3 A Generalized Framework of Neural Networks for Hamiltonian Systems

Based on the work by Philipp Horn, Veronica Saz Ulibarrena, Barry Koren, Simon Portegies Zwart, 2025, *Journal of Computational Physics*, Volume 521, Part 1, 113536. Adapted for this thesis.

ABSTRACT

When solving Hamiltonian systems using numerical integrators, preserving the symplectic structure may be crucial for many problems. At the same time, solving chaotic or stiff problems requires integrators to approximate the trajectories with extreme precision. Integrating Hamilton's equations to a level of scientific reliability such that the answer can be used for scientific interpretation, may lead to computationally expensive simulations. In some cases, a neural network can be a viable alternative to numerical integrators, offering high-fidelity solutions orders of magnitudes faster.

To understand the role of preservation of symplecticity in problems where neural networks are used, we analyze three well-known neural network architectures that include the symplectic structure inside the neural network's topology. Between these neural network architectures, many similarities can be found. This allows us to formulate a new, generalized framework for these architectures. In the generalized framework Symplectic Recurrent Neural Networks, SympNets, and HénonNets are special cases. Additionally, this new framework enables us to find novel neural network topologies by transitioning between the established ones.

We compare new Generalized Hamiltonian Neural Networks (GHNNs) against the already established SympNets, HénonNets, and physics-unaware multilayer perceptrons. This comparison is performed for the gravitational three-body problem. In order to perform a fair comparison, the hyperparameters of the different neural networks are chosen such that the prediction speeds of all four architectures are the same during inference. A special focus lies on the capability of the neural networks to generalize outside the training data. The GHNNs outperform all other neural network architectures for the problem considered.

3.1 Introduction

Specialized neural network architectures for scientific machine learning are still scarce. In most cases, neural network architectures from different areas of machine learning are used for scientific applications without any adaptations. If prior knowledge is included, it is usually done through additional terms to the loss function. This is the case of Physics-Informed Neural Networks (PINNs) (Raissi et al. (2019b)) and all of its variants.

Imposing physics constraints through the loss function does have advantages. For example, it does not require extensive expertise in neural network architectures and can be done quickly. It proves to be very useful in cases with limited data to train the neural network. However, including prior knowledge only through the loss function also has its drawbacks. Adding an additional loss term introduces a new hyperparameter, a parameter to determine the weighting between the error on the data and how far the neural network is allowed to deviate from the physics constraint. Furthermore, this constraint is only a soft one, which means that the neural network is not strictly constrained, it may deviate from it. Also, the neural network's loss only increases if the constraint is not obeyed at certain predetermined inputs called collocation points, with which the network is trained. The constraint is not enforced globally but point-wise only, and increasing the number of points slows down the training.

Including prior knowledge into the topology of the neural network requires the development of a new topology for every structure one wants to preserve. This is more labor-intensive and may break the approximation properties of neural network architectures for which we understand their behavior. However, if such a specialized topology can be found, the physics constraints are hard ones and are actually enforced everywhere; inside and outside the training data. Furthermore, it does not require the introduction of new hyperparameters.

3.1.1 Hamiltonian Systems

We focus on structure-preserving neural networks for Hamiltonian systems. While the neural networks analyzed in this work are applicable to all Hamiltonian systems, our numerical experiment is focused on a case of Hamiltonian systems from mechanics: the gravitational three-body problem. The state $\vec{s}$ of a Hamiltonian system is described by its generalized momentum $\vec{p}$ and generalized position $\vec{q}$. The system's dynamics are described by Hamilton's equations:

$$\begin{pmatrix} \dot{\vec{p}} \\ \dot{\vec{q}} \end{pmatrix} = -J\nabla\mathcal{H}(\vec{p},\vec{q}), \quad J = \begin{pmatrix} 0 & I_d \\ -I_d & 0 \end{pmatrix}, \quad \vec{p},\vec{q} \in \mathbb{R}^d, \tag{3.1}$$

with $\mathcal{H} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ being the Hamiltonian of the system and I_d the d-dimensional identity matrix.

The data the neural networks are trained on consists of multiple trajectories with different initial states. The trajectory data consists of multiple states $\vec{s}_i$ along the individual trajectories that are a fixed timestep h apart from each other. The feature and label pairs of the training data are the states $\vec{s}_i$ and $\vec{s}_{i+1}$, respectively. Hence, the neural networks learn an approximation of the flow map of the system $\varphi_h : \vec{s}(t) \mapsto \vec{s}(t+h)$. This flow map can also be approximated numerically by integrators if the Hamiltonian $\mathcal{H}$ of the system is known.

The motivation to use neural networks for Hamiltonian systems instead of numerical integrators can be diverse. One reason could be that the Hamiltonian of the system is unknown, only data of trajectories is observed. For other Hamiltonian systems, the use of numerical integrators can be computationally expensive. This can be the case because calculating the Hamiltonian is costly or because extremely small timestep sizes are required. For example in Breen et al. (2020b) the equations of motion are stiff and chaotic and therefore an accurate approximation requires small timesteps. In all these cases, the use of neural networks can be a viable alternative. However, for the case where the Hamiltonian of the system is known and one is interested in a faster alternative to numerical integrators, the generation of the training data and the training time have to be taken into account for the total computational cost. Therefore, a neural network surrogate can only provide a reasonable speedup if the number of trajectories that have to be calculated is much larger than the training set. For application to the three-body problem shown in Section 3.5, the Hamiltonian is known and the data is created by a numerical integrator.

It is well established that symplectic integrators can be superior to non-structure-preserving numerical integrators when applied to certain Hamiltonian systems. Especially long-term energy preservation and increased stability often are the two major benefits of symplectic integrators (Hairer et al. (2006)). In symplectic integrators, the numerical flow map $\Phi_h : \vec{s}_i \mapsto \vec{s}_{i+1}$ defined by the integrator has the same geometric structure as the real flow map. This structure is given by:

$$\left(\frac{\partial \varphi_h}{\partial \vec{s}}\right)^T J \, \frac{\partial \varphi_h}{\partial \vec{s}} = J, \quad \vec{s} = \begin{pmatrix} \vec{p} \\ \vec{q} \end{pmatrix} \in \mathbb{R}^{2d}. \tag{3.2}$$

This symplectic property is the structure that is preserved in most structure-preserving neural networks for Hamiltonian systems. Some benefits of this structure have been studied thoroughly for numerical integrators. However, symplecticity in numerical integrators is not always fully understood and the benefits cannot be carried over directly to neural networks.

In this work, we give a short overview of already published neural networks for Hamiltonian systems in Section 3.2. In Section 3.3, we unite three different neural network architectures in one generalized framework called Generalized Hamiltonian Neural Networks (GHNNs). We are able to do this by presenting a new point of view on SympNets and HénonNets. This generalized framework not only includes Symplectic Recurrent Neural Networks (SRNNs), SympNets, and HénonNets, but also new neural network topologies that lie beyond the capabilities of these three architectures. At the same time, the GHNNs preserve the good theoretical properties of the SympNets, and HénonNets. In Section 3.4, we highlight important aspects related to the implementation of GHNNs. By comparing GHNNs against SympNets, HénonNets, and non-structure-preserving neural networks in Section 3.5, we show the superior performance of a GHNN topology that cannot be categorized as SRNN, SympNet, or HénonNet. For completeness, in Section 3.5, we also compare our GHNNs with PINN-type neural networks. All neural networks are trained with topologies that offer similar prediction speeds to achieve a fair comparison.

3.2 Previously introduced NNs for Hamiltonian Systems

Many specialized neural network architectures for data from Hamiltonian systems have already been proposed (Greydanus et al. (2019b); Chen et al. (2020); Jin et al. (2020a); Burby et al. (2021); Xiong et al. (2021)), each of them promising to surpass the previously introduced ones in numerical experiments. Moreover, each architecture tries to remove restrictions imposed on the earlier architectures. SRNNs improve on HNNs by removing the need for time derivatives in the training data and by enforcing symplecticity. SympNets remove the restriction to separable Hamiltonian systems of SRNNs and prove a universal approximation theorem. HénonNets claim to achieve a higher accuracy than SympNets by learning Hénon maps instead of unit triangular updates.

Instead of only adding to this collection of neural networks, we would like to present a generalizing framework that combines three of these types of neural networks, namely: SRNNs, SympNets, and HénonNets, in a single architecture while allowing the creation of new neural network topologies that lie beyond the capabilities of these other architectures. To introduce this framework, we first have a look at the already published architectures.

3.2.1 Hamiltonian Neural Networks

The first approach to neural networks specifically designed for Hamiltonian systems was presented in Greydanus et al. (2019b). These Hamiltonian Neural Networks (HNN) try to learn the Hamiltonian of a system using a multilayer perceptron (MLP). To learn the Hamiltonian, not only data on positions and momenta is needed but also data on its derivatives. Then a mean-square loss can be formulated by iterating over all N datapoints:

$$\mathcal{L}_{\text{HNN}} = \frac{1}{N}\sum_{i=1}^{N}\left|\left|\begin{pmatrix}\dot{\vec{p}}_i\\ \dot{\vec{q}}_i\end{pmatrix} + J\nabla\mathcal{H}_\theta(\vec{p}_i,\vec{q}_i)\right|\right|_2^2. \tag{3.3}$$

For inference, any stable and consistent numerical integrator can be chosen to calculate $\vec{s}_{i+1}$ from $\vec{s}_i$. Also, the step size can be arbitrary since it is not part of the training. See Figure 3.1, for a schematic of a Hamiltonian Neural Network.

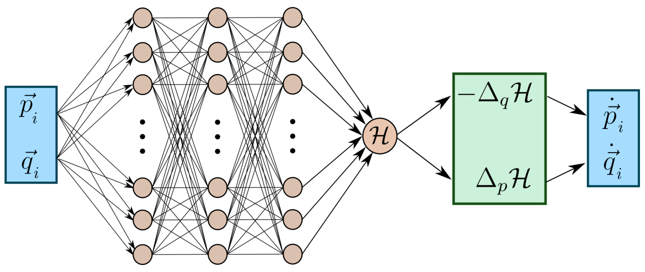

Figure 3.1: Schematic of a Hamiltonian Neural Network. The input and output layers are colored blue. All trainable parameters are in the multilayer perceptron (orange). In the green layer the gradients of the multilayer perceptron with respect to its inputs are calculated using automatic differentiation.

Hamiltonian Neural Networks require additional data and the symplectic structure is only preserved if a symplectic integrator is used during inference. Furthermore, not only an approximation error in the learned Hamiltonian is accumulated, but also an additional numerical error. For these reasons, we are not going to analyze HNNs in more detail. Instead, we take a closer look at the Symplectic Recurrent Neural Networks that are heavily inspired by HNNs but without the above-mentioned shortcomings.

3.2.2 Symplectic Recurrent Neural Networks

Symplectic Recurrent Neural Networks (SRNNs) were introduced in Chen et al. (2020). An SRNN learns a separable Hamiltonian

$$\mathcal{H}_\theta(\vec{p},\vec{q}) = T_{\theta_1}(\vec{p}) + U_{\theta_2}(\vec{q}), \tag{3.4}$$

parameterized by two independent neural networks, one for the kinetic energy T_{θ_1} and one for the potential energy U_{θ_2}. The main reason behind the use of recurrent neural networks is to compensate for noisy data by using multiple combined timesteps to calculate the error. However, the important improvement over Hamiltonian Neural Networks is not reflected in the name of this architecture and is also present if SRNNs are used with simple multilayer perceptrons instead of recurrent neural networks. The important improvement is that in SRNNs the integrator is always symplectic and is embedded in the neural network's topology itself. This means that the SRNN predicts the state of the Hamiltonian system after a timestep of size h, instead of only the derivatives of the state at the current time. Therefore, a loss function can be defined with the data as introduced in Section 3.1.1 and no additional information on the derivatives is needed. The error is then backpropagated through the integrator into the two neural networks composing the learned Hamiltonian (see Figure 3.2).

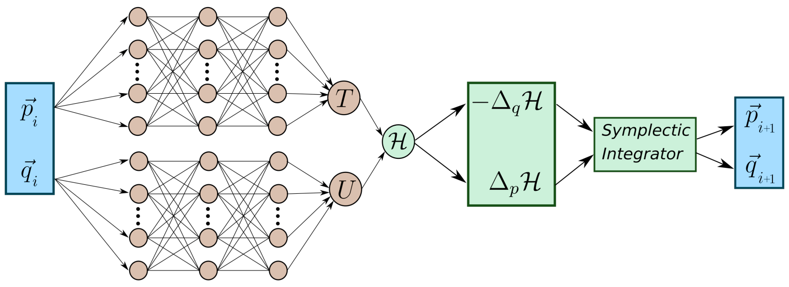

Figure 3.2: Schematic of a Symplectic Recurrent Neural Network with two independent multilayer percetrons (orange) for the learned Hamiltonian. Also, the symplectic integrator is included in the neural network itself. There are no trainable parameters in the green parts of the neural network. Instead, only predefined mathematical operations are performed.

A Hamiltonian Neural Network learns a function that approximates the real Hamiltonian of the system. The approximation error together with the numerical error of the integrator, which is used for inference, add up. The SRNN directly learns the flow map of the system. As noted in Chen et al. (2020), the learned Hamiltonian inside the SRNN adjusts to the symplectic integrator used during training and compensates for the numerical error of the integrator. Therefore, a convergence to a low value of the loss function does not imply that the learned Hamiltonian inside the neural network converges to the Hamiltonian underlying the data. Because of this behavior, the learned Hamiltonian only provides limited interpretability and it should not be used in combination with any integrator or timestep other than the one used during training.

3.2.3 SympNets

As a third approach we consider SympNets. SympNets have been introduced independently of the idea of learning a Hamiltonian and instead focus directly on learning symplectic maps (Jin et al. (2020a)). This is done by concatenating so-called upper and lower unit triangular updates (f_{up} and f_{low}, respectively):

$$f_{\text{up}}\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix} := \begin{pmatrix}\vec{p}+\nabla V(\vec{q})\\ \vec{q}\end{pmatrix} = \begin{bmatrix}I & \nabla V(\cdot)\\ 0 & I\end{bmatrix}\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix}, \tag{3.5}$$

$$f_{\text{low}}\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix} := \begin{pmatrix}\vec{p}\\ \vec{q}+\nabla V(\vec{p})\end{pmatrix} = \begin{bmatrix}I & 0\\ \nabla V(\cdot) & I\end{bmatrix}\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix}. \tag{3.6}$$

In the right-hand side of equations (3.5) and (3.6) we adopted the short-hand notation of Jin et al. (2020a). This is a slight abuse of matrix vector multiplication, which is helpful to obtain a compact notation of concatenated unit triangular updates. In every update, also called module, a different parameterized gradient ∇V is learned (Figure 3.3).

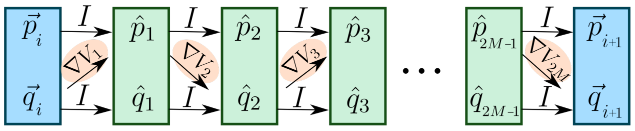

Figure 3.3: Schematic of a SympNet with $2M$ modules alternating between upper and lower modules. In orange it is highlighted where the trainable parameters are located.

Different possible parameterizations of ∇V can be introduced, the most general update being the gradient module (here as upper gradient module):

$$\nabla V(\vec{q}) := K^T \text{diag}(\vec{a})\sigma(K\vec{q}+\vec{b}). \tag{3.7}$$

In a gradient module, the trainable parameters are: a weight matrix $K \in \mathbb{R}^{n\times d}$, a bias $\vec{b} \in \mathbb{R}^n$ and a scale factor $\vec{a} \in \mathbb{R}^n$. The width n of this module can be freely chosen. Additionally, a nonlinearity is present in the form of an activation function σ. SympNets composed of only gradient modules are called G-SympNets.

For these SympNets, a universal approximation theorem can be proven (Jin et al. 2020a, Theorem 5). In this theorem, the width and depth of the G-SympNet are not constrained. This means that, in contrast to a multilayer perceptron where one hidden layer is enough to approximate a large class of functions, a SympNet might require a large number of also wide modules to approximate a given symplectic map (Hornik (1991)). The proof of the universal approximation capabilities of SympNets is based on the fact that four unit triangular updates can form a Hénon-like map (defined below) and that any symplectic map can be approximated by a concatenation of these Hénon-like maps (Turaev (2002)).

3.2.4 HénonNets

As an improvement to the SympNets, the HénonNets were published in Burby et al. (2021). Instead of learning the unit triangular updates, they directly focus on learning Hénon-like maps:

$$h[V,\vec{\eta}]\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix} := \begin{pmatrix}-\vec{q}+\nabla V(\vec{p})\\ \vec{p}+\vec{\eta}\end{pmatrix} = \begin{bmatrix}\nabla V(\cdot) & -I\\ I & 0\end{bmatrix}\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix} + \begin{pmatrix}0\\ \vec{\eta}\end{pmatrix}. \tag{3.8}$$

In order to create a Hénon layer, the same Hénon-like map is iterated four times. The reasoning behind this choice is the universal approximation theorem for Hénon-like maps (Turaev (2002)):

$$l[V,\vec{\eta}]\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix} := h[V,\vec{\eta}]\circ h[V,\vec{\eta}]\circ h[V,\vec{\eta}]\circ h[V,\vec{\eta}]\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix}. \tag{3.9}$$

In one Hénon layer, the function V and the vector $\vec{\eta}\in\mathbb{R}^d$ are learned. As a parameterization for V any multilayer perceptron can be used. See Figure 3.4, for a schematic of a single Hénon layer.

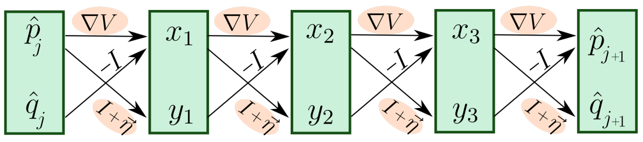

Figure 3.4: Schematic of a single Hénon layer.

Since the HénonNet architecture is based on the same theory as SympNets, they are universal approximators of symplectic maps as well (Burby et al. (2021)). Because HénonNets directly focus on learning Hénon-like maps and because of the greater flexibility in the choice of the function V, HénonNets are able to achieve a lower loss than SympNets with fewer training epochs. In Burby et al. (2021), different numerical experiments are performed to corroborate this statement.

3.3 Generalized Framework

When analyzing the three approaches in more detail, one can see that the SRNNs, SympNets, and HénonNets are actually more similar than they seem to be. This similarity allows us to introduce a generalized framework unifying all three neural network architectures. All possible SRNNs, SympNets, and HénonNets are included in this generalized framework. But also, new symplectic neural network topologies can be derived from it.

A major step to show the similarities between Symplectic Recurrent Neural Networks and SympNets was made in Horn et al. (2022). It shows that SympNets can be seen as a concatenation of specific SRNNs. Every two gradient modules learn a separable Hamiltonian, which is parameterized by two multilayer perceptrons with one hidden layer, one multilayer perceptron for the kinetic and one for the potential energy. With this learned Hamiltonian, one step of a Symplectic Euler (SE) method is performed. The next two layers learn a different Hamiltonian, perform another Symplectic Euler step and so on. This new point of view on SympNets is illustrated in Figure 3.5.

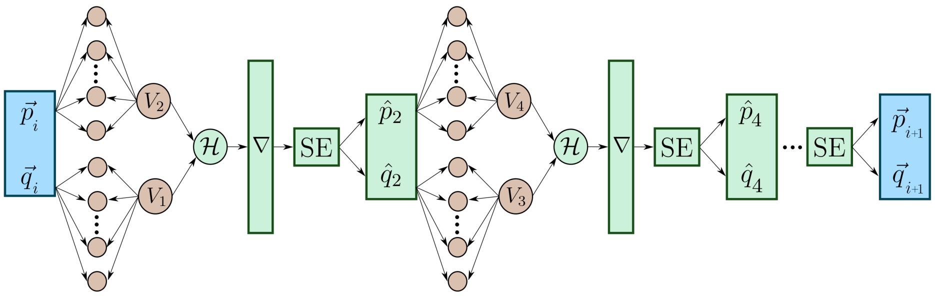

Figure 3.5: A SympNet as a concatenation of Symplectic Recurrent Neural Networks.

A similar equivalence exists between HénonNets and SympNets and therefore also between HénonNets and SRNNs. We can show that one Hénon layer is equivalent to a concatenation of four specific unit triangular updates:

$$l[V,\vec{\eta}]\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix}=\begin{bmatrix}I & \nabla V_4(\cdot)\\ 0 & I\end{bmatrix}\begin{bmatrix}I & 0\\ \nabla V_3(\cdot) & I\end{bmatrix}\begin{bmatrix}I & \nabla V_2(\cdot)\\ 0 & I\end{bmatrix}\begin{bmatrix}I & 0\\ \nabla V_1(\cdot) & I\end{bmatrix}\begin{pmatrix}\vec{p}\\ \vec{q}\end{pmatrix}, \tag{3.10}$$

with:

$$V_1 = f\circ V\,, \qquad \text{where } f(\vec{s}) = -\vec{s}, \tag{3.11a}$$
$$V_2 = V\circ f\,, \tag{3.11b}$$
$$V_3 = f\circ V\circ g\,, \qquad \text{where } g(\vec{s}) = -\vec{s}-\vec{\eta}, \tag{3.11c}$$
$$V_4 = V\circ h\,, \qquad \text{where } h(\vec{s}) = \vec{s}-\vec{\eta}. \tag{3.11d}$$

The derivation of this expression can be found in the original paper.

In a HénonNet, V is parameterized by any multilayer perceptron. Hence, V_1, V_2, V_3, and V_4 are given by the same neural network, only with slight modifications in their weights and biases:

- Multiplying the weights and biases in the output layer of V by -1 yields V_1.
- Multiplying the weights in the first hidden layer of V by -1 yields V_2.
- First subtracting $W_1\vec{\eta}$ from the biases in the first hidden layer of V, then multiplying the weights in the first hidden layer by -1 and finally multiplying the weights and biases in the output layer by -1 yields V_3.
- Subtracting $W_1\vec{\eta}$ from the biases in the first hidden layer of V yields V_4.

Here, W_1 is the weight matrix of the first hidden layer in V.

Therefore, HénonNets consist of the same unit triangular updates as SympNets. The main difference is that per Hénon layer one arbitrary multilayer perceptron is trained and four unit triangular updates are performed with slight modifications of this multilayer perceptron. Note that since modifications of the same multilayer perceptron are reused, the four unit triangular updates are not independent and share weights and biases. Since SympNets are, due to the unit triangular updates, equivalent to a concatenation of SRNNs and because HénonNets are built out of the same unit triangular updates, HénonNets are equivalent to a concatenation of SRNNs as well.

An overview of the three different neural network architectures is given in Table 3.1. All three approaches are phrased as possible multiple integration steps with learned Hamiltonians. Although SRNNs were originally introduced this way, we just saw that SympNets and HénonNets can be formulated in a similar fashion. Moreover, this highlights the fact that not only SRNNs learn a Hamiltonian, but SympNets and HénonNets learn multiple hidden Hamiltonians as well. Whether those Hamiltonians approximate the underlying Hamiltonian of the system is not clear. The purpose of presenting all approaches in this formulation is to showcase the similarities and to discuss the differences.

Table 3.1: Overview of three of the different neural networks for Hamiltonian systems that were introduced in the literature so far.

	SRNNs	**SympNets** ($2n$ gradient modules)	**HénonNets** (n Hénon layers)
Definition of the input to output mapping	$\mathrm{SRNN}(\vec{p},\vec{q}) = \mathrm{SI}(H_{\mathrm{NN}},\vec{p},\vec{q})$	$\mathrm{SympNet}(\vec{p},\vec{q}) = \mathrm{SE}(H_{\mathrm{NN}_n},\cdot,\cdot)\circ\cdots\circ\mathrm{SE}(H_{\mathrm{NN}_1},\vec{p},\vec{q})$	$\mathrm{HénonNet}(\vec{p},\vec{q}) = \mathrm{SE}(H_{\mathrm{NN}_{2n}},\cdot,\cdot)\circ\cdots\circ\mathrm{SE}(H_{\mathrm{NN}_1},\vec{p},\vec{q})$
Details of the learned Hamiltonians	$H_{\mathrm{NN}}(\vec{p},\vec{q}) = T_{\mathrm{NN}}(\vec{p}) + U_{\mathrm{NN}}(\vec{q})$	$H_{\mathrm{NN}_i}(\vec{p},\vec{q}) = T_{\mathrm{NN}_i}(\vec{p}) + U_{\mathrm{NN}_i}(\vec{q})$	$H_{\mathrm{NN}_{2i-1}}(\vec{p},\vec{q}) = V_{\mathrm{NN}_i}(-\vec{p}) + V_{\mathrm{NN}_i}(\vec{q})$ and $H_{\mathrm{NN}_{2i}}(\vec{p},\vec{q}) = V_{\mathrm{NN}_i}(-\vec{p}-\vec{\eta}) + V_{\mathrm{NN}_i}(\vec{q}-\vec{\eta})$
Parameterization of the learned Hamiltonians	T_{NN} and U_{NN} can be any multilayer perceptrons.	T_{NN_i} and U_{NN_i} are multilayer perceptrons with one hidden layer.	V_{NN_i} can be any multilayer perceptron.
The integrators that can be used	SI can be any symplectic integrator that is explicit for separable Hamiltonians.	SE is the Symplectic Euler integrator.	SE is the Symplectic Euler integrator.

The point of view in which layers correspond to integration steps for learned ordinary differential equations is not new. It is similar to the one introduced for ResNets in E (2017). Moreover, it is used to analyze the stability of neural networks in Haber & Ruthotto (2017) and to later introduce Neural Ordinary Differential Equations in Chen et al. (2018). Also, it is interesting to note that approximating symplectic maps by concatenating integration steps for Hamiltonian systems is close to the key idea in the proof that any symplectic map can be approximated by a concatenation of Hénon-like maps (Turaev (2002)). This is what both the universal approximation theorems of the SympNets and HénonNets are based on.

Alternatively, one could write all neural network architectures as concatenations of unit triangular updates. This is more in line with how SympNets and HénonNets are introduced but it is also possible for SRNNs. Since symplectic integrators that are explicit for separable Hamiltonian systems alternate between updating positions and momenta, they also perform unit triangular updates (Yoshida (1990b)). The formulation as unit triangular updates shows the similarities as well. However, it is less elegant for the SRNNs because which updates are performed and how the learned maps V are connected depends on the symplectic integrator that is used. In both formulations it becomes clear how one can create a generalized framework covering all these neural networks for Hamiltonian systems and even enabling to introduce new ones.

3.3.1 Generalized Hamiltonian Neural Networks

To get an overview of how the different methods intersect, we present Figure 3.6, where it is shown that there is already overlap between the different methods. Any SympNet with only two gradient modules is also an SRNN and any SRNN using the symplectic Euler method and only one hidden layer in both multilayer perceptrons is also a SympNet. Furthermore, any HénonNet using only one hidden layer in all multilayer perceptrons for the learned maps V is also a SympNet (but with additional constraints).

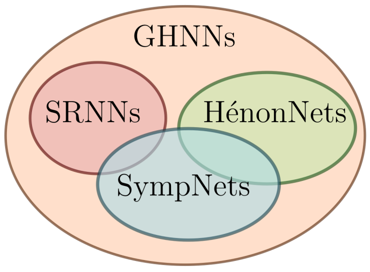

Figure 3.6: Venn diagram showing the overlap between the different neural network architectures for Hamiltonian systems.

We propose a novel type of neural networks for Hamiltonian systems. We denominate this new neural network architecture Generalized Hamiltonian Neural Networks (GHNNs). All types of neural networks for Hamiltonian systems discussed so far can be found as special cases of this framework. This is achieved by removing as many restrictions as possible from the individual layers, while still keeping the symplectic structure in the topology of the neural network.

In the same formulation as used in Table 3.1, in which layers are considered to be integration steps with learned Hamiltonians, the new architecture can be written as:

$$\begin{aligned} \text{GHNN}(\vec{p},\vec{q}) &= \text{SI}_n(\mathcal{H}_{\text{NN}_n},\cdot,\cdot)\circ\cdots\circ\text{SI}_1(\mathcal{H}_{\text{NN}_1},\vec{p},\vec{q}), \\ \mathcal{H}_{\text{NN}_i}(\vec{p},\vec{q}) &= T_{\text{NN}_i}(\vec{p}) + U_{\text{NN}_i}(\vec{q}), \end{aligned} \tag{3.12}$$

with T_{NN_i} and U_{NN_i} being any multilayer perceptrons and SI_i any symplectic integrator that is explicit for separable Hamiltonian systems.

To recognize the SRNNs as special cases of the GHNNs one has to restrict the GHNNs to one symplectic integrator ($n = 1$). The SympNets can be found in the GHNNs by using the symplectic Euler method for all symplectic integrators SI_i and only using one hidden layer in all multilayer perceptrons of all the learned Hamiltonians H_{NN_i}. Building a HénonNet from this general framework requires more restrictions. First, again only the symplectic Euler method can be used. Second, an even number of integrators has to be used. Finally, while general multilayer perceptrons can be used for the Hamiltonians, many weights and biases are shared between the kinetic and potential energies and also between two different Hamiltonians. The details of this weight sharing are provided in Table 3.1 and the list after Equations 3.11a to 3.11d.

Consequently, one way to create neural networks that are not included in one of the three existing architectures would be to use an integrator different from the symplectic Euler method. Or, instead of using a different integrator, another approach could be to use more than the one hidden layer in the multilayer perceptrons that compose the learned Hamiltonians in SympNets. One such GHNN with two hidden layers is shown in Figure 3.7.

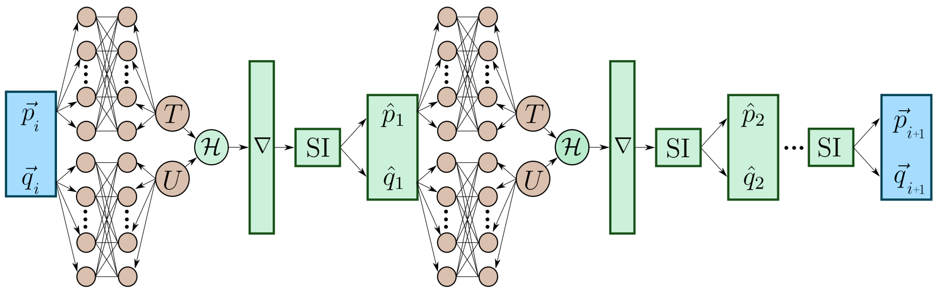

Figure 3.7: Schematic of a Generalized Hamiltonian Neural Network.

In SympNets, only one hidden layer is used since this suffices to prove the universal approximation theorem. However, from machine learning literature we know that using deep NNs instead of wide ones can lead to better approximation capabilities with fewer trainable parameters (Telgarsky (2015)).

3.4 Implementation

Contrary to how we introduced GHNNs, we do not implement them using the concept of learned Hamiltonians and performing integration steps. While SRNNs were implemented that way, it requires calculating gradients of the neural networks with respect to their input during training and also during inference. On the one hand, this can be done with automatic differentiation, but on the other hand, it considerably slows down the prediction speed of the neural network. For that reason, we instead precalculate the gradients of multilayer perceptrons with one and two hidden layers. The same is done in the original implementation of the gradient modules in SympNets. This does not imply a restriction of the GHNNs. Mathematically, they perform the same calculations as if they were implemented as concatenated integrators. It simply enables a better comparison with SympNets. For a fair comparison we also use this implementation for HénonNets.

Calculating the gradient of a multilayer perceptron with one hidden layer is straightforward. The result can be found in Jin et al. (2020a) for example:

$$\nabla_{\vec{q}}\text{MLP}(\vec{q}) = \nabla_{\vec{q}}\vec{w}^T\Sigma(W\vec{q}+\vec{b}) = W^T\text{diag}(\vec{w})\sigma(W\vec{q}+\vec{b}). \tag{3.13}$$

Here $W \in \mathbb{R}^{n\times d}$ refers to the weight matrix in the hidden layer, $\vec{b} \in \mathbb{R}^n$ to the bias vector in the hidden layer, $\vec{w} \in \mathbb{R}^n$ to the weight matrix/vector in the output layer and Σ to the activation function, with σ its derivative. Both are applied element-wise. In this calculation, $\vec{q}$ can be exchanged with $\vec{p}$ for the gradient needed for a lower triangular update.

The gradient of a multilayer perceptron with two hidden layers is far more convoluted. The derivation can be found in the original paper. The gradients of multilayer perceptrons with three or more hidden layers can be calculated as well. However, because the expression then becomes even more complicated, we compute these gradients using automatic differentiation. For our numerical experiments in the next section, only multilayer perceptrons with up to two layers are used for a fair computation speed comparison with the other neural network architectures.

All code to train the neural networks for this project is written in `Python` using `PyTorch` (Paszke et al. (2019)). The code including methods to generate and process the data for the numerical experiments can be found on `GitHub` (`https://github.com/AELITTEN/GHNN`). Our data can also be found on `Zenodo` (`https://zenodo.org/records/11032352`) and the 850 neural networks trained for this work are available in a separate GitHub repository
(`https://github.com/AELITTEN/NeuralNets_GHNN`).

3.5 Numerical Experiments

In order to compare the new Generalized Hamiltonian Neural Networks with existing neural networks for Hamiltonian systems and with physics-unaware neural networks as well, we perform multiple experiments. For a fair and objective comparison it is not obvious how to choose the topology hyperparameters of the different neural networks (such as the number of neurons and the number of layers). Instead of comparing neural networks with a similar number of trainable parameters, we compare neural networks with similar prediction speeds during inference and training. Due to the computationally expensive need to calculate the gradients of the learned Hamiltonians inside SRNNs, comparing SRNNs to the other neural networks is unfair. As a consequence, we do not include SRNNs in the experiments.

The hyperparameters of the neural networks that lead to similar prediction speeds, and are therefore used for the numerical experiments in this section, can be found in Table 3.2.

Table 3.2: Topology hyperparameters of the different neural networks used for the numerical experiments. The number of layers (except for MLP) is the number of layers of the MLPs that compose the learned Hamiltonians.

NN type	Learned Hamiltonains	Layers	Neurons per Layer	Trainable parameters 3-body problem
MLP	-	5	128	69260
SympNet	10	1	50	8000
HénonNet	10	1	50	2030
GHNN	5	2	25	8500
Deep HénonNet	6	2	25	2568

For all different types of neural networks, including GHNNs, it is not obvious how to choose the hyperparameters. As a starting point for the SympNet hyperparameters, we take a look at the SympNets from Jin et al. (2020a). To guarantee that we do not restrict the capabilities of the SympNets, we use the largest SympNets of this paper, which are the ones used for the 3-body problem considered in Jin et al. (2020a). Since GPUs are used for the training of the neural networks and for inference and since large matrix-vector products in each layer can be efficiently parallelized, the main hyperparameter determining the speed is the depth of the neural network and not the number of neurons in each layer. Hence, the total number of independent layers in the SympNets and GHNNs is kept the same. The SympNets have one hidden layer per multilayer perceptron, two multilayer perceptrons per learned Hamiltonian and ten learned Hamiltonians. The GHNNs have two hidden layers per multilayer perceptron, two multilayer perceptrons per learned Hamiltonian and five learned Hamiltonians. A HénonNet performs four sequential operations with one trained multilayer perceptron. It has one independent multilayer perceptron per two learned Hamiltonians. Therefore, only one quarter of the total number of independent layers of a SympNet can be used in a HénonNet to end up at the same prediction speed. The deep HénonNets are HénonNets with two hidden layers per multilayer perceptron. These were not tested in the original paper that introduced the HénonNets (Burby et al. (2021)). Since a physics-unaware multilayer perceptron does not need a large depth to be a universal approximator, only five layers are used.

While the general framework of our GHNNs allows the use of any symplectic integrator that is explicit for separable Hamiltonians, only the Symplectic Euler method is used for the numerical experiments. Usually a higher-order accurate integrator like the Störmer-Verlet integrator leads to higher accuracy in the prediction of the next state. But, since the Hamiltonians are only learned and do not necessarily correlate with the real Hamiltonian of the system, a better performance of a GHNN with a higher-order accurate integrator cannot be expected. Using the Störmer-Verlet integrator would definitely decrease the prediction speed and might also lead to problems during the training since the unit triangular updates are no longer independent, which complicates the optimization. Nevertheless, investigating the use of other integrators could be an interesting topic for future research.

How the total number of trainable parameters depends on depth, width and the dimension of the Hamiltonian system widely varies for the different neural network architectures. This can be seen in the last column in Table 3.2. The number of trainable parameters heavily depends on the dimension of the Hamiltonian system for SympNets and HénonNets. On the other hand, for the MLP and GHNN the dimension of the Hamiltonian system only has a small effect. This can be explained by the fact that the number of trainable parameters in both an MLP and a GHNN scales with n^2:

$$\begin{aligned}\text{Trainable parameters in an MLP} &= (l-1)n^2 + (4d+l)n + 2d, && (3.14a)\\ \text{Trainable parameters in a SympNet} &= 2m(d+2)n, && (3.14b)\\ \text{Trainable parameters in a HénonNet} &= m/2((l-1)n^2 + (d+l+1)n + d) && \\ &\overset{l=1}{=} m/2((d+2)n + d), && (3.14c)\\ \text{Trainable parameters in a GHNN} &= 2m((l-1)n^2 + (d+l+1)n), && (3.14d)\end{aligned}$$

where l refers to the number of hidden layers, n to the number of neurons per layer, d to the dimension of the Hamiltonian system (state $\vec{s}$ is in $\mathbb{R}^{2d}$) and m is the number of learned Hamiltonians, i.e., half the number of gradient modules or twice the number of Hénon maps, respectively.

For the experiment, 50 neural networks per architecture are trained using the Adam algorithm (Kingma & Ba (2015)). Loss plots for all neural networks and all numerical experiments are provided in Appendix 3.A. All 50 of these neural networks sharing the same architecture are identical except for the seeds in the random initialization of their weights. These weights are drawn from a normal distribution with a mean of 0 and a variance of 0.1. All biases are initialized as 0. The `PyTorch` seed is set to 1 for the first neural network, to 2 for the second, and so on. Resulting in different but repeatable initial weights for all neural networks with the same architecture. This allows a quantification of the influence of the initialization of the neural networks and enables us to be certain about whether one architecture is better than the other (given our chosen hyperparameters).

Error plots are provided to compare the different architectures. For these figures, the mean absolute error over the trajectories

$$\left\{ \begin{pmatrix} \vec{p}_i^T(t) & \vec{q}_i^T(t) \end{pmatrix}^T \mid i \in S_{\text{test}}, t \leq t_{\text{end}} \right\}$$

in the test data S_{test} is calculated for all neural networks. For a fixed time t the mean $\mu(t)$ over all 50 neural networks is calculated as:

$$\mu(t) = \frac{1}{50} \sum_{j=1}^{50} \mu_j(t), \tag{3.15}$$

with

$$\mu_j(t) = \frac{1}{2d|S_{\text{test}}|} \sum_{i \in S_{\text{test}}} \left\| \begin{pmatrix} \vec{p}_{i,\text{data}}(t) \\ \vec{q}_{i,\text{data}}(t) \end{pmatrix} - \begin{pmatrix} \vec{p}_{i,\text{NN}_j}(t) \\ \vec{q}_{i,\text{NN}_j}(t) \end{pmatrix} \right\|_1. \tag{3.16}$$

Additionally, as a measure of the variance due to different random initializations of the weights and biases, an area with the upper and lower bounds given by the 90% and 10% quantiles of $\mu_j(t)$ at each point in time is plotted in the figures.

3.5.1 3-Body Problem

We compare the neural networks on data of a gravitational 3-body problem. In this Hamiltonian system, three bodies of different masses orbit each other according to Newton's law of gravitation. The gravitational N-body problem is important for astrophysics. The dynamics of planets, stars and galaxies are described by this Hamiltonian system as long as no relativistic effects are considered.

The system, in all three spatial dimensions, is of the total dimension $d = 3N$, with $\vec{q} = \left(\vec{q}_1^T \cdots \vec{q}_N^T\right)^T \in \mathbb{R}^d$ and the canonical coordinates of each body $\vec{q}_i$ being the coordinates in Euclidean space. The Hamiltonian of the gravitational N-body problem can be written as

$$H(\vec{p},\vec{q}) = \frac{1}{2}\vec{p}^T M^{-1}\vec{p} - \sum_{i=1}^{N}\sum_{j=1}^{i} G\frac{m_i m_j}{||\vec{q}_j - \vec{q}_i||_2}, \tag{3.17}$$

where G is the gravitational constant and M the mass matrix, which consists of the masses m_i of the N bodies and is given in three-dimensional space by

$$M := \text{diag}(m_1, m_1, m_1, m_2, \cdots, m_N, m_N, m_N). \tag{3.18}$$

For our numerical experiments, we simplify the problem. First of all, we restrict ourselves to the 3-body problem in two spatial dimensions; the overall system is six-dimensional. Moreover, we choose all masses to be dimensionless and all equal to one; $m_i = 1$. Further, we scale the system to $G = 1$ (Heggie & Mathieu (1986)). This results in

$$H(\vec{p},\vec{q}) = \frac{1}{2}\vec{p}^T\vec{p} - \sum_{i=1}^{3}\sum_{j=1}^{i}\frac{1}{||\vec{q}_j - \vec{q}_i||_2}, \quad \text{with: } \vec{q}_i \in \mathbb{R}^2. \tag{3.19}$$

The data of the 3-body system is generated using the arbitrarily high-precision code `Brutus` (Boekholt & Portegies Zwart (2015a)). This enables us to make sure that the data the neural networks are trained on is actually the ground truth. Since the 3-body problem is a chaotic problem, the initial conditions are chosen such that the trajectories start with rather simple and easy to predict dynamics that become increasingly more chaotic over time. The initial conditions are the same as in the 3-body data of the papers introducing HNNs and SympNets (Greydanus et al. (2019b); Jin et al. (2020a)).

The random initial positions of the bodies are on a circle around the origin with the radius r drawn uniformly from $[0.9, 1.2]$. Furthermore, all three bodies have the same initial distance from each other. With these initial positions, one can calculate the momenta of the three bodies such that they would stay on the circle with radius r, orbiting the origin. These momenta are multiplied by different random uniform factors from $[0.8, 1.2]$ and chosen as initial momenta. With the initial conditions arranged in such a way, only small accelerations and slow exchange in kinetic and potential energy occur until $t \approx 5$. Afterwards, close encounters of the bodies may occur, leading to high accelerations and quick exchange in potential and kinetic energy. As in Jin et al. (2020a), 5000 trajectories are generated, a final time $t_{\text{end,train}} = 5$, and a step size $h = 0.5$ are used for training. In contrast to Jin et al. (2020a), a final time $t_{\text{end}} = 7$ is used for testing to check whether the neural networks can generalize from the simple behavior to the case with close encounters. The data is split into 80% training, 10% validation and, 10% test data.

The benefit of the high number of trainable parameters inside a multilayer perceptron becomes apparent in Figure 3.8. Inside the training data the multilayer perceptrons have an error about five times lower than the SympNets. However, the middle graph then directly reveals the most severe drawback of the physics-unaware multilayer perceptrons. After only two timesteps outside of training data, the error of the predictions by the multilayer perceptrons is higher than the error of the four physics-aware neural networks. Moreover, the error of the GHNNs inside the training data is five times lower than the error of the multilayer perceptrons. Hence, the GHNNs are highly accurate and able to generalize at the same time. Again, the HénonNets with one hidden layer per MLP have the highest error of all neural networks inside the training data and are able to generalize slightly better than the multilayer perceptrons outside the training data.

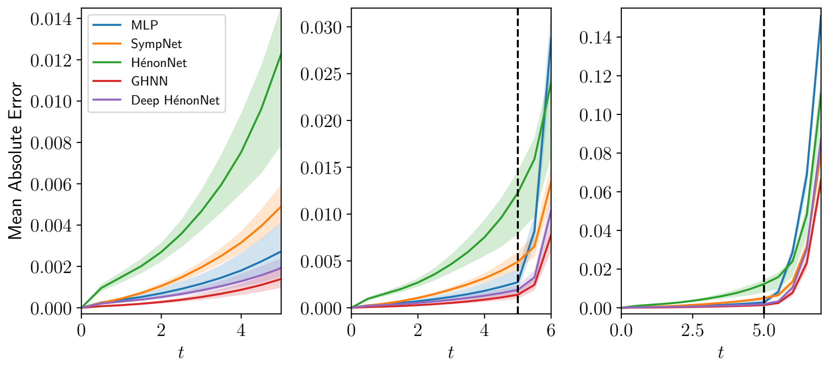

Figure 3.8: Comparison of the mean error over all the trajectories of the three bodies, in the test data for 50 MLPs, SympNets, HénonNets, and GHNNs. Additionally to the HénonNets with one hidden layer per learned V, the error of 50 HénonNets with two hidden layers is shown (Deep HénonNet). The lighter-colored areas indicate the variations between the differently initialized neural networks. The left graph shows the time range included in the training data. The middle graph shows the error inside the time range included in the training data and two steps outside, with the black dashed line indicating where the time range included in the training data ends. The right graph shows the errors along the full trajectories.

Deep HénonNets showed worse performance in similar experiments for the single and double pendulum (see original paper). In that case, the results is due to only three Hénon layers being used to achieve the same prediction speed as all the other neural networks. Since a large depth might be required by the structure-preserving neural networks in order to approximate certain symplectic maps, only three Hénon layers might be insufficient. For the 3-body problem, however, the roles are reversed. The deep HénonNets have a smaller error than all other methods except for the GHNN, both inside and outside the training data. This shows that to approximate the flow map of the 3-body problem, more complex updates of the state are of greater importance than many updates. For the same reason, the GHNNs outperform the SympNets by an order of magnitude.

3.6 Conclusion

We introduced an overarching framework that unifies three widely-used neural network architectures specifically designed to learn the symplectic flow maps of Hamiltonian systems: SRNNs, SympNets, and HénonNets. Additionally, this framework allows for the creation of novel neural network topologies that cannot be built from the individual architectures. These new Generalized Hamiltonian Neural Networks (GHNNs) combine the benefits and flexibility of all three aforementioned neural network architectures.

In the numerical experiment performed (single and double pendulum in the original paper and gravitational 3-body problem shown here), 50 neural networks of four different architectures (MLP, SympNet, HénonNet, and GHNN) were trained with the hyperparameters chosen such that a prediction is realizable at the same speed for all networks. In this experiment, our neural networks, the GHNNs, outperform all other neural networks for the single pendulum data. The extrapolation capabilities of the GHNNs are astonishingly good. The errors of the GHNNs and the SympNets are similar when trained on data for the double pendulum system. Of both, the errors are remarkably smaller than those of HénonNets and multilayer perceptrons. For the 3-body problem, the GHNNs have better accuracy than all other neural networks, with the errors being over one order of magnitude smaller than those of the SympNets. This drastic variation in the difference between the errors of the GHNNs and SympNets (also HénonNets and deep HénonNets) suggests that there are symplectic maps, where in order to approximate these, complex unit triangular updates are essential and others where many simple ones are the better choice. Due to the second layer in the MLPs of the GHNNs and deep HénonNets, far more complex unit triangular updates can be learned by these methods.

These numerical experiments show that a universal approximation theorem and a high number of trainable parameters are insufficient to guarantee good approximation capabilities. A universal approximation theorem can be proven for all three symplectic neural network architectures and the number of trainable parameters in the SympNets and GHNNs for the 3-body problem are almost the same. We conclude that it is important how the trainable parameters are arranged in the neural network. The choice of how to arrange the trainable parameters can decrease the prediction error more than one order of magnitude while preserving the same calculation speed.

At the same time, the HénonNets demonstrate that having too few trainable parameters also limits the accuracy of the predictions. In a comparison with a similar number of trainable parameters for all neural network architectures, as done in Burby et al. (2021), HénonNets might outperform other approaches. In addition, all neural networks in our numerical experiments were trained for many epochs to ensure that a good optimum was found in the loss landscape. This allowed for a fair comparison of the approximation capabilities of the different architectures. In Burby et al. (2021), the number of epochs is quite limited and the HénonNets have, opposite to our findings, a smaller error than the SympNets. This indicates that the additional structure in the HénonNets might be helpful in reaching a good approximation quickly but restricts the approximation when the neural network is trained for a long time.

Overall, the added symplecticity in all structure-preserving neural networks yields a better generalization and stability outside the training data. Moreover, the structure-preserving architectures achieve comparable or better accuracy than the multilayer perceptrons with far fewer trainable parameters. This can be useful for the application to large Hamiltonian systems where large neural networks are needed and memory is limited.

Acknowledgment

This work was funded by the Dutch Research Council (NWO), in the framework of the program 'Unraveling Neural Networks with Structure-Preserving Computing' (file number OCENW.GROOT.2019.044).

Appendix

3.A Loss during training of the different architectures

The training behavior of the different neural network architectures is given in Figure 3.9, which shows that the SympNets and the GHNNs achieve a similar loss for the single

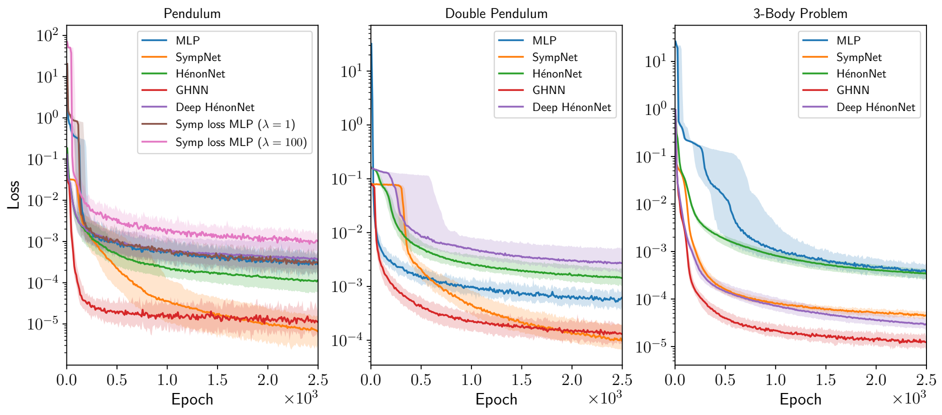

Figure 3.9: Comparison of the validation loss during the first 2500 epochs of training for all three Hamiltonian systems. The solid lines show the behavior of the median of the 50 neural networks trained per architecture. The lighter-colored areas indicate the variations between the differently initialized neural networks.

and double pendulum. However, the GHNNs need less than 500 epochs to reach a good minimum in the loss landscape, whereas the SympNets take around 1500 epochs to find a similar minimum.

In Figure 3.10, the loss is plotted over all 25000 epochs. It is clear that after 5000 epochs, the loss of almost all neural networks for all three Hamiltonian systems has nearly converged. After 5000 epochs, only the loss of the GHNNs on the pendulum data and the loss of the MLPs on the 3-body problem data keep improving significantly.

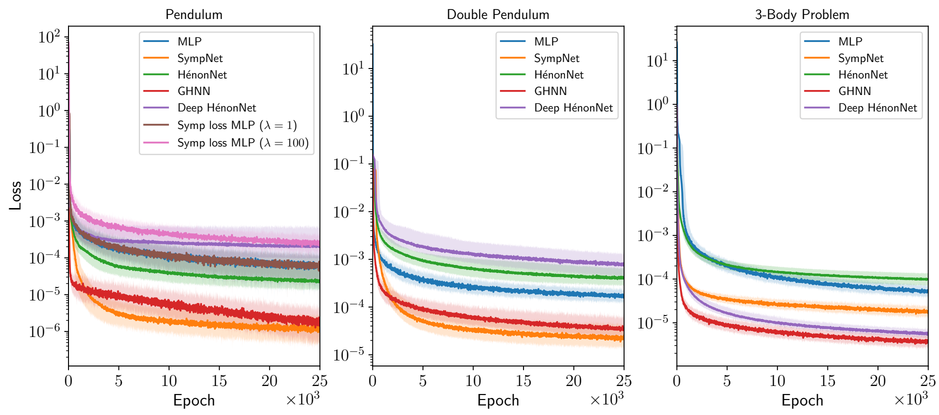

Figure 3.10: Comparison of the validation loss during the entire 25000 epochs of training for all three Hamiltonian systems. The solid lines show the behavior of the median of the 50 neural networks trained per architecture. The lighter-colored areas indicate the variations between the differently initialized neural networks.