Proceedings of Machine Learning Research vol 144:1–12, 2021

A unified framework for Hamiltonian deep neural networks

Clara Lucía Galimberti CLARA.GALIMBERTI@EPFL.CH
Liang Xu LIANG.XU@EPFL.CH
Giancarlo Ferrari Trecate GIANCARLO.FERRARITRECATE@EPFL.CH
Institute of Mechanical Engineering, École Polytechnique Fédérale de Lausanne, Switzerland.

Abstract

Training deep neural networks (DNNs) can be difficult due to the occurrence of vanishing/exploding gradients during weight optimization. To avoid this problem, we propose a class of DNNs stemming from the time discretization of Hamiltonian systems. The time-invariant version of the corresponding Hamiltonian models enjoys marginal stability, a property that, as shown in previous works and for specific DNNs architectures, can mitigate convergence to zero or divergence of gradients. In the present paper, we formally study this feature by deriving and analysing the backward gradient dynamics in continuous time. The proposed Hamiltonian framework, besides encompassing existing networks inspired by marginally stable ODEs, allows one to derive new and more expressive architectures. The good performance of the novel DNNs is demonstrated on benchmark classification problems, including digit recognition using the MNIST dataset.

Keywords: Deep Neural Networks, Dynamical Systems, Hamiltonian Systems, Gradient Dynamics.

1. Introduction

Deep learning has achieved remarkable success in different fields like computer vision, speech recognition and natural language processing (He et al., 2016; Xiong et al., 2017). There is also a growing interest in using deep neural networks (DNNs) for approximating complex controllers (Lucia and Karg, 2018; Zoppoli et al., 2020). In spite of several progresses, the training of DNNs still presents some difficulties. Most optimization algorithms for DNNs, such as stochastic gradient descent, involve the computation of gradients that, as observed in Bengio et al. (1994), can explode or vanish, hence making the learning problem ill-posed.

Recently, it has been shown that these issues are mitigated for specific classes of DNNs which stem from the time discretization of ordinary differential equations (ODEs) (Haber and Ruthotto, 2017; Haber et al., 2018; Chang et al., 2019; Lu et al., 2018; E, 2017). These results leverage the stability properties of the underlying ODE for characterizing relevant behaviours of the corresponding DNNs (Haber and Ruthotto, 2017). Specifically, instability of the underlying ODE results in unstable forward propagation in the DNN model, while convergence to zero of system states can lead to vanishing gradients during training. This observation suggests using DNN architectures based on dynamical system models that produce bounded and non-vanishing state trajectories. An example is provided by ODEs based on skew-symmetric maps, which have been used in Haber and Ruthotto (2017); Chang et al. (2019) for defining anti-symmetric DNNs. Another example is given by dynamical systems in the form

$$\dot{\mathbf{y}} = -\nabla_{\mathbf{z}} H(\mathbf{y}, \mathbf{z}), \quad \dot{\mathbf{z}} = \nabla_{\mathbf{y}} H(\mathbf{y}, \mathbf{z}), \tag{1}$$

where $H(\cdot,\cdot)$ is a Hamiltonian function. This class of ODEs has motivated the development of Hamiltonian-inspired DNNs in Haber and Ruthotto (2017), whose effectiveness has been shown in several benchmark classification problems (Haber and Ruthotto, 2017; Chang et al., 2019, 2018).

However, all these works consider only restricted classes of skew-symmetric maps or particular Hamiltonian functions, which, together with the specific structure of the dynamics in (1), limit the representation power of the resulting DNNs.

In this work, we leverage general models of Hamiltonian systems (van der Schaft, 2017) and provide a unified framework for defining Hamiltonian DNNs (H-DNNs for short), which, under very mild assumptions, encompasses anti-symmetric and Hamiltonian-inspired networks. Furthermore, we define new classes of DNNs, which are more expressive than those in Haber and Ruthotto (2017) and Chang et al. (2018, 2019), and can achieve comparable performance while using less layers. We show this feature using several benchmark classification problems, including digit recognition based on the MNIST dataset.

Hamiltonian dynamics can be marginally stable by construction, independent of the specific model parameters. We leverage this property and the use of regularized loss functions penalizing the variation of weights over consecutive layers (Haber and Ruthotto, 2017) for studying the well-posedness of the training process. To this purpose, we first consider the simplified setting where DNN weights are constant across layers, which corresponds to letting the regularization parameter grow to infinity. By analysing the underlying ODE, we prove the marginal stability of the backward gradient dynamics, which implies the absence of vanishing/exploding gradients during training. In addition, we present a simulation study showing that this property is also verified when the regularisation parameter is finite and network parameters can change across layers.

The reminder of our paper is organized as follows. Related works are discussed in Section 2. In Section 3, we present H-DNNs, and analyse the stability properties of the backward gradient dynamics. Numerical examples are provided in Section 4, which is followed by concluding remarks in Section 5. Throughout this work, we use the column convention for gradients, i.e. the gradient ∇f of a real-valued function $f(\mathbf{x})$ is a column vector.

2. Related works

2.1. DNN induced by ODE discretization

The connection between neural networks and ODEs can be established by starting from the nonlinear system

$$\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t), \boldsymbol{\theta}(t)), \quad \mathbf{y}(0) = \mathbf{y}_0, \quad \mathbf{y}(t) \in \mathbb{R}^n, \quad 0 \leq t \leq T, \tag{2}$$

where $\boldsymbol{\theta}(t) \in \mathbb{R}^{n_\theta}$ is a vector of parameters. For a given $N \in \mathbb{N}$, we consider the forward Euler discretization of (2) with time step $h = \frac{T}{N} > 0$, giving

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h\,\mathbf{f}(\mathbf{y}_j, \boldsymbol{\theta}_j), \qquad \text{for } j = 0, 1, \ldots, N-1. \tag{3}$$

Equation (3) can be seen as the model of a residual neural network with N layers (He et al., 2016), where $\mathbf{y}_j, \mathbf{y}_{j+1} \in \mathbb{R}^n$ are the input and output vectors of layer j, respectively. We highlight that, for the discretization of (2), one could replace the forward Euler method with different discretization methods, hence obtaining different DNN architectures (Haber and Ruthotto, 2017). In DNNs, (3)

is usually complemented with an output layer $\mathbf{y}_{N+1} = \mathbf{f}_N(\mathbf{y}_N, \boldsymbol{\theta}_N)$ that depends on the nature of the learning problem (e.g., regression or classification).

DNN training is performed by computing the network weights that minimize a loss function $\mathcal{L}(\mathbf{y}_{N+1}^1, \dots, \mathbf{y}_{N+1}^s, \boldsymbol{\theta})$, where $\{1, \dots, s\}$ is the index set of samples used to optimize the network weights and $\boldsymbol{\theta}$ collects all DNN parameters. A remarkable feature of ODE-inspired DNNs is that their properties can be studied, albeit in an approximate way, in terms of the continuous-time nonlinear system (2), which is often easier to analyse than (3) (Haber and Ruthotto, 2017).

2.2. Vanishing/exploding gradients

An obstacle that is commonly faced when training DNNs using gradient based optimization methods, is the problem of exploding/vanishing gradients. Gradient descent methods update the vector $\boldsymbol{\theta}$ as

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \gamma \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L} \tag{4}$$

where $\gamma > 0$ is the optimization step size. In particular, by using the chain rule, the gradient of the loss function w.r.t. the parameter i of layer j can be obtained as

$$\frac{\partial \mathcal{L}}{\partial \theta_{i,j}} = \frac{\partial \mathbf{y}_{j+1}}{\partial \theta_{i,j}} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{j+1}} = \frac{\partial \mathbf{y}_{j+1}}{\partial \theta_{i,j}} \left(\prod_{l=j+1}^{N-1} \frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{y}_l} \right) \frac{\partial \mathcal{L}}{\partial \mathbf{y}_N}. \tag{5}$$

The problem of vanishing/exploding gradients is commonly related to the layer gradient magnitudes $\left\| \frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{y}_l} \right\|_2$, $l = 0, \dots, N-1$. If these terms are all very small, since $\left\| \prod_{l=j+1}^{N-1} \frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{y}_l} \right\|_2 \leq \prod_{l=j+1}^{N-1} \left\| \frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{y}_l} \right\|_2$, the gradient $\frac{\partial \mathcal{L}}{\partial \theta_{i,j}}$ vanishes, and the training stops. Vice versa, if these terms are very large, $\frac{\partial \mathcal{L}}{\partial \theta_{i,j}}$ becomes very sensitive to perturbations in the vectors $\frac{\partial \mathbf{y}_{j+1}}{\partial \theta_{i,j}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{y}_N}$, and this can make the learning process unstable or cause overflow issues. Both problems are generally exacerbated when the number of layers N is large (Goodfellow et al., 2016).

For analysing the phenomenon of exploding/vanishing gradients in the context of *recurrent* neural networks, Chang et al. (2019) consider the system (2) and show that the matrix $\boldsymbol{\phi}(t, 0) = \frac{\partial \mathbf{y}(t)}{\partial \mathbf{y}(0)} \in \mathbb{R}^{n \times n}$ obeys the linear time-varying dynamics

$$\dot{\boldsymbol{\phi}}(t, 0) = \boldsymbol{\mathcal{J}}(t) \boldsymbol{\phi}(t, 0), \quad \boldsymbol{\phi}(0, 0) = \mathbf{I}, \tag{6}$$

where $\boldsymbol{\mathcal{J}}(t) = \frac{\partial^\top \mathbf{f}(\mathbf{y}, t)}{\partial \mathbf{y}(t)}$. In particular, $\boldsymbol{\phi}(t, 0)$ can be seen as the continuous-time counterpart of the neural network gradient $\frac{\partial \mathbf{y}_k}{\partial \mathbf{y}_0}$, which is similar to the terms appearing in (5). For the sake of simplicity, let us consider the simpler case where $\boldsymbol{\mathcal{J}}(t) = \boldsymbol{\mathcal{J}}$ is time-invariant. The properties that $\|\boldsymbol{\phi}(t, 0)\|_2$ neither diverges nor vanishes as $t \to +\infty$ corresponds to the marginal stability of (6), which is equivalent to requiring that each eigenvalue of $\boldsymbol{\mathcal{J}}$ has zero real part and its geometric and algebraic multiplicity coincide (see, e.g. Khalil (2002)). Under suitable assumptions, similar conditions can be also reached if $\boldsymbol{\mathcal{J}}(t)$ varies slowly enough in time (Ascher, 2008).

2.3. Anti-symmetric and Hamiltonian-inspired DNNs

Motivated by the goal of mitigating the problem of vanishing/exploding gradients, as well as of having a marginally stable forward dynamics (2)[1], various DNN architectures have been proposed. They are summarized below, where matrices $\mathbf{K}$ and $\mathbf{b}$ denote the trainable parameters and $\sigma(\cdot) : \mathbb{R} \to \mathbb{R}$ is an activation function applied element wise to a vector argument. The network structures are called MS$_i$-DNN ($i = 1, 2, 3$) and, for each of them, the underlying ODE as well as the discretization method used are specified.

- MS$_1$-DNN (Haber and Ruthotto, 2017)
  - Layer equation: $\mathbf{z}_{j+1} = \mathbf{z}_j - h\sigma(\mathbf{K}_{j,0}^\top \mathbf{y}_j + \mathbf{b}_{j,1})$ and $\mathbf{y}_{j+1} = \mathbf{y}_j + h\sigma(\mathbf{K}_{j,0}\mathbf{z}_{j+1} + \mathbf{b}_{j,2})$
  - Underlying ODE: $\left[\begin{smallmatrix}\dot{\mathbf{y}}\\ \dot{\mathbf{z}}\end{smallmatrix}\right](t) = \sigma\left(\left[\begin{smallmatrix}\mathbf{0} & \mathbf{K}_0(t)\\ -\mathbf{K}_0^\top(t) & \mathbf{0}\end{smallmatrix}\right]\left[\begin{smallmatrix}\mathbf{y}\\ \mathbf{z}\end{smallmatrix}\right](t) + \left[\begin{smallmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{smallmatrix}\right](t)\right)$
  - Discretization method: Verlet
- MS$_2$-DNN (Haber and Ruthotto, 2017; Chang et al., 2019)
  - Layer equation: $\mathbf{y}_{j+1} = \mathbf{y}_j + h\sigma(\mathbf{K}_j\mathbf{y}_j + \mathbf{b}_j)$ where all $\mathbf{K}_j$ matrices are skew-symmetric
  - Underlying ODE: $\dot{\mathbf{y}}(t) = \sigma(\mathbf{K}(t)\mathbf{y}(t) + \mathbf{b}(t))$ where $\mathbf{K}(t)$ are skew-symmetric $\forall t \geq 0$
  - Discretization method: forward Euler
- MS$_3$-DNN (Chang et al., 2018)
  - Layer equation: $\mathbf{y}_{j+1} = \mathbf{y}_j + h\mathbf{K}_{j,1}^\top \sigma(\mathbf{K}_{j,1}\mathbf{z}_j + \mathbf{b}_{j,1})$ and $\mathbf{z}_{j+1} = \mathbf{z}_j - h\mathbf{K}_{j,2}^\top \sigma(\mathbf{K}_{j,2}\mathbf{y}_{j+1} + \mathbf{b}_{j,2})$
  - Underlying ODE: $\left[\begin{smallmatrix}\dot{\mathbf{y}}\\ \dot{\mathbf{z}}\end{smallmatrix}\right](t) = \left[\begin{smallmatrix}\mathbf{K}_1^\top & \mathbf{0}\\ \mathbf{0} & -\mathbf{K}_2^\top\end{smallmatrix}\right](t)\, \sigma\left(\left[\begin{smallmatrix}\mathbf{0} & \mathbf{K}_1\\ \mathbf{K}_2 & \mathbf{0}\end{smallmatrix}\right](t)\left[\begin{smallmatrix}\mathbf{y}\\ \mathbf{z}\end{smallmatrix}\right](t) + \left[\begin{smallmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{smallmatrix}\right](t)\right)$
  - Discretization method: Verlet

It is worth to remark that these networks can achieve very good performance on different classification problems including benchmark problems in image classification such as MNIST and CIFAR 10 (Haber and Ruthotto, 2017; Chang et al., 2018, 2019). We highlight that models MS$_1$ and MS$_3$ have been called *Hamiltonian-inspired* in view of their similarity with Hamiltonian models (compare, e.g. the underlying ODE of MS$_1$-DNN and (1)). However, Haber and Ruthotto (2017); Chang et al. (2018) do not provide a precise Hamiltonian function for the corresponding ODEs.

3. Hamiltonian neural networks (H-DNNs)

In this section, we consider a general class of continuous time Hamiltonian system that we utilize for defining new DNN architectures. We also show that, under weak assumptions, MS$_i$-DNN models, $i = 1, 2, 3$, can be obtained as special cases of the proposed networks. Finally, by assuming constant weights, we analyse marginal stability of the backward gradient dynamics, and provide arguments for supporting the claim that vanishing/exploding gradients are not expected during training.

1. As shown in Haber and Ruthotto (2017), this property guarantees reduced sensitivity to perturbations and adversarial attacks on the input features.

3.1. Hamiltonian dynamics

We consider time-varying Hamiltonian systems (Guo and Cheng, 2006) defined by the ODE

$$\dot{\mathbf{y}}(t) = \mathbf{J}(\mathbf{y}, t)\frac{\partial H(\mathbf{y}, t)}{\partial \mathbf{y}} \tag{7}$$

where the interconnection matrix $\mathbf{J}(\mathbf{y}, t) \in \mathbb{R}^{n\times n}$ is skew-symmetric i.e. $\mathbf{J}(\mathbf{y}, t) = -\mathbf{J}^\top(\mathbf{y}, t)$ $\forall t \geq 0$, and $H(\mathbf{y}(t), t) \in \mathbb{R}$ is the Hamiltonian function. Both $\mathbf{J}$ and H are assumed to be smooth functions of all their arguments.

The more common notion of a time-invariant Hamiltonian system (van der Schaft, 2017) can be recovered when $\mathbf{J}$ and H do not depend upon time. Time-invariant Hamiltonian systems are marginally stable by construction when $H(\mathbf{y})$ is a positive definite function (Khalil, 2002). Therefore, as discussed in Section 2.2, they are a good candidate for defining well-posed DNNs. The same is true for the time-varying model (7), provided that the Hamiltonian changes slowly enough over time.

In the sequel, we focus on the following energy function

$$H(\mathbf{y}(t), t) = \left[\log(\cosh(\mathbf{K}(t)\mathbf{y}(t) + \mathbf{b}(t)))\right]^\top \mathbf{1} \tag{8}$$

where $\log(\cdot)$ and $\cosh(\cdot)$ are applied element-wise, and $\mathbf{1} = [1, \ldots, 1]^\top$. We obtain

$$\frac{\partial H(\mathbf{y}(t), t)}{\partial \mathbf{y}(t)} = \frac{\partial(\mathbf{K}(t)\mathbf{y}(t) + \mathbf{b}(t))}{\partial \mathbf{y}(t)} \frac{\partial H(\mathbf{y}(t), t)}{\partial(\mathbf{K}(t)\mathbf{y}(t) + \mathbf{b}(t))} = \mathbf{K}^\top(t)\tanh(\mathbf{K}(t)\mathbf{y}(t) + \mathbf{b}(t)) \tag{9}$$

where $\tanh(\cdot)$ is applied element-wise. Hence, system (7) becomes

$$\dot{\mathbf{y}}(t) = \mathbf{J}(\mathbf{y}, t)\mathbf{K}^\top(t)\tanh(\mathbf{K}(t)\mathbf{y}(t) + \mathbf{b}(t)). \tag{10}$$

3.2. H-DNNs: relations with existing networks and new architectures

We show that the underlying ODEs of the MS$_i$-DNNs (see Section 2.3) are particular instances of (10) when $\sigma(\cdot) = \tanh(\cdot)$ and

- for MS$_1$-DNN, $\mathbf{K}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{K}_0(t) \\ -\mathbf{K}_0^\top(t) & \mathbf{0} \end{bmatrix}$ is invertible $\forall t \geq 0$ and $\mathbf{J}(\mathbf{y}, t)\mathbf{K}^\top(t) = \mathbf{I}$,
- for MS$_2$-DNN, $\mathbf{K}(t) = -\mathbf{K}^\top(t)$ is invertible $\forall t \geq 0$ and $\mathbf{J}(\mathbf{y}, t)\mathbf{K}^\top(t) = \mathbf{I}$,
- for MS$_3$-DNN, $\mathbf{K}(t) = \begin{bmatrix} \mathbf{0} & \mathbf{K}_1(t) \\ \mathbf{K}_2(t) & \mathbf{0} \end{bmatrix}$ and $\mathbf{J}(\mathbf{y}, t) = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{bmatrix}$.

A necessary condition for the skew-symmetric $n \times n$ matrix $\mathbf{K}(t)$ to be invertible is that the size n of input features is even[2]. If n is odd, however, one can perform input-feature augmentation by adding an extra state initialized at zero to satisfy the previous condition (Dupont et al., 2019).

Next, we introduce two new architectures (called H$_i$-DNN, $i = 1, 2$) stemming from (10) when $\mathbf{J}(\mathbf{y}, t)$ is constant and forward Euler discretization with step $h > 0$ is applied. The resulting layer equations are

$$\mathbf{y}_{j+1} = \mathbf{y}_j + h\,\mathbf{J}\mathbf{K}_j^\top \tanh(\mathbf{K}_j\mathbf{y}_j + \mathbf{b}_j) \qquad j = 0, 1, \cdots, N-1 \tag{11}$$

2. For a $n \times n$ skew-symmetric matrix $\mathbf{A}$ we have, $\det(\mathbf{A}) = \det(\mathbf{A}^\top) = \det(\mathbf{A}^{-1}) = (-1)^n \det(\mathbf{A})$. If n is odd, then $\det(\mathbf{A}) = -\det(\mathbf{A}) = 0$. Thus, $\mathbf{A}$ is not invertible.

where we set $\mathbf{J}(\mathbf{y},t) = \left[\begin{smallmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{smallmatrix}\right]$ for H_1-DNN, and $\mathbf{J}(\mathbf{y},t) = \begin{bmatrix} 0 & 1 & \dots & 1 \\ -1 & 0 & \dots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & 0 \end{bmatrix}$ for H_2-DNN.

In spite of the specific choices of $\mathbf{J}$, both DNNs contain more trainable parameters in each layer than MS_i-DNN, $i = 1, 2, 3$. In this sense, they are more expressive than MS_i architectures and, as shown later in Section 4.1, one can use less layers while obtaining similar prediction accuracy.

Remark 1 *Although it is not guaranteed that Euler discretization preserves marginal stability of the continuous-time dynamics, it leads to simpler layer equations compared to more sophisticated discretization approaches, and can achieve good performance on benchmark examples (see Section 4.1). Moreover the discretization accuracy can be controlled through T and h. These features may be attractive to practitioners.*

3.3. Training algorithm

For all DNN architectures introduced in Section 3.2, we consider multicategory classification problems where M is the number of classes, and the input features and their corresponding true labels are $(\mathbf{y}_0^k, c^k), k = 1, \dots, s, c^k \in \{0, \dots, M-1\}$. The networks are trained by solving the optimization problem

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \frac{1}{s}\sum_{k=1}^{s} \mathcal{L}(\mathbf{f}_N(\mathbf{y}_N^k), c^k) + \alpha_c\, R_N(\boldsymbol{\theta}_N) + \alpha\, R(\mathbf{K}_{0,\dots,N-1}, \mathbf{b}_{0,\dots,N-1}) \\ \text{s.t.} \quad & \mathbf{y}_{j+1}^k = \mathbf{y}_j^k + h\,\mathbf{J}_j(\mathbf{y}_j^k)\mathbf{K}_j^\top \tanh(\mathbf{K}_j\mathbf{y}_j^k + \mathbf{b}_j), \quad j = 0, 1, \dots, N-1 \end{aligned} \tag{12}$$

where $R_N(\cdot)$ is the L_2 regularization term of the output layer[3] and $R(\cdot)$ is the regularization term of layers $0, \dots, N-1$. The output layer is problem dependent, e.g. for a two class classification problem, it is usually given by $f_N(\mathbf{y}_N, \boldsymbol{\theta}_N) = \sigma_c(\mathbf{W}\mathbf{y}_N + \mu)$ where $\mathbf{W} \in \mathbb{R}^{1\times n}$, $\mu \in \mathbb{R}$, $\boldsymbol{\theta}_N = (\mathbf{W}, \mu)$ and $\sigma_c(x) = \frac{1}{1+e^{-x}}$ is the logistic function. The minimization is done over $\boldsymbol{\theta}$, i.e., all the trainable parameters that define the network $\{\mathbf{K}_{0,\dots,N-1}, \mathbf{b}_{0,\dots,N-1}, \mathbf{W}, \mu\}$.

Following the work in Haber and Ruthotto (2017) and Chang et al. (2018), we define the regularization term for the H-DNNs as $R = R_K(\mathbf{K}_{0,\dots,N-1}) + R_b(\mathbf{b}_{0,\dots,N-1})$, where

$$R_K(\mathbf{K}_{0,\dots,N-1}) = \frac{h}{2}\sum_{j=1}^{N-1} \|\mathbf{K}_j - \mathbf{K}_{j-1}\|_F^2 \quad \text{and} \quad R_b(\mathbf{b}_{0,\dots,N-1}) = \frac{h}{2}\sum_{j=1}^{N-1} \|\mathbf{b}_j - \mathbf{b}_{j-1}\|^2 \tag{13}$$

so as to favour weights that vary smoothly between adjacent layers. The coefficients $\alpha \geq 0$ and $\alpha_c \geq 0$[4] are hyperparameters that represent the trade-off between fitting and regularization.

3.4. Stability of the backward gradient dynamics

As discussed in Section 2.2, to avoid vanishing/exploding gradients, we would like to ensure that the following terms are not vanishing nor exploding

$$\left(\prod_{l=j+1}^{N-1} \frac{\partial \mathbf{y}_{l+1}}{\partial \mathbf{y}_l}\right) = \frac{\partial \mathbf{y}_N}{\partial \mathbf{y}_{j+1}} \text{ for } j = N-2, \dots, 0. \tag{14}$$

3. For a two class classification problem, it is given by $R_N(\cdot) = \|\mathbf{W}\|^2 + \mu^2$. For multicategory problems, we refer the reader to Haber and Ruthotto (2017).
4. α_c is usually called *weight decay*.

This analysis can also be tackled from the continuous-time perspective by considering (2) and noting that (14) corresponds to $\frac{\partial \mathbf{y}(T)}{\partial \mathbf{y}(T-t)}$, where $t = h(j+1)$, $T = h(N-1)$ and h is the step size.

We call the evolution of $\boldsymbol{\phi}(T, T-t) \triangleq \frac{\partial \mathbf{y}(T)}{\partial \mathbf{y}(T-t)}$ the *backward gradient dynamics* because $T-t$ decreases from T to zero as t increases. This term is different from the one considered in (6) which captures the sensitivity to input features and not the evolution across layers of gradients appearing in backpropagation.

Our next goal is to obtain the dynamics of $\boldsymbol{\phi}(T, T-t)$ for the Hamiltonian model (10). We start from the simple case where the parameters of (10) are constant, i.e. $\mathbf{J}(\mathbf{y}(t), t) = \mathbf{J}$, $\mathbf{K}(t) = \mathbf{K}$, $\mathbf{b}(t) = \mathbf{b}$, and call the corresponding networks *time-invariant H-DNNs*. The following Lemma, whose proof can be found in Appendix A of Galimberti et al. (2021), provides the desired model.

Lemma 2 *Given the time-invariant ODE $\dot{\mathbf{y}}(t) = \mathbf{f}(\mathbf{y}(t))$, the time evolution of $\boldsymbol{\phi}(T, T-t)$ is given by*

$$\frac{d}{dt}\boldsymbol{\phi}(T, T-t) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_{\mathbf{y}(T-t)} \boldsymbol{\phi}(T, T-t)\,, \quad \boldsymbol{\phi}(T, T) = \mathbf{I}. \tag{15}$$

Since in our case $\mathbf{f}(\mathbf{y}(t)) = \mathbf{J}\mathbf{K}^\top \tanh(\mathbf{K}\mathbf{y}(t) + \mathbf{b})$, we have

$$\begin{aligned}\frac{\partial \mathbf{f}}{\partial \mathbf{y}} &= \frac{\partial}{\partial \mathbf{y}}\left(\mathbf{K}^\top \tanh(\mathbf{K}\mathbf{y} + \mathbf{b})\right)\mathbf{J}^\top = \frac{\partial}{\partial \mathbf{y}}\left(\tanh(\mathbf{K}\mathbf{y} + \mathbf{b})\right)\mathbf{K}\mathbf{J}^\top \\ &= \mathbf{K}^\top \mathrm{diag}\left(\tanh'(\mathbf{K}\mathbf{y} + \mathbf{b})\right)\mathbf{K}\mathbf{J}^\top = \mathbf{K}^\top \mathbf{D}(\mathbf{y})\mathbf{K}\mathbf{J}^\top\end{aligned} \tag{16}$$

where $\mathbf{D}(\mathbf{y}) = \mathrm{diag}\left(\tanh'(\mathbf{K}\mathbf{y} + \mathbf{b})\right)$ and $\tanh'(\cdot)$ computes element-wise the derivative of $\tanh(\cdot)$.

The next two lemmas, proved in Appendix A of Galimberti et al. (2021), show that the Jacobian matrix (16) satisfies the conditions for marginal stability.

Lemma 3 *The eigenvalues of $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ are purely imaginary.*

Lemma 4 *The algebraic and geometric multiplicity of each eigenvalue of $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ do coincide.*

As shown in the corresponding proofs, Lemma 3 hinges on results available in Chang et al. (2019). However, the multiplicity of the eigenvalues of $\frac{\partial \mathbf{f}}{\partial \mathbf{y}}$ (Lemma 4) has not been analysed in previous publications.

If $\mathbf{D}(\mathbf{y}(t))$ is constant, Lemmas 3 and 4 imply that $\boldsymbol{\phi}(T, T-t)$, neither diverges nor converges to zero, irrespectively of the weights $\mathbf{J}$, $\mathbf{K}$ and $\mathbf{b}$ and the final time T. This property, however, can be compromised by the time-varying nature of $\mathbf{D}(\mathbf{y}(t))$ and the weights ($\mathbf{K}(t)$ and $\mathbf{b}(t)$) as well as the time discretization process underlying (14). Nevertheless, Lemmas 3 and 4 suggest that if $\mathbf{D}(\mathbf{y}(t))$, $\mathbf{K}(t)$ and $\mathbf{b}(t)$ change slowly enough and h is sufficiently small, the growth or decrease of the terms $\left\|\frac{\partial \mathbf{y}_N}{\partial \mathbf{y}_{j+1}}\right\|_2$ can be kept under control.

While we do not provide a complete theoretical analysis when $\mathbf{D}(\mathbf{y}(t))$, $\mathbf{K}(t)$ and $\mathbf{b}(t)$ are time varying, we simply highlight that changes in the parameters $\mathbf{K}_j$ and $\mathbf{b}_j$ across the layers of H$_i$-DNN, $i = 1, 2$, can be controlled by suitably choosing the regularization parameter α in (12). Moreover, in Section 4.3, we provide a simulation study of the backward gradient dynamics confirming the absence of vanishing/exploding gradients when $\mathbf{K}_j$ and $\mathbf{b}_j$ are not constant.

Table 1: Classification accuracies over test sets for different examples using different network structures with 4 neurons (nf) each layer. The first three columns represent the existing structures while in the two last columns we present the results for the new H-DNNs. The first two best accuracies in each row are in bold. Last row presents the number of parameters per layer of each network.

		MS_1-DNN	MS_2-DNN	MS_3-DNN	H_1-DNN	H_2-DNN
Swiss roll	4 layers	77.1%	79.7%	90.1%	**93.6%**	**98.9%**
	8 layers	91.5 %	90.7%	87.0%	**99.0%**	**99.4%**
	16 layers	97.7%	99.7%	97.1%	**99.8%**	**99.8%**
	32 layers	**100%**	**100%**	98.4%	**99.8%**	99.2%
	64 layers	**100%**	**100%**	**100%**	**99.8%**	**100%**
Double moons	1 layer	92.5%	91.3%	97.6%	**100%**	**99.9%**
	2 layers	98.2%	94.9%	**99.8%**	**100%**	**100%**
	4 layers	**99.5%**	**100%**	**100%**	**100%**	**100%**
# parameters per layer		$\frac{nf^2}{4} + nf$	$\frac{nf^2+nf}{2}$	$\frac{nf^2}{2} + nf$	$nf^2 + nf$	$nf^2 + nf$

4. Numerical examples

4.1. Binary classification examples

We test MS- and H-DNNs introduced in Section 3 on two benchmark examples (the "Swiss roll" and the "double moons" datasets in Figures 1(*a*) and 1(*b*)) concerning binary classification with features in $\mathbb{R}^2$.

As in Haber and Ruthotto (2017), we consider MS- and H-DNNs with augmented input features (Dupont et al., 2019) so as to increase the modelling power. More specifically, input feature vectors are given by $\left[(\mathbf{y}_0^k)^\top \; 0 \; 0 \right]^\top \in \mathbb{R}^4$ where $\mathbf{y}_0^k \in \mathbb{R}^2, k = 1, \ldots, s$ are the input datapoints (see Figures 1(*a*) and 1(*b*)). We complement the DNNs with an output layer $y_{N+1} = f_N(\mathbf{y}_N, \boldsymbol{\theta}_N)$ (see Section 3.3). The optimization problem is solved using the Adam algorithm with minibatches (see Appendix B of Galimberti et al. (2021) for details), and standard cross-entropy (Goodfellow et al., 2016) as the loss function $\mathcal{L}$ in (12).

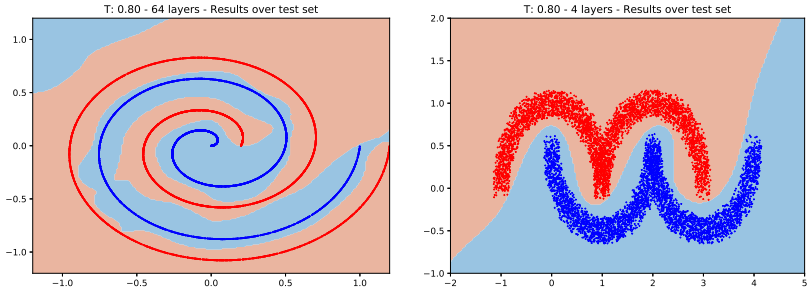
In Table 1, we present the classification accuracies over test sets for the network structures in Section 3 with different number of layers. It can be seen, for a fixed number of layers, that the performances of H_1-DNN and H_2-DNN are similar or better compared to the other networks. This can be motivated by the fact that, as discussed in Section 3.2, the new architectures are more expressive than MS_i-DNNs. We indicate in the last row of Table 1, the number of parameters per layer of each network. Note that networks with same number of parameters have similar performance.

The coloured regions in Figure 1 show the predictive power of an example network (H_1-DNN). It can be noticed that the datapoints do not lie close to the decision boundary, hence confirming the robustness of classification against perturbation of input features.

4.2. Experiments with the MNIST dataset

We evaluate our methods on a standard image classification benchmark: MNIST[5].

5. http://yann.lecun.com/exdb/mnist/



(*a*) (*b*)

Figure 1: Results for the H_1-DNN architecture with (*a*) 64 and (*b*) 4 layers. Labelled datapoints for (*a*) "Swiss roll" and (*b*) "double moons". Coloured regions representing the predictions of the trained DNNs.

Table 2: Classification accuracies over train and test sets for MNIST example using MS_1-DNN and H_2-DNN.

Number of layers	MS_1-DNN		H_2-DNN	
	Train	Test	Train	Test
0	93.730%	92.47%	93.828%	92.41%
2	99.570%	97.72%	99.815%	97.83%
4	99.970%	98.03%	99.789%	98.02%
8	99.982%	98.05%	99.707%	98.22%
16	100%	98.14%	99.503%	98.21%

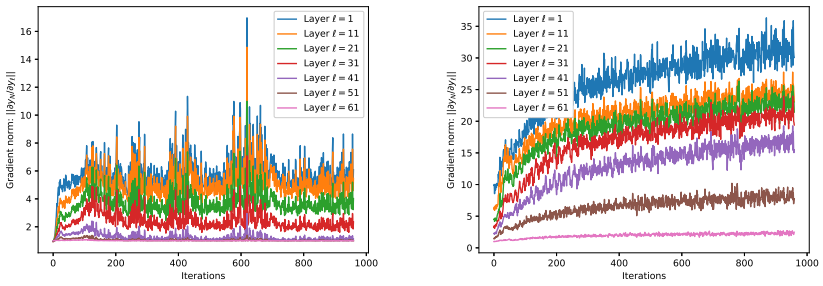
The dataset consists of 28×28 digital images in gray scale of hand-written digits from 0 to 9 with their corresponding labels. It contains 60,000 train examples and 10,000 test examples.

Following Haber and Ruthotto (2017), the network architecture consists of a convolutional layer followed by a Hamiltonian DNN and an output layer. The convolutional layer is a linear transformation that expands the data from 1 to 8 channels, and the output layer uses all the output values (i.e., no pooling is performed) for a linear transformation plus a softmax activation function to obtain a vector in $\mathbb{R}^{10}$ that represents the probabilities of the data to belong to each of the 10 classes.

For the Hamiltonian DNN, we use MS_1-DNNs and H_2-DNNs[6] with 2, 4, 8 and 16 layers. We set $h = 0.4$ for MS_1-DNNs and $h = 0.05$ for H_2-DNNs. Moreover, we include as a baseline, the results obtained when omitting the Hamiltonian DNN block, i.e., when considering only a convolutional layer followed by the output layer. The implementation details can be found in the Appendix B of Galimberti et al. (2021).

Table 2, summarizing the train and test accuracies of these networks, shows that both network structures achieve similar performance. Note that, while the training errors are almost zero, the test errors are reduced when incrementing the number of layers. Moreover, these results are in line with test accuracies obtained when using standard convolutional layers instead of Hamiltonian DNNs.

6. Similar results can be obtained using other MS or H-DNNs.



(*a*) (*b*)

Figure 2: Evolution of the 2-norm of $\frac{\partial \mathbf{y}_N}{\partial \mathbf{y}_\ell}$, $\ell = 1, 11, 21, 31, 41, 51, 61$, during the training (960 iterations) of a 64-layer (*a*) H_1-DNN and (*b*) *time-invariant* H_1-DNN.

4.3. Gradient analysis

In order to provide evidence of good numerical results during training, we analyse the evolution of the terms (14) for deep networks. At each optimization step, the gradient of the loss function with respect to each of the parameters is calculated (backward propagation). In this analysis, we study the Jacobian matrices $\frac{\partial \mathbf{y}_N}{\partial \mathbf{y}_{j+1}}$ in (14), and we plot in Figure 2 their norms for some layers and for each of the 960 iterations composing the training process when using H-DNNs.

Using "Double moons" example, we train a 64-layer H_1-DNN, a variant of the same network where we impose the parameters of all layers to coincide, i.e. a *time-invariant* network with $\mathbf{K}(t) = \mathbf{K}$ and $\mathbf{b}(t) = \mathbf{b}$, and a fully-connected neural network (FCNN) with 32 layers[7].

For the H-DNNs case, it can be seen that the norms of the terms $\frac{\partial \mathbf{y}_N}{\partial \mathbf{y}_{j+1}}$ for $j = 0, 10, \ldots, 60$ are bounded in the intervals $[1, 17]$ and $[1, 37]$ during the whole training. Results are similar when using deeper networks. Although it is not shown, we highlight that the gradients $\frac{\partial \mathcal{L}}{\partial \theta_{ij}}$ do converge to zero in approximately 500 and 300 iterations respectively, showing that the optimization algorithm has achieved a (possible local) minimum. When using FCNN, however, it can be shown that gradient norms quickly tends to zero once the network is deep enough. For instance, for the 32-layer FCNN, the training stops in approximately 200 iterations and the final test accuracy is only 50%.

5. Conclusions

We present a unified framework for DNNs based on Hamiltonian systems which encompasses existing classes of marginally stable networks. We define two new DNN structures, which are more flexible than existing ones, while having similar of better performance. We present the analysis of the backward gradient dynamics for *time-invariant* DNNs and show a simulation study for the *time-varying* case.

Our work is a first step towards the design of new families of H-DNNs since different Hamiltonian energy functions originate new architectures. Future research will also focus on the use of different discretization schemes for defining alternative layer equations.

7. See Appendix B of Galimberti et al. (2021) for implementation details.

References

Uri M. Ascher. *Numerical Methods for Evolutionary Differential Equations*. Society for Industrial and Applied Mathematics, USA, 2008. ISBN 0898716527.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI Conference on Artificial Intelligence*, 2018.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3140–3150. Curran Associates, Inc., 2019.

Weinan E. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5:1–11, Feb 2017.

Clara L. Galimberti, Liang Xu, and Giancarlo Ferrari Trecate. A unified framework for Hamlitonian deep neural networks. *arXiv preprint arXiv:2104.13166*, 2021.

Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

Yuqian Guo and Daizhan Cheng. Stabilization of time-varying Hamiltonian systems. *IEEE Transactions on Control Systems Technology*, 14(5):871–880, 2006.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, Dec 2017.

Eldad Haber, Lars Ruthotto, Elliot Holtham, and Seong-Hwan Jun. Learning across scales—multiscale methods for convolution neural networks. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Hassan K. Khalil. *Nonlinear Systems*. Prentice-Hall, Upper Saddle River, NJ, 3rd edition, 2002.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *6th International Conference on Learning Representations, ICLR 2018*, Jan 2018.

Sergio Lucia and Benjamin Karg. A deep learning-based approach to robust nonlinear model predictive control. *IFAC-PapersOnLine*, 51(20):511–516, 2018. ISSN 2405-8963.

Arjan van der Schaft. *L_2-Gain and Passivity Techniques in Nonlinear Control*. Springer, 2017.

Wayne Xiong, Jasha Droppo, Xuedong Huang, Frank Seide, Michael L. Seltzer, Andreas Stolcke, Dong Yu, and Geoffrey Zweig. The microsoft 2016 conversational speech recognition system. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5255–5259, Mar 2017.

Riccardo Zoppoli, Marcello Sanguineti, Giorgio Gnecco, and Thomas Parisini. *Neural Approximations for optimal control and decision*. Springer, 2020.

Acknowledgments

Research supported by the Swiss National Science Foundation under the NCCR Automation (grant agreement 51NF40_180545).