Structure preserving neural networks based on ODEs

Davide Murari[*] Elena Celledoni [*] Brynjulf Owren[*] Carola-Bibiane Schönlieb[†]
Ferdia Sherry[†]

Neural networks have gained popularity due to their ability to provide accurate solutions in various applications. Generally, the problems in which they are effective can be phrased as approximation tasks in high-dimensional spaces. This success is often based on experimental evidence, and more theoretical insight is needed. Recently, many authors have considered the design of deep learning architectures that satisfy certain mathematical properties such as stability, symplecticity, volume preservation, or constraints on the Lipschitz constant, [17, 14, 9, 30, 8, 10, 32, 15, 34, 36]. The networks presented in these articles are often quite specific since they have been derived ad hoc, typically ensuring only one particular property and without following a more general procedure.

There are multiple situations where one could be interested in networks with some prescribed structure. Referring to $F : \mathcal{X} \to \mathcal{Y}$ as the function approximated by the network, we report here three cases:

1. when F has some known property that is important for the application of interest,
2. when the geometry of $\mathcal{X}$ and $\mathcal{Y}$ prescribes a particular structure for F,
3. when we can approximate F sufficiently accurately with functions in $\mathcal{G}$, a family of functions with a property that makes them well suited to design the network layers.

This manuscript describes a general and systematic way to impose desired mathematical structures on neural networks. We remark that the interest in preserving the specific structure of a function can be found in the field of geometric numerical integration, [13, 18, 24, 16]. For this reason, the approach we introduce is based on defining the network layers as a (structure-preserving) discretisation of the solutions of a specific differential equation. To illustrate the methodology, we present the derivation of three structured neural networks. Each of them represents one of the three situations reported above.

There have been multiple attempts to formulate unifying principles for designing neural networks. We mention in particular the continuous-in-time interpretation of residual neural networks (see, e.g., [35, 22, 5, 29, 2]) on which our work builds upon. Residual neural networks (ResNets) are compositions of parametric maps that are either linear or of the form

$$\mathbb{R}^{n_i} \ni x \mapsto f_{\theta_i}(x) = x + h\Lambda(\theta_i, x) \in \mathbb{R}^{n_{i+1}}. \tag{1}$$

A typical expression for Λ is $\Lambda(\theta_i, x) = \Sigma(A_i x + b_i)$ with $\Sigma(z) = [\sigma(z_1), ..., \sigma(z_n)]$ and $\sigma : \mathbb{R} \to \mathbb{R}$. The continuous-in-time interpretation of ResNets arises noticing that if $n_i = n_{i+1}$, f_{θ_i} coincides with one $h-$step of the explicit Euler method applied to $\dot{x}(t) = \Lambda(\theta(t), x(t))$. In this work, we consider time-switching systems of the form

$$\dot{x}(t) = \Lambda(\theta(t), x(t)) = f_{s(t)}(x(t)),\ s : [0, T] \to \{1, ..., N\},\ f_i \in \mathcal{F}, \tag{2}$$

with s being piecewise constant (see, e.g., [28, 19]), and $\mathcal{F}$ a family of parametric vector fields. More concretely, we consider vector fields with weight functions that are piecewise constant in time. We introduce time-switching dynamical systems in more detail in Appendix A.

[*]Department of Mathematical Sciences, NTNU, N-7491 Trondheim, Norway (davide.murari@ntnu.no,elena.celledoni@ntnu.no, brynjulf.owren@ntnu.no)

[†]Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Wilberforce Road, Cambridge CB3 0WA, UK. (cbs31@cam.ac.uk, fs436@cam.ac.uk)

DLDE Workshop in the 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

This interpretation of ResNets gives the foundation of the approach we present. Indeed, we propose to modify two steps in the continuous-in-time interpretation of ResNets. We focus on properties of functions preserved under composition, such as being $1-$Lipschitz or volume-preserving. Let us call $\mathcal{P}$ the property of interest. Then our strategy to define a neural network satisfying $\mathcal{P}$ consists in

1. finding a family $\mathcal{F}$ of parametric vector fields having flow maps that satisfy $\mathcal{P}$,
2. replacing the explicit Euler method in (1) with suitable numerical methods that reproduce $\mathcal{P}$ also at the discrete level.

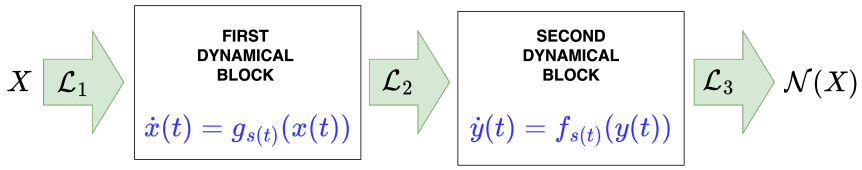

Figure 1: ResNet with two dynamical blocks and three linear layers. $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are linear maps.

As with classical ResNets, we also introduce linear layers in the network, see Figure 1. For this reason, we introduce the name "dynamical blocks" to refer only to the groups of layers obtained from an ODE. We remark that one can still get networks with the desired structure by constraining the linear layers to preserve the desired property $\mathcal{P}$.

The key reason that motivates our approach is that it is often easier to impose a property on a vector field than on its flow map. The additional advantage of this strategy is that the research areas of dynamical systems and numerical analysis are well developed. One can thus exploit this knowledge to design neural networks that reproduce the desired structure. Moreover, this approach enables us to derive new structured networks systematically and collocate other existing architectures into a more general setting, making their analysis easier (see [7]).

1 Volume-preserving neural networks

We start with the first case of interest. We focus, in particular, on volume-preserving functions. For this reason, we do not allow for dimensionality changes throughout the network. In this case, defining the approximating network based on switching systems and splitting methods can be a flexible strategy. Consider the switching system defined by $\dot{z}(t) = f_{s(t)}(z(t)) \in \mathbb{R}^n$ where for every value $s(t) = i$, the vector field f_i has a specific partitioning that makes it divergence-free. For example, if we have $n = 2m$,

$$f_{s(t)}(z) = \begin{bmatrix} u_{s(t)}(z[m:]) \\ v_{s(t)}(z[:m]) \end{bmatrix}, \quad u_i, v_j : \mathbb{R}^m \to \mathbb{R}^m$$

satisfies such a condition, and its flow map will be volume-preserving. We can numerically integrate such a vector field while preserving this property. Indeed, we can compose the exact flow maps of the two volume-preserving vector fields

$$f^1_{s(t)}(z) = \begin{bmatrix} u_{s(t)}(z[m:]) \\ 0 \end{bmatrix}, \quad f^2_{s(t)}(z) = \begin{bmatrix} 0 \\ v_{s(t)}(z[:m]) \end{bmatrix}, \quad f_{s(t)} = f^1_{s(t)} + f^2_{s(t)},$$

i.e. we can apply a splitting method to approximate the solutions (see [24]). This approach gives architectures that are close to the ones of RevNets (see, e.g., [10]). A particular class of these blocks can be obtained with second-order ODEs as $\ddot{x}(t) = f_{s(t)}(x(t))$ and, in particular, with second-order conservative ODEs like $\ddot{x}(t) = \nabla V_{s(t)}(x(t))$. The vector fields we reported in this section are quite general in order to illustrate the freedom one has in the design of the networks. However, we propose a specific choice of vector field in Appendix B for a better understanding.

2 Mass-preserving neural networks

Some function approximation problems deal with data having a particular structure. As an example, we now focus on problems where the target function $F : \mathbb{R}^n \to \mathbb{R}^m$ is known to satisfy

$T_n x := \sum_{i=1}^n x_i = T_m F(x) := \sum_{i=1}^m F(x)_i$. We refer to functions of this type as mass preserving functions (see [3]). The neural network approximating F should also be mass preserving to generate interpretable outputs.

We interpret the network layers as discrete flow maps of specific vector fields. A vector field $X \in \mathfrak{X}(\mathbb{R}^n)$ whose flow map preserves the sum of the components of the state vector is one having a linear first integral $g(x) = \mathbf{1}^T x = \sum_{i=1}^n x_i, \quad \mathbf{1} = [1, ..., 1]^T \in \mathbb{R}^n$. Any vector field X having g as a conserved quantity can be written in the form $X(x) = S(x)\mathbf{1}$ for a skew-symmetric matrix-valued function $S : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ (see [25]). Thus, we can model the network as a suitable time discretisation of an ODE of the form

$$\dot{y}(t) = (\hat{A}(y) - \hat{A}(y)^T)\mathbf{1}, \ \hat{A} : \mathbb{R}^n \to \mathbb{R}^{n\times n}. \tag{3}$$

The matrix function $\hat{A}$ is upper triangular, and to model it we reshape it into the parametric vector-function $A(x) = W^T\Sigma(Vx + v) \in \mathbb{R}^{n(n-1)/2}$. Furthermore, as presented in [12, Chapter 4], every Runge–Kutta method preserves linear first integrals. Thus, one can model mass preserving neural networks by composing layers of the following types:

1. Lifting layers: $L : \mathbb{R}^k \to \mathbb{R}^{k+s}$, $L(x_1, ..., x_k) = (x_1, ..., x_k, 0, ..., 0)$,
2. Projection layers: $P : \mathbb{R}^{k+s} \to \mathbb{R}^k$, $P(x_1, ..., x_k, x_{k+1}, ..., x_{k+s}) = (x_1 + o, ..., x_k + o)$ with $o = \sum_{i=1}^s x_{k+i}/s$,
3. Dynamical blocks: one-step explicit Euler discretisations of (3).

A more explicit presentation of these dynamical blocks can be found in Appendix B. To test the neural network architecture, we approximate the flow map of the ODE

$$\dot{y} = \begin{bmatrix} 0 & -y_3y_1^2 & y_2y_3 \\ y_3y_1^2 & 0 & -\sin y_1 \\ -y_2y_3 & \sin y_1 & 0 \end{bmatrix} \mathbf{1} = X(y). \tag{4}$$

We approximate the time$-\Delta t$ flow map of such a system, with $\Delta t = 1/40$. The network is trained in a recurrent manner (see, e.g., [8, 6]). As training data we use tuples of the form $\{(x_i = y_i^0, y_i^1, ..., y_i^M)\}_{i=1,...,N}$, where $y_i^{j+1} = \Psi^{\Delta t}(y_i^j)$ is an accurate approximation of the exact time-Δt flow of X. Figure 2 compares the results obtained with an accurate approximation of the true solution of (4) to those coming from the mass-preserving neural network.

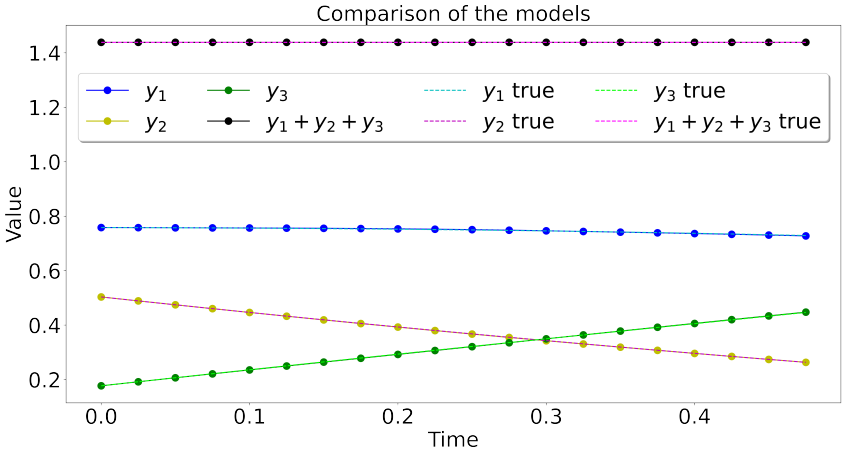

Figure 2: One test trajectory up to time $T = 0.5$. We report the sum of the components, which is also preserved by the network. The components of the vector (y_1, y_2, y_3) are the network predictions.

3 Lipschitz constrained neural networks

We now focus on the problem of classifying points of a set $\mathcal{X} \subset \mathbb{R}^n$ into its C classes. This problem consists hence in approximating the function $\ell : \mathcal{X} \to \{1, ..., C\}$ that assigns all the points of $\mathcal{X}$ to

their correct classes. Neural networks can provide accurate solutions to this problem, but they can be very sensitive to suitable input perturbations called adversarial attacks (see [31, 23, 4, 11]). In [33], the authors characterise this sensitivity problem mathematically. Calling $\mathcal{N}$ the approximating network and $\mathcal{M}_{\mathcal{N}}(x) = \mathcal{N}(x)^T e_{\ell(x)} - \max_{j \neq \ell(x)} \mathcal{N}(x)^T e_j$ its margin at $x \in \mathcal{X}$, they show that $\mathcal{N}$ classifies as x the points in $B_\varepsilon(x) = \{y \in \mathbb{R}^n : \|y - x\|_2 < \varepsilon\}$ if $\mathcal{M}_{\mathcal{N}}(x) \geq \sqrt{2}\varepsilon \operatorname{Lip}(\mathcal{N})$[3]. Motivated by this result, we present a strategy to constrain the Lipschitz constant of ResNets to the value of 1. To show the flexibility of the proposed framework and obtain expressive networks, we impose such property without relying only on $1-$Lipschitz layers, differently from [5, 36, 26].

Consider the two families of vector fields

$$\mathcal{F}_c = \{-A^T \Sigma(Ax + b) \in \mathbb{R}^n : A^T A = I_n\}, \quad \sigma(s) = \max\{x, x/2\}, \quad \text{and}$$
$$\mathcal{F} = \{X \in \mathfrak{X}(\mathbb{R}^n) : \operatorname{Lip}(X) \leq 1\}.$$

Here we denote with $I_n \in \mathbb{R}^{n \times n}$ the identity matrix. We remark that $-A^T \Sigma(Ax + b) = -\nabla_x V(x, A, b)$ for $V(x, A, b) = \mathbf{1}^T \Gamma(Ax + b)$ that is strongly-convex in the x variable. Here we denote with $\mathbf{1} = [1, ..., 1] \in \mathbb{R}^n$, and $\Gamma(z) = [\gamma(z_1), ..., \gamma(z_n)]$ with $\gamma' = \sigma$. The strong-convexity of V is fundamental in our derivation.

With suitable step restrictions, the composition of the Euler steps $\Psi_1^{h_1}(x) = x - h_1 A^T \Sigma(Ax + b)$ and $\Psi_2^{h_2}(x) = x + h_2 X(x)$, $X \in \mathcal{F}$, can be made 1-Lipschitz. Indeed, by direct computation, we get

$$\operatorname{Lip}(\Psi_1^{h_1} \circ \Psi_2^{h_2}) \leq \operatorname{Lip}(\Psi_1^{h_1}) \cdot \operatorname{Lip}(\Psi_2^{h_2}) \leq \sqrt{1 - h_1 + h_1^2}(1 + h_2). \tag{5}$$

Thus, the composition $\Psi_2^{h_2} \circ \Psi_1^{h_1}$ is $1-$Lipschitz if the step sizes are properly chosen. The derivation of (5) is presented in detail in Appendix C. Repeating a similar reasoning for $\Psi_2^{h_2/2} \circ \Psi_1^{h_1/2} \circ \Psi_2^{h_2/2} \circ \Psi_1^{h_1/2}$, one can obtain weaker restrictions on the step sizes. This is the solution we adopt for the following experiment.

We now apply this reasoning to design a neural network with increased robustness to L^2-PGD adversarial attacks on the CIFAR-10 dataset. Instead of considering the generic $\mathcal{F}$, we work with the subfamily $\mathcal{F}_e = \{\Sigma(Bx + c) : B \in \mathbb{R}^{n \times n}, \|B\|_2 \leq 1, c \in \mathbb{R}^n\}$. The step sizes are constrained to satisfy the $1-$Lipschitz constraint derived above. Weight orthogonality comes from the regularisers proposed in [34]. The L^2-PGD adversarial attacks are generated with Foolbox [27]. We compare the results with a baseline ResNet characterised by updates of the form $x \mapsto x + B\Sigma(Ax + b)$, and hence having approximately double the number of weights. Table 1 highlights that for Lipschitz constrained ResNets, the accuracy obtained for highly perturbed images is considerably better than the one with the baseline ResNet. The lower accuracy for clean images is due to the constraints we impose on the network and the reduced number of weights, making it harder to train. On the other hand, other alternation strategies and constraining techniques might lead to better results on this side.

Type	$\varepsilon = 0.0$	$\varepsilon = 0.06$	$\varepsilon = 0.14$	$\varepsilon = 0.3$	$\varepsilon = 0.5$	$\varepsilon = 1.0$
Lipschitz, Margin = 0.07	0.808	0.790	0.775	0.739	0.678	0.516
Lipschitz, Margin = 0.15	0.796	0.782	0.761	0.718	0.657	0.505
Baseline, Margin = 0.07	0.896	0.870	0.816	0.670	0.461	0.114
Baseline, Margin = 0.15	0.903	0.856	0.790	0.642	0.424	0.073

Table 1: We denote with ε the magnitude of the adversarial perturbations. With 'Lipschitz' we refer to Lipschitz constrained networks. All the networks have been trained with hinge loss function [1] for different margin values.

[3]Here we denote with $\|\cdot\|_2$ the Euclidean norm of $\mathbb{R}^n$, with $e_i = [0, ..., 0, 1, 0, ..., 0] \in \mathbb{R}^n$ the $i - th$ canonical basis vector and $\operatorname{Lip}(\mathcal{N})$ is the Lipschitz constant of $\mathcal{N}$.

4 Conclusion

In this work, we have introduced a framework to combine the design of ODE models with the choice of proper numerical methods to obtain neural networks with prescribed properties. We illustrate the procedure deriving three families of structured networks. Some numerical experiments also support their practicality.

References

[1] *Hinge Loss Pytorch*. `https://pytorch.org/docs/stable/generated/torch.nn.MultiMarginLoss.html`.

[2] A. AGRACHEV AND A. SARYCHEV, *Control on the manifolds of mappings with a view to the deep learning*, Journal of Dynamical and Control Systems, (2021), pp. 1–20.

[3] S. BLANES, A. ISERLES, AND S. MACNAMARA, *Positivity-preserving methods for population models*, arXiv preprint arXiv:2102.08242, (2021).

[4] N. CARLINI AND D. WAGNER, *Towards evaluating the robustness of neural networks*, in 2017 IEEE Symposium on Security and Privacy, IEEE, 2017, pp. 39–57.

[5] E. CELLEDONI, M. J. EHRHARDT, C. ETMANN, R. I. MCLACHLAN, B. OWREN, C.-B. SCHÖNLIEB, AND F. SHERRY, *Structure-preserving deep learning*, European Journal of Applied Mathematics, 32 (2021), pp. 888–936.

[6] E. CELLEDONI, A. LEONE, D. MURARI, AND B. OWREN, *Learning Hamiltonians of constrained mechanical systems*, Journal of Computational and Applied Mathematics, 417 (2023), p. 114608.

[7] E. CELLEDONI, D. MURARI, B. OWREN, C.-B. SCHÖNLIEB, AND F. SHERRY, *Dynamical systems' based neural networks*, arXiv preprint arXiv:2210.02373, (2022).

[8] Z. CHEN, J. ZHANG, M. ARJOVSKY, AND L. BOTTOU, *Symplectic Recurrent Neural Networks*, in International Conference on Learning Representations, 2020.

[9] C. L. GALIMBERTI, L. FURIERI, L. XU, AND G. FERRARI-TRECATE, *Hamiltonian deep neural networks guaranteeing non-vanishing gradients by design*, arXiv preprint arXiv:2105.13205, (2021).

[10] A. N. GOMEZ, M. REN, R. URTASUN, AND R. B. GROSSE, *The reversible residual network: Backpropagation without storing activations*, Advances in Neural Information Processing Systems, 30 (2017).

[11] I. J. GOODFELLOW, J. SHLENS, AND C. SZEGEDY, *Explaining and harnessing adversarial examples*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015.

[12] E. HAIER, C. LUBICH, AND G. WANNER, *Geometric Numerical integration: structure-preserving algorithms for ordinary differential equations*, Springer, 2006.

[13] E. HAIRER, M. HOCHBRUCK, A. ISERLES, AND C. LUBICH, *Geometric numerical integration*, Oberwolfach Reports, 3 (2006), pp. 805–882.

[14] J. HERTRICH, S. NEUMAYER, AND G. STEIDL, *Convolutional proximal neural networks and plug-and-play algorithms*, Linear Algebra and its Applications, 631 (2021), pp. 203–234.

[15] P. JIN, Z. ZHANG, A. ZHU, Y. TANG, AND G. E. KARNIADAKIS, *SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems*, Neural Networks, 132 (2020), pp. 166–179.

[16] F. KANG AND S. ZAI-JIU, *Volume-preserving algorithms for source-free dynamical systems*, Numerische Mathematik, 71 (1995), pp. 451–463.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, *Backpropagation applied to handwritten zip code recognition*, Neural computation, 1 (1989), pp. 541–551.

[18] B. Leimkuhler and S. Reich, *Simulating Hamiltonian dynamics*, no. 14, Cambridge University Press, 2004.

[19] D. Liberzon, *Switching in systems and control*, vol. 190, Springer, 2003.

[20] ——, *Switched systems*, in Handbook of networked and embedded control systems, Springer, 2005, pp. 559–574.

[21] D. Liberzon and A. S. Morse, *Basic problems in stability and design of switched systems*, IEEE Control Systems Magazine, 19 (1999), pp. 59–70.

[22] Y. Lu, A. Zhong, Q. Li, and B. Dong, *Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations*, in International Conference on Machine Learning, PMLR, 2018, pp. 3276–3285.

[23] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, *Towards Deep Learning Models Resistant to Adversarial Attacks*, in International Conference on Learning Representations, 2018.

[24] R. I. McLachlan and G. R. W. Quispel, *Splitting methods*, Acta Numerica, 11 (2002), pp. 341–434.

[25] R. I. McLachlan, G. R. W. Quispel, and N. Robidoux, *Geometric integration using discrete gradients*, Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 357 (1999), pp. 1021–1045.

[26] L. Meunier, B. Delattre, A. Araujo, and A. Allauzen, *Scalable Lipschitz Residual Networks with Convex Potential Flows*, arXiv preprint arXiv:2110.12690, (2021).

[27] J. Rauber, W. Brendel, and M. Bethge, *Foolbox: A Python toolbox to benchmark the robustness of machine learning models*, arXiv preprint arXiv:1707.04131, (2017).

[28] D. Ruiz-Balet and E. Zuazua, *Neural ODE control for classification, approximation and transport*, arXiv preprint arXiv:2104.05278, (2021).

[29] L. Ruthotto and E. Haber, *Deep neural networks motivated by partial differential equations*, Journal of Mathematical Imaging and Vision, 62 (2020), pp. 352–364.

[30] B. Smets, J. Portegies, E. J. Bekkers, and R. Duits, *PDE-based group equivariant convolutional neural networks*, Journal of Mathematical Imaging and Vision, (2022), pp. 1–31.

[31] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, *Intriguing properties of neural networks*, CoRR, abs/1312.6199 (2014).

[32] A. Trockman and J. Z. Kolter, *Orthogonalizing Convolutional Layers with the Cayley Transform*, in International Conference on Learning Representations, 2021.

[33] Y. Tsuzuku, I. Sato, and M. Sugiyama, *Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural network*, Advances in Neural Information Processing Systems, 31 (2018).

[34] J. Wang, Y. Chen, R. Chakraborty, and S. X. Yu, *Orthogonal convolutional neural networks*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 11505–11515.

[35] E. Weinan, *A proposal on machine learning via dynamical systems*, Communications in Mathematics and Statistics, 1 (2017), pp. 1–11.

[36] M. Zakwan, L. Xu, and G. Ferrari-Trecate, *On Robust Classification using Contractive Hamiltonian Neural ODEs*, arXiv preprint arXiv:2203.11805, (2022).

[37] M. Zelikin, A. A. Agrachev, Y. Sachkov, and Y. L. Sachkov, *Control theory from the geometric viewpoint*, vol. 2, Springer Science & Business Media, 2004.

A Time-switching systems

In this appendix, we briefly introduce the notion of time-switching systems that has an essential structural role in the framework we present. A switching system is a dynamical system that belongs to the class of hybrid systems, i.e. systems having both a discrete-time behaviour and a continuous one. There is a broad literature about them, see e.g. [19],[21], but not all the theory is necessary for our discussion. More precisely, we work with time-switching systems. Their dynamics can be characterised by ODEs of the form

$$\dot{z}(t) = f_{s(t)}(z(t)) \in \mathbb{R}^n,$$

where $s : \mathbb{R}^+ \to \{1, ..., N\}$ is a piecewise constant time-switching signal. This means that for each value $s(\bar{t})$ the dynamics is ruled by the vector field $f_i \in \mathfrak{X}(\mathbb{R}^n)$, where $i = s(\bar{t})$. This switching signal defines how the dynamics switches from one vector field to another. Usually, the vector fields among which the switching occurs have similar expressions, which is also the case of how we employ them to define neural networks. In the language typical to geometric control theory ([37]), these systems are called affine control systems, and they are expressed as

$$\dot{z}(t) = \sum_{i=1}^{N} \chi_i(t) f_i(z(t)),$$

with

$$\chi_i(t) := \begin{cases} 1, & s(t) = i, \\ 0, & s(t) \neq i. \end{cases}$$

When we consider the dynamics in the time interval $[0, T]$, supposing that the time instants at which $s(t)$ has a discontinuity are $t_1 < t_2 < ... < t_{M-1}$, we denote with $\Phi^T(x) = \Phi_{f_M}^{\Delta t_M} \circ ... \circ \Phi_{f_1}^{\Delta t_1}(x)$ the final position of the point x. Here, with $\Phi_{f_i}^{\Delta t_i}(x)$ we denote the Δt_i-solution of the Cauchy problem

$$\begin{cases} \dot{z}(t) = f_i(z(t)), \\ z(0) = x_0. \end{cases} \tag{6}$$

Standard existence and uniqueness results of solutions to ODEs are based on the continuity in time and Lipschitz regularity in the spatial variable. However, continuity in time is a requirement that becomes too restrictive in the setting of switching systems. Thus, we need to allow for less regular solutions to extend existence and uniqueness results to these problems as reported in theorem A.1

Theorem A.1 (Carathéodory existence and uniqueness [20])**.** *Consider the non-autonomous ODE*

$$\begin{cases} \dot{y}(t) = f(t, y(t)), \\ y(t_0) = t_0, \end{cases}$$

and let $t \in I \subset [t_0, +\infty)$. Assume f satisfies

- *$f(t, y)$ is Lipschitz-continuous in y for all fixed $t \in I$,*
- *$f(t, y)$ is piecewise-continuous in time.*

Then, the ODE is satisfied almost everywhere by one and only one function $t \mapsto y(t)$, which is absolutely continuous.

In other terms, if the regularity assumptions on f hold, there is only one solution to the integral equation

$$y(t) = y_0 + \int_{t_0}^{t} f(s, y(s)) ds,$$

which is differentiable almost everywhere.

The time-switching systems we consider, as a consequence, admit a unique solution if the ODE in (6) has a unique solution for any subsystem f_i. This is not a problem in the setting we consider, since we define neural networks starting from Lipschitz-continuous vector fields.

B Some explicit constructions of dynamical blocks

The three examples presented in the manuscript rely on constructing dynamical blocks that satisfy some desired properties. The presentation is meant to be quite general, without specifying precisely the expressions of the dynamical blocks one could use. To increase the understanding of the strategy, we illustrate briefly three explicitly written dynamical blocks, one for each of the examples.

B.1 Volume-preserving dynamical blocks

We now specialise the reasoning of section 1. We propose a particular choice of partitioned vector field as a fundamental element of these dynamical blocks. Consider $y = [x, v] \in \mathbb{R}^{2m}$, then we can define the two vector fields

$$f^1_{\theta_1}(y) = \begin{bmatrix} \Sigma(Av + a) \\ 0 \end{bmatrix}, \quad f^2_{\theta_2}(z) = \begin{bmatrix} 0 \\ \Sigma(Bx + b) \end{bmatrix},$$

with $\theta_1 = (A, a)$ and $\theta_2 = (B, b)$. These two vector fields are divergence-free; hence, their flow maps are volume-preserving. We can therefore build volume-preserving dynamical blocks composing flow maps of the form

$$\Psi^h_{\theta_1}(x, v) = [x + h\Sigma(Av + a), v],$$
$$\Psi^h_{\theta_2}(x', v') = [x', v' + h\Sigma(Bx' + b)].$$

This corresponds to alternating explicit-Euler steps of the vector fields $f^1_{\theta_1}$ and $f^2_{\theta_2}$. Indeed, these steps coincide with the exact flow maps of the two vector fields and are hence volume-preserving. More explicitly, the volume-preserving dynamical blocks can be built by composing maps of the form

$$y = \begin{bmatrix} x \\ v \end{bmatrix} \mapsto T_{(\theta_1,\theta_2)}(y) = y_{\text{new}} = \begin{bmatrix} x_{\text{new}} \\ v_{\text{new}} \end{bmatrix} = \begin{bmatrix} x + h\Sigma(Av + a) \\ v + h\Sigma(Bx_{\text{new}} + b) \end{bmatrix}.$$

We remark that along the network, the weights θ_1 and θ_2 can be changed to improve the network expressivity.

B.2 Mass-preserving dynamical blocks

In section 2, we have already presented how we model the skew-symmetric matrix-valued functions defining the dynamics. For clarity, we repeat it here and show explicitly how the mass-preserving dynamical blocks can be defined. We recall that a differential equation $\dot{y}(t) = f(y(t))$ on $\mathbb{R}^n$ is said to have a first integral $g : \mathbb{R}^n \to \mathbb{R}$ if $\nabla g(y) \cdot f(y) \equiv 0$. Thus mass-preserving ODEs admit the linear first integral $g(y) = \mathbf{1}^T y$ if and only if they can be rewritten as

$$\dot{y}(t) = (\hat{A}(y) - \hat{A}(y)^T)\mathbf{1}$$

with $\hat{A} : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ that is an upper triangular matrix. To model $\hat{A}$, we flatten it into a vector $A(y) \in \mathbb{R}^{n(n-1)/2}$ and we choose to model A with a feedforward neural network as

$$A_\theta(y) = W^T\Sigma(Vy + v)$$

for some weights $\theta = (W, V, v)$. To highlight the dependency on these parameters, we add a θ to the notation also in $\hat{A}_\theta$. To obtain the mass-preserving dynamical blocks, it only remains to find a numerical method that preserves the linear first integral g also at a discrete level. All Runge–Kutta methods do that, and we can prove it for explicit-Euler in a single line. Indeed, if for a vector field $f \in \mathfrak{X}(\mathbb{R}^n)$ we have $0 = b^T f(y)$ with $b \in \mathbb{R}^n$, then $g(y) = b^T y$ is a first integral of f and doing one Euler step for f we get

$$g(y_{n+1}) = b^T y_{n+1} = b^T(y_n + hf(y_n)) = b^T y_n + hb^T f(y_n) = b^T y_n = g(y_n).$$

Therefore, we see that g is conserved by the explicit Euler method. We conclude that a way to define mass-preserving dynamical blocks is by composing maps of the form

$$y \mapsto y_{\text{new}} = T_\theta(y) = y + h(\hat{A}_\theta(y) - \hat{A}_\theta(y)^T)\mathbf{1},$$

where θ is possibly changed along the network.

B.3 1-Lipschitz dynamical blocks

We have already reported a specific choice for the layers in 1-Lipschitz networks in section 3. We hence just briefly summarise the reasoning here. Let us first introduce the vector fields that we work with

$$f^c_{\theta_1}(y) = -A^T\Sigma(Ay+a), \quad f^e_{\theta_2}(y) = \Sigma(By+b), \quad \theta_1 = (A,a),\, \theta_2 = (B,b).$$

We also constrain the weights as $A^TA = I$ and $\|B\|_2 \leq 1$. The strategy we implement and propose in order to construct 1-Lipschitz dynamical blocks, is to compose explicit-Euler steps defined as

$$\begin{aligned}\Psi^{h_1}_{\theta_1}(y) &= y - h_1 A^T\Sigma(A^Ty+a)\\ \Psi^{h_2}_{\theta_2}(y) &= y + \Sigma(By+b).\end{aligned}$$

Hence the Lipschitz dynamical blocks can be obtained composing maps of the form

$$T_{(\theta_1,\theta_2)} = \Psi^{h_1}_{\theta_1} \circ \Psi^{h_2}_{\theta_2}$$

where we can also change the weights θ_1 and θ_2 along the layers. However, to have $T_{(\theta_1,\theta_2)}$ that is $1-$Lipschitz, we need to impose the step-size restriction reported in equation (5), and derived in more detail in Appendix C. This restriction can be imposed in various ways. For example, in case of trainable time-steps one can project the time steps after every optimisation step. It is also an option to fix them so that they satisfy the constraint.

C Detailed derivation of Lipschitz bounds

In this appendix, we present a more detailed derivation of the bound on the Lipschitz constant of a dynamical block reported in equation (5). This estimate aims to ensure that the composition of possibly expansive flow maps and contractive ones gives a 1-Lipschitz map. The two flow maps we consider are

$$\begin{aligned}&\Psi^{h_1}_1(x) = x - h_1 A^T\Sigma(Ax+b), \quad \sigma(x) = \max\{x, x/2\},\ A^TA = I,\\ &\Psi^{h_2}_2(y) = y + h_2 X(y), \quad \mathrm{Lip}(X) \leq 1.\end{aligned}$$

We first evaluate the bounds on the individual Lipschitz constants. We start with $\Psi^{h_1}_1$, introducing the notation

$$\delta(A,b,x,y) = A^T\Sigma(Ay+b) - A^T\Sigma(Ax+b)$$

to get

$$\begin{aligned}\|\Psi^{h_1}_1(y) - \Psi^{h_1}_1(x)\|^2 &= \langle y - x - h_1\delta(A,b,x,y), y - x - h_1\delta(A,b,x,y)\rangle\\ &= \|y-x\|^2 - 2h_1\langle y-x, \delta(A,b,x,y)\rangle + h_1^2\|\delta(A,b,x,y)\|^2\end{aligned}$$

Analysing the single terms, we notice that

$$\begin{aligned}-h_1\langle y-x, \delta(A,b,x,y)\rangle &= -h_1\langle y-x, A^T\Sigma(Ay+b) - A^T\Sigma(Ax+b)\rangle\\ &\leq -h_1\frac{\lambda_{\min}(A^TA)}{2}\|y-x\|^2\end{aligned}$$

by the strong convexity of σ. Here we denote the smallest eigenvalue of B with $\lambda_{\min}(B)$. Then it is also true that

$$h_1^2\|\delta(A,b,x,y)\|^2 = h_1^2\|A^T\Sigma(Ay+b) - A^T\Sigma(Ax+b)\| \leq h_1^2\|A\|^2\|y-x\|^2 \leq h_1^2\|y-x\|^2.$$

This derivation leads to the final estimate

$$\|\Psi^{h_1}_1(y) - \Psi^{h_1}_1(x)\|^2 \leq (1 - h_1\lambda_{\min}(A^TA) + h_1^2)\|y-x\|^2 = (1 - h_1 + h_1^2)\|y-x\|^2$$

if we suppose $A^TA = I$. On the other hand, for the discrete flow map $\Psi^{h_2}_2$, we have

$$\begin{aligned}\|\Psi^{h_2}_2(y) - \Psi^{h_2}_2(x)\| &= \|y + h_2X(y) - x - h_2X(x)\| \leq (1 + h_2\,\mathrm{Lip}(X))\|y-x\|\\ &\leq (1+h_2)\|y-x\|.\end{aligned}$$

This allows us to conclude what we reported in equation (5), i.e. that

$$\mathrm{Lip}(\Psi_1^{h_1} \circ \Psi_2^{h_2}) \leq \mathrm{Lip}(\Psi_1^{h_1}) \cdot \mathrm{Lip}(\Psi_2^{h_2}) \leq \sqrt{1 - h_1 + h_1^2} \cdot (1 + h_2).$$

Thus, if

$$(h_1, h_2) \in \mathcal{R} = \{(h_1, h_2) \in [0, 1]^2 : \ (1 + h_2)\sqrt{1 - h_1 + h_1^2} \leq 1\}, \tag{7}$$

the composition of the two functions is $1-$Lipschitz. Furthermore, if a dynamical block is made by the alternation of k pairs of flow maps with the same structure as $\Psi_1^{h_1}$ and $\Psi_2^{h_2}$, the condition in (7) on each pair is sufficient to get a $1-$Lipschitz dynamical block.

We remark that such reasoning can be extended without major changes to other choices of vector fields, other alternation strategies and other numerical methods.