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, (haber@math.ubc.ca)
[2]Emory University, Department of Mathematics and Computer Science, Atlanta, GA, USA (lruthotto@emory.edu)
[3]Xtract Technologies Inc., Vancouver, Canada, (info@xtract.tech)

May 10, 2017

Abstract

Deep neural networks have become valuable tools for supervised machine learning, e.g., in the classification of text or images. While offering superior results over traditional techniques to find and express complicated patterns in data, deep architectures are known to be challenging to design and train such that they generalize well to new data. An important issue that must be overcome is 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 interpreting 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.

1 Introduction

In this work, we propose new architectures for deep neural networks (DNN) and show their effectiveness for solving supervised machine learning (ML) problems (for a general overview of DNN and ML see, e.g., [33, 17, 1] and references 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. We 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., machine learning 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., [4, 28, 25, 30] and references therein). Their depth (i.e., their number of layers) allows 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 $(\mathbf{y}_1, \mathbf{c}_1), \dots (\mathbf{y}_s, \mathbf{c}_s)$, 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 dimensionality, and thus the computational complexity, of the parameter estimation problem. Additionally, more labeled data is required to reliably calibrate very deep networks. Therefore, despite the fact that neural networks have been used since the early 70's, deep learning has only recently revolutionized the industry fueled by advances in computational hardware and the availability of massive data sets.

Well-known issues in deep learning are the size and non-convexity of the associated optimization problem. Traditionally, stochastic gradient descent methods have been used [35]. It has been observed that the results can be highly dependent on the choice of optimization algorithm and sample size [10, 7]. Furthermore, it has been noted that some optimization algorithms yield DNNs that generalize poorly to new unlabeled data.

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 [7], 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 [3]. These results are unsettling since instability of the forward problem usually implies ill-posedness of the inverse problem (i.e., the learning problem).

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

1. Given a network architecture and parameters obtained by some optimization process, is the forward propagation problem well posed?

2. Is the learning problem well posed? In other words, given sufficient training data, 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. That is, 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 ODE. Therefore, stability of the network corresponds to the stability of its underlying ODE. Based on this observation, we develop stability criteria for the commonly used residual network (ResNet) architecture [23] and develop *new network architectures* that are unconditionally 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 forward propagation with ResNet 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 architectures with guaranteed stability. In Section 5 we propose regularization functions favoring smoothness of parameters and multilevel ideas to initialize deep networks. In Section 6 we demonstrate the effectiveness of our new architectures on several model problems and explore their use for more realistic learning 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., [33, 17, 1, 18]). First, we outline *forward propagation* techniques that nonlinearly transform the input features. Second, we describe the *classification* process that predicts the class label probability using the features at the output layer (i.e., the output of the forward propagation). Finally, we formulate the learning problem that aims at estimating parameters of the forward and classification problems 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 the ResNet [23] model that has been very successful in classifying images using deep network architectures (see [18] 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 (2.1) is parametrized by the nonlinear pointwise 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 [23] 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 assumed to be smooth and monotonically non-decreasing. As two commonly used examples, we consider the hyperbolic tangent and the rectified linear unit 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 $\mathbf{e}_k \in \mathbb{R}^s$ denotes the k-dimensional vector of all ones. For Bernoulli variables ($m = 1$) 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 operations are applied element-wise. For multinomial distributions ($m > 1$) 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. This understanding provides new opportunities for applying theoretical and computational techniques from parameter estimation to deep learning problems. We present the learning processing as an optimization problem, i.e.,

$$\min_{\mathbf{W},\mu,\mathbf{K}_{0,\ldots,N-1},b_{0,\ldots,N-1}} \quad \frac{1}{s}S(\mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top), \mathbf{C}) + \alpha R(\mathbf{W}, \mu, \mathbf{K}_{0,\ldots,N-1}, b_{0,\ldots,N-1}) \tag{2.4a}$$

$$\text{subject to} \quad \mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma(\mathbf{Y}_j\mathbf{K}_j + b_j), \quad j = 0, 1, \ldots, N-1, \tag{2.4b}$$

where the loss function S is convex in its first argument and measures the quality of the predicted class label probabilities, the regularizer R penalizes undesirable (e.g., highly oscillatory) parameters, and the parameter $\alpha > 0$ trades off minimizing the data fit and regularity of the parameters. A simple example of 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, also referred to as weight decay, has been used [19], although, other possibilities enforcing sparsity or other structure have been proposed [34]. 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 [8].

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, $(\mathbf{W}, \mu)$, using the current values 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.,

$$\frac{1}{s}S(\mathbf{h}(\mathbf{Y}_N\mathbf{W} + \mathbf{e}_s\mu^\top), \mathbf{C}) = \frac{1}{s}\sum_{i=1}^{s} S(\mathbf{h}(\mathbf{y}_i^\top\mathbf{W} + \mu^\top), \mathbf{c}_i^\top) \approx \frac{1}{T}\sum_{i\in\mathcal{T}} S(\mathbf{h}(\mathbf{y}_i^\top\mathbf{W} + \mu^\top), \mathbf{c}_i^\top), \tag{2.5}$$

where $\mathcal{T} \subset \{1, 2, \ldots, s\}$ is a randomly chosen set updated during 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, as well as the computational resources available. In each iteration of the block coordinate descent scheme, our algorithm approximately solves the resulting classification problem using a Newton preconditioned conjugate gradient (PCG) method. Subsequently, the weights of the forward propagation are updated using a Gauss-Newton-PCG method. Note that gradient and approximate 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. However, this requires storing (or re-computing) the values at the output layers; see, e.g., Section 4.4 for derivative computation. Therefore, as also suggested in [9], 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 is 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. e.g., 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 do poorly on a perturbation of the data. In other words, networks whose forward propagation is not well-posed 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., [3]. These phenomena refer to the gradient of the objective function in (2.4a) and pose a severe challenge for very deep architectures. Note that the gradient represents the sensitivity of the output with respect to a perturbation of 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 result in DNNs that generalize well.

To understand the phenomenon, consider the forward propagation in ResNets (2.1). As pointed out in [21], the forward propagation is an explicit Euler discretization of the nonlinear ODE

$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}^\top(t)\mathbf{y}(t) + b(t)), \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0, \tag{3.6}$$

over a time interval $t = [0, T]$. The final time $T > 0$ and the rate in which $\mathbf{K}(t)$ changes in time controls the depth of the network. The ODE is stable if

$$\max\left(Re(\lambda(\mathbf{J}(t)))\right) \leq 0, \quad \forall t \in [0, T], \tag{3.7}$$

where $Re(\cdot)$ denotes the real part, $\lambda(\cdot)$ are the eigenvalues of a square matrix, and $\mathbf{J}$ is the Jacobian of the right hand side in (3.6), i.e.,

$$\mathbf{J}(t) = \left(\nabla_{\mathbf{y}}\left(\sigma(\mathbf{K}(t)^\top\mathbf{y} + b(t))\right)\right)^\top = \operatorname{diag}\left(\sigma'(\mathbf{K}(t)^\top\mathbf{y} + b(t))\right)\mathbf{K}(t)^\top. \tag{3.8}$$

Since the activation function σ is typically monotonically non-decreasing, i.e., $\sigma'(\cdot) \geq 0$, (3.7) is satisfied if

$$\max\left(Re(\lambda(\mathbf{K}(t)))\right) \leq 0, \quad \forall t \in [0, T]. \tag{3.9}$$

To ensure the stability of the overall 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 *The ResNet (2.1) is stable if*

$$|1 + h\max(\lambda(\mathbf{K}_j))| \leq 1, \quad \forall j = 0, 1, \ldots, N-1. \tag{3.10}$$

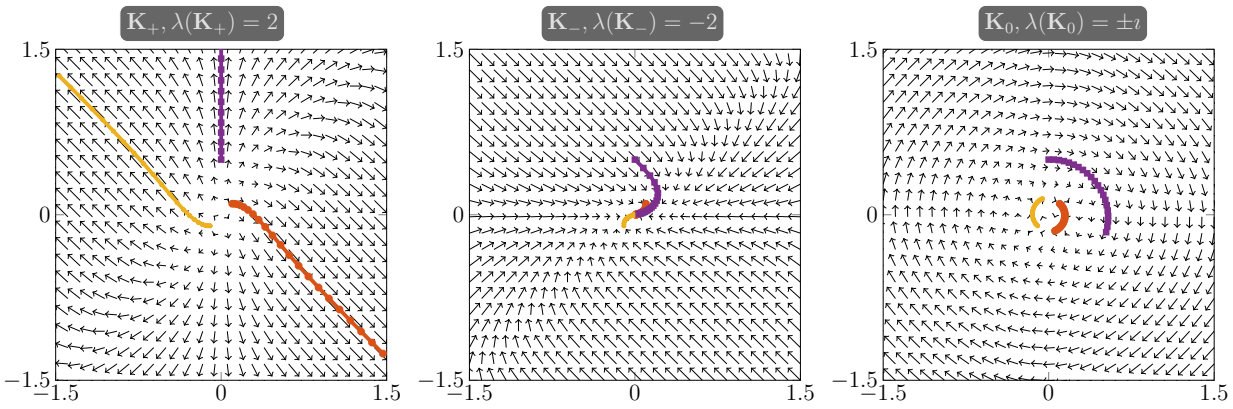

Figure 1: Phase plane diagrams for ResNets with $N = 10$ identical layers parameterized by the weight matrices in (3.11) starting with three different features. Colored lines indicate the values at hidden layers and arrows depict the force field. Left: Due to its positive eigenvalues, the features diverge 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 anti-symmetric matrix $\mathbf{K}_0$ leads to rotations, preserves distances between the features, and yields well-posed forward propagation and learning.

Proof 1 *See, e.g., [2] for the proof of stability criteria for the forward Euler method.*

The above discussion suggests that the stability of the continuous forward propagation (3.9) and its discrete analog (3.10) 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 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 of ResNet using a simple example.

Example 1 (Stability of ResNet) *For $s = 3$ and $n = 2$ we consider the forward propagation through a ResNet given by (2.1). We consider three networks consisting of $N = 10$ identical layers, i.e., for each layer we use the activation $\sigma_{\text{ht}} = \tanh$, $h = 0.1$, $b = 0$, 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*

$$\mathbf{K}_+ = \begin{pmatrix} 2 & -2 \\ 0 & 2 \end{pmatrix}, \quad \mathbf{K}_- = \begin{pmatrix} -2 & 0 \\ 2 & -2 \end{pmatrix}, \quad \text{and} \quad \mathbf{K}_0 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \tag{3.11}$$

where $\lambda(\mathbf{K}_+) = 2$, $\lambda(\mathbf{K}_-) = -2$ and $\lambda(\mathbf{K}_0) = \pm\imath$. 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 yields 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 origin. While the forward propagation satisfies* (3.9) *and is thus stable, the learning problem, which requires inversion of this process, is ill-posed. Finally, the anti-symmetric 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 (3.9) is a necessary condition for successful learning, the second case suggests that it is not sufficient. In general, when $Re(\lambda(\mathbf{K}(t))) < 0$ for most times and the network is deep (e.g., long time integration) differences between 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 (2.4a) will vanish since small changes in the transformation weights 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 (2.4b) with $Re\left(\max(\lambda(\mathbf{K}))\right) > 0$ consists of neurons that amplify the signal with no upper bound, which is not biological, and a ResNet with $Re\left(\max(\lambda(\mathbf{K}))\right) \ll 0$ 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 information in the input signal.

In summary, our discussion of ResNets illustrates that the stability of the forward propagation and well-posedness of the learning problem can be obtained for deep networks when

$$Re\left(\lambda(\mathbf{K}(t))\right) \approx 0, \quad \forall t \in [0, T]. \tag{3.12}$$

In this case, the forward propagation causes only moderate amplification or signal 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 deep neural networks and lead to well-posed inverse problems. Our approaches, presented in Section 4.1 and 4.2, use different means to enforce Jacobians whose eigenvalues have very small real parts (see (3.12)). The methods in Section 4.2 are inspired by Hamiltonian systems and we present leapfrog and Verlet integration techniques for forward propagation in Section 4.3. Finally, we compute the derivative of the Verlet method using back propagation [38] and discuss the relation between back propagation and the older and more general adjoint method [27] in Section 4.4.

4.1 Anti-Symmetric Weight Matrices

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

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + h\sigma\left(\frac{1}{2}\mathbf{Y}_j\left(\mathbf{K}_j - \mathbf{K}_j^\top\right) + b_j\right), \quad j = 0, 1, \ldots, N-1, \tag{4.13}$$

which is a forward Euler discretization of the ODE

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

Since $\mathbf{K}(t) - \mathbf{K}(t)^\top$ is anti-symmetric its eigenvalues are imaginary, which also holds for the Jacobian of the right hand side as shown in (3.8). Thus, the ResNet parameterized by the anti-symmetric weight matrix is stable and preserves information given an appropriate integration technique and sufficiently small time steps.

It is possible to add some diffusion to the process, e.g., by adding a negative definite matrix. While this might improve the robustness against noise in the feature vectors, we do not explore it in this work for brevity.

4.2 Hamiltonian Inspired Neural Networks

Restricting the parameter space to anti-symmetric matrices 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 of conserving vs. increasing or dissipating the energy of the system; see [2, Ch.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 (3.6) we propose to use the following second order ODE

$$\ddot{\mathbf{y}}(t) = \sigma(\mathbf{K}^\top(t)\mathbf{y}(t) + b(t)), \quad \mathbf{y}(0) = \mathbf{y}_0. \tag{4.14}$$

The dynamical system in (4.14) is stable for all weight matrices with non-positive real eigenvalues, i.e., that satisfy (3.9). This constraint can either be added to (2.4) or like the previous section, be enforced by design. The latter approach can be obtained, e.g., by using

$$\mathbf{K}(\mathbf{C}) = -\mathbf{C}^\top \mathbf{C}, \quad \text{for some} \quad \mathbf{C} \in \mathbb{R}^{n\times n}. \tag{4.15}$$

Note that in contrast to the anti-symmetric model in (4.13), the negative definite model is more challenging to ensure. Guaranteeing (3.9) either requires adding a *nonlinear* transformation as in (4.15), or involves expensive eigenvalue computations, if added as constraint to (2.4).

The forward propagations (2.1), (4.13), or (4.14) require additional constraints on $\mathbf{K}$ and are limited to square weight matrices. We therefore propose a network that is *intrinsically* stable, 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

$$\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}(t)\mathbf{z}(t) + b(t)) \quad \text{and} \quad \dot{\mathbf{z}}(t) = -\sigma(\mathbf{K}(t)^\top \mathbf{y}(t) + b(t)). \tag{4.16}$$

Note that (4.16) is a special case of ResNet with an augmented variable $\mathbf{z} \in \mathbb{R}^m$ and the associated ODE

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

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 anti-symmetric. To ensure stability of the discrete version the time step size needs to be sufficiently small; see Lemma 3.10.

4.3 Symplectic Forward Propagation

In this section, we use symplectic integration techniques to solve the discrete versions of the Hamiltonian-inspired networks (4.14) and (4.16). 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 (2.4b). For the second-order ODE (4.14) we propose the conservative leapfrog discretization

$$\mathbf{y}_{j+1} = \begin{cases} 2\mathbf{y}_j + h^2\sigma(\mathbf{K}_j\mathbf{y}_j + b_j), & j = 0\\ 2\mathbf{y}_j - \mathbf{y}_{j-1} + h^2\sigma(\mathbf{K}_j\mathbf{y}_j + b_j), & j = 1, 2, \ldots, N-1 \end{cases}. \tag{4.17}$$

For the augmented network (4.16) we use a Verlet integration where for $j = 0, 1, \ldots, N-1$ we have

$$\mathbf{z}_{j+\frac{1}{2}} = \mathbf{z}_{j-\frac{1}{2}} - h\sigma(\mathbf{K}_j^\top \mathbf{y}_j + b_j) \tag{4.18a}$$

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h\sigma(\mathbf{K}_j\mathbf{z}_{j+\frac{1}{2}} + b_j) \tag{4.18b}$$

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.

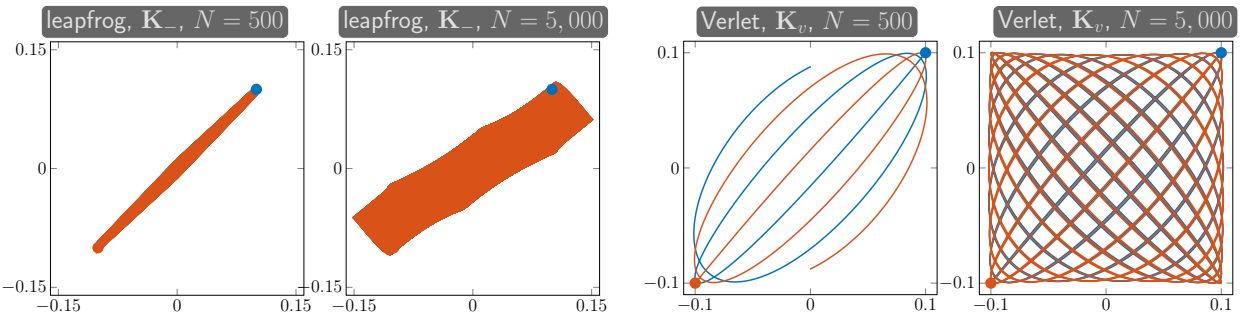

Figure 2: Phase space diagrams for the forward propagation using the leapfrog and Verlet methods starting from $\mathbf{y}_1 = [0.1, 0.1]^\top$ (blue) and $\mathbf{y}_2 = [-0.1, -0.1]^\top$ (red). For each network we show the short term ($N = 500$) and long term ($N = 5,000$) behavior for identical layers. In both cases non-trivial behavior can be observed even for constant weight matrices.

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 (3.11) (recall that $\lambda(\mathbf{K}) = -2$) and a step size of $h = 1$. To illustrate the Verlet integration (which can handle non-square matrices) 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 $N = 5,000$ layers. We depict the phase plane diagrams for both networks and both depths in Figure 2. Even though we use identical layers with constant matrices 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 [24]. Given the ResNet derivatives, the anti-symmetric model can be differentiated using the chain rule and time stepping. The leapfrog method can be treated in a similar way to the usual ResNet architecture and therefore the details are omitted.

Our presentation follows the standard approach used in machine learning known as back propagation [24]; however, we note that back propagation is a special case of the adjoint method [5] used in time dependent optimal control; see, e.g.,[6]. The adjoint method is more general than back propagation as it can also be used to 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 from 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 of (4.18) 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 computed along the same line. Clearly, the values of $\mathbf{z}_j$ and $\mathbf{y}_j$ at the hidden layers $j \leq k$ are independent of $\mathbf{K}_k$. Applying the chain rule to (4.18) we see that

$$\begin{aligned}\frac{\partial \mathbf{z}_{k+\frac{1}{2}}}{\partial \mathbf{K}_k} &= h\mathsf{diag}\left(\sigma'(\mathbf{K}_k\mathbf{y}_k + b_k)\right)(\mathbf{y}_k \otimes \mathbf{I}) =: \mathbf{C}_1 \\ \frac{\partial \mathbf{y}_{k+1}}{\partial \mathbf{K}_k} &= -h\mathsf{diag}\left(\sigma'(-\mathbf{K}_k^\top \mathbf{z}_{k+\frac{1}{2}} + b_k)\right)(\mathbf{I} \otimes \mathbf{Z}_{k+\frac{1}{2}} + \mathbf{K}_k^\top \otimes \mathbf{I}) =: \mathbf{C}_2,\end{aligned} \tag{4.19}$$

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

$$\begin{aligned}\frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k} &= \frac{\partial \mathbf{z}_{j-\frac{1}{2}}}{\partial \mathbf{K}_k} + h\mathsf{diag}\left(\sigma'(\mathbf{K}_j\mathbf{y}_j + b_j)\right)\mathbf{K}_j \frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k} \\ \frac{\partial \mathbf{y}_{j+1}}{\partial \mathbf{K}_k} &= \frac{\partial \mathbf{y}_j}{\partial \mathbf{K}_k} - h\mathsf{diag}\left(\sigma'(-\mathbf{K}_j^\top \mathbf{z}_{j+\frac{1}{2}} + b_j)\right)\mathbf{K}_j^\top \frac{\partial \mathbf{z}_{j+\frac{1}{2}}}{\partial \mathbf{K}_k}\end{aligned} \tag{4.20}$$

Combining equations (4.19) and (4.20) the derivatives can be written compactly as a block linear system

$$\begin{pmatrix} \mathbf{I} & & & & & & \\ & \mathbf{I} & & & & & \\ -\mathbf{I} & \mathbf{B}_{k+1} & \mathbf{I} & & & & \\ & -\mathbf{I} & \mathbf{A}_{k+1} & \mathbf{I} & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -\mathbf{I} & \mathbf{A}_N & \mathbf{I} & \\ & & & & -\mathbf{I} & \mathbf{B}_N & \mathbf{I} \end{pmatrix} \begin{pmatrix} \frac{\partial \mathbf{z}_{k+\frac{1}{2}}}{\partial \mathbf{K}_k} \\ \frac{\partial \mathbf{y}_{k+1}}{\partial \mathbf{K}_k} \\ \vdots \\ \\ \frac{\partial \mathbf{z}_{N+\frac{1}{2}}}{\partial \mathbf{K}_k} \\ \frac{\partial \mathbf{y}_{N+1}}{\partial \mathbf{K}_k} \end{pmatrix} = \begin{pmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \\ 0 \\ \vdots \\ \\ 0 \end{pmatrix}, \tag{4.21}$$

where

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

The system (4.21) 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, including notes on implementation, see, e.g., [37].

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 sensitivities. 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 (4.21) 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 [38] is essentially a particular implementation of the adjoint method discussed much earlier in [27].

5 Regularization

In this section we present derivative-based regularization functions and a multilevel learning approach to ensure smoothness of parameters in deep learning. By biasing the learning process towards smooth time dynamics we aim at improving 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, 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. While this regularization reduces the magnitude of the weights, it is not sensitive 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.

The above interpretation of forward propagation as a time dependent process implies 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

$$R(\mathbf{K}) = \frac{1}{h}\sum \|\mathbf{K}_j - \mathbf{K}_{j-1}\|_F^2 \quad \text{and} \quad R(b) = \frac{1}{2h}\sum (b_j - b_{j-1})^2. \tag{5.22}$$

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., [18, Ch.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 size of each column in $\mathbf{W}$ is $n_1 \cdot n_2 \cdot n_3$ and the operation $\mathbf{Y}_N(j,:)^\top\mathbf{W}(:,j)$ 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., [12].

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 machine learning also known as *pooling* aims at achieving this goal. 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 [22]. 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 $\mathbf{w}_k \in \mathbb{R}^n$ in CNN as discretizations of image functions $y_j : \Omega \to \mathbb{R}$ and $w_k : \Omega \to \mathbb{R}$ where $\Omega \subset \mathbb{R}^2$ is the image domain. Assuming y_j are 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

$$\mathbf{h}(\mathbf{y}_j^\top \mathbf{w}_k + \mu_k) \approx \mathbf{h}\left(\int_\Omega y(x)w(x)\,dx + \mu_k\right). \tag{5.23}$$

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

$$R(\mathbf{w}_k) = \frac{1}{2}\|\mathbf{L}\mathbf{w}_k\|^2, \tag{5.24}$$

where $\mathbf{L}$ is a discretized differential operator. This regularization also embeds the optimal w_k into a suitable function spaces. For example, using an image gradient ensures that w_k is in the Sobolev space $H^1(\Omega, \mathbb{R})$. 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 Multilevel Learning

As another means of regularizing the problem, we exploit a multilevel 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 ideas in image processing where multilevel strategies are commonly used to decrease the risk of being trapped in local minima; see, e.g., [32]. More details about multilevel methods in learning can be found in [21].

The basic idea is to begin by solving the learning problem using a network with only a few layers and then prolongate the estimated weights of the forward propagation to initialize the optimization problem for the next network that features twice as many layers. We repeat this process until a user-specified maximum number of layers is reached.

Besides realizing some obvious computational savings on the smaller 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, deeper networks are in our experience difficult to initialize. Additionally, the Gauss-Newton-PCG method, which we use to estimate the parameters of the forward propagation, typically converges within a few iterations when the initial parameters are close to a local minimum.

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 [29, 31], which is a common benchmark problem in image classification.

6.1 Concentric Ellipses

As a first test, we consider a small-scale problem in two dimensions. The test data consists of 600 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 500 training examples and 100 examples used for validation.

We train the original ResNet, the anti-symmetric ResNet, and the Hamiltonian networks with leapfrog and Verlet propagation using the block coordinate descent method and a multilevel strategy with $4, 8, 16, 32, 64, 128, 256$ and 1024 layers. Each block coordinate descent iteration consists of a classification step ($\leq$2 iterations of Newton-PCG with ≤ 2 PCG iterations) and a Gauss-Newton-PCG step to update the propagation weights ($\leq$10 PCG iterations preconditioned by regularization operator). In all examples we use the logistic regression hypothesis function (2.2) and tanh activation function. The final time is $T = 20$ and the width of the network is $n = 2$.

We show the performance of the multilevel scheme in the left subplot of Figure 3. For this simple data set, all forward propagation methods eventually achieve an optimal validation accuracy of 100%. It is notable though that the validation accuracy does not increase monotonically for the ResNets and leapfrog methods, while it does so for the Verlet forward propagation. 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 4,096 points each. The data is perturbed by adding 1% Gaussian white noise to the features. Every other point along the curve is removed from the data set and used for validation. See left subplot in 5 for a plot of the training data.

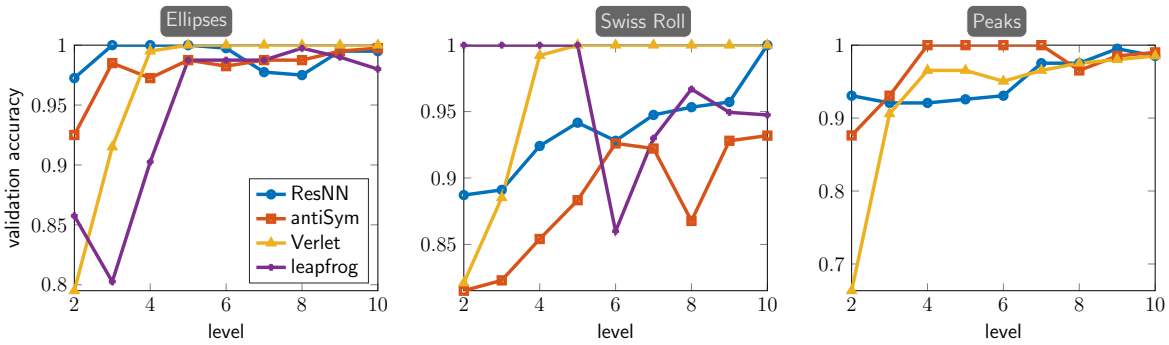

Figure 3: Multilevel convergence for the two-dimensional examples in Sections 6.1–6.3. For each level (level ℓ corresponds to DNN with 2^ℓ layers) we show the best validation accuracy (the optimal value is 1, which corresponds to a validation error of 0%).

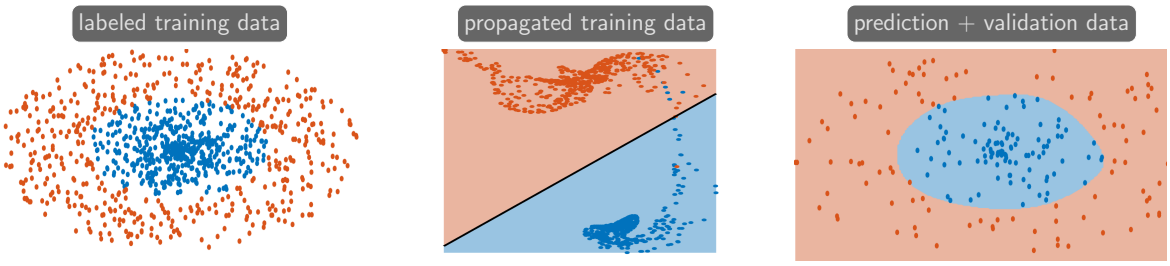

Figure 4: Classification results for the ellipse problem from Section 6.1 using a Hamiltonian Neural Network with Verlet propagation with $N = 1,024$ layers. Left: Labeled input features representing two concentric ellipses that are not linearly separable. Center: The output features are linearly separable and can be accurately classified. Right: We show the predictions of the network (colors in the background) superimposed by the validation data.

Given the data we use the same parameterization of the multilevel and block coordinate descent method as in the previous example. For all networks we use a width of $n = 2$, the tanh activation, logistic regression function (2.2), and choose a final time of $T = 10$. The propagation weights enforcing smoothness in time with $\alpha = 10^{-6}$. No regularization is used in the classification.

We plot the validation accuracy for each network and the different steps of the multilevel 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 $N = 1,024$, however, the convergence is considerably faster for the Hamiltonian network that reaches the optimal accuracy with $N = 32$ layers. Interestingly, the accuracy of the leapfrog network is optimal for networks up to 32 layers in this example and deteriorates for deeper networks. We visualize the results obtained using the Verlet method in Figure 5.

Figure 5: Classification results for the Swiss roll example described in Section 6.2 using a Hamiltonian network with $N = 1,024$ 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 the affine linear logistic regression classifier can separate them. Right: We show the interpolation obtained by our network (colored background) and the validation data.

6.3 Peaks

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

$$\begin{aligned} f(x) = \quad & 3(1-x_1)^2 \exp(-(x_1^2) - (x_2+1)^2) - 10(x_1/5 - x_1^3 - x_2^5) \\ & \exp(-x_1^2 - x_2^2) - 1/3 \exp(-(x_1+1)^2 - x_2^2), \end{aligned}$$

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 $s = 1,100$ sample points are divided into five classes each with 202 points. 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 880 examples. We use tanh for the activation, the softmax hypothesis function (2.3), and choose a final time of $T = 10$. We regularize the propagation weights enforcing smoothness in time with a regularization parameter of $\alpha = 10^{-5}$. 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.

We first train the standard ResNet and the anti-symmetric variant where we use a width of $n = 8$ by duplicating the features four times. The performance of both models at the final level is comparable; see also right subplot in Figure 3. The validation accuracy for $N = 1,024$ is around 98.5 % for the standard and 99.1% for the anti-symmetric version. It is notable though that the anti-symmetric ResNet outperforms on levels 4 and 7 (100% validation accuracy).

For the Hamiltonian network with Verlet forward propagation we use a narrower network containing only the original features (i.e., $n = 2$ / no duplication). As in the previous experiments the accuracy generally improves with increasing depth (with one exception between levels 5 and

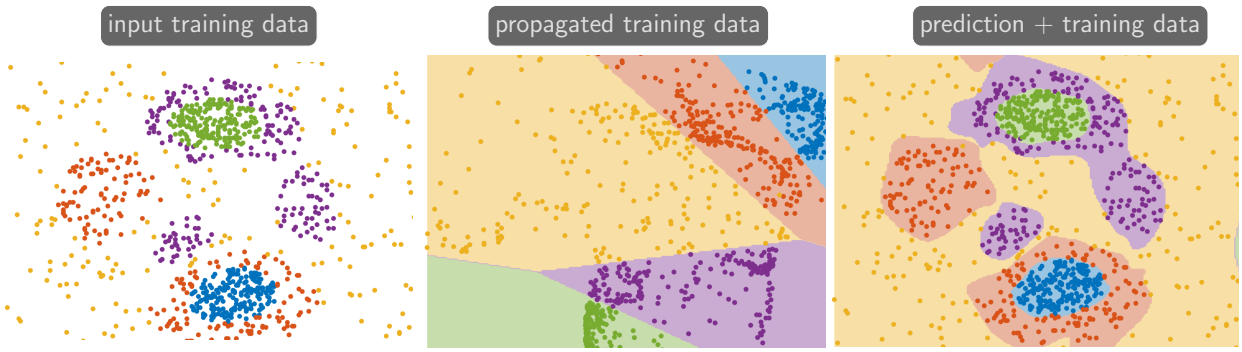

Figure 6: Classification results for peaks example described in 6.3 using a Hamiltonian network with $n = 1,024$ layers and the Verlet forward propagation. Left: We illustrate the training data by colored dots that represent the class. Center: We show the propagated features and the predictions of the softmax classifier. Right: We depict the predictions of our network (colored background) and the training data.

6). The optimal accuracy is obtained at the final level ($N = 1,024$) with a validation accuracy of 98.5%. 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 [29, 31] 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. The MNIST problem is considered a solved problem and it has been shown that, using *data augmentation* techniques, DNNs can achieve accuracies of around 0.2%; see, e.g., [13, 14]. Data augmentation increases the dimensionality of the feature space by adding transformed versions of the input data, e.g., by applying spatial transformations. We do not use data augmentation in the following, since our example aims at comparing the different forward propagation techniques for narrow but deep architectures. We compare using the standard and anti-symmetric ResNet and the first-order Hamiltonian network using Verlet integration.

We use a three-level multilevel 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 $n = 4,704$ 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 5,000 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

	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%

Table 1: Multilevel training result for MNIST data set (see Section 6.4). We show the training error (TE) and validation error (VE) for the standard and anti-symmetric 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 anti-symmetric ResNet giving slightly lower validation errors at each level.

most 5 iterations of Newton-PCG with up to 10 inner iterations. We use a discretized Laplacian as a regularization operator in (5.24) and its shifted symmetric product as a preconditioner to favor smooth modes; see [11]. The regularization parameter used in 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 multilevel 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 errors are reduced using a 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) gave already a validation accuracy of 98.41%, which was improved to 98.47% with subsequent training. We illustrate the results of the anti-symmetric 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

We propose new architectures for deep neural networks that improve stability of the forward propagation and through regularization improve the well-posedness of the learning task. We also propose a multilevel 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. This work also exposes the relation between deep learning and dynamic inverse problems laying a foundation for fruitful research directions in data science.

Our stability discussion in Section 3 provides new insights why ResNets [23], a commonly used forward propagation scheme in deep learning, can be unstable. Our result is based on a continuous interpretation of ResNets as a system of nonlinear ODEs and a standard stability argument. We provide an intuitive example that shows how the spectral properties of the transformation matrix determine whether gradients explode, vanish, or are stable. Based on our findings and numerical

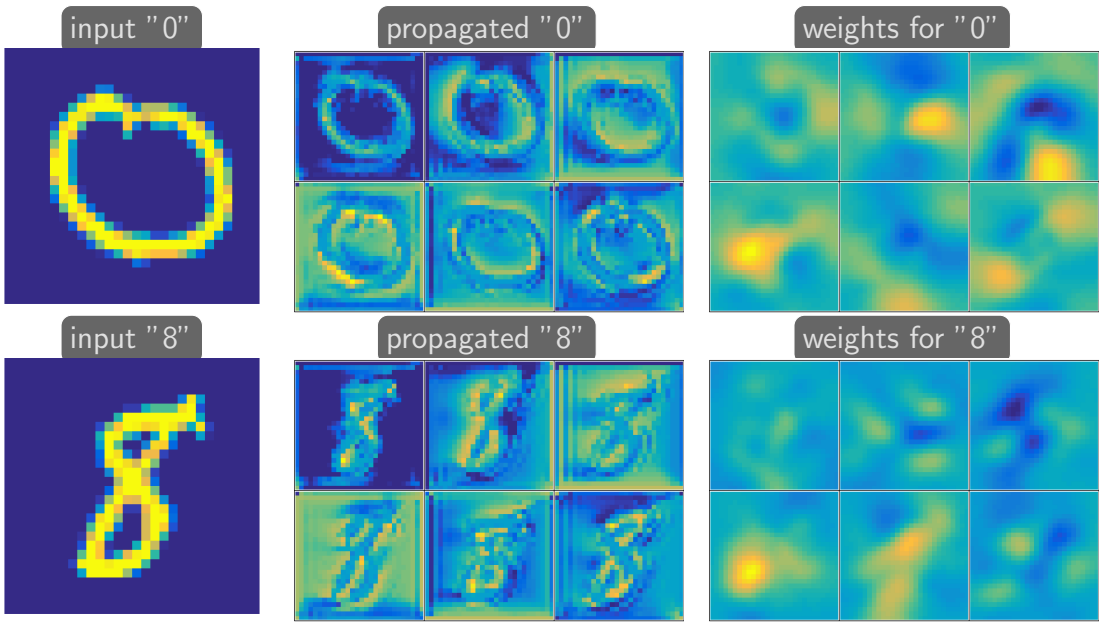

Figure 7: Classification results for the MNIST data set (see Section 6.4) using an anti-symmetric ResNet with $N = 16$ layers and 6 neurons. We show two randomly chosen input images, their propagated versions at the output layer, and the classification weights for the two digits shown. The smoothness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

experience, we argue that it is desirable to restrict ResNets to matrices whose real parts of the spectrum are close to zero. We also emphasize that the well-posedness of the learning problem in ResNets relies on sufficiently small time steps.

Based on our theoretical considerations, we propose three new forward propagation methods in Section 4 that lead to 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 is discretized using the leapfrog method and the first-order propagation uses the Verlet method. While 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 stabilize the learning problem for very deep architectures differs significantly from existing approaches, e.g., *batch normalization* [26]. The idea in batch normalization is to add hidden layers that rescale the propagated features to prevent vanishing or exploding gradients and to improve overall conditioning of the optimization problem; see [7, Sec. 6]. The scaling coefficients are estimated during the stochastic optimization using mini-batches. Thus, the output values of the network are affected by the selection of the batches, which introduces additional noise in the stochastic optimization scheme [7]. By design, our proposed methods ensure that the overall scaling of features is preserved, which is why batch normalization is not applied in our numerical

experiments.

To improve the ability to generalize and simplify training 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, smooth classification weights. As additional regularization, 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 [32], however, to the best of our knowledge not commonly employed in deep learning.

We illustrate our methods using three academic test problems with available ground-truth and the MNIST problem [31], 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 [35] is commonly used to train deep neural networks due to its computational efficiency and empirical evidence supporting its generalizations. 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 [9, 36]. Another thrust of future work is the development of automatic parameter selection strategies for deep learning based on the approaches presented, e.g., in [16, 22, 39, 20, 15]. A particular challenge in this application is the nontrivial relationship between the regularization parameters chosen for the classification and forward propagation parameters.

8 Acknowledgements

The second author's work is supported by National Science Foundation (NSF) award DMS 1522599.

References

[1] Yaser S Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012.

[2] U.M. Ascher. *Numerical methods for Evolutionary Differential Equations*. SIAM, Philadelphia, 2010.

[3] Y BENGIO, P SIMARD, and P FRASCONI. Learning Long-Term Dependencies with Gradient Descent Is Difficult. *Ieee Transactions on Neural Networks*, 5(2):157–166, 1994.

[4] Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[5] Go A Bliss. The use of adjoint systems in the problem of differential corrections for trajectories. *JUS Artillery*, 51:296–311, 1919.

[6] Alfio Borzì and Volker Schulz. *Computational optimization of systems governed by partial differential equations*, volume 8. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2012.

[7] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[8] Martin Burger, Guy Gilboa, Stanley Osher, and Jinjun Xu. Nonlinear inverse scale space methods. *Commun. Math. Sci.*, 4(1):179–212, 03 2006.

[9] Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the Use of Stochastic Hessian Information in Optimization Methods for Machine Learning. 21(3):977–995, 2011.

[10] Richard H Byrd, Gillian M Chin, Jorge Nocedal, and Yuchen Wu. Sample size selection in optimization methods for machine learning. *Mathematical programming*, 134(1):127–155, 2012.

[11] D Calvetti and E Somersalo. Priorconditioners for linear systems. *Inverse Problems*, 2005.

[12] Tony F Chan and Jianhong Shen. *Image Processing and Analysis*. Society for Industrial and Applied Mathematics, 2010.

[13] Dan Ciresan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-Performance Neural Networks for Visual Object Classification. *arXiv.org*, 2011.

[14] Dan Ciresan, Ueli Meier, and Juergen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *arXiv.org*, 2012.

[15] Eric de Sturler and Misha E Kilmer. A Regularized Gauss–Newton Trust Region Approach to Imaging in Diffuse Optical Tomography. *SIAM Journal on Scientific Computing*, 33(5):3057–3086, 2011.

[16] H.W. Engl, M. Hanke, and A. Neubauer. *Regularization of Inverse Problems*. Kluwer, 1996.

[17] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[19] Ian Goodfellow, Honglak Lee, Quoc V Le, Andrew Saxe, and Andrew Y Ng. Measuring invariances in deep networks. In *Advances in neural information processing systems*, pages 646–654, 2009.

[20] E. Haber and D. Oldenburg. A GCV based methods for nonlinear inverse problem. *Computational Geoscience*, 4, 2000. n1.

[21] Eldad Haber, Lars Ruthotto, and Elliot Holtham. Learning across scales-a multiscale method for convolution neural networks. *arXiv preprint arXiv:1703.02009*, 2017.

[22] P. C. Hansen. *Rank-Deficient and Discrete Ill-Posed Problems*. SIAM, Philadelphia, 1997.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[25] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[26] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv.org*, 2015.

[27] Rudolf Emil Kalman et al. Contributions to the theory of optimal control. *Bol. Soc. Mat. Mexicana*, 5(2):102–119, 1960.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[29] Y LeCun, B E Boser, and J S Denker. Handwritten digit recognition with a back-propagation network. *Advances in Neural Networks*, 1990.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[31] Yann Lecun and Corinna Cortes. The MNIST database of handwritten digits.

[32] J. Modersitzki. *FAIR: Flexible Algorithms for Image Registration*. SIAM, Philadelphia, 2009.

[33] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[34] Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[35] H Robbins and S Monro. A Stochastic Approximation Method. *The annals of mathematical statistics*, 1951.

[36] F. Roosta-Khorasani and M. W. Mahoney. Sub-Sampled Newton Methods II: Local Convergence Rates. *ArXiv e-prints*, January 2016.

[37] Kai Rothauge, Eldad Haber, and Uri Ascher. Numerical computation of the gradient and the action of the hessian for time-dependent pde-constrained optimization problems. *arXiv preprint arXiv:1509.03801*, 2015.

[38] D.E. Rumelhart, Geoffrey Hinton, and J. Williams, R. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[39] C. Vogel. *Computational methods for inverse problem.* SIAM, Philadelphia, 2001.