A unified framework for Hamiltonian deep neural networks

Clara Lucía Galimberti¹, Liang Xu¹ and Giancarlo Ferrari Trecate¹

{clara.galimberti, liang.xu, giancarlo.ferraritrecate}@epfl.ch

¹Institute of Mechanical Engineering, École Polytechnique Fédérale de Lausanne, Switzerland.
April 28, 2021

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 [1,2]. There is also a growing interest in using deep neural networks (DNNs) for approximating complex controllers [3,4]. 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 [5], 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) [6,7,8,9,10]. These results leverage the stability properties of the underlying ODE for characterizing relevant behaviours of the corresponding DNNs [6]. 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 [6,8] for defining anti-symmetric DNNs. Another example is given by dynamical systems in the form

$$\dot{y} = -\nabla_z H(y, z), \quad \dot{z} = \nabla_y H(y, z),$$

(1)

where $H(\cdot, \cdot)$ is a Hamiltonian function. This class of ODEs has motivated the development of Hamiltonian-inspired DNNs in [6], whose effectiveness has been shown in several benchmark classification problems [6,8,11].

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 [11], limit the representation power of the resulting DNNs.
In this work, we leverage general models of Hamiltonian systems [12] 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 [6] and [11, 8], 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 [6] 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 $\nabla y$ of a real-valued function $f(x)$ 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

$$\dot{y}(t) = f(y(t), \theta(t)), \quad y(0) = y_0, \quad y(t) \in \mathbb{R}^n, \quad 0 \leq t \leq T, \quad (2)$$

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

$$y_{j+1} = y_j + h f(y_j, \theta_j), \quad \text{for } j = 0, 1, \ldots, N - 1. \quad (3)$$

Equation (3) can be seen as the model of a residual neural network with $N$ layers [11], where $y_j, y_{j+1} \in \mathbb{R}^n$ 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 [6]. In DNNs, (3) is usually complemented with an output layer $y_{N+1} = f_N(y_N, \theta_N)$ 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

$$\mathcal{L}(y_{N+1}, \ldots, y_{N+1}, \theta),$$

where $\{1, \ldots, s\}$ is the index set of samples used to optimize the network weights and $\theta$ 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) [6].

### 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 $\theta$ as

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

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

$$\frac{\partial \mathcal{L}}{\partial \theta_{i,j}} = \frac{\partial y_{j+1}}{\partial \theta_{i,j}} \frac{\partial \mathcal{L}}{\partial y_{j+1}} = \frac{\partial y_{j+1}}{\partial \theta_{i,j}} \left( \prod_{l=j+1}^{N-1} \frac{\partial y_{l+1}}{\partial y_{l}} \right) \frac{\partial \mathcal{L}}{\partial y_{N}}. \quad (5)$$

2
The problem of vanishing/exploding gradients is commonly related to the layer gradient magnitudes \( \left\| \frac{\partial y_l}{\partial y_{l-1}} \right\|_2 \), \( l = 0, \ldots, N - 1 \). If these terms are all very small, since \( \prod_{l=j+1}^{N-1} \left\| \frac{\partial y_{l+1}}{\partial y_l} \right\|_2 \leq \prod_{l=j+1}^{N-1} \left\| \frac{\partial y_{l+1}}{\partial y_l} \right\|_2 \), the gradient \( \frac{\partial C}{\partial \theta_{i,j}} \) vanishes, and the training stops. Vice versa, if these terms are very large, \( \frac{\partial C}{\partial \theta_{i,j}} \) becomes very sensitive to perturbations in the vectors \( \frac{\partial y_{l+1}}{\partial y_l} \) and \( \frac{\partial C}{\partial \theta_{i,j}} \), 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 \cite{13}.

For analysing the phenomenon of exploding/vanishing gradients in the context of recurrent neural networks, the authors of \cite{8} consider the system (2) and show that the matrix \( \phi(t, 0) = \frac{\partial y(t)}{\partial y(0)} \in \mathbb{R}^{n \times n} \) obeys the linear time-varying dynamics

\[
\dot{\phi}(t, 0) = J(t)\phi(t, 0), \quad \phi(0, 0) = I,
\]

where \( J(t) = \frac{\partial^2 f(y, t)}{\partial y(t)} \). In particular, \( \phi(t, 0) \) can be seen as the continuous-time counterpart of the neural network gradient \( \frac{\partial y(t)}{\partial y(0)} \), which is similar to the terms appearing in (5). For the sake of simplicity, let us consider the simpler case where \( J(t) = J \) is time-invariant. The properties that \( \|\phi(t, 0)\|_2 \) neither diverges nor vanishes as \( t \to +\infty \) correspond to the marginal stability of (3), which is equivalent to requiring that each eigenvalue of \( J \) has zero real part and its geometric and algebraic multiplicity coincide (see, e.g. \cite{14}). Under suitable assumptions, similar conditions can be also reached if \( J(t) \) varies slowly enough in time \cite{15}.

### 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 \cite{24}, various DNN architectures have been proposed. They are summarized below, where matrices \( K \) and \( b \) 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.

- **MS1-DNN** \cite{6}
  - Layer equation: \( x_{j+1} = x_j - h\sigma(K_{j,0}y_j + b_{j,1}) \) and \( y_{j+1} = y_j + h\sigma(K_{j,0}x_{j+1} + b_{j,2}) \)
  - Underlying ODE: \( \frac{\dot{y}}{y}(t) = \sigma \left( \begin{bmatrix} 0 & 
abla \sigma(t) \end{bmatrix} \right), \frac{\dot{x}}{x}(t) = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \)
  - Discretization method: Verlet

- **MS2-DNN** \cite{6, 8}
  - Layer equation: \( y_{j+1} = y_j + h\sigma(K_{j,0}y_j + b_{j}) \) where all \( K_j \) matrices are skew-symmetric
  - Underlying ODE: \( y(t) = \sigma(K(t)y(t) + b(t)) \) where \( K(t) \) are skew-symmetric \( \forall t \geq 0 \)
  - Discretization method: forward Euler

- **MS3-DNN** \cite{11}
  - Layer equation: \( x_{j+1} = x_j + hK_{j,1}\sigma(K_{j,1}z_j + b_{j,1}) \) and \( z_{j+1} = z_j - hK_{j,2}\sigma(K_{j,2}y_{j+1} + b_{j,2}) \)
  - Underlying ODE: \( \frac{\dot{y}}{y}(t) = \begin{bmatrix} 0 \\ -K_2 \end{bmatrix} \left( \begin{bmatrix} 0 & K_1 \\ K_2 & 0 \end{bmatrix} \right) \) \( \frac{\dot{x}}{x}(t) = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \)
  - 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 \cite{6, 11, 8}. We highlight that models MS1 and MS3 have been called Hamiltonian-inspired in view of their similarity with Hamiltonian models (compare, e.g. the underlying ODE of MS1-DNN and (1)). However, \cite{6, 11} do not provide a precise Hamiltonian function for the corresponding ODEs.

\footnote{As shown in \cite{8}, this property guarantees reduced sensitivity to perturbations and adversarial attacks on the input features.}
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 vanishing/exploding of the backward gradient dynamics, and provide arguments for supporting the claim that vanishing/exploding gradients are not expected during training.

3.1 Hamiltonian dynamics

We consider time-varying Hamiltonian systems [16] defined by the ODE

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

(7)

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

The more common notion of a time-invariant Hamiltonian system [12] can be recovered when $J$ and $H$ do not depend upon time. Time-invariant Hamiltonian systems are marginally stable by construction when $H(y)$ is a positive definite function [13]. 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

$$H(y(t), t) = [\log(\cosh(K(t)y(t) + b(t)))]^\top 1$$

(8)

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

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

(9)

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

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

(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, $K(t) = \begin{bmatrix} 0 & K_0(t) \\ -K_0^\top(t) & 0 \end{bmatrix}$ is invertible $\forall t \geq 0$ and $J(y, t)K^\top(t) = I$,
- for MS$_2$-DNN, $K(t) = -K_1^\top(t)$ is invertible $\forall t \geq 0$ and $J(y, t)K^\top(t) = I$,
- for MS$_3$-DNN, $K(t) = \begin{bmatrix} 0 & K_2(t) \\ K_3(t) & 0 \end{bmatrix}$ and $J(y, t) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

A necessary condition for the skew-symmetric $n \times n$ matrix $K(t)$ to be invertible is that the size $n$ of input features is even.\footnote{For a $n \times n$ skew-symmetric matrix $A$ we have, $\det(A) = \det(A^\top) = \det(A^{-1}) = (-1)^n \det(A)$. If $n$ is odd, then $\det(A) = -\det(A) = 0$. Thus, $A$ is not invertible.} If $n$ is odd, however, one can perform input-feature augmentation by adding an extra state initialized at zero to satisfy the previous condition [17].

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

$$y_{j+1} = y_j + hJK_j^\top \tanh(K_jy_j + b_j) \quad j = 0, 1, \ldots, N - 1$$

(11)

where we set $J(y, t) = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$ for H$_1$-DNN, and $J(y, t) = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ -1 & 0 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \cdots & 0 \end{bmatrix}$ for H$_2$-DNN.

In spite of the specific choices of $J$, 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 multiclassification problems where $M$ is the number of classes, and the input features and their corresponding true labels are $(y^k_i, c^k), k = 1, \ldots, s, c^k \in \{0, \ldots, M - 1\}$. The networks are trained by solving the optimization problem

$$
\min_{\theta} \frac{1}{s} \sum_{k=1}^{s} \mathcal{L}(f_N(y^k_N), c^k) + \alpha_c R_N(\theta_N) + \alpha R(K_{0, \ldots, N-1}, b_{0, \ldots, N-1})
$$

s.t. \quad y^k_{t+1} = y^k_t + h J_j(y^k_j)K_j \tanh(K_j y^k_j + b_j), \quad j = 0, 1, \ldots, N - 1 \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 $0, \ldots, N - 1$. The output layer problem dependent, e.g. for a two class classification problem, it is usually given by $f_N(y_N, \theta_N) = \sigma_c(W y_N + \mu)$ where $W \in \mathbb{R}^{1 \times n}$, $\mu \in \mathbb{R}$, $\theta_N = (W, \mu)$ and $\sigma_c(x) = \frac{1}{1 + e^{-x}}$ is the logistic function. The minimization is done over $\theta$, i.e., all the trainable parameters that define the network $(K_{0, \ldots, N-1}, b_{0, \ldots, N-1}, W, \mu)$.

Following the work in [6] and [11], we define the regularization term for the H-DNNs as $R = R_K(K_{0, \ldots, N-1}) + R_0(b_{0, \ldots, N-1})$, where

$$
R_K(K_{0, \ldots, N-1}) = \frac{h}{2} \sum_{j=1}^{N-1} \| K_j - K_{j-1} \|_F^2 \quad \text{and} \quad R_0(b_{0, \ldots, N-1}) = \frac{h}{2} \sum_{j=1}^{N-1} \| b_j - b_{j-1} \|_2^2 \tag{13}
$$

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 2.3 to avoid vanishing/exploding gradients, we would like to ensure that the following terms are not vanishing nor exploding

$$
\left( \prod_{t=j+1}^{N-1} \frac{\partial y_{t+1}}{\partial y_t} \right) = \frac{\partial y_N}{\partial y_{j+1}} \quad \text{for} \quad j = N - 2, \ldots, 0. \tag{14}
$$

This analysis can also be tackled from the continuous-time perspective by considering \([2]\) and noting that \([13]\) corresponds to $\frac{\partial y_N}{\partial y_{T-t}}$, where $t = h(j + 1)$, $T = h(N - 1)$ and $h$ is the step size.

We call the evolution of $\Phi(T, T-t) \triangleq \frac{\partial y_N}{\partial y_{T-t}}$ 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 $\Phi(T, T-t)$ for the Hamiltonian model \([10]\). We start from the simple case where the parameters of \([10]\) are constant, i.e. $J(y(t), t) = J$, $K(t) = K$, $b(t) = b$, and call the corresponding networks time-invariant H-DNNs. The following Lemma, whose proof can be found in Appendix A.1 provides the desired model.

Lemma 1 Given the time-invariant ODE $\dot{y}(t) = f(y(t))$, the time evolution of $\Phi(T, T-t)$ is given by

$$
\frac{d}{dt} \Phi(T, T-t) = \frac{\partial f}{\partial y(T-t)} \Phi(T, T-t), \quad \Phi(T, T) = I. \tag{15}
$$

\(^3\)For a two class classification problem, it is given by $R_N(\cdot) = \| W \|^2 + \mu^2$. For multiclass problems, we refer the reader to [6].

\(^4\)\(\alpha_c\) is usually called weight decay.
Since in our case $f(y(t)) = JK^T \tanh(Ky(t) + b)$, we have

$$\frac{\partial f}{\partial y} = \frac{\partial}{\partial y} (K^T \tanh(Ky + b)) J^T = \frac{\partial}{\partial y} (\tanh(Ky + b)) KJ^T$$

$$= K^T \text{diag}(\tanh'(Ky + b)) KJ^T$$

$$= K^T D(y) KJ^T$$

(16)

where $D(y) = \text{diag}(\tanh'(Ky + b))$ and $\tanh'(\cdot)$ computes element-wise the derivative of $\tanh(\cdot)$.

The next two lemmas, proved in Appendix A.2 and A.3, show that the Jacobian matrix (16) satisfies the conditions for marginal stability.

**Lemma 2** The eigenvalues of $\frac{\partial f}{\partial y}$ are purely imaginary.

**Lemma 3** The algebraic and geometric multiplicity of each eigenvalue of $\frac{\partial f}{\partial y}$ do coincide.

As shown in the corresponding proofs, Lemma 2 hinges on results available in [8]. However, the multiplicity of the eigenvalues of $\frac{\partial f}{\partial y}$ (Lemma 2) has not been analysed in previous publications.

If $D(y(t))$ is constant, Lemmas 2 and 3 imply that $\phi(T, T-t)$, neither diverges nor converges to zero, irrespectively of the weights $J$, $K$ and $b$ and the final time $T$. This property, however, can be compromised by the time-varying nature of $D(y(t))$ and the weights $(K(t) + b(t))$ as well as the time discretization process underlying (14). Nevertheless, Lemmas 2 and 3 suggest that if $D(y(t))$, $(K(t) + b(t))$ change slowly enough and $h$ is sufficiently small, the growth or decrease of the terms $\|\frac{\partial y}{\partial y_{j+1}}\|_2$ can be kept under control.

While we do not provide a complete theoretical analysis when $D(y(t))$, $(K(t) + b(t))$ are time varying, we simply highlight that changes in the parameters $K_j$ and $b_j$ across the layers of $H_2$-DNN, $i = 1, 2$, can be controlled by suitably choosing the regularization parameter $\alpha$ 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 $K_j$ and $b_j$ are not constant.

## 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 1a and 1b) concerning binary classification with features in $\mathbb{R}^2$.

As in [6], we consider MS- and H-DNNs with augmented input features [17] so as to increase the modelling power. More specifically, input feature vectors are given by $[y^k_o \circ 0]^T \in \mathbb{R}^4$ where $y^k_o \in \mathbb{R}^2$, $k = 1, \ldots, s$, are the input datapoints (see Figures 1a and 1b). We complement the DNNs with an output layer $y_{N+1} = f_N(y_N, \theta_N)$ (see Section 3.3). The optimization problem is solved using the Adam algorithm with minibatches (see Appendix A.1 for details), and standard cross-entropy $L$ 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

| # parameters per layer | $\#$ | $\#$ | $\#$ | $\#$ | $\#$ |
|------------------------|------|------|------|------|------|
| $\#	ext{ parameters per layer}$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ |
| $\#	ext{ parameters per layer}$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ |
| $\#	ext{ parameters per layer}$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ |
| $\#	ext{ parameters per layer}$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ |
| $\#	ext{ parameters per layer}$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ | $\alpha$ |

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.

MS1-DNN | MS2-DNN | MS3-DNN | H2-DNN | H2-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% |
| 4 layers | 99.5% | 100% | 100% | 100% | 100% |
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                | 93.73%     | 92.47% | 93.82%     | 92.41% |
| 2                | 99.57%     | 97.72% | 99.81%     | 97.83% |
| 4                | 99.97%     | 98.03% | 99.78%     | 98.02% |
| 8                | 99.98%     | 98.05% | 99.70%     | 98.22% |
| 16               | 100%       | 98.14% | 99.50%     | 98.21% |

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$_1$-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\footnote{http://yann.lecun.com/exdb/mnist/}.

The dataset consists of $28 \times 28$ 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\footnote{Similar results can be obtained using other MS or H-DNNs.}, 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\footnote{Similar results can be obtained using other MS or H-DNNs.} 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 Appendix B.2.

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.
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 $\frac{\partial y_N}{\partial y_{j+1}}$ 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 $K(t) = K$ and $b(t) = b$ and a fully-connected neural network (FCNN) with 32 layers.

For the H-DNNs case, it can be seen that the norms of the terms $\frac{\partial y_N}{\partial y_{j+1}}$ for $j = 0, 10, \cdots, 60$ 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 $\frac{\partial l}{\partial \theta_{ij}}$ do converge to zero in approximately 500 and 300 iterations respectively, showing the optimization algorithm has achieved a (possible local) minimum. When using FCNN, however, it can be shown that gradient norms quickly tend 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.

Acknowledgements

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

---

7 See Appendix B.3 for implementation details.
References

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[2] W. Xiong, J. Droppo, X. Huang, F. Seide, M. L. Seltzer, A. Stolcke, D. Yu, and G. Zweig, “The microsoft 2016 conversational speech recognition system,” in 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Mar 2017, pp. 5255–5259.

[3] S. Lucia and B. Karg, “A deep learning-based approach to robust nonlinear model predictive control,” IFAC-PapersOnLine, vol. 51, no. 20, pp. 511–516, 2018.

[4] R. Zoppoli, M. Sanguineti, G. Gnecco, and T. Parisini, Neural Approximations for optimal control and decision. Springer, 2020.

[5] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, 1994.

[6] E. Haber and L. Ruthotto, “Stable architectures for deep neural networks,” Inverse Problems, vol. 34, no. 1, p. 014004, Dec 2017.

[7] E. Haber, L. Ruthotto, E. Holtham, and S.-H. Jun, “Learning across scales—multiscale methods for convolution neural networks,” in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[8] B. Chang, M. Chen, E. Haber, and E. H. Chi, “AntisymmetricRNN: A dynamical system view on recurrent neural networks,” in International Conference on Learning Representations, 2019.

[9] Y. Lu, A. Zhong, Q. Li, and B. Dong, “Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations,” in 6th International Conference on Learning Representations, ICLR 2018, Jan 2018.

[10] W. E, “A proposal on machine learning via dynamical systems,” Communications in Mathematics and Statistics, vol. 5, pp. 1–11, Feb 2017.

[11] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham, “Reversible architectures for arbitrarily deep residual neural networks,” in AAAI Conference on Artificial Intelligence, 2018.

[12] A. van der Schaft, L2-Gain and Passivity Techniques in Nonlinear Control. Springer, 2017.

[13] I. J. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[14] H. K. Khalil, Nonlinear Systems, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 2002.

[15] U. M. Ascher, Numerical Methods for Evolutionary Differential Equations. USA: Society for Industrial and Applied Mathematics, 2008.

[16] Y. Guo and D. Cheng, “Stabilization of time-varying Hamiltonian systems,” IEEE Transactions on Control Systems Technology, vol. 14, no. 5, pp. 871–880, 2006.

[17] E. Dupont, A. Doucet, and Y. W. Teh, “Augmented neural ODEs,” in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 3140–3150.
A Proofs

A.1 Proof of Lemma [1]

Given the ODE
\[ \dot{y} = f(y), \quad y(0) = y_0, \]  \hspace{1cm} (17)
we want to calculate the dynamics of \( \frac{\partial y(T)}{\partial y(T-t)} \).

Proof: The solution to (17) is
\[ y(t) = y(0) + \int_0^t f(y(\tau))d\tau \]  \hspace{1cm} (18)
Evaluating (17) in \( t = T \) and \( t = T-t \) and subtracting them, we obtain
\[ y(T) = y(T-t) + \int_{T-t}^T f(y(\tau))d\tau = y(T-t) + \int_0^t f(y(s+T-t))ds \]
where to obtain the second equality, we change the integration variable by defining \( \tau = s + T - t \). Therefore, we have
\[ \frac{\partial y(T)}{\partial y(T-t)} = I + \int_0^t \frac{\partial y(s+T-t)}{\partial y(T-t)} \frac{\partial f}{\partial y(y(s+T-t))} ds \]
As a result we have that
\[ \frac{\partial y(T)}{\partial y(T-t)} = \frac{\partial y(T-t)}{\partial y(T-t-\delta)} \frac{\partial y(T)}{\partial y(T-t)} \]
\[ = \left( I + \int_0^\delta \frac{\partial y(s+T-t-\delta)}{\partial y(T-t-\delta)} \frac{\partial f}{\partial y(y(s+T-t-\delta))} ds \right) \frac{\partial y(T)}{\partial y(T-t)} \]
Therefore, it follows that
\[ \frac{\partial y(T)}{\partial y(T-t-\delta)} \frac{\partial y(T)}{\partial y(T-t)} - \frac{\partial y(T)}{\partial y(T-t)} = \left( \int_0^\delta \frac{\partial y(s+T-t)}{\partial y(T-t-\delta)} \frac{\partial f}{\partial y(y(s+T-t-\delta))} ds \right) \frac{\partial y(T)}{\partial y(T-t)} \]
Dividing both sides by \( \delta \) and taking the limit for \( \delta \to 0 \), we obtain
\[ \frac{d}{dt} \frac{\partial y(T)}{\partial y(T-t)} = \frac{\partial f}{\partial y(y(T-t))} \frac{\partial y(T)}{\partial y(T-t)} . \]  \hspace{1cm} (19)
\[ \square \]

A.2 Proof of Lemma [2]

Proof: Since \( J^\top \) is a skew-symmetric matrix, \( KJ^\top K^\top \) is also skew-symmetric. By following the proof in the Appendix of [1], we have that the eigenvalues of \( KJ^\top K^\top D \) are all imaginary. Since \( \text{eig}(K^\top DKJ^\top) = \text{eig}(KJ^\top K^\top D) \), we obtain that the eigenvalues of \( \frac{\partial f}{\partial y(y(T-t))} \) are all imaginary. \( \square \)

A.3 Proof of Lemma [3]

We introduce the following Lemma which is needed to prove Lemma [4]. In the following, \( Q^\top \) and \( Q^* \) denote the transpose and the conjugate transpose of \( Q \), respectively.

Lemma 4 Given a real skew-symmetric matrix \( Q = -Q^\top \in \mathbb{R}^{n \times n} \) and a positive definite matrix \( P \in \mathbb{R}^{n \times n} \), we have \( PQ \) is diagonalizable.
Proof: Assume, by contradiction, that \( PQ \) is not diagonalizable. Then it has an eigenvalue \( \lambda \) with a corresponding eigenvector \( x \neq 0 \) and a corresponding generalized eigenvector \( z \neq 0 \) such that
\[
(PQ - \lambda I)x = 0, \quad (PQ - \lambda I)z = x
\]
Therefore, since \( P \) is invertible and we have
\[
(Q - \lambda P^{-1})x = 0, \quad (Q - \lambda P^{-1})z = P^{-1}x
\]
By left multiplying the above equations with \( z^* \) and \( x^* \) respectively, we have
\[
x^*(Q - \lambda P^{-1})x = 0 \quad \text{ (20)}
\]
\[
x^*(Q - \lambda P^{-1})z = x^*P^{-1}x \quad \text{ (21)}
\]
Taking the conjugate transpose of LHS of (20), we have
\[
x^*(Q^* - \lambda^* P^{-1})z = -x^*Qz - \lambda^*x^*P^{-1}z = 0
\]
Therefore, we have \( x^*Qz = -\lambda^*x^*P^{-1}z \) and substituting into (21), one obtains
\[
-(\lambda^* + \lambda)x^*P^{-1}z = x^*P^{-1}x \quad \text{ (22)}
\]
Since \( \lambda \) is pure imaginary, we know that the LHS of (22) is equal to zero. However, since \( x \neq 0 \), the RHS of (22) is not zero. Therefore, we have a contradiction concluding the proof. □

Proof: (Lemma 3) Since \( J^T \) is skew-symmetric and \( K^T D(y)K \) is positive definite, in view of Lemma 4 we know that \( \frac{\partial f}{\partial y} \) is diagonalizable for all \( y \). Therefore, the algebraic and geometric multiplicity of repeated eigenvalues of \( \frac{\partial f}{\partial y} \) coincides. □

B Implementation details

DNN architectures and training algorithms are implemented using PyTorch library.

B.1 Binary classification datasets

For two-class classification problems we used 5000 datapoints and a mini-batch size of 125, for both training and test data. For the optimization algorithm, we use coordinate gradient descent, i.e. a modified version of stochastic gradient descent (SGD) with Adam \((\beta_1 = 0.9, \beta_2 = 0.999)\) \[6\]. The cross-entropy loss has been used to compare the predicted outputs and the true labels. In every iteration of the algorithm, first the optimal weights of the output layer are computed given the last updated parameters of the hidden layers, and then, a step update of the hidden layers’ parameters is performed by keeping fixed the output parameters. The training consist in 50 epochs and each of them has maximum 10 iterations to compute the output layer weights. The learning rate was set to 0.05 and the weight decay for the output layer is constant and set to \(2 \times 10^{-4}\). The \(\alpha\) coefficient of the regularization is set to \(5 \times 10^{-3}\).

B.2 MNIST dataset

We use the complete MNIST dataset (60,000 training examples and 10,000 test examples) and a mini-batch size of 100. For the optimization algorithm we use SGD with Adam and cross-entropy loss. The learning rate is initialized to be 0.04 and decayed with \(\gamma = 0.8\) each epoch. The total training step is 40 epochs. We use \(\alpha = 1 \times 10^{-3}\) for the regularization term in all H-DNNs. The weight decay is constant and set to \(2 \times 10^{-4}\).

B.3 Fully-connected neural network

The layer equation of a fully connected network is \( y_{k+1} = \sigma(K_k y_k + b_k) \), with activation function \( \sigma(\cdot) = \tanh(\cdot) \) and trainable parameters \( K_k \in \mathbb{R}^{n \times n} \) and \( b_k \in \mathbb{R}^{n} \) for \( k = 0, 1, \ldots, N - 1 \) where \( N \) is the number of layers of the network. We use the same implementation as described in Appendix B.1 with a weight decay of \(2 \times 10^{-4}\).