Stable Neural Flows

Stefano Massaroli[1,⋆], Michael Poli[2,⋆], Michelangelo Bin[3,⋆],
Jinkyoo Park[2], Atsushi Yamashita[1] and Hajime Asama[1]

***Abstract*— We introduce a provably stable variant of *neural ordinary differential equations* (neural ODEs) whose trajectories evolve on an energy functional parametrised by a neural network. *Stable neural flows* provide an implicit guarantee on asymptotic stability of the depth–flows, leading to robustness against input perturbations and low computational burden for the numerical solver. The learning procedure is cast as an optimal control problem, and an approximate solution is proposed based on adjoint sensivity analysis. We further introduce novel regularizers designed to ease the optimization process and speed up convergence. The proposed model class is evaluated on non–linear classification and function approximation tasks.**

I. INTRODUCTION

Neural networks are function compositions of the form

$$u \mapsto f_S \circ \cdots \circ f_0(u) := \phi_S(u)$$

which realize a possibly very complex nonlinear mappings between input and output spaces. Recent works [1], [2], [3], [4], explored the continuum limit of neural networks where the input–output mapping is realised by the solution (flow) of an ordinary differential equation $\frac{dx}{ds} = f(s, x(s)),\ x(0) = u$, defined on a compact *depth domain* $\mathcal{S} \subset \mathbb{R}$, where $s \in \mathcal{S}$ denotes the *depth* variable. At its core, this approach turns the task of *learning* an input–output map into a data–driven search for a suitable vector field f. In this context, f is often parametrised by a neural network $f := f(s, x(s), w(s))$ in which $w \in \mathcal{W}$. *Neural ordinary differential equations* (neural ODEs) have been successfully used as building blocks for more elaborate data–driven models [5], [6] and the formulation has been adapted to support other classes of differential equations, e.g. hybrid systems [7] or *stochastic differential equations* (SDE) [8]. While [3] models f as multi–layer feed–forward networks and discrete convolution operators, the framework has also been extended to spectral graph convolutions and networked models [9]. Neural ODEs have also been shown to improve upon previous learning methods in the approximation of physical systems [10], [11].

Although the framework has seen application in machine learning tasks, the lack of a rigorous derivation of the optimization process and stability guarantees prevent its widespread use in control applications. The unconstrained

[1]Stefano Massaroli is with the Department of Precision Engineering, the University of Tokyo, Tokyo, Japan massaroli@robot.t.u-tokyo.ac.jp

[2]Michael Poli is with the Department of Industrial Engineering, Korean Advanced Institute of Science and Technology, Daejeon, South Korea poli.m@kaist.ac.kr

[2]Michelangelo Bin is with the Department of Department of Electrical and Electronic Engineering, Imperial College London, London, United Kingdom

⋆Equal contribution authors

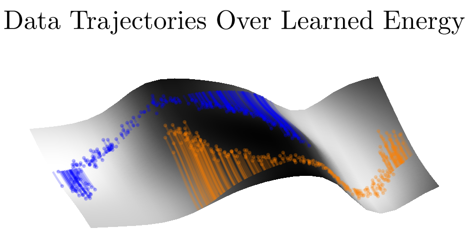

Fig. 1. Trajectories of the *half moons* data over learned energy functional, dissipated along the flows.

form of f may yield unstable and *stiff* dynamics, which in turn leads to an increase in sensitivity to input perturbations [12] and computational overheads of numerical solvers. Furthermore, as highlighted in [12], unconstrained neural ODEs can give rise to chaotic behaviors. The sensitivity to small deviations of the input data features (e.g. *adversarial attacks*) renders these models dangerous in practice: although they may fit training data, generalization to unseen data becomes unreliable due to error propagation.

Imposing stability in discrete neural networks as a first–order design principle has lead to a variety of high performance model variants [13], [14]. Within the neural ODE framework, heuristic approaches to stability [15] have been experimentally shown to improve robustness of neural ODEs. However, these approaches do not provide stability guarantees and require tuning a specific regularization term. In this work, we introduce a *provably* stable neural ODE variant, *stable neural flows*, whose trajectories evolve on monotonically non–increasing level sets of an energy functional parametrised by a neural network. The training task is approached as an optimal control problem, and a recursive adaptation procedure is specifically developed to approximate its solution. The proposed adaptation law is based on a gradient descent procedure covering both temrinal and backpropagated settings. Finally, we introduce *ad hoc* regularizers to ease the optimization process for the proposed model.

Notation: $\mathbb{N}$ and $\mathbb{R}$ ($\mathbb{R}^+$) are the sets of natural and real (positive real) numbers; $\mathcal{C}^n$ is the set of n-times continuously differentiable functions. Moreover, $\dot{x} := \frac{dx}{ds}$ and ∂ denotes the transposed gradient operator $\partial_x := \partial^\top/\partial x$. $\mathbb{E}[\cdot]$ is the *expected value* operator. $\mathbb{0}_n$ is the origin of $\mathbb{R}^n$, $\mathbb{0}_{n\times m}$, $\mathbb{I}_n$ are the n–by–m null matrix and nth order and identity matrix, respectively.

II. Stable Neural Flows

In this section we introduce *stable neural flows* as a stable variant of neural ODEs. In particular, we define a function ε, which we call *energy function*. ε is parametrised by a neural network and steers the flows of the state x be steered along its negative gradient directions. First, we start with some necessary preliminary concepts, which will be followed by the definition of our proposed model class.

A. Preliminaries

Let $(\mathcal{T}, \geq)$ be a linearly ordered set, called the *data time* set (typically $\mathcal{T} = \mathbb{N}$). We suppose to be given an *input-output data stream*, which is a net of the form $\{(u_t, y_t)\}_{t\in\mathcal{T}}$ of input-output pairs $(u_t, y_t) \in \mathbb{R}^{n_u} \times \mathbb{R}^{n_y}$.

Definition 1 (Neural ODE): With $h_u : \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$, $h_y : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ two affine maps, called the *input projection* and *output projection* respectively, a neural ODE is a system of the form

$$\begin{aligned} \dot{x} &= f\left(u_t, x(s), w(s)\right) \\ x(0) &= h_u(u_t) \qquad\qquad s \in \mathcal{S} \\ \hat{y}_t &= h_y(x(S)) \end{aligned} \tag{1}$$

where $\mathcal{S} := [0, S]$ $(S \in \mathbb{R}^+)$ is the *depth domain* and $f : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \times \mathcal{W} \to \mathbb{R}^{n_x}$ is a neural network with weights $w \in \mathcal{W}$, being $\mathcal{W}$ a given pre-specified class of functions $\mathcal{S} \to \mathbb{R}^{n_w}$.

At each $t \in \mathcal{T}$, system (1) takes as input u_t and produces as output its *best estimate* $\hat{y}_t$ of the corresponding output y_t. Our degree of freedom in (1) is the choice of the parameter w inside $\mathcal{W}$. Therefore, our model set is the family of systems of the form (1) obtained as w ranges in $\mathcal{W}$. Note that, in many practical cases, it is convenient to extend the search of the model by "learning" the input and output projections h_u and h_y. This results particularly useful when high–dimensional embeddings of the input u_t are contextually required by the specific problem [16], [12].

Within the scope of this paper we only limit our analysis to *depth–invariant* weights, i.e. we assume that W is the set of constant functions. For ease of notation, we thus identify $\mathcal{W}$ with $\mathbb{R}^{n_w}$, and in the following we do not distinguigh between the two.

Remark 1 (Well–Posedness): If f is Lipschitz, for each u_t the initial value problem in (1) admits a unique solution x defined in the whole $\mathcal{S}$. If this is the case, there is a mapping ϕ from $\mathbb{R}^{n_w} \times \mathbb{R}^{n_u}$ to the space of absolutely continuous functions $\mathcal{S} \mapsto \mathbb{R}^{n_x}$ such that $x_t := \phi(w, u_t)$ satisfies the ODE in (1). This in turn implies that, for all $t \in \mathcal{T}$, the map

$$(w, u_t) \mapsto \gamma(w, u_t) := h_y\big(\phi(w, u_t)(S)\big)$$

satisfies $\hat{y}_t = \gamma(w, u_t)$. For the sake of compactness, we denote $\phi(w, u_t)(s)$ by $\phi_s(w, u_t)$, for any $s \in \mathcal{S}$.

B. Stable Neural Flows

Definition 2 (Stable neural flow): A *stable neural flow* is a variant of (1) having the form

$$\dot{x} = -\partial_x\, \varepsilon(u_t, x(s), w) \quad s \in \mathcal{S}, \forall t \in \mathcal{T} \tag{2}$$

where $\varepsilon : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_w} \to \mathbb{R}$ is a neural network which is $\mathcal{C}^\infty$ and bounded from below.

Under the above assumptions, the solutions of (2) are forward complete for any compact depth domain $\mathcal{S}$ and $t \in \mathcal{T}$, according to Remark 1. Moreover, the explicit dependence of the energy function on the input data u_t allows the model to learn a family of energy functionals *ad hoc* for each data point, rather that a single one. This greatly increases the *expressivity* of the model while reducing the stiffness of the differential equation, allowing it to learn complex nonlinear mappings without introducing additional structure [12].

Stability of the proposed model is assessed by the following:

Proposition 1 (Stability): Every closed set $\mathcal{M} \in \mathbb{R}^{n_x}$ such that

$$\forall x \in \mathcal{M},\ \partial_x\, \varepsilon = \mathbb{0}_{n_x} \text{ and } \partial_x^2\, \varepsilon \succ 0$$

which is contained in an open neighborhood $\mathcal{U} \supset \mathcal{M}$ satisfying

$$\forall x \in \mathcal{U} \setminus \mathcal{M},\ \partial_x\, \varepsilon \neq \mathbb{0}_{n_x}$$

is locally asymptotically stable.

Proof: For all $x \in \mathcal{M}$, $\dot{x} = \mathbb{0}_{n_x}$ and, thus, $\mathcal{M}$ is forward invariant. Let $\mathcal{U} \supset \mathcal{M}$ be an open neighborhood of $\mathcal{M}$ such that $\forall x \in \mathcal{U} \setminus \mathcal{M}$, $\partial_x\, \varepsilon \neq \mathbb{0}_{n_x}$ and let $V(x) = \varepsilon(x)$ be a candidate Lyapunov function. It holds: $\forall x \in \mathcal{U}$ $\dot{V} = -\langle \partial_x V, \partial_x V\rangle \leq 0$ and $\forall x \in \mathcal{M}$ $\dot{V} = 0$. Thus $\mathcal{M}$ is asymptotically stable. ■

Remark 2: From Prop. 1, it follows that every isolated set $\mathcal{M}$ of local minima of ε, has a basin of attraction. Thus, for almost all initial conditions, the system will dissipate all the energy reaching some stable set. In fact, for all initial conditions $h_u(u_t) \in \mathbb{R}^{n_x}$ which are not local maxima of ε nor belong to the basins of attraction of its saddle points, there exists a stable set $\mathcal{M}$ such that $\lim_{s\to\infty} \phi_s(w, u_t) \in \mathcal{M}$.

Note that, even in a compact $\mathcal{S} = [0, S]$ we can seek a suitable set of parameters w to reach an arbitrary level of steady state (i.e. closeness to some invariant $\mathcal{M}$).

C. Variants of the Model

1) Port–Hamiltonian inspired model: By noticing that (2) is formally an autonomous *port–Hamiltonian* system [17], [18], we can generalise the model to

$$\dot{x} = A(x, w_A)\partial_x \varepsilon(u_t, x, w_\varepsilon), \quad w := (w_A, w_\varepsilon) \tag{3}$$

where it is easy to prove that stability, i.e. dissipativity of ε, is preserved whenever A is chosen such that $A(x) + A^\top(x) \prec 0$ for any x. Note that such an A gives more "freedom" to the model by combining different (negative) gradient directions to effectively steer the states towards some energy minima.

2) Second–order model: A special case of stable neural flows modeling is to mimic the classical Hamiltonian dynamics of a mechanical system as follows. Let $x := (q, p) \in \mathbb{R}^{n_x}$, $p, q \in \mathbb{R}^{n_v}$ $(n_v = n_x/2)$. A second–order stable neural flows model can be defined as

$$\begin{aligned} \dot{q} &= p \\ \dot{p} &= -\alpha p - \partial_q \varepsilon(u_t, q(s), w) \end{aligned} \tag{4}$$

where $\alpha \in \mathbb{R}^+$ is a trainable parameter. Within this framework, stability can be proven by defining a total energy function $\varphi(q,p)=\frac{1}{2}p^\top p+\varepsilon(q,w)$. In fact, it holds

$$\begin{aligned}\frac{d}{ds}\varphi(q,p) &= \begin{bmatrix}\partial_q^\top\varphi & \partial_p^\top\varphi\end{bmatrix}\begin{bmatrix}\dot q\\ \dot p\end{bmatrix}\\ &= \partial_q^\top\varepsilon p-\alpha p^\top p-p^\top\partial_q\varepsilon\\ &= -\alpha p^\top p\le 0\quad\forall s\in\mathcal{S}\end{aligned}\tag{5}$$

and (q,p) eventually converges to some fixed point $(q^*,\mathbb{0}_{n_v})$ where q^* is a local minimizer of ε. Also in this case, inspired by port–Hamiltonian models, we can define a stability–preserving generalization of (4) as

$$\begin{bmatrix}\dot q\\ \dot p\end{bmatrix}=\begin{bmatrix}\mathbb{0} & B\\ -B & A\end{bmatrix}\begin{bmatrix}\partial_x\varepsilon\\ p\end{bmatrix}$$

where $A=A^\top\preceq 0$ and $B=B^\top$.

3) Stochastic Model: Motivated by the stochastic variants of neural ODEs [7] and their growing interest within the research community, we define the *stable stochastic neural flows* resembling a drift–diffusion model

$$dx=-\partial_x\varepsilon(x(s),w)ds+\sqrt{2\beta^{-1}}dW(s)\tag{6}$$

where $W(s)$ is a Weiner process. In this context, the energy $\varepsilon(x,w)$ assumes the role of *drift potential* while the diffusion coefficient $\beta\in\mathbb{R}^+$ is a (optimisable) model parameter.

Here, stability can be assessed by recalling the flows of the *joint probability density* $\rho(x,s)$. In fact, $\rho(x,s)$ evolves according to the *Kolmogorov forward* partial differential equation [19],

$$\partial_s\rho=\partial_x\cdot\rho\partial_x\varepsilon+\beta^{-1}\partial_x\cdot\partial_x\rho\tag{7}$$

where "$\cdot$" denotes the inner product. (7) have a unique stationary solution $\rho_{\mathrm{ss}}(x)$ corresponding to the Boltzmann distribution $\rho_{\mathrm{ss}}(x)=\lim_{s\to\infty}\rho(x,s)=\kappa e^{-\beta\varepsilon(x,w)}$, $(\kappa\in\mathbb{R})$ which is reached while the Lyapunov functional $\mathbb{E}_\rho[\varepsilon+\beta^{-1}\log\rho]$ (often called *free energy*) decays along the flow. Besides, optimising the parameters of such models requires extending the *adjoint sensitivity analysis* to a stochastic setting, treated e.g. in [20] and more recently in [8]. The experimental evaluation of this model is therefore out of the scope of the paper and it will be treated in future work.

III. Training The Model

After fixing the model structure, our remaining degree of freedom lies in the choice of the parameters w. In this work, we cast this task in an optimal control setting. In particular, we define a smooth scalar cost function $\ell(w,x,u_t,y_t)$ measuring how well the model fits the data and, consequently, we optimise the parameters to minimise ℓ for all $t\in\mathcal{T}$. This procedure is also referred to as *training* in machine learning terminology. From now on, we denote $\ell(w,x,u_t,y_t)$ as ℓ_t.

A. Training Process: an Optimal Control Perspective

We start by defining the training for standard neural ODEs. In the following, we assume $\mathcal{T}$ to be a finite set. The training process can be then defined as the following constrained nonlinear program

$$\begin{aligned}\min_{w\in\mathbb{R}^{n_w}}\ &\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\ell_t\\ \text{subject to}\ \ &\dot x=f(u_t,x(s),w)\quad s\in[0,S]\\ &x(0)=h_u(u_t)\\ &\hat y_t=h_y(x(S))\\ &\forall t\in\mathcal{T}\end{aligned}\tag{8}$$

While, in general, it is not possible to obtain an analytic solution to (8), as customary in the pertinent literature, we can approximate a locally optimal value of the model's parameters recursively by *gradient descent* (GD) [21].

Let $k\in\mathbb{N}$ be the counter the GD iterations. In case all the input–output data are available offline, the GD solution is obtained by iterating

$$w_{k+1}=w_k-\eta\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\frac{d\ell_t}{dw}\quad(\eta\in\mathbb{R}^+)\tag{9}$$

since $\frac{d}{dw}\sum_{t\in\mathcal{T}}\ell_t=\sum_{t\in\mathcal{T}}\frac{d\ell_t}{dw}$. On the other hand, if the input–output data becomes available sequentially, we have $k=t$ and we implement the *stochastic* version of GD

$$w_{t+1}=w_t-\eta\frac{d\ell_t}{dw}$$

where t is reset to 0 whenever $t=|\mathcal{T}|$. With a sufficiently small value of η, and a sufficiently large number of steps, (stochastic) GD converges arbitrary close to a local minimizer of $\frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\frac{d\ell_t}{dw}$ [21].

Regardless of the choice of ℓ_t, the gradients with respect to the ODE parameters w have to be computed. According to [3], gradients can be computed with $\mathcal{O}(1)$ memory efficiency through adjoint sensitivity analysis [22], [23]. In the next two subsections we derive the gradients for stable neural flows via adjoint sensitivity analysis and the Lagrange multipliers method. We consider two types of cost functions: the *terminal cost*, in which the cost only weights the terminal error, computed for $s=S$ and the *back–propagated cost*, where, instead the cost is distributed on the whole domain $\mathcal{S}$.

B. Terminal Cost Gradients

In this setting, $\ell_t:=g(x(S),y_t)$ where g is a smooth scalar function. The following hold:

Proposition 2 (Terminal cost gradient): Consider a *terminal* cost $\ell_t:=g(x(S),y_t)$. Then,

$$\frac{d\ell_t}{dw}=\mu(0)$$

where $\mu(s)\in\mathbb{R}^{n_w}$ satisfies the initial value problem

$$\begin{aligned}\dot\mu^\top(s)&=-\lambda^\top(s)\frac{\partial f}{\partial w},\quad \mu(S)=\mathbb{0}_{n_w}\\ \dot\lambda^\top(s)&=-\lambda^\top(s)\frac{\partial f}{\partial x},\quad \lambda^\top(S)=\frac{\partial\ell_t}{\partial x(S)}\end{aligned}$$

solved backward in $[0,S]$.

Proof: Let us define a *Lagrange multiplier* $\lambda(s)\in\mathbb{R}^{n_x}$ and let $\mathcal{L}_t$ be a perturbed loss function defined as

$$\mathcal{L}_t:=\ell_t(x(S))-\int_0^S\lambda^\top(\tau)\left[\dot x(\tau)-f(u_t,x(\tau),w)\right]d\tau$$

Since $\dot x-f(x,w)=0$ by construction, the integral term in $\mathcal{L}_t$ is always null and, thus, $\lambda(s)$ can be freely assigned while $\frac{d}{dw}\mathcal{L}_t = \frac{d}{dw}\ell_t$. For the sake of compactness we do not explicitly write the dependence on variables of the considered functions unless strictly necessary. Note that,

$$\int_0^S \lambda^\top \dot{x} d\tau = \lambda^\top(\tau)x(\tau)\big|_0^S - \int_0^S \dot{\lambda}^\top x d\tau \tag{10}$$

Hence,

$$\mathcal{L}_t = \ell_t(x(S)) - \lambda^\top(\tau)x(\tau)\big|_0^S + \int_0^S \left(\dot{\lambda}^\top x + \lambda^\top f\right) d\tau \tag{11}$$

We can compute the gradient of ℓ_t with respect to w as

$$\begin{aligned}\frac{d\ell_t}{dw} = \frac{d\mathcal{L}_t}{dw} &= \frac{\partial \ell_t}{\partial x(S)}\frac{dx(S)}{dw} \\ &- \left(\lambda^\top(S)\frac{dx(S)}{dw} - \lambda^\top(0)\cancel{\frac{dx(0)}{dw}}\right) \\ &+ \int_0^S \left[\dot{\lambda}^\top \frac{dx}{dw} + \lambda^\top\left(\frac{\partial f}{\partial w} + \frac{\partial f}{\partial x}\frac{dx}{dw} + \frac{\partial f}{\partial u_t}\cancel{\frac{du_t}{dw}}\right)\right] d\tau\end{aligned}$$

which, by reorganizing the terms, yields to

$$\begin{aligned}\frac{d\ell_t}{dw} &= \left[\frac{\partial \ell_t}{\partial x(S)} - \lambda^\top(S)\right]\frac{dx(S)}{dw} + \\ &+ \int_0^S \left(\dot{\lambda}^\top + \lambda^\top \frac{\partial f}{\partial x}\right)\frac{dx}{dw} d\tau \\ &+ \int_0^S \lambda^\top \frac{\partial f}{\partial w} d\tau\end{aligned} \tag{12}$$

Now, if $\lambda(s)$ satisfies the initial value problem

$$\dot{\lambda}^\top(s) = -\lambda^\top(s)\frac{\partial f}{\partial x}, \quad \lambda^\top(S) = \frac{\partial \ell_t}{\partial x(S)} \tag{13}$$

to be solved backward in $[0, S]$; (12) reduces to

$$\begin{aligned}\frac{d\ell_t}{dw} &= \int_0^S \lambda^\top \frac{\partial f}{\partial w} d\tau \\ &= -\int_S^0 \left(\frac{\partial \ell_t}{\partial x(S)} - \int_S^\tau \lambda^\top(\zeta)\frac{\partial f}{\partial x} d\zeta\right)\frac{\partial f}{\partial w} d\tau\end{aligned} \tag{14}$$

Note that (14) can be computed by solving backward the system of ODEs

$$\begin{aligned}\dot{\mu}^\top &= -\lambda^\top \frac{\partial f}{\partial w}, \quad \mu(S) = \mathbb{0}_{n_w} \\ \dot{\lambda}^\top &= -\lambda^\top \frac{\partial f}{\partial x}, \quad \lambda^\top(S) = \frac{\partial \ell_t}{\partial x(S)}\end{aligned} \tag{15}$$

Then, $\frac{d\ell}{dw} = \mu(0)$, proving the result. ■

For the stable neural ODE (2), system (15) becomes

$$\dot{\mu}^\top = \lambda^\top \frac{\partial}{\partial w}\frac{\partial \varepsilon}{\partial x}, \quad \dot{\lambda}^\top = \lambda^\top \frac{\partial^2 \varepsilon}{\partial x^2} \tag{16}$$

while, for the second order model (4), it holds

$$\dot{\mu}^\top = \lambda^\top \begin{bmatrix}\mathbb{0}_{n_q \times n_w} \\ -\partial_w \partial_q \varepsilon\end{bmatrix}, \quad \dot{\lambda}^\top = \lambda^\top \begin{bmatrix}\mathbb{0}_{n_q} & \mathbb{I}_{n_q} \\ -\partial_q^2 \varepsilon & -\alpha\mathbb{I}_{n_q}\end{bmatrix}. \tag{17}$$

C. *Back–Propagated Cost Gradients*

We can relax the results of the previous section to integral cost functions of type

$$\ell_t = \int_0^S g(x(\tau), y_t) d\tau$$

In this context, similarly to the terminal cost case, the cost gradient are obtained by the following.

Proposition 3 (Back–propagated cost gradient): Let $\ell_t = \int_0^S g(x(\tau), y_t)d\tau$. It holds,

$$\frac{d\ell_t}{dw} = \mu(0)$$

where μ satisfies the initial value problem

$$\begin{aligned}\dot{\mu}^\top(s) &= -\lambda^\top(s)\frac{\partial f}{\partial w}, & \mu(S) &= \mathbb{0}_{n_w} \\ \dot{\lambda}^\top(s) &= -\lambda^\top(s)\frac{\partial f}{\partial x} - \frac{\partial g}{\partial x}, & \lambda(S) &= \mathbb{0}_{n_x}\end{aligned}$$

solved backward in $[0, S]$.

Proof: Proceeding in parallel to the terminal cost case yields

$$\begin{aligned}\frac{d\ell_t}{dw} &= \int_0^S \frac{\partial g}{\partial x}\frac{dx}{dw} d\tau \\ &- \lambda^\top(S)\frac{dx(S)}{dw} \\ &+ \int_0^S \left[\dot{\lambda}^\top \frac{dx}{dw} + \lambda^\top\left(\frac{\partial f}{\partial w} + \frac{\partial f}{\partial x}\frac{dx}{dw}\right)\right] d\tau\end{aligned} \tag{18}$$

which leads to

$$\begin{aligned}\frac{d\ell_t}{dw} &= -\lambda^\top(S)\frac{dx(S)}{dw} \\ &+ \int_0^S \left(\dot{\lambda}^\top + \lambda^\top \frac{\partial f}{\partial x} + \frac{\partial g}{\partial x}\right)\frac{dx}{dw} d\tau \\ &+ \int_0^S \lambda^\top \frac{\partial f}{\partial w} d\tau\end{aligned} \tag{19}$$

Therefore, if $\lambda(s)$ satisfies

$$\dot{\lambda}^\top = \lambda^\top \frac{\partial f}{\partial x} - \frac{\partial g}{\partial x}, \quad \lambda(S) = \mathbb{0}_{n_x} \tag{20}$$

we obtain:

$$\begin{aligned}\frac{d\ell_t}{dw} &= \int_0^S \lambda^\top \frac{\partial f}{\partial w} d\tau \\ &= -\int_S^0 \left[\int_S^\tau \left(\lambda^\top(\zeta)\frac{\partial f}{\partial x} + \frac{\partial g}{\partial x}\right) d\zeta\right]\frac{\partial f}{\partial w} d\tau\end{aligned} \tag{21}$$

Again, $d\ell/dw$ can be recovered as $\mu(0)$ by solving backward the ODE

$$\begin{aligned}\dot{\mu}^\top(s) &= -\lambda^\top(s)\frac{\partial f}{\partial w}, & \mu(S) &= \mathbb{0}_{n_w} \\ \dot{\lambda}^\top(s) &= -\lambda^\top(s)\frac{\partial f}{\partial x} - \frac{\partial g}{\partial x}, & \lambda(S) &= \mathbb{0}_{n_x}\end{aligned} \tag{22}$$

■

D. *Gradients with Respect to Integration Bound*

It is also possible to optimise for the integration bound[1] S with GD iterates as in (9) alongside w. In the case of the terminal cost loss, we have

$$\begin{aligned}\frac{d\ell_t}{dS} &= \frac{\partial \ell_t}{\partial x(S)}\frac{dx(S)}{dS} = \frac{\partial \ell_t}{\partial x(S)}\frac{d}{dS}\int_0^S f(x, w)d\tau \\ &= \frac{\partial \ell_t}{\partial x(S)} f(x(S), w),\end{aligned} \tag{23}$$

where the Leibniz integral rule has been used in the last equality. Hence, for stable neural ODEs the cost gradient is directly correlated to the gradient of the energy stored in the

[1]Which can be interpreted, from an optimal control point of view, as the horizon of the control problem.

system at the end of the integration, i.e.

$$\frac{d\ell_t}{dS} = -\frac{\partial \ell_t}{\partial x(S)}\partial_x \varepsilon(x(S), w) \quad (24)$$

In the case of back–propagated cost, instead, we have

$$\frac{\partial \ell_t}{\partial S} = g(x(S)). \quad (25)$$

E. Gradients with respect to input and output projections

In case h_u and h_y depend on two sets of parameters v_u and v_y, which we want to optimise with GD alongside w, we need to compute their respective gradients. In the terminal cost case, it holds that

$$\ell_t = g(x(s)) = g\left[h_y\left(h_u(u_t) + \int_0^S f(x(\tau))f\tau\right)\right]$$

and, therefore,

$$\frac{d\ell_t}{dv_u} = \frac{\partial g}{\partial h_y}\frac{\partial h_y}{\partial h_u}\frac{\partial h_u}{\partial v_u}, \quad \frac{d\ell_t}{dv_y} = \frac{\partial g}{\partial h_y}\frac{\partial h_y}{\partial v_y}$$

Moreover, in the case of back–propagated cost, the gradients with respect to v_u can be computed as

$$\begin{aligned}\frac{d\ell_t}{dv_u} &= \frac{d}{dv_u}\int_0^S g(x(\tau))d\tau \\ &= \int_0^S \frac{d}{dv_u} g\left[h_u(u_t, v_u) + \int_0^\tau f(u_t, x(\zeta), w)d\zeta\right]d\tau \\ &= \int_0^S \frac{\partial g}{\partial h_u}\frac{\partial h_u}{\partial v_u}d\tau = \int_0^S \frac{\partial g}{\partial h_u}d\tau\frac{\partial h_u}{\partial v_u}\end{aligned}$$

thus resulting in $\frac{d\ell_t}{dv_u} = \chi(0)\frac{\partial h_u}{\partial v_u}$, where $\chi(s)$ satisfies

$$\dot{\chi}^\top(s) = -\frac{\partial g}{\partial h_u}, \ \chi^\top(S) = \mathbb{0}_{n_x}$$

F. On Training Regularisers

While we enforce stability to stable neural flows by structuring the vector field as the negative gradient of a bounded energy functional, we have no guarantees that the system approaches steady–steady at the end of a bounded depth domain. Therefore, it would be beneficial to introduce suitable soft constraints as additive term to the cost ℓ_t to accelerate convergence to steady–state. To this end, we can augment the loss with a terminal cost term proposed by [12]:

$$\begin{aligned}\ell_t^\star &= \ell_t + \frac{\gamma}{2}\|f(u_t, x(S), w)\|_2^2 \\ &= \ell_t + \frac{\gamma}{2}\|\partial_x\varepsilon(u_t, x(S), w)\|_2^2\end{aligned} \quad (26)$$

with $\gamma \in \mathbb{R}^+$. Note that this regularization term may be also successfully used in standard *vanilla* models to encourage regularity and stability of the flows [12].

IV. EXPERIMENTAL EVALUATION

In all the following experiments, the ODEs have been solved with a Dormand–Prince adaptive–step solver and absolute and relative tolerances set to 10^{-6} while the scalar γ for the regularized loss (26) has been set to 10^{-2}.

A. Function approximation

We consider the approximation of the function $y = -u$ ($u \in \mathbb{R}$). While standard, neural ODEs [3] cannot tackle this simple problem [16], [12], we show how a stable model dependent on u_t is able to learn a proper family of energy functions whose steepest direction is followed. In this case, ε was chosen as multi–layer perceptron taking x, u_t ad inputs with layers $2, 16, 16, 1$ and *hyperbolic tangent* (tanh) activation at each layer but the last one. The output is then squared to enforce lower boundedness. The training was performed by uniformly sampling $u_t \in \mathcal{U} := [-1, 1]$ and computing $y_t = -u_t$. Figure 2 shows how the state set $\mathcal{U}$ is adjusted by the model to lie on a lower energy configuration of the energy functional ϵ. This minimum energy configuration is shaped by f during optimization to correspond to the desired function $y = -u$. Figure 3 offers a different perspective on the same task and highlights the ability of f to learn crossing flows.

B. Nonlinear classification via stable neural flows

We tested several models with different two-dimensional nonlinear classification tasks, including the *half–moons* and *three spirals* datasets. The objective of each neural ODE model tested was then steering the input points of different classes towards linearly separable manifolds, so that the linear output projection map could minimize the chosen cost function. For each problem, we sampled the classes in the input space and added Gaussian noise. The *output projection* map $h_y : \mathbb{R}^{n_x} \to \mathbb{R}^{n_y}$ was chosen as a linear affine map $\hat{y}_t = h_y(x) := W_y^\top x + b_y$ where $v_y := \text{vec}(W_y, b_y) \in \mathbb{R}^{(n_x+1)n_y}$ has been optimized together with the models' parameters w. In both cases ε was parametrised by a multilayer perceptron with two hidden layer of 32 units each and tanh activation. The scalar output was then filtered by a sigmoid function to enforce lower boundedness of ε.

We evaluate different variants of the proposed method. For half–moons, we employ a stable neural flow in port–Hamiltonian form (3) with $A := -\text{diag}(|a_1|, |a_2|)$, $w_A := (a_1, a_2)$ but with no explicit dependence of ε from u_t (data–independent). The model is equipped with an $\mathbb{R}^2 \to \mathbb{R}^2$ *input projection* map $h_u(x) := W_u^\top u_t + b_u$, with parameters, $v_u := \text{vec}(W_u, b_u) \in \mathbb{R}^6$ included in the optimization procedure. We compute the quadratic cost of outputs $\hat{y}_t$ and corresponding labels $y_t \in \{0, 1\}$. The three spirals experiments, on the other hand, involve the use of a data–dependent stable neural flow (2) optimized to minimize the *cross-entropy* [24] cost of $\hat{y}_t$ and *one–hot* encoded labels $y_t \in \mathbb{R}^3$. Figures 4 and 5 show that the model correctly steers input–data towards such linearly separable clusters, indicated in black. The depth–flows converge to the desired values before $S := 1$, confirming stability.

V. CONCLUSION AND FUTURE WORK

We enhance *neural ordinary differential equations* with intrinsic stability properties. Key to *stable neural ODEs* is replacing an unconstrained vector field with an energy manifold whose gradient drives the data–flows. Furthermore, the proposed model class is augmented with *ad hoc* regularizers designed to accelerate convergence to steady–state. Distilling an energy representation of the input manifold provides provides a robust approach to continuous–depth

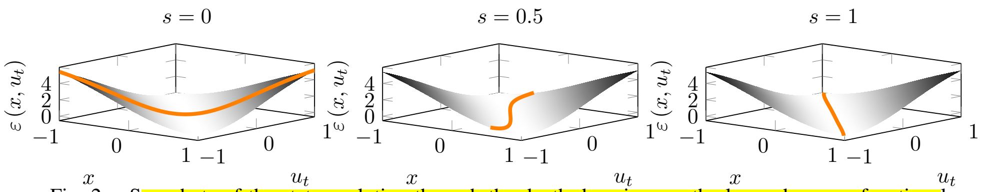

Fig. 2. Snapshots of the state evolution through the depth domain over the learned energy functional.

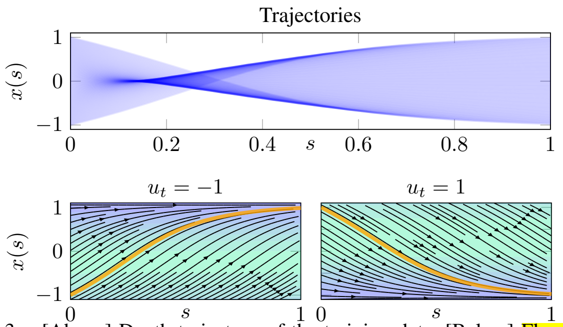

Fig. 3. [Above] Depth trajectory of the training data. [Below] Flows over the data–dependent learned vector field for $u_t = -1$ and $u_t = 1$

deep learning and naturally paves the way to future work involving observers and controllers based on neural ODEs.

REFERENCES

[1] Sho Sonoda and Noboru Murata. Double continuum limit of deep neural networks. In *ICML Workshop Principled Approaches to Deep Learning*, 2017.

[2] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

[3] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[4] Sho Sonoda and Noboru Murata. Transport analysis of infinitely deep neural network. *The Journal of Machine Learning Research*, 20(1):31–82, 2019.

[5] Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

[6] Cagatay Yildiz, Markus Heinonen, and Harri Lahdesmaki. Ode2vae: Deep generative second order odes with bayesian neural networks. In *Advances in Neural Information Processing Systems*, pages 13412–13421, 2019.

[7] Junteng Jia and Austin R Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9843–9854, 2019.

[8] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020.

[9] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*, 2019.

[10] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems*, pages 15353–15363, 2019.

[11] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

[12] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes. *arXiv preprint arXiv:2002.08071*, 2020.

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

[14] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.

[15] YAN Hanshu, DU Jiawei, TAN Vincent, and FENG Jiashi. On robustness of neural ordinary differential equations. In *International Conference on Learning Representations*, 2019.

[16] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

[17] 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.

[18] S. Massaroli, M. Poli, F. Califano, A. Faragasso, J. Park, A. Yamashita, and H. Asama. Porthamiltonian approach to neural network training. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 6799–6806, Dec 2019.

[19] Kenneth Caluya and Abhishek Halder. Gradient flow algorithms for density propagation in stochastic systems. *IEEE Transactions on Automatic Control*, 2019.

[20] Robert J Elliott and Michael Kohlmann. The adjoint process in stochastic optimal control. In *Stochastic Differential Systems*, pages 115–127. Springer, 1989.

[21] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, 2019.

[22] Lev Semenovich Pontryagin, EF Mishchenko, VG Boltyanskii, and RV Gamkrelidze. The mathematical theory of optimal processes. 1962.

[23] Yang Cao, Shengtai Li, and Linda Petzold. Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software. *Journal of computational and applied mathematics*, 149(1):171–191, 2002.

[24] John E Shore and Robert M Gray. Minimum cross-entropy pattern classification and cluster analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (1):11–17, 1982.

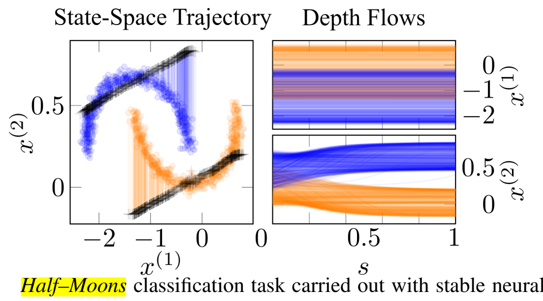

Fig. 4. *Half–Moons* classification task carried out with stable neural flows where ε is u_t–independent.

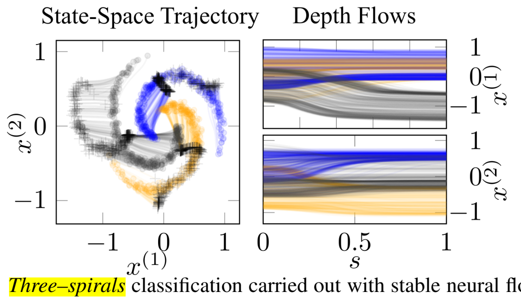

Fig. 5. *Three–spirals* classification carried out with stable neural flows and *data-dependent* energy function ε.