Proceedings of Machine Learning Research 277:1–25, 2025 Machine Learning Meets Differential Equations: From Theory to Applications

A Unified Framework for Neural Computation and Learning Over Time

Stefano Melacci STEFANO.MELACCI@UNISI.IT
DIISM, University of Siena (Italy)

Alessandro Betti ALESSANDRO.BETTI@IMTLUCCA.IT
IMT School for Advanced Studies Lucca (Italy)

Michele Casoni M.CASONI@STUDENT.UNISI.IT
DIISM, University of Siena (Italy)

Tommaso Guidi TOMMASO.GUIDI@UNISI.IT
DIISM, University of Siena (Italy)

Matteo Tiezzi MATTEO.TIEZZI@IIT.IT
PAVIS, Istituto Italiano di Tecnologia, Genova (Italy)

Marco Gori MARCO.GORI@UNISI.IT
DIISM, University of Siena (Italy)

Editors: Cecília Coelho, Bernd Zimmering, M. Fernanda P. Costa, Luís L. Ferrás, Oliver Niggemann

Abstract

This paper proposes Hamiltonian Learning, a novel unified framework for learning with neural networks "over time", i.e., from a possibly infinite stream of data, in an online manner, without having access to future information. In this paper, the problem of learning over time is rethought from scratch, leveraging tools from optimal control theory, which yield a unifying view of the temporal dynamics of neural computations and learning. Hamiltonian Learning is based on differential equations that: (*i*) can be integrated without the need of external software solvers; (*ii*) generalize the well-established notion of gradient-based learning in feed-forward and recurrent networks; (*iii*) open to novel perspectives. The proposed framework is showcased by experimentally proving how it can recover gradient-based learning, comparing it to out-of-the box optimizers, and describing how it is flexible enough to switch from fully-local to partially/non-local computational schemes, possibly distributed over multiple devices, and BackPropagation without storing activations. Hamiltonian Learning is easy to implement and can help researches approach in a principled and innovative manner the problem of learning over time.

Keywords: Differential Equations, Optimization, Optimal Control, Learning Over Time.

1. Introduction

Motivations. A longstanding challenge in machine learning with neural networks is the one of designing models and learning strategies that are naturally conceived to learn "over time", progressively adapting to the information from a stream of data (Wang et al., 2024; Gori and Melacci, 2023). This implies dealing with possibly infinite streams, online learning, no access to future information, thus going beyond classic statistical methods exploiting offline-collected datasets. In this paper, tools from optimal control theory (Bertsekas, 2012) are exploited to rethink learning over time *from scratch*, proposing a unifying framework

© 2025 S. Melacci, A. Betti, M. Casoni, T. Guidi, M. Tiezzi & M. Gori.

named Hamiltonian Learning (HL). Differential equations drive the learning dynamics, which are integrated going "forward" in time, i.e., without back-propagating to the past. Although related tools are widely used (in a different way) in reinforcement learning (Khetarpal et al., 2022), and there exist works exploiting control theory in the context of neural networks (Chen et al., 2018; Kidger et al., 2020; Araujo et al., 2023), to the best of our knowledge they have not addressed the general problem of learning over time from a continuous, possibly infinite, stream of data. A few recent exceptions do exist, but they are focused on specific cases with significant limitations (Betti et al., 2022, 2024). HL is rooted on a state-space formulation of the neural model, that yields computations which are fully local in space and time.

Contributions. (*i*) We propose a unified framework for neural computation and learning over time, exploiting well-established tools from optimal control theory. HL is designed to leverage the Hamilton Equations (HE) (Bertsekas, 2012) to learn in a forward manner, facing an initial-value problem, instead of a boundary-value problem (Section 2–preliminaries, Section 3–HL). (*ii*) We (formally and experimentally) show that, by integrating the differential equations with the Euler method, thus without any off-the-shelf solvers for differential equations, our framework recovers the most popular gradient-based learning, i.e., BackPropagation and BackPropagation Through Time (Section 4). (*iii*) We discuss how HL provides a flexible view of neural computation over a stream of data, which is fully local in time and space. This favors parallelization, distributed computation, and it also generalizes memory efficient BackPropagation (also Through Time) (Section 5). The generality of HL is not intended to provide tools to solve classic issues in continual learning (e.g., catastrophic forgetting, stability–refer to (Betti et al., 2025) for stability of HL dynamics), but to provide researches with a flexible framework that, due to its novelty, generality, and accessibility, might open to novel achievement in learning over time.

Related Work. Dynamic Programming (Bardi et al., 1997)) usually involves boundary-value problems. These problems are approached with iterative forward/backward schemes over the whole considered time interval (or exploiting receding horizon control (Garcia et al., 1989)). Betti et al. (2022) evaluated HE in continual online learning, even if artificially forcing the costate dynamic to converge to zero. The work of LeCun (LeCun, 1988) mentions HEs and control theory, in relation with classical mechanics. While Neural ODE and related works (Chen et al., 2018; Kidger et al., 2020; Gholaminejad et al., 2019; Zhang et al., 2019; Massaroli et al., 2020) are also based on optimal control, here we focus on the problem of learning over time in an online manner, with a possibly infinite horizon, which is different from what is commonly done in the literature of Neural ODE/CDE. There exist several works focussed on learning in an online manner from streamed data, such as in the case of physics-inspired models (Betti et al., 2019; Tiezzi et al., 2020) or approaches to continual online learning (Mai et al., 2022; Wang et al., 2024). The framework of this paper is devised by revisiting the learning problem from scratch, instead of trying to directly adapt classic statistical approaches typical of offline learning. The classic approach to learning online with recurrent models is RTRL (Irie et al., 2024; Marschall et al., 2020), which, differently from HL, requires to store and progressively update the temporal Jacobian matrix, with high space/time complexities. Forward-only learning (Kohan et al., 2018; Ororbia and Mali, 2023; Hinton, 2022) share with HL the principles of locality in the computations.

2. Preliminaries

Notation. At time $t \geq 0$ data $\mathbf{u}_t \in \mathbb{R}^d$ is provided to a neural network (for inference and learning), with learnable parameters collected into vector $\boldsymbol{\theta}$, with $t \in [0, N)$, $N > 0$, that could be possibly ∞. Our framework is devised in a continuous-time setting, which, as usual, is implemented by evaluating functions at discrete time instants, that might be not evenly spaced. Vectors are indicated in bold (they are column vectors–lowercase), where $\mathbf{o}^T$ is the transpose of $\mathbf{o}$. The notation $[\mathbf{o}, \mathbf{u}]$ is the concatenation of the comma-separated vectors in brackets. Given an m-real-valued function, e.g., $r(\mathbf{u})$ returning vector $\mathbf{o}$, the Jacobian matrix with respect to its argument of length d is a matrix in $\mathbb{R}^{m \times d}$. As a consequence, gradients of scalar functions ($m = 1$) are row vectors. The only exception to this rule is in the case of derivatives with respect to time, indicated with $\dot{\mathbf{o}}$, that are still column vectors, as $\mathbf{o}$. We will frequently follow the convention of writing $\mathbf{o}$ in place of $r(\mathbf{u}, \mathbf{h})$, keeping track of the dependencies with respect to the arguments of r. Dependencies do not propagate through time. The operator $\odot$ is the Hadamard product.

Feed-Forward and Recurrent Networks. A feed-forward neural model (including convolutional nets) is a (possibly deep) architecture that, given input $\mathbf{u}$, computes the values of the output neurons as $\mathbf{h} = f(\mathbf{u}, \boldsymbol{\theta})$. Recurrent models are designed to process finite sequences of samples, such as $\mathbf{u}_0$, $\mathbf{u}_1$, …, $\mathbf{u}_{n-1}$, where we can evaluate the output values of the recurrent neurons at step $\kappa + 1$ as,

$$\mathbf{h}_{\kappa+1} = f(\mathbf{u}_\kappa, \mathbf{h}_\kappa, \boldsymbol{\theta}), \quad \text{for } \kappa = 0, 1, \ldots, n-1. \tag{1}$$

In this case, computations depend on the initial $\mathbf{h}_0$ which is commonly set to zeros or to random values (Medsker et al., 2001). Notice that $\boldsymbol{\theta}$ is not a function of κ, since the same weights are used to process the whole sequence. Moreover, Eq. 1 is a generalization of feed-forward networks, which are obtained by keeping κ fixed and setting the second argument of f to $\mathbf{0}$ (vector of zeros). In the case of classic online gradient, weights are updated after having processed each example (feed-forward nets), or each sequence (recurrent nets),

$$\boldsymbol{\theta}_{\upsilon+1} = \boldsymbol{\theta}_\upsilon - \gamma \tilde{\boldsymbol{\theta}}_\upsilon \tag{2}$$

where υ is a generic step index, $\gamma > 0$ is the learning rate and $-\gamma\tilde{\boldsymbol{\theta}}_\upsilon$ is the variation.

State and Stream. We consider an extended definition of state with respect to the one of recurrent networks, $\mathbf{h}$. In particular, the state here is $[\mathbf{h}, \theta]$, which represents a snapshot of the model at a certain time instant, including all the information that is needed to compute Eq. 1 when an input is given. Differently from dataset-oriented approaches, here we consider a unique source of information, the stream $\mathcal{S}$, that, at time t, yields an input-target pair, $\mathcal{S}_t = (\mathbf{u}_t, \hat{\mathbf{y}}_t)$.[1] We will avoid explicitly mentioning batched data, and, for simplicity, we will assume a consistent spacing of τ between consecutive samples (see Appendix A).

3. Hamiltonian Learning

Control theory is focused on finding valid configurations (controls) to optimally drive the temporal dynamics of a system. In the case of neural networks, we aim at finding the

1. Targets could also be present only for some t's.

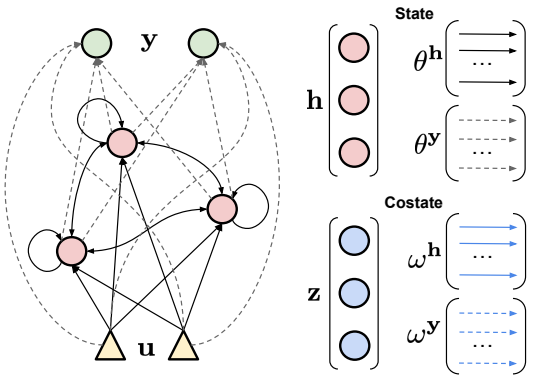


Figure 1: Left: State net (solid lines) and output net (dashed lines); Right: state ($\mathbf{h}$ and $\boldsymbol{\theta} = [\boldsymbol{\theta}^{\mathbf{h}}, \boldsymbol{\theta}^{\mathbf{y}}]$). Costate ($\mathbf{z}$ and $\boldsymbol{\omega} = [\boldsymbol{\omega}^{\mathbf{h}}, \boldsymbol{\omega}^{\mathbf{y}}]$) is also shown (not part of the net).

Init $\mathbf{h}_0$, $\boldsymbol{\theta}_0$, $\mathbf{z}_0$, $\boldsymbol{\omega}_0$, t. Select $\eta \geq 0$, $\boldsymbol{\beta} > 0$, and func ϕ_t.

while true **do**
  $(\mathbf{u}_t, \hat{\mathbf{y}}_t) \leftarrow \mathcal{S}_t$
  $\dot{\mathbf{h}}_t = f^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \theta^{\mathbf{h}}_t)$ ($\mathcal{E}\star 1$)
  $\mathbf{y}_t = f^{\mathbf{y}}(\mathbf{u}_t, \mathbf{h}_t, \theta^{\mathbf{y}}_t)$
  $\dot{\boldsymbol{\theta}}_t = -\boldsymbol{\beta} \odot \boldsymbol{\omega}_t$ ($\mathcal{E}\star 2$)
  $H' = L(\mathbf{y}_t, \hat{\mathbf{y}}_t)\phi_t + \mathbf{z}_t^T \dot{\mathbf{h}}_t$ (8)
  $\dot{\mathbf{z}}_t = \left(\frac{\partial H'}{\partial \mathbf{h}_t}\right)^T - \eta \mathbf{z}_t$ ($\mathcal{E}\star 3$)
  $\dot{\boldsymbol{\omega}}_t = \left(\frac{\partial H'}{\partial \boldsymbol{\theta}_t}\right)^T - \eta \boldsymbol{\omega}_t$ ($\mathcal{E}\star 4$)
  $\mathbf{h}_{t+\tau} = \mathbf{h}_t + \tau \dot{\mathbf{h}}_t$
  $\boldsymbol{\theta}_{t+\tau} = \boldsymbol{\theta}_t + \tau \dot{\boldsymbol{\theta}}_t$
  $\mathbf{z}_{t+\tau} = \mathbf{z}_t + \tau \dot{\mathbf{z}}_t$
  $\boldsymbol{\omega}_{t+\tau} = \boldsymbol{\omega}_t + \tau \dot{\boldsymbol{\omega}}_t$
  $t = t + \tau$
end while

Algorithm 1: Hamiltonian Learning (valid also for uneven time steps).

"optimal" way to drive the predictions of the model over time, by controlling the changes in $\boldsymbol{\theta}$. Formal aspects of control theory can be found in Appendix B.

State-Space Formulation. Following the formalism of continuous-time state-space models (Gu and Dao, 2023), we split the computations of the network as the outcome of two sub-networks: the *neuron state network*, implementing function $f^{\mathbf{h}}$, is responsible of computing how $\mathbf{h}$ changes over time; the *output network*, implementing function $f^{\mathbf{y}}$, is a recurrence-free portion of the network that receives data from $\mathbf{h}$ and, if needed, from the input units, propagating information to the output ones, as sketched in Fig. 1. Formally,

$$\dot{\mathbf{h}}_t = f^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t) \qquad \text{NEURON STATE NETWORK} \tag{3}$$

$$\mathbf{y}_t = f^{\mathbf{y}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{y}}_t) \qquad \text{OUTPUT NETWORK} \tag{4}$$

$$\dot{\boldsymbol{\theta}}_t = \boldsymbol{\beta} \odot \bar{\boldsymbol{\theta}}_t \qquad \text{WEIGHT VELOCITY} \tag{5}$$

where $\boldsymbol{\theta}^{\mathbf{h}}_t$ and $\boldsymbol{\theta}^{\mathbf{y}}_t$ are weights of the two networks, both of them function of time, with $\boldsymbol{\theta}_t = [\boldsymbol{\theta}^{\mathbf{h}}_t, \boldsymbol{\theta}^{\mathbf{y}}_t]$. Eq. 5 formalizes the dynamics of the weights, that will change over time. The term $\boldsymbol{\beta}$ is a customizable vector of positive numbers of the same size of $\boldsymbol{\theta}$, that can be used to tune the relative importance of each weight, while $\bar{\boldsymbol{\theta}}$ is the unscaled weight velocity. Computing $\mathbf{h}$ (resp. $\boldsymbol{\theta}$) requires us to integrate Eq. 3 (resp. Eq. 5). When using the explicit Euler's method with step τ, we get $\mathbf{h}_{t+\tau} = \mathbf{h}_t + \tau \dot{\mathbf{h}}_t$, and $\boldsymbol{\theta}_{t+\tau} = \boldsymbol{\theta}_t + \tau \dot{\boldsymbol{\theta}}_t$, given some $\mathbf{h}_0$ and $\boldsymbol{\theta}_0$. Notice that no future information is exploited and that, at time t, we do not keep track of the way $\mathbf{h}_t$ was computed, discarding dependencies from the past.

Learning Basics. The appropriateness of the network's predictions at t is measured by a differentiable loss function L. It is crucial to introduce regularity on the temporal dimension, to avoid trivial solutions in which weights abruptly change from a time instant to the following one. We define *cost*, computed by function C, as a mixture of instantaneous loss $L(\mathbf{y}_t, \hat{\mathbf{y}}_t)$, and a penalty that discourages fast changes to the weights of the model,

$$C(\mathbf{h}_t, \boldsymbol{\theta}_t, \bar{\boldsymbol{\theta}}_t, t) = L\left(f^{\mathbf{y}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{y}}_t), \hat{\mathbf{y}}_t\right) \phi_t + \frac{1}{2}\|\bar{\boldsymbol{\theta}}_t\|^2, \tag{6}$$

where $\|\bar{\boldsymbol{\theta}}\|^2$ is the squared Euclidean norm of vector $\bar{\boldsymbol{\theta}}$. In Eq. 6 we also introduced a (customizable) positive scalar-valued function ϕ_t to tune the scale of the loss over time. The explicit dependence of C on time is also due to the data (input-target) streamed at time t.

Optimal Control. The goal of learning is to minimize the *total cost* on the considered time horizon w.r.t. the dynamics that drive the way weights change over time, Eq. 5,

$$\bar{\boldsymbol{\theta}} \in \arg\min_{\underline{\boldsymbol{\theta}}} \int_0^N e^{\eta t} C(\mathbf{h}_t, \boldsymbol{\theta}_t, \underline{\boldsymbol{\theta}}_t, t)\ dt, \tag{7}$$

which is basically saying that we are looking for the best way to move/control the weight values over time (Bertsekas, 2012). For this reason, $\bar{\boldsymbol{\theta}}$ represents the optimal trajectory of the weight velocity, i.e., the optimal *control* of the problem. The cost is scaled by an exponential function with $\eta \geq 0$ that has no effects when $\eta = 0$. This choice introduces a form of dissipation. future).

Costate. Let us introduce an adjoint variable named *costate*, $[\mathbf{z}, \boldsymbol{\omega}]$, which has the same structure/dimensions of the state $[\mathbf{h}, \boldsymbol{\theta}]$. Since $\boldsymbol{\theta} = [\boldsymbol{\theta}^{\mathbf{h}}, \boldsymbol{\theta}^{\mathbf{y}}]$, we also have $\boldsymbol{\omega} = [\boldsymbol{\omega}^{\mathbf{h}}, \boldsymbol{\omega}^{\mathbf{y}}]$, see Fig. 1. The costate is a popular notion in the field of optimal control, following the Pontryagin's maximum principle (Bertsekas, 2012). In our proposal, instead of trying to directly compute the optimal way in which the weights change over time, i.e., $\bar{\boldsymbol{\theta}}$ of Eq. 5 (that is a function valid $\forall t$), we will redefine the control in terms of the newly introduced costate, that will make computations manageable. Thus, we will first estimate the costate at each t and, afterwards, we will use it to evaluate $\bar{\boldsymbol{\theta}}$.

Hamilton Equations. The way the costate changes over time and, more generally, the whole dynamics of the model are described by the so-called Hamilton Equations (HEs), which are the key component of HL. HEs are ODEs that provide a way of solving the problem of Eq. 7, since it can be formally shown that a minimum of Eq. 7 con be reconstructed from a solution of such ODEs (Bertsekas, 2012; Betti et al., 2024). Since the HEs formalize how to compute $\dot{\mathbf{h}}$, $\dot{\boldsymbol{\theta}}$, $\dot{\mathbf{z}}$, $\dot{\boldsymbol{\omega}}$ (where the first two ones are the same as Eq. 3, Eq. 5), they also formally describe how weights change over time, i.e., $\dot{\boldsymbol{\theta}}$, which is what we need to let the network evolve. HEs are computed by differentiating the so-called Hamiltonian H, which is a scalar function that sums the integrand of Eq. 7 with the dot product between state $[\mathbf{h}, \boldsymbol{\theta}]$ and costate $[\mathbf{z}, \boldsymbol{\omega}]$. The Hamiltonian is defined in a way such that, at each t, it is at a minimum value with respect to the control (Appendix B). Given H, the HEs are $\dot{\mathbf{h}} = \partial H / \partial \mathbf{z}$, $\dot{\boldsymbol{\theta}} = \partial H / \partial \boldsymbol{\omega}$, $\dot{\mathbf{z}} = -\partial H / \partial \mathbf{h}$, and $\dot{\boldsymbol{\omega}^{\mathbf{h}}} = -\partial H / \partial \boldsymbol{\theta}$ (Lewis et al., 2012; Betti et al., 2024).

Hamiltonian Learning. HEs can only be exploited in boundary-value problems, requiring knowledge of the state at $t = 0$ and of the costate at $t = N$, and then performing dynamic programming in the range $[0, N]$, updating the costate going backward in time. Which is not feasible for possibly infinite intervals. The proposed HL framework consists of a revisited instance of the Hamiltonian and of the HEs in order to learn over time in a forward manner. $(i.)$ We avoid to deal with an exponential function of time in H, that leads to numerical issues, by re-parameterizing the costate in an appropriate manner, yielding an exponential-free Hamiltonian H' and exponential-free HEs (derivations in Appendix C). Moreover, $(ii.)$ $\dot{\boldsymbol{\theta}}$ has a simple form in the HEs, which does not require automatic differentiation to be computed, thus we can avoid involving it in the computation of H'. Hence, we replace the Hamiltonian H with what we refer to as *Robust Hamiltonian* H', which has no exponential

functions of time and does not depend at all on the second part of the costate, $\dot{\boldsymbol{\omega}}$,

$$H'(\mathbf{h}_t, \boldsymbol{\theta}_t) \ = L(\mathbf{y}_t, \hat{\mathbf{y}}_t)\phi_t + \mathbf{z}_t^T \dot{\mathbf{h}}_t. \tag{8}$$

The first two HEs consist of

$$\dot{\mathbf{h}}_t = f^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}_t^{\mathbf{h}}) \tag{$\mathcal{E}\star1$}$$

$$\dot{\boldsymbol{\theta}}_t = -\boldsymbol{\beta} \odot \boldsymbol{\omega}_t \tag{$\mathcal{E}\star2$}$$

where Eq. $\mathcal{E}\star1$ is the same as Eq. 3 (as expected), while Eq. $\mathcal{E}\star2$ is what we can directly compute without involving automatic differentiation, and it involves both the parameters of the state network and the ones of the output net (since $\boldsymbol{\theta}$ is the concatenation of both of them). The two remaining HEs are computed by differentiating the Robust Hamiltonian,

$$\dot{\mathbf{z}}_t = -s\left(\frac{\partial H'(\mathbf{h}_t, \boldsymbol{\theta}_t)}{\partial \mathbf{h}_t}\right)^T - \eta \mathbf{z}_t = -s\left(\phi_t \frac{\partial L(\mathbf{y}_t, \hat{\mathbf{y}}_t)}{\partial \mathbf{h}_t} + \mathbf{z}_t^T \frac{\partial \dot{\mathbf{h}}_t}{\partial \mathbf{h}_t}\right)^T - \eta \mathbf{z}_t \tag{$\mathcal{E}\star3$}$$

$$\dot{\boldsymbol{\omega}}_t = -s\left(\frac{\partial H'(\mathbf{h}_t, \boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}_t}\right)^T - \eta \boldsymbol{\omega}_t = \begin{bmatrix} \dot{\boldsymbol{\omega}}_t^{\mathbf{h}} \\ \\ \dot{\boldsymbol{\omega}}_t^{\mathbf{y}} \end{bmatrix} = \begin{bmatrix} -s\left(\mathbf{z}_t^T \frac{\partial \dot{\mathbf{h}}_t}{\partial \boldsymbol{\theta}_t^{\mathbf{h}}}\right)^T - \eta \boldsymbol{\omega}_t^{\mathbf{h}} \\ \\ -s\left(\phi_t \frac{\partial L(\mathbf{y}_t, \hat{\mathbf{y}}_t)}{\partial \boldsymbol{\theta}_t^{\mathbf{y}}}\right)^T - \eta \boldsymbol{\omega}_t^{\mathbf{y}} \end{bmatrix}, \tag{$\mathcal{E}\star4$}$$

where the additive terms weighed by η are due to the re-parametrization that allowed us to remove the exponential function, and that act as dissipation terms, avoiding explosion of the dynamics. See Appendix C for all the derivations. In the case of Eq. $\mathcal{E}\star4$, we also explicitly split the HE considering the contribution from the state networks and the one from the output network, respectively. The binary value $s \in \{-1, 1\}$ is equal to 1 in the original HEs, which are solved backward in time. (*iii.*) We follow the basic intuition of reverting the direction of time when evaluating Eq. $\mathcal{E}\star3$ and Eq. $\mathcal{E}\star4$, and solving them going forward in time. Hence, we replace the terminal condition of the boundary-value problem with an initial one, setting the costate at time $t = 0$ to zeros, and forcing $s = -1$ in Eq. $\mathcal{E}\star3$ and Eq. $\mathcal{E}\star4$, changing the sign of (the non-dissipative part of) the costate-related HEs. Of course, the original guarantees of optimality are lost, but it opens to a viable way of learning in temporally causal manner. As we will show in Section 4, this choice is not only driven by intuition, but it is what allows HL to correctly generalize common gradient-based learning. We summarize HL in Algorithm 1, that is all that is needed to be implemented to setup it, since derivatives can be computed by automatic differentiation.

Instantaneous Propagation. When time does not bring any important information for the task at hand, e.g., when not considering time-series, then we can cancel the effects of the selected integration procedure by a residual-like transition function in Eq. 3,

$$f^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}_t^{\mathbf{h}}) = \tau^{-1}\left(-\mathbf{h}_t + \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}_t^{\mathbf{h}})\right), \tag{9}$$

where the neuron state network is basically a scaled residual network. Indeed, we add $\mathbf{h}$ to the outcome of the neural computation implemented by the newly introduced function $\hat{f}^{\mathbf{h}}$, and then we divide by τ. In this way, when integrating the state equation (Eq. 4) with the Euler's method, we get $\mathbf{h}_{t+\tau} = \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}_t^{\mathbf{h}})$.

4. Recovering Gradient-based Learning

In order to help trace connections with existing mainstream technologies, in this section we neglect the important locality properties of HL to clarify the relations to the usual gradient-based minimization by BackPropagation (BP) or BackPropagation Through Time (BPTT) (Pascanu et al., 2013). Full derivations and extended descriptions are in Appendix D.

Feed-Forward Networks (Output Nets). Implementing a feed-forward net by means of Eq. 4 sounds natural due to the static nature of the output map $f^{\mathbf{y}}$. The notion of neuron state is lost, thus $\mathbf{h}_t = \mathbf{0}$, $\forall t$, and, consequently H' of Eq. 8 boils down to the usual loss function. The costate-related HE of Eq. $\mathcal{E}\star 4$ (lower part), due to our choice of $s = -1$, is the gradient of the loss with respect to the weights (let us temporarily discard the dissipation term). When integrating the output-net-related HEs, Eq. $\mathcal{E}\star 4$, Eq. $\mathcal{E}\star 2$, respectively, setting $H' = L$ and $s = -1$, we get a momentum-based computation of the gradient of L with respect to $\boldsymbol{\theta}^{\mathbf{y}}$ (Sutskever et al., 2013). However, in classic gradient-based learning, a specific order of operations is strictly followed: first gradient computation and, only afterwards, weights are updated with the just computed gradients, and we have to enforce this constraint in HL as well (Appendix D). Fig. 2 reports the outcome of comparing gradient-based learning using popular out-of-the-box tools and HL, showing that they lead to the same trajectories of the loss function, considering MLPs, Transformers (ViT), ResNets (details in Appendix E).

Feed-Forward Networks (State Nets). Implementing a feed-forward network using the state network requires to clear the state $\mathbf{h}_t$ and costate $\mathbf{z}_t$ for each t, setting them to $\mathbf{0}$, to avoid propagating neuron-level information from previous examples. In this case, $f^{\mathbf{y}}$ is the identity function, and we can exploit the residual-like formulation of Eq. 9 to compensate the effects of the integration procedure (that depends on τ). Then, all the comments we made about the analogies with momentum-based gradient learning, BP, and learning rate scheduling are valid also in this case (derivations in Appendix D). See Fig. 2 for comparisons with plain gradient-descent (MLP, ViT, ResNet).

Recurrent Neural Networks (Automatic Unfolding). Given a sequence of n samples, streamed at times $t_0, \dots t_{n-1}$, learning in Recurrent Neural Nets (RNNs) is driven by BPTT (Pascanu et al., 2013), where n states are stored while going forward, with constant weights $\boldsymbol{\theta}$, and then gradients are computed in a backward manner. The most straightforward way to trace a connection between BPTT and HL is by conceiving the recurrent network in its unfolded form, and by rethinking $\mathcal{S}$. If we assume $\mathcal{S}$ to yield one-full sequence at each time step, the unfolded RNN becomes a feed-forward network, and it can be implemented as discussed in the above text. As usual, the unfolding can be done on the fly, thus either $f^{\mathbf{h}}$ or $f^{\mathbf{y}}$ can be specifically implemented as RNNs, reusing existing software implementations. This is what we did in Movie Review experiments (IMDb (Maas et al., 2011)) of Fig. 2.

Recurrent Neural Networks (Hamiltonian Learning). A more intriguing way to recover BPTT is obtained by keeping our original definition of $\mathcal{S}$, which yields a sequence streamed one-token-per-time (n tokens), from t_0 to t_{n-1}, followed by the same tokens in reverse order.[2] We still have to store the sequence of the n generated $\mathbf{h}$'s, loosing locality. When $t \leq t_{n-1}$, given some initial $\mathbf{h}_{t_0}$, $\boldsymbol{\theta}_{t_0}$, the first two HEs, Eq. $\mathcal{E}\star 1$, Eq. $\mathcal{E}\star 2$, will drive the evolution of the system. When $t > t_{n-1}$, we basically swap the relevance of the two pairs of HEs, giving emphasis to Eq. $\mathcal{E}\star 3$ and Eq. $\mathcal{E}\star 4$. This is motivated by fact that we

2. Truncated BPTT of length r can be recovered by streaming the last $r-1$ tokens in reverse order.

are going to process the reversed sequence, and we do care about tracking the sensitivity of the model with respect to changes in the learnable parameters, which is the role of the costate. When integrating the costate equations, using the proposed $s = -1$, we get the largely known update rules of BPTT for sequential data, as detailed in Appendix D and G.

5. Discussion & Future Perspectives

We presented HL, a unified framework for neural computation and learning over time, exploiting tools from control theory. Differential equations drive learning, leveraging stateful networks that are fully local in time and space, recovering classic BackPropagation (and BPTT) in the case of non-local models. HL represents a novel perspective, that might inspire researchers to further investigate learning over time in a principled manner, being useful framework where to study known stability issues in learning over time.

Fully Local Learning. The stateful nature of learning allows us to instantiate networks that are not only temporally local, but also spatially local, once we design $f^{\mathbf{h}}$ to return the variations of the output values of *all* the its neurons. Given the graph that describes the parent/child relations among neurons, being it Directed Acyclic/Cyclic Graph, all the neurons can compute their outputs in parallel (Meloni et al., 2022) (Appendix A), thanks to the state $\mathbf{h}_t$ (and/or the current $\mathbf{u}_t$)–computed during the previous step–to feed their inputs, with important connections to biologically plausible computational models and related studies (Stork, 1989; Sacramento et al., 2018; Meulemans et al., 2021).

Memory Efficient BackPropagation. Neural ODEs/CDEs (Chen et al., 2018; Kidger et al., 2020) exploit a stateful model to implement the way the signal propagates through a feed-forward network (Chen et al., 2018) or when providing fixed-length sequences (Kidger et al., 2020), using an ODE solver to update the state and, afterwards, to compute gradients. These models do not require to store the output of all the "layers" (which is a vague concept in Neural ODE/CDEs) while backpropagating the learning signal, since the second call to the ODE solver can basically estimate the output of the previous layer given the output

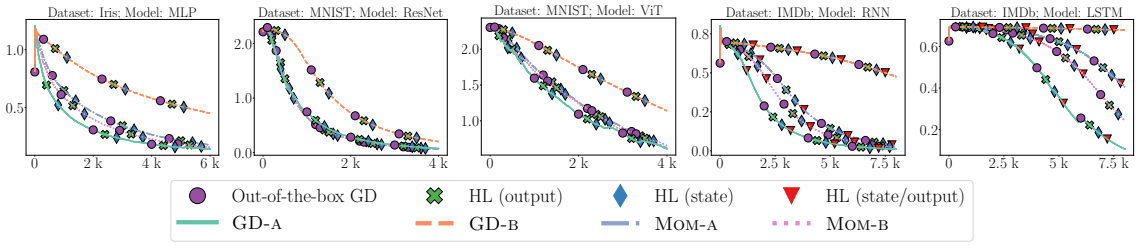


Figure 2: Comparing HL with out-of-the-box Pytorch optimizers, setup in Appendix E. Each figure is about a different dataset/architecture, reporting the loss L in function of time; there are 4 scenarios (curve color/linestyle): w/o (GD-A, GD-B) and w/ momentum (Mom-A, Mom-B). In HL, we implemented each net by solely considering the output function (*output*), the state function (*state*), or, in the case of RNNs and LSTMs, jointly considering both (*state/output*). *The difference between weights from the Pytorch optimizer and HL is zero or within the order of round-off error: curves of the same scenario overlap.*

of the following one.[3] Similarly to Neural ODE, in HL, given $\mathbf{u}$ (generic input sample–we removed the time index), we can distribute the sequential computation of layers over time (see Appendix A, Eq. 10). If we integrate the state-transition with the midpoint rule (Zhu et al., 2022), the direction of the integration can be also reversed, hence the midpoint rule can also yield the previous state given the following one(s). This is related to what happens in Reversible ReLU (Gomez et al., 2017),that allow to compute gradients without storing activations. Interestingly, in HL, the same properties are simply the outcome of having changed the integration technique.

Acknowledgments

This work was funded by project CONSTR: a COllectionless-based Neuro-Symbolic Theory for learning and Reasoning", PARTENARIATO ESTESO "Future Artificial Intelligence Research - FAIR", SPOKE 1 "Human-Centered AI" Università di Pisa, "NextGenerationEU", CUP I53C22001380006. This work was also supported by the University of Siena (Piano per lo Sviluppo della Ricerca - PSR 2024, F-NEW FRONTIERS 2024), under the project "TIme-driveN StatEful Lifelong Learning" (TINSELL).

References

Alexandre Araujo, Aaron J Havens, Blaise Delattre, Alexandre Allauzen, and Bin Hu. A unified algebraic perspective on lipschitz neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=k71IGLC8cfc.

Martino Bardi, Italo Capuzzo Dolcetta, et al. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*, volume 12. Springer, 1997.

Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

Alessandro Betti, Marco Gori, and Stefano Melacci. Cognitive action laws: The case of visual features. *IEEE transactions on neural networks and learning systems*, 31(3):938–949, 2019.

Alessandro Betti, Lapo Faggi, Marco Gori, Matteo Tiezzi, Simone Marullo, Enrico Meloni, and Stefano Melacci. Continual learning through hamilton equations. In *Conference on Lifelong Learning Agents*, pages 201–212. PMLR, 2022.

Alessandro Betti, Michele Casoni, Marco Gori, Simone Marullo, Stefano Melacci, and Matteo Tiezzi. Neural time-reversed generalized riccati equation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 7935–7942, 2024.

Alessandro Betti, Marco Gori, and Stefano Melacci. Stability of state and costate dynamics in continuous time recurrent neural networks. *ESANN 2025 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium)*, 2025. URL https://doi.org/10.14428/esann/2025.ES2025-164.

3. Costate in HL corresponds to adjoint sensitivity in Neural ODE.

Piermarco Cannarsa and Carlo Sinestrari. *Semiconcave functions, Hamilton-Jacobi equations, and optimal control*, volume 58. Springer Science & Business Media, 2004.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Richard Courant and David Hilbert. *Methods of mathematical physics: partial differential equations*. John Wiley & Sons, 2008.

R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

Amir Gholaminejad, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. In *Proceedings of IJCAI-19*, pages 730–736, 7 2019. doi: 10.24963/ijcai.2019/103. URL https://doi.org/10.24963/ijcai.2019/103.

Mariano Giaquinta and Stefan Hildebrandt. *Calculus of variations II*, volume 311. Springer Science & Business Media, 2013.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. *Advances in neural information processing systems*, 30, 2017.

Marco Gori and Stefano Melacci. Collectionless artificial intelligence. *arXiv preprint arXiv:2309.06938*, 2023.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

Kazuki Irie, Anand Gopalakrishnan, and Jürgen Schmidhuber. Exploring the promise and limits of real-time recurrent learning. *ICLR 2024, arXiv preprint arXiv:2305.19044*, 2024.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, 2022.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

Adam A Kohan, Edward A Rietman, and Hava T Siegelmann. Error forward-propagation: Reusing feedforward connections to propagate errors in deep learning. *arXiv preprint arXiv:1808.03357*, 2018.

Yann LeCun. A theoretical framework for back-propagation. *Proceedings of the 1988 connectionist models summer school*, 1, 1988.

Frank L Lewis, Draguna Vrabie, and Vassilis L Syrmos. *Optimal control*. John Wiley & Sons, 2012.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJg8TeSFDH.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2021.10.021. URL https://www.sciencedirect.com/science/article/pii/S0925231221014995.

Owen Marschall, Kyunghyun Cho, and Cristina Savin. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of machine learning research*, 21(135):1–34, 2020.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *Advances in Neural Information Processing Systems*, 33:3952–3963, 2020.

Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

Enrico Meloni, Lapo Faggi, Simone Marullo, Alessandro Betti, Matteo Tiezzi, Marco Gori, and Stefano Melacci. Partime: Scalable and parallel processing over time with deep neural networks. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 665–670, 2022. doi: 10.1109/ICMLA55696.2022.00110.

A. Meulemans, Matilde Tristany F., Javier Garcia Ordóñez, Pau Vilimelis Aceituno, João Sacramento, and Benjamin F Grewe. Credit assignment in neural networks through deep feedback control. *Advances in NeurIPS*, 34:4674–4687, 2021.

Alexander Ororbia and Ankur A Mali. The predictive forward-forward algorithm. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 45, 2023.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. *Advances in neural information processing systems*, 31, 2018.

Stork. Is backpropagation biologically plausible? In *International 1989 Joint Conference on Neural Networks*, pages 241–246 vol.2, 1989. doi: 10.1109/IJCNN.1989.118705.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Matteo Tiezzi, Stefano Melacci, Alessandro Betti, Marco Maggini, and Marco Gori. Focus of attention improves information transfer in visual features. *Advances in NeurIPS*, 33: 22194–22204, 2020.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. Anodev2: A coupled neural ode framework. *Advances in NeurIPS*, 32, 2019.

Aiqing Zhu, Pengzhan Jin, Beibei Zhu, and Yifa Tang. On numerical integration in neural ordinary differential equations. In *International Conference on Machine Learning*, pages 27527–27547. PMLR, 2022.

Appendix A. Further Details

Stream. The perspective of this paper is the one in which learning consists of the online processing of a single stream of data, that could be possibly infinite. In order to convert the classic notion of dataset to the one of stream $\mathcal{S}$, we can consider that $\mathcal{S}$, at time t, yields an input-target-tag triple, $\mathcal{S}_t = (\mathbf{u}_t, \hat{\mathbf{y}}_t, \delta_t)$, being δ_t a binary tag whose role will be clear in the following. A given dataset of samples can be streamed one sample after the other, possibly randomizing the order in case of stochastic learning, and $\delta_t = 1$, $\forall t$. If samples are sequences, then they are streamed one after the other. However, in this case each single sequence must be further streamed token-by-token, thus $\mathcal{S}$ provides a triple with $\delta_t = 1$ for the last token of the current sequence (otherwise $\delta_t = 0$), to preserve the information about the boundary between consecutively streamed sequences.

Scheduling. We assume the most basic schedule of computations: the agent which implements our neural networks monitors the stream and, when a batch of data is provided, it starts processing it, entering a "busy" state, and leaving it when it is done. Data from $\mathcal{S}$ is discarded when the agent is busy. At time t_j, the agent is aware of the time $\Delta(t_j)$ that passed from the previously processed sample.

Distributed Neural Computations. The aforementioned notion of spatial locality can be structured in several different ways. In a multi-layer feed-forward network, we can create

groups of consecutive layers, and assume $f^{\mathbf{h}}$ returns the variation of the output values of the *last layer of each group*, thus the state $\mathbf{h}$ is not composed by all the neuron outputs of the net. Computations within each group happens instantaneously, and delays are only among groups. This weakly spatially local architecture corresponds to the one exploited to build pipeline parallelism in neural networks (Meloni et al., 2022), when each group is processed on a different devices/GPUs. Interestingly, the notion of costate in HL is what allows the transfer of learning-related information among groups, that in (Meloni et al., 2022) is stored into appositely introduced variables.

Delay-free State Computation. HL offers the basic tools also to distribute layer-wise computations over time in a fully customizable way, going beyond the ones showcased in Section 5, for example by interleaving delay-free computations with delayed weight updates. Consider the case of a feed-forward network with ν layers, where $\mathbf{h}$ is the output of *all* the neurons of the net and $\hat{f}^{\mathbf{h}}$ is the function that computes such outputs. The data is sequentially provided, $\mathbf{u}_0$, $\mathbf{u}_1$, ..., $\mathbf{u}_\infty$, and the network is expected to make predictions on each of them (samples are i.i.d. in this case). We can introduce a time-variant state transition function $f^{\mathbf{h}}$, that now explicitly depends on time, without compromising the theoretical grounding of HL, since assuming this dependency is pretty natural in optimal control. Replacing time with indices κ, to simplify the notation, we have

$$\begin{aligned} f^{\mathbf{h}}(\mathbf{u}_{\lfloor \kappa/\nu \rfloor}, \mathbf{h}_\kappa, \boldsymbol{\theta}^{\mathbf{h}}_\kappa, \kappa) = \mathbf{1}_{\mathrm{mod}(\kappa,\nu)} \odot \\ \tau^{-1}\left(-\mathbf{h}_\kappa + \hat{f}^{\mathbf{h}}(\mathbf{u}_{\lfloor \kappa/\nu \rfloor}, \mathbf{h}_\kappa, \boldsymbol{\theta}^{\mathbf{h}}_\kappa)\right), \text{ for } \kappa = 0, 1, \ldots, \infty, \end{aligned} \tag{10}$$

where time/step κ is in the last argument, differently from what has been presented so far. We considered the model of Eq. 9, being $\mathbf{1}_\kappa$ a binary indicator vector which is 1 in the positions of the neurons of the κ-th layer and mod computes the remainder of the integer division. Each input sample is virtually repeated for ν steps, to wait for the signal to propagate through the network. At each step κ, only the portion of the state associated to a single layer is updated, due to the indicator function. Thus, after every κ/ν steps, $\mathbf{h}$ is composed of delay-free outputs of all the neurons of the network for a given input $\mathbf{u}_{\lfloor \kappa/\nu \rfloor}$. The model of Eq. 10 continuously updates weights over time, for each κ, using the forward-estimate of the costate. This means that while predictions are delay-free, there is a delay in the way weights are updated.

Appendix B. Optimal Control Theory

A control system consists of a pair (f, A), where $A \subset \mathbb{R}^m$ is called the *control set* and it is the set of admissible values of the control[4] and $f\colon \mathbb{R}^n \times A \times [t_0, +\infty) \to \mathbb{R}^n$ is a continuous function that is called the *dynamics of the system*. Using the notation (t) instead of subscript t as in we did in the main paper, the *state equation* associated to the system then is

$$\begin{cases} \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \boldsymbol{\alpha}(t), t), & \text{a.e. in } [t_0, +\infty) \\ \mathbf{x}(t_0) = \bar{\mathbf{x}} \end{cases} \tag{11}$$

4. Usually A is assumed to be closed even if in some cases, for instance when we want to choose $A = \mathbb{R}^m$ it is possible to trade the boundedness of A with the coercivity of the lagrangian (see (Cannarsa and Sinestrari, 2004) ch. 7, p. 214 for details).

where $t_0 \in \mathbb{R}$, $\bar{\mathbf{x}} \in \mathbb{R}^n$. The function $\boldsymbol{\alpha}$, misurable, is what is usually called the *control strategy* or *control function* and we denote with $\mathbf{x}(\,\cdot\,; t_0, \bar{\mathbf{x}}, \boldsymbol{\alpha})$ the unique solution[5] of Eq. (11).

Then an *optimal control* problem consists in choosing the control policy $\boldsymbol{\alpha}$ in such a way that a functional, usually called *cost* is minimized. In other words the objective of such class of problems is to steer the dynamics of the system so that it performs "well" according to a certain criteria. In this paper we are interested in what is known as control problem of *Bolza type*, in which, given a control system (f, A), a terminal cost $g \in C(\mathbb{R}^n)$ a time $N > 0$ and a function $\ell(\cdot, \cdot, s) \in C(\mathbb{R}^n \times A \times [t, N]; \mathbb{R})$ usually called *lagrangian*, for all $(t, \tilde{\mathbf{x}}) \in [0, N] \times \mathbb{R}^n$ the functional to be minimized is the total cost:

$$C_{t,\tilde{\mathbf{x}}}(\boldsymbol{\alpha}) := \int_t^N \ell(\mathbf{x}(s; t, \tilde{\mathbf{x}}, \boldsymbol{\alpha}), \boldsymbol{\alpha}(s), s)\, ds + g(\mathbf{x}(N; t, x, \boldsymbol{\alpha})). \tag{12}$$

Hence, more concisely the problem that optimal control (of Bolza type) is concerned with is

$$\inf C_{t,\tilde{\mathbf{x}}}(\boldsymbol{\alpha}) \tag{13}$$

over all possible control trajectories $\boldsymbol{\alpha}\colon [t, N] \to A$.

Here we briefly summarize some of the classical results that comes from the method of *dynamic programming* that are most relevant in this work and we refer to the excellent books (Bardi et al., 1997; Cannarsa and Sinestrari, 2004) for additional details, and in general for a more precise and consistent definition of the theory. The main idea behind dynamic programming is to embed the minimization problem (13) into a larger class of such problems through the definition of the *value function*:

$$V(t, \tilde{\mathbf{x}}) := \inf\{C_{t,\tilde{\mathbf{x}}}(\boldsymbol{\alpha}) : \boldsymbol{\alpha}\colon [t, T] \to A \quad \text{is measurable}\}.$$

In particular under suitable regularity conditions on the lagrangian, on the dynamics and on the terminal cost it is possible to show that the value function V satisfies the following Hamilton-Jacobi equation:

$$\begin{cases} V_t(t, \tilde{\mathbf{x}}) + H(\tilde{\mathbf{x}}, \nabla V(t, \tilde{\mathbf{x}}), t) = 0 & (t, \tilde{\mathbf{x}}) \in (0, N) \times \mathbb{R}^n \\ V(T, \tilde{\mathbf{x}}) = g(\tilde{\mathbf{x}}) & \tilde{\mathbf{x}} \in \mathbb{R}^n, \end{cases} \tag{14}$$

where the hamiltonian is defined as

$$\begin{aligned} H(\tilde{\mathbf{x}}, p, t) &= \min_{\mathbf{a} \in A} [p \cdot f(\tilde{\mathbf{x}}, \mathbf{a}, t) + \ell(\tilde{\mathbf{x}}, \mathbf{a}, t)] \\ &= -\max_{\mathbf{a} \in A} [-p \cdot f(\tilde{\mathbf{x}}, \mathbf{a}, t) - \ell(\tilde{\mathbf{x}}, \mathbf{a}, t)]. \end{aligned}$$

Once the value function is determined, the optimal control, i.e., the solution of the problem can then be recovered by means of what is known as the *synthesis procedure* (see (Bardi et al., 1997)). Basically once V is known we can define for all $\tilde{\mathbf{x}} \in \mathbb{R}^n$ and for all $\bar{t} \in [t, T]$ the *optimal feedback map* $S(\tilde{\mathbf{x}}, \bar{t}) \in \arg\min_{a \in A} \nabla V(\bar{t}, \tilde{\mathbf{x}}) \cdot f(\tilde{\mathbf{x}}, a, \bar{t}) + \ell(\tilde{\mathbf{x}}, \mathbf{a}, \bar{t})$ and with S compute the optimal trajectory $\mathbf{x}^*$ as a solution of $\dot{\mathbf{x}}(\bar{t} = f(\mathbf{x}(\bar{t}), S(\mathbf{x}(\bar{t}), \bar{t}), \bar{t})$ with initial condition $\mathbf{x}(t) = \bar{\mathbf{x}}$. Then, finally, the optimal control $\boldsymbol{\alpha}^*$ can be computed as $\boldsymbol{\alpha}^*(\bar{t}) = S(\mathbf{x}^*(\bar{t}), \bar{t})$.

5. We are indeed assuming that for all $a \in A$, $f(\cdot, a)$ is Lipschitz.

Hamilton Equations. An alternative approach to the problem in Eq. (13) relies on an alternative representation of the value function which is obtained through the method of characteristics (see (Courant and Hilbert, 2008)) and makes it possible to compute use a system of ODE instead of a PDE like Eq. (14) to construct the solution of the optimal control problem. This approach is also related to the Pontryagin Maximum Principle (Giaquinta and Hildebrandt, 2013).

Define the costate function as $\mathbf{p}(\bar{t}) = \nabla V(\bar{t}, \mathbf{x}(\bar{t}))$ and consider the following system of ODEs:

$$\begin{cases} \dot{\mathbf{x}}(\bar{t}) = H_p(\mathbf{x}(\bar{t}), \mathbf{p}(\bar{t}), \bar{t}) & \text{for } \bar{t} \in (t, N] \\ \dot{\mathbf{p}}(\bar{t}) = -H_x(\mathbf{x}(\bar{t}), \mathbf{p}(\bar{t}), \bar{t}) & \text{for } \bar{t} \in (t, N] \\ \mathbf{x}(t) = \bar{\mathbf{x}} & \text{INITIAL CONDITION} \\ \mathbf{p}(N) = \nabla g(\mathbf{x}(N)) & \text{TERMINAL CONDITION}, \end{cases} \tag{15}$$

where H_x and H_p are the partial derivatives of H with respect to its first and second argument respectively. Then if we are able to find the solution $(\mathbf{x}^*, \mathbf{p}^*)$ of the system (15), then the optimal control can be directly computed as

$$\boldsymbol{\alpha}^*(\bar{t}) \in \arg\min_{\mathbf{a} \in A} \mathbf{p}^*(\bar{t}) \cdot f(\mathbf{x}^*(\bar{t}), \mathbf{a}, \bar{t}) + \ell(\mathbf{x}^*(\bar{t}), \mathbf{a}, \bar{t}).$$

This way, we converted the problem of solving Eq. 13 in the point-wise minimization problem of the equation above.

Appendix C. Robust Hamiltonian and Forward Hamiltonian Equations

Considering the cost of Eq. 6, the Hamiltonian function is the outcome of adding to the instantaneous cost the dot product between state and costate, evaluated at the minimum with respect to the control (using the specific formalization of the main paper, differently from the previous section which was a generic introduction to Optimal Control),

$$\begin{aligned} H(\mathbf{h}, \boldsymbol{\theta}, \mathbf{z}, \boldsymbol{\omega} | \bar{\boldsymbol{\theta}}) = \\ &= e^{\eta t} C(\mathbf{h}, \boldsymbol{\theta}, \bar{\boldsymbol{\theta}}, t) + \mathbf{z}^T f^{\mathbf{h}}(\mathbf{u}, \mathbf{h}, \boldsymbol{\theta}^{\mathbf{h}}) + \boldsymbol{\omega}^T (\beta \odot \bar{\boldsymbol{\theta}}) \\ &= e^{\eta t} L\left(f^{\mathbf{y}}(\mathbf{u}, \mathbf{h}, \boldsymbol{\theta}^{\mathbf{y}}), \hat{\mathbf{y}}\right) \phi + \frac{e^{\eta t}}{2} \|\dot{\boldsymbol{\theta}}\|^2 \\ &\quad + \mathbf{z}^T f^{\mathbf{h}}(\mathbf{u}, \mathbf{h}, \boldsymbol{\theta}^{\mathbf{h}}) + \boldsymbol{\omega}^T (\beta \odot \bar{\boldsymbol{\theta}}) \end{aligned}$$

where we dropped the time index to keep the notation simple and where, from Eq. 5, $\dot{\boldsymbol{\theta}} = \boldsymbol{\beta} \odot \bar{\boldsymbol{\theta}}$. The definition of H remarks that, for each quadruple of arguments in Eq. 16, we are in a stationary point w.r.t. to the control $\bar{\boldsymbol{\theta}}$. From now on, we completely avoid reporting the arguments of the involved functions, considering them to be the ones of Eq. 16. We get

$$\frac{\partial H}{\partial \bar{\boldsymbol{\theta}}} = e^{\eta t} \bar{\boldsymbol{\theta}} + \boldsymbol{\omega} \odot \boldsymbol{\beta}. \tag{16}$$

Setting Eq. 16 to zero (since it is known to be a stationary point), we get,

$$\bar{\boldsymbol{\theta}} = -e^{-\eta t} \beta \odot \boldsymbol{\omega}. \tag{17}$$

Once we use Eq. 17 as value of the first argument of Eq. 16, the Hamiltonian becomes,

$$\begin{aligned} H &= e^{\eta t} L\phi + \frac{e^{\eta t}}{2}\|e^{-\eta t}\boldsymbol{\beta}\odot\boldsymbol{\omega}\|^2 + \mathbf{z}^T f^{\mathbf{h}} \\ &\quad - e^{-\eta t}\boldsymbol{\omega}^T(\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\boldsymbol{\omega}) \\ &= e^{\eta t} L\phi + \frac{e^{-\eta t}}{2}\|\boldsymbol{\beta}\odot\boldsymbol{\omega}\|_2^2 + \mathbf{z}^T f^{\mathbf{h}} \\ &\quad - e^{-\eta t}\boldsymbol{\omega}^T(\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\boldsymbol{\omega}) \end{aligned} \tag{18}$$

where we considered $\dot{\boldsymbol{\theta}} = \boldsymbol{\beta}\odot\bar{\boldsymbol{\theta}} = -e^{-\eta t}\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\boldsymbol{\omega}$ (merging its definition $\dot{\boldsymbol{\theta}} = \boldsymbol{\beta}\odot\bar{\boldsymbol{\theta}}$ with Eq. 17). Using the expression of H in Eq. 18, we can now compute the HEs,

$$\dot{\mathbf{h}} = \frac{\partial H}{\partial \mathbf{z}} = f^{\mathbf{h}} \tag{$\mathcal{E}\star\tilde{1}$}$$

$$\begin{aligned} \dot{\boldsymbol{\theta}} = \frac{\partial H}{\partial \boldsymbol{\omega}} &= e^{-\eta t}(\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\boldsymbol{\omega}) - 2e^{-\eta t}(\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\boldsymbol{\omega}) \\ &= -e^{-\eta t}(\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\boldsymbol{\omega}) \end{aligned} \tag{$\mathcal{E}\star\tilde{2}$}$$

$$\dot{\mathbf{z}} = -s\frac{\partial H}{\partial \mathbf{h}} = -se^{\eta t}\phi\frac{\partial L}{\partial \mathbf{h}} - s\mathbf{z}^T\frac{\partial f^{\mathbf{h}}}{\partial \mathbf{h}} \tag{$\mathcal{E}\star\tilde{3}$}$$

$$\dot{\boldsymbol{\omega}} = -s\frac{\partial H}{\partial \boldsymbol{\theta}} = -se^{\eta t}\phi\frac{\partial L}{\partial \boldsymbol{\theta}} - s\mathbf{z}^T\frac{\partial f^{\mathbf{h}}}{\partial \boldsymbol{\theta}}. \tag{$\mathcal{E}\star\tilde{4}$}$$

In order to avoid evaluating the exponential, we replace $\mathbf{z}$ with $e^{\eta t}\hat{\mathbf{z}}$ and $\boldsymbol{\omega}$ with $e^{\eta t}\hat{\boldsymbol{\omega}}$, which yield $\dot{\mathbf{z}} = e^{\eta t}\dot{\hat{\mathbf{z}}} + \eta e^{\eta t}\hat{\mathbf{z}}$ and $\dot{\boldsymbol{\omega}} = e^{\eta t}\dot{\hat{\boldsymbol{\omega}}} + \eta e^{\eta t}\hat{\boldsymbol{\omega}}$. Hence, the last two HEs can be written in function of the newly introduced variables $\hat{\mathbf{z}}$ and $\hat{\boldsymbol{\omega}}$,

$$\dot{\mathbf{h}} = f^{\mathbf{h}^T} \tag{$\mathcal{E}\star\tilde{1}$}$$

$$\dot{\boldsymbol{\theta}} = -\boldsymbol{\beta}\odot\boldsymbol{\beta}\odot\hat{\boldsymbol{\omega}} \tag{$\mathcal{E}\star\tilde{2}$}$$

$$e^{\eta t}\dot{\hat{\mathbf{z}}} + \eta e^{\eta t}\hat{\mathbf{z}} = -se^{\eta t}\phi\frac{\partial L}{\partial \mathbf{h}} - se^{-\eta t}\hat{\mathbf{z}}^T\frac{\partial f^{\mathbf{h}}}{\partial \mathbf{h}} \tag{$\mathcal{E}\star\tilde{3}$}$$

$$e^{\eta t}\dot{\hat{\boldsymbol{\omega}}} + \eta e^{\eta t}\hat{\boldsymbol{\omega}} = -se^{\eta t}\phi\frac{\partial L}{\partial \boldsymbol{\theta}} - se^{\eta t}\hat{\mathbf{z}}^T\frac{\partial f^{\mathbf{h}}}{\partial \boldsymbol{\theta}}. \tag{$\mathcal{E}\star\tilde{4}$}$$

which, also replacing $\boldsymbol{\beta}$ with $\hat{\boldsymbol{\beta}}^{1/2}$, can be simplified to

$$\dot{\mathbf{h}} = f^{\mathbf{h}} \tag{$\mathcal{E}\star\tilde{1}$}$$

$$\dot{\boldsymbol{\theta}} = -\hat{\boldsymbol{\beta}}\odot\hat{\boldsymbol{\omega}} \tag{$\mathcal{E}\star\tilde{2}$}$$

$$\dot{\hat{\mathbf{z}}} = -s\phi\frac{\partial L}{\partial \mathbf{h}} - s\hat{\mathbf{z}}^T\frac{\partial f^{\mathbf{h}}}{\partial \mathbf{h}} - \eta\hat{\mathbf{z}} \tag{$\mathcal{E}\star\tilde{3}$}$$

$$\dot{\hat{\boldsymbol{\omega}}} = -s\phi\frac{\partial L}{\partial \boldsymbol{\theta}} - s\hat{\mathbf{z}}^T\frac{\partial f^{\mathbf{h}}}{\partial \boldsymbol{\theta}} - \eta\hat{\boldsymbol{\omega}}. \tag{$\mathcal{E}\star\tilde{4}$}$$

Despite being defined as partial derivatives of the Hamiltonian of Eq. 16, the first two HEs above can be computed without involving any differentiation. This suggest that we can avoid

computing in H those terms that are responsible of yielding such first two HEs. Considering the last two HEs, the first two terms in the right-hand side are the derivatives of

$$L\phi + \hat{\mathbf{z}}^T f^{\mathbf{h}} \tag{19}$$

with respect to $\mathbf{h}$ and $\boldsymbol{\theta}$, multiplied by $-s$. The offsets $\eta\hat{\mathbf{z}}$ and $\eta\hat{\boldsymbol{\omega}}$ act like dissipation factors. Eq. 19 is the Robust Hamiltonian that was defined in Eq. 8. Comparing it with the "real" Hamiltonian of Eq. 16, there is no exponentiation and the costate of the weights is not there at all, thus it is simpler/faster to compute. Finally, in order to get the HEs reported in the main paper, i.e., Eq. $\mathcal{E}\star1$, Eq. $\mathcal{E}\star2$, Eq. $\mathcal{E}\star3$, Eq. $\mathcal{E}\star4$, we just need to go back to the original notation based on $\boldsymbol{\beta}$, $\dot{\mathbf{z}}$, $\mathbf{z}$, $\dot{\boldsymbol{\omega}}$, $\boldsymbol{\omega}$ instead of $\hat{\boldsymbol{\beta}}$, $\dot{\hat{\mathbf{z}}}$, $\hat{\mathbf{z}}$, $\dot{\hat{\boldsymbol{\omega}}}$. Notice that this is not a variable change, as we did before, but just a replacement of the notation.

Appendix D. Recovering Gradient-based Learning (Full Derivations)

HL offers a wide perspective to model learning over time with strong locality, that will be showcased in Section 5. However, in order to help trace connections with existing mainstream technologies, in this section we neglect the important locality properties of HL to clarify the relations to the usual gradient-based minimization by BackPropagation (BP) or BackPropagation Through Time (BPTT) (Pascanu et al., 2013). BPTT explicitly requires to store all the intermediate states, while BP assumes instantaneous propagation of the signal through the network. First, we show how to recover gradient-based learning in feed-forward networks (BP), presenting two different ways of modeling them in HL, either by means of the output network or of the state one. Then, we do the same for recurrent networks, where, in addition to the state/output network based implementations, we also show how HL can explicitly generalize BPTT by tweaking the way data is streamed.

Feed-Forward Networks (Output Nets). Implementing a feed-forward net by means of Eq. 4 sounds natural due to the static nature of the output map $f^{\mathbf{y}}$. The notion of neuron state is lost, thus $\mathbf{h}_t = \mathbf{0}$, $\forall t$, and, consequently H' of Eq. 8 boils down to the usual loss function. The costate-related HE of Eq. $\mathcal{E}\star4$ (lower part), due to our choice of $s = -1$, is the gradient of the loss with respect to the weights (let us temporarily discard the dissipation term). When integrating the output-net-related HEs, Eq. $\mathcal{E}\star4$, Eq. $\mathcal{E}\star2$, respectively, setting $H' = L$ and $s = -1$, we get

$$\boldsymbol{\omega}^{\mathbf{y}}_{t+\tau} = \boldsymbol{\omega}^{\mathbf{y}}_t + \tau \left(\frac{\partial L(\mathbf{y}_t, \hat{\mathbf{y}}_t)}{\partial \boldsymbol{\theta}^{\mathbf{y}}_t} \phi_t \right)^T - \tau\eta\boldsymbol{\omega}^{\mathbf{y}}_t = (1 - \tau\eta)\boldsymbol{\omega}^{\mathbf{y}}_t + \tau\phi_t \left(\frac{\partial L(\mathbf{y}_t, \hat{\mathbf{y}}_t)}{\partial \boldsymbol{\theta}_t} \right)^T \tag{20}$$

$$\boldsymbol{\theta}^{\mathbf{y}}_{t+\tau} = \boldsymbol{\theta}^{\mathbf{y}}_t - \tau\boldsymbol{\beta}^{\mathbf{y}}\boldsymbol{\omega}^{\mathbf{y}}_t. \tag{21}$$

Eq. 20 is a momentum-based computation of the gradient of L with respect to $\boldsymbol{\theta}^{\mathbf{y}}$ (Sutskever et al., 2013), where $1 - \tau\eta$ is the momentum factor and $\tau\phi_t$ is a time-dependent dampening coefficient (in other words, the costate variable $\boldsymbol{\omega}^{\mathbf{y}}_t$ effectively corresponds to "past gradients" as in the gradient descent momentum term). Eq. 21 is the classic weight update step, as the one we introduced in Eq. 2, with (per-parameter) learning rate $\tau\boldsymbol{\beta}^{\mathbf{y}}$, being $\boldsymbol{\beta}^{\mathbf{y}}$ the function $\boldsymbol{\beta}$ restricted to the parameters of the output network. However, in classic gradient-based learning, a specific order of operations is strictly followed: first gradient computation and, only afterwards, weights are updated with the just computed gradients. Hence, in HL we

have to replace $\boldsymbol{\omega}^{\mathbf{y}}_t$ of Eq. 21 with $\boldsymbol{\omega}^{\mathbf{y}}_{t+\tau}$, and then use the definition of $\boldsymbol{\omega}^{\mathbf{y}}_{t+\tau}$ given by Eq. 20. Under this constraint, we get a perfect match with classic gradient-based learning. Notice that when $\eta = 1/\tau$, the momentum term disappears. In such a setting, if ϕ is an exponential function, it acts as an exponential scheduler of the learning rate, while if ϕ is constant then no scheduling is applied (Li and Arora, 2020). Of course, we can also get rid of the momentum term by zeroing the weight-costate, $\boldsymbol{\omega}^{\mathbf{y}}_t = \mathbf{0}$ right before processing the example at time t. In Appendix E we report the exact rules to map learning parameters in momentum based gradient-descent and HL parameters. Fig. 2 reports the outcome of comparing gradient-based learning (with and without momentum) using popular out-of-the-box tools (different learning rates, momentum terms, damping factors and networks) and HL, showing that they lead to the same trajectories of the loss function, considering MLPs, Transformers (ViT), ResNets (see also Appendix E).

Feed-Forward Networks (State Nets). Implementing a feed-forward network using the state network requires to clear the state $\mathbf{h}_t$ and costate $\mathbf{z}_t$ for each t, setting them to $\mathbf{0}$, to avoid propagating neuron-level information from previous examples. In this case, $f^{\mathbf{y}}$ is the identity function, and we can exploit the residual-like formulation of Eq. 9 to compensate the effects of the integration procedure (that depends on τ). In such a setting, we get (details in Appendix F)

$$\mathbf{h}_{t+\tau} = \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t) = \tau \dot{\mathbf{h}}_t, \tag{22}$$

$$\mathbf{z}^T_{t+\tau} = -\tau s \phi_t \frac{\partial L(\mathbf{h}_{t+\tau}, \hat{\mathbf{y}}_t)}{\partial \mathbf{h}_{t+\tau}}, \tag{23}$$

where we used $\mathbf{h}_{t+\tau}$ as argument of L instead of $\mathbf{h}_t$. This choice is the direct counterpart of what we did when enforcing sequential ordering in gradient computation and weight update in the previously discussed implementation. In fact, we want the evaluation of the loss L to happen after the state neurons have been updated ($\mathbf{h}_{t+\tau}$), which is coherent with what happens when performing a prediction first, and only afterward evaluating the loss. Notice that, for the same reason, we also differentiate with respect to $\mathbf{h}_{t+\tau}$. Eq. 22 and Eq. 23 collect the two HEs of Eq. $\mathcal{E}\star 1$ and Eq. $\mathcal{E}\star 3$, respectively. In turn, the HE of Eq. $\mathcal{E}\star 4$ becomes,

$$\dot{\boldsymbol{\omega}}^{\mathbf{h}}_t = -s \left(\mathbf{z}^T_{t+\tau} \frac{\partial \dot{\mathbf{h}}_t}{\partial \boldsymbol{\theta}^{\mathbf{h}}_t} \right)^T - \eta \boldsymbol{\omega}^{\mathbf{h}}_t = \left(\phi_t \frac{\partial L(\mathbf{h}_{t+\tau}, \hat{\mathbf{y}}_t)}{\partial \mathbf{h}_{t+\tau}} \frac{\partial \mathbf{h}_{t+\tau}}{\partial \boldsymbol{\theta}^{\mathbf{h}}_t} \right)^T$$

$$- \eta \boldsymbol{\omega}^{\mathbf{h}}_t = \left(\phi_t \frac{\partial L(\mathbf{h}_{t+\tau}, \hat{\mathbf{y}}_t)}{\partial \boldsymbol{\theta}^{\mathbf{h}}_t} \right)^T - \eta \boldsymbol{\omega}^{\mathbf{h}}_t,$$

where, in the first equality, we replaced $\mathbf{z}_{t+\tau}$ with its definition reported in Eq. 23, while in the second one we replaced $\dot{\mathbf{h}}_t$ with $\tau^{-1}\mathbf{h}_{t+\tau}$ accordingly to Eq. 22, and discarded s, since $s^2 = 1$ (that is expected, since we are collapsing the dynamics of the two costate HEs). The last equality holds due to the chain rule. When integrated, it yields the same equation that we discussed in the previous implementation, Eq. 20.[6] Then, Eq. 21 still holds also in this case. As a consequence, all the comments we made about the analogies with momentum-based gradient learning, BP, and learning rate scheduling are valid also in this case. See Fig. 2 for comparisons with plain gradient-descent (MLP, ViT, ResNet).

6. Of course, replacing $\boldsymbol{\theta}^{\mathbf{y}}$ with $\boldsymbol{\theta}^{\mathbf{h}}$ and $\boldsymbol{\omega}^{\mathbf{y}}$ with $\boldsymbol{\omega}^{\mathbf{h}}$.

Recurrent Neural Networks (Automatic Unfolding). Given a sequence of n samples, streamed at times $t_0, \dots t_{n-1}$, learning in Recurrent Neural Nets (RNNs) is driven by BPTT (Pascanu et al., 2013), where n states are stored while going forward, with constant weights $\boldsymbol{\theta}$, and then gradients of $\sum_{i=0}^{n-1} L(\mathbf{y}_{t_i}, \hat{\mathbf{y}}_{t_i})$ with respect to $\boldsymbol{\theta}$ are computed in a backward manner. The most straightforward way to trace a connection between BPTT and HL is by conceiving the recurrent network in its unfolded form, and by rethinking $\mathcal{S}$. If we assume $\mathcal{S}$ to yield one-full sequence at each time step, then we can switch our attention from the RNN to the feed-forward network obtained by unfolding it, that receives as input the whole sequence at once. Being it a feed-forward network, it can be implemented as discussed in the above text. As usual, the unfolding can be done on the fly, thus either $f^{\mathbf{h}}$ or $f^{\mathbf{y}}$ can be specifically implemented as RNNs, reusing existing software implementations. This is what we did in the Movie Review experiments (IMDb (Maas et al., 2011) data) of Fig. 2.

Recurrent Neural Networks (Hamiltonian Learning). A more intriguing way to recover BPTT is obtained by keeping our original definition of $\mathcal{S}$, which yields a sequence streamed one-token-per-time (n tokens), from t_0 to t_{n-1}, followed by the same tokens in reverse order (without repeating the one at time t_{n-1}, i.e., the sample at t_{n-1+j} is equal to the one at t_{n-1-j}, $\forall j \geq 0$). We still exploit an identity output function and the formulation of Eq. 9 to compensate the effects of the integration. Of course we have to enforce the processing of the streamed data with the same weights, and only afterwards compute their variation, that can be achieved forcing $\boldsymbol{\beta} = \mathbf{0}$, $\forall t \neq 2t_n - t_0$, which is indeed a degenerate condition for $\boldsymbol{\beta}$ (see Eq. $\mathcal{E}\star 2$). When $t \leq t_{n-1}$, given some initial $\mathbf{h}_{t_0}$, $\boldsymbol{\theta}_{t_0}$, the first two HEs, Eq. $\mathcal{E}\star 1$, Eq. $\mathcal{E}\star 2$, will drive the evolution of the system. Coherently with BPTT, we must store the sequence of the n generated $\mathbf{h}$'s, loosing locality. The last two HEs, Eq. $\mathcal{E}\star 3$, Eq. $\mathcal{E}\star 4$, do not bring any information, since changes in the costate will not affect $\boldsymbol{\theta}$, due to our choice of zeroing $\boldsymbol{\beta}$. When $t = t_{n_1}$, all the HEs play an important role, while when $t > t_{n-1}$, we basically swap the relevance of the two pairs of HEs, giving emphasis to Eq. $\mathcal{E}\star 3$ and Eq. $\mathcal{E}\star 4$, and ignoring Eq. $\mathcal{E}\star 1$ and Eq. $\mathcal{E}\star 2$, since we already stored the n states. This is motivated by fact that we are going to process the reversed sequence, and we do care about tracking the sensitivity of the model with respect to changes in the learnable parameters, which is the role of the costate. The sequence of neuron states was already pre-computed, and it can be retrieved by mapping time by $\psi(t)\colon t \to 2t_{n-1} - 2\tau - t$, for $t > t_{n-1}$. When integrating the costate equations, using the proposed $s = -1$, we get (details in Appendix G),

$$\mathbf{z}_{t+\tau}^T = \tau\phi_t \frac{\partial L(\mathbf{h}_{t+\tau}, \hat{\mathbf{y}}_t)}{\partial \mathbf{h}_{t+\tau}} + \mathbf{z}_t^T \frac{\partial \mathbf{h}_{t+2\tau}}{\partial \mathbf{h}_{t+\tau}} - \tau\eta\mathbf{z}_t^T \tag{24}$$

$$\begin{aligned}\boldsymbol{\omega}_{t+\tau}^{\mathbf{h}} &= \boldsymbol{\omega}_t^{\mathbf{h}} + \tau \left(\mathbf{z}_{t+\tau}^T \frac{\partial \dot{\mathbf{h}}_t}{\partial \boldsymbol{\theta}_t^{\mathbf{h}}}\right)^T - \tau\eta\boldsymbol{\omega}_t^{\mathbf{h}} \\ &= \boldsymbol{\omega}_t^{\mathbf{h}} + \left(\mathbf{z}_{t+\tau}^T \frac{\partial \mathbf{h}_{t+\tau}}{\partial \boldsymbol{\theta}_t^{\mathbf{h}}}\right)^T - \tau\eta\boldsymbol{\omega}_t^{\mathbf{h}}.\end{aligned} \tag{25}$$

where in the HE of Eq. $\mathcal{E}\star 3$, coherently with what we did for feed-forward networks, we anticipated the state-update, thus all the terms involving $\mathbf{h}_t$ or $\dot{\mathbf{h}}_t$ are replaced with $\mathbf{h}_{t+\tau}$ and $\dot{\mathbf{h}}_{t+\tau}$, respectively. In order to match BPTT, at $t = t_{n-1}$ we set the costate to $\mathbf{0}$ to clear past information (notice that while $\mathbf{z}$ must be set to zeros, we can also avoid resetting

$\boldsymbol{\omega}^{\mathbf{h}}$, that will inherently introduce momentum, as already discussed), and we replace t with $\psi_t - \tau$ in the time index of non-costate-related quantities, yielding

$$\mathbf{z}_{t+\tau}^T = \tau\phi_{\psi_t-\tau}\frac{\partial L(\mathbf{h}_{\psi_t}, \hat{\mathbf{y}}_{\psi_t-\tau})}{\partial \mathbf{h}_{\psi_t}} + \mathbf{z}_t^T\frac{\partial \mathbf{h}_{\psi_t+\tau}}{\partial \mathbf{h}_{\psi_t}} - \tau\eta\mathbf{z}_t^T \tag{26}$$

$$\boldsymbol{\omega}_{t+\tau}^{\mathbf{h}} = \boldsymbol{\omega}_t^{\mathbf{h}} + \left(\mathbf{z}_{t+\tau}^T\frac{\partial \mathbf{h}_{\psi_t}}{\partial \boldsymbol{\theta}_{\psi_t-\tau}^{\mathbf{h}}}\right)^T - \tau\eta\boldsymbol{\omega}_t^{\mathbf{h}}. \tag{27}$$

When integrating the costate, the step size τ still has a role. If we select $\tau = 1$ and $\eta = 0$ (no dissipation), or $\phi_t = \tau^{-1}$ and $\eta = 0$, Eq. 26 and Eq. 27 are the largely known update rules of BPTT for sequential data, as detailed in Appendix G. This correspondence has the important role of confirming the validity on the choice we made in setting $s = -1$ in HL.

Recurrent Neural Networks (Truncated). Truncated BPTT of length r can be recovered by only resetting the state at the very beginning of the input sequence and, once the end of the sequence has been reached, by streaming the last $r-1$ tokens in reverse order, going back to the previously discussed case (which implies setting neuron-costate to zero right before processing the last token of the original sequence). If $r = 1$, we match the case of feed-forward neural networks, even if the initial state is not zeroed/ignored.

Appendix E. Out-of-the Box Tools vs. Hamiltonian Learning

The code of HL and the script that runs all the comparisons is attached to the submission.

Converting Parameter Values. In our experimental comparison, we exploited out-of-the box tools for gradient-based optimization in neural networks. The form of the update step in the case of gradient with momentum we considered

$$\begin{aligned}\mathbf{b}_{t+\tau} &= \mu\mathbf{b}_t + (1-\rho)\mathbf{g}_t \\ \boldsymbol{\theta}_{t+\tau} &= \boldsymbol{\theta}_t + \gamma(-\mathbf{b}_{t+\tau})\end{aligned} \tag{28}$$

where $\mathbf{g}$ is the gradient of the loss with respect to $\boldsymbol{\theta}$ (with $\mathbf{b}_\tau = \mathbf{g}_0$), while μ, ρ, γ are the momentum term, the dampening factor, and the learning rate, respectively, coherently with the Pytorch implementation of the SGD optimizer[7], that is what we used in our comparisons (time indices are adjusted following the notation of this paper).

After having selected any $\tau > 0$, we can map the optimizer parameters onto the parameters of Hamiltonian Learning as in the following table:

Pytorch Optim.	Hamiltonian Learning
γ	$\beta\tau$
μ	$(1-\tau\eta)$
ρ	$1-\tau\phi$

from which we have $\beta = \gamma/\tau$, $\eta = (1-\mu)/\tau$ and $\phi = (1-\rho)/\tau$. In Section 4 we compared the following cases:

7. https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

Pytorch Optim.	GD-a	GD-b	Mom-a	Mom-b
γ	0.01	0.001	0.01	0.01
μ	0	0	0.05	0.1
ρ	0	0	0.6	0.5

while we set τ to 1, 0.5, 1, 0.5 in the four cases of the table above, respectively.

Data & Setup. In this study, the batch size was consistently set to 1. We exploited a subsample of Iris[8] (Fisher, 1988), MNIST [9] and the IMDb datasets (Maas et al., 2011). The Iris dataset included all 150 samples across its 4 classes, with training conducted over 40 epochs. From the MNIST dataset, 100 test set samples were used, with 10 samples per class across 10 classes, also trained for 40 epochs. For the IMDb dataset, 100 training set samples were selected, split evenly between its 2 classes, and trained for 80 epochs due to the more challenging convergence. Experiments were executed on a Linux server with Pytorch 2.3.0, equipped with a 10th-Gen Intel CPU and 64GB of RAM (experiments take minutes to run, log, and plot results).

Models & Metrics. We evaluated six different model architectures: a single-layer neural network (Linear), a multi-layer perceptron with one hidden layer of 30 neurons using the tanh activation function (MLP), a small ResNet comprising 4 residual blocks (ResNet), a simple Vision Transformer with 1 encoder layer (30 hidden neurons, patch size 7x7), 1 layer normalization, 1 feedforward network (30 to 30 neurons), and 1 head (ViT), a recurrent neural network using pre-trained GloVe embeddings (50-dimensional), with a hidden state of 30 neurons and tanh activation, followed by a linear layer for classification (RNN), and a similar architecture to the RNN but with an LSTM unit instead of the RNN (LSTM). We report each model performance using the loss function values (cross-entropy) and accuracy metrics, during the model training phase.

Results. Figures 3-4 report the attained results in terms of loss values and accuracy, respectively, comparing the proposed HL method with several out-of-the-box gradient descent methods. The first and second rows illustrate the loss values for the different tested models, across the datasets (columns). For the Iris dataset (first column), both the Linear model and the MLP show that the proposed HL method (represented by green crosses and blue diamonds) aligns perfectly with the loss value of the out-of-the-box gradient descent (GD) methods (purple circles) in all the tested learning configurations (GD-a/b and Mom-a/b – various dashed lines). The same trend is observed in the MNIST dataset (second column) for the ResNet and ViT models, where HL maintains a loss value comparable to other GD methods. On the IMDb dataset (third column), both the RNN and LSTM models using HL also exhibit identical loss convergence dynamics to the ones of established GD techniques. Figure 4 focuses on accuracy. In the Iris dataset, the linear model and MLP achieve accuracy levels with HL that are consistent with those of the other methods. This consistency is further demonstrated in the MNIST dataset for the ResNet and ViT models, where HL's accuracy exactly recovers the one of other gradient descent approaches. For the IMDb dataset, both the RNN and LSTM models show that HL achieves accuracy identical to the traditional methods.

8. Iris dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license. https://archive.ics.uci.edu/dataset/53/iris
9. http://yann.lecun.com/exdb/mnist/

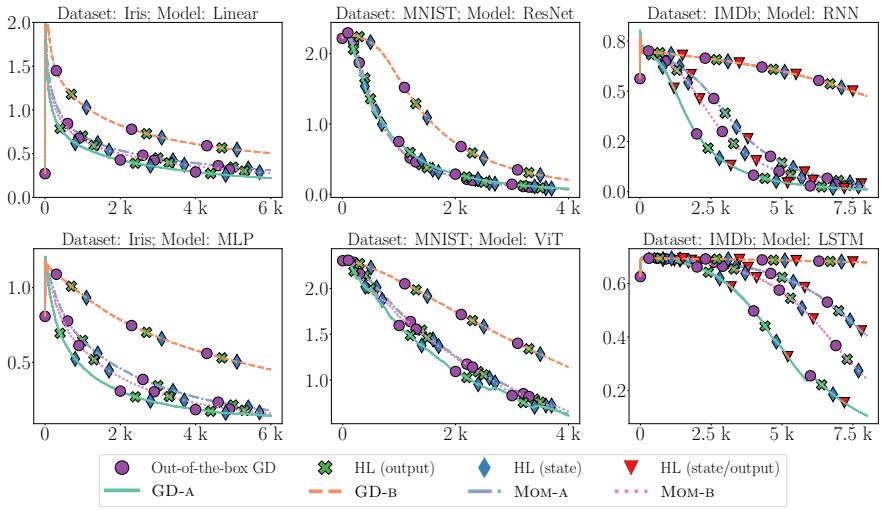


Figure 3: Experimental comparisons, Loss Values. We report the outcome of comparing gradient-based learning (with and without momentum, denoted with GD and Mom, respectively) using popular out-of-the-box tools (we tested two different configurations, denoted with the suffix "-A" and "-B", respectively, having different learning rates, momentum terms, damping factors) and Hamiltonian Learning (HL, setting τ, β, η, ϕ to values that we theoretically show to be coherent with the parameters of out-of-the-box tools– see Appendix E). When considering HL, we can implement the selected model by solely considering the output function (output), the state function (state), or we can split it putting a portion into the state and a portion into the output (state/output). The plot shows the perfect alignment in terms of cost function values during the models' training phase (x-axis, training steps).

Quantitative Numerical Differences. The starting point of the optimization is the same for all the compared cases, of course. We measured the numerical difference in weights values at the final training step, comparing the values attained by proposed HL method and the out-of-the-box gradient descent methods. In details, we compared the mean, min, max absolute differences of weights. It is numerically zero in most cases or within the order of round-off error. The Mom-A learning configuration is the only one which presented barely measurable differences, that are in the order of $\approx 1.0e^{-10}$ for the mean and absolute weight value difference, for all the considered models and settings. This indicates that HL produces the same model parameters, further supporting how it can generalize traditional optimization methods.

Appendix F. Feed-forward Networks and State Net

When implementing a feed-forward network using the state network, we rely on the instantaneous propagation of Eq. 9. We get

$$\begin{aligned}\mathbf{h}_{t+\tau} &= \mathbf{h}_t + \tau \cdot \tau^{-1}\left(-\mathbf{h}_t + \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t)\right) \\ &= \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t)\end{aligned}. \tag{29}$$

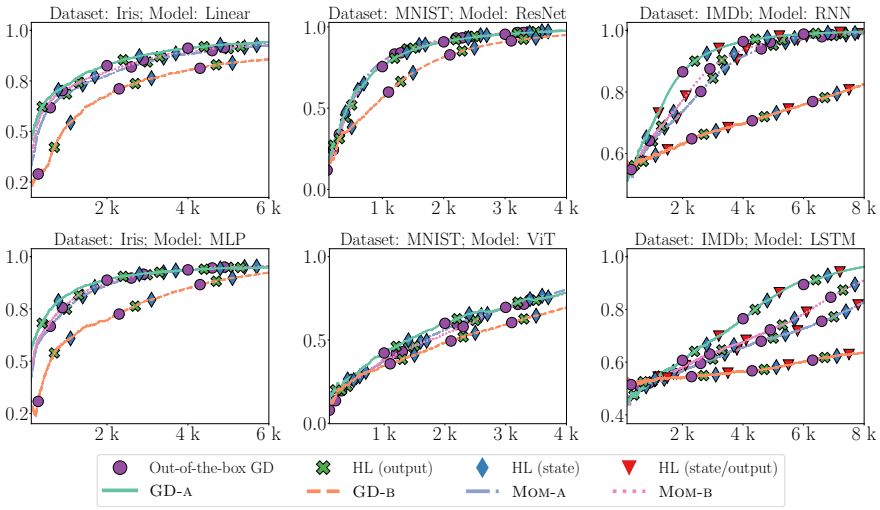


Figure 4: Experimental comparisons, Accuracy. Same setting of Figure 3. The plot shows the perfect alignment in terms of accuracy values during the models' training phase (x-axis, training steps).

In order to avoid dependencies from the past, we clear the state, $\mathbf{h}_t = \mathbf{0}$ and, since $\dot{\mathbf{h}}_t = \tau^{-1}(-\mathbf{h}_t + \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t))$, we get $\dot{\mathbf{h}}_t = \tau^{-1}\hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t)$. Comparing this last equation with Eq. 29, we obtain

$$\mathbf{h}_{t+\tau} = \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t) = \tau\dot{\mathbf{h}}_t. \tag{30}$$

which is the same as Eq. 22. Finally, Eq. 23 is trivially obtained from Eq. $\mathcal{E}\star 3$, setting $\mathbf{z}_t = \mathbf{0}$, to avoid propagating past information.

Appendix G. Learning in Recurrent Networks

Learning in Recurrent Neural Networks (RNNs) is usually instantiated with the goal of minimizing a loss function defined on a window of sequential samples whose extremes are a and b, with $b \geq a$, such as $\sum_{\kappa=a}^{b} L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)$, where $\mathbf{h}_\kappa$ is the hidden state at step κ. Weights are constant while processing the data of the window. Let us consider the case of a dataset of sequences, sampled in a stochastic manner, and let us assume that at a certain stage of the optimization the values of the weights of the RNN are indicated with $\boldsymbol{\theta}$, while $a = 1$ and b is the sequence length. Once gradients $\partial \sum_{\kappa=a}^{b} L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)/\partial\boldsymbol{\theta}$ are computed, weights are updated following the direction of the negative gradient.

Even if we avoided explicitly showing the dependence of L on $\boldsymbol{\theta}$, to keep the notation simple, the loss function at step κ, i.e., $L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)$, depends on $\boldsymbol{\theta}$ both due to its direct involvement into the computation of the current hidden state $\mathbf{h}_\kappa$ and also through all the previous states, up to $\mathbf{h}_a$. When applying the chain rule, following the largely known

BackPropagation Through Time (BPTT) (Pascanu et al., 2013), we get,

$$\begin{aligned}\frac{\partial \sum_{\kappa=a}^{b} L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)}{\partial \boldsymbol{\theta}} &= \sum_{\kappa=a}^{b} \frac{\partial L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)}{\partial \boldsymbol{\theta}} \\ &= \sum_{\kappa=a}^{b} \sum_{q=a}^{\kappa} \frac{\partial L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)}{\partial \mathbf{h}_\kappa} \frac{\partial \mathbf{h}_\kappa}{\partial \mathbf{h}_q} \frac{\partial \mathbf{h}_q}{\partial \boldsymbol{\theta}} \\ &= \sum_{q=a}^{b} \left(\sum_{\kappa=q}^{b} \frac{\partial L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)}{\partial \mathbf{h}_\kappa} \frac{\partial \mathbf{h}_\kappa}{\partial \mathbf{h}_q} \right) \frac{\partial \mathbf{h}_q}{\partial \boldsymbol{\theta}},\end{aligned} \tag{31}$$

where in the last derivation we swapped the order of the two sums in κ and q, adjusting the extremes of the summations. This can be easily done while noticing that the two summations in κ and q are represent the sum of the elements on a triangular matrix (lower triangular, if κ is the row index), thus we can sum on rows or columns first, interchangeably. We grouped some terms in brackets since the $\partial \mathbf{h}_q / \partial \boldsymbol{\theta}$ does not depend on index κ. Such terms can be iteratively evaluated going backward in time, exploiting the following relation,

$$\frac{\partial \mathbf{h}_b}{\partial \mathbf{h}_q} = \prod_{\kappa=q+1}^{b} \frac{\partial \mathbf{h}_\kappa}{\partial \mathbf{h}_{\kappa-1}} \tag{32}$$

with $q < b$, that can be used to compute the gradient w.r.t. $\boldsymbol{\theta}$ in an iterative manner, going over the input sequence in a backward manner. In fact, from Eq. 32 we get

$$\frac{\partial \mathbf{h}_b}{\partial \mathbf{h}_q} = \frac{\partial \mathbf{h}_b}{\partial \mathbf{h}_{q+1}} \frac{\partial \mathbf{h}_{q+1}}{\partial \mathbf{h}_q}, \tag{33}$$

which, starting from $q = b-1$, can be efficiently evaluated for $q = b-1, b-1, \ldots, a$. We introduce additional variables to progressively accumulate gradients, i.e., $\tilde{\mathbf{h}}$ and $\tilde{\boldsymbol{\theta}}$, with $\tilde{\mathbf{h}}_b = \mathbf{0}$ and $\tilde{\boldsymbol{\theta}}_b = \mathbf{0}$. For $q = b, b-1, \ldots, a$ (here we have to consider also the gradient of the loss, that is why q starts from b), we have

$$\tilde{\mathbf{h}}_{q-1} = \frac{\partial L(\mathbf{h}_q, \hat{\mathbf{y}}_\kappa)}{\partial \mathbf{h}_q} + \tilde{\mathbf{h}}_q^T \frac{\partial \mathbf{h}_{q+1}}{\partial \mathbf{h}_q} \tag{34}$$

$$\tilde{\boldsymbol{\theta}}_{q-1} = \tilde{\boldsymbol{\theta}}_q + \tilde{\mathbf{h}}_{q-1}^T \frac{\partial \mathbf{h}_q}{\partial \boldsymbol{\theta}}, \tag{35}$$

where $\partial \mathbf{h}_q / \partial \boldsymbol{\theta}$ can be immediately evaluated without involving data of the other time instants. Eq. 35 implements the (backward) summation on index q of Eq. 31 (bottom), where the term in brackets of Eq. 31 (bottom) is $\tilde{\mathbf{h}}_q$. The summation in such a term in brackets is what is evaluated by Eq. 34. Finally,

$$\frac{\partial \sum_{\kappa=a}^{b} L(\mathbf{h}_\kappa, \hat{\mathbf{y}}_\kappa)}{\partial \boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}_a. \tag{36}$$

We can compare the BPTT equations Eq. 34 and Eq. 35 with the ones from Hamiltonian Learning, Eq. 26 and Eq. 27. Before going into details of the comparison, we mention that,

to get Eq. 24, we exploited

$$\begin{aligned}\frac{\partial \dot{\mathbf{h}}_{t+\tau}}{\partial \mathbf{h}_{t+\tau}} &= \tau^{-1}\frac{\partial(-\mathbf{h}_{t+\tau} + \hat{f}^{\mathbf{h}}(\mathbf{u}_{t+\tau}, \mathbf{h}_{t+\tau}, \boldsymbol{\theta}^{\mathbf{h}}_{t+\tau}))}{\partial \mathbf{h}_{t+\tau}} \\ &= \tau^{-1}\left(-I + \frac{\partial \mathbf{h}_{t+2\tau}}{\partial \mathbf{h}_{t+\tau}}\right),\end{aligned} \tag{37}$$

and we considered $s = 1$. Similarly, to get Eq. 25 we exploited

$$\frac{\partial \dot{\mathbf{h}}_t}{\partial \boldsymbol{\theta}^{\mathbf{h}}_t} = \tau^{-1}\frac{\partial(-\mathbf{h}_t + \hat{f}^{\mathbf{h}}(\mathbf{u}_t, \mathbf{h}_t, \boldsymbol{\theta}^{\mathbf{h}}_t))}{\partial \boldsymbol{\theta}^{\mathbf{h}}_t} = \tau^{-1}\frac{\partial \mathbf{h}_{t+\tau}}{\partial \boldsymbol{\theta}^{\mathbf{h}}_t}, \tag{38}$$

since $\mathbf{h}_t$ does not depend on $\boldsymbol{\theta}^{\mathbf{h}}_t$ (again, $s = -1$). Going back to the comparison between the BPTT equations Eq. 34 and Eq. 35 and the ones from Hamiltonian Learning, Eq. 26 and Eq. 27, we notice that the gradient accumulators $\tilde{\mathbf{h}}$ and $\tilde{\boldsymbol{\theta}}$ play the same role of the costates $\mathbf{z}$ and $\boldsymbol{\omega}^{\mathbf{h}}$, respectively. BPTT proceeds backward, while Hamiltonian Learning goes forward, processing the sequence in reverse order. Thus, $\mathbf{z}_{t+\tau}$ and $\mathbf{z}_t$ correspond to $\tilde{\mathbf{h}}_{q-1}$ and $\tilde{\mathbf{h}}_q$, respectively (same comment for the case of $\boldsymbol{\omega}^{\mathbf{h}}$ and $\tilde{\boldsymbol{\theta}}$). In Hamiltonian Learning, we need to recover the states stored when processing the original sequence, thus all the $\mathbf{h}$ terms involve map ψ applied to time t. Target $\hat{\mathbf{y}}_\kappa$ is associated to $\hat{\mathbf{y}}_{\psi_t - \tau}$, showing that the time index of $\hat{\mathbf{y}}$ is precedent with respect to the one of $\mathbf{h}$ in the loss function. This is coherent with what we did when implementing feed-forward nets with the state network, Eq. 23, where we have the same offset between the times of $\mathbf{h}$ and $\hat{\mathbf{y}}$. Setting $\phi_t = \tau^{-1}$, and no dissipation ($\eta = 0$), the correspondence between the two pairs of equations is perfect (i.e. the pairs of BPTT equations and the pairs of the costate-related Hamiltonian Learning equations).