Port-Hamiltonian Approach to Neural Network Training

Stefano Massaroli
The University of Tokyo
32 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Michael Poli
Stanford University
27 PUBLICATIONS **36** CITATIONS
SEE PROFILE

Federico Califano
University of Twente
51 PUBLICATIONS **359** CITATIONS
SEE PROFILE

Angela Faragasso
The University of Tokyo
37 PUBLICATIONS **620** CITATIONS
SEE PROFILE

Port–Hamiltonian Approach to Neural Network Training

Stefano Massaroli[1,†,⋆] and Michael Poli[2,⋆],
Federico Califano[3], Angela Faragasso[1,†], Jinkyoo Park[2], Atsushi Yamashita[1,†] and Hajime Asama[1,‡]

Abstract— **Neural networks are discrete entities: subdivided into *discrete* layers and parametrized by weights which are iteratively optimized via *difference* equations. Recent work proposes networks with layer outputs which are no longer quantized but are solutions of an ordinary differential equation (ODE); however, these networks are still optimized via discrete methods (e.g. gradient descent). In this paper, we explore a different direction: namely, we propose a novel framework for learning in which the parameters themselves are solutions of ODEs. By viewing the optimization process as the evolution of a port-Hamiltonian system, we can ensure convergence to a minimum of the objective function. Numerical experiments have been performed to show the validity and effectiveness of the proposed methods.**

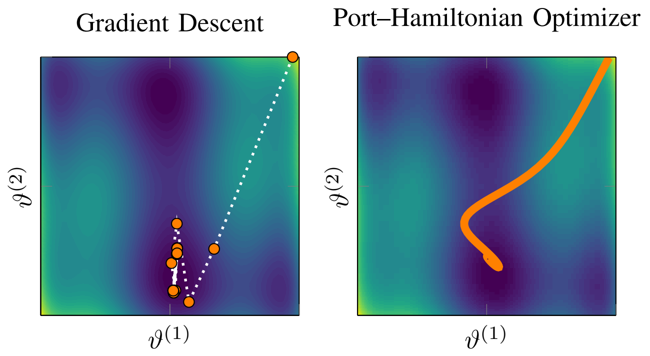

Fig. 1. Comparison between the *discrete* optimizer gradient descent and our *continuous* port-Hamiltonian approach.

I. Introduction

Neural networks are universal function approximators [1]. Given enough *capacity*, which can arbitrarily be increased by adding more parameters to the model, they can approximate any Borel–measurable function mapping finite–dimensional spaces. Each layer of a neural network performs an affine transformation to its input and generates an output which is then fed into the next layer. Backpropagation [2] is at the core of modern deep learning, and most state-of-the-art architectures for tasks such as image segmentation [3], generative tasks [4], image classification [5] and machine translation [6] rely on the effective combination of universal approximators and line search optimization methods: most notably *stochastic gradient descent* (SGD), Adam [7] RMSProp [8] and recently RAdam [9].

Training neural networks is a non–convex optimization problem which aims to obtain globally or locally optimal values for its parameters by minimizing an objective function that is usually designed ad–hoc for the application at hand. The landscape of such objective functions is often highly non–convex and finding global optima is in general an NP–complete problem [10], [11]. Optimality guarantees for algorithms such as gradient descent do not hold in this setting; moreover, the discrete nature of neural networks adds complications to the development of a proper theoretical understanding with sufficient convergence conditions. Despite the empirical successes of deep learning, these reasons alone lead many to question whether or not relying on these standard methods could be a limitation to the advancement of deep learning research. In this work, we offer a new perspective on the optimization of neural networks, where parameters are no longer iteratively updated via difference equations, but are instead solutions of ODEs. This is achieved by equipping the parameters with autonomous port-Hamiltonian dynamics. Port-Hamiltonian (PH) systems [12], [13], [14] have been introduced to model dynamical systems coming from different physical domains in a unified manner. This framework turned out to be fruitful in dealing with passivity based control (PBC) [15], [16], [17] since dissipativity information is explicitly encoded in PH systems, i.e. under mild assumptions those systems are passive. The aim of this work is to take advantage of such a structure and build a proper PH system associated to a neural network, in which the parameters of the latter are the states of the PH system. Within this framework, the weights evolve in time on a continuous trajectory along strictly decreasing level sets of the energy function, i.e. the objective function of the optimization problem, eventually landing in one of its minima. In this way, local optimality is intrinsically guaranteed by the PH dynamics.

This paper is structured as follows: Section II discusses previous works on continuous–time and energy–based ap-

[1]Stefano Massaroli, Angela Faragasso, Atsushi Yamashita and Hajime Asama are with the Department of Precision Engineering, The University of Tokyo, 7-1-3 Hongo, Bunkyo, Tokyo, Japan
{massaroli,faragasso,yamashita,asama}@robot.t.u-tokyo.ac.jp

[2]Michael Poli and Jinkyoo Park are with the Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Eoeun-dong, Yuseong-gu, Daejeon, South Korea
{poli_m,jinkyoo.park}@kaist.ac.kr

[3]Federico Califano is with the Faculty of Electrical Engineering, Mathematics & Computer Science (EWI), Robotics and Mechatronics (RAM), University of Twente, Hallenweg 23 7522NH, Enschede, The Netherlands
f.califano@utwente.nl

†*IEEE Member*, ‡*IEEE Fellow*

⋆These authors contributed equally to the work.

proaches for neural networks. In Section III, a formal introduction to neural networks to resolve some notational conflicts between control and learning theory. Section IV introduces port–Hamiltonian systems and their application to the training of neural networks. Next, in Section V, the performance of the proposed method is evaluated on a series of tasks and the results are discussed. Finally, in Section VI conclusions are drawn and future work is discussed.

II. Related Work

Recent works [18] have shown that it is possible to model residual layers as continuous blocks. This allows for a smooth transition between input and output: the input is integrated for a fixed time, which can be seen as the continuous analog of the number of network layers in the discrete case. By using the adjoint integration method Neural Ordinary Differential Equation Networks (ODE-Nets) offer improved memory efficiency and their performance is comparable to regular neural networks. ODE-Nets, however, are still optimized via discrete gradient descent methods. A similar idea was previously proposed in [19], which introduces Hamiltonian dynamics as a means of modeling network activations. [20] introduces the Hamiltonian function as a useful physics-driven prior for learning conservative dynamics. [21] explores a connection between non-convex optimization and viscous Hamilton–Jacobi partial differential equations by introducing a modified version of stochastic gradient descent, Entropy–SGD. Entropy–SGD is applied to a function that is more convex in its input than the original loss function and yields faster convergence times. Similarly, a connection between Hamiltonian dynamics and learning was proposed in [22] and [23]. Energy-based models have been explored in the past [24] [25]. Hopfield neural networks are designed to learn binary patters by iteratively reducing their *energy* until convergence to an attractor. The energy is defined as a Lyapunov function of the weights of the network such that convergence to a local mininum is guaranteed. The binary-valued units i of a Hopfield network are updated via a discrete procedure which checks if the weighted sum of the neighbouring units values does not reach a threshold, in which case the value of i is flipped.

III. Problem Setting

A. Notation

The set $\mathbb{R}$ ($\mathbb{R}^+$) is the the set of real (non negative real) numbers. The set of squared–integrable functions $z : \mathbb{R} \to \mathbb{R}^m$ is $\mathcal{L}_2^m$ while the set of d–times continuously differentiable functions is $\mathcal{C}^d$. Let $\langle \cdot, \cdot \rangle : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ denote the inner product on $\mathbb{R}^m$ and $\|v\|_2 \triangleq \sqrt{\langle v, v\rangle}$ its induced norm. The origin of $\mathbb{R}^n$ is $\mathbb{0}_n$. Let $\mathcal{H} : \mathbb{R}^n \to \mathbb{R}$ be $\mathcal{C}^1$ and let $\partial\mathcal{H} \in \mathbb{R}^n$ be its transposed gradient, i.e. $\partial\mathcal{H} \triangleq (\nabla\mathcal{H})^\top \in \mathbb{R}^n$. In ambiguous cases, the variable with respect to which $\mathcal{H}$ is differentiated may appear as subscript, e.g. $\partial_x \mathcal{H}$. Indexes of vectors are indicated in superscripts, e.g. if v is a vector $v^{(i)}$ indicates the i-th entry. Given two vectors u, $v \in \mathbb{R}^n$, let $(u, v) \triangleq [u^\top, v^\top]^\top$.

B. Introduction to neural networks

In order to provide a definition of *neural networks* suitable for the scope of this paper, we must clarify the class of mathematical objects that they can handle. In particular, only networks whose input and outputs are vectors are treated. Indeed, the concepts presented hereafter can be naturally extended to more complex networks[1].

Definition 3.1 (Neural Network): A neural network is a map

$$f : \mathcal{U} \times \mathcal{K} \to \mathcal{Y},$$

being $\mathcal{U} \subset \mathbb{R}^{n_u}$ the *input space*, $\mathcal{Y} \subset \mathbb{R}^{n_y}$ the *output space* and $\mathcal{K} \subseteq \mathbb{R}^p$ the manifold where the parameters characterizing the neural network live. The parameters, collected in a vector ϑ, are assumed time dependent. Hence,

$$y = f(u, \vartheta(t)) \quad u \in \mathcal{U},\ y \in \mathcal{Y},\ \vartheta \in \mathcal{K}. \tag{1}$$

If samples $\hat{u}_i, \hat{y}_i$ ($i = 1, \dots, s$) of the input and output spaces are provided, the parameters ϑ may be tuned in order to minimize an arbitrary cost function, e.g. the squared–norm of the output reconstruction error

$$\|e_i\|_2^2 \triangleq \|\hat{y}_i - f(\hat{u}_i, \vartheta)\|_2^2 \quad \forall i = 1, \dots, s. \tag{2}$$

Example 3.2 (Fully Connected Network): In a *fully–connected* neural network, the j–th element $y_i^{(j)}$ of the output of the i–th layer is

$$y_i^{(j)} = \sigma\left(\sum_{k=1}^{h_{i-1}} y_{i-1}^{(j)} w_{i,j,k} + b_{i,j}\right) \quad \forall j = 1, \dots, h_i\ , \tag{3}$$

where h_i is the number of neurons in the i–th layer, $w_{i,j,k}, b_{i,j} \in \mathbb{R}$, $\sigma_i : \mathbb{R} \to \mathbb{R}$ is called *activation function*[2] and $y_0 \triangleq u$. Indeed, y_i can be symbolically rewritten in vector form as

$$y_i = \sigma_i\left(W_i y_{i-1} + b_i\right),$$

where

$$W_i = \begin{bmatrix} w_{i,1,1} & w_{i,1,2} & \cdots & w_{i,1,h_{i-1}} \\ w_{i,2,1} & w_{i,2,2} & \cdots & w_{i,2,h_{i-1}} \\ \vdots & \vdots & \ddots & \vdots \\ w_{i,h_i,1} & w_{i,h_i,2} & \cdots & w_{i,h_i,h_{i-1}} \end{bmatrix}, b_i = \begin{bmatrix} b_{i,1} \\ b_{i,2} \\ \vdots \\ b_{i,h_i} \end{bmatrix}$$

and σ_i is thought to be acting component–wise. Thus, for the i–th layer a vector ϑ_i containing all the weights and biases can be defined as

$$\vartheta_i = [w_{i,1,1}, \dots, w_{i,h_i,h_{i-1}}, b_{i,1}, \dots, b_{i,h_i}]^\top \in \mathbb{R}^{h_i(1+h_{i-1})}. \tag{4}$$

Therefore, the overall vector containing all the parameters of a fully connected neural network with l layers is

$$\vartheta = (\vartheta_1, \vartheta_2, \dots, \vartheta_l) \in \mathbb{R}^p, \tag{5}$$

[1]Aforementioned networks often deal with multi–dimensional arrays (*holors* [26]) which are referred as *tensors* by the artificial intelligence community.

[2]Usually σ_i is a nonlinear function.

where

$$p = \sum_{i=1}^{l} h_i(1 + h_{i-1}).$$

C. *Training of a Neural Network*

Let $\mathcal{U}_s$, $\mathcal{Y}_s$ be finite and ordered subsets of the input and output spaces, i.e.

$$\mathcal{U}_s = \{\hat{u}_1, \hat{u}_2, \dots, \hat{u}_i, \dots, \hat{u}_s\} \subset \mathcal{U},$$
$$\mathcal{Y}_s = \{\hat{y}_1, \hat{y}_2, \dots, \hat{y}_i, \dots, \hat{y}_s\} \subset \mathcal{Y},$$

such that

$$\exists \Psi : \mathcal{U} \to \mathcal{Y} \ : \ \hat{y}_i = \Psi(\hat{u}_i) \ \forall i = 1, \dots, s \ .$$

The aim of the *training* process of a neural network is to find a value of the parameters $\vartheta \in \mathcal{K}$ such that the elements of $\mathcal{U}_s$ are mapped by f defined in (1) minimizing a given objective function dependent on the output samples, e.g. (2).

Let $\mathcal{J} : \mathcal{U} \times \mathcal{Y} \times \mathcal{K} \to \mathbb{R}$ be the objective (*loss*) function; then, the solution of the training problem is

$$\vartheta^* = \arg\min_{\vartheta} \mathcal{J}(\hat{u}_i, \hat{y}_i, \vartheta) \quad \forall \hat{u}_i \in \mathcal{U}_s, \hat{y}_i \in \mathcal{Y}_s \, . \qquad (6)$$

Consider a function $\Gamma : \mathcal{U} \times \mathcal{Y} \times \mathcal{K} \to \mathcal{K}$. Traditionally, a locally optimal solution of (6) can be obtained by iterating a difference equation of the form:

$$\vartheta_{t+1} = \vartheta_t + \Gamma(\hat{u}_{t+1}, \hat{y}_{t+1}, \vartheta_t) \quad t = 1, \dots, s \ , \qquad (7)$$

for a *sufficient* [3] number of steps, where the specific choice of Γ determines the difference between training algorithms.

In contrast to such state–of–the–art methods, our approach is to equip the weights with continuous–time dynamics. In particular, we model the behavior of the parameters with port-Hamiltonian systems. Due to the unique structure of this class of dynamical systems, asymptotic convergence toward an optimal solution will be automatically guaranteed.

IV. Training of Neural Networks: the Port–Hamiltonian Approach

A. *Introduction to port–Hamiltonian systems*

A port–Hamiltonian (PH) system has an input–state–output representation

$$\begin{cases} \dot{\xi} = [J(\xi) - R(\xi)]\partial\mathcal{H}(\xi) + g(\xi)v \\ z = g^\top(\xi)\partial\mathcal{H}(\xi) \end{cases} , \qquad (8)$$

with state $\xi \in \mathcal{X} \subset \mathbb{R}^n$, input $v \in \mathcal{V} \subset \mathbb{R}^m$ and output $z \in \mathcal{Z} \subset \mathbb{R}^m$. The function $\mathcal{H} : \mathcal{X} \to \mathbb{R}$ is called *Hamiltonian function* and has the role of a generalized energy while $J(\xi) = -J^\top(\xi) \in \mathbb{R}^{n\times n}$ represents power preserving–interconnections, $R(\xi) = R(\xi) \geq 0$ models dissipative effects and $g(\xi) \in \mathbb{R}^{n\times m}$ describes the way in which external power is distributed into the system. In general, $\mathcal{X}$ is an n–dimensional manifold, $\mathcal{V}$ is a m–dimensional vector space and $\mathcal{Z} = \mathcal{V}^*$ is its dual space. Consequently the natural pairing $\langle v, z\rangle \triangleq z^\top v$ can be defined, which carries the unit measure of power (when modeling physical systems). For compactness, from now on let us define $F(\xi) \triangleq J(\xi) - R(\xi)$ and omit the dependence on ξ of $\mathcal{H}$ and F.

Assumption 4.1: Assumptions for PH systems

1. $F, g, \mathcal{H}$ are assumed smooth enough such that solutions are forward–complete for all initial conditions $\xi_0 \in \mathcal{X}$, $v \in \mathcal{L}_2^m$;
2. $\mathcal{H}$ is lower–bounded in $\mathcal{X}$, i.e.

$$\exists \zeta \in \mathbb{R} : \ \forall x \in \mathcal{X} \ \ \mathcal{H}(\xi) > \zeta.$$

From these assumptions it follows that PH systems are passive (see [15], [14]), i.e.

$$\dot{\mathcal{H}} \leq z^\top v.$$

As a consequence, in the autonomous case ($v = 0$) the Hamiltonian function is always non–increasing along trajectories. In particular,

$$\dot{\mathcal{H}} = \langle \partial\mathcal{H}, \dot{\xi}\rangle = -(\partial\mathcal{H})^\top R \partial\mathcal{H} \leq 0 \ \ \forall t \geq 0.$$

Thus, any strict minimum of $\mathcal{H}$ is a Lyapunov stable equilibrium point of the system. Furthermore, the control law $v = -kz$ ($k > 0$), usually referred as *damping injection*, asymptotically stabilizes the equilibria [15].

Therefore, depending on the initial condition and the basins of attraction of the minima of $\mathcal{H}$, the state will eventually land in one minimum point of the Hamiltonian function. This latter property is the key that allows the use of PH systems for the training of neural networks.

B. *Equip the network with port–Hamiltonian dynamics*

The proposed approach consists in describing the dynamics of the neural network's parameters using an autonomous PH system. In fact, if the Hamiltonian function coincides with the loss function of the learning problem, i.e. $\mathcal{H} \triangleq \mathcal{J}$, we guarantee asymptotic convergence to a minimum of $\mathcal{J}$, i.e. solution of the problem (6) and hence successful training of the neural network.

Generally, a desirable property of the parameter dynamics is to reach a minimum of $\mathcal{J}$ with null velocity $\dot{\vartheta}$. In the port–Hamiltonian framework, this can be achieved with mechanical–like equations. Let $\omega \triangleq M(\vartheta)\dot{\vartheta}$ be a vector of fictitious generalized momenta where $M = M^\top > 0$ is the generalized inertia matrix. The role of M is to give different *weight* to the parameters and model specific couplings between their dynamics. Then, let the state of the PH system be

$$\xi \triangleq (\vartheta, \omega) \in \mathbb{R}^{2p} \, .$$

Hence, the loss function might be redefined adding a term $\mathcal{J}_{\mathtt{kin}}(\vartheta, \omega)$ equivalent to a pseudo kinetic energy:

$$\mathcal{J}^*(\hat{u}, \hat{y}, \xi) \triangleq \mathcal{J}(\hat{u}, \hat{y}, \vartheta) + \underbrace{\omega^\top M^{-1}(\vartheta)\omega}_{\mathcal{J}_{\mathtt{kin}}(\vartheta,\omega)} .$$

Note that $\mathcal{J}(\hat{u}, \hat{y}, \vartheta)$ represents the potential energy of the fictitious mechanical system.

As for general p degrees–of–freedom mechanical system in PH form, the choice of J and R is:

$$J \triangleq \begin{bmatrix} O_p & I_p \\ -I_p & O_p \end{bmatrix} \in \mathbb{R}^{2p\times 2p}, \ R \triangleq \begin{bmatrix} O_p & O_p \\ O_p & B \end{bmatrix} \in \mathbb{R}^{2p\times 2p},$$

[3]Here sufficient is intended in a statistical learning theory sense

where O_p, I_p are respectively the p–dimensional zero and identity matrices while $B = B^\top > 0$, $B \in \mathbb{R}^{p\times p}$. Therefore, the autonomous PH model of the parameters dynamics obtained by setting $\mathcal{H} = \mathcal{J}^*$ is:

$$\begin{bmatrix}\dot\vartheta\\ \dot\omega\end{bmatrix} = (J - R)\begin{bmatrix}\partial_\vartheta \mathcal{J}^*\\ \partial_\omega \mathcal{J}^*\end{bmatrix}$$
$$\Leftrightarrow \quad \dot\xi = \underbrace{\begin{bmatrix}0 & I_n\\ -I_n & -B\end{bmatrix}}_{F} \partial\mathcal{J}^*. \tag{9}$$

Hence, trajectories of ϑ, ω will unfold on continuously decreasing level sets of $\mathcal{J}^*$ which plays the role of a *generalized* energy.

Example 4.2: Suppose

$$M(\vartheta) = I_p \Rightarrow \omega = \dot\vartheta$$

and let $\mathcal{J}$ be the mean–squared–error loss. Therefore, a possible choice of the loss function $\mathcal{J}^*$ is

$$\mathcal{J}^*(\hat u, \hat y, \xi) = \frac{1}{2}\left[\alpha\|\hat y - f(\hat u, \vartheta(t))\|_2^2 + \beta\vartheta^\top\vartheta + \dot\vartheta^\top\dot\vartheta\right], \tag{10}$$

with $\alpha, \beta \in \mathbb{R}^+$. Indeed, every minima of $\mathcal{J}^*$ is placed in $\dot\vartheta = \mathbb{0}_p$. The gradient of $\mathcal{J}^*$ is

$$\partial\mathcal{J}^* = \left(\alpha\frac{\partial f}{\partial\vartheta}[\hat y - f(\hat u, \vartheta)] + \beta\vartheta, \dot\vartheta\right).$$

With this choice of $\mathcal{J}^*$, the dynamics of the parameters become a (nonlinear) second–order ordinary differential equation:

$$\ddot\vartheta + \alpha\frac{\partial f}{\partial\vartheta}[\hat y - f(\hat u, \vartheta)] + \beta\vartheta + B\dot\vartheta = \mathbb{0}_p\ .$$

Remark 4.3: The term $\beta\vartheta^\top\vartheta$ in (10) is introduced as a *regularization* tool. Regularization is a fundamental technique in machine learning that is widely used in order to find solutions with smaller norm (e.g. *weight decay*) or to enforce sparsity in the parameters (e.g. *L1–regularization*). Note that this is a particular case of the Tikhonov regularization term $\|\Lambda\vartheta\|_2^2$ with $\Lambda = \beta I_p$ [27], also known as weight decay [28].

Remark 4.4: In the context of mechanical–like PH system, a consistent choice of the power port is

$$g \triangleq \begin{bmatrix}O_p & O_p\\ O_p & I_p\end{bmatrix},$$

selecting as input the fictitious generalized forces and as output the velocities $z = \dot\vartheta$. Hence, during the training of the neural network, a control input $v = -k(t)\dot\vartheta$ ($k(t) \geq 0\ \forall t$) might be applied to dynamically modify the rate with which the parameters of the network are optimized. This opens different scenarios for designing a $k(t)$ which increases the probability of reaching the global minimum of $\mathcal{J}^*$. In fact, the choice of k determines the shape of the basins of attraction of the minima of $\mathcal{J}^*$ [29]. This open problem is left for future work.

Definition 4.5 (Port–Hamiltonian neural network): We define a *port–Hamiltonian neural network* (PHNN) as a neural network whose parameters ϑ have continuous–time dynamics (9):

$$\begin{cases}\dot\xi = F\partial\mathcal{J}^*\\ y = f(u, \xi)\end{cases}. \tag{11}$$

Note that a PHNN is uniquely defined by the triplet $(f, F, \mathcal{J}^*)$.

Example 4.6 (Linear Classifier): Consider a fully connected network (see Example 3.2) with a single layer, h neurons[4] and l classes, i.e. $u \in \mathcal{U} \subset \mathbb{R}^h$, $y \in \mathcal{Y} \subset \mathbb{R}^l$. Therefore,

$$y = f(u, \vartheta) \triangleq \begin{bmatrix}w_{1,1} & w_{1,2} & \cdots & w_{1,h} & b_1\\ w_{2,1} & w_{2,2} & \cdots & w_{2,h} & b_2\\ \vdots & \vdots & \ddots & \vdots & \vdots\\ w_{l,1} & w_{l,2} & \cdots & w_{l,h} & b_l\end{bmatrix}\begin{bmatrix}u\\ 1\end{bmatrix}. \tag{12}$$

Let ϑ and the loss function $\mathcal{J}^*$ be defined as in (4) and (10) respectively. Hence,

$$\xi = (\vartheta, \dot\vartheta) \in \mathbb{R}^{2l(h+1)}\,. \tag{13}$$

Then,

$$\partial\mathcal{J}^* = \begin{bmatrix}\partial_\vartheta \mathcal{J}^*\\ \partial_{\dot\vartheta} \mathcal{J}^*\end{bmatrix} = \begin{bmatrix}\alpha\langle(\hat u, 1), \hat y - f(\hat u, \vartheta)\rangle + \beta\vartheta\\ \dot\vartheta\end{bmatrix},$$

$$\dot\xi = F\partial\mathcal{J}^* = \begin{bmatrix}\dot\vartheta\\ -\alpha\langle(\hat u, 1), \hat y - f(\hat u, \vartheta)\rangle - \beta\vartheta - B\dot\vartheta\end{bmatrix}.$$

C. Training of PHNNs

Let us assume that a dataset of inputs $\mathcal{U}_s$ and outputs (*labels*) $\mathcal{Y}_s$ is available. In this section two training techniques will be introduced.

1) Sequential data training: As already pointed out, given an initial condition ξ_0, the system will converge to a minimum ϑ^* of $\mathcal{J}$, i.e. to a minimum $(\vartheta^*, \mathbb{0}_p)$ of $\mathcal{J}^*$. However, the location of the minima strictly depends on the training data. The *sequential* training approach relies on iteratively feeding one tuple $\hat u_i, \hat y_i$ to the PHNN integrating the differential equation for a time t^* in each iteration. This process can be carried out from scratch several times (*epochs*).

Let τ be a *timer*, i.e. $\dot\tau = 1$ and ζ a *cycle counter*, both initialized to 0. After the initialization step, a first tuple $\hat u_1, \hat y_1$ is fed to the PHNN and integration starts from $\tau = 0$. When $\tau = t^*$ a new tuple is fetched, τ is reset, ζ is increased by 1 and the state ξ is carried over. The process is repeated until $\zeta = s$ and the first *epoch* is complete, at which point the first tuple will be fetched once again. This technique is reminiscent of the way in which SGD updates are performed in practice.

The PHNN with the *update and converge* training can be represented by means of an *hybrid dynamical system* (see [30]) whose graphical representation is shown in Fig. 2.

2) Batch training: In the *batch* method the neural network is trained using the entire dataset simultaneously. To do this,

[4]In this case, the number of neurons equals the dimension of the input space.

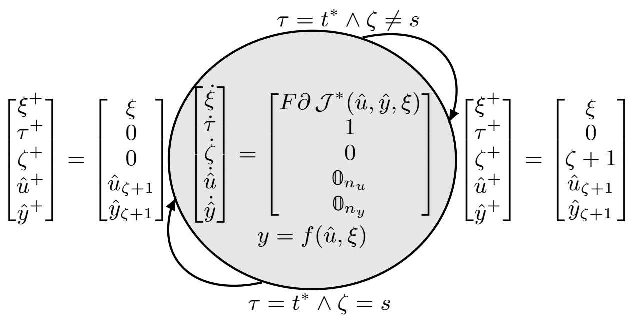
$\tau = t^* \wedge \zeta \neq s$
$\tau = t^* \wedge \zeta = s$
$\begin{bmatrix}\xi^+\\ \tau^+\\ \zeta^+\\ \hat{u}^+\\ \hat{y}^+\end{bmatrix} = \begin{bmatrix}\xi\\ 0\\ 0\\ \hat{u}_{\zeta+1}\\ \hat{y}_{\zeta+1}\end{bmatrix}$
$\begin{bmatrix}\dot{\xi}\\ \dot{\tau}\\ \dot{\zeta}\\ \dot{\hat{u}}\\ \dot{\hat{y}}\end{bmatrix} = \begin{bmatrix}F\partial\mathcal{J}^*(\hat{u},\hat{y},\xi)\\ 1\\ 0\\ \mathbb{0}_{n_u}\\ \mathbb{0}_{n_y}\end{bmatrix}$
$y = f(\hat{u},\xi)$
$\begin{bmatrix}\xi^+\\ \tau^+\\ \zeta^+\\ \hat{u}^+\\ \hat{y}^+\end{bmatrix} = \begin{bmatrix}\xi\\ 0\\ \zeta+1\\ \hat{u}_{\zeta+1}\\ \hat{y}_{\zeta+1}\end{bmatrix}$

Fig. 2. Hybrid automata: Conceptual representation of the hybrid system modeling the *sequential training* of the neural network.

the loss function is redefined as the average of the losses of each sample, i.e.

$$\mathcal{J}^*_{\texttt{batch}}(\mathcal{U}_s,\mathcal{Y}_s,\xi) \triangleq \frac{1}{s}\sum_{i=1}^{s}\mathcal{J}^*(\hat{u}_i,\hat{y}_i,\xi) \triangleq \sum_{i=1}^{s}\mathcal{J}^*_i .$$

Thanks to the linearity of differentiation, it is also possible to compute the gradient as the average of the gradients of the single losses:

$$\partial\mathcal{J}^*_{\texttt{batch}} = \frac{1}{s}\sum_{i=1}^{s}\partial\mathcal{J}^*_i .$$

Then, the training is simply achieved by integrating

$$\dot{\xi} = F\partial\mathcal{J}^*_{\texttt{batch}} .$$

Remark 4.7: Note that the sequential training will stop at one of the minima of $\mathcal{J}^*(\hat{u}_\zeta,\hat{y}_\zeta,\xi)$ (depending of the time at which the procedure is stopped), which might not necessarily coincide with a minimum of $\mathcal{J}^*_{\texttt{batch}}(\mathcal{U}_s,\mathcal{Y}_s,\xi)$.

D. Computational complexity

Theoretical space and time complexity of the proposed method are comparable to standard gradient descent and depend on the specific ODE solving algorithm employed. Let p be the number of parameters of the neural network to optimize. Regular gradient descent has space complexity linear in p (i.e $\mathcal{O}(p)$), whereas PHNNs require an additional state per parameter, the momentum, thus also yielding linear space complexity. Similarly, time complexity is linear in ϵ, the number of gradient descent steps necessary for convergence to a neighbourhood of a minimum.

V. Numerical Experiment

The effectiveness of PHNNs has been evaluated on the following two classes of numerical experiments. As an initial test, the PHNN has been tasked with learning a linear boundary between two classes of points by using a sequential training approach. The second experiment, on the other hand, deals with non-linear vector field approximation via the use of the batch training method. All the experiments have been implemented in *Python*[5]

[5]The code is available at: `https://github.com/Zymrael/PortHamiltonianNN`.

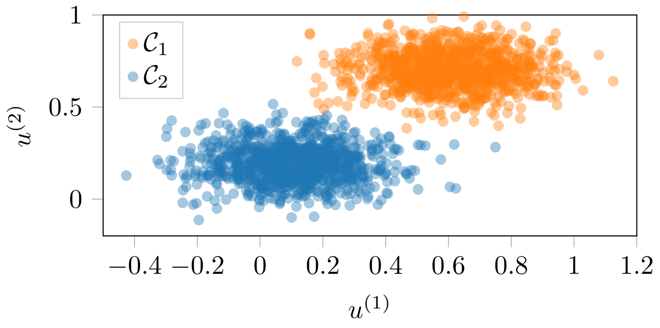

Fig. 3. Dataset used to train the neural network in the numerical experiment

A. Task 1: Learning a linear boundary

Consider the PHNN of Example 4.6 in the case $h = 2$, $l = 2$, i.e., $u \triangleq [u^{(1)}, u^{(2)}]^\top \in \mathbb{R}^2$, $y \triangleq [y^{(1)}, y^{(2)}]^\top \in \mathbb{R}^2$. The aim of the numerical experiment is to learn a linear boundary separating two classes of points sampled from two bivariate Gaussian distributions $\mathcal{N}_1, \mathcal{N}_2$. We will refer to the two classes as $\mathcal{C}_1$ and $\mathcal{C}_2$. The neural network model is

$$\begin{bmatrix}y^{(1)}\\ y^{(2)}\end{bmatrix} = \begin{bmatrix}w_{1,1}u^{(1)} + w_{1,2}u^{(2)} + b_1\\ w_{2,1}u^{(1)} + w_{2,2}u^{(2)} + b_2\end{bmatrix}$$

and the corresponding parameter vector is

$$\vartheta \triangleq [w_{1,1}, w_{1,2}, w_{2,1}, w_{2,2}, b_1, b_2]^\top \in \mathbb{R}^6 \Rightarrow \xi \in \mathbb{R}^{12} .$$

The dataset has been built sampling a total of 1000 points from each distribution and has been collected in $\mathcal{U}_s$ in a shuffled order. The result is shown in Fig. 3. The corresponding reference outputs have been computed and stored in $\mathcal{Y}_s$. In particular,

$$\hat{y} = \Psi(\hat{u}) \triangleq \begin{cases}[1,0]^\top & \hat{u}\in\mathcal{C}_1\\ [0,1]^\top & \hat{u}\in\mathcal{C}_2\end{cases} \quad \forall\hat{u}\in\mathcal{U}_s . \qquad (14)$$

Then, the training procedure has been performed on a single sample $(\hat{u},\hat{y}) = ([0.6, 0.6]^\top, [1, 0]^\top)$. The weights and their velocities have been initialized as $\xi_0 = [0.6, -2.3, -0.1, -1.1, -1.2, 0.3, -1.2, 0.3, 0.2, 1.6, -0.4, 1.6]^\top$. The system parameters have been chosen as, $B = I_6$, $\alpha = 1$ and $\beta = 0$. The resulting ODE has been numerically integrated for a time $t_f = 5s$. The resulting weight trajectories are reported in Fig. 4. Black is used to highlight parameters that are used to compute the first output element $y^{(1)}$ whereas blue is similarly used for parameters of $y^{(2)}$. Furthermore, the time evolution of the output of the neural network and the one of the loss function are shown in Fig. 5.

In order to show the effect of the regularization term $\beta\vartheta^\top\vartheta$ we performed the same experiment multiple times varying β in the interval $[0, 3]$. At each iteration, the relative output tracking squared error

$$e_r \triangleq \frac{\|\hat{y} - y(t_f)\|_2^2}{\|\hat{y}\|_2^2}$$

and the norm $\|\vartheta\|_2$ of the parameters vector, have been computed. This shows that the effect of β is comparable to the effect of weight decay in neural networks optimized

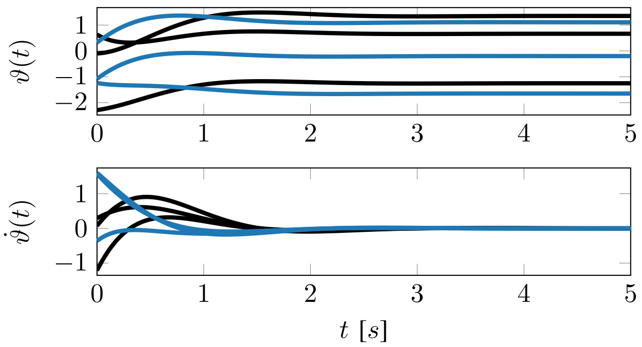

Fig. 4. Time evolution of the parameters ϑ and their velocities ω. Black indicates parameters of $y^{(1)}$ while blue parameters of $y^{(2)}$.

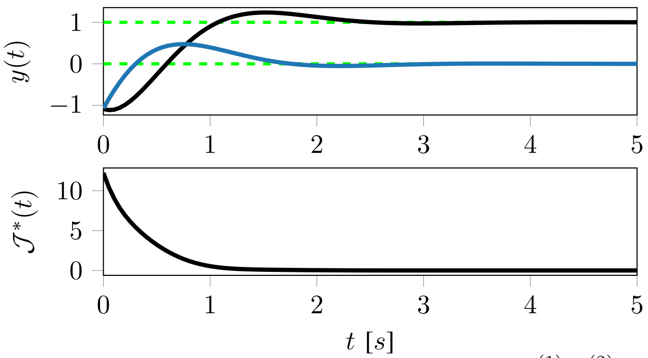

Fig. 5. [Above] Time evolution of the estimated output y. $y^{(1)}$, $y^{(2)}$ are indicated with black and blue lines respectively. [Below] Decay in time of the loss function $\mathcal{J}^*$.

via discrete methods, namely a reduction of the parameter norm. The results are shown in Fig. 6.

Subsequently, the dataset $\mathcal{U}_s$, $\mathcal{Y}_s$ has been split in a *training set* and a *test set* with a ratio of $3:1$. Then, the optimization of the network's parameters has been performed with the *sequential* method by using exclusively training set data while classification accuracy of the trained network has been evaluated on the test set. The chosen values of the parameters are the following: $B = 100I_6$, $\alpha = 1$, $\beta = 0.001$, $t^* = 0.1$ and ξ_0 has been initialized as before. The training procedure has been carried out for 100 epochs. The time evolution of the parameters and the loss function (one value per epoch) is shown in Fig. 7. As the loss is non-increasing with the number of epochs, the parameters converge to constant values. Furthermore, a *decision boundary* has been plotted in Fig. 8 which shows how the the linear boundary learned by the network during training correctly separates

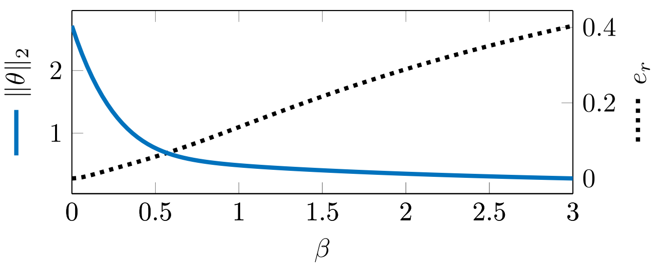

Fig. 6. Effect of the regularization term on the output reconstruction error and on the parameters vector norm.

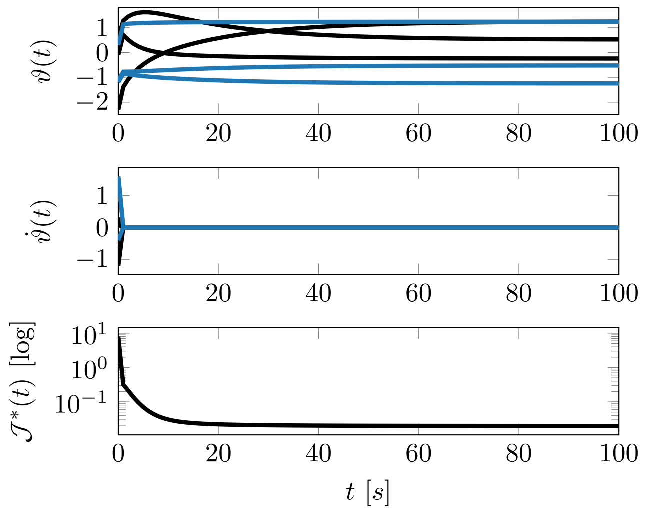

Fig. 7. Time evolution of the parameters during the sequential training on the linear boundary problem. Black indicates parameters of $y^{(1)}$ while blue parameters of $y^{(2)}$.

the two classes, correctly classify all the points of the test set.

B. Task 2: Learning a vector field

To further test the performance of the proposed training approach in a more complex scenario, the problem of approximating a vector field has been addressed. Consider a nonlinear ODE

$$\frac{du}{dx} = \Phi(u) \qquad u \in \mathbb{R}^n,\ \Phi : \mathbb{R}^n \to \mathbb{R}^n,\ x \in \mathbb{R}\,. \tag{15}$$

The learning task consisted in training a fully–connected neural network to approximate the vector field Φ by using only some samples of the state u. Thus, input data have been generated collecting state observations along a trajectory in $s+1$ points x_i

$$\hat{u}_i \triangleq u(x_i).$$

The corresponding labels have been computed approximating the state derivative via forward difference, i.e.

$$\hat{y}_i = \frac{u(x_{i+1}) - u(x_i)}{x_{i+1} - x_i} \approx \Phi(u(x_i)) \quad \forall i = 1, \dots, s\ .$$

Hence, the input and output datasets $\mathcal{U}_s$, $\mathcal{Y}_s$ have been built and, then, the neural network has been trained with the PHNN method. The objective was to obtain a network able to infer the knowledge of the vector field, learned on a single

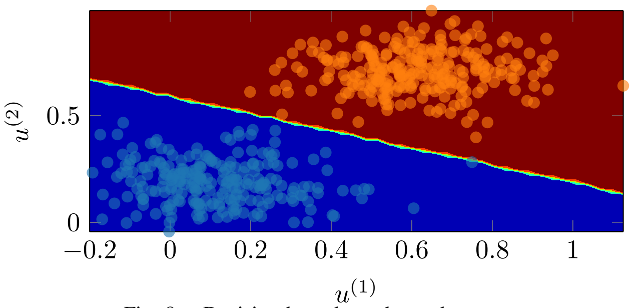

Fig. 8. Decision boundary plot and test set.

trajectory, to a wider region of the state space. Thus, the metric chosen to evaluate the training performance has been the absolute approximation error in a domain $\mathcal{D}$:

$$\mathcal{E}(u) \triangleq \|\Phi(u) - f(u, \vartheta^*)\|_2 \quad u \in \mathcal{D},$$

where ϑ^* is the optimized vector of parameters. Notice that the accuracy of the results is increased by the choice of nonlinear activation function σ.

The chosen ODE model has been a *Duffing oscillator* [31]

$$\frac{du}{dx} = \begin{bmatrix} u^{(2)} \\ -u^{(1)} - u^{(2)} - 0.5\left(u^{(1)}\right)^3 \end{bmatrix}.$$

Given the initial condition $u_0 = [1.5, 1]^\top$, a trajectory $u(t)$ was numerically integrated in $x \in [0, 8]$ via the `odeint` solver of *Scipy* library and 400 evenly distributed measurements have been collected (i.e., $\delta x = x_{i+1} - x_i = 0.2 \; \forall i = 1, \ldots, s$). The vector field and the computed trajectory are shown in Fig. 10.

A three layers neural network has been selected with the two hidden layers having a width $h_1 = h_2 = 16$. The total number of network parameters is $p = 354$. The design of a network with two hidden layers instead of a single, larger hidden layer or additional, narrower layers is motivated by [32] and [33]. While traditionally depth has been regarded as the more important attribute, recent developments have shown that a correct balance of depth and width can be beneficial for neural network performance.

The activation function has been selected as $\sigma(\cdot) \triangleq \frac{1}{\gamma}\ln(1 + e^{-\gamma(\cdot)})$ ($\gamma = 10$). This function, referred as *softplus*, is the smooth counterpart of the more popular *ReLu* activation. ReLu offers fast convergence to a minimum due to its linear region but is not differentiable in 0. While in practice this drawback rarely causes problem due to numerical approximation, the choice of softplus was made to not violate the smoothness assumption of $\mathcal{J}^*$.

The PH model of the parameters and the objective function have been defined as in Example 4.2 with $\alpha = 1$, $\beta = 0$ and $B = 0.5 \cdot I_p$. ϑ and $\dot{\vartheta}$ has been initialized sampling a Gaussian distribution with unitary variance and a uniform distribution over $[0, 1)$ respectively. The training has been performed with *batch* method by numerically integrating the PH model for 100s. The training outcomes of first $30s$ the are shown in Fig. 9. It can be noticed that after $30s$ most of the parameters have converged, thus reaching a minimum of $\mathcal{J}^*_{\texttt{batch}}$. Around the $20s$ point some of the velocities show a ripple, followed by a variation of the corresponding parameters. The loss decay is simultaneously accelerated during this event due to the dissipation term $B\dot{\vartheta}$. This behavior is most likely due to the state passing through a saddle point of the $\mathcal{J}^*_{\texttt{batch}}$.

The error $\mathcal{E}(u)$ has then been computed for $u \in \mathcal{D} \triangleq [-1, 1.5] \times [-1.9, 1]$. Figure 11 shows that the reconstruction error is highest in the state-space regions from which the neural network received no training information. The neural network has been able to infer the shape of the vector field elsewhere, especially in regions with a higher training data density.

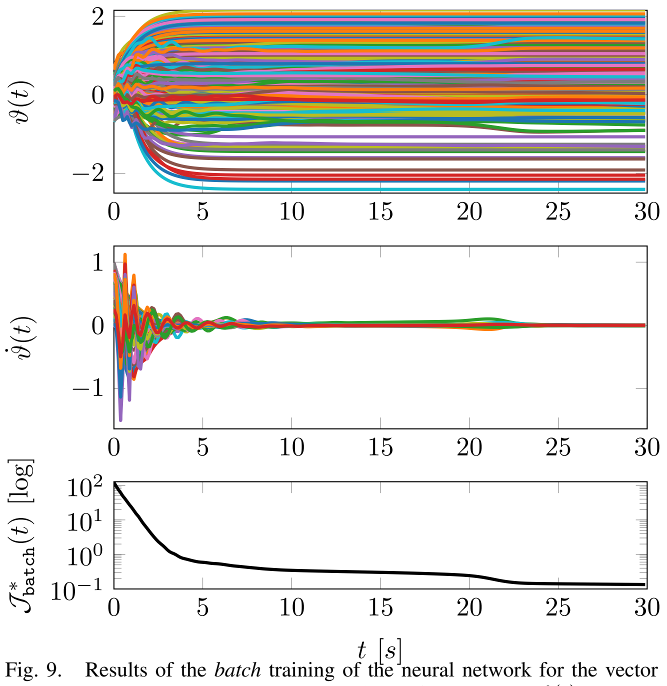

Fig. 9. Results of the *batch* training of the neural network for the vector field reconstruction. [Above] Trajectory of the 354 parameters $\vartheta(t)$ and their velocities $\dot{\vartheta}$. [Below] Decay of the loss function over time.

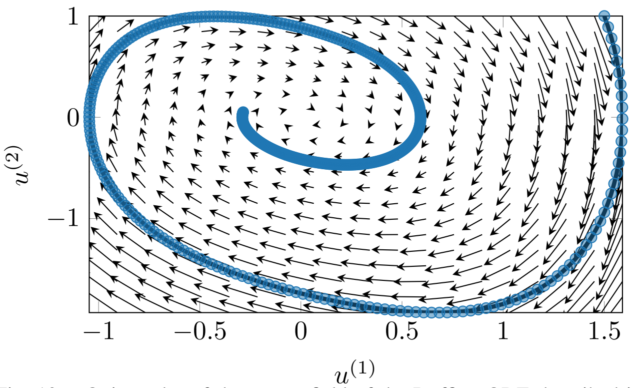

Fig. 10. Quiver plot of the vector field of the Duffing ODE described in. The blue points are sampled from a single continuous trajectory and used for the batch training procedure.

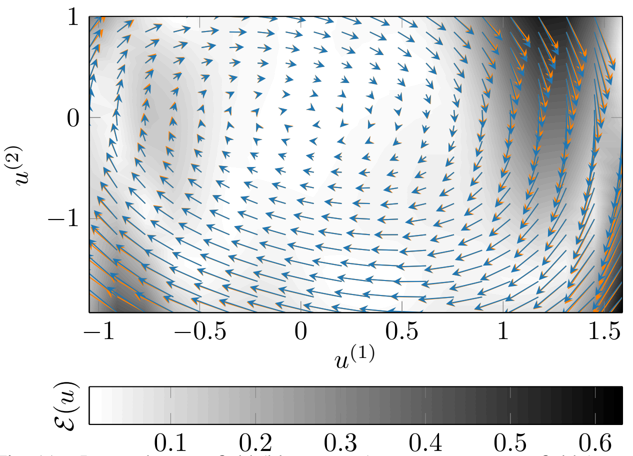

Fig. 11. Learned vector field (blue arrows) versus true vector field (orange arrows) and absolute reconstruction error.

VI. CONCLUSIONS

In this work we provide a new perspective on the process of neural network optimization. Inspired by their modular nature, we design objective function and parameter training dynamics in such a way that the neural network itself behaves as an autonomous Port-Hamiltonian system. A result is the implicit guarantee on convergence to a minimum of the loss function due to PH passivity.

In the context of training neural networks, escaping from saddle points has been a challenge due to the non–convexity and high–dimensionality of the optimization problem. The proposed framework is promising since it it circumvents the problem of getting stuck at saddle points by guaranteeing convergence to a minimum of the loss function.

In juxtaposition with the discrete nature of many other popular neural network optimization schemes currently used in state-of-the-art deep learning models, our framework features a continuous evolution of the parameters. Future work will be carried out in order to exploit this property to shed more light on some of the underlying characteristics of neural networks, especially those with a high number of layers, the behavior of which is proving to be quite challenging to model.

Additionally, this framework enables a treatment of neural networks based on physical systems and PH control which can increase the performance of the learning procedure and the probability of finding the global minimum of the objective function. Here, we performed experiments on classification and vector field approximation and determined that the proposed method scales up to neural networks of non-trivial size.

REFERENCES

[1] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[2] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[4] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[5] 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.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[8] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[9] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[10] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6391–6401, 2018.

[11] Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pages 494–501, 1989.

[12] Bernhard M Maschke and Arjan J van der Schaft. Port-controlled hamiltonian systems: modelling origins and systemtheoretic properties. *IFAC Proceedings Volumes*, 25(13):359–365, 1992.

[13] Vincent Duindam, Alessandro Macchelli, Stefano Stramigioli, and Herman Bruyninckx. *Modeling and control of complex physical systems: the port-Hamiltonian approach*. Springer Science & Business Media, 2009.

[14] Arjan van der Schaft, Dimitri Jeltsema, et al. Port-hamiltonian systems theory: An introductory overview. *Foundations and Trends® in Systems and Control*, 1(2-3):173–378, 2014.

[15] Romeo Ortega, Arjan J Van Der Schaft, Iven Mareels, and Bernhard Maschke. Putting energy back in control. *IEEE Control Systems Magazine*, 21(2):18–33, 2001.

[16] Romeo Ortega, Arjan Van Der Schaft, Bernhard Maschke, and Gerardo Escobar. Interconnection and damping assignment passivity-based control of port-controlled hamiltonian systems. *Automatica*, 38(4):585–596, 2002.

[17] Romeo Ortega, Arjan Van Der Schaft, Fernando Castanos, and Alessandro Astolfi. Control by interconnection and standard passivity-based control of port-hamiltonian systems. *IEEE Transactions on Automatic control*, 53(11):2527–2542, 2008.

[18] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6572–6583, 2018.

[19] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

[20] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *arXiv preprint arXiv:1906.01563*, 2019.

[21] Pratik Chaudhari, Adam Oberman, Stanley Osher, Stefano Soatto, and Guillaume Carlier. Deep relaxation: partial differential equations for optimizing deep neural networks. *Research in the Mathematical Sciences*, 5(3):30, 2018.

[22] James W Howse, Chaouki T Abdallah, and Gregory L Heileman. Gradient and hamiltonian dynamics applied to learning in neural networks. In *Advances in Neural Information Processing Systems*, pages 274–280, 1996.

[23] Wieslaw Sienko, Wieslaw Citko, and Dariusz Jakóbczak. Learning and system modeling via hamiltonian neural networks. In *International Conference on Artificial Intelligence and Soft Computing*, pages 266–271. Springer, 2004.

[24] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[25] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[26] Parry Hiram Moon and Domina Eberle Spencer. *Theory of holors: A generalization of tensors*. Cambridge University Press, 2005.

[27] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999.

[28] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[29] Stefano Massaroli, Federico Califano, Angela Faragasso, Atsushi Yamashita, and Hajime Asama. Multistable energy shaping of linear time–invariant systems with hybrid mode selector. In *Submitted to 11th IFAC Symposium on Nonlinear Control Systems (NOLCOS 2019)*, 2019.

[30] Arjan J Van Der Schaft and Johannes Maria Schumacher. *An introduction to hybrid dynamical systems*, volume 251. Springer London, 2000.

[31] Ivana Kovacic and Michael J Brennan. *The Duffing equation: nonlinear oscillators and their behaviour*. John Wiley & Sons, 2011.

[32] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, pages 6231–6239, 2017.

[33] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016.