmatrix}\cos(\theta)\\ \sin(\theta)\end{pmatrix},$$

for $r \in [0,1]$ and $\theta \in [0, 4\pi]$ at 513 points each. Every other point along the curve is removed from the data set and used for validation; see left subplot in figure 5.

Given the data we use the same parameterization of the multi-level and block coordinate descent method as in the previous example. For all networks, except the Hamiltonian network with Verlet forward propagation, we increase the dimensionality of the input feature space to $n = 4$. As before we use the tanh activation, logistic regression function in equation (2.2), and choose a final time of $T = 20$. To enforce smoothness of the propagation weights in time we employ the regularizer in equation (5.23) weighted by a factor of $\alpha = 5 \cdot 10^{-3}$. No regularization is used in the classification.

We plot the validation accuracy for each network and the different steps of the multi-level strategy in the center subplot of figure 3. The standard ResNet and the Hamiltonian NN with Verlet forward propagation both achieve an optimal validation accuracy for $N = 1024$. However, the convergence considerably faster for the Hamiltonian network that reaches the optimal accuracy with $N = 32$ layers. We visualize the results obtained using the Verlet method in figure 5.

6.3. Peaks

We propose a new challenging test problem for classification into multiple classes using the `peaks` function in MATLAB®, which reads,

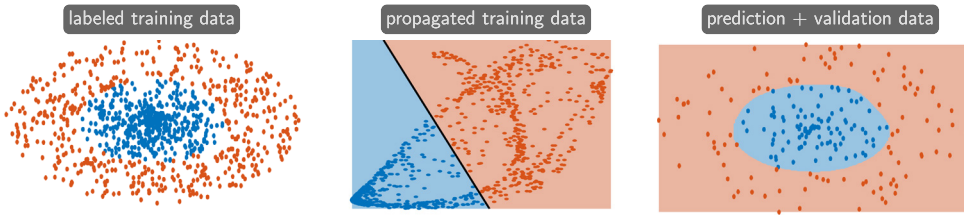

Figure 4. Classification results for the ellipse problem from section 6.1 using a Hamiltonian neural network with Verlet propagation with $N = 1024$ layers. Left: labeled input features representing two concentric ellipses that are not linearly separable. Center: the output features are linearly separable and can be accurately classified. Right: we show the predictions of the network (colors in the background) superimposed by the validation data (This figure is optimized for screen use as provided in the online version of this journal.).

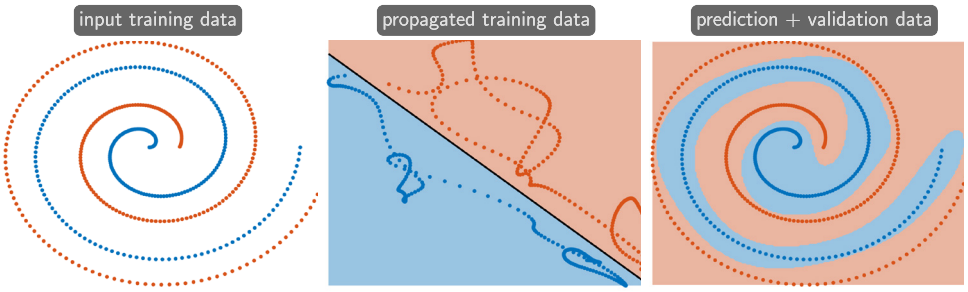

Figure 5. Classification results for swiss roll example described in section 6.2 using a Hamiltonian network with $N = 1024$ layers and the Verlet forward propagation. Left: we show the training data (red and blue dots). Center: the Verlet method propagates the features so that they can be separated by the affine linear logistic regression classifier. Right: we show the interpolation obtained by our network (colored background) and the validation data (This figure is optimized for screen use as provided in the online version of this journal.).

$$f(x) = 3(1-x_1)^2 \exp(-(x_1^2) - (x_2+1)^2) - 10(x_1/5 - x_1^3 - x_2^5) \\ \exp(-x_1^2 - x_2^2) - 1/3 \exp(-(x_1+1)^2 - x_2^2),$$

where $x \in [-3,3]^2$. The peaks function is smooth but has some nonlinearities and most importantly non-convex level sets. We discretize the function on a regular 256×256 grid and divide the points into 5 different classes based on their function value. The points in each class are then randomly subsampled such that the training data approximately represents the volumes of the level sets. In our case the $s = 5000$ sample points are divided evenly into five classes of size 1000. We illustrate the test data in the left subplot of figure 6.

We randomly choose 20% of the data for validation and train the networks using the remaining examples. We use tanh as activation, the softmax hypothesis function in equation (2.3), and choose a final time of $T = 5$. To enforce smoothness of the propagation weights in time we employ the regularizer in equation (5.23) weighted by a factor of $\alpha = 5 \cdot 10^{-6}$. No regularization is used in the classification. We use the same multi-level strategy and identical parameters in the block coordinate descent methods as in the previous examples.

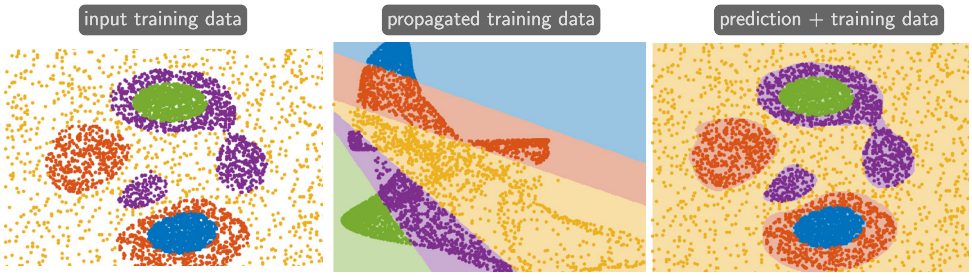

Figure 6. Classification results for peaks example described in section 6.3 using a Hamiltonian network with $n = 1024$ layers and the Verlet forward propagation. Left: we illustrate the training data by colored dots that represent the class. Center: we show the propagated features and the predictions of the softmax classifier. Right: we depict the predictions of our network (colored background) and the training data (This figure is optimized for screen use as provided in the online version of this journal.).

We first train the standard ResNet and the antisymmetric variant where we use a width of $n = 8$ by duplicating the features. The performance of both models is approximately the same; see right subplot in figure 3. The optimal accuracy, achieved for $N = 1024$, is around 98.8%.

For the Hamiltonian network with Verlet forward propagation we use a narrower network containing only the original features (i.e. $n = 2$ and no duplication). As in the previous experiments the accuracy generally improves with increasing depth (with one exception between levels 4 and 5). The optimal accuracy is obtained at the final level ($N = 1024$) with a validation error of 99.1%. We illustrate the results for the Verlet network in figure 6. The center subplot shows how the forward propagation successfully rearranges the features such that they can be labeled using a linear classifier. The right subplot shows that the prediction function fits the training data, but also approximates the true level sets.

6.4. MNIST

We use the MNIST dataset of hand-written digits [36, 37] to illustrate the applicability of our methods in image classification problems. The data set consists of 60 000 labeled digital images of size 28×28 showing hand written digits from 0 to 9. We randomly divide the data into 50 000 training and 10 000 validation examples. We train the network using the standard and antisymmetric ResNet and the first-order Hamiltonian network using Verlet integration.

We use a three-level multi-level strategy where the number of layers is 4, 8, and 16. In each step we use the block coordinate descent method to approximately solve the learning problem and prolongate the forward propagation parameters to the next level using piecewise linear interpolation. The width of the network is 6 (yielding $n = 4704$ features used in the classification) and we use 3×3 convolution operators that are fully connected within a given layer to model the linear transformation matrices $\mathbf{K}_0, \ldots, \mathbf{K}_{N-1}$. The final time is set to $T = 6$. To compute the Gauss–Newton step we first compute the full gradient over all 50 000 examples and then randomly subsample 5000 terms for Hessian computations in the PCG step. The maximum number of iterations is set to 20 at each layer.

Within each step of the block coordinate descent we solve the classification problem using at most 5 iterations of Newton-PCG with up to 10 inner iterations. We use a discretized Laplacian as a regularization operator in equation (5.25) and its shifted symmetric product as a preconditioner to favor smooth modes; see [15]. The regularization parameter used in

Table 1. Multi-level training result for MNIST data set (see section 6.4). We show the training error (TE) and validation error (VE) for the standard and antisymmetric ResNet and the Hamiltonian inspired network with Verlet forward propagation for increasing number of layers in the network. The different forward propagation methods yield comparable results with the antisymmetric ResNet giving slightly lower validation errors at each level.

	ResNet		Anti symmetric ResNet		Hamiltonian Verlet	
Layers	TE (%)	VE (%)	TE (%)	VE (%)	TE (%)	VE (%)
4	0.96	1.71	1.13	1.70	1.49	2.29
8	0.80	1.59	0.92	1.46	0.82	1.60
16	0.73	1.53	0.91	1.38	0.35	1.58

the classification is 0.01. The output values for all examples are used at the final layer and no pooling is performed.

Smooth time dynamics are enforced by penalizing the time derivatives as outlined in section 5.1 with a manually calibrated regularization parameter of $\alpha = 0.005$. The regularization operator is used to precondition the PCG iterations in the Gauss–Newton method.

To show the performance of the multi-level strategy, we summarize the training and validation errors at the respective levels in table 1. The table shows similar performance for all forward propagation methods. Note that both the validation and training error are reduced for larger number of layers, but no overfitting is observed. In this experiment, the multi-level approach considerably simplified the initialization of the deep networks. For example, the initial parameters of the standard ResNet at the final level ($N = 16$ layers) already gave a validation accuracy of 98.41%, which was improved to 98.47%. We illustrate the results of the antisymmetric ResNet, which yields slightly superior performance in terms of validation error, in figure 7. The smoothness of the classification weights enforced by our regularizer can be seen in the right column.

7. Summary and conclusions

In this paper, we expose the relationship between deep learning and dynamic inverse problems and lay the foundation for fruitful research at the interface of inverse problems and data science. Specifically, we propose new architectures for deep neural networks (DNN) that improve the stability of the forward propagation. Using derivative-based regularization, we also improve the well-posedness of the learning task. We also propose a multi-level strategy that simplifies the choice of initial parameters and thereby simplifies the training of very deep networks. Our forward propagation methods are inspired by Hamiltonian systems and motivated by a stability analysis that is based on a continuous formulation of the learning problem.

Our stability discussion in section 3 provides new insights why ResNets [27], a commonly used forward propagation scheme in deep learning, can be unstable. Our result is based on a continuous interpretation of a simplified version of ResNets as a system of nonlinear ODEs and a standard stability argument. Using intuitive examples, we show the impact of the spectral properties of the transformation matrices on the stability of the forward propagation and state conditions under which gradients explode, vanish, or are stable. We also note that stability depends on the smoothness of the network parameters over time. Based on our findings and numerical experience, we argue that it is desirable to restrict ResNets to matrices that lead to Jacobians whose real parts of the spectrum are close to zero and penalize the time-derivative

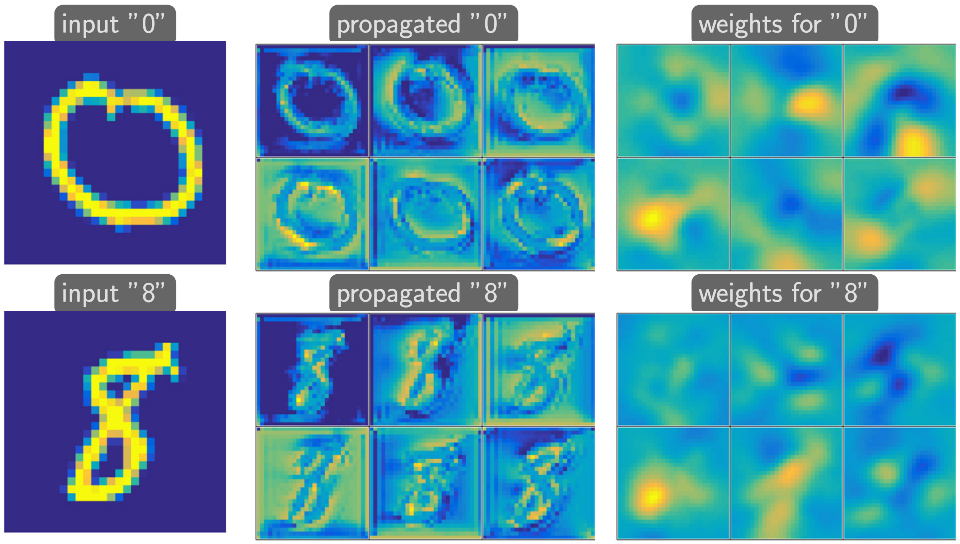

Figure 7. Classification results for the MNIST data set (see section 6.4) using an antisymmetric ResNet with $N = 16$ layers and 6 neurons. We show two randomly chosen input images, their propagated versions at the output layer, and the classification weights for the two digits shown. The smoothness enforced by the second-order regularization operator is evident in the weights.

of network parameters through regularization. We also emphasize that the well-posedness of the learning problem in ResNets relies on sufficiently small time steps. Extending our analysis to more general versions of ResNet [27] is straightforward and will be investigated in future work.

Motivated by our theoretical considerations, we propose three new forward propagation methods in section 4 that can be used to obtain well-posed learning problems for arbitrarily deep networks. The first one is a simple modification of ResNet that applies a linear transformation to the transformation matrices to annihilate the real parts of their eigenvalues. The other two methods are inspired by Hamiltonian systems and provide alternative ways to preserve in information in the forward propagation. The Hamiltonian networks require special integration techniques; see [2] for details on their treatment. The second-order forward propagation can be discretized using the leapfrog method, and our implementation of the first-order propagation uses the Verlet method. While the stability of the leapfrog method requires matrices with non-positive real eigenvalues, the Verlet method does not require any restrictions and yields the best performance in our numerical experiments.

Our approach to stabilizing the learning problem for very deep architectures differs significantly from existing approaches, most importantly *batch normalization* [31]. Batch normalization scales values at hidden layers to prevent vanishing or exploding gradients and has been shown to improve the efficiency of training deep networks. In contrast to that our approach does not modify the values of the features, but alleviates the need of normalization by constructing stable forward propagation methods.

In order to improve the generalization quality, improve stability, and simplify training of deep networks we also propose new regularization approaches that depend on our continuous formulation of the problem. We use derivative-based regularizers to favor smooth time dynamics and, for image classification problems, spatially smooth classification weights.

To further regularize and simplify the initialization of the learning algorithm, we employ a multi-level learning strategy that gradually increases the depth of the network. In our experiments, this approach has been a simple yet effective way to obtain good initializations for the learning problems. Our regularization methods are commonly used in imaging science, e.g. image registration [39], however, to the best of our knowledge not commonly employed in deep learning. Our numerical examples show that approximately solving the regularized learning problem yields works that generalize well even when the number of network parameters exceeds the number of training features.

We illustrate our methods using three academic test problems with available ground-truth and the MNIST problem [37], which is a commonly used benchmark problem for image classification. Our experiments show that the proposed new architectures yield results that are competitive with the established ResNet architecture. This is particularly noteworthy for the proposed anti-symmetric ResNet, where we restrict the dimensionality of the search space to ensure stability.

By establishing a link between deep learning and dynamic inverse problems, we are positive that this work will stimulate further research by both communities. An important item of future work is investigating the impact of the proposed architectures on the performance of learning algorithms. Currently, stochastic gradient descent [43] is commonly used to train deep neural networks due to its computational efficiency and empirical evidence supporting its generalizations [10]. A specific question is if the improvements in the forward propagation and regularization proposed in this paper will lead to better generalization properties of subsampled second-order methods such as [13, 44]. Another thrust of future work is the development of automatic parameter selection strategies for deep learning based on the approaches presented, e.g. in [18, 19, 24, 26, 48]. A particular challenge in this application is the nontrivial relationship between the regularization parameters chosen for the classification and forward propagation parameters.

Acknowledgments

This work is supported in part by National Science Foundation award DMS 1522599 and by the NVIDIA Corporation through their donation of a Titan X GPU.

ORCID iDs

Lars Ruthotto https://orcid.org/0000-0003-0803-3299

References

[1] Abu-Mostafa Y S, Magdon-Ismail M and Lin H-T 2012 *Learning from Data* vol 4 (New York: AMLBook)
[2] Ascher U 2010 *Numerical methods for Evolutionary Differential Equations* (Philadelphia: SIAM)
[3] Ascher U, Mattheij R and Russell R 1995 *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations* (Philadelphia: SIAM)
[4] Ascher U and Petzold L 1998 *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations* (Philadelphia: SIAM)
[5] Bengio Y 2009 Learning deep architectures for AI *Found. Trends® Mach. Learn.* **2** 1–127
[6] Bengio Y, Simard P and Frasconi P 1994 Learning long-term dependencies with gradient descent is difficult *IEEE Trans. Neural Netw.* **5** 157–66

[7] Bliss G A 1919 The use of adjoint systems in the problem of differential corrections for trajectories *JUS Artillery* **51** 296–311
[8] Bornemann F A and Deuflhard P 1996 The cascadic multigrid method for elliptic problems *Numer. Math.* **75** 135–52
[9] Borz A and Schulz V 2012 *Computational Optimization of Systems Governed by Partial Differential Equations* vol 8 (Philadelphia: SIAM)
[10] Bottou L 2012 Stochastic gradient descent tricks *Neural Networks: Tricks of the Trade* (Berlin: Springer)
[11] Bottou L, Curtis F E and Nocedal J 2016 Optimization methods for large-scale machine learning (arXiv:1606.04838)
[12] Burger M, Gilboa G, Osher S and Xu J 2006 Nonlinear inverse scale space methods *Commun. Math. Sci.* **4** 179–212
[13] Byrd R H, Chin G M, Neveitt W and Nocedal J 2011 On the use of Stochastic Hessian information in optimization methods for machine learning *SIAM J. Opt.* **21** 977–95
[14] Byrd R H, Chin G M, Nocedal J and Wu Y 2012 Sample size selection in optimization methods for machine learning *Math. Program.* **134** 127–55
[15] Calvetti D and Somersalo E 2005 Prior conditioners for linear systems *Inverse Problems* **4** 1397
[16] Cao Y, Li S, Petzold L and Serban R 2003 Adjoint sensitity analysis for differential-aglebraic equations: the adjoint DAE system and its numerical solution *SIAM J. Sci. Comput.* **24** 1076–89
[17] Chan T F and Shen J 2010 *Image Processing and Analysis* (Philadelphia: SIAM)
[18] de Sturler E and Kilmer M E 2011 A regularized Gauss–Newton trust region approach to imaging in diffuse optical tomography *SIAM J. Sci. Comput.* **33** 3057–86
[19] Engl H, Hanke M and Neubauer A 1996 *Regularization of Inverse Problems* (Dordrecht: Kluwer)
[20] Evans L C 1998 *Partial Differential Equations* (San Francisco: American Mathematical Society)
[21] Friedman J, Hastie T and Tibshirani R 2001 *The Elements of Statistical Learning* (*Springer Series in Statistics* vol 1) (Berlin: Springer)
[22] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (Cambridge, MA: MIT Press)
[23] Goodfellow I, Lee H, Le Q V, Saxe A and Ng A Y 2009 Measuring invariances in deep networks *Advances in Neural Information Processing Systems* pp 646–54
[24] Haber E and Oldenburg D 2000 A GCV based methods for nonlinear inverse problems *Comput. Geosci.* **4** 41–63
[25] Haber E, Ruthotto L and Holtham E 2017 Learning across scales-a multiscale method for convolution neural networks *AAAI Conf. Proc. (2018)* (arXiv:1703.02009)
[26] Hansen P C 1997 *Rank-Deficient and Discrete Ill-Posed Problems* (Philadelphia: SIAM)
[27] He K, Zhang X, Ren S and Sun J 2016 Deep residual learning for image recognition *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition* pp 770–8
[28] He K, Zhang X, Ren S and Sun J 2016 Identity mappings in deep residual networks *European Conf. on Computer Vision* (Springer) pp 630–45
[29] Hestenes M and Stieffel E 1952 Mehtods of conjugate gradients for solving linear systems *J. Res. Nat. Bur. Stand.* **49** 409–36
[30] Hinton G *et al* 2012 Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups *IEEE Signal Process. Mag.* **29** 82–97
[31] Ioffe S and Szegedy C 2015 Batch normalization: accelerating deep network training by reducing internal covariate shift *Int. Conf. Machine Learning* pp 448–56
[32] Kalman R E *et al* 1960 Contributions to the theory of optimal control *Bol. Soc. Mat. Mex.* **5** 102–19
[33] Keskar N S, Mudigere D, Nocedal J, Smelyanskiy M and Tang P T P 2016 On large-batch training for deep learning: generalization gap and sharp minima (arXiv:1609.04836)
[34] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks *Advances in Neural Information Processing Systems* pp 1097–105
[35] LeCun Y, Bengio Y and Hinton G 2015 Deep learning *Nature* **521** 436–44
[36] LeCun Y, Boser B E and Denker J S 1990 Handwritten digit recognition with a back-propagation network *Adv. Neural Netw.* 396–404
[37] Lecun Y and Cortes C The MNIST database of handwritten digits (http://yann.lecun.com/exdb/mnist/)
[38] Martens J and Sutskever I 2012 Training deep and recurrent networks with Hessian-free optimization *Neural Networks: Tricks of the Trade* (Berlin: Springer) pp 479–535

[39] Modersitzki J 2009 *FAIR: Flexible Algorithms for Image Registration* (Philadelphia: SIAM)
[40] Møller M F 1993 A scaled conjugate gradient algorithm for fast supervised learning *Neural Netw.* **6** 525–33
[41] Ng A Y 2004 Feature selection, l 1 veraus l 2 regularization, and rotational invariance *Proc. of the 21st Int. Conf. on Machine Learning* (ACM) p 78
[42] Nocedal J and Wright S 1999 *Numerical Optimization* (New York: Springer)
[43] Robbins H and Monro S 1951 A Stochastic approximation method *Ann. Math. Stat.*
[44] Roosta-Khorasani F and Mahoney M W 2016 Sub-sampled Newton methods II: local convergence rates
[45] Rothauge K, Haber E and Ascher U 2015 Numerical computation of the gradient and the action of the hessian for time-dependent pde-constrained optimization problems (arXiv:1509.03801)
[46] Rumelhart D, Hinton G and Williams J R 1986 Learning representations by back-propagating errors *Nature* **323** 533–8
[47] Saad Y 2003 *Iterative Methods for Sparse Linear Systems* 2nd edn (Philadelphia: SIAM)
[48] Vogel C R 2001 *Computational Methods for Inverse Problems* (Philadelphia: SIAM)