Institute of Applied Analysis and Numerical Simulation

Research Group: Numerical Mathematics

Simulation Technology Degree Course

Bachelor Thesis

Symplectic Neural Networks

First Reviewer	Second Reviewer	Advisor
Prof. Dr. B. Haasdonk	Prof. Dr. D. Pflüger	Patrick Buchfink, M.Sc.
Institute of Applied Analysis and Numerical Simulation (IANS)	Institute of Parallel and Distributed Systems (Scientific Computing)	Institute of Applied Analysis and Numerical Simulation (IANS)

Submitted by

Author	Steffen Müller
Student ID	3260643
SimTech ID	119
Submission Date	01.12.2020

Abstract

We use physics-based neural networks to learn and predict the dynamics of Hamiltonian systems from data. We build on previous work on symplectic networks (SympNets) for identifying Hamiltonian systems from data. After recapitulating the architecture of SympNets, we extend SympNets by proposing symplectic variants of batch normalization and convolution. We achieve to improve the baseline performance of SympNets via batch normalization. We empirically verify that SympNets with batch normalization are more robust with respect to faster learning rates. With convolution, we make SympNets suitable for high-dimensional Hamiltonian ODE systems arising from semi-discretization of Hamiltonian PDEs. In particular, we successfully extrapolate trajectories of the the semi-linear wave equation with SympNets in our numerical experiments. We show how SympNets relate to two conventional geometric integrators commonly used for time integration of Hamiltonian systems.

Acknowledgements

I want to thank Patrick Buchfink for reviewing this thesis and for the numerous fruitful and enjoyable discussions.

Contents

1 Introduction

Nowadays, neural networks are successfully used in a lot of different contexts and fields, for example in object detection, object classification, speech recognizition, robotics, chemistry or medicine [3]. However, most of the time the neural networks, or in general the machine learning models, rely on a large amount of training data to achieve the desired performance. Prior structural knowledge is not used beneficially. In the recent field of physics-based machine learning, the objective is to embed prior knowledge from physics into the machine learning model, to for example reduce the dependency on a large amount of training data, or to improve the quality of the results. In the context of simulations, machine learning models can act as surrogate models, where the training data is generated from the original simulation. Often, these simulations are computationally expensive to execute and therefore one can only generate a limited amount of training data. However, most of the time there exists prior knowledge about the simulated system, which could be used beneficially.

In this thesis, we focus on predicting the dynamics of a special class of dynamical systems, so-called Hamiltonian (ODE) systems, with neural networks. The neural networks are fed from example data of the dynamical behavior of the Hamiltonian system, possibly from real-word measurements or simulation results. Generally, mechanical systems resulting from physical principles are Hamiltonian (ODE) systems as long as no dissipative elements are introduced [10]. Such conservative systems arise in a lot of applications, for example rigid body systems, molecular systems in chemistry or astronomicial systems [10]. Furthermore, high-dimensional Hamiltonian ODE systems arise from so-called Hamiltonian PDEs upon discretization in space [10]. Canonical Hamiltonian systems are fully described by a so-called Hamiltonian, which can be interpeted as the total energy of the system. Poincaré has shown in 1899 that these Hamiltonian systems have an exceptional property: The flow map, which operates on the so-called phase space and which advances the state of a Hamiltonian system by a certain time t, is symplectic. Informally, symplecticity is a stronger occurence of volume and orientation preservation [10]. This exceptional property of the flow map led to the development of so-called geometric numerical integrators, which are symplectic themselves. It turns out that these geometric integrators offer much better solutions over long time than their non-symplectic counterparts [5]. We want to learn such a flow map with a neural network. Motivated by the good results of the geometric integrators, it suggests itself to intrinsically embed the symplectic property into the structure of the neural networks.

Deco and Brauer [2] proposed a neural network architecture for the related problem of volume-preservation already in 1995. There have already been several neural network-based proposals how to identify Hamiltonian systems from data. For example, Greydanus et al. [4] propose a neural network to learn the Hamiltonian of the Hamiltonian system. After the Hamiltonian is learned, conventional numerical integrators can be used to predict the dynamics of the Hamiltonian system. Their approach involves computing the gradient of the neural network during training and during prediction with the numerical integrator. In addition, their approach requires time derivatives for the training data. Opposed to this indirect method, Jin et al. [9] instead learn the flow, which advances the state of a Hamiltonian system by a certain time t based on a previous (initial) state, directly with a symplectic neural network (SympNet). SympNets are constructed in a special way such that they are guaranteed to be symplectic. The approach by Jin et al. does not require computing the gradient of the neural network during training and prediction, making training and prediction more efficient. Furthermore, the networks proposed by Jin et al. only need examples from phase space instead of time derivatives for the training data.

In this thesis, we extend SympNets by introducing the concepts of batch normalization and convolution to SympNets, while maintaining their symplectic property. We achieve to improve the baseline performance of SympNets by proposing a symplectic version of batch normalization. The SympNets as proposed by Jin et al. are not suitable for high-dimensional Hamiltonian, as the parameter space would explode for such high-dimensional systems. By bringing the concept of convolution to SympNets, we make SympNets applicaple for high-dimensional Hamiltonian systems arising from Hamiltonian PDEs.

1.1 Outline

In Section 2, we introduce the necessary theory for Hamiltonian systems which is required in the following sections. In particular, we give the exact definition of a symplectic map. In Section 3, after we briefly introduce basic neural network terminology, we recapitulate the architecture of SympNets (symplectic networks) as proposed by Jin et al. in [9]. Jin et al. have also provided universal approximation theorems for SympNets. We repeat these statements in Section 4 and briefly discuss reasonable nonlinear activation functions for SympNets. In Section 5, we extend SympNets by introducing convolution and batch normalization to SympNets, while still maintaining their symplectic property. In Section 6, we build a connection between SympNets and two conventional geometric integrators. In Section 7, we compare the performance of SympNets with and without normalization. In addition, we compare their performance to neural networks which do not intrinsically preserve the symplectic structure. Furthermore, we apply convolutional SympNets to a high-dimensional Hamiltonian system arising from the semi-linear wave equation. In Section 8, we summarize our results and give an outlook on possible further topics to be explored.

In accordance with the examination regulations from 22th July 2016 of the bachelor's program *Simulation Technology* of the University of Stuttgart, Sections 2 to 4 and 6 belong to the propaedeuticum. The remaining sections belong to the bachelor thesis.

1.2 Notation

Given some entries $v_1, \dots, v_n \in \mathbb{R}$, we construct a vector $v \in \mathbb{R}^n$ with

$$v = \left(v_i\right)_{i=1}^{n}.$$

We denote the i-th entry of a vector $v \in \mathbb{R}^n$ with v_i or $(v)_i$. We write a matrix $A \in \mathbb{R}^{n\times k}$ consisting of the column vectors $v_1, \dots, v_k \in \mathbb{R}^n$ as $A = (v_1, \dots, v_k)$. Given some matrices $A, B, C, D \in \mathbb{R}^{n_1\times n_2}$, we write

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \in \mathbb{R}^{2n_1\times 2n_2}$$

for the matrix consisting of the blocks A, B, C and D. The n-by-n identity matrix is denoted by $I_n \in \mathbb{R}^{n\times n}$. If the dimension n can be inferred from context we may just write I. We denote the n-dimensional 1-vector with $1_n \in \mathbb{R}^n$.

We denote the k-th partial derivative of a function $f : \mathbb{R}^n \to \mathbb{R}, x \mapsto f(x)$ with $\frac{\partial}{\partial x_k} f(x)$, $k = 1, \dots, n$.

Given a multi-dimensional function $f : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$, $x \mapsto f(x)$, we denote the Jacobian with

$$\frac{\partial f}{\partial x} \in \mathbb{R}^{n_2\times n_1}.$$

If we want to emphasize that the Jacobian is evaluated at $v \in \mathbb{R}^{n_1}$, we write

$$\left.\frac{\partial f}{\partial x}\right|_{x=v} \in \mathbb{R}^{n_2\times n_1}.$$

For a scalar-valued function $f : \mathbb{R}^n \to \mathbb{R}$, we denote the gradient with $\nabla f \in \mathbb{R}^n$ and the Hessian matrix with $\nabla^2 f \in \mathbb{R}^{n\times n}$.

We write $\mathbb{N}$ for the set of naturnal numbers without zero, and we write $\mathbb{N}_0$ for the set of naturnal numbers with zero. We denote the set of discrete functions from $\mathbb{Z}$ to $\mathbb{R}$ as $\mathbb{R}^{\mathbb{Z}} := \{f : \mathbb{Z} \to \mathbb{R}\}$.

We write the Kronecker delta as

$$\delta_{ij} := \begin{cases} 1 & : i = j \\ 0 & : \text{else} \end{cases}.$$

The number $d \in \mathbb{N}$ will have a special meaning in the context of Hamiltonian systems. We reserve the usage of d for this purpose throughout this thesis. We define the skew-symmetric matrix $J \in \mathbb{R}^{2d\times 2d}$ as

$$J := \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix}.$$

For $q, p \in \mathbb{R}^d$ and maps $f_{11}, f_{12}, f_{21}, f_{22} : \mathbb{R}^d \to \mathbb{R}^d$, we define the block operator as

$$\begin{bmatrix} f_{11} & f_{12} \\ f_{21} & f_{22} \end{bmatrix} \begin{pmatrix} q \\ p \end{pmatrix} := \begin{pmatrix} f_{11}(q) + f_{12}(p) \\ f_{21}(q) + f_{22}(p) \end{pmatrix} \in \mathbb{R}^{2d}.$$

We denote the identity map with id.

2 Problem setup

Let us introduce the basic theory for Hamiltonian systems and describe the problem setup in more detail.

Definition 1. *A matrix $A \in \mathbb{R}^{2d\times 2d}$ is called symplectic if $A^T J A = J$.*

Definition 2. *A differentiable map $\phi : U \to \mathbb{R}^{2d}$ (where $U \subset \mathbb{R}^{2d}$ is an open set) is called symplectic if the Jacobian matrix $\frac{\partial\phi}{\partial x}$ is symplectic everywhere, i.e.*

$$\left(\frac{\partial\phi}{\partial x}\right)^T J \left(\frac{\partial\phi}{\partial x}\right) = J.$$

Definition 3. *A Hamiltonian (ODE) system can be written in canonical form as*

$$\begin{aligned} \dot{y}(t) &= J\nabla H(y(t)) \quad \text{for } t \in I \subset \mathbb{R} \\ y(t_0) &= y_0, \end{aligned}$$

where $y : I \subset \mathbb{R} \to \mathbb{R}^{2d}$, $t \mapsto y(t) = (q(t), p(t))$ and $y_0 = (q_0, p_0) \in \mathbb{R}^{2d}$ the initial value at $t_0 \in I$. The function $H : \mathbb{R}^{2d} \to \mathbb{R}$ is called the Hamiltonian or the total energy. The entries of $q = q(t) \in \mathbb{R}^d$ are called generalized coordinates and the entries of $p = p(t) \in \mathbb{R}^d$ are called conjugate momenta. The phase space $\mathcal{V} = \mathbb{R}^{2d}$ has even dimension for Hamiltonian systems.

Note that the Hamiltonian H is a first integral, i.e. the total energy is preserved, because

$$\frac{\mathrm{d}}{\mathrm{d}t} H(y(t)) = [\nabla H(y(t))]^T \dot{y}(t) = [\nabla H(y(t))]^T J\nabla H(y(t)) = 0$$

since J is skew-symmetric.

Let $\phi_{t,H} : U \subset \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ be the flow for a canonical Hamiltonian system with Hamiltonian H for a fixed time t, i.e.

$$\phi_{t,H}\begin{pmatrix} q_0 \\ p_0 \end{pmatrix} = \begin{pmatrix} q(t; q_0, p_0) \\ p(t; q_0, p_0) \end{pmatrix},$$

where $q(t; q_0, p_0)$ and $p(t; q_0, p_0)$ denote the solution of the Hamiltonian system at time t for initial values $y_0 = (q_0, p_0) \in \mathbb{R}^{2d}$. Poincaré has shown that the flow of a Hamiltonian system is symplectic.

Theorem 1. *(Poincaré 1899) Let $H : \mathbb{R}^{2d} \to \mathbb{R}$ be a twice continuously differentiable function on $U \subset \mathbb{R}^{2d}$. Then, for each fixed t, the flow $\phi_{t,H}$ is a symplectic map wherever it is defined.*

Proof. We refer to Hairer et al. [5, Theorem 2.4, p. 184] or to Leimkuhler and Reich [10, Theorem 1, p. 54]. □

Note that a Hamiltonian (ODE) system in general can be written as

$$\begin{aligned} \dot{y}(t) &= S\nabla H(y(t)) \quad \text{for } t \in I \subset \mathbb{R} \\ y(t_0) &= y_0 \end{aligned}$$

with an arbitrary nondegenerate skew-symmetric matrix $S \in \mathbb{R}^{2d\times 2d}$. However there always exists a transformation to express such a Hamiltonian ODE system in canonical form (see Peng and Mohseni [12, Remark 3.8]). Thus, we restrict w.l.o.g. to canonical systems.

Our goal is to learn the flow map $\phi_{t,H} : U \to \mathbb{R}^{2d}$ for fixed t with a neural network. To be specific, for fixed t, we supply the neural network with training pairs $(y_i, \phi_{t,H}(y_i))$ ($y_i \in \mathbb{R}^{2d}$ and $i = 1, \ldots, n_{\text{train}}$) and want the neural network to be able to predict $\phi_{t,H}(y)$ for arbitrary $y \in \mathbb{R}^{2d}$. This leads to the idea that we structurally embed symplecticity into the neural network itself.

Symplecticity has already been successfully embedded into numerical integrators for ODEs, which led to the development of geometric integrators, see for example Hairer et al. ([5]).

3 Architecture of SympNets

In this section we recapitulate the architecture of SympNets (Symplectic Networks) as proposed by Jin et al. in [9]. We prove symplecticity for all layer types.

First, let us briefly introduce neural network terminology which we will use. We refer to [3] for a comprehensive look at neural network concepts.

Definition 4. *(Layer) A neural network layer with input dimension n_1 and output dimension n_2 and parameters $\theta \in \mathbb{R}^{n_\theta}$ is a map $\phi : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$, $x \mapsto \mathcal{L}(x;\theta) = \mathcal{L}(x)$.*

A common layer is the so-called (fully-connected) linear layer $\phi(x) = Wx + b$ with parameters $W \in \mathbb{R}^{n_2 \times n_1}$ and $b \in \mathbb{R}^{n_2}$, i.e. $\theta = (vec(W)^T, b^T)^T \in \mathbb{R}^{n_2 n_1 + n_2}$.

In our special case we will always have $n_1 = n_2 = 2d$.

Definition 5. *(Neural Network) A neural network Φ is a composition of one or multiple layers with compatible input and ouput dimensions:*

$$\Phi(x;\Theta) = \phi_{n_L}\left(\phi_{n_L-1}\left(\cdots(\phi_2(\phi_1(x;\theta_1);\theta_2)\cdots\right);\theta_{n_L-1}\right);\theta_{n_L})$$

We denote the vector of all layer parameters with $\Theta = (\theta_1^T, \ldots, \theta_{n_L}^T)^T$.

We use neural networks in the context of supervised learning. In supervised learning, we want to learn a function based on some training data $\mathcal{T} = \{(x_1, y_1)\ldots,(x_{n_{\text{train}}}, y_{n_{\text{train}}})\} \subset \mathbb{R}^{n_x} \times \mathbb{R}^{n_y}$. Here, x_i refers to input and y_i refers to the expected output.

If the training data $\mathcal{T}$ does not fit into memory, we split the training data into disjoint subsets $\{\mathcal{B}_i\}_{i=1}^{n_\text{b}}$ with $\mathcal{T} = \mathcal{B}_1 \cup \cdots \mathcal{B}_{n_\text{b}}$. We call every $\mathcal{B}_i$ a mini-batch.

For a given loss function L, the neural network tries to minimize the loss function L by learning locally optimal parameters Θ_{min}. Typically, the loss function L is expressed as

$$L(\mathcal{B};\Theta) = \sum_{(x_i,y_i)\in\mathcal{B}} l(\Phi(x_i;\Theta), y_i),$$

where $l : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \to \mathbb{R}$ gives the loss for an individual training sample and $\mathcal{B}$ refers to a mini-batch or to the whole training data $\mathcal{T}$.

The neural network trains a local optimum Θ_{min} via an iterative algorithm from the gradient descent family. For example, given initial parameters Θ_0, the standard gradient descent algorithm is defined as

$$\Theta_{i+1} = \Theta_i - \gamma\nabla_\Theta L(\mathcal{T};\Theta_i).$$

Here, $\gamma \in \mathbb{R}$ denotes the learning rate. If the training data $\mathcal{T}$ is split into multiple mini-batches $\{\mathcal{B}_i\}_{i=1}^{n_\text{b}}$, every gradient-descent step is executed with a different mini-batch $\mathcal{B}_i$, i.e.

$$\Theta_{i+1} = \Theta_i - \gamma\nabla_\Theta L(\mathcal{B}_{((i \bmod n_\text{b})+1)};\Theta_i).$$

In this case, the algorithm is called stochastic gradient descent, because it estimates the gradient of the whole training data based on smaller mini-batches. The iteration index $i \in \mathbb{N}$ denotes the so-called epoch.

The algorithm for computing the gradient is called backpropagation, which is a special case of reverse-mode automatic differentation. Basically, the backpropagation algorithm first computes the loss value in a so-called forward pass and keeps the intermediate values in a computational graph. Afterwards, in the so-called backward pass, the chain rule is applied programmatically on the computational graph to compute the gradient, while reusing the intermediate values from the forward pass.

3.1 General architecture

Let us recapitulate the general architecture of SympNets. We use that the composition of symplectic maps is again symplectic. So we impose that every layer of the neural network must be symplectic, which leads to the overall neural network to be symplectic.

Definition 6. *(Unit Triangular Layer) A layer $\phi : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is called a unit triangular layer with layer transform $\mathrm{T} : \mathbb{R}^d \to \mathbb{R}^d$, $p \mapsto \mathrm{T}(p)$ and bias parameter $b \in \mathbb{R}^{2d}$, if ϕ can be expressed as*

$$\phi_{\textit{up}} \begin{pmatrix} q \\ p \end{pmatrix} = \begin{bmatrix} id & \mathrm{T} \\ 0 & id \end{bmatrix} \begin{pmatrix} q \\ p \end{pmatrix} + b = \begin{pmatrix} q + \mathrm{T}(p) \\ p \end{pmatrix} + b \quad \textit{(upper unit triangular layer)}$$

or

$$\phi_{\textit{low}} \begin{pmatrix} q \\ p \end{pmatrix} = \begin{bmatrix} id & 0 \\ \mathrm{T} & id \end{bmatrix} \begin{pmatrix} q \\ p \end{pmatrix} + b = \begin{pmatrix} q \\ \mathrm{T}(q) + p \end{pmatrix} + b. \quad \textit{(lower unit triangular layer)}$$

If we do not explicitly specify the bias parameter b, we assume that b is a non-learnable constant with value $b = 0$. The layer transform also depends on (learnable) parameters $\theta \in \mathbb{R}^{n_\theta}$, i.e. $\mathrm{T}(p) = \mathrm{T}(p;\theta)$, but we suppress the dependency on the parameters for shorter notation.

A SympNet only consists of unit triangular layers. In order that the unit triangular layers ϕ_{up} or ϕ_{low} are symplectic, we have to put additional requirements on the layer transform T, see Lemma 1 later in this section.

This structure is a special version of the structure proposed by Deco and Brauer in [2] for volume-conserving neural networks. Both SympNets and the volume-conserving neural networks proposed by Deco and Brauer are so-called residual neural networks:

Definition 7. *(Residual neural network) A neural network is called a residual neural network if it can be expressed as a composition of residual blocks*

$$R(x;\theta) = R(x) = \mathcal{F}(x;\theta) + x$$

with input $x \in \mathbb{R}^{n_1}$ and parameters $\theta \in \mathbb{R}^{n_\theta}$. The map $\mathcal{F}$ represents the residual mapping. In other words, the neural network Φ then can be expressed as

$$\Phi(x) = (R_n \circ \cdots \circ R_1)(x)$$

with residual blocks $R_1, \ldots, R_n$.

The term "residual neural network" was firstly introduced by He et al. in [6]. A SympNet is indeed a residual neural network, because a unit triangular layer itself forms a residual block:

$$\phi_{\text{up}} \begin{pmatrix} q \\ p \end{pmatrix} = \left(\begin{pmatrix} T(p) \\ 0 \end{pmatrix} + b \right) + \begin{pmatrix} q \\ p \end{pmatrix}$$

Lemma 1. *An upper or lower unit triangular layer $\phi_{\textit{up}}$ or $\phi_{\textit{low}}$ is symplectic if and only if the Jacobian of the layer transform $\mathrm{T} \in C(\mathbb{R}^d, \mathbb{R}^d)$ is symmetric everywhere.*

Proof. We show the result for ϕ_{up} only. The proof is analogous for ϕ_{low}. We have that

$$\frac{\partial \phi_{\text{up}}}{\partial(q,p)} = \begin{pmatrix} I & \frac{\partial \mathrm{T}}{\partial p} \\ 0 & I \end{pmatrix}.$$

It follows

$$\begin{aligned} \left(\frac{\partial \phi_{\text{up}}}{\partial(q,p)}\right)^T J \left(\frac{\partial \phi_{\text{up}}}{\partial(q,p)}\right) &= \left(\frac{\partial \phi_{\text{up}}}{\partial(q,p)}\right)^T \begin{pmatrix} 0 & I \\ -I & -\frac{\partial \mathrm{T}}{\partial p} \end{pmatrix} \\ &= \begin{pmatrix} I & 0 \\ \left(\frac{\partial \mathrm{T}}{\partial p}\right)^T & Id \end{pmatrix} \begin{pmatrix} 0 & I \\ -I & -\frac{\partial \mathrm{T}}{\partial p} \end{pmatrix} \\ &= \begin{pmatrix} 0 & I \\ -I & \left(\frac{\partial \mathrm{T}}{\partial p}\right)^T - \frac{\partial \mathrm{T}}{\partial p} \end{pmatrix} \overset{!}{=} J. \end{aligned}$$

Thus, ϕ_{up} is symplectic if and only if $\left(\frac{\partial \mathrm{T}}{\partial p}\right)^T - \frac{\partial \mathrm{T}}{\partial p} = 0$, i.e. if and only if the Jacobian $\frac{\partial \mathrm{T}}{\partial p}$ is symmetric everywhere. $\square$

The next corollary is useful to construct symplectic unit triangular layers.

Corollary 1. *Let $V : \mathbb{R}^d \to \mathbb{R},\ p \mapsto V(p)$ be a function in $C^2(\mathbb{R}^d)$. Then the upper and lower triangular layers with layer transform*

$$\mathrm{T}(p) = \nabla V(p)$$

are symplectic. We call V a potential.

Proof. The Jacobian matrix of ∇V corresponds to the Hessian matrix of V, i.e. $\frac{\partial(\nabla V)}{\partial p} = \nabla^2 V$. The Hessian is everywhere symmetric due to $V \in C^2(\mathbb{R}^d)$ (Lemma of Schwarz), thus the result follows with Lemma 1. □

3.2 Linear layers

In this section, we introduce symplectic linear layers. These linear layers can be interpreted as the symplectic version of the fully-connected linear layers used in a fully-connected neural network (FNN).

Definition 8. *(Linear layers) Given a symmetric matrix $S \in \mathbb{R}^{d\times d}$ and bias $b \in \mathbb{R}^{2d}$, we call the upper and lower unit triangular layers with layer transform $\mathrm{T}(p) = Sp$ the (symplectic) upper and lower linear layers ℓ_{up} and ℓ_{low}.*

The linear layers can be expressed with matrix-vector multiplication, for example

$$\ell_{\text{up}} \begin{pmatrix} q \\ p \end{pmatrix} = \begin{pmatrix} I & S \\ 0 & I \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix} + b.$$

The matrix S and the bias b are learnable parameters. In practice, we parametrize the symmetric matrix $S \in \mathbb{R}^{d\times d}$ with $S = A^T + A$ via another arbitrary matrix $A \in \mathbb{R}^{d\times d}$, as most optimization methods used for learning neural networks are designed for unconstrained optimization problems.

The linear layers are symplectic, because the Jacobian of the layer transform $\frac{\partial \mathrm{T}}{\partial p} = S$ is a symmetric matrix by defintion. Alternatively, we can apply Corollary 1 by choosing the potential $V(p) := p^T A p$.

Proof. For $k = 1, \dots, d$, we have

$$\begin{aligned}(\nabla V(p))_k &= \frac{\partial}{\partial p_k}\left(\sum_{i,j=1}^{d} A_{ij} p_i p_j\right) = \sum_{i,j=1}^{d} A_{ij} \frac{\partial}{\partial p_k}(p_i) p_j + \sum_{i,j=1}^{d} A_{ij} p_i \frac{\partial}{\partial p_k}(p_j) \\ &= \sum_{j=1}^{d} A_{kj} p_j + \sum_{i=1}^{d} A_{ik} p_i = ((A + A^T)p)_k.\end{aligned}$$

Choose $A = \frac{1}{2}S \implies A + A^T = S$. Thus, symplecticity follows with Corollary 1. □

We may enhance expressivity of a linear layer by alternately composing multiple ℓ_{up} and ℓ_{low}. We define $\mathcal{L}^n_{\text{up}},\ \mathcal{L}^n_{\text{low}} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ with

$$\begin{aligned}\mathcal{L}^n_{\text{up}} \begin{pmatrix} q \\ p \end{pmatrix} &:= \begin{pmatrix} I & 0/S_n \\ S_n/0 & I \end{pmatrix} \cdots \begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} \begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix} + b \\ \mathcal{L}^n_{\text{low}} \begin{pmatrix} q \\ p \end{pmatrix} &:= \begin{pmatrix} I & S_n/0 \\ 0/S_n & I \end{pmatrix} \cdots \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ S_1 & I \end{pmatrix} + b.\end{aligned}$$

The composed layers $\mathcal{L}^n_{\text{up}}$ and $\mathcal{L}^n_{\text{low}}$ are again symplectic because they are a composition of the symplectic maps ℓ_{up} and ℓ_{low}.

Jin et al. show in [8] that $\mathcal{L}^9_{up}$ can parametrize every symplectic linear map. In other words, the set of all possible $\mathcal{L}^9_{\text{up}}$ is equal to the set of all symplectic linear maps.

It does not make sense to put two upper linear layers or two lower linear layers after each other, because this reduces to a single upper or lower linear layer. We briefly show this statement for two upper linear layers $\ell_{up,2}$ and $\ell_{up,1}$:

$$\begin{aligned}(\ell_{up,2} \circ \ell_{up,1}) \begin{pmatrix} q \\ p \end{pmatrix} &= \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} \left(\begin{pmatrix} I & S_1 \\ 0 & I \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix} + b_1 \right) + b_2 \\ &= \begin{pmatrix} I & S_1 + S_2 \\ 0 & I \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix} + \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} b_1 + b_2 \\ &= \begin{pmatrix} I & S \\ 0 & I \end{pmatrix} \begin{pmatrix} q \\ p \end{pmatrix} + b,\end{aligned}$$

with $S := S_1 + S_2$ and $b := \begin{pmatrix} I & S_2 \\ 0 & I \end{pmatrix} b_1 + b_2$.

3.3 Activation layers

In this section, we introduce symplectic activation layers, which play the same role as the activation layers in fully-connected neural networks. In contrast to the linear layers introduced in the previous section, the activation layers are nonlinear as long as the chosen activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is nonlinear.

Definition 9. *(Activation layers) Given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ and coefficients $a \in \mathbb{R}^d$, we call the upper and lower unit triangular layers with layer transform*

$$\mathrm{T}(p) = (a_i \sigma(p_i))_{i=1}^d$$

the upper and lower activation layers $\mathcal{N}_{up}$ and $\mathcal{N}_{low}$.

The coefficients $a \in \mathbb{R}^d$ are learnable parameters.

Corollary 2. *Given an activation function $\sigma \in C^1(\mathbb{R})$, the activation layers $\mathcal{N}_{up}$ and $\mathcal{N}_{low}$ are symplectic.*

Proof. Let $\mathcal{A} : \mathbb{R} \to \mathbb{R}$ be an integral of σ, i.e. $\frac{\mathrm{d}}{\mathrm{d}x}\mathcal{A} = \sigma$. We define the potential

$$V(p) := \sum_{k=1}^d a_k \mathcal{A}(p_k).$$

Then $V \in C^2(\mathbb{R}^d, \mathbb{R})$ and for $k = 1, \dots d$ follows

$$(\nabla V(p))_i = \frac{\partial}{\partial p_i} \left(\sum_{k=1}^d a_k \mathcal{A}(p_k) \right) = a_i \sigma(p_i) = (\mathrm{T}(p))_i \, .$$

Symplecticity follows with Corollary 1. □

3.4 Gradient layers

The so-called gradient layers are nonlinear layers with greater expressivity than activation layers. The layer transform T of a gradient layer can be compared to two fully-connected linear layers with a nonlinear activation in-between. The name "gradient layer" is motivated by the fact that the layer transform T can approximate, under certain conditions, an arbitrary ∇V for $V \in C^1(\mathbb{R}^d)$, see Jin et al. [9, Lemma 4]. The gradient layers are, among others, inspired by the symmetric layer in Ruthotto and Haber [14].

Definition 10. *(Gradient layers) Given a width $n \in \mathbb{N}$, $K \in \mathbb{R}^{n \times d}$, $a, c \in \mathbb{R}^n$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, we call the upper and lower triangular layers with layer transform*

$$\mathrm{T}(p) = K^T \Big(a_j \sigma\left((Kp)_j + c_j \right) \Big)_{j=1}^n$$

the upper and lower gradient layers $\mathcal{G}_{up}$ and $\mathcal{G}_{low}$.

Here, $K \in \mathbb{R}^{n\times d}$ and $a, c \in \mathbb{R}^n$ are learnable parameters. The number $n \in \mathbb{N}$ denotes the width of the gradient layer and can be chosen freely. In practice, we choose $n \gg d$ for large expressivity.

Corollary 3. *Given an activation function $\sigma \in C^1$, the gradient layers $\mathcal{G}_{up}$ and $\mathcal{G}_{low}$ are symplectic.*

Proof. Let $\mathcal{A} : \mathbb{R} \to \mathbb{R}$ be an integral of σ, i.e. $\frac{\mathrm{d}}{\mathrm{d}x}\mathcal{A} = \sigma$. We define the potential

$$V(p) := \sum_{j=1}^{n} a_j \mathcal{A}((Kp)_j + c_j).$$

Then $V \in C^2(\mathbb{R}^d, \mathbb{R})$ and for $i = 1, \ldots, d$ follows

$$(\nabla V(p))_i = \frac{\partial}{\partial p_i}\left(\sum_{j=1}^{n} a_j \mathcal{A}((Kp)_j + c_j)\right) = \sum_{j=1}^{n} a_j \sigma((Kp)_j + c_j)\underbrace{\frac{\partial}{\partial p_i}((Kp)_j)}_{=K_{ji}=K^T_{ij}} = (\mathrm{T}(p))_i.$$

With Corollary 1 follows that the Gradient layers $\mathcal{G}_{up}$ and $\mathcal{G}_{low}$ are symplectic. □

Note that activation layers are a subset of gradient layers. We can see this by choosing $n = d$, $K = I_d$ and $c = 0$.

4 Univeral approximation theorems

In [9], Jin et al. show approximation theorems for SympNets. We repeat the statements here and refer to [9] for proofs. We define the norm on $C^r(W, \mathbb{R}^n)$ for a compact set $W \subset \mathbb{R}^m$ as

$$\|f\|_{C^r(W,\mathbb{R}^n)} := \sum_{|\alpha|\le r} \max_{1\le i\le n} \sup_{x\in W} \left|\frac{\partial^{|\alpha|}}{\partial x_1^{\alpha_1}\cdots x_m^{\alpha_m}} f_i(x)\right|,$$

where $f = (f_1, \ldots, f_n)^T \in C^r(W, \mathbb{R}^n)$ and $\alpha \in \mathbb{N}_0^m$ denotes a multi-index, i.e. $|\alpha| = \alpha_1 + \cdots + \alpha_m$.

Definition 11. *(r-finite) Let $r \in \mathbb{N}_0$ be given. The function σ is r-finite if $\sigma \in C^r(\mathbb{R})$ and $0 < \int \left|\frac{\mathrm{d}^r}{\mathrm{d}x^r}\sigma\right| d\lambda < \infty$. Here, λ is the Lebesgue measure on $\mathbb{R}$.*

Definition 12. *(r-uniformly dense on compacta) Let $m, n \in \mathbb{N}$, $r \in \mathbb{N}_0$ be given, $U \subset \mathbb{R}^m$ an open set, and $S_1 \subset C^r(U, \mathbb{R}^n)$. Then we say S_2 is r-uniformly dense on compacta in S_1 if $S_2 \subset S_1$ and for all $f \in S_1$, compact $W \subset U$ and every $\epsilon > 0$, there exists a $g \in S_2$ such that $\|f - g\|_{C^r(W,\mathbb{R}^n)} < \epsilon$.*

Definition 13. *(LA-SympNet) We call a neural network Φ a LA-SympNet if it can be expressed as a composition of linear and alternating lower and upper activation layers*

$$\Phi = \mathcal{L}^n \circ \left(\mathcal{N}_{\textit{up/low}} \circ \mathcal{L}^n\right) \circ \cdots \circ \left(\mathcal{N}_{\textit{up}} \circ \mathcal{L}^n\right) \circ \left(\mathcal{N}_{\textit{low}} \circ \mathcal{L}^n\right).$$

Here, $\mathcal{L}^n = \mathcal{L}^n_{\textit{low}}$ or $\mathcal{L}^n = \mathcal{L}^n_{\textit{up}}$, where $n \in \mathbb{N}$ denotes the number of the so-called (linear) sublayers. The depth of the LA-SympNet is determined by the number of activation layers $\mathcal{N}_{\textit{up/low}}$.

Definition 14. *(G-SympNet) We call a neural network Φ a G-SympNet if it can be expressed as a composition of alternating lower and upper gradient layers*

$$\Phi = \mathcal{G}_{\textit{up/low}} \cdots \circ \mathcal{G}_{\textit{up}} \circ \mathcal{G}_{\textit{low}},$$

where all gradient layers share the same width $n \in \mathbb{N}$. The depth of the G-SympNet is determined by the number of gradient layers $\mathcal{G}_{\textit{up/low}}$.

Let us denote the set of all symplectic maps in $C^r(U, \mathbb{R}^{2d})$ for an open set $U \subset \mathbb{R}^{2d}$ as

$$\mathcal{SP}^r(U) = \left\{\phi \in C^r(U, \mathbb{R}^{2d}) : \phi \text{ is a symplectic map}\right\}$$

Theorem 2. *(Approximation theorem for LA-SympNets) For all $r \in \mathbb{N}$ and open sets $U \subset \mathbb{R}^{2d}$, the set of all LA-SympNets is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$ if the activation function σ is r-finite.*

Theorem 3. *(Approximation theorem for G-SympNets) For all $r \in \mathbb{N}$ and open sets $U \subset \mathbb{R}^{2d}$, the set of all G-SympNets is r-uniformly dense on compacta in $\mathcal{SP}^r(U)$ if the activation function σ is r-finite.*

The approximation theorems do not give any information about the required size of the SympNet, or how long it takes to learn a SympNet, but they state that for every smooth symplectic map, there exists a LA-SympNet or G-SympNet which approximates the smooth symplectic map arbitrarily well on a compact set, under certain conditions.

Jin et al. have already shown in [9, Lemma 1] that the sigmoid activation function $\sigma(x) = \frac{1}{1+e^{-x}}$ is r-finite for $r \in \mathbb{N}$. This implies that the tanh activation function, $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, also is r-finite for $r \in \mathbb{N}$, because with $\tanh(x) = 2\sigma(2x) - 1$, it follows for $r \in \mathbb{N}$ that $\tanh \in C^r(\mathbb{R})$ and

$$\frac{\mathrm{d}^r}{\mathrm{d}x^r}\tanh(x) = 2^{r+1}\frac{\mathrm{d}^r}{\mathrm{d}x^r}\sigma(2x).$$

Thus we have

$$0 < \int_{-\infty}^{\infty} \left| \frac{\mathrm{d}^r}{\mathrm{d}x^r}\tanh(x) \right| dx = 2^{r+1}\int_{-\infty}^{\infty} \left| \frac{\mathrm{d}^r}{\mathrm{d}x^r}\sigma(2x) \right| dx = 2^{r+1}\int_{-\infty}^{\infty} \left| \frac{\mathrm{d}^r}{\mathrm{d}x^r}\sigma(\tilde{x}) \right| d\tilde{x} < \infty,$$

since the sigmoid function σ is r-finite. This is the motivation why we conduct our numerical experiments with the sigmoid and tanh activation function. In addition to these two activation functions, we conduct the experiments with the ELU activation function defined as

$$\mathrm{ELU}(x) = \begin{cases} x & : x > 0 \\ a(\exp(x) - 1) & : x \leq 0 \end{cases}$$

The ELU activation function is continously differentiable for $a = 1$, which makes it a reasonable choice in our context. However, the ELU activation function is not r-finite for all $r \in \mathbb{N}_0$, since it is only continously differentiable once and the integral $\int \left| \frac{\mathrm{d}^r}{\mathrm{d}x^r}\mathrm{ELU} \right| d\lambda$ diverges for $r = 0$ and $r = 1$. Universal approximation theorems for SympNets for non-r-finite activation functions is an open topic to be explored.

5 Convolution and Normalization for SympNets

In this section we introduce new extensions to SympNets. In particular, we bring the concept of convolution to SympNets and we propose a possibility how to embed normalization into SympNets while maintaining their symplecticity.

5.1 Introduction to Convolution

Convolution in neural networks has been successfully used in a variety of applications, because it enables parameter sharing and leads to sparse-connectivity, opposed to a fully-connected layer [3]. Basically, in convolutional neural networks, matrix multiplication is replaced by convolution, which also is a linear operation.

We introduce convolution in a rather abstract way in order to simplify the following proofs for symplecticity. General convolution is defined on infinite-dimensional function spaces. Neural networks implement a finite-dimensional version, as the operation has to be computable.

Given two discrete functions $f, g \in \mathbb{R}^{\mathbb{Z}}$ convolution is defined as

$$* : \mathbb{R}^{\mathbb{Z}} \times \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}, \quad (f * g)(\tau) = \sum_{a=-\infty}^{\infty} f(a)g(\tau - a).$$

However, most popular neural network libraries actually do not implement real convolution in convolution layers. Instead, they implement the so called cross-correlation, but call it convolution. Cross-correlation is a flipped convolution and is defined as

$$\star : \mathbb{R}^{\mathbb{Z}} \times \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}, \quad (f \star g)(\tau) = \sum_{a=-\infty}^{\infty} f(a)g(\tau + a).$$

Because most popular neural network libraries implement cross correlation and call it convolution, we stick to the same convention and only use cross-correlation from now on. Still, the choice is arbitrary, as the following statements could be constructed analogously with convolution.

We work with finite-dimensional vectors in neural networks. Therefore, we define a bijective mapping between vectors in $\mathbb{R}^n$ and functions in $\mathbb{R}^{\mathbb{Z}}$ via zero-continuation on $\mathbb{Z}$:

$$\mathcal{I}_n : \mathbb{R}^n \to \mathbb{R}^{\mathbb{Z}}, \quad (\mathcal{I}_n(x))(\tau) := \sum_{a=1}^{n} x_a \delta_{\tau a}$$
$$\mathcal{I}_n^{\text{inv}} : \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^n, \quad \mathcal{I}_n^{-1}(f) := (f(\tau))_{\tau=1}^{n}$$

Additionially, we define a shift operator $\mathcal{S}_\zeta$, which shifts a discrete function $f \in \mathbb{R}^{\mathbb{Z}}$ by $\zeta \in \mathbb{Z}$ positions in the right direction:

$$\mathcal{S}_\zeta : \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}, \quad (\mathcal{S}_\zeta(f))(\tau) := f(\tau - \zeta)$$

Definition 15. *(Valid cross-correlation) We define the valid cross-correlation of a finite-dimensional kernel $k \in \mathbb{R}^{n_k}$ and a finite-dimensional input $x \in \mathbb{R}^{n_x}$ with $n_x \geq n_k$ as*

$$\star_{\mathsf{v}} : \mathbb{R}^{n_k} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_x - n_k + 1}, \quad \star_{\mathsf{v}}(k, x) := \mathcal{I}_{n_x - n_k + 1}^{-1}(\mathcal{S}_{-1}(\mathcal{I}_{n_k}(k)) \star \mathcal{I}_{n_x}(x)).$$

In other words, the valid cross-correlation $\star_{\mathsf{v}}$ evaluates the cross-correlation only at positions where the finite-dimensional kernel k and the finite-dimensional input x fully overlap and puts the result into a finite-dimensional vector.

The kernel k will be a parameter which the neural network should learn. Note that in this context it does not make a difference if we use convolution or cross-correlation, because the neural network would just learn a flipped version of the kernel k in the opposite case.

We define a stride operator $\mathcal{D}_s$ for a stride number $s \in \mathbb{N}$ with

$$\mathcal{D}_s : \mathbb{R}^{\mathbb{Z}} \to \mathbb{R}^{\mathbb{Z}}, \quad (\mathcal{D}_s)(\tau) := f(s\tau).$$

The stride operators skips s entries of $f \in \mathbb{R}^{\mathbb{Z}}$.

Definition 16. *(Valid cross-correlation with stride) Given a finite-dimensional kernel $k \in \mathbb{R}^{n_k}$, a finite-dimensional input $x \in \mathbb{R}^{n_x}$, stride number $s \in \mathbb{N}$ and output dimension*

$$n_{\mathit{out}} = \left\lfloor \frac{n_x - n_k}{s} \right\rfloor + 1$$

the valid-cross correlation with stride is defined as

$$\star_{\mathsf{v}} : \mathbb{R}^{n_k} \times \mathbb{R}^{n_x} \to \mathbb{R}^{n_{\mathit{out}}}, \quad \star_{\mathsf{v}}(k, x) := (\mathcal{I}_{n_{\mathit{out}}}^{-1} \circ D_s)(\mathcal{S}_{-1}(\mathcal{I}_{n_k}(k)) \star \mathcal{I}_{n_x}(x)).$$

Note that Definition 16 equals Definition 15 for stride number $s = 1$.

5.2 Convolution layers

The valid cross correlation (without stride) has the output dimension $n_x - n_k + 1$. Thus, the valid cross-correlation $\star_{\mathsf{v}}$ decreases the dimension of the input x. However, we want to incorporate the valid cross-correlation as the layer transform T of a unit triangular layer. The layer transform T must keep the dimension d. Therefore we apply a so-called padding operation on the layer transform input $p \in \mathbb{R}^d$ before passing it to the valid cross-correlation. The padding operation increases the dimension of p in order that the composition of padding and cross-correlation keeps the overall dimension. Let $c \in \mathbb{N}$ denote the number by which the padding operator increases the input dimension, such that the input dimension for the cross-correlation is $n_x = d + c$. In order that the composition of padding and cross-correlation keeps the dimension, it must hold

$$n_x - n_k + 1 = \underbrace{(d+c)}_{\substack{\text{increased dim.} \\ \text{by } c \text{ via padding}}} - n_k + 1 = d \iff c = n_k - 1.$$

For symmetry reasons, we pad the same number of values at the beginning and the end of the input vector p. This means that the number of padded values c has to be even, i.e. we can write $c = 2m$ for a $m \in \mathbb{N}_0$. Consequently, the equation above implies that the kernel size n_k has to be odd, i.e. $n_k = 2m + 1$. We define two different padding operators, which both increase the input vector dimension by $2m$.

Definition 17. *(Constant padding) Given the padding values $l, r \in \mathbb{R}^m$, we define constant padding $c_{\mathit{pad},m}$ as*

$$c_{\mathit{pad},m} : \mathbb{R}^d \to \mathbb{R}^{d+2m}, \quad c_{\mathit{pad},m}(p) = c_{\mathit{pad},m}(p; l, r) := \left(l^T, p^T, r^T\right)^T.$$

The constant padding $c_{\text{pad},m}$ is an affine linear map in p, because we can write

$$c_{\text{pad},m}(p; l, r) = c_{\text{pad}}(p; 0_m, 0_m) + c_{\text{pad},m}(0_d; l, r) \tag{1}$$

and $c_{\text{pad},m}(p; 0_m, 0_m)$ is linear regarding p.

Definition 18. *(Symmetric padding) We define symmetric padding as*

$$s_{\mathit{pad},m} : \mathbb{R}^d \to \mathbb{R}^{d+2m}, \quad s_{\mathit{pad},m}(p) := (\underbrace{p_m, p_{m-1} \dots, p_1}_{m \text{ values}}, \underbrace{p_1, p_2 \dots, p_d}_{d \text{ values}}, \underbrace{p_d, p_{d-1} \dots, p_{d-m+1}}_{m \text{ values}})^T.$$

The symmetric padding $s_{\text{pad},m}$ is a linear map. We call the operation symmetric padding, because the outermost left and right inner vector entries are reflected at the beginning and the end of the vector.

Given an odd kernel $k \in \mathbb{R}^{2m+1}$, we set

$$\hat{k} = \hat{k}(k) := \mathcal{S}_{-m-1}(\mathcal{I}_{2m+1}(k)).$$

Then $\hat{k}(-m) = k_1, \dots, \hat{k}(m) = k_{2m+1}$ and $\hat{k}(\tau) = 0$ for $|\tau| > m$. With $\hat{k}$, we can express the valid cross-correlation as

$$\star_{\mathsf{v}}(k, x) = \mathcal{I}^{-1}_{n_x - 2m}(\mathcal{S}_m(\hat{k}) \star \mathcal{I}_{n_x}(x)),$$

because

$$\mathcal{S}_m(\hat{k}) = \mathcal{S}_m(\mathcal{S}_{-m-1}(\mathcal{I}_{2m+1}(k))) = \mathcal{S}_{m+(-m-1)}(\mathcal{I}_{2m+1}(k)) = \mathcal{S}_{-1}(\mathcal{I}_{2m+1}(k)).$$

Definition 19. *(Symmetric kernel) We call an odd kernel $k \in \mathbb{R}^{2m+1}$ symmetric if*

$$\hat{k}(\tau) = \hat{k}(-\tau)$$

for all $\tau \in \mathbb{Z}$.

Let us now define symplectic convolution layers by combining padding and valid cross-convolution.

Definition 20. *(Convolution layers) Given a symmetric kernel $k \in \mathbb{R}^{2m+1}$, padding $\chi_{\mathit{pad},m} = c_{\mathit{pad},m}$ or $\chi_{\mathit{pad},m} = s_{\mathit{pad},m}$, we call the upper and lower unit triangular layers with bias $b \in \mathbb{R}^{2d}$ and layer transform*

$$\mathrm{T}(p) := \star_{\mathsf{v}}(k, \chi_{\mathit{pad},m}(p))$$

the (symplectic) convolution layers $\mathcal{C}_{up}$ and $\mathcal{C}_{low}$.

The symmetric kernel $k \in \mathbb{R}^{2m+1}$ is parametrized by $m+1$ learnable parameters, see Section 5.2.1 later. If $\chi_{\text{pad},m} = c_{\text{pad},m}$, the padding values $l, r \in \mathbb{R}^m$ can either be learnable parameters or constant. The convolution layers can be used without any bias ($b = 0$), a constant non-learnable bias ($b \neq 0 =$ const), or with a learnable bias parameter (b is a learnable parameter). We stick to $b = 0$ in our experiments.

With $\hat{\chi}_{\text{pad},m}(p) := \mathcal{I}_{d+2m}(\chi_{\text{pad},m}(p))$, we can write the layer transform as

$$\mathrm{T}(p) = \mathcal{I}_d^{-1}(\mathcal{S}_m(\hat{k}) \star \hat{\chi}_{\text{pad},m}(p)). \tag{2}$$

Lemma 2. *Let $f : \mathbb{R}^d \to \mathbb{R}^d$ be a linear map. Then the Jacobian matrix $\frac{\partial f}{\partial x}$ is constant, i.e.*

$$\left.\frac{\partial f}{\partial x}\right|_{x=v} = \left.\frac{\partial f}{\partial x}\right|_{x=w} \quad \forall v, w \in \mathbb{R}^d$$

and for the Jacobian-vector product holds

$$\left(\left.\frac{\partial f}{\partial x}\right|_{x=v}\right) w = f(w) \quad \forall v, w \in \mathbb{R}^d$$

.

Proof. f is linear $\implies$ There exists a matrix representation $f(x) = Ax$ with $A \in \mathbb{R}^{d\times d}$
$\implies \left.\frac{\partial f}{\partial x}\right|_{x=v} = A =$ const. $\forall v \in \mathbb{R}^d \implies \left(\left.\frac{\partial f}{\partial x}\right|_{x=v}\right) w = Aw = f(w) \quad \forall v, w \in \mathbb{R}^d$. □

As the Jacobian matrix for a linear map is constant, we omit the evaluation point, i.e. $\left.\frac{\partial f}{\partial x}\right|_{x=v} = \frac{\partial f}{\partial x}$.

Theorem 4. *The convolution layers $\mathcal{C}_{up}$ and $\mathcal{C}_{low}$ with padding $\chi_{\text{pad},m} = c_{\text{pad},m}$ are symplectic.*

Proof. We have to show that the Jacobian of the layer transform $\mathrm{T}(p)$ is symmetric (Lemma 1). Because of (1), it suffices to show the case $l, r = 0$ (the Jacobian of a constant has only zero-valued entries). For $l, r = 0$, the layer transform $\mathrm{T}(p)$ is a linear map, because it is a composition of linear maps only.

Define the bilinear form

$$b : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \quad b(v, w) := \left\langle v,\ \left(\frac{\partial \mathrm{T}}{\partial p}\right) w \right\rangle. \tag{3}$$

To show symmetry of the Jacobian, we show that $b(e_i, e_j) = b(e_j, e_i)$ for all $i, j = 1, \dots, d$:

$$\begin{aligned} b(e_i, e_j) &= \left\langle e_i,\ \left(\frac{\partial \mathrm{T}}{\partial p}\right) e_j \right\rangle = \langle e_i,\ \sigma_{\mathcal{C}}(e_j)\rangle \quad \text{(Lemma 2)} \\ &\overset{(2)}{=} \left\langle e_i,\ \mathcal{I}_d^{-1}(\mathcal{S}_m(\hat{k}) \star \hat{c}_{\text{pad},m}(e_j))\right\rangle \\ &= \left\langle e_i,\ \left(\sum_{a=-\infty}^{\infty} \hat{k}(a-m)\, \underbrace{(\hat{c}_{\text{pad},m}(e_j))(\tau + a)}_{=\delta_{(\tau+a)(j+m)}}\right)_{\tau=1}^{d} \right\rangle \\ &= \hat{k}((j - i + m) - m) = \hat{k}(j - i) \end{aligned}$$

The kernel k is symmetric, thus $\hat{k}(j-i) = \hat{k}(i-j) \implies b(e_i, e_j) = b(e_j, e_i)$. □

Theorem 5. *The convolution layers $\mathcal{C}_{up}$ and $\mathcal{C}_{low}$ with padding $\chi_{\text{pad},m} = s_{\text{pad},m}$ are symplectic.*

Proof. With $\chi_{\text{pad},m} = s_{\text{pad},m}$, the layer transform T is linear, because it is a composition of linear maps. Thus, we proof the statement the same way as Theorem 4 above. For $\tau, a \in \mathbb{Z}$ we have

$$\hat{s}_{\text{pad},m}(e_j)(\tau + a) = \delta_{(\tau+a)(m-j+1)} + \delta_{(\tau+a)(j+m)} + \delta_{(\tau+a)(2d+m-j+1)}.$$

Consequently, for $\chi_{\text{pad},m} = s_{\text{pad},m}$ and $i, j = 1, \dots, d$, the bilinear form (3) becomes

$$\begin{aligned} b(e_i, e_j) &= \left\langle e_i,\ \left(\sum_{a=-\infty}^{\infty} \hat{k}(a-m)(\hat{s}_{\text{pad},m}(e_j))(\tau + a)\right)_{\tau=1}^{d} \right\rangle \\ &= \hat{k}((m - j + 1 - i) - m) + \hat{k}((j + m - i) - m) + \hat{k}((2d + m - j + 1 - i) - m) \\ &= \hat{k}(1 - j - i) + \hat{k}(j - i) + \hat{k}(1 + 2d - j - i). \end{aligned}$$

The kernel k is symmetric, thus $\hat{k}(j-i) = \hat{k}(i-j) \implies b(e_i, e_j) = b(e_j, e_i)$. □

5.2.1 Parametrization of a symmetric kernel

Let $\{b_1, b_2, \dots b_{m+1}\} \subset \mathbb{R}^{2m+1}$ be a basis of the space $\{k \in \mathbb{R}^{2m+1} : k$ is a symmetric kernel$\}$. The symmetric kernel k is then parametrized by the coefficients $\beta \in \mathbb{R}^{m+1}$ via

$$k = (b_1, b_2, \dots, b_{m+1})\beta = \beta_1 b_1 + \beta_2 b_2 + \dots + \beta_{m+1} b_{m+1}.$$

One choice is the canonical basis with basis vectors $b_1, \dots, b_{m+1} \in \mathbb{R}^{2m+1}$ given by

$$(b_i)_{j+m+1} = \hat{k}(b_i)(j) = \begin{cases} 1 & : |j| = i-1 \\ 0 & : else \end{cases} \qquad (j = -m, \dots, m).$$

Another possible basis choice inspired by finite differences is:

$$\begin{aligned} b_1^{FD} &= (0, \dots, 0,1, 0, \dots, 0)^T \in \mathbb{R}^{2m+1}, \\ b_2^{FD} &= (0, \dots, 0, 1, -2, 1, 0, \dots, 0)^T \in \mathbb{R}^{2m+1}, \\ b_3^{FD} &= (0, 0, \dots, 0, 1, -4,6, -4, 1, 0, \dots, 0)^T \in \mathbb{R}^{2m+1} \\ &\vdots \end{aligned}$$

In general, the entries for a basis vector $b_i^{FD} \in \mathbb{R}^{2m+1}$ $(1 \leq i \leq m)$ originate from Pascal's triangle.

$$\left(b_i^{FD}\right)_{j+m+1} = \hat{k}(b_i^{FD})(j) = \begin{cases} (-1)^j \binom{2(i-1)}{j+i-1} & : |j| < i \\ 0 & : \text{else} \end{cases} \qquad (j = -m, \dots, m)$$

The symmetry of the kernel basis vectors b_i^{FD} follows with the definition of the binomial coefficient:

$$\begin{aligned} \binom{2(i-1)}{j+i-1} &= \frac{2(i-1)!}{(j+i-1)!(2(i-1) - (j+i-1))!} \\ &= \frac{2(i-1)!}{(j+i-1)!(i-j-1)!} \end{aligned}$$

Thus we have

$$\binom{2(i-1)}{(-j)+i-1} = \binom{2(i-1)}{j+i-1}$$

and for $i = 1, \dots, m$ and $j = 0, \dots, m$ holds

$$\begin{aligned} \hat{k}(b_i^{FD})(-j) &= \begin{cases} (-1)^{-j} \binom{2(i-1)}{(-j)+i-1} : |-j| < i \\ 0 : else \end{cases} \\ &= \begin{cases} (-1)^j \binom{2(i-1)}{j+i-1} : |j| < i \\ 0 : else \end{cases} \\ &= \hat{k}(b_i^{FD})(j). \end{aligned}$$

So $\{b_i^{FD}\}_{i=1}^{m+1}$ is a valid basis of the space $\{k \in \mathbb{R}^{2m+1} : k$ is a symmetric kernel$\}$.

Ruthotto and Haber [14] discuss similar parametrizations for convolution kernels, albeit in a more general PDE context. They suggest for example to parametrize the kernel by taking a linear combination of basis kernels refering to reaction, convection and diffusion terms. In our numerical experiments, it turns out that parametrization plays an important role how well a neural network learns.

5.3 Convolution Gradient Layers

In this section, we propose convolution for gradient layers. Ruthotto and Haber [14] discuss convolution for a similar residual layer in a more general PDE context.

For a fixed kernel k, valid cross-correlation is a linear map regarding the second argument. Therefore there exists a representation matrix A_k with $\star_\mathsf{v}(k,p) = A_k p$. Transposed valid cross-correlation is defined as the linear map associated with the transpose A_k^T of A_k. We denote the transposed valid cross-correlation by $\star_\mathsf{v}^T$.

Again, most popular neural network libraries say convolution, but actually implement valid cross-correlation. In this case, transposed convolution actually refers to transposed valid cross-correlation.

For large $d \in \mathbb{N}$, it can make sense to use convolution inside Gradient layers instead of a full matrix K. Let us repeat the layer transfrom T for Gradient layers:

$$\mathrm{T}(p) = K^T \left(a_j \sigma \left((Kp)_j + c_j \right) \right)_{j=1}^{n}$$

A gradient layer with width $n \in \mathbb{N}$ can be implemented by using valid cross-correlation (with stride) for $K^T \in \mathbb{R}^{d \times n}$ and the corresponding transposed valid cross-correlation (with stride) for $K \in \mathbb{R}^{n \times d}$. The choice of transpose is intentional, because we want to upscale the input $p \in \mathbb{R}^d$ ($n >> d$). Cross-correlation (without padding) decreases the dimension. Therefore we first apply transposed cross-correlation on p, because the transpose increases the dimension. A large kernel size n_k and a large stride number s allow for significant upscaling. We emphasize that the kernel k has not to be symmetric in this setting. It is reasonable that the parametrization for $a \in \mathbb{R}^n$ and $c \in \mathbb{R}^n$ also respects the convolution in a certain way. One possibility to parametrize $a \in \mathbb{R}^n$ with a parameter $\tilde{a} \in \mathbb{R}^{n_k}$ is

$$c = \star_\mathsf{v}^T(\tilde{c}, 1_d).$$

The vector $c \in \mathbb{R}^n$ can be parametrized with a parameter $\tilde{c} \in \mathbb{R}^{n_k}$ analogously. To summarize, the convolution gradient layers then can be defined as:

Definition 21. *(Convolution gradient layers) Given a kernel size $n_k \in \mathbb{N}$, a stride number $s \in \mathbb{N}$, kernel parameters $k, \tilde{a}, \tilde{c} \in \mathbb{R}^{n_k}$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, we call the upper and lower triangular layers with layer transform*

$$T(p) = \star_\mathsf{v}\left(k, \left(a_j \sigma\left(\left(\star_\mathsf{v}^T(k,p)\right)_j + c_j\right)\right)_{j=1}^{n}\right) \quad \text{with } c = \star_\mathsf{v}^T(\tilde{c}, 1_d) \text{ and } a = \star_\mathsf{v}^T(\tilde{a}, 1_d)$$

the upper and lower convolution gradient layers $\mathcal{G}_{up}^{conv}$ and $\mathcal{G}_{low}^{conv}$. The width $n \in \mathbb{N}$ corresponds to the output dimension of the transposed valid cross-correlation $\star_\mathsf{v}^T$ and thus depends on the kernel size n_k and the stride number s.

The convolution gradient layers are symplectic by construction, because they are a special parametrization of gradient layers.

5.4 Normalization

Batch normalization is a standard method to accelerate training initially proposed by Ioffe and Szegedy in [7]. One motivation for batch normalization is that the sigmoid activation function saturates for large absolute values, which in turn results in vanishing gradients. The motivation of batch normalization then is to prevent getting into the saturating regime of the sigmoid function by normalizing the input. In a more general remark, Santurkar et al. [15] argue that batch normalization smooths the optimization landscape. We introduce a possibility to incorporate batch normalization into activation and gradient layers while maintaining their symplecticity.

Given a mini-batch input $\mathcal{B} = \{x_1, x_2, \ldots, x_{n_\mathcal{B}}\} \subset \mathbb{R}^n$ with size $n_\mathcal{B}$, we define the batch normalization transformation as

$$\eta_{\gamma,\beta} : \mathbb{R}^n \to \mathbb{R}^n, \quad \eta_{\gamma,\beta}(x) := \left(\frac{\gamma_j}{(\sigma_\mathcal{B}^2)_j + \epsilon} \left(x_j - (\mu_\mathcal{B})_j \right) + \beta_j \right)_{j=1}^{n}.$$

where n is the input dimension of the input $x \in \mathbb{R}^n$ and $\gamma, \beta \in \mathbb{R}^n$ are learnable parameters. The scalar $\epsilon \in \mathbb{R}$ is a small positive value to avoid division by zero. Here, $\sigma_{\mathcal{B}}^2 \in \mathbb{R}^n$ refers to the mini-batch variance and $\mu_{\mathcal{B}} \in \mathbb{R}^n$ refers to the mini-batch mean. The batch normalization results in zero mean and unit variance of the whole mini-batch input.

The learnable parameters $\gamma, \beta \in \mathbb{R}^n$ allow the neural network to modify the normalization during training if necessary. The batch normalization transform is able to represent the identity transform by setting appropriate γ, β.

The mean $\mu_{\mathcal{B}} \in \mathbb{R}^n$ and variance $\sigma_{\mathcal{B}}^2 \in \mathbb{R}^n$ are estimated by:

$$\mu_{\mathcal{B}} = \frac{1}{n_{\mathcal{B}}} \sum_{i=1}^{n_{\mathcal{B}}} x_i$$
$$\sigma_{\mathcal{B}}^2 = \frac{1}{n_{\mathcal{B}}} \sum_{i=1}^{n_{\mathcal{B}}} (x_i - \mu_{\mathcal{B}})^2$$

During training the mini-batch variance $\sigma_{\mathcal{B}}^2 \in \mathbb{R}^n$ and the mini-batch $\mu_{\mathcal{B}} \in \mathbb{R}^n$ are continously updated in the forward pass. To be precise, for a new mini-batch input $\mathcal{B} = \{x_1, x_2, \dots, x_{n_{\mathcal{B}}}\} \subset \mathbb{R}^n$, the mean $\mu_{\mathcal{B}}$ and variance $\sigma_{\mathcal{B}}^2$ are updated based on $\mathcal{B}$, before $\eta_{\gamma,\beta}(x_k)$ for $x_k \in \mathcal{B}$ is evaluated (see Algorithm 1). When training has finished the neural network does not update $\sigma_{\mathcal{B}}^2$ and $\mu_{\mathcal{B}}$ anymore. Instead, the neural network remembers the last value from training.

Note that the batch normalization transformation may be incorporated into a layer deep inside a neural network. If this is the case, the mini-batch input $\mathcal{B}$ refers to the collective output of the previous layer.

Algorithm 1 Batch normalization transform

```
Input: mini batch $\mathcal{B} = \{x_1, x_2, \dots, x_{n_{\mathcal{B}}}\} \subset \mathbb{R}^n$ and $x_k \in \mathcal{B}$
Output: $\eta_{\gamma,\beta}(x_k) \in \mathbb{R}^n$
  if training_mode then        // Update mean and variance if training, otherwise keep previous values
    $\mu_{\mathcal{B}} \leftarrow \frac{1}{n_{\mathcal{B}}} \sum_{i=1}^{n_{\mathcal{B}}} x_i$
    $\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{n_{\mathcal{B}}} \sum_{i=1}^{n_{\mathcal{B}}} (x_i - \mu_{\mathcal{B}})^2$
  end if
  return $\left( \frac{\gamma_j}{(\sigma_{\mathcal{B}}^2)_j + \epsilon} \left( (x_k)_j - (\mu_{\mathcal{B}})_j \right) + \beta_j \right)_{j=1}^{n}$
```

The batch normalization transform can be implemented in a straightforward way with modern neural network libraries, as during training the Jacobian $\frac{\partial \eta_{\gamma,\beta}}{\partial(\gamma,\beta)}$ is obtained via automatic differentiation of the batch normalization transform.

If the training data is small enough, so that splitting the data into multiple mini batches is not necessary, the mean and variance are calculated for the whole training data set. This is the case for our numerical experiments, as we work with very small training data sets.

5.4.1 Normalized gradient layers

In this section, we incorporate the batch normalization transform into gradient layers.

Definition 22. *(Normalized gradient layers - Variant 1) Given a width $n \in \mathbb{N}$, $K \in \mathbb{R}^{n \times d}$, $a, c \in \mathbb{R}^n$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, we call the upper and lower triangular layers with layer transform*

$$\mathrm{T}(p) = K^T \left(a_j \sigma\left(\overline{p}_j\right) \right)_{j=1}^{n} \quad \textit{with } \overline{p} = \eta_{\gamma,\beta}\left(Kp + c\right) \in \mathbb{R}^n$$

the upper and lower normalized gradient layers $\mathcal{G}_{up}^{\eta}$ and $\mathcal{G}_{low}^{\eta}$ (variant 1).

Definition 23. *(Normalized gradient layers - Variant 2) Given a width $n \in \mathbb{N}$, $K \in \mathbb{R}^{n\times d}$, $a, c \in \mathbb{R}^n$ and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$, we call the upper and lower triangular layers with layer transform*

$$\mathrm{T}(p) = K^T \left(a_j \frac{\gamma_j}{(\sigma_{\mathcal{B}}^2)_j + \epsilon} \sigma\left(\overline{p}_j\right) \right)_{j=1}^n \quad \text{with } \overline{p} = \eta_{\gamma,\beta}\left(Kp + c\right) \in \mathbb{R}^n$$

the upper and lower normalized gradient layers $\mathcal{G}^\eta_{up}$ and $\mathcal{G}^\eta_{low}$ (variant 2). The variance $\sigma^2_{\mathcal{B}} \in \mathbb{R}^n$ refers to the mini-batch variance of $Kp + c$.

Here, $K \in \mathbb{R}^{n\times d}$ and $a, c, \gamma, \beta \in \mathbb{R}^n$ are learnable parameters. It turns out in our numerical experiments that variant 1 outperforms variant 2 or is equally good as variant 2 in almost all cases.

Corollary 4. *Given an activation function $\sigma \in C^1(\mathbb{R})$, the normalized gradient layers $\mathcal{G}^\eta_{up}$ and $\mathcal{G}^\eta_{down}$ (variant 1 and variant 2) are symplectic.*

Proof. Let $\mathcal{A} : \mathbb{R} \to \mathbb{R}$ be the antiderivative of σ, i.e. $\frac{\mathrm{d}}{\mathrm{d}x}\mathcal{A} = \sigma$. For arbitrary $\alpha \in \mathbb{R}^n$, we define the potential

$$V(p) := \sum_{j=1}^n \alpha_j a_j \mathcal{A}(\overline{p}_j) \quad \text{with } \overline{p} = \eta_{\gamma,\beta}\left(Kp + c\right) \in \mathbb{R}^n.$$

Then $V \in C^2(\mathbb{R}^d, \mathbb{R})$ and for $i = 1, \ldots, d$ follows

$$(\nabla V(p))_i = \frac{\partial}{\partial p_i}\left(\sum_{j=1}^n \alpha_j a_j \mathcal{A}(\hat{p}_j)\right) = \sum_{j=1}^n \alpha_j a_j \sigma(\overline{p}_j) \frac{\partial}{\partial p_i}\overline{p}_j = \sum_{j=1}^n \alpha_j a_j \sigma(\overline{p}_j) \frac{\gamma_j}{(\sigma^2_{\mathcal{B}})_j + \epsilon} \underbrace{K_{ji}}_{=K^T_{ij}}$$

$$\implies \nabla V(p) = K^T \left(\alpha_j a_j \frac{\gamma_j}{(\sigma^2_{\mathcal{B}})_j + \epsilon} \sigma\left(\overline{p}_j\right)\right)_{j=1}^n .$$

If we choose $\alpha_j = 1$, it follows $\nabla V(p) = \mathrm{T}(p)$ for variant 2. If we choose $\alpha_j = \frac{(\sigma^2_{\mathcal{B}})_j + \epsilon}{\gamma_j}$, it follows $\nabla V(p) = \mathrm{T}(p)$ for variant 1. With Corollary 1 follows that both normalized Gradient layer variants $\mathcal{G}^\eta_{up}$ and $\mathcal{G}^\eta_{low}$ are symplectic. □

5.4.2 Normalized convolution gradient layers

Analogously to the convolution gradient layers introduced in Section 5.3, we can implement a normalized convolution gradient layer (variant 1 and variant 2) with width $n \in \mathbb{N}$ using valid cross-correlation (with stride) for $K^T \in \mathbb{R}^{d\times n}$ and the corresponding transposed valid cross-correlation (with stride) for $K \in \mathbb{R}^{n\times d}$. The layer transform T for the normalized version (variant 1) of the convolution gradient layers then can be written as

$$T(p) = \star_{\mathsf{v}}\left(k,\ \left(a_j \sigma\left(\overline{p}_j\right)\right)_{j=1}^n\right) \quad \text{with } \overline{p} = \eta_{\gamma,\beta}\left(\underbrace{\star^T_{\mathsf{v}}(k,p) + c}_{:=\tilde{p}}\right),\ c = \star^T_{\mathsf{v}}(\tilde{c}, 1_d) \text{ and } a = \star^T_{\mathsf{v}}(\tilde{a}, 1_d).$$

The variant 2 follows analogously by inserting the factor $\frac{\gamma_j}{(\sigma^2_{\mathcal{B}})_j + \epsilon}$ at the respective place. It is reasonable that the normalization respects the convolution in a certain way. This also involves calculating the mean and variance in a way that the convolution is respected.

In our numerical experiments, we only use normalized convolution gradient layers with equal kernel size n_k and stride number s, i.e. $n_k = s$. For this special configuration, the output dimension of the transposed cross-correlation is $n_k d$ and the vector $\tilde{p} \in \mathbb{R}^{n_k d}$, which is the input for the batch normalization transform, can be expressed as

$$\tilde{p}_i = \left(\star^T_{\mathsf{v}}(k,p) + \star^T_{\mathsf{v}}(\tilde{c}, 1_d)\right)_i = \begin{cases} k_i p_1 + \tilde{c}_i & : 1 \le i \le n_k \\ k_{i-n_k} p_2 + \tilde{c}_{i-n_k} & : n_k + 1 \le i \le 2n_k \\ k_{i-2n_k} p_3 + \tilde{c}_{i-2n_k} & : 2n_k + 1 \le i \le 3n_k \\ \ldots \end{cases} \quad (1 \le i \le n_k d).$$

So the transposed cross-correlation results in a upscaled vector $\tilde{p} \in \mathbb{R}^{n_k d}$ with blocks of size n_k. Therefore, we implement the batch normalization transform $\overline{p} = \eta_{\gamma,\beta}(\tilde{p})$ in such a way that $\tilde{p}_i$ is normalized the same as $\tilde{p}_{i+n_k}$ is constructed for all $i = 1, \dots, n_k(d-1)$. In particular, this means that $(\mu_{\mathcal{B}})_i = (\mu_{\mathcal{B}})_{i+n_k}$, $(\sigma^2_{\mathcal{B}})_i = (\sigma^2_{\mathcal{B}})_{i+n_k}$, $\gamma_i = \gamma_{i+n_k}$ and $\beta_i = \beta_{i+n_k}$ for $i = 1, \dots, n_k(d-1)$ and that the parameters γ and β can be parametrized with vectors $\tilde{\gamma} \in \mathbb{R}^{n_k}$ and $\tilde{\beta} \in \mathbb{R}^{n_k}$. All respective entries of $\tilde{p} \in \mathbb{R}^{n_k d}$ belonging together are used to compute the mean and variance. As before, the mean and variance are computed over all training samples in the mini-batch $\mathcal{B}$. Similar normalization schemes can be constructed for other configurations of the kernel size n_k and stride value s.

5.4.3 Normalized activation layers

Definition 24. *(Normalized activation layers - Variant 1) Given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ and coefficients $a \in \mathbb{R}^d$, we call the upper and lower unit triangular layers with bias $b = 0$ and layer transform*

$$\mathrm{T}(p) = (a_i \sigma(\hat{p}_i))_{i=1}^d \quad \text{with } \hat{p} = \eta_{\gamma,\beta}(p) \in \mathbb{R}^d$$

the normalized upper and lower activation layers $\mathcal{N}^\eta_{\mathit{up}}$ and $\mathcal{N}^\eta_{\mathit{low}}$ (variant 1).

Definition 25. *(Normalized activation layers - Variant 2) Given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ and coefficients $a \in \mathbb{R}^d$, we call the upper and lower unit triangular layers with bias $b = 0$ and layer transform*

$$\mathrm{T}(p) = \left(a_i \frac{\gamma_i}{(\sigma^2_{\mathcal{B}})_i + \epsilon} \sigma(\hat{p}_i) \right)_{i=1}^d \quad \text{with } \hat{p} = \eta_{\gamma,\beta}(p) \in \mathbb{R}^d$$

the normalized upper and lower activation layers $\mathcal{N}^\eta_{\mathit{up}}$ and $\mathcal{N}^\eta_{\mathit{low}}$ (variant 2). The variance $\sigma^2_{\mathcal{B}} \in \mathbb{R}^d$ refers to the mini-batch variance of p.

The coefficients $a \in \mathbb{R}^d$ are learnable parameters.

Corollary 5. *Given an activation function $\sigma \in C^1(\mathbb{R})$, the normalized activation layers $\mathcal{N}^\eta_{up}$ and $\mathcal{N}^\eta_{low}$ (variant 1 and variant 2) are symplectic.*

Proof. Symplecticity follows because the normalized activation layers are a special case of normalized gradient layers (choose $n = d$, $K = I_d$ and $c = 0$ for normalized gradient layers). We have already shown that both normalized gradient layer variants are symplectic in Corollary 4. □

6 Relation to geometric integrators

The symplectic Euler and Störmer-Verlet schemes are two geometric integrators. Geometric integrators are numerical integrators for ODE systems, which preserve a geometric property. In our context, this geometric property is symplecticity. Precisely, if we denote the numerical integrator with $\phi^{\mathsf{h}}_{t,H} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$, the map $\phi^{\mathsf{h}}_{t,H}$ is symplectic.

The two variants of the symplectic Euler scheme are given by:

$$\begin{aligned} p_{n+1} &= p_n - h\nabla_q H(p_{n+1}, q_n) \\ q_{n+1} &= q_n + h\nabla_p H(p_{n+1}, q_n) \end{aligned} \quad \text{or} \quad \begin{aligned} p_{n+1} &= p_n - h\nabla_q H(p_n, q_{n+1}) \\ q_{n+1} &= q_n + h\nabla_p Hp(p_n, q_{n+1}) \end{aligned}$$

The p-staggered variant of the Störmer-Verlet scheme is given by

$$p_{n+1/2} = p_n - \frac{h}{2}\nabla_q H(p_{n+1/2}, q_n)$$

$$q_{n+1} = q_n + \frac{h}{2}\left(\nabla_p H(p_{n+1/2}, q_n) + \nabla_p H(p_{n+1/2}, q_{n+1})\right)$$

$$p_{n+1} = p_{n+1/2} - \frac{h}{2}\nabla_q H(p_{n+1/2}, q_{n+1})$$

and the q-staggered variant of the Störmer-Verlet scheme is given by

$$\begin{aligned} q_{n+1/2} &= q_n + \frac{h}{2}\nabla_p H(p_n, q_{n+1/2}) \\ p_{n+1} &= q_n - \frac{h}{2}\left(\nabla_q H(p_n, q_{n+1/2}) + \nabla_q H(p_{n+1}, q_{n+1/2})\right) \\ q_{n+1} &= q_{n+1/2} + \frac{h}{2}\nabla_p H(p_{n+1}, q_{n+1/2}). \end{aligned}$$

We refer to Hairer et al. [5, p. 189 and p. 190] for details.

If the Hamiltonian H is separable, i.e. $H(q,p) = U(q) + V(p)$, the symplectic Euler and Störmer-Verlet schemes become explicit. Then given $\nabla_p H : \mathbb{R}^d \to \mathbb{R}^d$, $p \mapsto \nabla_p H(p)$ and $\nabla_q H : \mathbb{R}^d \to \mathbb{R}^d$, $q \mapsto \nabla_q H(q)$, the left variant of the symplectic Euler scheme can be expressed as

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{bmatrix} id & h\nabla_p H \\ 0 & id \end{bmatrix} \begin{bmatrix} id & 0 \\ -h\nabla_q H & id \end{bmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

and the right variant of the symplectic Euler scheme as

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{bmatrix} id & 0 \\ -h\nabla_q H & id \end{bmatrix} \begin{bmatrix} id & h\nabla_p H \\ 0 & id \end{bmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}.$$

Similarly, the p-staggered Störmer-Verlet scheme can be expressed as

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{bmatrix} id & 0 \\ -\frac{h}{2}\nabla_q H & id \end{bmatrix} \begin{bmatrix} id & h\nabla_p H \\ 0 & id \end{bmatrix} \begin{bmatrix} id & 0 \\ -\frac{h}{2}\nabla_q H & id \end{bmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}$$

and the q-staggered Störmer-Verlet scheme as

$$\begin{pmatrix} q_{n+1} \\ p_{n+1} \end{pmatrix} = \begin{bmatrix} id & \frac{h}{2}\nabla_p H \\ 0 & id \end{bmatrix} \begin{bmatrix} id & 0 \\ -h\nabla_q H & id \end{bmatrix} \begin{bmatrix} id & \frac{h}{2}\nabla_p H \\ 0 & id \end{bmatrix} \begin{pmatrix} q_n \\ p_n \end{pmatrix}.$$

To conclude, the explicit Euler and Störmer-Verlet schemes can be expressed with the same unit triangular structure we use for SympNets. This means that a SympNet is able to reproduce both schemes exactly or approximately, provided appropriate layer choices.

Symplecticity for the explicit Euler and Störmer-Verlet schemes follows directly from Corollary 1 and the fact that the composition of symplectic maps is again symplectic.

7 Numerical experiments

In this section, we compare the performance of different SympNet architectures for predicting flows of Hamiltonian systems. Additionially, we compare SympNets to network architectures which do not intrinsically conserve the symplectic property. After the neural networks have been trained with a certain training set, we use them as numerical integrators to predict a trajectory for a given initial value y_0 of a Hamiltonian ODE. A single evaluation of the neural network then represents one time iteration.

In all our experiments, we use the mean-square error (MSE) loss defined by

$$L(\mathcal{B}, \Theta) = \sum_{(x_i, y_i) \in \mathcal{B}} \|\Phi(x_i; \Theta) - y_i\|_2^2,$$

where $\mathcal{B}$ refers to a mini-batch and Θ refers to the neural network parameters. We work with very small training data. Thus, we do not split the training data into mini-batches. We initialize all bias parameters with 0. All other parameters are initialized randomly from the normal distribution with mean 0 and standard deviation 0.01. For training, we use the AMSgrad variant of the Adam optimizer [13]. The standard Adam optimizer led to large loss spikes at later epochs and often failed to converge to stable parameter values. This phenomenon was greatly reduced by switching to the AMSgrad variant.

The Adam alogrithm (and the AMSgrad variant) computes individual adaptive learning rates for different parameters, based on a baseline learning rate hyperparameter $\gamma \in \mathbb{R}$.

In all our experiments the training and test data are subsets of $\mathcal{V} \times \mathcal{V}$, where $\mathcal{V} = \mathbb{R}^{2d}$ denotes the phase space of the Hamiltonian system. In particular, given some phase space samples $x_1, \ldots, x_n \in \mathcal{V}$, we generate data samples $\mathcal{T} \subset \mathcal{V} \times \mathcal{V}$ with the q-staggered Störmer-Verlet integrator $\phi^{\mathsf{h}}_{t,H}$ for a fixed time $t \in \mathbb{R}$ and the respective Hamiltonian H:

$$\mathcal{T} = \{(x_i, \phi^{\mathsf{h}}_{t,H}(x_i))\}_{i=1}^{n_{\text{train}}}$$

The data samples $\mathcal{T}$ are split into a training data set $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{test}}$, i.e. $\mathcal{T}_{\text{train}} \cup \mathcal{T}_{\text{test}} = \mathcal{T}$ and $\mathcal{T}_{\text{train}} \cap \mathcal{T}_{\text{test}} = \emptyset$. We denote the size of the training data set with $n_{\text{train}} \in \mathbb{N}$ and the size of the test data set with $n_{\text{test}} \in \mathbb{N}$. The training data $\mathcal{T}_{\text{train}}$ and test data $\mathcal{T}_{\text{test}}$ are used to calculate the respective training and test loss.

7.1 Low-dimensional systems

In this section, we deal with low-dimensional Hamiltonian systems with $d = 1$. The neural network architectures used in the experiments with low-dimensional systems are listed in Table 1.

For all low-dimensional experiments, the training and test data are generated with the fixed time $t = 0.1$ from phase space samples uniformly sampled from a compact set $\mathcal{D} \subset \mathbb{R}^2$.

7.1.1 Harmonic Oscillator

For $q, p \in \mathbb{R}$, the Hamiltonian for the Harmonic Oscillator is given by

$$H(q,p) = \frac{p^2}{2m} + \frac{1}{2}kq^2.$$

For the experiment, we choose $m = 1$ and $k = 1$.

The flow for the Harmonic Oscillator is linear, because the resulting ODE is linear. Thus, the ODE can be solved via the variation of constants formula, which results in a linear flow. As we have mentioned, Jin et al. have shown in [8] that $\mathcal{L}^9_{up}$ can parametrize every symplectic linear map. Consequently, a SympNet consisting out of nine alternating upper and lower linear layers should be able to learn the flow for the Harmonic Oscillator. Indeed, Figure 1 shows that such a linear SympNet successfully learns the flow for the Harmonic Oscillator with a very low test loss.

7.1.2 Simple Pendulum

For $q, p \in \mathbb{R}$, the Hamiltonian for the Simple Pendulum is given by

$$H(q,p) = \frac{p^2}{2ml^2} + mgl(1 - \cos(q)).$$

For the experiment, we choose $m = 1, g = 1$ and $l = 1$. The flow of the Simple Pendulum is nonlinear. Therefore, the flow cannot be represented by linear models, which makes the Simple Pendulum a good experiment to look into. The generalized coordinate q denotes the angle of the pendulum.

We conduct two different experiments with different phase space samples originating from two different sets $\mathcal{D}$:

$$\begin{aligned} \mathcal{D}_{\text{swing}} &= [-\frac{\pi}{2}, \frac{\pi}{2}] \times [-\sqrt{2}, \sqrt{2}] \quad \text{(swinging)} \\ \mathcal{D}_{\text{swing+rotate}} &= [-20, 20] \times [-2.5, 2.5] \quad \text{(swinging and rotating)} \end{aligned}$$

Physically, $\mathcal{D}_{\text{swing}}$ mostly contains phase spaces states where the pendulum is swinging, in contrast to $\mathcal{D}_{\text{swing+rotate}}$. We choose $n_{\text{train}} = 40$ and $n_{\text{test}} = 400$ for all experiments involving the Simple Pendulum. The test data is sampled from the same compact set $\mathcal{D}$ as the training data.

Architecture	Layers	Parameters
L-SympNet	Nine alternating upper and lower linear layers, with bias in last layer, see Definition 8. Initially proposed by Jin et al. in [9].	11
FNN	A fully-connected network with overall input and output dimension 2. The network has one hidden (fully-connected) linear layers with input and output dimension 50. Nonlinear activation layers are placed between all linear layers.	2802
LA-SympNet	LA-SympNet with depth 5 and 4 sublayers, see Definition 13. Initially proposed by Jin et al. in [9].	41
N1-LA-SympNet	Same as LA-SympNet, but with normalized activation layers (variant 1, see Definition 24) instead of activation layers.	51
N2-LA-SympNet	Same as LA-SympNet, but with normalized activation layers (variant 2, see Definition 25) instead of activation layers.	51
G-SympNet	G-SympNet with depth 4 and width 30, see Definition 14. Initially proposed by Jin et al. in [9].	360
N1-G-SympNet	Same as G-SympNet, but with normalized Gradient layers (variant 1, see Definition 22) instead of Gradient layers.	600
N2-G-SympNet	Same as G-SympNet, but with normalized Gradient layers (variant 2, see Definition 23) instead of Gradient layers.	600
LARGE-FNN	A fully-connected network with overall input and output dimension 2. The network has two hidden (fully-connected) linear layers with input and output dimension 70. Nonlinear activation layers are placed between all linear layers.	10292
LARGE-LA-SympNet	LA-SympNet with depth 40 and 9 sublayers, see Definition 13. Initially proposed by Jin et al. in [9].	491
LARGE-N1-LA-SympNet	Same as LARGE-LA-SympNet, but with normalized activation layers (variant 1, see Definition 24) instead of activation layers.	571
LARGE-N2-LA-SympNet	Same as LARGE-LA-SympNet, but with normalized activation layers (variant 2, see Definition 25) instead of activation layers.	571

Table 1: Table of all neural network architectures for the low-dimensional experiments. The activation function used inside the activation and gradient layers is either sigmoid, tanh or ELU and is explicitly stated in the respective experiment.

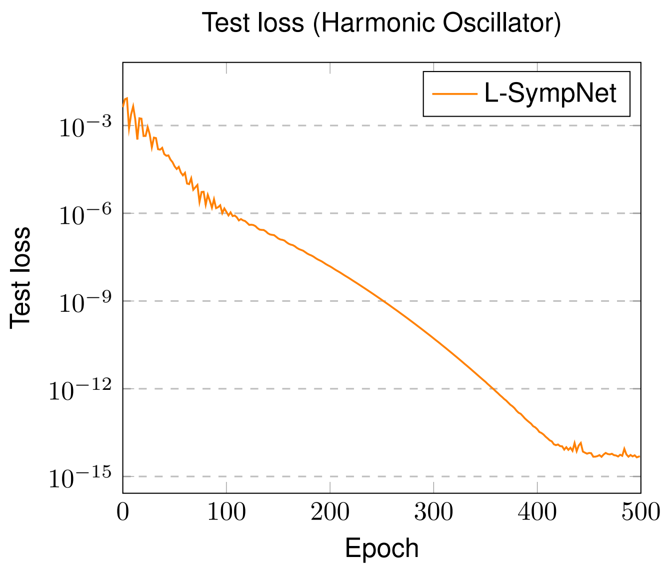

Figure 1: Test loss for Harmonic Oscillator (500 epochs). The L-SympNet consists out of nine alternating upper and lower linear layers. The training and test data are generated from random phase space samples in $\mathcal{D} = [-2, 2] \times [-2, 2]$. The training data size is $n_{\text{train}} = 40$ and the test data size is $n_{\text{test}} = 400$. The learning rate is set to $\gamma = 0.01$.

	Test loss (learning rate $\gamma = 0.01$)		
Architecture	Sigmoid	Tanh	ELU
FNN	$2.28 \cdot 10^{-6}$	$1.06 \cdot 10^{-6}$	$3.74 \cdot 10^{-6}$
LA-SympNet	$3.13 \cdot 10^{-8}$	$1.03 \cdot 10^{-6}$	$5.53 \cdot 10^{-7}$
N1-LA-SympNet	$7.15 \cdot 10^{-9}$	$4.99 \cdot 10^{-9}$	$3.79 \cdot 10^{-8}$
N2-LA-SympNet	$8 \cdot 10^{-8}$	$7.99 \cdot 10^{-9}$	$3.8 \cdot 10^{-8}$
G-SympNet	$3.01 \cdot 10^{-7}$	$9.78 \cdot 10^{-8}$	$3.92 \cdot 10^{-7}$
N1-G-SympNet	$4.83 \cdot 10^{-8}$	$5.78 \cdot 10^{-9}$	$3.92 \cdot 10^{-7}$
N2-G-SympNet	$2.13 \cdot 10^{-7}$	$1.3 \cdot 10^{-7}$	$5.75 \cdot 10^{-7}$

Table 2: Test losses for Simple Pendulum with training and test data generated from $\mathcal{D}_{\text{swing}}$ after 10^5 epochs.

Figure 2 and Table 2 show the training and test loss for the different architectures and activation functions with training and test data generated from $\mathcal{D}_{\text{swing}}$ and learning rate $\gamma = 0.01$. Note that incorporating normalization results in better training and test losses, and may also stabilize the learning process. The SympNets do not tend to overfit. The N1-G-SympNet with tanh activation function can even compete with the LA-SympNets, whereas without normalization the G-SympNet is not able to do so. In contrast, the fully-connected networks (FNN) tend to overfit, because they have a high test loss (right column) compared to the training loss (left column). Figure 3 and Figure 4 show phase plots, the total energy over time and the predicted position over time for the two best-performing architectures and the FNN. Note that no architecture is able to successfully predict the phase flow for a initial value outside the training data, i.e. $(q_0, p_0) \notin \mathcal{D}_{\text{swing}}$. The total energy is maintained with the SympNets, in contrast to the FNN.

Figure 5 shows the test losses if the training and test data is generated from $\mathcal{D}_{\text{swing+rotate}}$ instead. We perform the experiment with learning rate $\gamma = 0.01$ and faster learning rate $\gamma = 1.0$. Table 3 shows the final test loss for the first learning rate $\gamma = 0.01$. Table 4 shows the final test loss for the faster learning rate $\gamma = 1.0$. Compared to to the previous case with training data from $\mathcal{D}_{\text{swing}}$, the chosen activation function has a greater influence on model performance. The LARGE-LA-SympNet and the normalized variants LARGE-N1-LA-SympNet and LARGE-N2-LA-SympNet completely fail to learn the flow if the training data is generated from $\mathcal{D}_{\text{swing+rotate}}$, although their depth has been increased such that they have the same amount of parameters as the G-SympNets (see Table 1). So the in general good performance of the LA-SympNets in the previous case with training data from $\mathcal{D}_{\text{swing}}$ does not transfer to this case

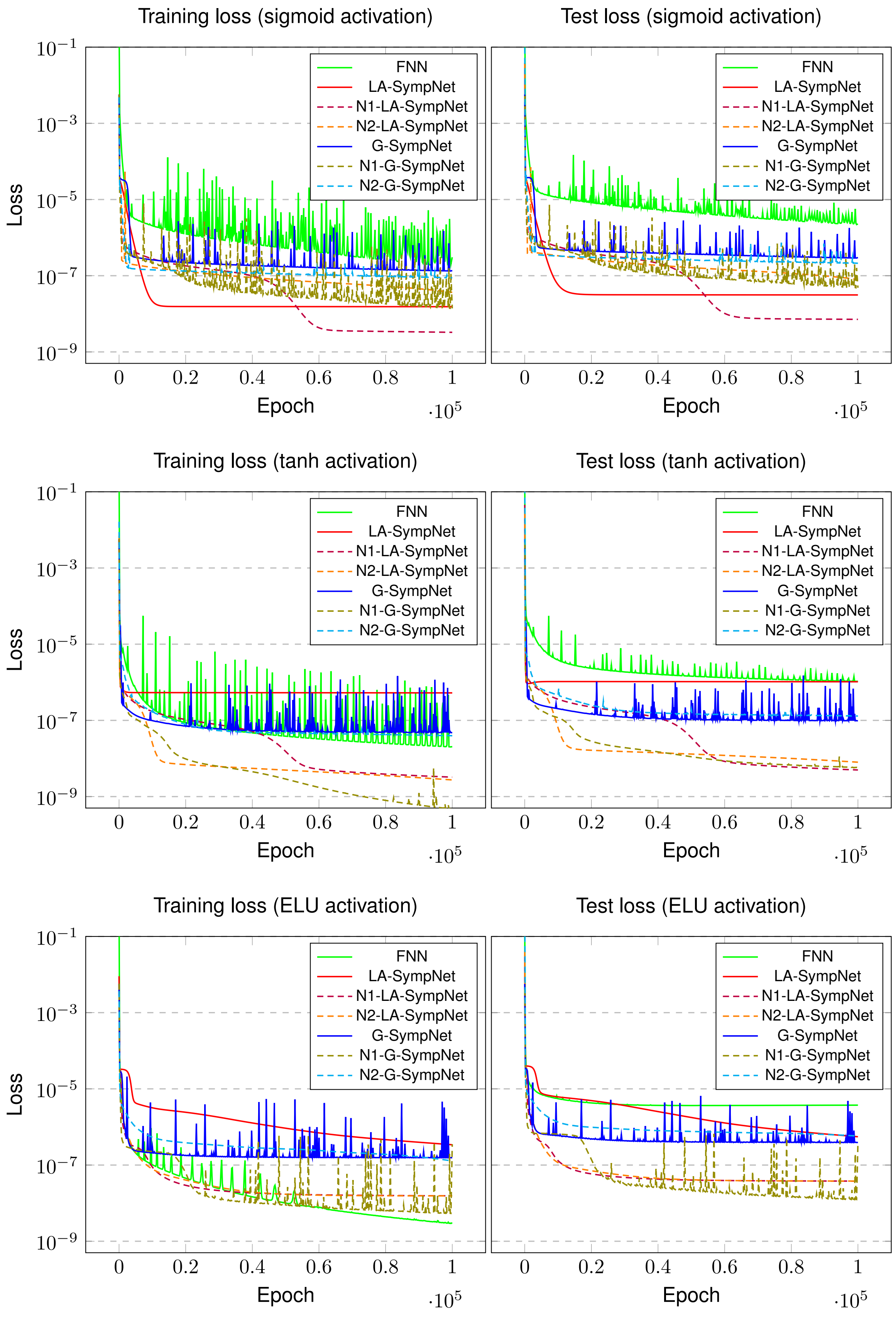

Figure 2: Training and test loss for simple pendulum with training and test data generated from $\mathcal{D}_{\text{swing}}$. (First row) sigmoid activation (Second row) tanh activation (Third row) ELU activation. Normalization leads to improved performance.

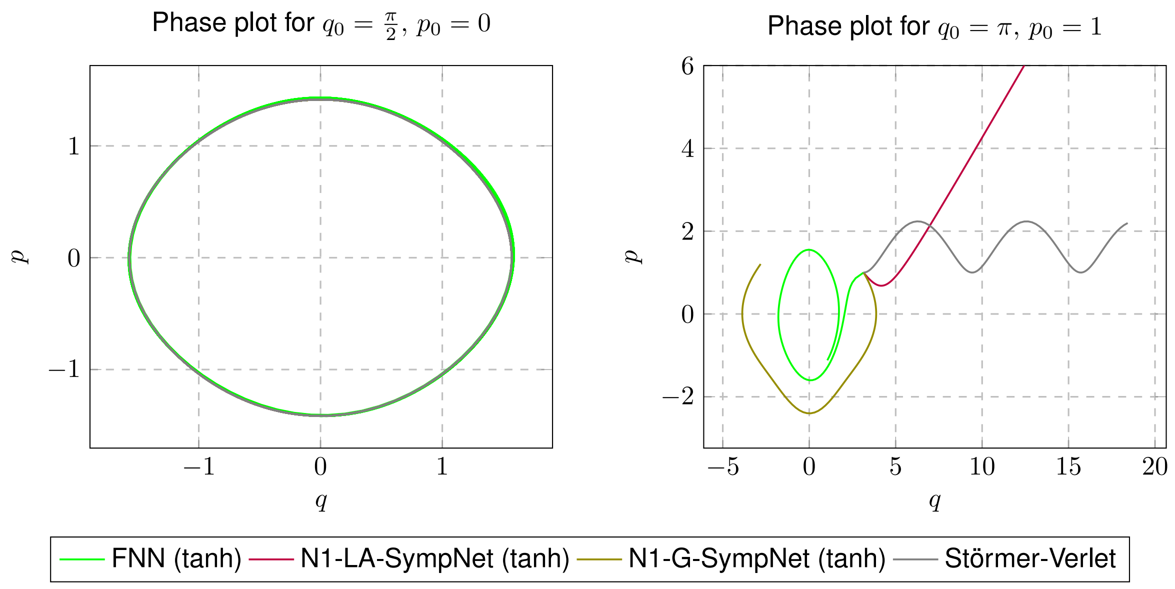

Figure 3: Phase plots for simple pendulum. Training and test data generated from $\mathcal{D}_{\text{swing}}$. (Left) The phase plots of the SympNets and the exact solution are indistinguishable. (Right) The neural networks fail to generalize for a $(q_0, p_0) \notin \mathcal{D}_{\text{swing}}$.

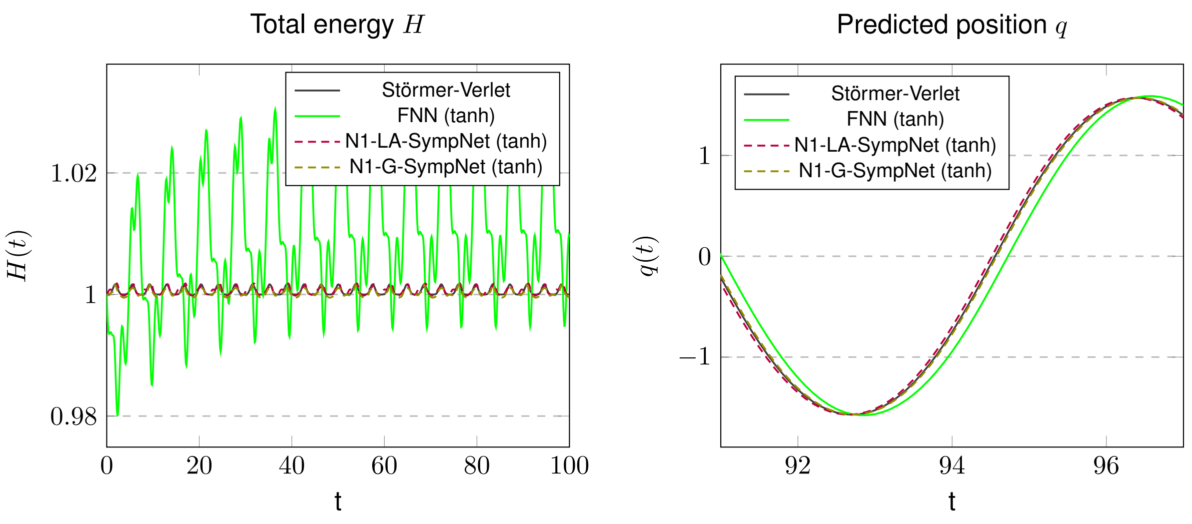

Figure 4: Total energy and position over time for simple pendulum for $q_0 = \frac{\pi}{2}$, $p_0 = 0$. Training and test data generated from $\mathcal{D}_{\text{swing}}$. The total energy is maintained with the SympNets, in contrast to the FNN. The prediction of the position over long time is better with SympNets than with the FNN.

	Test loss (learning rate $\gamma = 0.01$)		
Architecture	Sigmoid	Tanh	ELU
LARGE-FNN	$1.26 \cdot 10^{-5}$	$3.46 \cdot 10^{-4}$	$3.85 \cdot 10^{-5}$
LARGE-LA-SympNet	$2.5 \cdot 10^{-3}$	$2.21 \cdot 10^{-3}$	$2.59 \cdot 10^{-3}$
LARGE-N1-LA-SympNet	$2.5 \cdot 10^{-3}$	$2.5 \cdot 10^{-3}$	$2.49 \cdot 10^{-3}$
LARGE-N2-LA-SympNet	$2.5 \cdot 10^{-3}$	$2.5 \cdot 10^{-3}$	$2.49 \cdot 10^{-3}$
G-SympNet	$9.58 \cdot 10^{-7}$	$9.36 \cdot 10^{-6}$	$1.5 \cdot 10^{-3}$
N1-G-SympNet	$4.64 \cdot 10^{-5}$	$5.97 \cdot 10^{-6}$	$1.55 \cdot 10^{-3}$
N2-G-SympNet	$1.77 \cdot 10^{-5}$	$1.96 \cdot 10^{-5}$	$8.27 \cdot 10^{-4}$

Table 3: Test losses for Simple Pendulum with training and test data generated from $\mathcal{D}_{\text{swing+rotate}}$ after $3 \cdot 10^5$ epochs.

with training data from $\mathcal{D}_{\text{swing+rotate}}$. For training data from $\mathcal{D}_{\text{swing}}$, the SympNets do not work well with the ELU activation function, whereas the FNN with ELU activation still achieves a comparatively low loss for the slower learning rate $\gamma = 0.01$ (see the right plot in the first row of Figure 5).

For training and test data generated from $\mathcal{D}_{\text{swing+rotate}}$, normalization does not lead to a significant change in prediction performance, compared to what we observed with $\mathcal{D}_{\text{swing}}$. For $\mathcal{D}_{\text{swing+rotate}}$, the G-SympNet with sigmoid activation performs best overall (see left plot in first row of Figure 5). Still, normalization slightly improves the performance for the tanh and ELU activation (see mid and right plot in first row of Figure 5). Furthermore, normalization shows more robustness with respect to faster learning rates (see second row in Figure 5). The G-SympNet without normalization fails to converge for the faster learning rate $\gamma = 1.0$, in contrast to the N1-G-SympNet and N2-G-SympNet.

The generalized coordinate q is an angular coordinate for the simple pendulum. This is the reason why $\lim_{t\to\infty} q(t) = \infty$ in the rotating case. This circumstance makes learning the neural network hard, especially if the goal is to have a good prediction over a long time. The Störmer-Verlet scheme for the Simple Pendulum contains the sine function. Consequently, the neural networks have to learn a representation for multiple periods of the sine function, opposed to the swinging case where it suffices to learn the sine function on $[-\pi/2, \pi/2]$. This could be an explanation why the good performance of the LA-SympNets for the swinging case does not transfer to the rotating case, because learning a sine function with the structure of LA-SympNets is much more challenging as with the structure of the G-SympNets. With G-SympNets, a single gradient layer is able to represent the sine function on a compact set (see Jin et al. [9, Lemma 4]), whereas with the LA-SympNets the linear layers need to work together with the nonlinear activation layers to do so. Furthermore, intuitively, the large values of q in the rotating case could mask the swinging case, where $|q| \leq \pi/2$. This might explain why normalization does not lead to a similar performance improvement with training data from $\mathcal{D}_{\text{swing+rotate}}$ instead of $\mathcal{D}_{\text{swing}}$. This issue needs further investigation. We comment this issue in the Outlook (Section 8.2).

For almost all experiments, the first variant of the normalization layers (N1) performs better or equally good to the second variant of the normalization layers (N2). There are only two cases where the N2 normalization variant has a slightly better final test loss than the N1 normalization variant (see Figure 5, right plot in the first row and mid plot in the second row).

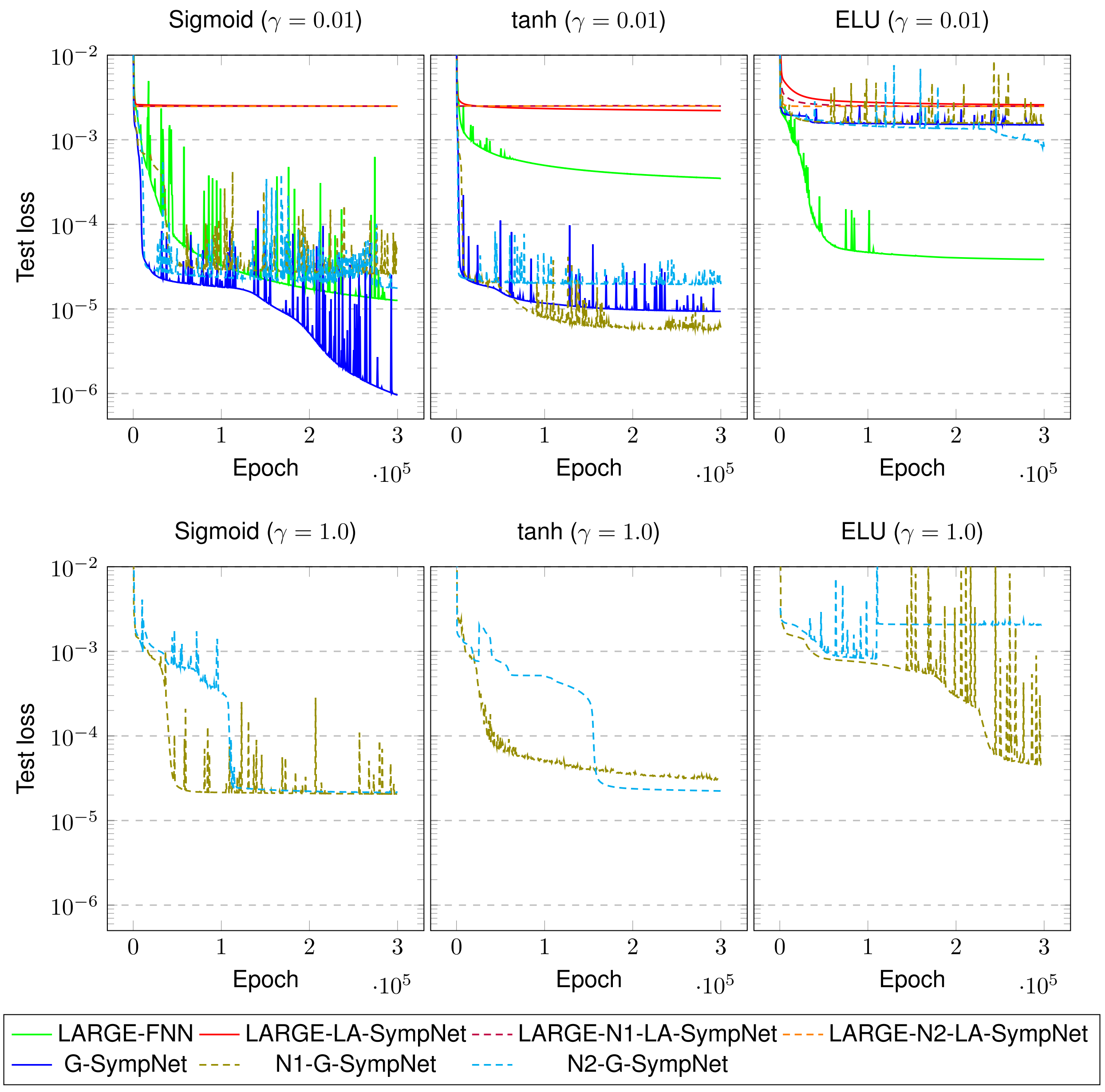

Figure 5: Test loss for simple pendulum with training and test data from $\mathcal{D}_{\text{swing+rotate}}$. (First row) Learning rate $\gamma = 0.01$. (Second row) Faster learning rate $\gamma = 1.0$. Test losses bigger than 10^{-2} and architectures that diverged during training are not displayed.

Architecture	Test loss (learning rate $\gamma = 1.0$) Sigmoid	Tanh	ELU
LARGE-FNN	64.25	12.91	$5.28 \cdot 10^6$
LARGE-LA-SympNet	NaN	NaN	NaN
LARGE-N1-LA-SympNet	NaN	NaN	NaN
LARGE-N2-LA-SympNet	NaN	NaN	NaN
G-SympNet	150.42	$7.18 \cdot 10^{-5}$	$4.75 \cdot 10^5$
N1-G-SympNet	$2.06 \cdot 10^{-5}$	$3.02 \cdot 10^{-5}$	$4.69 \cdot 10^{-5}$
N2-G-SympNet	$2.16 \cdot 10^{-5}$	$2.24 \cdot 10^{-5}$	$2.07 \cdot 10^{-3}$

Table 4: Test losses for simple pendulum after $3 \cdot 10^5$ epochs with training and test data generated from $\mathcal{D}_{\text{swing+rotate}}$ and faster learning rate $\gamma = 1.0$. NaN means that the test loss diverged. Only G-SympNets with normalization are able to handle the faster learning rate.

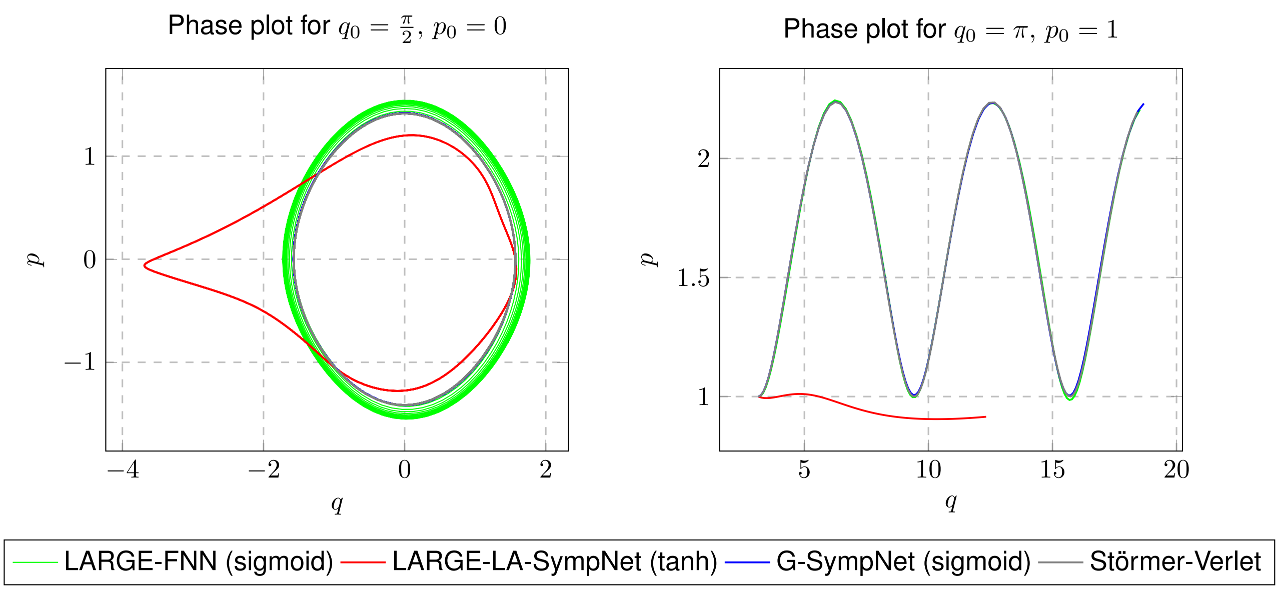

Figure 6: Phase plots for simple pendulum. Training and test data uniformly sampled from $\mathcal{D}_{\text{swing+rotate}}$. (Left) Only the G-SympNet is indistinguishable from the exact phase plot. The FNN does not have a closed orbit. (Right) The LA-SympNet fails to predict the phase plot for $q_0 = \pi$, $p_0 = 1$.

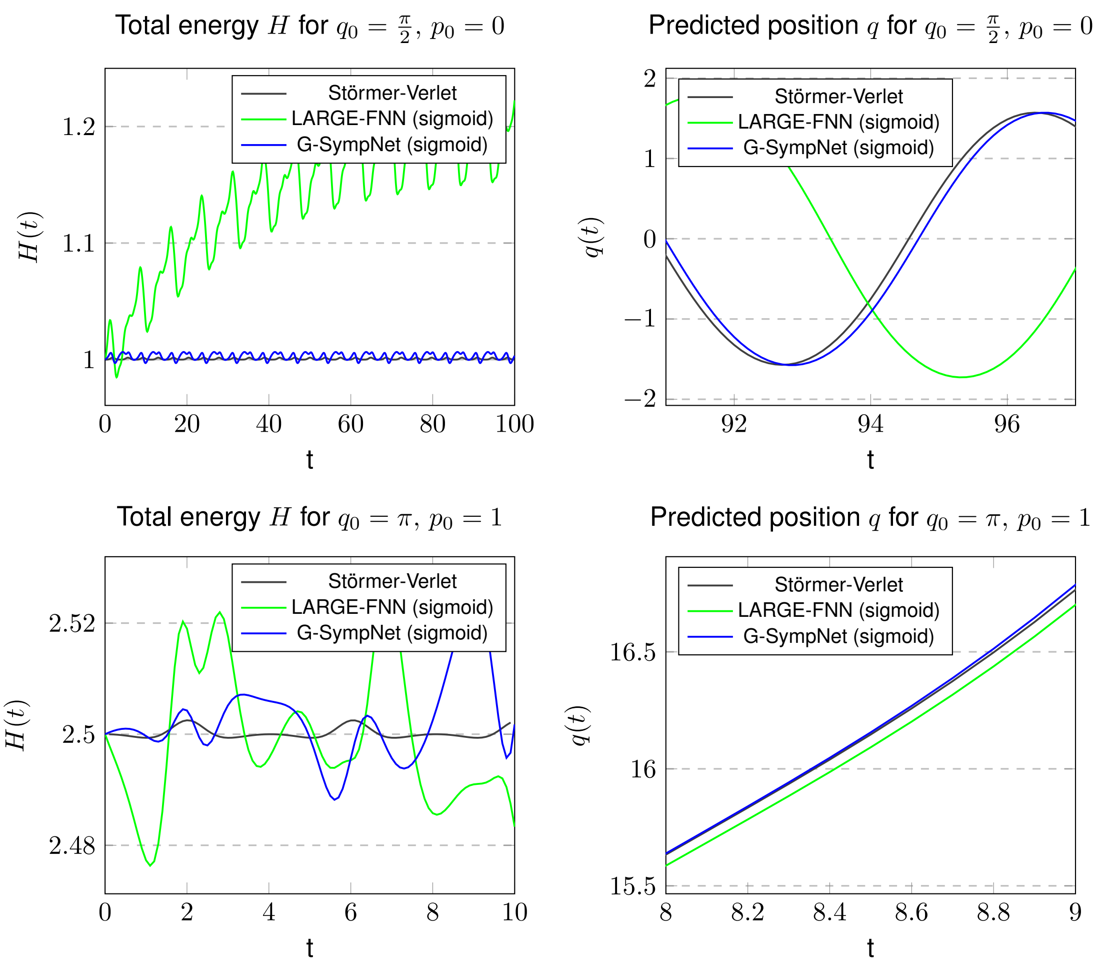

Figure 7: Training and test data generated from $\mathcal{D}_{\text{swing+rotate}}$. The learning rate is set to $\gamma = 0.01$. The plots show the total energy and position over time for the simple pendulum for $q_0 = \frac{\pi}{2}$, $p_0 = 0$ (first row) and $q_0 = \pi$, $p_0 = 1$ (second row). The total energy is better maintained with the SympNet for $q_0 = \frac{\pi}{2}$, $p_0 = 0$. The prediction of the position over long time is better with the SympNet.

7.2 High-dimensional systems

In this section, we work with a high-dimensional Hamiltonian system with $d \gg 1$ arising from a semi-discretized partial differential equation. Namely, the one-dimensional semi-linear wave equation with constant speed $c \in \mathbb{R}$, nonlinear term $g : \mathbb{R} \to \mathbb{R}$, domain length $l \in \mathbb{R}$ and domain $\Omega = [-l/2, l/2]$

$$\frac{\partial^2}{\partial t^2} u(t,x) = c^2 \frac{\partial^2}{\partial x^2} u(t,x) - g(u(t,x)) \quad \forall x \in \Omega,\, t \in I \subset \mathbb{R}$$

with initial conditions

$$\begin{aligned} u(t_0, x) &= u_0(x) \\ \frac{\partial}{\partial t} u(t_0, x) &= w_0(x) \quad \forall x \in \Omega \end{aligned}$$

and Dirichlet boundary conditions

$$u(t, -l/2) = u_{\text{left}},\, u(t, l/2) = u_{\text{right}} \quad \forall t \in I \subset \mathbb{R}$$

can be transformed into a Hamiltonian ODE system upon discretization with finite differences (see [1] and [12] for details). Here, $u(x,t) \in \mathbb{R}$ denotes the unknown time-dependent displacement field on the domain Ω.

Given an equally-spaced grid $\{x_i\}_{n=0}^{n+1} \subset \Omega$, $x_0 = -l/2$, $x_{n+1} = l/2$, with grid width Δx, initial values $y_0 = (\bar{q}_0^T, \bar{p}_0^T)^T \in \mathbb{R}^{2n}$, $\bar{q}_0 = (u_0(x_i))_{i=1}^n \in \mathbb{R}^n$, $\bar{p}_0 = (w_0(x_i))_{i=1}^n \in \mathbb{R}^n$, the resulting Hamiltonian system is given by

$$\begin{aligned} \dot{y}(t) &= \frac{J_{2n}}{\Delta x} \nabla H(y(t)) \quad \text{for } t \in I \subset \mathbb{R} \\ y(t_0) &= y_0 \end{aligned}$$

with Hamiltonian

$$H(q,p) = \sum_{i=1}^n \Delta x \left(\frac{1}{2} p_i^2 + \frac{c^2 (q_{i+1} - q_i)^2}{4 \Delta x^2} + \frac{c^2 (q_i - q_{i-1})^2}{4 \Delta x^2} + G(q_i) \right) \quad (q, p \in \mathbb{R}^n),$$

where $G'(u) = \int_0^u g(v) dv$, $q_0 := u_{\text{left}}$ and $q_{n+1} := u_{\text{right}}$. The generalized coordinates $q_i(t)$ then refer to $u(t, x_i)$ and the conjugate momenta $p_i(t)$ then refer to $\frac{\partial}{\partial t} u(t, x_i)$. The resulting Hamiltonian system has a large phase space dimension $d = n \gg 1$. It suggests itself to use convolution layers, since otherwise the parameter space would explode. It is also physically reasonable to use convolution layers, because the original PDE describes the system locally. Thus it makes sense to share parameters on the domain. The neural network architectures used in the experiments are listed in Table 5. With these architectures, we implicitly assume that the PDE coefficients do not depend on the position variable x. We also assume that the grid points are equidistant. Both conditions are met in our case with the wave equation. If one or both conditions do not apply, sharing the same parameters over the whole domain would not work anymore without adding further logic, because the current position x then would influence local system behavior.

In the nonlinear case, we also use that the nonlinearity g only depends on $u(t,x)$ at a single point x, and not on values on a neighborhood around x. This motivates why we set the kernel size and stride number to the same value for the convolution gradient layers in the CG-SympNets (see Table 5). This means that the (transposed) cross-correlation has no overlap, which implies that the spatial grid points do not influence each other inside the convolution gradient layers. So the intuition behind the proposed CG-Sympnet architecture is that the (linear) convolution layers learn the linear part, while the nonlinear convolution gradient layers learn the nonlinear part.

In our experiments with the wave equation, the phase spaces samples used to generate the training and test data originate from the trajectory starting from y_0, i.e. $x_i = y_{i-1} = \phi_{t,H}^{\mathsf{h}}(x_{i-1})$ for $i = 1, \ldots, n_{\text{train}} + n_{\text{test}}$. The data is generated for the fixed time $t = 0.01$ in all experiments. The first n_{train} samples go to the training data, the remaining samples to the test data. In that sense, the test loss is an indicator how well the model extrapolates in time. The training size $n_{\text{train}} \in \mathbb{N}$ is chosen such that it contains all data up to the total accumulated time $T = 1$, i.e. $n_{\text{train}} = 100$. The test data contains the trajectory samples for total accumulated time $T = 1$ to $T = 10$. All experiments are conducted with the learning rate $\gamma = 0.1$.

Architecture	Layers	Parameters
CNN	A standard purely linear convolutional neural network with four 1-dimensional convolution layers with kernel size 3 and constant zero-padding to keep the input dimension.	12
C-SympNet	A purely linear SympNet consisting out of four alternating upper and lower (symplectic) convolution layers with kernel size 3 and constant zero-padding, see Definition 20.	8
LARGE-CNN	A standard convolutional neural network consisting of four consecutive blocks of the following layers: 1-dimensional convolution layer with kernel size 3 and symmetric padding, 1-dimensional transposed convolution layer with kernel size 100 and stride 100, nonlinear activation layer and 1-dimensional convolution layer with kernel size 100 and stride 100.	809
CG-SympNet	A SympNet consisting of four consecutive blocks of the following two layers: upper (or lower) 1-dimensional symplectic convolution layer with kernel size 3 and symmetric padding (see Definition 20) followed by a upper (or lower) convolution gradient layer with kernel size 100 and stride 100 (see Section 5.3). The four blocks as a whole alternate between the upper and lower variants, i.e. a single block in itself contains either only the upper or only the lower variants.	1208
N1-CG-SympNet	Same as CG-SympNet, but with normalized convolution gradient layers (variant 1, see Section 5.4.2) instead of convolution gradient layers.	2008
N2-CG-SympNet	Same as CG-SympNet, but with normalized convolution gradient layers (variant 2, see Section 5.4.2) instead of convolution gradient layers.	2008

Table 5: Table of all neural network architectures for the high-dimensional experiments. The activation function used inside the nonlinear activation layers and the convolution gradient layers is either sigmoid, tanh or ELU and is explicitly stated in the respective experiment. If not stated otherwise, we use the FD basis to parametrize the symmetric kernels inside the symplectic convolution layers.

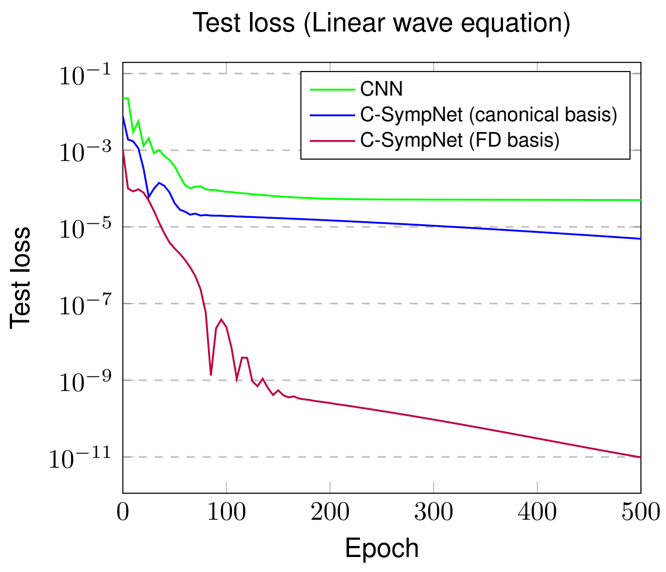

Figure 8: Test losses for the linear wave equation experiment after 500 epochs.

7.2.1 Linear wave equation

The first experiment is conducted with the linear wave equation, i.e. $G(u) = g(u) = 0$. We set $c = 1, l = 1$ and use 100 grid points for the discretization. The initial values are set to a bump function

$$\begin{aligned} u_0(x) &= \begin{cases} \frac{1}{2}\left(\exp\left(-\left(\frac{6x}{w}\right)^2\right) - \exp\left(-\left(\frac{6w/2}{w}\right)^2\right)\right) & : |x| \le w \\ 0 & : \text{else} \end{cases} \\ w_0(x) &= 0 \end{aligned}$$

with $w = l/4$, see first plot in Figure 9. The Dirichlet boundary values are set to $u_{\text{left}} = u_{\text{right}} = 0$. This corresponds to the linear wave equation transport problem with fixed ends. As with the Harmonic Oscillator, the resulting Hamiltonian ODE is linear and thus the flow is linear. Therefore, a linear model like the CNN and C-Sympnet architectures (see Table 5) should be able to represent the flow. The Dirichlet boundary values are encoded in the convolution layers of the C-SympNet via constant zero-padding.

The test losses for the CNN and C-SympNet architectures are depicted in Figure 8. We parametrize the symmetric kernels of C-SympNet with the canonical basis and with the finite-difference inspired (FD) basis. As it turns out, the C-SympNet with FD basis has a much lower test loss and converges much faster than the C-SympNet with the canonical basis. The naive non-physically informed linear CNN architecture performs worst as expected. The CNN architecture is naive, because each convolution layer of the CNN architecture uses the same kernel for the whole input (q, p).

The extrapolation performance of all architectures can be seen in Figure 9 (second and third plot). The C-SympNet with FD basis is indistinguishable from the Störmer-Verlet solution. The extrapolation performance of the C-SympNet with the canonical basis is still quite good, although its test loss is much larger than the test loss for the C-SympNet with the FD basis. The CNN solution gets unstable after a few time iterations.

The FD basis gives interpretability in terms of finite differences, which makes it a favorable choice in our context.

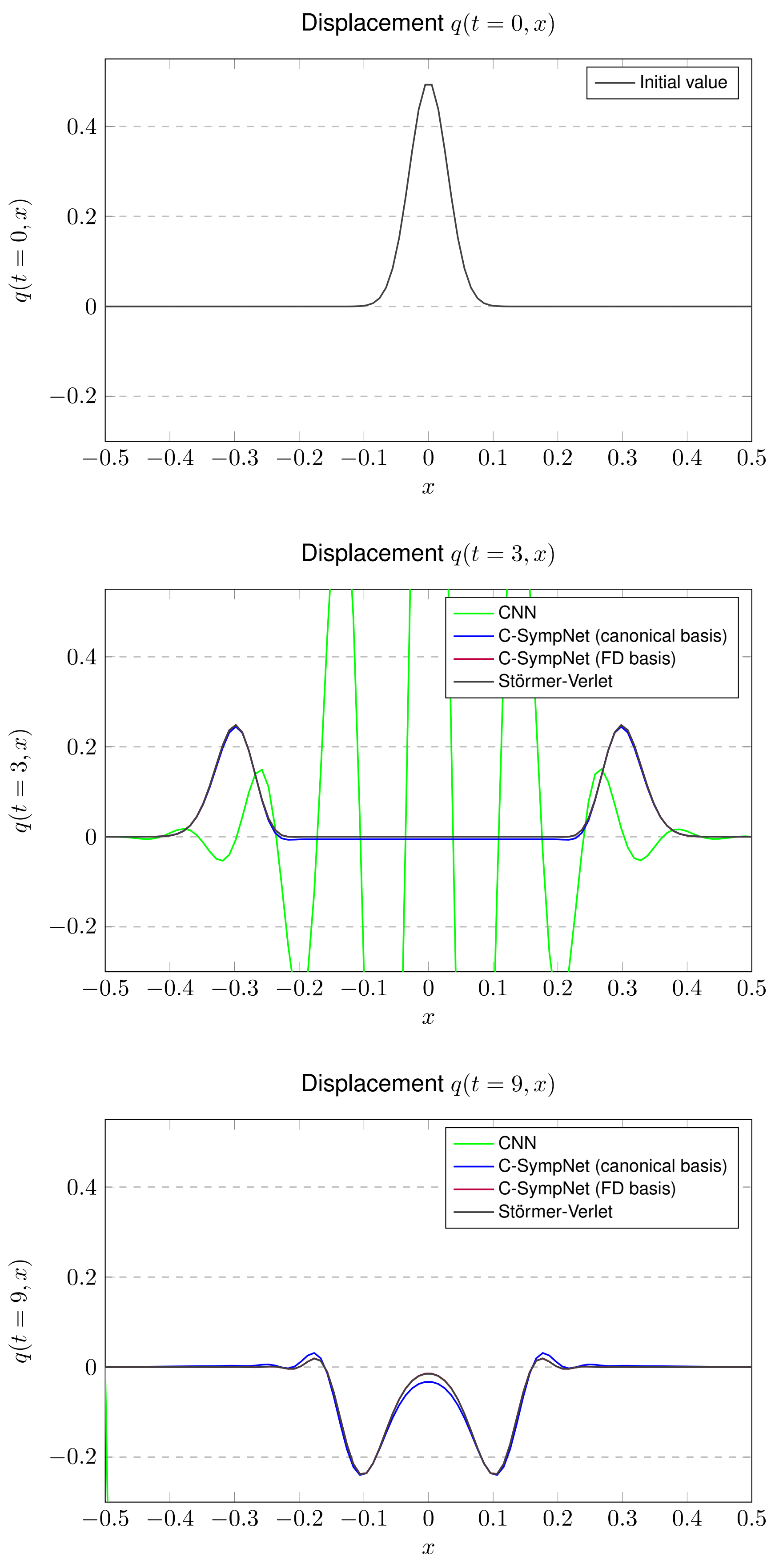

Figure 9: Displacement $q(t, x)$ for $t = 0$ (initial values) and $t = 3, 9$ (extrapolation) for the linear wave equation. The C-SympNet with FD basis is indistinguishable from the Störmer-Verlet solution.

	Test loss		
Architecture	Sigmoid	Tanh	ELU
CG-SympNet	$8.62 \cdot 10^{-10}$	$7.98 \cdot 10^{-10}$	$9.34 \cdot 10^{-10}$
N1-CG-SympNet	$1.13 \cdot 10^{-9}$	$9.35 \cdot 10^{-10}$	$6.36 \cdot 10^{-10}$
N2-CG-SympNet	$2.58 \cdot 10^{-7}$	$5.92 \cdot 10^{-7}$	$3.77 \cdot 10^{-6}$
LARGE-CNN	$2.6 \cdot 10^{-3}$	$3.26 \cdot 10^{-3}$	$1.72 \cdot 10^{-3}$

Table 6: Test losses for Sine-Gordon experiment after 2000 epochs.

7.2.2 Sine-Gordon

The Sine-Gordon equation is a semi-linear wave equation with $G(u) = 1 - \cos(u)$, $g(u) = \sin(u)$ and $c = 1$. One can show that the Sine-Gordon equation has the solution

$$u_{\text{sol}}(t, x) = 4 \arctan \left[\exp \left(\frac{x - x_0 - vt}{\sqrt{1 - v^2}} \right) \right].$$

For our experiment, we set the initial values

$$\begin{aligned} u_0(x) &= u_{\text{sol}}(0, x) \\ w_0(x) &= \frac{\partial}{\partial t} u_{\text{sol}}(0, x) \end{aligned}$$

with $v = 0.2$ and the Dirichlet boundary values to $u_{\text{left}} = 0$, $u_{\text{right}} = 2\pi$. Furthermore, we set $l = 50$ and use 2000 grid points for the discretization.

For this experiment, the flow is nonlinear and thus a purely linear model will not suffice. The LARGE-CNN and the CG-SympNets are nonlinear models. The Dirichlet boundary values u_{left} and u_{right} could again be embedded into the CG-SympNets via constant padding, either by specifying the padding values u_{left} and u_{right} before training or by making the padding values learnable during training. Instead, we use symmetric padding this time, because for the discretized solution it holds $u_{\text{left}} = q_0 \approx q_1$ and $q_n \approx q_{n+1} = u_{\text{right}}$ on the time domain in which we are interested.

The training and test losses for the LARGE-CNN and CG-SympNets for different activation functions are shown in Figure 10 and Table 6. Again, the naive LARGE-CNN performs worst as expected, because it applies the same kernels on the whole input vector (q, p). In this experiment, N1 normalization always outperforms N2 normalization by a great margin. The N1-CG-SympNet and the CG-SympNet (without normalization) perform nearly the same. The best performing architecture is the N1-CG-SympNet with ELU activation, see Table 6. Actually, N1 normalization only improved the performance for the ELU activation, while it led to a small decrease of performance for the sigmoid and tanh activation.

The extrapolation performance of the two best performing SympNet architectures and the LARGE-CNN can be seen at Figure 11. The LARGE-CNN gets unstable after a few time iterations, whereas the CG-SympNet and the N1-CG-SympNet still give a reasonable solution at $t = 9$.

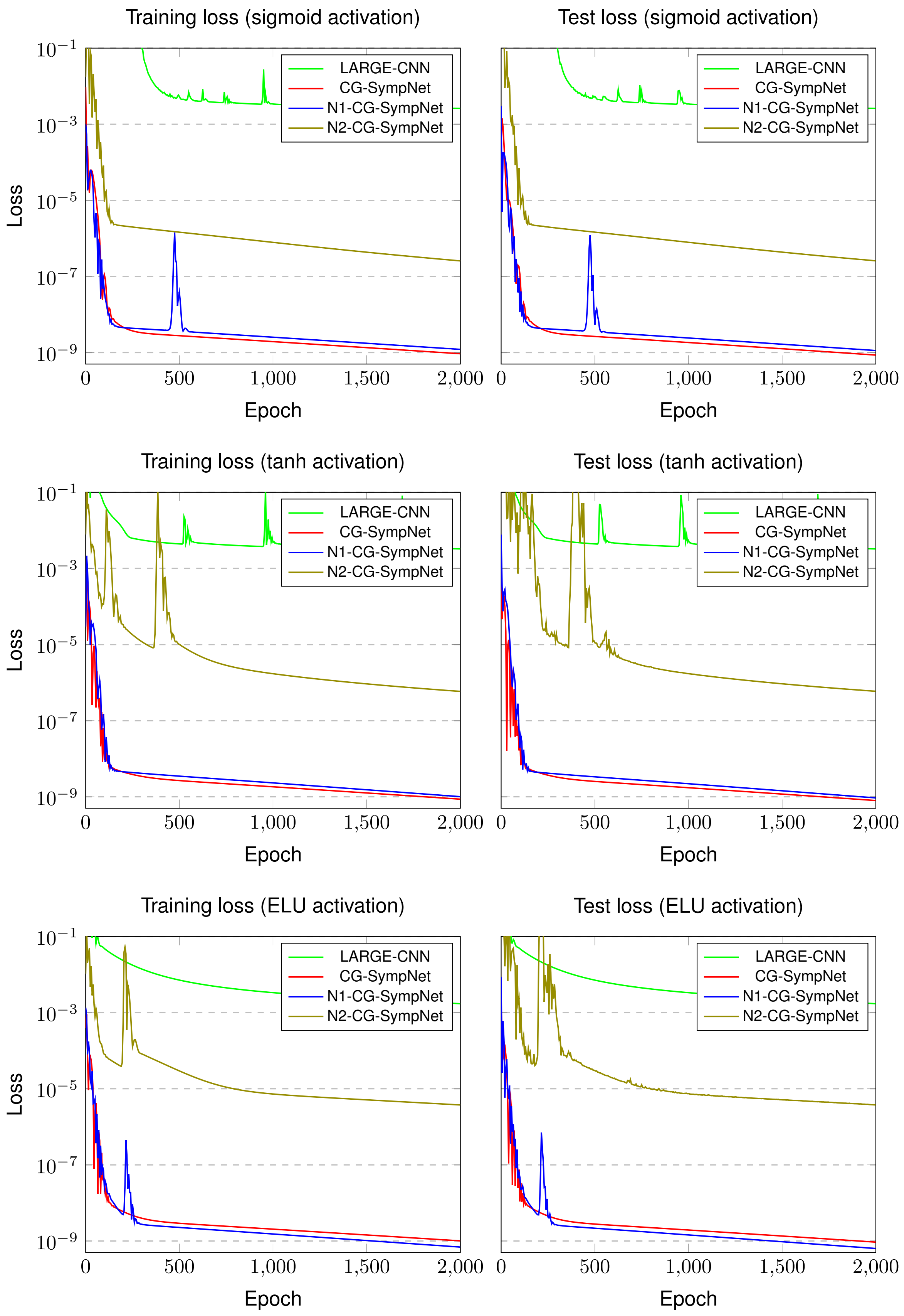

Figure 10: Training and test losses for the Sine-Gordon experiment (First row) sigmoid activation (Second row) tanh activation (Third row) ELU activation

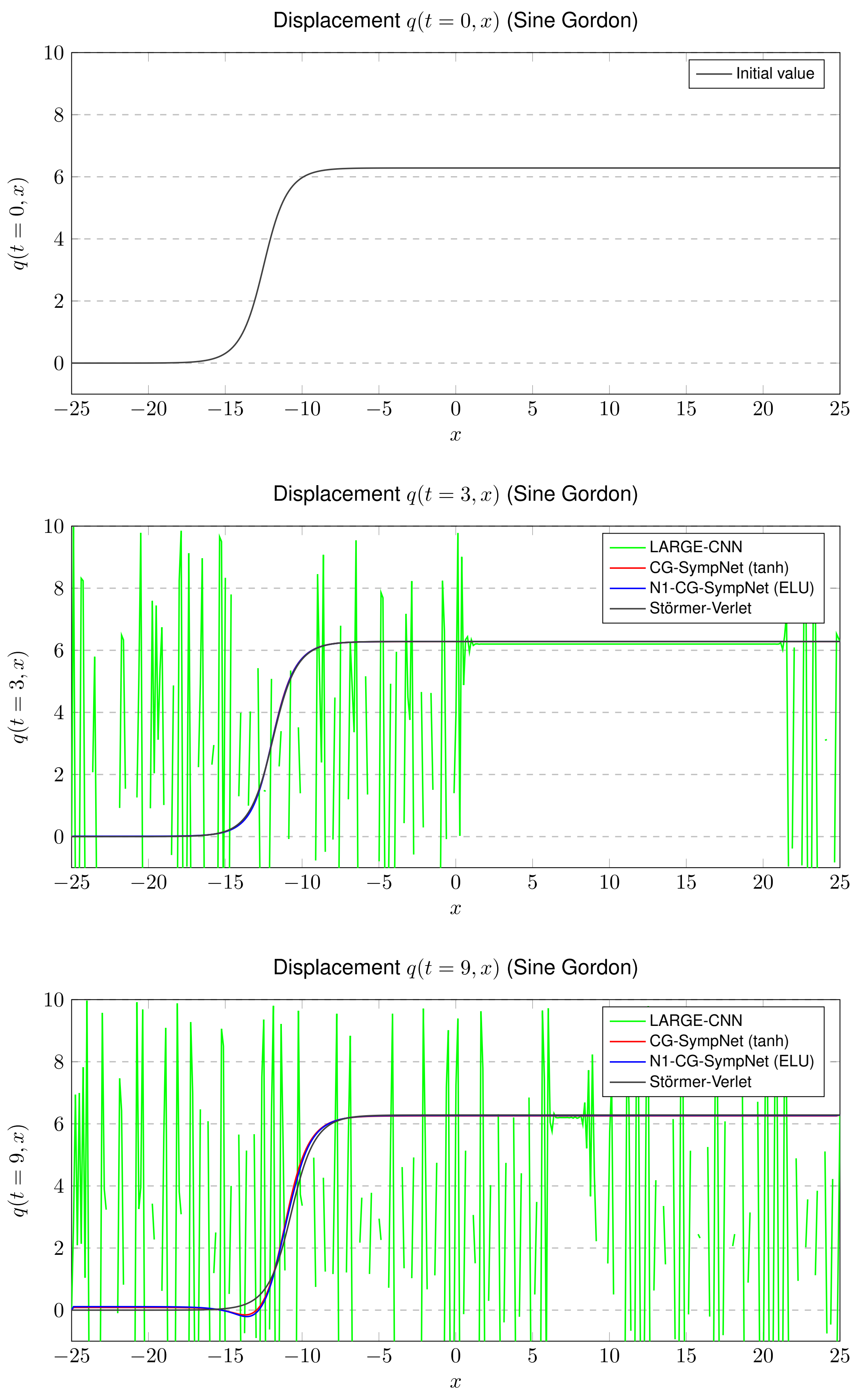

Figure 11: Displacement $q(t, x)$ for $t = 0$ (initial values) and $t = 3, 9$ (extrapolation) for the Sine-Gordon experiment.

8 Résumé

8.1 Summary and conclusion

The main contribution of our work is to transfer the concepts of convolution and batch normalization to SympNets, while preserving their symplecticity. Adding convolution makes SympNets applicaple for high-dimensional Hamiltonian systems, which typically arise from semi-discretized partial differential equations.

We have successfully reproduced the good performance of the LA-SympNets observed by Jin et al. for the simple pendulum swinging case. Furthermore, we managed to make the G-SympNet competitive with the LA-SympNet for the simple pendulum swinging case by proposing a method to add batch normalization to SympNets while preserving symplecticity. We have seen that the good performance of the LA-SympNet does not transfer to the rotating case of the simple pendulum, at least if no further logic is added to prevent the angular coordinate to increase monotonically in the rotating case. We saw that SympNets preserve the total energy in the simple pendulum swinging case and offer good long-term predictions compared to models which do not preserve the symplectic structure.

Ioffe and Szegedy observed that batch normalization leads to more robustness with respect to faster learning rates. We reproduced this observation with our special batch normalization variant for SympNets. We empirically noted that embedding physical knowledge into the structure of the model (like in our case symplecticity) reduces overfitting.

We applied SympNets with convolution to the semi-linear wave equation and successfully extrapolated single trajectories of the resulting Hamiltonian ODE.

We conclude that in our case it is a worthwile undertaking to structurally embed a priori knowledge of a system into a machine learning model, because it improves prediction performance and preserves potentially important properties of the original system, like in our case the preservation of total energy.

8.2 Outlook

As already noted by Jin et al., the unit triangular structure lends itself to be used in a recurrent neural network setting. For example, the neural network could form a loop by connecting the last layer of the neural network to the first layer again. When unfolding the loop, this can be interpreted as parameter sharing between several consecutive blocks of the neural network. If we interpret a single block as a numerical integrator, this is equivalent to applying the same numerical integrator several times, i.e. the problem to predict the flow for time t is divided in t/n smaller problems, where $n \in \mathbb{N}$ denotes the number how often the loop connection is executed. Thus, this might improve model performance. This direction could be explored further and tested empirically.

As also Jin et al. already noted, a unit triangular layer is always invertible, which means that every SympNet is reversible. So storing the intermediate values in the forward pass before computing the gradient is actually not necessary. This paves the way for memory-efficient implementations of SympNets.

The performance of the SympNets for the simple pendulum rotating case has not been as good as for the swinging case. We have already noted that the angular coordinate q of the simple pendulum monotonically increases when the pendulum rotates, which might explain this observation. A simple fix worth trying might be to take the angular coordinate q modulo 2π, or incorporate other appropriate coordinate transformations. Another possible approach could be to divide the phase space into multiple domains with a different (sub-)neural network responsible for each domain, and implement transition maps between the different domains. The simple pendulum phase plot is 2π-periodic on the q-axis. The universal approximation theorems however only apply for compact sets. It is an open research topic how to improve the extrapolation performance of neural networks for periodic functions [16].

We have only worked with one-dimensional convolution so far. It could be investigated how to generalize our proposal for two and three dimensions. This would add even more application areas, as it would allow to predict the flow for two and three dimensional systems in space. Besides, it could be investigated how to enable discretizations with non-equidistant grids.

So far, there only exists universal approximation theorems for SympNets for r-finite activation functions. Universal approximation theorems for other activation functions, for example the continously differentiable ELU activation function, is still an open topic to be explored.

We believe that the idea to intrinsically embed a priori knowledge about a system into a machine learning model is very promising and should be further explored also in other contexts.

Appendices

A Implementation

The symplectic neural network was implemented in Python via the PyTorch neural network library [11]. We used the PyTorch Tensorboard extension to get live monitoring of the neural networks during training and for experiment tracking and comparison (`https://pytorch.org/docs/stable/tensorboard.html`). We used Docker containers for a reproducible development and execution environment. The source code has been managed with Git version control and hosted in GitLab. We used the Visual Studio Code Remote SSH extension for comfortable editing of source code files on a remote cluster from the local development machine (`https://code.visualstudio.com/docs/remote/ssh`). We refer to the supplemented source code for instructions how to run the experiments.

B Declaration of authorship

I hereby certify

1. that this thesis has been composed by me and is based on my own work, unless stated otherwise,
2. that all direct or indirect sources used are acknowledged as references and all extracts from work of others, either verbatim or in spirit, are stated as such,
3. that neither the thesis itself nor parts of this thesis have been part of another examination procedure,
4. that neither the thesis itself nor parts of this thesis have been published and
5. that all copies of this thesis, either digital or printed, coincide.

Therewith, this declaration of authorship is in accordance with the examination regulations from 22th July 2016 of the bachelor's program *Simulation Technology* of the University of Stuttgart.

Steffen Müller
Name

30.11.2020, Lorch [illegible]
Date, City, Signature

References

[1] T. Bridges. Numerical methods for Hamiltonian PDEs. *Journal of Physics A: Mathematical and General*, 39:5287, 04 2006. doi: 10.1088/0305-4470/39/19/S02.

[2] G. Deco and W. Brauer. Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures. *Neural Networks*, 8(4):525 – 535, 1995. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(94)00108-X. URL http://www.sciencedirect.com/science/article/pii/089360809400108X.

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618.

[4] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 15379–15389. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/26cd8ecadce0d4efd6cc8a8725cbd1f8-Paper.pdf.

[5] E. Hairer, C. Lubich, and G. Wanner. *Geometric Numerical Integration*. Springer-Verlag, 2006. doi: 10.1007/3-540-30666-8. URL https://doi.org/10.1007%2F3-540-30666-8.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/ioffe15.html.

[8] P. Jin, Y. Tang, and A. Zhu. Unit triangular factorization of the matrix symplectic group. *arXiv e-prints*, art. arXiv:1912.10926, Dec. 2019.

[9] P. Jin, Z. Zhang, A. Zhu, Y. Tang, and G. E. Karniadakis. SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems. *Neural Networks*, 132:166 – 179, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.08.017. URL http://www.sciencedirect.com/science/article/pii/S0893608020303063.

[10] B. Leimkuhler and S. Reich. *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005. doi: 10.1017/CBO9780511614118.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[12] L. Peng and K. Mohseni. Symplectic model reduction of Hamiltonian systems. *SIAM Journal on Scientific Computing*, 38(1):A1–A27, jan 2016. doi: 10.1137/140978922.

[13] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=ryQu7f-RZ.

[14] L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, Apr 2020. ISSN 1573-7683. doi: 10.1007/s10851-019-00903-1. URL https://doi.org/10.1007/s10851-019-00903-1.

[15] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 2483–2493. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf.

[16] L. Ziyin, T. Hartwig, and M. Ueda. Neural Networks Fail to Learn Periodic Functions and How to Fix It. *arXiv e-prints*, art. arXiv:2006.08195, June 2020.