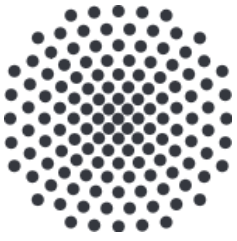
Dimensionality reduction of manifolds of dynamical systems with a rank-1 saddle by application of autoencoder artificial neural networks

Master's thesis of
Melissa Jacqueline Lober

August 10, 2021

First examiner: Prof. Dr. Jörg Main
Second examiner: Prof. Dr. Udo Seifert



Institut für Theoretische Physik I
Universität Stuttgart
Pfaffenwaldring 57, 70550 Stuttgart

Contents

1 Introduction

This thesis revolves around the dimensionality reduction of a special manifold in a rank-1 saddle system that models the transitions of classical objects between two states in the context of Transition State Theory (TST) [1–5]. This reduction of dimension is performed via the application of a machine learning (ML) method, the autoencoder (AE) artificial neural network. Another key focus of this thesis is the analysis and reproduction of the dynamics on this manifold in its reduced form given by the AE models, via further ML techniques.

In a previous approach [6], this manifold was parameterized by describing it via a certain subset of the original coordinates. If necessary, the missing coordinates are reconstructed by a feed forward neural network given the remaining ones. In contrast to this approach, the parameterization via the AEs is chosen freely by the models without any restrictions. Due to this freedom of choice, one hopes for a dimension reduction method that is more flexible and is also able to adapt to strongly curved manifolds of more complex systems. In addition, one hopes to gain further insights into the structures and dynamics on the manifold, since the lower-dimensional representation given by the AEs is constructed without human bias.

As indicated above, a key focus of this work is the modeling of transitions of classical objects between two states. Besides the obvious chemical reactions, there are many more changes of state that can be found in topics ranging from e.g. cosmology [7], atomic physics [8] and cluster formation [9, 10] to Bose-Einstein condensates [11–14]. The states are usually characterized through local minima on a potential energy surface and are energetically separated by a barrier. Reactant objects travel over this energy barrier in order to transition into the product state. The particular system which is subject to this thesis is described by one unstable reaction coordinate and one stable degree of freedom, the orthogonal mode. Further, the system is time-dependent and the barrier oscillates along the reaction coordinate. Two manifolds, a stable and an unstable one, can be identified in the phase space of such a system. Particles that are initialized on the stable manifold approach the maximum of the barrier asymptotically fast, while those initialized on the unstable one do the same when propagated backwards in time. These two manifolds intersect in a region that defines another manifold, the Normally Hyperbolic Invariant Manifold (NHIM) [15, 16]. This particular manifold is subject to the dimensionality reduction by the AE models. When particles are initialized on

the NHIM they are bound to it and stay on it forever when propagated both, forward and backward in time. The NHIM and the dynamics on it are of great interest, since they provide insights into the general reaction dynamics of the system and enable the calculation of decay rates which are a measure of the system's stability.

Interpreting the structure of the multidimensional and time-dependent NHIM, as well as propagating particles on it can become very difficult and computationally expensive, especially for higher-dimensional systems. In this thesis, it is shown how the dimension reduction of the NHIM of a two-dimensional model system via AEs can provide a lower-dimensional representation that is easier to interpret, yet still captures all of the NHIM's relevant features. Further, the AE's role as a stabilizer of particles on the NHIM is examined. In addition, four ML methods are developed to successfully predict trajectories on the NHIM in its lower-dimensional representation, given by the AE models, for different time scales. With these methods it is possible to generate trajectories of particles initialized anywhere on the compressed form of the NHIM and then transform them back into the original space of the system via the AEs. Applying these methods could potentially replace the costly propagation of particles in higher-dimensional systems.

Outline of the thesis

This thesis is structured into three main parts. Chapter 2 explains all the underlying theoretical principles that this work is built on. In Chapter 3, all of the ML methods that are applied for the dimensionality reduction of the NHIM, as well as various methods for the prediction of trajectories on it are presented. The outcomes of these methods are analyzed in Chapter 4. The following describes the contents of all chapters in more detail.

Chapter 2 starts off by introducing the principles of a rank-1 saddle system in the context of TST, both for one-dimensional systems and multidimensional time-dependent systems. Further, the particular model system that this thesis revolves around is presented in more detail in Section 2.2. Section 2.3 is about the theoretical principles of ML in general, as well as about three ML techniques, the feed forward neural network, the autoencoder and the recurrent neural network.

In Chapter 3, two different AE models for the parameterization of the NHIM are presented, as well as various methods for the prediction of trajectories in the reduced space of the NHIM, given by the respective AE model.

Chapter 4 begins by analyzing the effects that a change in the AE's hyperparameters can have on its performance. The AE is further examined as a tool for stabilizing points on the unstable NHIM in Sec. 4.1.2. In addition, the dynamics on the NHIM are investigated through Poincaré maps of trajectories in the lower-dimensional representation of the NHIM given by one of the AE models presented in Section 3.1.2. Section 4.2 presents

the results of all four methods for the prediction of trajectories. In Section 4.2.5, the resulting trajectories in the LS are transformed back into the original space of the system and validated there.

2 Theory

This chapter presents all necessary underlying theoretical principles of this thesis. It is split into three sections – Section 2.1 covering rank-1 saddle systems in Transition State Theory, Section 2.2 introducing the particular system that this thesis revolves around and Section 2.3 presenting the basics of various machine learning techniques used throughout this work. In Sec. 2.1, it is explained what characteristics define a one-dimensional as well as a driven multidimensional system and how certain manifolds can be identified in the phase space. Special emphasis is on one particular manifold which is subject to a dimension reduction by autoencoder neural networks. This machine learning technique, among others such as feed forward neural networks and recurrent neural networks, is presented in Sec. 2.3.

2.1 Rank-1 saddle systems in Transition State Theory

A main focus of Transition State Theory (TST) is reaction rates and pathways of chemical reactions. However, there are many more changes of state that can be regarded as a reaction besides chemical ones. Whenever such a change of state can be described through classical equations of motion, TST is a useful tool to describe the transitions from one state to the other. These states are usually characterized through local minima on a potential energy surface and are energetically separated by a barrier. Reactant objects travel over this energy barrier, the saddle, in order to transition into the product state. In a *rank-1* saddle system there is only one unstable degree of freedom, the reaction coordinate, and an arbitrary amount of stable degrees of freedom, the orthogonal modes.

This section presents the description of a one-dimensional static system, as well as a driven multidimensional one. Further, the so called *Binary Contraction Method* for the calculation of positions of a certain manifold in the system's phase space is explained, as well as a method on how to calculate decay rates in rank-1 saddle systems. A more detailed description of the principles explained here is given in Ref. [17], from which this section takes most of its inspiration.

2.1.1 One-dimensional static systems

In a static and one-dimensional system, there is only the unstable reaction coordinate and no other degrees of freedom, as well as no movement of the barrier. The system described in this subsection is open, meaning particles traveling away from the barrier never return. Its simplicity makes the system well suited as an introductory example of reactions in time-invariant systems.

Figure 2.1 (a) shows a sketch of the potential energy landscape of an exemplary static and one-dimensional system. The coordinate x marks the reaction coordinate along which a reactant particle transitions into the opposite state. Here, the reactant state $\mathcal{R}$ and product state $\mathcal{P}$ are clearly separated by the maximum of the barrier at $x = 0$, the *saddle point*. In order for a trajectory to be reactive, it has to be initialized with a momentum towards the barrier, as well as have enough energy to pass the saddle point and overcome it. A reactive trajectory can either transition from the reactant side $\mathcal{R}$ to the product side $\mathcal{P}$ or the other way around. Whenever a trajectory is approaching the barrier but its energy is not high enough to pass the saddle point, it returns to its respective originating side and is declared non-reactive. Consequently, it is the saddle point that determines whether trajectories are reactive or non-reactive. Besides reactive and non-reactive trajectories there is a special kind of trajectory that cannot be classified as one or the other. It is the trajectory that rests on the saddle point itself and stays there for all time. It can be regarded as an unstable intermediate state between the two configurations and is therefore called the *transition state* (TS). This state is not only characterized by its position on top of the barrier but also by having zero momentum. Consequently, a full description of the TS can only be given in the phase space of the system.

Figure 2.1 (b) provides a sketch of the static one-dimensional system in the phase space with reaction coordinate x and the associated momentum p_x. In this phase space, there are two manifolds $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$, given by the two thick lines forming a cross. When initialized on the stable manifold $\mathcal{W}_\mathrm{s}$ and propagated forward in time, trajectories approach the intersection of $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$ for $t \to \pm\infty$. Those initialized on the unstable manifold $\mathcal{W}_\mathrm{u}$ show the same behavior when propagated backward in time and otherwise depart from the intersection asymptotically fast.

The two manifolds separate the phase space into two reactive (top and bottom) and two non-reactive (left and right) domains. The four thin black arrows indicate the general flux of trajectories in these areas. Trajectories initialized on the reactant side $x < 0$ with a momentum $p_x > 0$ towards the barrier will either have a momentum large enough to overcome the barrier (upper section) or they will be rejected and depart back to their initial side (left section). For a trajectory on the reactant side $x < 0$ and a small initial momentum away from the barrier $p_x < 0$, it departs from the saddle region right away and never returns (left section). These are trajectories that have been reflected by the

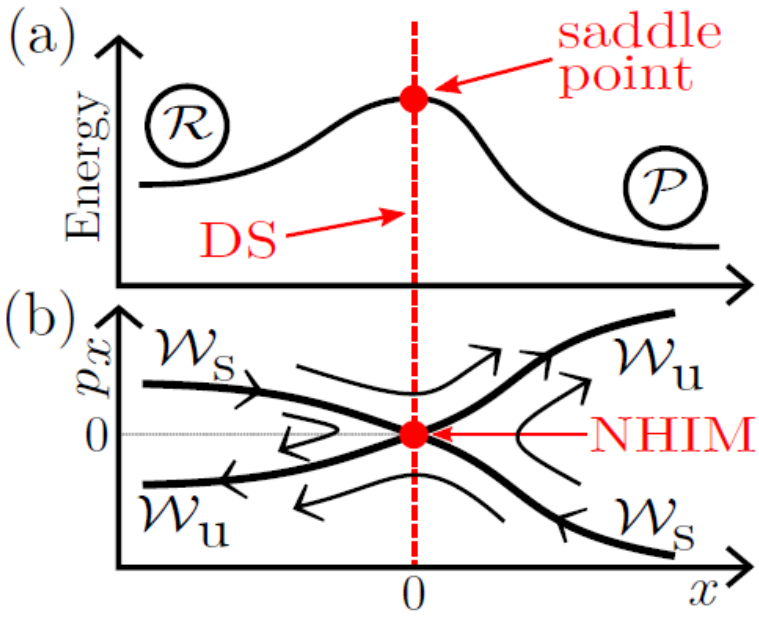


Figure 2.1: (a) Sketch of the energy barrier of a one-dimensional static system. The coordinate x is the *reaction coordinate* and the red dashed line marks the *dividing surface* (DS), separating the system into a *reactant* $\mathcal{R}$ and *product* $\mathcal{P}$ configuration. The maximum of the barrier is marked by the *saddle point* as a red dot. (b) Sketch of the corresponding phase space (x, p_x) of the system depicted in (a). A stable $\mathcal{W}_\mathrm{S}$ and an unstable $\mathcal{W}_\mathrm{u}$ manifold marked by the thick black lines separate the system into two reactive and two non-reactive domains, indicated by the thin black arrows. Their intersection marks the location of a hyperbolic fixed point, the NHIM, illustrated as the red dot. To this NHIM, the DS is attached and its position $x = 0$ is identical to that of the saddle point. This figure is taken from Ref. [17].

barrier in the past and are therefore classified as non-reactive. However, trajectories that have reacted over the barrier from the product side over to the reactant side in the past, have a larger negative momentum and belong to the reactive area below the intersection. The classification of trajectories initialized on the product side follows the same ideas and with that, the full phase space is separated into four distinct areas.

The intersection of the manifolds $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$ marks the position of another manifold, the *normally hyperbolic invariant manifold* (NHIM) [18–22]. Its defining property is that when a trajectory is initialized on it, it will stay there forever and never leave when propagated both forward and backward in time. Since such trajectories are bound to the NHIM for all time, they are not affected by the system's dynamics and the NHIM is

said to be *invariant*. The dynamics close to the NHIM are classified as being *normally hyperbolic*, due to exponentially growing deviations from the unstable manifold over time [23]. The NHIM corresponds to the TS in phase space and the two terms are often used interchangeably. In this system, the NHIM is fully defined by a single trajectory at position $x = 0$ and momentum $p_x = 0$. However, for multidimensional and time-dependent systems, the NHIM itself becomes a multidimensional and time-dependent object and detaches from the saddle point. Its complex description as the intersection of the stable and unstable manifold then enables finding it in the phase space, as will be explained in more detail in the following subsections.

The NHIM serves as an anchor point of the *dividing surface* (DS), shown as the red dashed line in Figure 2.1. As the name suggests, this surface divides the phase space into reactant and product configuration. In this static and one-dimensional example, the DS is a simple vertical line, suggesting that there is no dependence on the reactive momentum p_x. When defining a DS it is important to ensure that it is recrossing-free, meaning that trajectories can only cross it once. For a vertical DS, this property cannot be ensured for all systems, for example for those presented in [9, 24]. However, for many systems, such as the system that this thesis focuses on, the choice of a vertical DS serves its purpose of separating product and reactant configurations sufficiently well.

2.1.2 Multidimensional driven systems

In reality, there is often a variety of factors influencing a reaction, hence many degrees of freedom that have to be considered. When modeling more realistic transitions, it is usually necessary to introduce additional variables to the system and make it multidimensional. In past works, the isomerization reactions of e.g. LiCN [25, 26] and ketene [27] were examined by modeling more complex multidimensional systems. Furthermore, a system can become time-dependent, for example when it is subjected to an external and time-dependent perturbation, like an oscillating electromagnetic field driving a reaction. This section explains how multidimensional systems with a rank-1 saddle can be analyzed and understood by making use of the principles of TST. Furthermore, a method for finding the NHIM in a high-dimensional space is presented, as well as the effects of time dependency on the system.

As mentioned above, a system with a rank-1 saddle is defined by having only one unstable degree of freedom, the reaction coordinate x, and arbitrarily many stable degrees of freedom, the orthogonal modes $\boldsymbol{y_i}$. Hence, in an n-dimensional system, there are $(n-1)$ orthogonal modes. The saddle point position $(x, \boldsymbol{y_i})$ is given by the maximum of the barrier along the reaction coordinate and the minimum of any orthogonal stable degree of freedom. As indicated in the sketch in Figure 2.2 of a two-dimensional system with one stable degree of freedom, there exist trajectories bound by the stable orthogonal

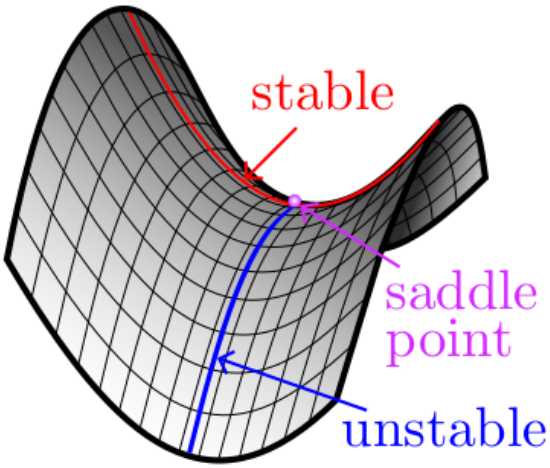


Figure 2.2: Sketch of the barrier region of an exemplary two-dimensional system with one unstable and one stable degree of freedom. The corresponding saddle point is marked as a purple point, while the red and blue line mark the stable and unstable directions of the system, respectively. This figure is taken from Ref. [17].

modes, as well as unstable trajectories approaching or departing from the barrier region along the reaction coordinate.

The phase space of a n-dimensional system is $2n$-dimensional $(x, \boldsymbol{y_i}, p_x, \boldsymbol{p_{y_i}})$, making it hard to imagine even for a four-dimensional phase space of a simple two-dimensional system with only one additional degree of freedom. The stable and unstable manifolds $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$ that divide the phase space into reactive and non-reactive regions have a dimensionality of $(2\,n-1)$. Since the NHIM, or TS, is the intersection of the two manifolds, it has a dimensionality of $(2\,n-2)$. Again, the NHIM serves as an anchor point of a $(2\,n-1)$-dimensional DS independent of the given momentum in the unstable degree of freedom. While the NHIM is a single point in a static one-dimensional system and consists of only one trajectory with zero momentum, it becomes a multidimensional object in higher-dimensional systems with an infinite amount of trajectories bound to this area for all time. These trajectories oscillate in the orthogonal modes of the system with various momenta while balancing along the unstable reaction coordinate. Any deviation from the NHIM will cause the trajectories to fall down and leave the barrier region along the unstable direction.

For each set of coordinates $(\boldsymbol{y}, \boldsymbol{p_y})$ of the orthogonal modes, the NHIM can be identified as the intersection of the stable and unstable manifolds $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$ in the corresponding (x, p_x)-plane. This is similar to the one-dimensional case illustrated in Figure 2.1, with the difference being that this intersection does no longer define the NHIM fully, but

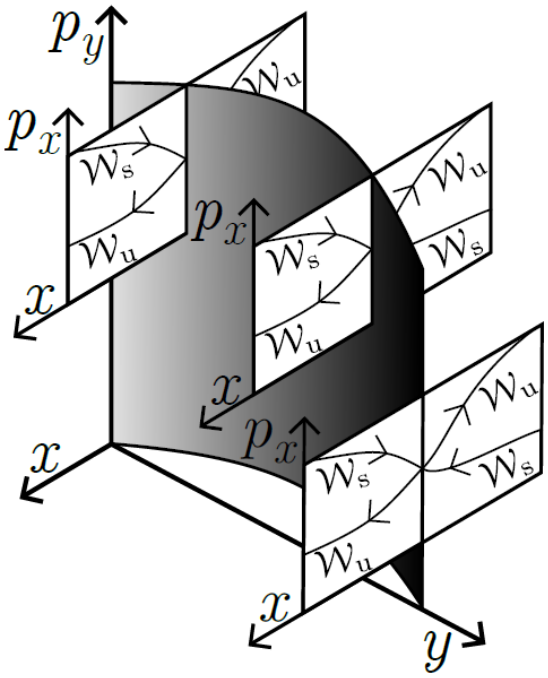


Figure 2.3: Representational sketch of the two-dimensional NHIM in the phase space (x, y, v_x, v_y) of a two-dimensional static system. The coordinate x corresponds to the unstable reaction coordinate, while y corresponds to the stable orthogonal mode. At three selected positions (y, v_y) the cross section (x, v_x) is shown, including the stable and unstable manifolds $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$. For each selected position (y, v_y), the intersections of the two manifolds give a position (x, v_x) on the NHIM. The full NHIM, shown as the grey surface, is a continuous representation of such intersections. This figure is taken from Ref. [17].

rather only gives a single position of the multidimensional object for a specific set of bath coordinates $(\boldsymbol{y}, \boldsymbol{p_y})$. Figure 2.3 illustrates how the NHIM is obtained in a representation of the four-dimensional phase space of a two-dimensional system. The NHIM is shown as a two-dimensional surface and the NHIM's position (x, p_x) is determined by the intersection of the respective stable and unstable manifolds for three different sets of orthogonal mode coordinates (y, p_y). A full description of the NHIM would be given by continuously determining the respective intersections for all possible sets of positions (y, p_y) of the orthogonal modes.

In the close neighborhood of a static barrier, the NHIM can be approximated by using normal form expansion [21, 28]. However, this procedure is perturbative and there are other methods that enable to determine single points on the NHIM numerically exact. There are two methods that are suited for the task of determining single points on the NHIM by following the procedure described in Figure 2.3. Both methods can also be

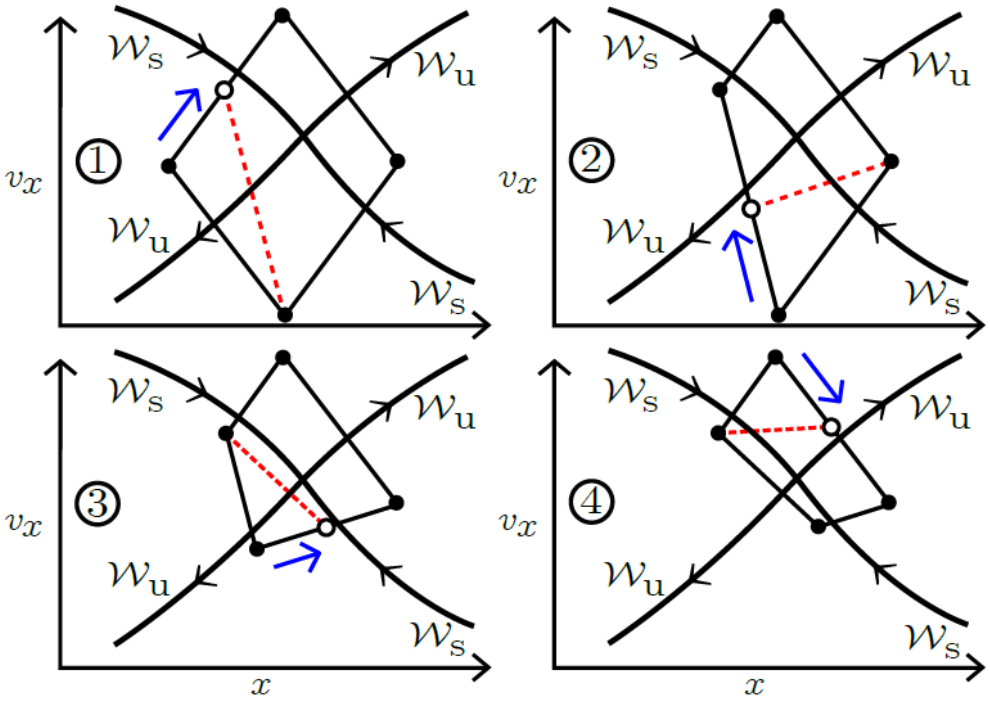


Figure 2.4: Illustration of the iterative process of the *binary contraction method* (BCM). The method starts off with a quadrangle with one vertex in each of the four areas separated by the stable and unstable manifolds $\mathcal{W}_\mathrm{s}$ and $\mathcal{W}_\mathrm{u}$ in the phase space of the system. In each step, the quadrangle's vertices move closer together, eventually revealing the intersection of the two manifolds. This figure is taken from Ref. [17].

applied to dynamical systems. One of the methods relies on identifying the stable and unstable manifolds via Lagragian descriptors [29], while the other one, and significantly faster one, is based on analyzing the dynamics of reactive and non-reactive trajectories in the close neighborhood of the barrier. The latter method is the *binary contraction method* (BCM) [16] and is presented in the following subsection.

Binary Contraction Method

As introduced before, the BCM is a fast method for finding single positions of the NHIM. It is based upon determining reactive and non-reactive regions in the phase space as described in the static one-dimensional example in Figure 2.1 (b). This is done by propagating trajectories initialized at various positions in the phase space and observing their behavior. If a trajectory is repelled by the barrier, its initial position is defined to be part of a non-reactive region, while the initial positions of trajectories that cross the

saddle point are classified as being part of a reactive region.
Figure 2.4 describes the process of the BCM in more detail. The first and crucial step is to select a position in each of the four regions in the phase space and form a quadrangle. This is a non-trivial task, since the locations of the stable and unstable manifolds, separating the different regions, are not known. To ensure that the quadrangle does not consist out of two or more positions in the same region, it is either necessary to select the positions sufficiently far away from each other and form a large quadrangle, or to simply make an educated guess.

After this step is completed, a trajectory is propagated both forward and backward in time starting from each vertex of the quadrangle until it clearly departs to either side of the barrier. This is how each initial position can be classified to be part of one of the four regions. If it is not possible to assign a position to every region, the vertices of the quadrangle were not chosen well enough and the first step has to be repeated, possibly with a larger quadrangle. In Ref. [30], a fallback method is presented that ensured the BCM to converge. This fallback method is based on initializing the quadrangle on various positions along the manifolds in order to locate their intersection.

After ensuring that the quadrangle is chosen adequately, the BCM iteratively shrinks its area while moving its center increasingly closer to the intersection, as depicted in Figure 2.4 for the first four iterations. Subsequently, each side of the quadrangle is cut in half and the resulting new vertex is again classified as being part of one of the four regions via the propagation method. This is how the center of the quadrangle approaches the intersection of the stable and unstable manifold with each iteration until it is small enough to have reached the desired accuracy.

In multidimensional systems, the BCM can be applied to the phase space plane (x, p_x) of each set of orthogonal modes $(\boldsymbol{y}, \boldsymbol{v_y})$ and fixed time t in order to find the corresponding position $(x_{\mathrm{NHIM}}, p_{x,\mathrm{NHIM}})$ of the NHIM.

Periodically driven systems

When introducing time-dependency into a system, for example by having a periodically oscillating barrier along the unstable degree of freedom, there are a few important changes that should be noted regarding the NHIM. However, all statements made for the multidimensional system are still valid in a time-dependent system at a fixed moment in time.

Unlike in Figure 2.1, where the NHIM is attached to the saddle point, the NHIM's position in phase space is completely detached from it in driven systems. In a system with an oscillating barrier, the NHIM also performs an oscillation and its frequency is the same as that of the saddle point. However, the saddle point oscillates with a significantly higher

amplitude than the NHIM. Generally, the NHIM's oscillation amplitude increases non-trivially with a decreasing oscillation frequency of the barrier. The NHIM's oscillation amplitude can never be higher than that of the barrier, since trajectories bound on the NHIM rely on the repelling force of the moving barrier to keep them on course.

It is important to note that the NHIM's movement during an oscillation period is non-trivial and its shape is constantly shifting. In earlier work, the NHIM's shape and movement was often compared to a flag blowing in the wind REFS. The NHIM is deformed strongest near the saddle point and this is also where its oscillation amplitude is largest.

With the NHIM being an anchor point for the DS, the DS moves along with the NHIM with the same oscillation frequency and amplitude.

2.1.3 Decay rates

One of the main goals of TST is to calculate the rates of transitions between two states. Since there can be various kinds of rate definitions for one system, it is important to clearify which rate definition is relevant in the context of TST. This subsection answers this question and presents a method on how to obtain this particular rate in more detail.

When dealing with the rate of a reaction, that rate usually refers to the *Kramers' rate* [31, 32]. It is the escape rate of a diffusion model in which Brownian particles are initialized in a thermal system and occasionally gather enough energy to travel over the barrier and react. However, in the context of TST, there is another rate that becomes relevant. It is the *decay rate* of a population of particles initialized close to the TS. As mentioned in Sections 2.1.1 and 2.1.2, the TS is unstable and trajectories that are initialized with any deviation from it depart from the barrier region exponentially fast. The population's decay over time is described by the decay rate of the activated complex.

The decay rate is associated to only a rather small area near the NHIM. Consequently, it depends solely on the system's dynamics in this area, meaning the equations of motion at play and the nature of the barrier. The decay rate is therefore independent of the respective ensemble. There are multiple methods for determining decay rates of a system in TST. Three of such methods were developed or elaborated in previous works of the ITP1 and are all presented in Ref. [17]. Two of them are based on propagating populations of reactive trajectories, either by using the full description of the system given by the equations of motion or by selectively propagating trajectories very near to the NHIM for a short time according to descriptions derived from the linearized dynamics of this area. The first method is the *ensemble method*, while the latter one known as the *local manifold analysis*. The third method for obtaining decay rates relies on analyzing the stability of trajectories that are bound to the NHIM for all time. This method

is the *Floquet method*. It should be noted that the Floquet method only aims at the calculation of mean decay rates that are valid for a whole oscillation period, while the enemble method and the local manifold analysis deliver time-dependent instantaneous decay rates. Despite their different approaches, all three methods yield the same mean decay rates, as it is shown in Ref. [17].

Floquet rates

Since this thesis revolves around the NHIM and the dynamics on it, the Floquet method, first introduced in Ref. [33], is very well suited for the task of calculating decay rates of the investigated systems, as it is based on analyzing trajectories on the NHIM. In earlier publications [6, 16, 34], the method was elaborated further and can now also be used to calculate the average decay rates of periodic and quasi-periodic trajectories on the NHIM of a multidimensional and time-dependent system. The following paragraphs take inspiration from Ref. [17], where this method as well as the other two methods mentioned above are explained in more detail.

The stability of a trajectory on the NHIM of a n-dimensional system is described by a *monodromy matrix*, defined by

$$\boldsymbol{M}(T) = \boldsymbol{\sigma}(t_0 + T,\, t_0)\,, \tag{2.1}$$

where T is the period length of a bound and periodic trajectory and $\boldsymbol{\sigma}$ is the $2\,n$-dimensional fundamental matrix given by

$$\frac{\mathrm{d}\boldsymbol{\sigma}(t, t_0)}{\mathrm{d}t} = \boldsymbol{J}(t)\boldsymbol{\sigma}(t, t_0)\,. \tag{2.2}$$

Here, $\boldsymbol{J}(t)$ is the instantaneous Jacobian at a given time t, calculated according to

$$\dot{\boldsymbol{\gamma}}(t) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \frac{\partial^2 \boldsymbol{\mathcal{H}}(t)}{\partial \boldsymbol{\gamma}(t^2)}\, \boldsymbol{\gamma}(t) = \boldsymbol{J}(t)\, \boldsymbol{\gamma}(t)\,, \tag{2.3}$$

where $\boldsymbol{\gamma}(t) = (x(t), p_x(t))^{\mathrm{T}}$ is the trajectory's current state and $\boldsymbol{\mathcal{H}}(t)$ is the Hamiltonian of the system.

After determining the monodromy matrix $\boldsymbol{M}(T)$, the *Floquet exponents* $\mu_{u,s}$ can be calculated via

$$\mu_{\mathrm{u,s}} = \frac{1}{T}\, \ln |m_{\mathrm{u,s}}|\,. \tag{2.4}$$

Here, m_{u} and m_{s} denote the respective eigenvalues of the monodromy matrix $\boldsymbol{M}(T)$ along the unstable and stable directions of the system. The resulting Floquet exponents are a measure of the stability of the corresponding trajectory and determine how fast

trajectories, that are initialized close to each other in the respective region, separate from each other. The correlated *Floquet rate* for this process is then simply given by

$$k_{\mathrm{F}} = \mu_{\mathrm{u}} - \mu_{\mathrm{s}} \,. \tag{2.5}$$

The Floquet exponents are valid for a full oscillation period of a bound trajectory, hence the Floquet rate is not instantaneous but rather a mean decay rate of the activated complex near this trajectory. Further, for a multidimensional system with quasi-periodic trajectories on the NHIM, the Floquet rate has to be determined in the limit of infinite time, since the theoretical period length of a quasi-periodic trajectory is infinite and $\boldsymbol{\sigma}(t, t_0)$ changes over time. Therefore, the Floquet exponents are in theory given by

$$\mu_{\mathrm{u,s}} = \lim_{t\to\infty} \frac{1}{t} \ln |m_{\mathrm{u,s}}(t)|. \tag{2.6}$$

In practice, the integration time is simply chosen large enough to capture the general dynamics of the respective trajectory.

2.2 Model system

This section presents the particular system that is subject to this thesis. Its potential energy landscape is designed to describe a rank-1 saddle system with one unstable degree of freedom y. In addition, it incorporates the influence of an external perturbation, e.g. an alternating electric field, causing the barrier of the system to oscillate along the unstable reaction coordinate x. The resulting potential energy landscape of the time-dependent and multidimensional model system is given by

$$V(x, y, t) = E_{\mathrm{b}} \exp(-a\,[x - \hat{x}\sin(\omega_x\, t)]^2) + \frac{\omega_y^2}{2}\left[y - \frac{2}{\pi}\arctan(2x)\right]^2 , \tag{2.7}$$

where x and y are nonlinearly coupled by the arctan-function, ensuring an interaction between the two dimensions, rather than an effective separation of the system into two independent one-dimensional systems. The parameter E_{b} defines the height of the Gaussian barrier, which is oscillating with the amplitude $\hat{x}$ and a frequency of ω_x. The orthogonal mode is described by a harmonic potential with an angular frequency of ω_y. The set of standard parameters is given by the following dimensionless values: $E_{\mathrm{b}} = 2$, $\hat{x} = 0.4$, $\omega_x = \pi$, $\omega_y = 2$. Unless it is stated otherwise, the potential energy landscape used throughout this work is generated with these parameters. The resulting potential is shown in Figure 2.5 at time $t = 0$. The reactant and product configurations $\mathcal{R}$ and $\mathcal{P}$ are located at two local minima and are separated by a barrier. The saddle point is located on top of the barrier and is marked by the white cross.

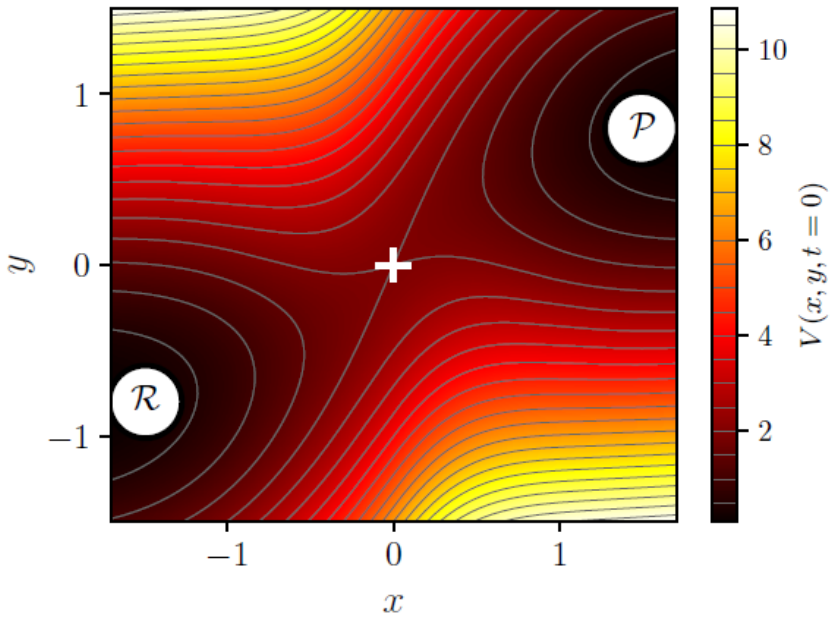


Figure 2.5: The potential energy landscape $V(x, y)$ of the driven two-dimensional model system described by Eq. (2.7) for the standard parameters and time $t = 0$. The reactant $\mathcal{R}$ and product $\mathcal{P}$ configurations are located at the local minima of the potential and are separated by a barrier. The white cross marks the saddle point. This figure is taken from Ref. [17].

For this particular two-dimensional and time-dependent system, the corresponding NHIM is also a two-dimensional surface which is oscillating in its phase space. Its positions can be determined by the BCM presented in Sec. 2.1.2 with very high precision of 10^{-12}, as shown in Ref. [30]. Trajectories on the NHIM are calculated by propagating an initial position on the NHIM according to the classical equations of motion that can be derived from the potential of Eq. (2.7). For this propagation, the numerical *Verlocity Verlet* integrator [35] is used. When propagating particles on the NHIM, is must be kept in mind that trajectories on the NHIM are unstable, as explained in Sec. 2.1.2. Since a numerical integrator can never be perfect due to numerical limitations, the trajectories will depart from the NHIM eventually. In order to prevent that, they are stabilized by projecting them back on the NHIM from time to time during the propagation. Suppose a trajectory initialized on the NHIM with precision 10^{-12} with the BCM is propagated for a certain amount of time Δt, e.g. one oscillation period T. After this time the trajectory at $(y_{\text{traj.}}(\Delta t), v_y^{\text{traj.}}(\Delta t))$ might deviate from the NHIM by Δx and Δv_x along the unstable reaction coordinate. In order to determine the position at which the trajectory should be, the NHIM's position $(x^{\text{NHIM}}, v_x^{\text{NHIM}})$ is calculated for the given orthogonal mode coordinates of the trajectory $(y^{\text{traj.}}, v_y^{\text{traj.}})$ via the BCM at time Δt. Under the as-

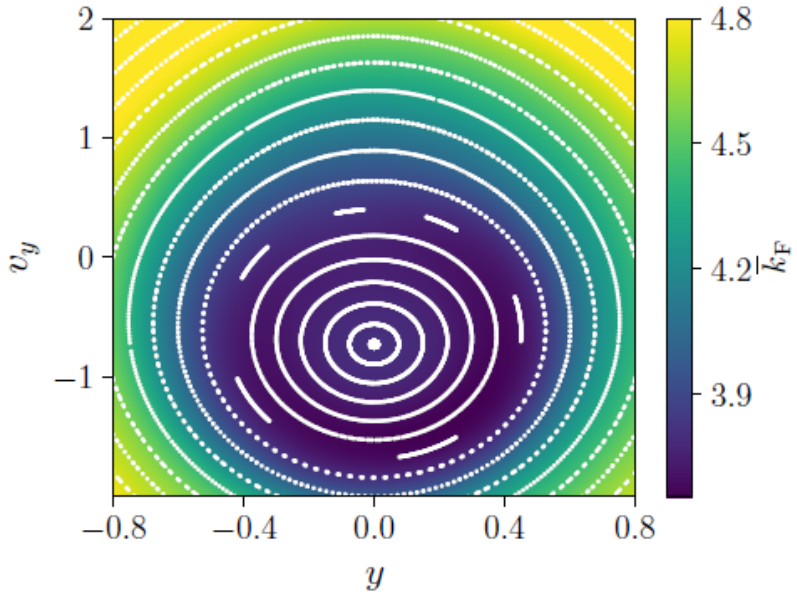


Figure 2.6: A Poincaré surface of section (PSOS) of the plane (y, v_y) of trajectories on the NHIM of a driven two-dimensional model system given by Eq. (2.7). The quasi-periodic trajectories on the NHIM move on stable tori, shown as the white dots. The white fixed point in the center marks the presence of an exactly periodic trajectory. The trajectories' mean Floquet rates $\overline{k}_\mathrm{F}$ are given as a color encoding in the background. The closer trajectories are to the elliptic fixed point in the center, the lower their associated Floquet rate becomes. This figure is taken from Ref. [17].

sumption that the trajectory's deviation from the NHIM is considerably smaller than its movement along the stable degree of freedom, the trajectory's position is projected back onto the NHIM by simply relocating it to the position $(x^\mathrm{NHIM}, v_x^\mathrm{NHIM})$ and effectively setting Δx and Δv_x to zero.

By repeatedly stabilizing the trajectories via projecting them back onto the NHIM, they can be calculated for very long time scales which is crucial for the calculation of, for example, a Poincaré surface of section (PSOS). A PSOS is obtained by plotting positions (y, v_y) of multiple trajectories on the NHIM at multitudes of the period T of the external driving. For a better understanding of the dynamics on the NHIM of the two-dimensional model system, Figure 2.6 shows such a PSOS. The resulting image reveals one exactly periodic trajectory, known as a *fixed point*, in the center and a variety of quasi-periodic trajectories moving around it on stable tori. In Ref. [36], it was shown that additional fixed points can emerge via *bifurcations* when tuning the parameters of the model system. The image also shows Floquet rates, presented in Sec. 2.1.3, in

form of a color-encoding, revealing that they are constant along the individual tori. The Floquet rate is minimal for the exactly periodic trajectory and grows continuously for trajectories at higher absolute values of y and v_y.

Calculating positions on the NHIM, propagating particles on the NHIM, as well as calculating decay rates can become computationally expensive very fast, especially in higher-dimensional and time-dependent systems. Besides, due to the nonlinearities of the sytems, as well as their potentially numerous degrees of freedom, understanding and analyzing the dynamics on the critical manifold might become extremely difficult because they become increasingly hard to interpret. In the past, machine learning (ML) was proven to be a helpful tool for the analysis of complicated and nonlinear systems. In Refs. [16, 37], ML algorithms were used to describe the NHIM continuously by interpolating single points obtained via the BCM. In Ref. [38] decay rates were predicted given only only the initial position of a trajectory on the NHIM without having to do any propagation. In this thesis, ML is used to increase the interpretability of the dynamics on the NHIM of the model system by reducing its dimensionality to only the space of the NHIM. In addition, other ML techniques are used to analyze, reproduce and predict trajectories in this reduced space. In doing so, the propagation and stabilization of trajectories on the NHIM could potentially be supported or even replaced by much faster ML algorithms.

2.3 Machine learning techniques

Nowadays, machine learning (ML) algorithms are present in many technical devices that we use on a daily basis, ranging from video filters to smart home assistants like Amazon's Alexa. ML is used for various tasks like prediction, image recognition, speech recognition and even for medical diagnoses, to only name a few. Also in natural sciences, including theoretical physics, the variety of ML algorithms and their ability to interpolate large data sets, detect anomalies, cluster data, etc. create a manifold of new possibilities for doing research.

Generally, ML is an algorithmic approach that uses statistical analysis of input data in order to take certain actions or make predictions. While doing so, the algorithms analyze the data and improve their performance on the given task without an explicit prescription to do so. Due to this autonomous improvement over iterations, the algorithms are regarded as *learning* from the data. ML algorithms can be distinguished by three different learning paradigms: supervised learning, unsupervised learning and reinforcement learning. The following paragraphs give a short overview on supervised and unsupervised learning, as well as a mix between the two, the self-supervised learning.

In supervised learning, the data handled by the model is already labeled. For instance, in an image classification task, the ML algorithm is presented with images $\boldsymbol{x}$ that were previously labeled with the corresponding category tag $\boldsymbol{y}$ that the model is then trained to predict. The task for the model is to find a mapping function $\mathcal{F}_\theta(\boldsymbol{x}) = \boldsymbol{y}$ with parameters θ, that predicts the labels for the given images as accurately as possible. Usually, the target vector $\boldsymbol{y}$ is n-dimensional, with n being the number of all possible categories. In each entry, the probability for the corresponding category is given. While training, the parameters θ of the mapping function are optimized in order to minimize the loss $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ which is a measure of the difference between the target vector given by the model $\boldsymbol{y} = \mathcal{F}_\theta(\boldsymbol{x})$ and the target output $\hat{\boldsymbol{y}}$.

In unsupervised learning, the ML model handles unlabeled data, meaning there is no given target vector $\boldsymbol{y}$. The goal of unsupervised learning in ML is to detect features and structures within the distribution of the input data. The two main applications are the clustering of data and the reduction of the dimensionality of a given data distribution. This thesis revolves around the latter of these two applications. Oftentimes, the dimensionality reduction leads to a useful representation of the input data and the resulting lower-dimensional space can be used for further analysis with other methods. Examples for methods used for dimensionality reduction are the classic Principle Component Analysis (PCA) [39], as well as autoencoder neural networks [40] with the latter being the focus of this thesis.

A mix between supervised and unsupervised learning is the *self-supervised learning*. In this case, there is no prior labeling of data involved and the model rather creates the target vector itself during the training process. Such a learning paradigm is oftentimes used when wanting to roughly optimize the parameters of a large neural network model with unlabeled data before fine-tuning them for a specific task, for which labeled data is available. Here, a model is pre-trained on a *pretext task* with unlabeled data. Afterwards, the model is presented with labeled data in order to perform a more complex subsequent task, the *downstream task*. Typical applications of such models can be found in computer vision [41–43], where the networks at play are generally very large and there exists a lot of unlabeled data.

2.3.1 Feed forward neural networks

A very popular ML method is the application of artificial neural networks inspired by the biological neural networks found in the brain. A single neuron i is modeled as a perceptron [44] and approximated by the following equation:

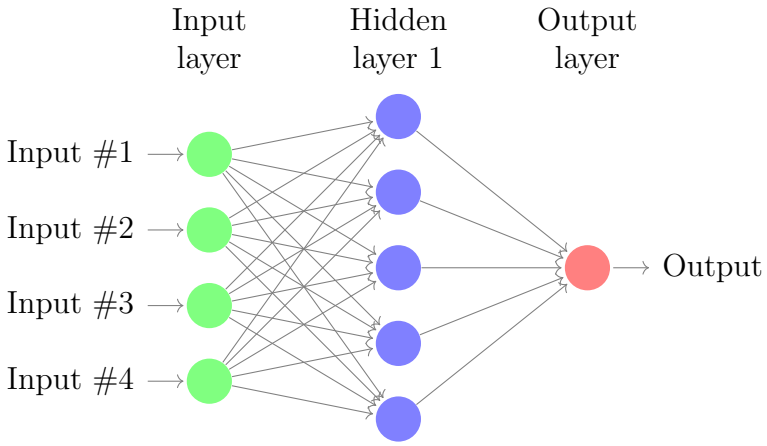


Figure 2.7: Sketch of an exemplary fully connected feed forward neural network architecture. This particular network has input dimension four, one hidden layer with dimension five and a single node in the output layer.

$$y_i = \text{Perceptron}_i(\boldsymbol{x}, \theta) = f\left(\sum_{j=1}^{N} W_{ij} x_j + b_i\right) . \tag{2.8}$$

The function parameters are given by the parameter $\theta = \{\boldsymbol{W}, \boldsymbol{b}\}$ which consists out of the weight matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$. The number of input points – corresponding to the number of synaptic input connections in the biological brain – is given by the parameter N. The output value of the neuron is given by the scalar function f of the sum over the product of the weight matrix and all input points, as well as the bias vector. The function f is known as the activation function and is typically chosen to be nonlinear and differentiable, as this enables the network to solve more complex tasks.

When stacking multiple neurons together to form a layer, one comes up with a *single-layer perceptron*. Nowadays, in deep neural network models, multiple layers are connected in order to create a multi-layer perceptron (MLP). Layers that do not classify as the input or output layer are called hidden layers. In Ref. [45] it was proven that models with a minimum of one hidden layer can be used as universal function approximators. Figure 2.7 shows the sketch of an exemplary neural network with four input neurons in the input layer, one output neuron in the output layer and one hidden layer in between. Each node is connected to all nodes of the previous layer. For such a model, the outputs y_i for inputs x_i are given by

$$y_i = \mathrm{MLP}_i(\boldsymbol{x}, \theta) = f^{(2)} \left(\sum_{j=1}^{N^{(2)}} W_{ij} \cdot f^{(1)} \left(\sum_{k=1}^{N^{(1)}} W_{jk}^{(1)} x_k + b_j^{(1)} \right) + b_i^{(2)} \right) . \tag{2.9}$$

There are multiple ways in which the nodes of a neural network can be connected to each other. When there are cyclic connections of the nodes in a neural network, it is classified as being a recurrent neural network (RNN). These types of networks are suited for handling temporal sequences and are presented in more detail in Section 2.3.3. In a feed forward neural network the connections do not form a cycle and are oriented towards the respective subsequent layer, as can be seen in the model illustrated in Figure 2.7. Whenever all nodes of one layer have a connection to all the nodes of the previous layer, the network is said to be fully connected.

Activation functions

Originally, the activation function was chosen to be the Heaviside step function given by

$$f(x) = \begin{cases} 1\,, & \text{for } x \geqq 0 \\ 0\,, & \text{for } x < 0 \end{cases} . \tag{2.10}$$

Consequently, the neurons' outputs were binary and would either pass one or zero to the neurons of the subsequent layer. However, neural networks with neurons of this type are not suited to perform more complex tasks. This is why nowadays, the activation function is usually chosen to be nonlinear, differentiable and smooth.

A differentiable and smooth alternative to the Heaviside step function is the sigmoid function defined by

$$\sigma(x) = \frac{1}{1 + e^{-x}} . \tag{2.11}$$

Since the sigmoid function lays in the range between zero and one, it is often used as the activation function of the last layer of a neural network that is used to predict a probability, for example the probability for the category of an image in a binary classification task.

Another variant of a sigmoid function is the hyperbolic tangent given by

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} . \tag{2.12}$$

The advantage of using a tanh as the activation function compared to using the sigmoid function of Eq. (2.11) is that the tanh exists in the range from -1 to 1. Consequently, inputs of zero are also mapped near zero and negative inputs are mapped negatively.

Another popular activation function is the Rectified Linear Unit (ReLU) [46], which is estimated to currently be the most commonly and widely used activation function. The ReLU function is defined by

$$\mathrm{ReLU}(x) = \max\{0, x\} = \begin{cases} x\,, & \text{for } x > 0 \\ 0\,, & \text{for } x \leqq 0 \end{cases}. \tag{2.13}$$

The ReLU function is oftentimes used in convolutional neural networks [47] and stands out by its simplicity and constant derivative. Such a constant derivative can be of great advantage when wanting to prevent vanishing gradients in the training of very deep neural networks.

Loss functions

Before a network can be trained on a given data set, a loss function has to be defined. Neural networks usually seek to minimize the output of the loss function, simply known as the *loss*, by optimizing their parameters. The loss function has to evaluate the output of the model in a way that ensures that a better loss corresponds to a better model. Choosing the right loss function for the given task is crucial, since it is the network's only description of what the optimization task is about.

For a regression problem of predicting a real-valued quantity, a very popular and simple loss function is the *Mean Squared Error* (MSE), which is simply the squared Euclidean distance of two vectors, given by

$$\mathrm{MSE}\,(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N} \sum_{i}^{N} (y_i - \hat{y}_i)^2\,, \tag{2.14}$$

where N is size of the model's output vector $\boldsymbol{y}$ and target vector $\hat{\boldsymbol{y}}$.

Other common loss functions, mainly used for classification tasks, where the model predicts probability distributions, are for example the *Cross-Entropy*, also known as the *Logarithmic loss*, given by

$$H(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i} y_i \log\left(\hat{y}_i\right) \tag{2.15}$$

or the Kullback-Leibler divergence given by

$$D_{\mathrm{KL}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_i y_i \log\left(\frac{y_i}{\hat{y}_i}\right) . \tag{2.16}$$

It should also be noted that any loss function can easily be extended by an additional term, further specifying the given problem or preventing overfitting.

Training

At the start of the training of a neural network, its weights are initialized randomly. During training, the model's parameters are then optimized in order to minimize the value of the chosen loss function. There are many different algorithms to choose from for the optimization of the function given by a neural network. However, the most commonly used method is through gradient descent, implemented by the *backpropagation* algorithm [48]. Here, the loss $\mathcal{L}(y, \hat{y})$ is minimized by calculating its gradients with respect to all weight parameters θ of the model by the chain rule. A weight update via gradient descent can be given by

$$\theta_{\mathrm{idx}+1} = \theta_{\mathrm{idx}} - \eta \bigtriangledown_{\theta_{\mathrm{idx}}} \mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) , \tag{2.17}$$

where idx is the iteration step index and η, known as the *learning rate*, is a hyperparameter that controls the step size of the algorithm. The loss gradient is calculated layer by layer and is propagated back through the model.

When using a pure gradient descent method, finding the global minimum of the loss landscape is only guaranteed if the loss landscape is convex and has no local minima. However, in most cases, there are multiple local minima in which the algorithm can get stuck when relying on pure gradient descent. A very popular method that deals with this issue is the *stochastic gradient descent* (SGD). Here, the loss signal is calculated for stochastically sampled subsets of the dataset, the mini-batches, instead of for the whole dataset at once. The model is said to be training for one *epoch* after it has processed the whole training dataset once.

A very popular optimization algorithm in deep learning is the *Adam* [49] optimizer, which is an extension to SGD. Its key feature is that, in contrast to classical SGD, the learning rate η is uniquely defined for each weight parameter and is adapted during the training process. The Adam optimizer was shown to be very effective, especially on strongly non-convex loss landscapes.

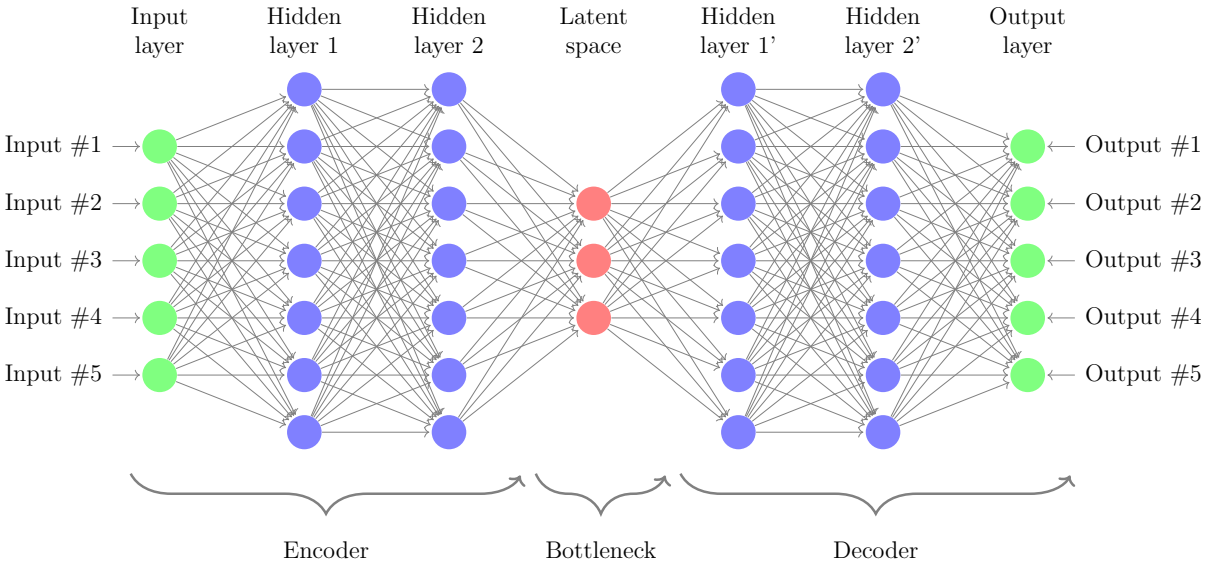


Figure 2.8: Sketch of an exemplary autoencoder neural network architecture. This autoencoder is symmetrical and can be split into an *encoder* and a *decoder* part of equal size. The decoder takes the information given in the five-dimensional input layer, passes it through the two hidden layers and *encodes* it to dimension three in the model's *bottleneck* or *latent space*. The decoder part takes the outputs of the latent space nodes and *decodes* them back into the original five-dimensional input space. The AE is trained to reconstruct the input data.

2.3.2 Autoencoders

Autoencoders (AE) [40] are a special type of artificial neural network and are mostly used to discover structures and features in a given dataset, as well as reduce its dimensionality. An AE poses an unsupervised learning task as a supervised one by defining its input vector $\boldsymbol{x}$ as the target vector $\hat{\boldsymbol{y}}$ and comparing it to the respective output vector $\boldsymbol{y}$ of the model. Hence, an AE's goal is to reconstruct the input data it is given.

An AE model consists out of three parts: the encoder, the bottleneck or latent space (LS) and the decoder. The encoder encodes the input data into the lower-dimensional latent space while the decoder decodes this representation back into the original space. Figure 2.8 shows a sketch of an exemplary AE model with input dimension five, latent space dimension three and two hidden layers in the encoder and decoder part, respectively. Due to this special architecture, the five-dimensional input is first broken down to only three dimensions in the model's latent space and is then decoded back into the original five-dimensional space while being processed by the model.

In order for the input data to be fully reconstructed in the model's output layer, it

is cruical that the latent space contains a full representation of all relevant quantities describing the input data. Therefore, the dimension of the LS cannot be smaller than the input data's lowest-dimensional representation in the real space. On the other hand, if the LS dimension is chosen to be considerably bigger than the lowest possible dimension, the AE's LS might simply be a copy of parts of the input data and is therefore not an encoding of only the deep correlations of the dataset. It should also be noted that an AE can only be successful if the features of the given dataset are correlated in some way, because otherwise, the data can never be fully captured in a lower-dimensional space like the LS.

As mentioned above, the AE is trained on reconstructing the input data. It does so by minimizing the *reconstruction loss* which is the error between the input and output data. The loss function is usually chosen to be the MSE defined in Eq. (2.14).

In comparison to other methods for reducing the dimensionality of a dataset, an AE stands out by its ability to capture nonlinear correlations as well as to easily decode the lower-dimensional representation back into the original space. In contrast to other methods for dimensionality reduction like for example the popular *t-distributed stochastic neighbor embedding* (t-SNE) clustering algorithm [50], AEs trained on a certain space are able to process new data from the same distribution and represent it in the existing LS.

Due to the above mentioned properties of AEs, their application is suited for a variety of given tasks besides dimension reduction, such as anomaly detection [51] and denoising of data [52]. Anomaly detection is a common task for AE models, since a fully trained model has learned to represent a dataset in its most compressed form and captures all the relevant correlations between its key features. Consequently, when the AE is confronted with data that does not suit these very specific descriptions, the representation in the LS and therefore the reconstruction in the output layer fails and the data is declared an anomaly. An AE used for the denoising of data is confronted with noisy input data and compares the resulting output to the noise-free version of the input data. Since the LS only captures the key features of the input data, the random and correlation free noise is usually ignored, passing a representation of a noise-free input to the decoder part.

2.3.3 Recurrent neural networks

Unlike normal feed forward neural networks, recurrent neural netwoks (RNN) learn how to make decisions not only based on the current input but also based on previous inputs via an internal state. This property makes RNNs suited for dealing with sequences of data, where the next state cannot be predicted by simply looking at the current state and information about the data's history is indispensable.

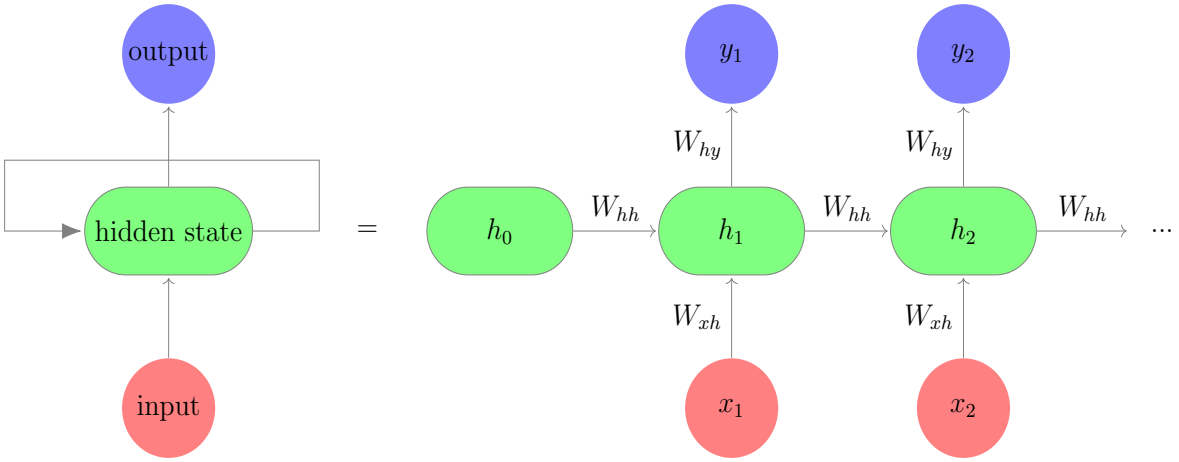


Figure 2.9: Sketch of a RNN cell. The RNN cell consists out of an input and an output layer with a hidden state and a recurrent connection in between. When processing a temporal sequence x_t for $t = 1, .., n$, the intermediate representation of the hidden state h_t is drawn from the current input point x_t, as well as the previous hidden state h_{t-1}. This is how the network keeps a history of all previous data points of the sequence and predicts the final output y_n according to this description. The weight parameters W_{hh}, W_{xh} and W_{hy} are kept constant throughout the processing of the whole sequence, making the model *recurrent*.

There are many different types of RNNs. One of the simplest types is the Elman [53], simply referred to as *RNN*. Generally, a RNN cell consists out of an input and an output layer, as well as a hidden layer, also referred to as the *hidden state*, as it is shown by the sketch of a simple Elman RNN in Figure 2.9. When training the RNN model on a sequence, it processes the data points one by one in the order of the sequence. At the start of the training, the hidden state h_0 is initialized independently of the input data. When processing the first input x_1, an intermediate representation h_1 is build in the hidden state of the cell which not only includes information about the input point but also about the previous hidden state h_0. The same is repeated for the subsequent input point x_2, whose representation in the hidden state h_1 also includes information about the previous hidden state h_1. This procedure is repeated for the whole length of the sequence, which eventually leads to a hidden state that includes information about the history of all data points and from which the final output is determined.

The RNN cell of Figure 2.9 includes three sets of weight parameters: W_{xh} for calculating the hidden state from the given input point, W_{hh} for calculating the hidden state from the previous hidden state and lastly W_{hy} for the calculation of the output from the current hidden state. The network's parameters also include two bias vectors b_h and b_y for the calculation of the hidden state h, as well as the output y. With t being the

sequence index, the hidden state and output is given by

$$h_t = W_{xh} + W_{hh}\, h_{t-1} + b_h\,, \tag{2.18}$$

$$y_t = W_{hy}\, h_t + b_y\,. \tag{2.19}$$

During the processing of a sequence of data, these weight parameters do not change, which explains why the RNN is *recurrent* and why the RNN cell in Figure 2.9 is illustrated with a recurrent connection for the hidden state. They are only updated after the final output is predicted and it can be compared to the target vector to estimate a loss. This error can then be backpropagated through time to reach all previous time steps, since all hidden states are remembered and chained together.

However, backpropagating an error through many time steps can lead to a *vanishing gradient* which means that earlier time steps have little to no influence on the adjustment of the weight parameters and therefore the estimation of the output is mostly based on the latest input points of the sequence. This is why a RNN cell might not be capable of capturing dependencies over long time scales and is said to only have a short-term memory. There are two common architectures used to mitigate the issue of short-term memory in RNNs: the Long Short-Term Memory model (LSTM) [54] and the Gated Recurrent Units model (GRU) [55]. Both of these models use *gates* which learn what information to add or remove for the calculation of the hidden state.

3 Methods

This chapter presents all methods used for the parameterization of the NHIM and the prediction of trajectories in the autoencoder's latent space.

Section 3.1 presents the dimension reduction of the NHIM via two different autoencoder models. While one is trained on the oscillation phase of the NHIM as the timely input, the other one takes the absolute time as an input and its latent space has to be extended artificially when looking at trajectories in it for longer times.

Section 3.2 explains four methods that aim to find descriptions for the dynamics in the reduced space of the NHIM and predicting new trajectories starting from anywhere in this space. The methods presented in Sections 3.2.1 and 3.2.2 deal with finding a description of a differential system of equations of the system's dynamics. One method uses a genetic programming algorithm to perform a symbolic regression analysis in order to find an analytical description for it, while the other one uses a feed forward neural network to describe the system's differential system of equations. In both cases, one can generate new trajectories starting from anywhere in the latent space by solving the respective description numerically. Another method has the goal to train a feed forward neural network to interpolate between trajectory data points in the reduced space in order to have a direct description for trajectories in the known ranges of the training data. Section 3.2.4 introduces a special kind of artificial neural network that is commonly used for time series data, the recurrent neural network. Here, a combination of a feed forward and a recurrent neural network is trained to iteratively predict the time sequence of trajectories in the autoencoder's latent space.

3.1 Parameterizing the NHIM

3.1.1 Autoencoder – absolute time input

When reducing the dimensionality of a space with an autoencoder, it is crucial to match the autoencoder's architecture to the given task. The number of input neurons must match the number of variables describing the space and the number of latent space neurons cannot be smaller than its lowest-dimensional representation. Otherwise, the

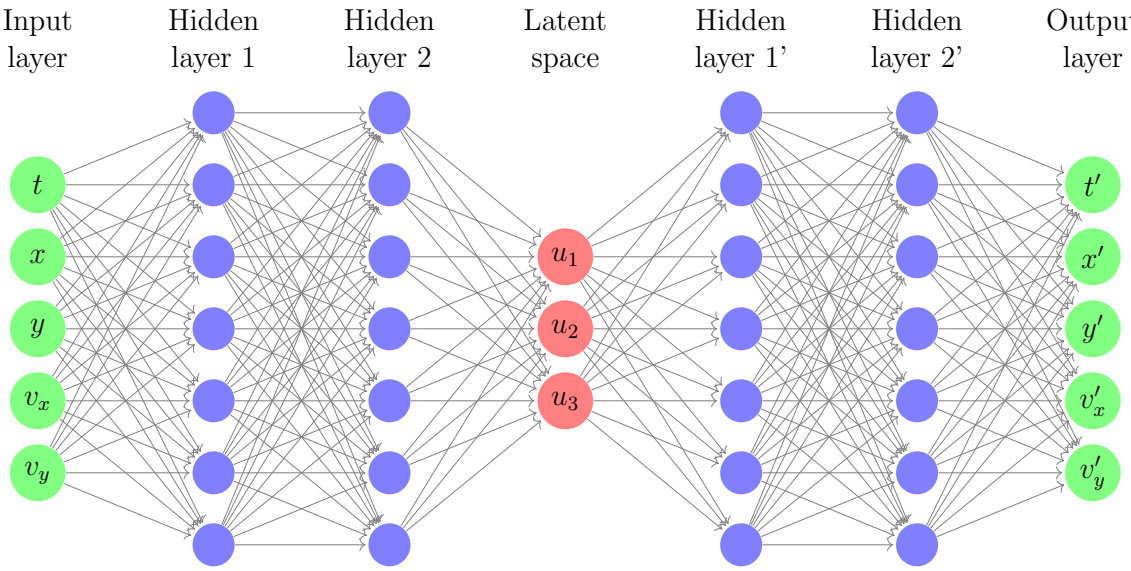


Figure 3.1: Sketch of an exemplary architecture of an autoencoder used for the parameterization of the NHIM. The input layer (t, x, y, v_x, v_y) is given by the system's time as well as the NHIM's position in the phase space. The coordinates of the autoencoder's latent space are denoted with u_1, u_2 and u_3. Since the goal of an autoencoder is to match the output to the input, the output layer is given by (t', x', y', v'_x, v'_y).

parameterization of the space via an autoencoder will not be successful, as is showed in Section 4.1.1.

For the task of parameterizing the NHIM it is important to understand its dimensions. A single point on the oscillating NHIM is defined through its coordinates in the system's phase space with positions (x, y) and velocity components (v_x, v_y) as well as the time t. This results in a five-dimensional space. The advantage of taking the absolute time instead of the oscillation phase as an input variable is that such a model is not only able to parametrize periodic systems but also non-periodic systems with any given time dependency. As it is known from Section 2.1.2, the NHIM is a two-dimensional object in the phase space. Hence, in combination with the time information, it can be seen as a three-dimensional manifold in a five-dimensional space. Consequently, an autoencoder suited for the task of reducing this five-dimensional space to only the three-dimensions of the NHIM would have input dimension five and a latent space dimension of three.

Figure 3.1 shows an exemplary architecture of such an autoencoder with two hidden layers in the encoder and decoder part respectively. The latent space coordinates, freely chosen by the autoencoder will be referred to as u_1, u_2, and u_3 throughout this work.

Extension of the latent space

Artificial neural networks are well suited for interpolation of data but will usually not perform well for extrapolation. Autoencoders map data outside of the space they were trained on back to this space. This can be an advantage when using the network for denoising of data but can be a problem when wanting to use a trained autoencoder to handle data outside of the training data ranges.

Since the NHIM is exactly periodic, in principle, a latent space generated by training a network with data of only one oscillation period contains all relevant information for describing the NHIM for all time. However, trajectories on the NHIM are mostly quasi periodic, meaning they have to be observed over longer time scales than that in order to fully capture their dynamics. One could solve this problem of coming up with a LS, that does not cover a large enough time scale, by simply training the autoencoder on a large amount of oscillation periods instead of only a single one. However, doing so comes with big drawbacks such as the linearly growing computational power with amount of training data. This section presents another way to come up with a LS in which trajectories can be represented continuously for all time. The method is specifically designed for the three-dimensional LS of the AE with absolute time input. It should be noted that this method is only suited for exactly periodic systems.

The idea is to artificially extend the latent space, describing only one or very few oscillation periods, by duplicating it and stacking it along the time axis. This is only easily possible if the latent space is very symmetrical and the time axis of the original space is straight in its compressed form in the LS as well. It should be noted that there is no guarantee for this and many autoencoders generate latent spaces in which the time axis is curved. For such models, the extension of the latent space is not possible with the method described here. The likelihood of a straight axis is increased for models trained on data of more than just one oscillation period. Figure 3.2 (a) shows an exemplary autoencoder's latent space describing two oscillation periods of the NHIM. The space is very symmetrical and the time axis is linear, making it very well suited for an artificial extension in time.

Duplicating this latent space twice and then stacking it along the time axis results in a latent space shown in Figure 3.2 (b). The new space is now a description of the same system but for six instead of two oscillation periods and is invariant under the transformation $t \to t + n\,T$, where t is the system time and T is the oscillation period. The shifts between the duplicated latent spaces segments are calculated in a way that ensures that trajectories on the NHIM remain as smooth as possible in the transitions between the extensions. In order to do so, a trajectory is calculated and projected to the time interval of the first two oscillation periods, so it can be fully displayed in the autoencoder's latent space. Afterwards, one can calculate the distances between its position at the end of the given time interval and its respective subsequent position that

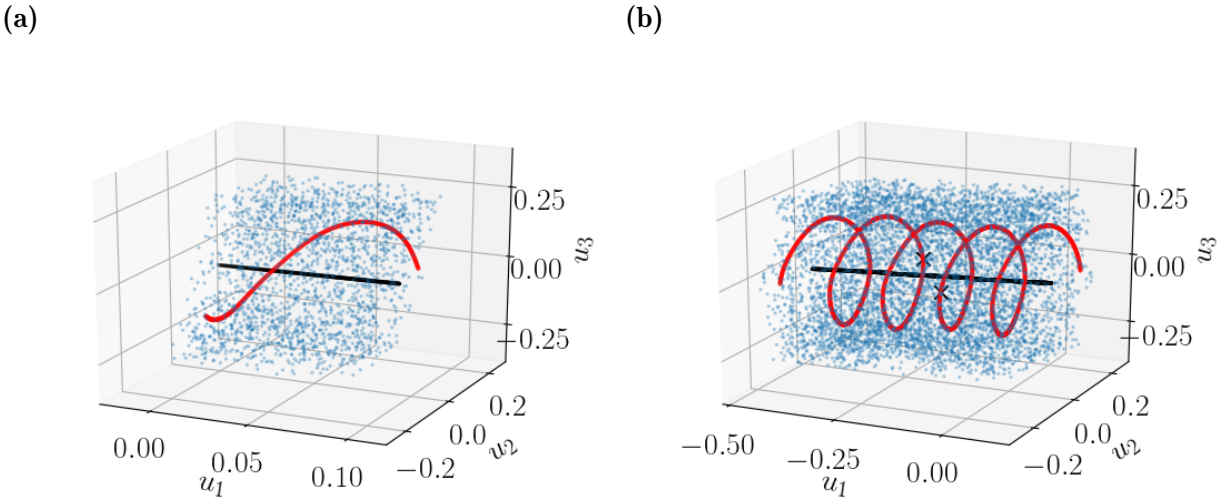


Figure 3.2: (a) The latent space (u_1, u_2, u_3) of an autoencoder with a configuration of $[5 \to 5 \to 4 \to 3 \to 4 \to 5 \to 5]$ nodes per layer trained to convergence on positions of two oscillation periods of the NHIM of the two-dimensional model system given by Eq. (2.7). The NHIM's positions, transformed to the latent space, are given by the blue points. The black line shows the latent space representation of the time axis of the original space. The red line represents the course of a trajectory on the NHIM for two periods of the external driving. (b) The artificially extended latent space (a), covering a time interval of six instead of only two oscillation periods of the NHIM. The black crosses mark the intersections of the three latent space segments that were stacked together as explained in Section 3.1.1.

is now mapped to the start of this time interval. This distance is approximately equal to the shift one has to perform on the duplicated latent space in order to have a smooth trajectory in the extended space. The shift is calculated for a total of 100 trajectories for each extension step individually and then averaged over all trajectories.

By using this method, it is possible to observe the dynamics in a latent space that covers the whole time scale without having to do computationally expensive training of the model.

3.1.2 Autoencoder – oscillation phase input

When reducing the dimensionality of periodic systems, considering the oscillation phase instead of the absolute time can be of great advantage. In doing so, the system is fully described by just one single oscillation period and even quasi periodic trajectories can be represented fully in this space since every position on the NHIM in combination with the

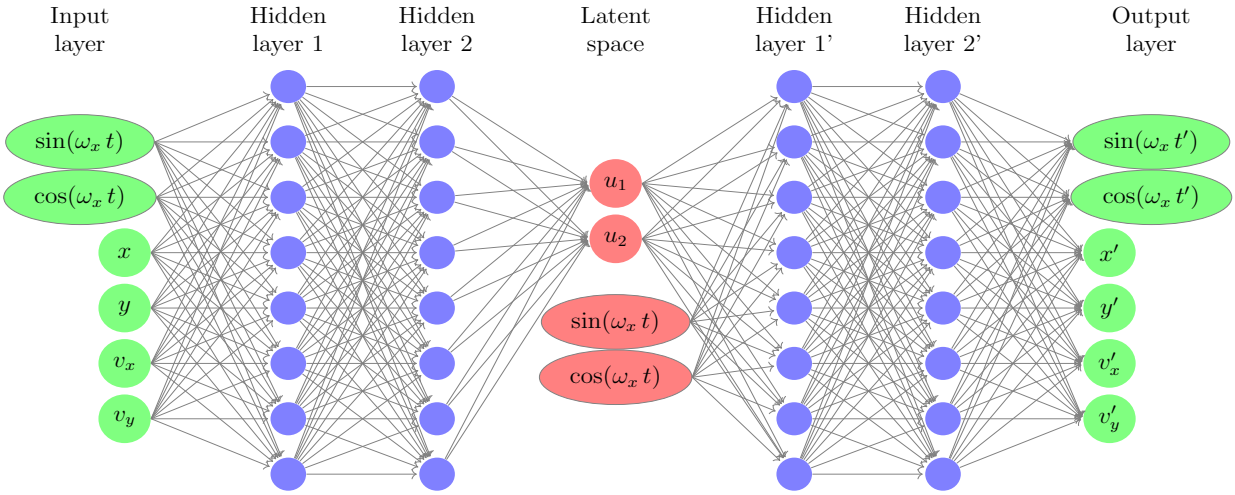


Figure 3.3: Similar to Figure 3.1, this sketch illustrates an exemplary autoencoder architecture. However, besides the NHIM's position (x, y, v_x, v_y) in the phase space, the input of this model is now also given by its phase information via $\sin(\omega_x\, t)$ and $\cos(\omega_x\, t)$. The encoder generates a two- instead of a three-dimensional latent space which is a lower-dimensional representation of the NHIM's phase space position. The phase information is given to the decoder part separately and is not learned by the encoder. Consequently, the latent space is four-dimensional, while only two dimensions are constructed by the autoencoder itself. The decoder part then processes this information in order to reconstruct the input.

system's oscillation phase is unique. Therefore, an artificial extension of the resulting latent space is no longer needed to capture the entire system's dynamics.

Figure 3.3 shows a sketch of an exemplary architecture of an autoencoder suited for the task of parametrizing the NHIM while considering its oscillation phase as an input. The input variables are the phase information in the form of $\sin(\omega_x\, t)$ and $\cos(\omega_x\, t)$, as well as the position in the phase space given by x, y, v_x and v_y. As it is shown in Section 4.1.1, the phase information of a periodic system can not be broken down to a single dimension in the latent space. Therefore, in order for the decoder to be able to reconstruct the input, the latent space dimension must be larger than three. In the particular architecture used throughout this work, the encoder reduces the six input dimensions to only two in latent space nodes which correspond to the spatial information about the NHIM in the phase space. While training, the phase information in form of $\sin(\omega_x\, t)$ and $\cos(\omega_x\, t)$ is combined with the resulting two dimensions of the encoder. In doing so, one comes up with a two-dimensional space that, in combination with the already known phase information, fully describes the NHIM and its dynamics for all time.

3.2 Predicting trajectories

3.2.1 Symbolic regression with gplearn

The method presented in this section deals with finding an analytical equation of the differential equation system of trajectories in the reduced space of the autoencoder presented in Section 3.1.1. The search for analytical equations is performed via a symbolic regression analysis. Symbolic regression is a variant of regression analysis for finding a mathematical expression that describes a given data set, while being as simple and accurate as possible. There are multiple ways to perform a symbolic regression analysis but what they all have in common is that the particular algorithm generates the expression all by itself and no model is given as the starting point. The initial expressions are built randomly by arranging constants, state variables, mathematical operators and analytic functions. The method presented here uses the most commonly used method for symbolic regression – *genetic programming*. The particular algorithm used here is an open-source software offered by *scikit-learn* called *gplearn*.

In genetic programming the algorithms start off from a population of randomly composed programs, in this case equations, and then alter them similarly to natural genetic processes, such as mutations and crossovers. By doing so repeatedly for many generations and letting only the fittest ones survive, the programs evolve towards a program which is more and more suited for the given task.

The gplearn algorithm implements this idea by first generating a population of equations in the form of *expression trees*. Figure 3.4 shows an exemplary expression tree using only the basic mathematical operators like addition, subtraction and multiplication. The algorithm then decides which ones of these expression trees make it to the next generation by splitting the population into smaller subsets and choosing only the fittest individual to be part of the next generation. This process is a *tournament*. The fitness of an expression is determined by the mean absolute error of the prediction given by the equation compared to the ground truth data. Besides simply reproducing the chosen expressions for the next generation, most of them are altered by several genetic operations. These include crossover, subtree mutation, hoist mutation and point mutation.

Crossover follows the principle of mixing the genetic material between individuals. In a first tournament the parent expression is selected and a random subtree of it is selected to be replaced. A randomly chosen subtree of the winner of a second tournament is then inserted into the parent, creating an offspring for the next generation. A sketch of this principle can be seen in Figure 3.5.

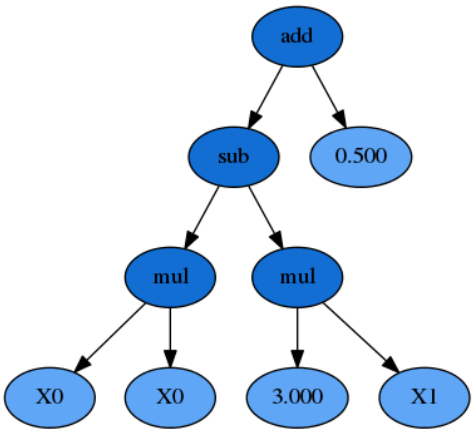


Figure 3.4: Sketch of an *expression tree* commonly used by symbolic regression algorithms. This tree is to be read from the bottom to the top and represents equation $Y = 0.5 + X_0 \cdot X_0 - 3 \cdot X_1$. This figure is taken from Ref. [56].

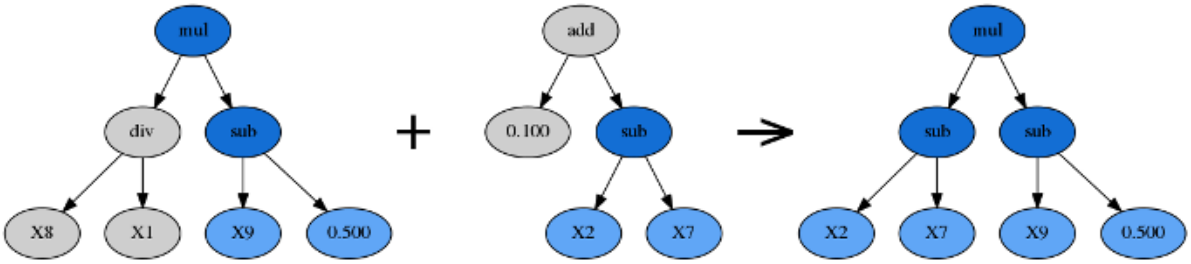


Figure 3.5: Sketch of the crossover principle. The subtree of an expression tree is replaced by the subtree of another one. This figure is taken from Ref. [56].

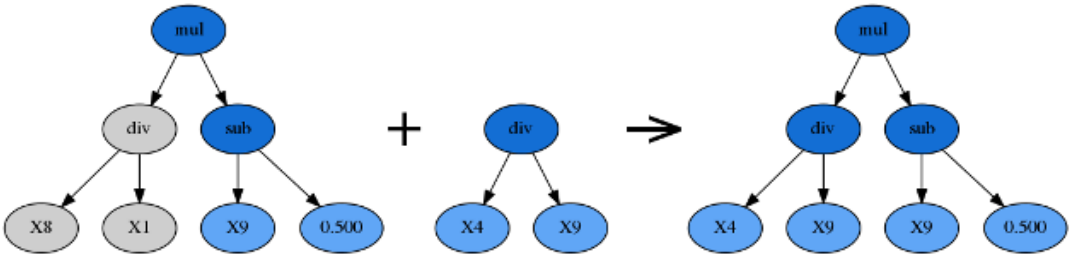


Figure 3.6: Sketch of the subtree mutation process. The subtree of an expression tree is replaced by a randomly generated subtree. This figure is taken from Ref. [56].

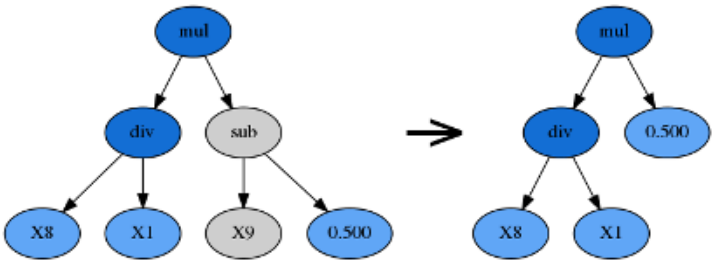


Figure 3.7: Sketch of the hoist mutation process. A subtree of a subtree of an existing expression tree is *hoisted* into the location of the original subtree. This figure is taken from Ref. [56].

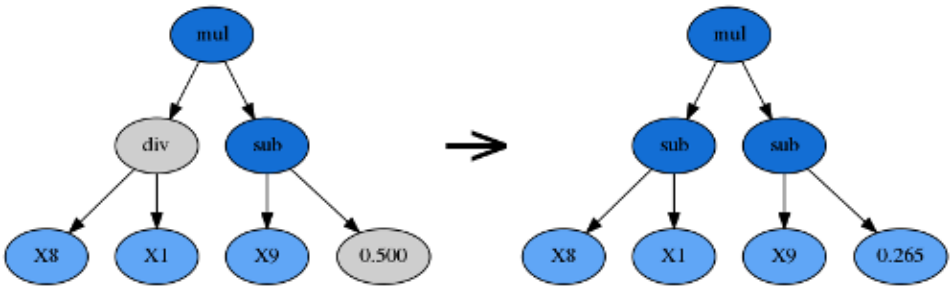


Figure 3.8: Sketch of the point mutation process. Here, randomly selected nodes of an existing expression tree are selected to be replaced by randomly chosen new ones. This figure is taken from Ref. [56].

Similar to the crossover principle, a random subtree of the winner of a tournament is chosen to be replaced for the subtree mutation. However, it is not replaced by the subtree of another expression with a high performance, rather a new subtree is generated randomly and taken as the replacement for the original subtree. Figure 3.6 shows an exemplary sketch of this process. With this mutation it is hoped to maintain diversity in the population by reintroducing functions that might have become extinct.

The hoist mutation is used to prevent overly long expressions in the population. Again, a random subtree of the winner of a tournament is chosen. But instead of it being replaced, a random subtree of this subtree is chosen to be *hoisted* into the location of the original subtree. A sketch of this process can be seen in Figure 3.7.

Lastly, the gplearn algorithm uses point mutation, one of the most common forms of mutation in genetic programming. Similar to the subtree mutation, it is used to maintain diversity in the population by introducing mathematical operators, variables and constants that are not necessarily part of the previous population. Random nodes of the winner of a tournament are selected to be replaced by ones that are randomly chosen by the algorithm. Here, nodes are replaced with ones of the same category, such as constants, variables and functions requiring the same number of arguments. Figure 3.8 shows a sketch of a point mutation performed for two nodes at once.

The ratio of how many expressions are altered with which mutation process can be chosen by the user. Furthermore, important parameters are the population size as well as a list of the mathematical operators which the algorithm can use for building expression trees. Another crucial parameter is the parsimony coefficient. Similar to the hoist mutation, it is used to fight bloating and controls a penalty by which expressions of smaller size are preferred compared to those of larger size. It is also possible to define functions to be used as building blocks in the expressions itself or as an estimate for the expression's fitness. Every function used is protected of returning an infinite quantity by the algorithm by introducing exceptional rules when necessary.

The evolution process comes to an end when either an expression reaches the predefined minimum accuracy or the number of generations reaches its predefined maximum value.

For the task of finding an analytical differential equation system for trajectories in the autoencoder's latent space, first, the training data for the gplearn algorithm is generated by calculating the numerical gradients in respect to the system time t, via *numpy gradient*. Then, the gplearn algorithm is used to find functions $f_{1,2,3}$ given by

$$\dot{u}_1 = f_1(u_1, u_2, u_3)\,, \tag{3.1}$$
$$\dot{u}_2 = f_2(u_1, u_2, u_3)\,, \tag{3.2}$$
$$\dot{u}_3 = f_3(u_1, u_2, u_3)\,. \tag{3.3}$$

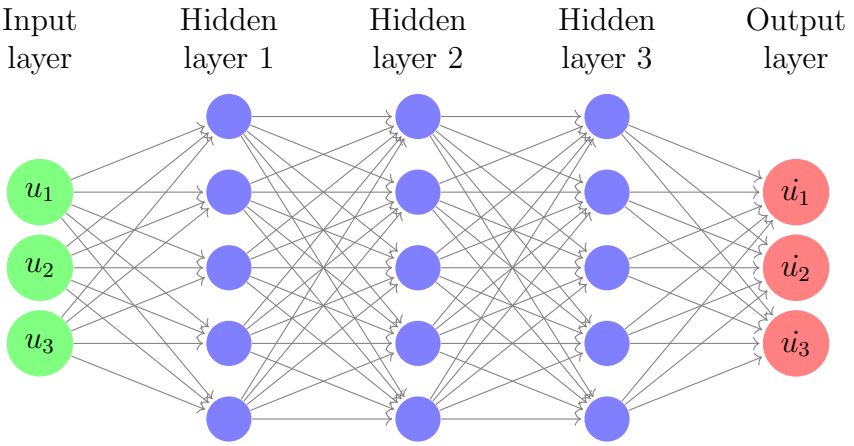


Figure 3.9: Sketch of an exemplary architecture of a feed forward neural network for the interpolation of the numerical gradients $(\dot{u}_1, \dot{u}_2, \dot{u}_3)$, given the latent space coordinates (u_1, u_2, u_3) of trajectories on the NHIM.

When suitable equations are found, the resulting analytical differential equation system can be solved numerically by using the function *odeint* from the open source library *scipy*. By doing so, new trajectories starting from anywhere in the sampled space can be generated very quickly and without having to use the simulation.

3.2.2 Neural network for equations of motion

Another way to come up with a description of the trajectories' equations of motion (EOM) in the latent space of the autoencoder, described in Section 3.1.1, is by using a feed forward neural network (NN). When having calculated the numerical gradients in respect to the time of the latent space coordinates $(\dot{u}_1, \dot{u}_2, \dot{u}_3)$ for a given set of (u_1, u_2, u_3) by using *numpy gradient*, a network as it is shown in Figure 3.9 can be used to interpolate the data. This method is referred to as the *EOM-NN method* throughout this work. The input dimension is defined by the latent space coordinates (u_1, u_2, u_3), while the network predicts the respective numerical gradients $(\dot{u}_1, \dot{u}_2, \dot{u}_3)$. Training such a network will lead to a description given by

$$\begin{pmatrix} \dot{u}_1 \\ \dot{u}_2 \\ \dot{u}_3 \end{pmatrix} = \mathrm{NN} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} . \tag{3.4}$$

In order to generate trajectories in the latent space, this prescription is solved numerically using the function *odeint* from the open source library *scipy*. Since the network

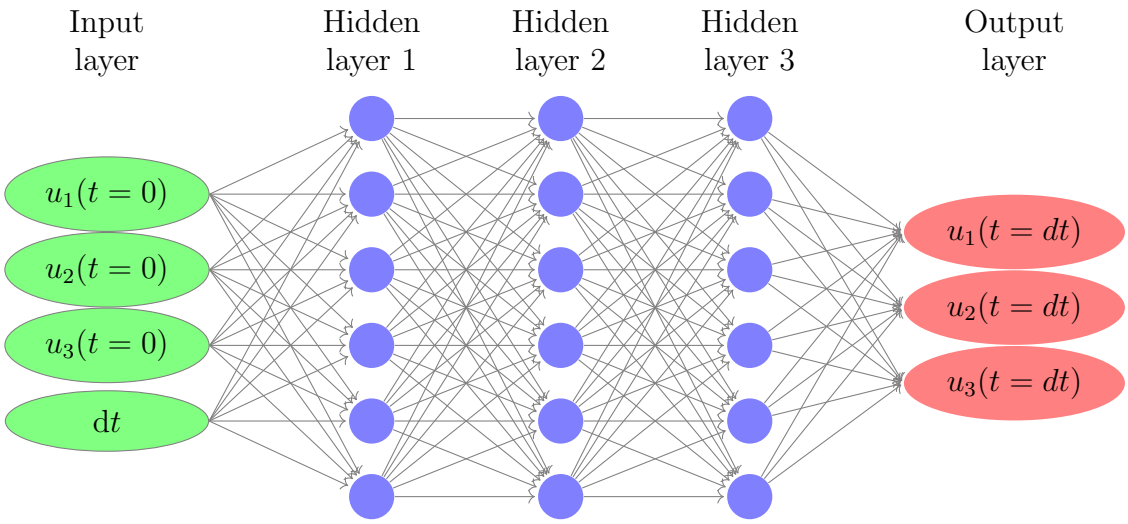


Figure 3.10: Sketch of an exemplary architecture of a feed forward neural network used to predict a trajectory's latent space position (u_1, u_2, u_3) at time $t = \mathrm{d}t$, given its position at time $t = 0$, as well as the desired time interval $\mathrm{d}t$ at which its position is to be predicted.

interpolates the data, it is possible to calculate a trajectory starting from anywhere in the sampled ranges of the space.

3.2.3 Neural network for prediction

When solving a differential equation system or a description of such numerically, the solution will eventually deviate over time if the description does not match the dynamics perfectly. Since the numerical solving is done iteratively, any deviation from the reference data will result in an increasingly growing error over time. In order to avoid such behavior, this method – the *prediction method* – does not differentiate between data of earlier or later times. Here, a feed forward neural network is used to predict the position of any trajectory at a given time step from only its initial position and the time delta information. Figure 3.10 shows an exemplary sketch of a model suited for this task. Its input dimension is the latent space coordinates of a given trajectory at time zero $u_1(t = 0)$, $u_2(t = 0)$ and $u_3(t = 0)$, as well as the time delta information $\mathrm{d}t$ at which the network should predict the trajectory's position. The output dimension is the trajectory's position at time $\mathrm{d}t$. This is an interpolation task and the resulting description is given by

$$\begin{pmatrix} u_1(t_0+\mathrm{d}t) \\ u_2(t_0+\mathrm{d}t) \\ u_3(t_0+\mathrm{d}t) \end{pmatrix} = \mathrm{NN} \begin{pmatrix} u_1(t_0) \\ u_2(t_0), \quad \mathrm{d}t \\ u_3(t_0) \end{pmatrix} . \tag{3.5}$$

3.2.4 Recurrent neural network model for prediction

The method presented in this section deals with the prediction of trajectories via an iterative approach. The trajectories lay in the LS of an AE of the type presented in Sec. 3.1.2 with the oscillation phase as an input. The advantage of an iterative approach is that the time scale for which the trajectories can be predicted is not limited, as for example in the non-iterative prediction method. However, the challenge of such approaches is that they are prone to errors that increase with each iteration step. This is an issue, especially for models that make a prediction solely based on the, perhaps faulty, prediction of the last time step. In order to mitigate this problem, the method presented here introduces a special kind of neural network, the *recurrent neural network* (RNN). As explained in Sec. 2.3.3, the special architecture of the RNN enables it to keep track of the history of a sequence, rather than making a prediction based on only the current input point.

Figure 3.11 shows a sketch of the structure of the model. It consists out of three building blocks: two feed forward neural networks and a combination of a RNN cell and a feed forward network. In order to distinguish it from the other methods presented above, this model is referred to as the *RNN model* throughout this work. Each building block has a different purpose. First, one of the feed forward networks encodes the initial position $\boldsymbol{u}_0$ of the respective trajectory into a higher dimensional vector $\boldsymbol{z}_0$. The resulting vector is later given as an input to the RNN cell and has the purpose of always providing the information to which starting point, hence to which trajectory, the currently processed points belong. This can help to prevent the model from confusing the courses of trajectories with the ones near by. In addition, the trajectories are deterministic, meaning that, in principle, their initial positions hold all of the information necessary to predict them for all time, making this information potentially very valuable to the network. The other feed forward neural network takes in the position $\boldsymbol{u}_\mathrm{t}$ at time step t and predicts the subsequent position $\boldsymbol{u}'_{\mathrm{t}+1}$ at time step $t+1$. In theory, any position in the LS alone holds all information necessary to predict the following time steps. However, the feed forward network is never perfect and thus, this resulting subsequent position is to be corrected first. In order to do so, it is passed on to the RNN cell, together with the position $\boldsymbol{u}_\mathrm{t}$ of the current time step and the encoded initial position $\boldsymbol{z}_0$ given by the other feed forward network. The RNN cell then learns to correct the predicted position $\boldsymbol{u}'_{\mathrm{t}+1}$ for a potential deviation from the course of the ground truth data. The main purpose of the subsequent feed forward network of the RNN cell is to adjust the dimensionality of

the output of the RNN cell to the three-dimensional deviation $\Delta \boldsymbol{u}'_{t+1}$. This deviation is then added to the prediction $\boldsymbol{u}'_{t+1}$ of the other feed forward network, resulting in the model's final output $\boldsymbol{u}_{t+1}$. Since this is an iterative approach, the model's output vector is then taken as its next input.

The training of such a model is a little more complex than that of the previous approaches. Instead of simply letting the network predict the 50 oscillation period long sequence of a trajectory at once, the model is trained to predict an increasing amount of time steps starting with only an average of three for the first 100 epochs. Every 100 epochs, the amount of time steps to predict is increased by one. In order to prevent the model from having problems with this sudden change of the task, the number of time steps is randomly increased, as well as decreased by one throughout the training process. In addition, the correction value $\Delta \boldsymbol{u}'_{t+1}$ predicted by the RNN cell is squared and added to the overall loss of the model in order to ensure that it is kept small. This can help to have the main flow of information through the easily trainable feed forward network, as it is intended for this model.

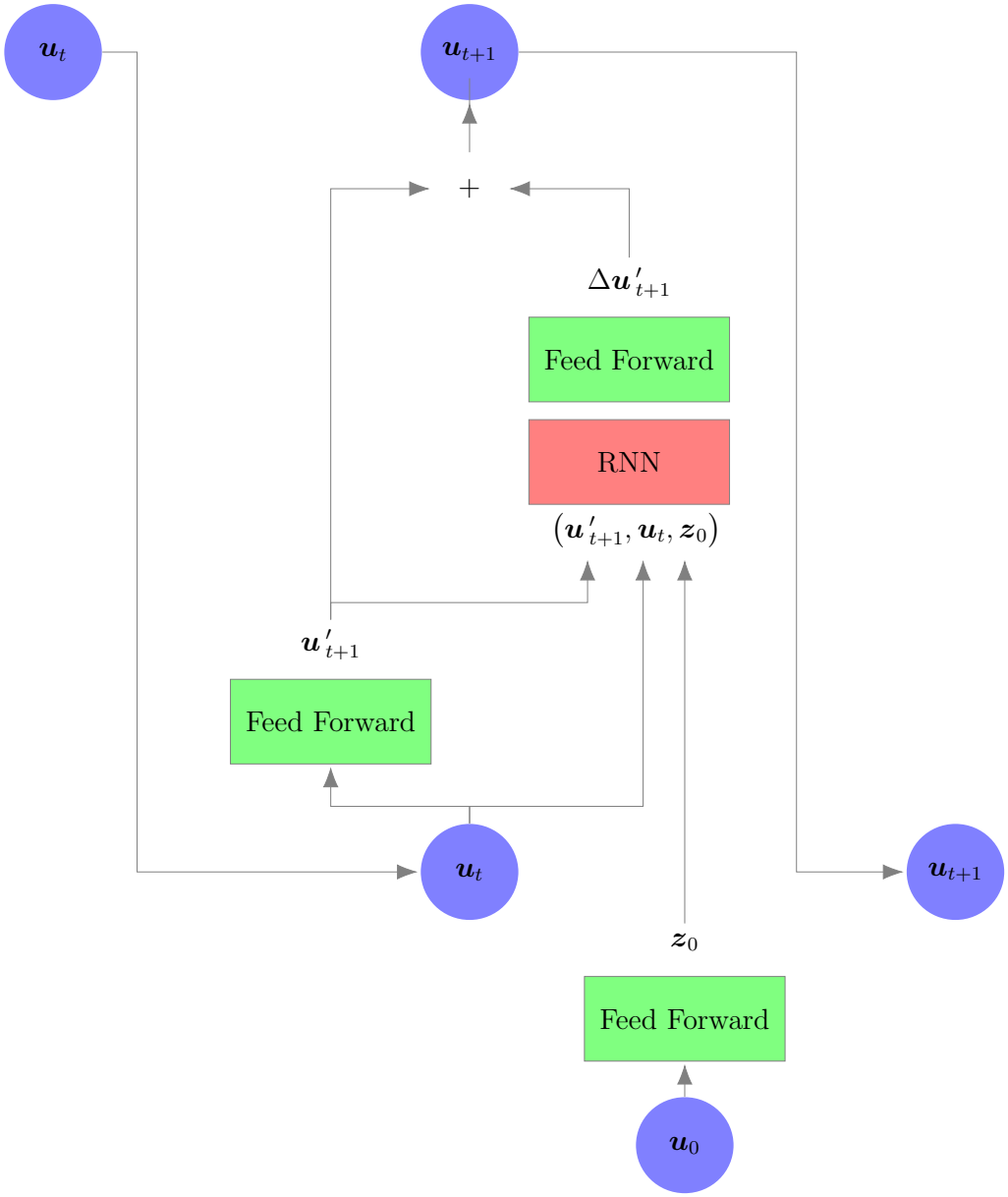


Figure 3.11: Sketch of the RNN model described in Sec. 3.2.4. The model consists out of three building blocks. A feed forward neural network encodes the trajectories' initial positions $\boldsymbol{u}_0$ into a higher-dimensional vector $\boldsymbol{z}_0$, while another one predicts the subsequent position $\boldsymbol{u}'_{t+1}$ of a trajectory, given its current position $\boldsymbol{u}_t$. These two resulting vectors together with the current position are then passed to a RNN cell, which, together with a feed forward network for the adjustment of the vector's dimensionality, predicts a possible deviation $\Delta\boldsymbol{u}'_{t+1}$ of the predicted subsequent position $\boldsymbol{u}'_{t+1}$ of the other feed forward neural network and the ground truth data. This deviation is then added to the previously predicted position, resulting in the model's output $\boldsymbol{u}_{t+1}$, which is then taken as the next input of the model.

4 Results

4.1 Parameterization of the NHIM and its dynamics

Before presenting the findings of parametrizing the NHIM via various AE models, it is important to clarify how the training data for the respective networks is generated. As mentioned in Section 2.1.2, single positions (x, v_x) of the NHIM can be estimated via the BCM for a fixed set of orthogonal mode coordinates (y, v_y). Since the goal is to capture the overall dynamics on the NHIM and not only that in its very center closest to the saddle point, the NHIM's positions are calculated in a broader range for orthogonal mode coordinates (y, v_y) that are sampled randomly and uniformly in the ranges $-3 < y < 3$ and $-7 < v_y < 7$. After calculating the corresponding positions (x, v_x) via the BCM, this leads to ranges of $-1 < x < 1$ and $-2.5 < v_x < 2.5$. While doing so, the parameters of the BCM are chosen to ensure an accuracy of approximately 10^{-12} for the points on the NHIM. For a system with the standard parameters listed in Section 2.2, the dataset of the NHIM for one oscillation period $t = 2$ consists of 200 equidistant time stamps $\mathrm{d}t = 0.01$ with 1000 NHIM points each, hence a total of 200.000 data points.

Trajectories on the NHIM are generated by propagating them as described in Section 2.2 with a time step of $\mathrm{d}t = 0.01$, while repeatedly stabilizing them on the NHIM. Their initial positions (y, v_y) are sampled uniformly in the same regions of the NHIM as listed above and their corresponding positions (x, v_x) are calculated via the BCM. The resulting trajectories lay on the NHIM with a mean accuracy in the order of 10^{-6}. It should be noted that trajectories are not part of the training data, rather they are used for the analysis of the resulting LS of a fully trained AE by revealing how the model displays these substructures of the NHIM.

4.1.1 Autoencoder architectures

When using neural networks to solve a problem it is crucial to optimize the network's hyperparameters for the specific task. Tunable parameters are for example the depth of a model or number of hidden layers, the number of nodes per layer, the activation functions, the learning rate and the batch size used while training. Usually, the optimal settings of a network have to be specifically determined for each different task. There

are no clear guidelines on how to optimize all parameters at once though it is known what the effects of each parameter can possibly be on the network's performance. For example adding layers and nodes to a network increases its number of tunable parameters and thus can make it easier to find descriptions of the problem it is trying to solve. However, increasing the number of tunable parameters can also cause the model to overfit the training data. This is reflected in a higher error for the validation data than the training data. Increasing the learning rate can prevent the model from getting stuck in a local minimum but can also result in unstable learning dynamics and potentially even hinder convergence. Throughout this work the autoencoders are trained with the Adam optimizer (see Section 2.3.1) and with an initial learning rate of $lr = 3 \times 10^{-4}$ which is adapted by the Adam optimizer during training.

In this section it is shown how different autoencoder architectures, specifically the dimension of the latent space, number of hidden layers, the number of nodes per layer as well as the chosen activation function influence the autoencoder's performance on finding a parameterization of the oscillating NHIM. It should be noted that all results are very specific to the task that the autoencoders perform and it is not possible to come up with clear predictions for other applications based on the outcomes of the dimension reduction of our particular system. However, there are still some insights and tendencies that can be derived for systems similar to ours.

Dimensionality of the latent space

Since the oscillating NHIM is a three-dimensional object in a five-dimensional space, it should be possible to find a three-dimensional parameterization of it. Since neural networks are universal function approximators, for an autoencoder of the type described in Section 3.1.1 it should be possible to find a suitable three-dimensional representation of the NHIM that contains all information necessary to reconstruct it from only this reduced space. This section deals with the verification of this hypothesis and analysis of an autoencoder's performance when trying to reduce the dimension of the NHIM to dimensions smaller and larger than three.

In order to do so, six autoencoders with latent space dimensions one to six were trained on the same dataset of one oscillation period of a NHIM with an angular frequency of $\omega_x = \pi$. All autoencoders have five hidden layers in both, the encoder and decoder part and a node sequence of $[5 \to 50 \to 50 \to 100 \to 100 \to 50 \to$ LS-dim. $\to 50 \to 100 \to 100 \to 50 \to 50 \to 5]$. Figure 4.1 shows the resulting training behavior. For a latent space dimension smaller than three, the performance is significantly worse than for dimensions equal or larger than three. The performance noticeably improves when increasing the latent space dimension from one to three, however, increasing it further does not make a difference. Even increasing it to dimensions larger than the input

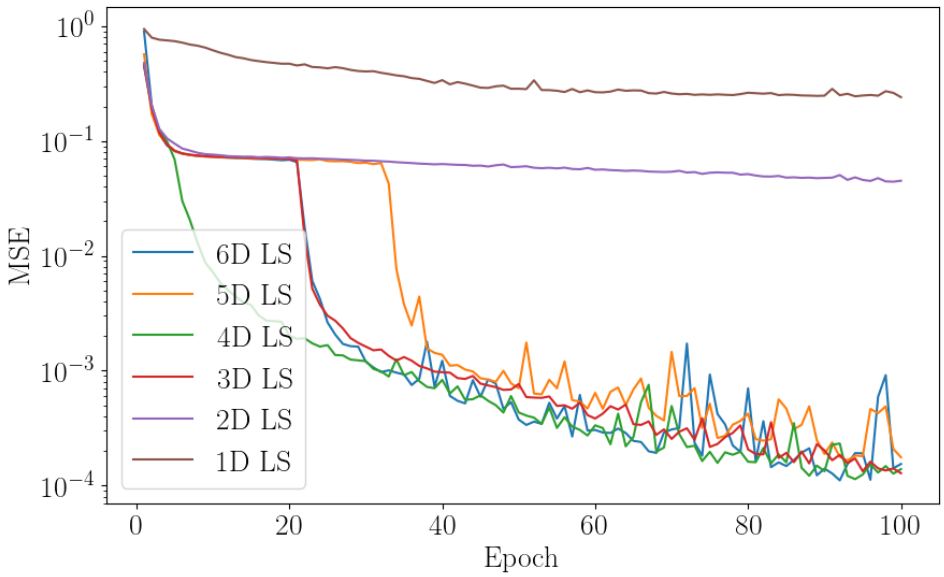


Figure 4.1: Convergence of the mean squared error (MSE) of autoencoders during training with a varying number latent space nodes. All models have a configuration of [5 → 50 → 50 → 100 → 100 → 50 → LS-dim. → 50 → 100 → 100 → 50 → 50 → 5] nodes per layer and are trained on one oscillation period of NHIM data. Autoencoders with a LS-dimension smaller than three converge to relatively high MSE values, while those with a LS-dimension equal or bigger than three all converge to rather small and approximately equal values. As expected, the autoencoders can only generate a valid representation of the NHIM in the latent space if the LS dimension is equal or larger than the NHIM's lowest dimensional representation in the original space. With an AE one usually strives to parameterize the input in the lowest dimensional representation possible.

dimension does not seem to make the task of representing the NHIM in this new space significantly easier for the autoencoder.

Another striking characteristic of the loss convergence seen in Figure 4.1 is that the majority of the autoencoders' loss curves follow that of the two-dimensional latent space autoencoder for a certain amount of epochs before abruptly decaying to much smaller values. Since the two-dimensional latent space autoencoder lacks one dimension needed for a good performance, it simply neglects one of the input dimensions and only returns an average of this particular variable for most of its training process. Very often, the neglected variable is the time information of the system. A two-dimensional latent space autoencoder will never be able to correctly represent all input variables at once, while those of higher dimension are in principle capable of doing so. Therefore, even if these autoencoders have trouble to incorporate the time information right from the

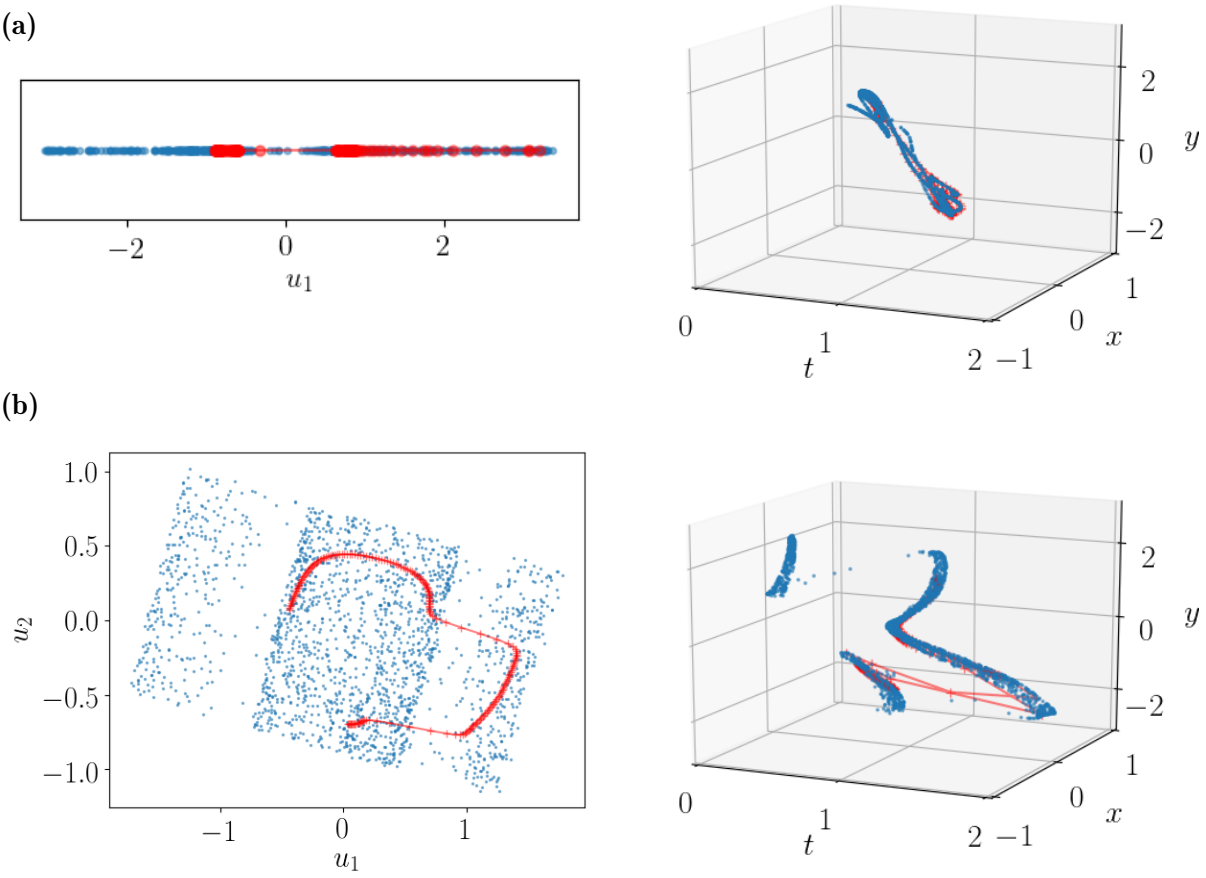


Figure 4.2: (a) Left: A two-dimensional LS of an AE, of the same kind as described in Figure 4.1, trained on the three-dimensional moving NHIM in the original five-dimensional space. The representation of the NHIM in this reduced space is marked by blue points and seems to be split into three different areas. The red line, representing a trajectory on the NHIM in the LS, exhibits discontinuities whenever it jumps from one of these three areas to another. Right: Three dimensions (t, x, y) of the original five-dimensional input space, reconstructed by an AE. The AE fails to accurately reconstruct the positions of the NHIM. (b) Left: A one-dimensional LS of an AE, of the same kind as described in Figure 4.1, trained on the three-dimensional NHIM in the original five-dimensional space. Blue points mark points of the NHIM. The red points show the representation of a trajectory on the NHIM in the LS. Right: Three dimensions (t, x, y) of the original five-dimensional input space, reconstructed by an AE. The AE fails to accurately reconstruct positions of the NHIM.

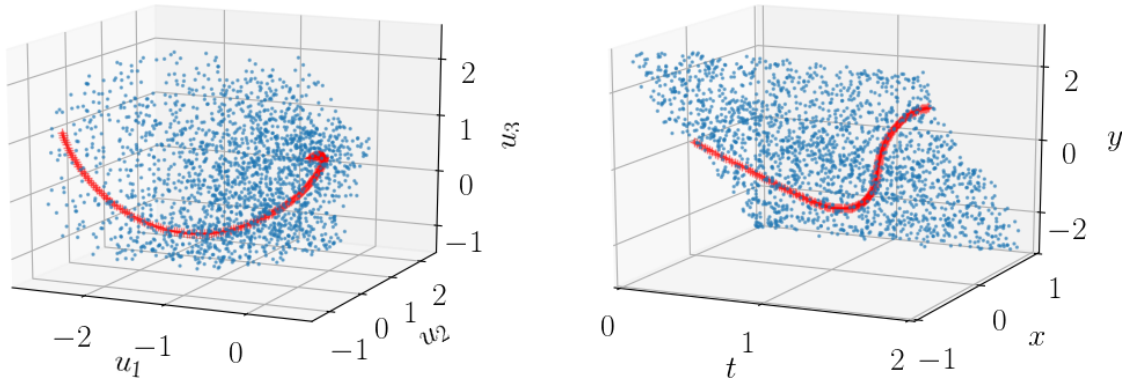


Figure 4.3: Left: A three-dimensional LS of an AE, of the same kind as described in Figure 4.1, trained on the three-dimensional moving NHIM in the original five-dimensional space. The representation of the NHIM in this reduced space is marked by blue points. The red line, representing a trajectory on the NHIM in the LS, is smooth and continuous. Right: Three dimensions (t, x, y) of the original five-dimensional input space, reconstructed by an AE. The NHIM is correctly represented as a smooth nonlinear surface and the NHIM as well as the trajectory match the original data with a high accuracy.

beginning, they usually do so at a later time during the training which then results in a rapid decay of their training loss. Factors that increase the likelihood of an autoencoder to incorporate the time information right from the beginning are an increased number of nodes per layer, as well as a dataset of a NHIM for more than just one oscillation period.

Figure 4.2 shows the full latent space, as well as three of the five output dimensions for fully trained autoencoders with a one- and a two-dimensional latent space. As expected, both AE are not able to reconstruct the original data and show discontinuities in the LS, as well as in the output. By contrast, the AE with a three-dimensional LS is able to reconstruct the input with high accuracy and its LS resembles a contiguous space. As can be seen in Figure 4.3, trajectories in the LS are a clearly defined subspaces and reveal a general structure in the cloud-like object.

As expected, the autoencoder with the three-dimensional latent space performs very well, while those with a one and two-dimensional latent space are not able to reconstruct the original data and show discontinuities in the latent space as well as in the output.

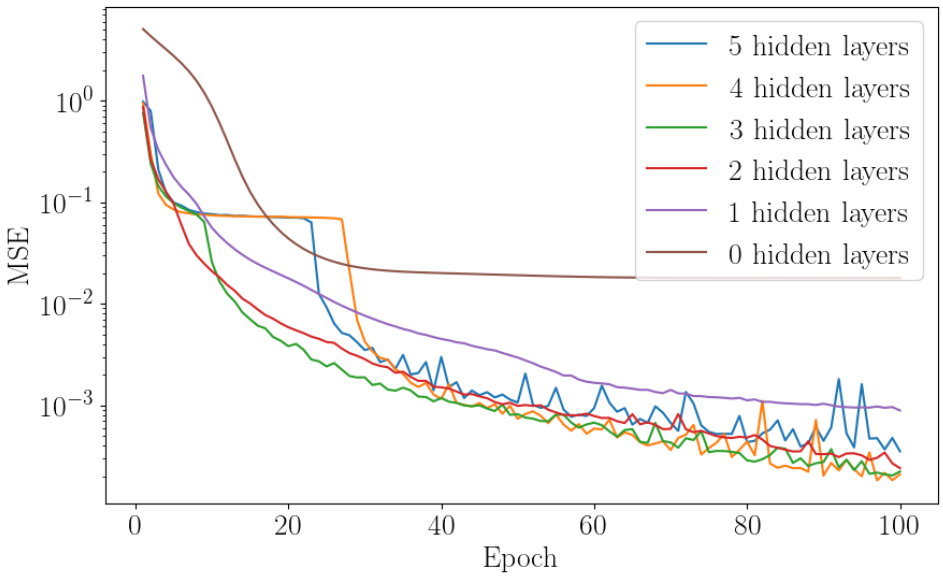


Figure 4.4: Convergence of the mean squared error of the validation dataset during the training of autoencoders with varying numbers of hidden layers. All autoencoders have 50 nodes per hidden layer and are trained on one oscillation period of NHIM data. Up to a number of hidden layers of two, the AE's performance increases significantly. For numbers higher than two, there is no distinct difference of the values at which the MSE of the AEs converge.

Number of hidden layers

Here, we deal with analyzing the influence that the depth of an autoencoder, meaning its number of hidden layers, can have on performance. Six autoencoders each with 50 nodes per hidden layer but varying number of hidden layers were trained on one oscillation period of NHIM data for a potential with an angular frequency of $\omega_x = \pi$. Figure 4.4 shows the loss convergence of all six autoencoders with zero to five hidden layers in the encoder and decoder part, respectively. While the performance of an architecture of zero hidden layers is significantly worse than such with one or more hidden layers, there is also an increase in performance when going from one to two hidden layers. However, increasing the number of hidden layers further does not lead to a distinct improvement. The step in the loss curves of the architectures with four and five hidden layers is due to the autoencoders going from the optimization of only two of the three latent space coordinates to including all input variables and optimizing for all three latent space dimensions just like in the comparison of the different LS dimensions.

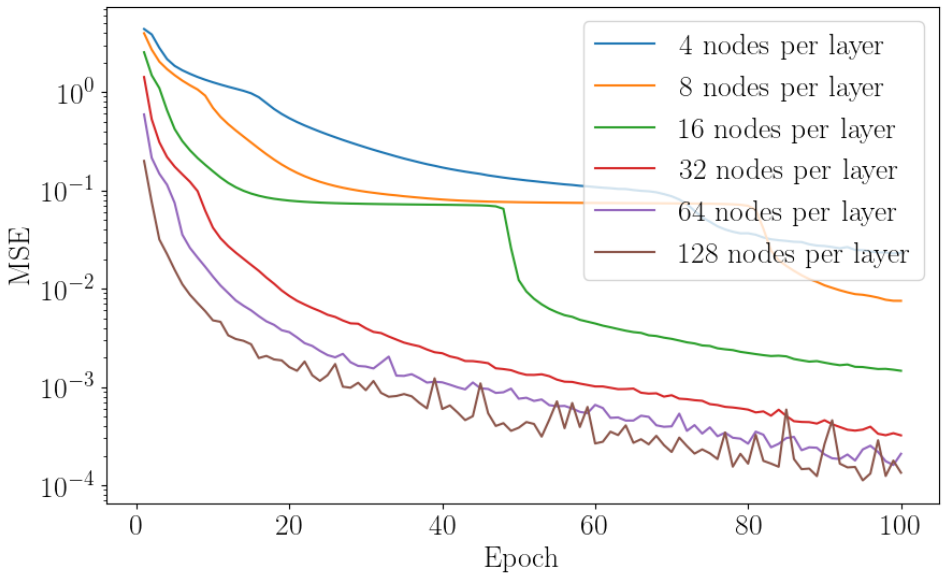


Figure 4.5: Convergence of the mean squared error of the validation dataset during the training of autoencoders with varying numbers nodes per layer. All autoencoders have two hidden layers and are trained on one oscillation period of NHIM data. The AEs performances increase with the number of nodes per layer. However, at approximately 32 nodes per layer, increasing the number of nodes further does not decrease the MSE by much.

Nodes per layer

Here, it is analyzed how a varying number of nodes in an autoencoder can affect its performance. Six autoencoders each with two hidden layers, in the encoder and decoder part respectively, are trained on one oscillation period of NHIM data for a potential with an angular frequency of $\omega_x = \pi$. Their number of nodes per layer vary from four to 128. Figure 4.5 shows the respective loss curves of all six autoencoders. There is a clear trend visible regarding the loss right at the start of the training after just one epoch as well as at the end after 100 epochs: autoencoders with more nodes per layer have a smaller training loss. However, the improvement in performance becomes less significant with increasing number of nodes. Another tendency, that is visible in this plot and is also mentioned for the depth comparison, is the increased likelihood of an autoencoder to optimize for all three latent space coordinates at once at an earlier time in the training process for a large number of nodes per layer compared to only few nodes per layer. The models with 4, 8, and 16 nodes per layer exhibit the problem of neglecting one input variable for the first part of the training process and therefore only optimizing for two of the three latent space coordinates. For the models with 32 nodes and higher, this problem does no longer occur.

Activation function

Another factor that can heavily influence a model's performance is the choice of the activation function. All networks trained throughout this work are designed to have an activation function for the nodes of each layer except for the latent space and output nodes. As described in Section 2.3.1, a lot of models, especially those for feature extraction from images, use the popular ReLU-function (see Eq. 2.13) as an activation function. However, for the purpose of parametrizing the nonlinear and smooth NHIM it is suitable to use a nonlinear and differentiable function like the tanh. In this section, two autoencoders, one with the ReLU-function and one with the tanh as the activation function are compared in order to analyze the resulting differences and also to justify using the tanh for all models trained throughout this work. Besides their different activation functions, both autoencoders have five hidden layers in encoder and decoder part and a configuration of $[5 \to 50 \to 50 \to 100 \to 100 \to 50 \to \text{LS-dim.} \to 50 \to 100 \to 100 \to 50 \to 50 \to 5]$ nodes per layer. They were trained on one oscillation period of NHIM data in a system with an angular frequency of $\omega_x = \pi$. After training them to convergence, both autoencoders have a very similar mean squared error of 1.9×10^{-4} for the ReLU-autoencoder and 1.3×10^{-4} for the tanh-autoencoder. However, the differences between both models becomes evident when looking at the resulting latent spaces and outputs, depicted in Figure 4.6. The ReLU-autoencoder comes up with a rather complicated looking and curved latent space, while that of the tanh-autoencoder is a compact sphere-like object in which the course of the exemplary trajectory is very smooth. This behavior is also evident in the output of both models, where the trajectory illustrated for the tanh-autoencoder is significantly more smooth than that resulting from the autoencoder using the ReLU-function. Since one of the main goals of this thesis is to study trajectories in the reduced space not only by looking at them but also by applying other machine learning methods to capture their dynamics and predict new trajectories, having a rather simple latent space with smooth trajectories is of great advantage. This is why autoencoders with a tanh as the activation function work best for our purposes, even though other activation functions might also lead to a good performance in regards to the training loss.

4.1.2 The autoencoder's stabilizing properties

Generally, neural networks perform much better on interpolation than extrapolation tasks. Likewise, a fully trained autoencoder typically does not extrapolate and find a plausible description of data outside of the known training data. It will rather map the input data to the reduced version of the space it was trained on, even when the given input data is not a part of that space. When wanting to use autoencoders to reduce spaces outside of the known ranges, this characteristic is a clear disadvantage. However, it can be of great benefit when wanting to use the models for the denoising of data. For

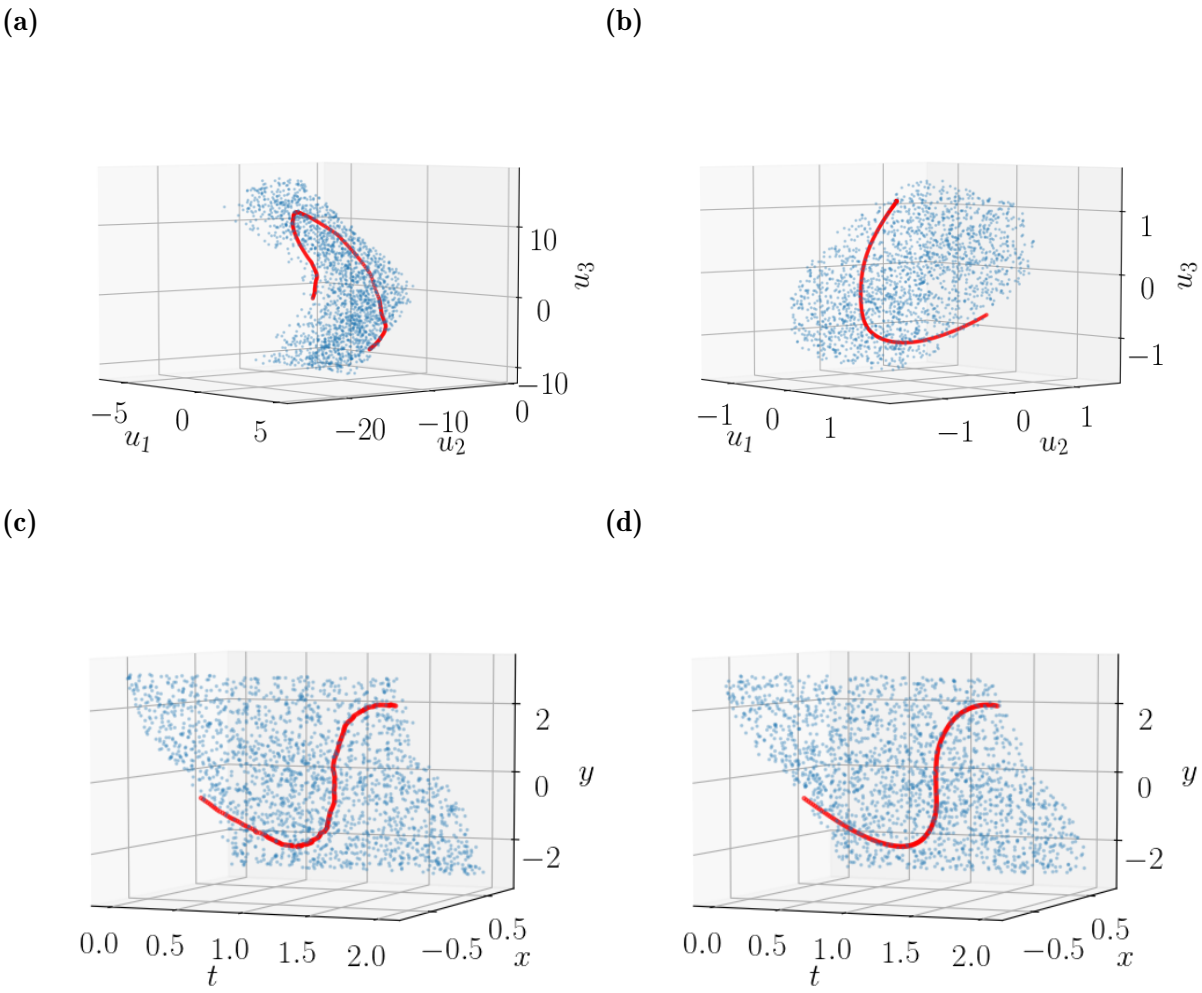


Figure 4.6: Comparison of latent spaces (a), (b) and outputs (c), (d) of AEs with different activation functions. Plots (a) and (c) belong to an AE with the ReLU-function, defined in Eq. 2.13, as the activation function. Plots (b) and (d) are the LS and output of an AE with the tanh as the activation function. Transformed positions on the NHIM are given by the blue points and the red line marks a trajectory on the NHIM. The LS in (a) is non-trivially curved and the trajectory in the output in (c) is not smooth. By contrast, the AE with a tanh as the activation function generates a rather simple LS (b) and is able to reconstruct a smooth trajectory in (d).

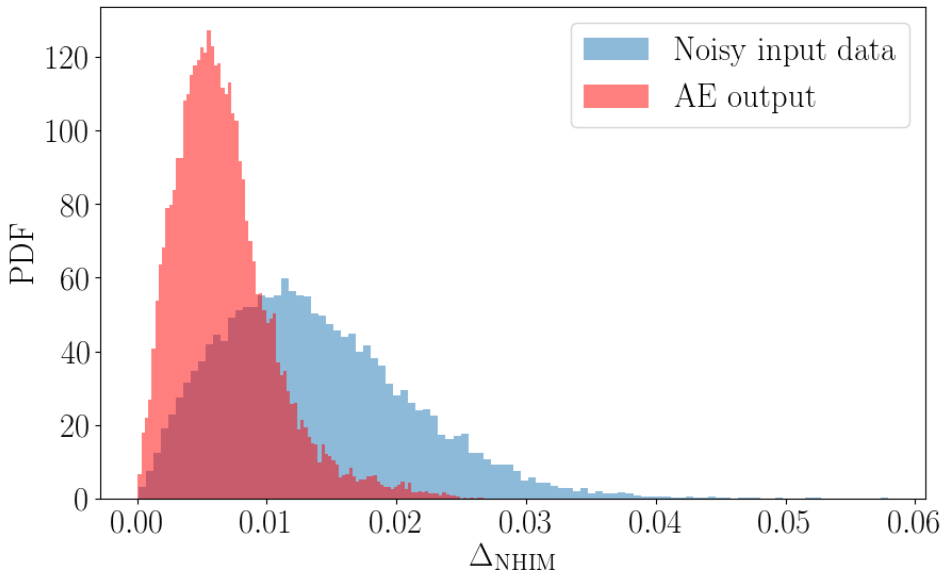


Figure 4.7: Probability density function (PDF) of the deviations from the NHIM of the input and output data of an AE model, previously trained on one oscillation period of the NHIM. The data processed by the fully trained AE is data of the same NHIM it was trained on, but with added Gaussian noise with a standard deviation of 0.01. The distribution of the AE's output data is sharper and has a significantly lower mean value compared to that of its input data. Consequently, the AE maps the noisy input data back to the actual positions of the NHIM it was trained on and can thus act as a stabilizer of points on the NHIM.

example, when an autoencoder is fully trained on points of the NHIM, it will project data that deviates slightly from the NHIM back onto it.

This behavior can be observed in Figure 4.7. Here, an autoencoder with 5 hidden layers and a configuration of [5 $\rightarrow$ 50 $\rightarrow$ 50 $\rightarrow$ 100 $\rightarrow$ 100 $\rightarrow$ 50 $\rightarrow$ LS-dim. $\rightarrow$ 50 $\rightarrow$ 100 $\rightarrow$ 100 $\rightarrow$ 50 $\rightarrow$ 50 $\rightarrow$ 5] nodes per layer was trained to convergence on data of one oscillation period of the NHIM in a system with an angular frequency of $\omega_x = \pi$. Then, Gaussian noise was added to the data by randomly sampling from a normal distribution with standard deviation 0.01. Figure 4.7 shows a histogram of the deviation from the NHIM calculated as described in Section 2.2 for both the noisy input data and the resulting output of the model after this data is processed. The distribution of the output data is sharper than that of the input data and its center is at a significantly lower deviation from the NHIM. This result confirms that the autoencoder stabilizes the noisy data by projecting it back to the known ranges of the NHIM.

Section 2.2 described how trajectories calculated on the unstable NHIM have to be

projected back onto it from time to time due to numerical inaccuracies. For a two-dimensional system like this one, this projection is rather simple. However, for higher-dimensional systems, the task of stabilizing trajectories on the NHIM might become much more complex. This is where a fully trained autoencoder describing the respective NHIM with high accuracy could be used for the stabilization instead.

4.1.3 Autoencoder – oscillation phase input

This section presents the performance of the autoencoder method presented in Section 3.1.2, in which the model generates an essentially two-dimensional latent space given the oscillation phase of the NHIM instead of the absolute time of the system as an input variable.

When reducing the dimensionality of a periodic system for more than one oscillation period, one has to keep in mind that it takes at least two variables to uniquely represent the periodic phase over time. Therefore, the latent space dimension of an autoencoder trained on such input data, has to be at least four. As described in Section 3.1.2, in a model with four latent space variables, it is suitable to dedicate two of them solely towards the representation of the phase input which are not learned by the model itself. The other two variables are learned by the model to represent the NHIM's position in the phase space.

Figure 4.8 shows a comparison of two autoencoders that are similar in configuration and training except for the latent space dimension which is three and four. The AEs have a configuration of $[6 \to 50 \to 100 \to 50 \to \text{LS-dim.} \to 50 \to 100 \to 50 \to 6]$ nodes per layer and are trained on one oscillation period of the NHIM of a system with angular frequency $\omega_x = \pi$. The lack of a forth latent space dimension for the first autoencoder results in a number of discontinuities that are very visible in the trajectory displayed here for three oscillation periods. Naturally, this affects the model's output which shows some inaccuracies as well. By contrast, the other autoencoder generates a two-dimensional latent space which, in combination with the already known phase information, is a full description of the periodic NHIM. The resulting space is very symmetrical and the trajectories seemingly oscillate around some middle point without exhibiting discontinuities. The trajectories' quasi periodicity becomes very visible when looking at this reduced space, making it well suited for further analysis of the dynamics on the NHIM.

Poincaré maps in the latent space

Having a two-dimensional latent space in which trajectories can be represented for all time enables a straight forward analysis of their dynamics in this space. In a dynamical

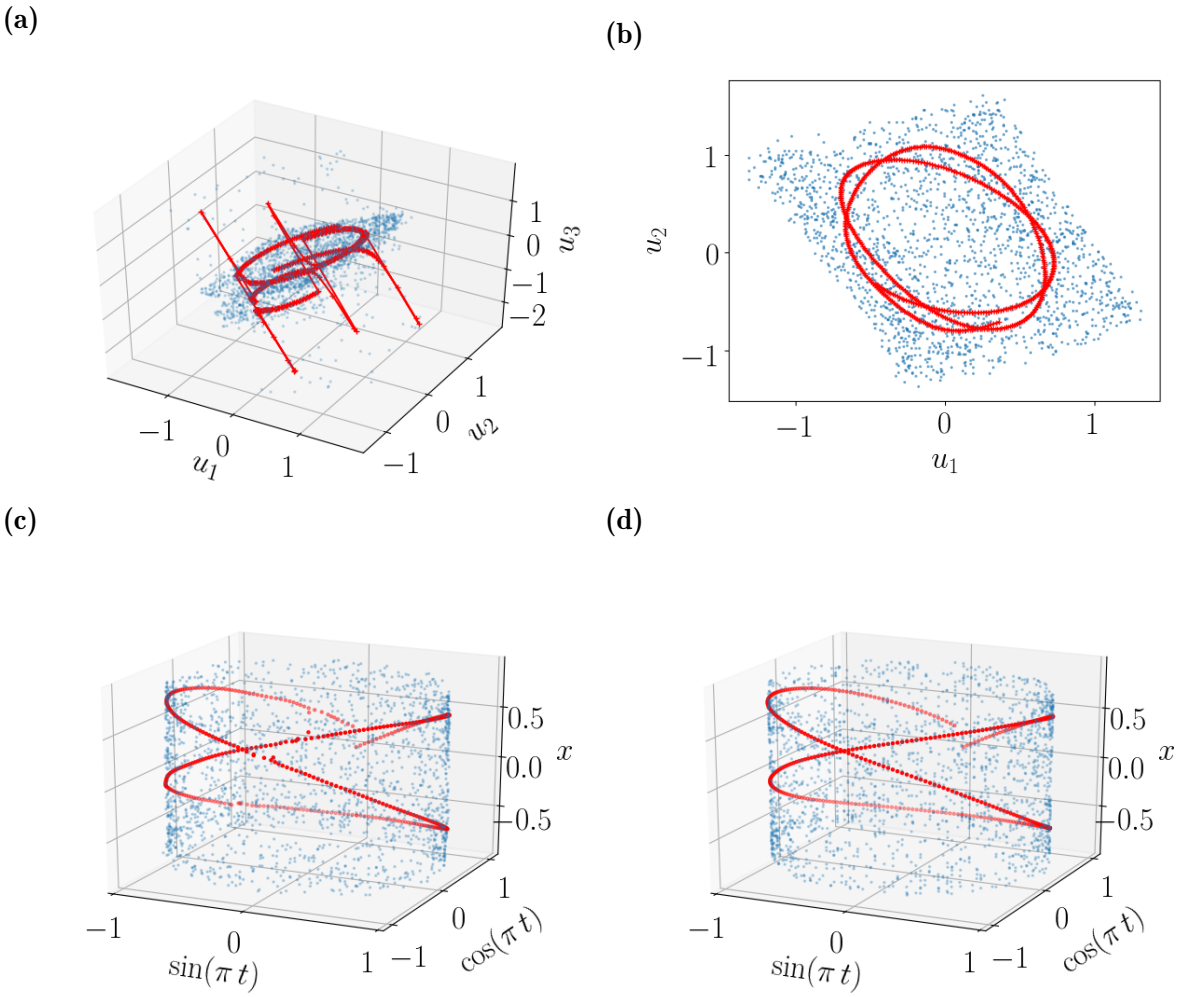


Figure 4.8: Comparison of latent spaces (a), (b) and outputs (c), (d) of AEs, of the type presented in Sec. 3.1.2, with a LS dimension of three vs. four. Plots (a) and (c) belong to an AE with a LS dimension of three, while plots (b) and (d) belong to an AE with LS dimension four. Translated positions on the NHIM are given by the blue points and the red line marks a trajectory on the NHIM. An AE with only three nodes in the LS is not able to capture the NHIM's structure and dynamics fully, as can be seen by the discontinuities in (a) and (b). By contrast, an AE with LS dimension four successfully reduces the NHIM's phase space position to two dimensions, as can be seen in (b). From these two dimensions generated by the AE itself and the added two dimensions describing the phase information, the AE is able to fully reconstruct its input, as can be seen in (d).

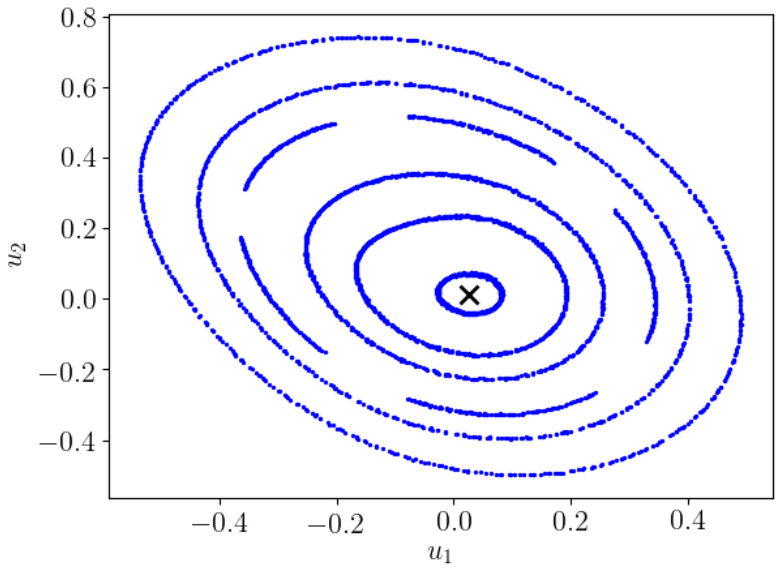
system with periodic and quasi periodic orbits, calculating a Poincaré map, just like the one presented in Section 2.1.3, can provide a simplified view on the otherwise complex dynamics and lead to good insights about the system's stability. When determining a Poincaré map of a periodic system there are usually many different possibilities for choosing the lower-dimensional subspace that the periodic orbits intersect with. However, for a latent space like the one seen in Figure 4.8 (b), where there is only a single plane describing the trajectories' positions in the phase space, there is only one single option to choose from. The only variable in calculating the Poincaré map is the specific phase for which it is determined and this property hardly has any influence on the overall result.

In order to generate a Poincaré map for a system with an angular frequency of $\omega_x = \pi$, the trajectories' positions u_1 and u_2 in the latent space of the autoencoder with the oscillation phase input were plotted at every start of the NHIM's oscillation period, meaning $u_3 = \sin(\omega_x\, t) = 0$ and $u_4 = \cos(\omega_x\, t) = 1$. The resulting Poincaré map is shown in Figure 4.9 (a). The presence of one single fixed point circled by all other intersection points indicates that there is only one single exactly periodic trajectory on the particular NHIM of this system while all others are quasi periodic.

For a system with an angular frequency of $\omega_x = 3/4\,\pi$, the dynamics on the NHIM become more interesting. A parameterization of this system's NHIM was generated by training an autoencoder similar to the one above but with training data of the NHIM of this particular system. The corresponding Poincaré map of trajectories in this latent space was generated in the same way as above. The resulting image can be seen in Figure 4.9 (b). Here, there are two fixed points, hence two exactly periodic trajectories on the NHIM. This also drastically affects the course of the quasi periodic trajectories making each of them bend around one of the two fixed points. This radical switch in the dynamics resulting from a change in the system's angular frequency is a *bifurcation*.

As mentioned above, there is only one way to choose the intersection for the PSOS in an essentially two-dimensional space. Therefore, Figures 4.9 (a) and (b) are both full descriptions of the dynamics on the respective NHIM encoded in only a single image.

(a)



(b)

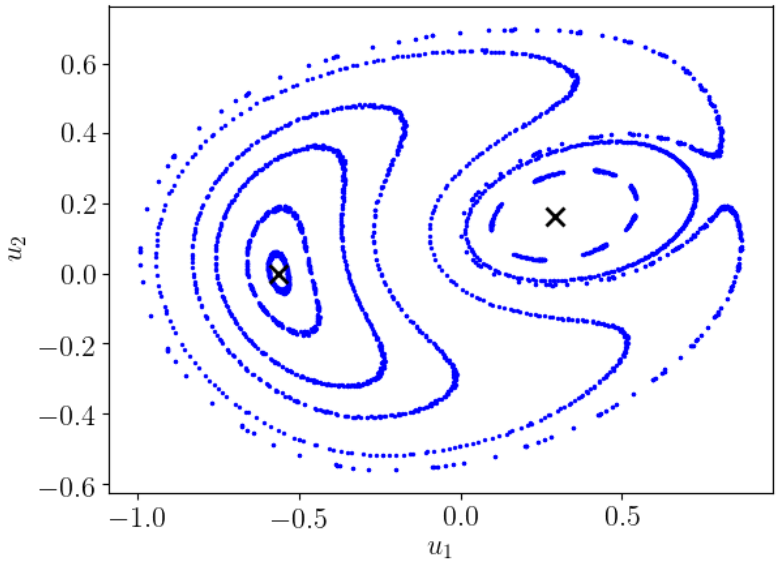


Figure 4.9: Poincaré map of the latent space of an AE trained on data of the NHIM. The quasi-periodic trajectories on the NHIM are given by the blue points and an elliptic fixed point of an exactly periodic trajectory is marked by the black cross. (a) A system with an angular frequency of $\omega = \pi$. The quasi-periodic trajectories oscillate around one elliptic fixed point in the center. (b) A system with an angular frequency of $\omega_x = (3/4)\pi$. This system exhibits a bifurcation with two elliptic fixed points.

4.2 Predicting trajectories

This section presents the outcomes of the four methods, presented in Section 3.2, for the prediction of trajectories in the latent space of an autoencoder, trained on the moving NHIM. The first two methods revolve around finding a description of the differential equation system for the dynamics on the NHIM; the third method is used to directly predict trajectories by interpolation via a neural network and the last method iteratively predicts the trajectories' subsequent positions with the help of recurrent neural networks.

All analysis of the results presented in Sec. 4.2.1-4.2.4 is limited to the reduced space given by an autoencoder. The validation of the predicted trajectories in the original five-dimensional system is presented in the upcoming Section 4.2.5.

Sections 4.2.1-4.2 present methods that are tested in the latent space of an autoencoder of the type described in Section 3.1.1 with the absolute time as an input variable. Section 4.2.4 presents a method that uses recurrent neural networks and is designed for the latent space of the autoencoder described in Section 3.1.2 which processes the oscillation phase instead of the absolute time of the system. All methods revolve around predicting trajectories for the same system with an angular frequency of $\omega_x = \pi$.

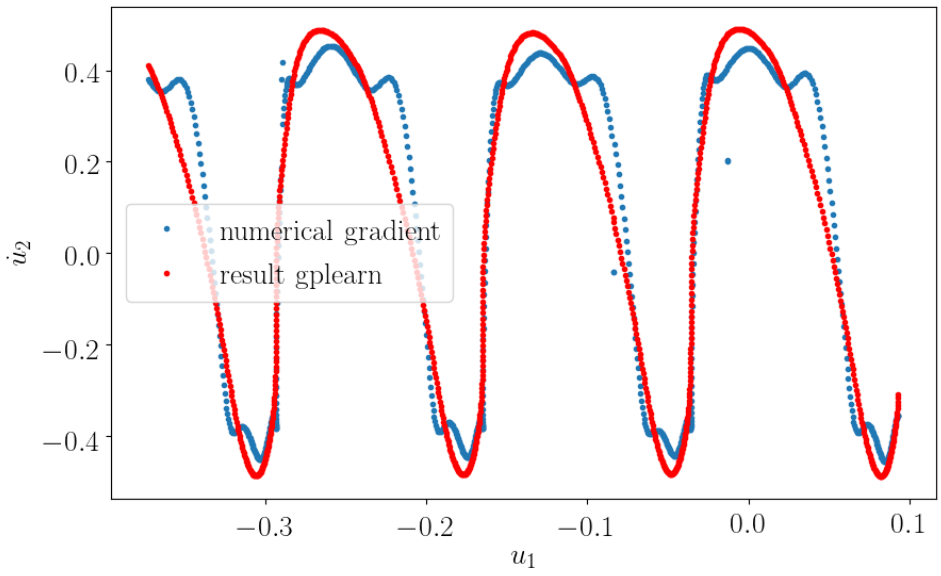
The autoencoder whose latent space is used for the first three methods has a rather small configuration of only $[5 \to 5 \to 4 \to 3 \to 4 \to 5 \to 5]$ nodes per layer. Due to its small architecture, the training loss converges at around MSE $= 3 \times 10^{-3}$. This is one to two orders of magnitude larger than what can be achieved with a network of 50 to 100 nodes per layer. The reason why this particular autoencoder model is still chosen for all three methods is its strikingly symmetrical latent space, making it ideally suited for the artificial extension in time, as described in Section 3.1.1. It was originally trained on data of two oscillation periods of the NHIM and its latent space was then artificially extended in order to cover a time scale of 40 oscillation periods.

For the forth method, a much larger autoencoder is trained with a configuration of $[6 \to 50 \to 100 \to 50 \to 2\ (+2) \to 50 \to 100 \to 50 \to 6]$ nodes per layer. The final mean squared error loss at convergence is MSE $= 1.3 \times 10^{-5}$.

4.2.1 Symbolic regression with gplearn – results

The goal of this method is to find an analytical differential equation system of the dynamics on the NHIM, represented in the reduced three-dimensional latent space of the AE. This is done by using a genetic programming algorithm to do a symbolic regression analysis of a given dataset.

(a)



(b)

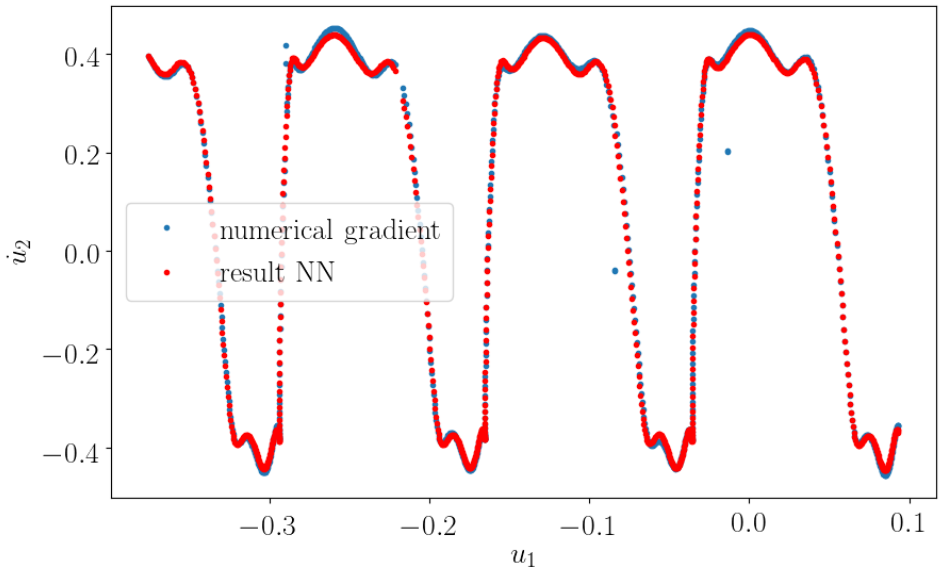


Figure 4.10: Comparison of the projections of $\dot{u}_2$ over u_1 of the numerical calculation of $\dot{u}_2$ and the predicted solution. (a) The predicted solution is estimated via the analytical Eq. (4.2) resulting from the *gplearn* algorithm. The course given by the analytical equation roughly follows that given by the numerical gradient, but fails to capture its finer details. (b) The predicted solution is estimated by a neural network of the type presented in Section 3.2.2. The course given by the model closely follows that given by the numerical gradient.

This dataset consists out of a total of 90 trajectories, each with a different initial position on the NHIM, which were previously transformed to the latent space via a fully trained AE. The trajectories capture a total length of 40 oscillation periods. In order to prepare the dataset, it is cleaned from statistical outliers after calculating the trajectories' numerical gradients in respect to the time.

The algorithm's tunable parameters, further explained in Section 3.2.1, are set to

$$\begin{aligned} \text{population\_size} &= 5000\,, \\ \text{p\_crossover} &= 0.6\,, \\ \text{p\_subtree\_mutation} &= 0.14\,, \\ \text{p\_hoist\_mutation} &= 0.01\,, \\ \text{p\_point\_mutation} &= 0.24\,, \\ \text{parsimony\_coefficient} &= 0.001\,. \end{aligned}$$

The list of permitted mathematical operators is defined to consist of "add", "sub", "mul", "div", "sin", "cos", "tan" and "tanh". With these settings, the algorithm is run for as many generations as the fitness of the most accurate equation of the respective generation continues to decrease. The resulting equations found by the gplearn algorithm have fitnesses $f(\dot{u}_1) = 0.0159$, $f(\dot{u}_2) = 0.0459$, and $f(\dot{u}_3) = 0.0178$ and are given by the following expressions:

$$\dot{u}_1 = 0.126 \cdot \tan(0.446) \cdot (u_3 - \tanh(\cos(\arctan(u_1))) + 2u_2)\,, \tag{4.1}$$

$$\dot{u}_2 = -0.661 \cdot \tanh(u_3) + 0.1u_1 - u_3\,, \tag{4.2}$$

$$\dot{u}_3 = -0.081 \cdot u_1 - 0.098 \cdot (u_2 - 2u_3) + \frac{u_2}{0.452} + u_2\,. \tag{4.3}$$

Since there are no restrictions regarding the order in which mathematical expressions are used, the stacking of trigonometric functions as in Equation (4.1) is not a rarity. Such expressions can be prevented by using other algorithms with more options for introducing restrictions like in the AIFeynman algorithm [57]. However, trying it on the data of trajectories in the latent space does not lead to better results than the gplearn algorithm.

Figure 4.10 (a) shows a comparison between the original data of the numerical gradients calculated via *numpy gradient* and the result of Equation (4.2) of $\dot{u}_2$ over u_1 for the first few oscillations of the external driving. Just like it can be seen in this figure, Equations (4.1)-(4.3) are accurate enough to capture the overall course of the data but often fail to capture finer details.

In order to use the obtained differential equation system to generate new trajectories, it can be solved numerically for any length of time, starting from any position in the latent

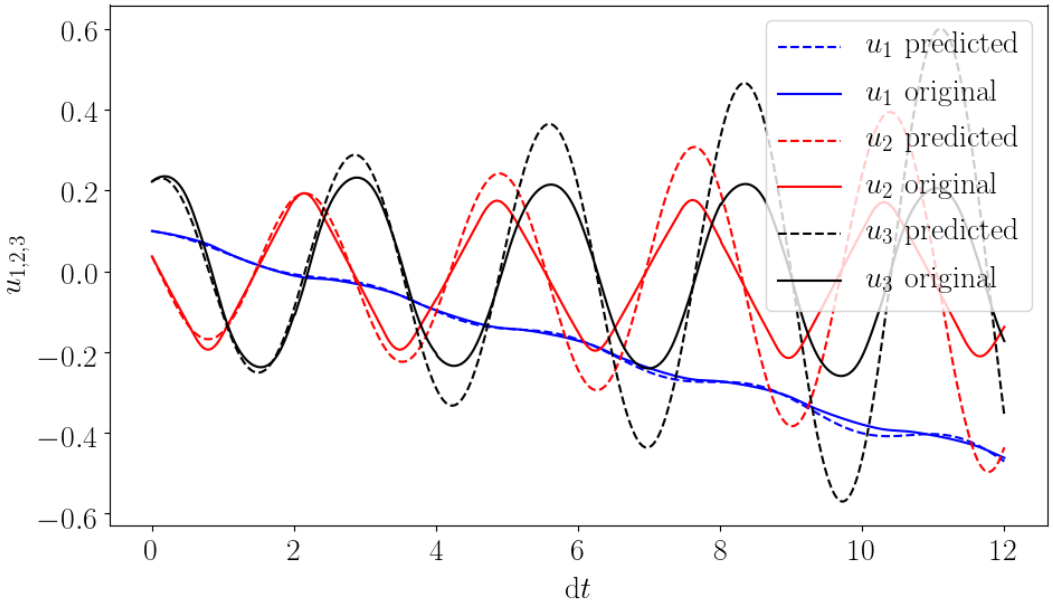


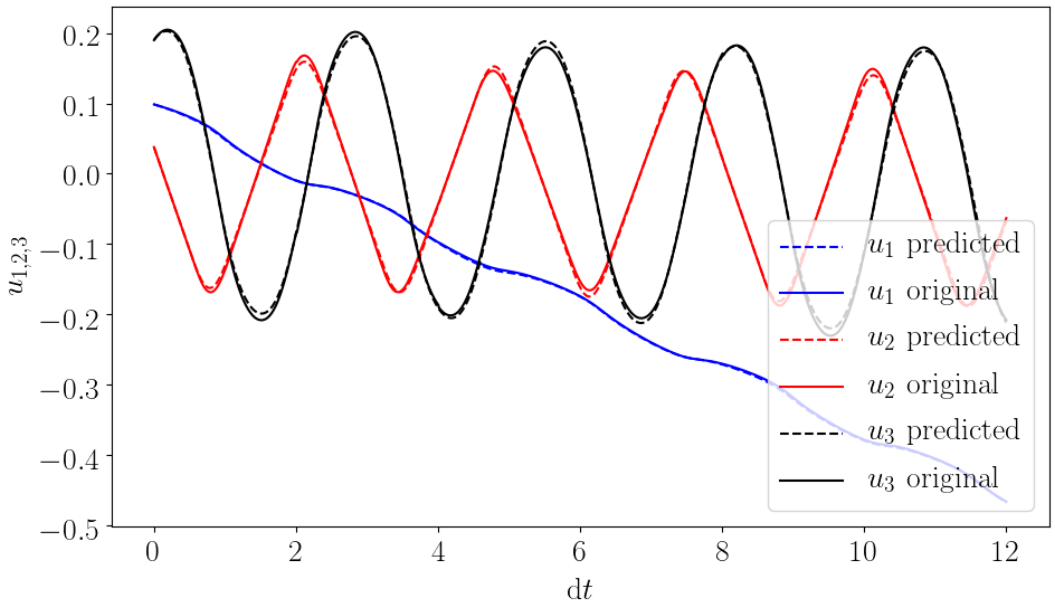
Figure 4.11: Comparison of LS coordinates (u_1, u_2, u_3) for six oscillation periods of the NHIM, obtained by numerically solving the differential equation system (4.1)-(4.3) given by the *gplearn* algorithm (dashed lines) and by transforming an original trajectory on the NHIM into the LS by the AE (solid lines). The numerical solution quickly deviates from the original trajectory. However, the estimate given by the numerical solution matches the original course sufficiently well for at least one period of the external driving.

space. Figure 4.11 shows a comparison of the latent space coordinates of an original trajectory and of one that was generated by solving Equations (4.1)-(4.3) numerically with the same initial position. What can be seen in the figure is an increasingly growing deviation of the generated trajectory from the original one over time. This behavior is expected, since the differential equation system does not describe the data perfectly. Consequently, the equations found with the gplearn algorithm are not suited to generate trajectories for long times, as they become physically implausible for the system. However, if the goal is to quickly generate many trajectories for only one oscillation period of the NHIM starting from various positions in the reduced space, hence the NHIM, using Equations (4.1)-(4.3) is a suitable possibility.

4.2.2 Neural network for equations of motion – results

Due to the insufficient accuracy of the analytical differential equation system, found by the symbolic regression analysis, for predicting longer time scales, the method presented

(a)



(b)

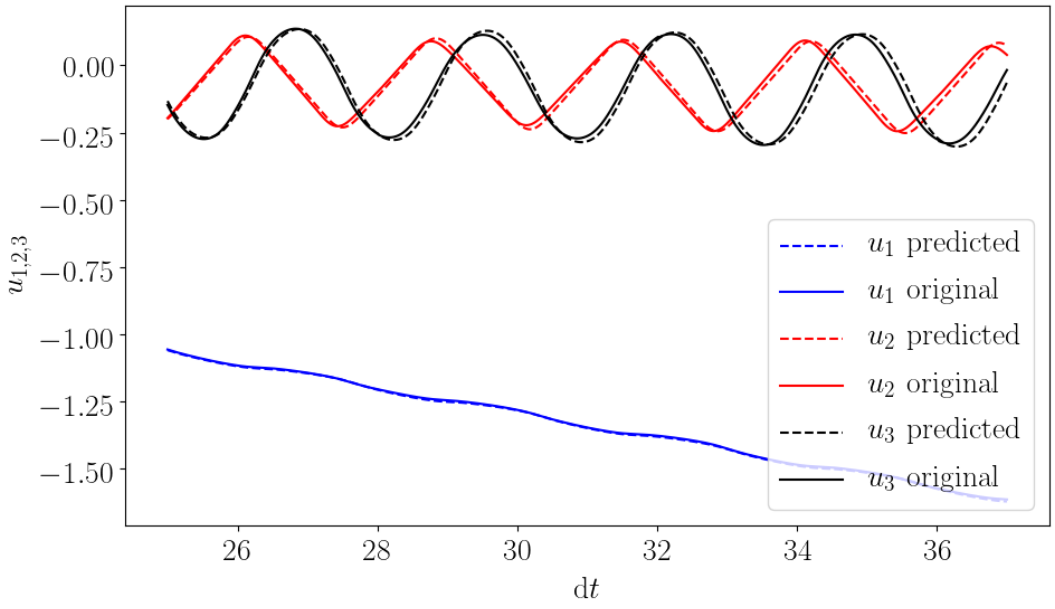


Figure 4.12: Comparison of LS coordinates (u_1, u_2, u_3) for six oscillation periods of the NHIM, obtained by numerically solving the description given by a neural network of the type described in Section 3.2.2 (dashed lines) and by transforming an original trajectory on the NHIM into the LS by the AE (solid lines). There is hardly any visible deviation between the courses of the original and the predicted trajectory for the first six periods of the external driving (a), while a small deviation starts to emerge at around 13 oscillation periods and starts to slowly grow from there (b).

here deals with finding a more accurate description via interpolation with neural networks. As described in Section 3.2.2, a feed forward neural network (NN) is trained on predicting the equations of motion (EOM), describing the dynamics on the NHIM, at their current positions in the reduced space. This method is referred to as the *EOM-NN method*. Similar to Sec.4.2.1, the gradients in respect to the time of 90 trajectories with a length of 40 oscillation periods are calculated numerically for the training dataset. The network chosen for the task of interpolating the data has a configuration of $[3 \rightarrow 50 \rightarrow 100 \rightarrow 100 \rightarrow 100 \rightarrow 50 \rightarrow 3]$ nodes per layer and was trained until convergence at a mean squared error of MSE $= 1.2 \times 10^{-4}$. There is no overfitting of the training data since validation and training loss converge in the same way.

Figure 4.10 (b) shows a projection of $\dot{u}_2$ over u_1 for both the original data calculated numerically and the output of the neural network model for the first few oscillations periods of a trajectory of the validation set. Unlike in Figure 4.10 (a), the network successfully captures the full course of the data.

The description of the trajectories' differential equation system given by the fully trained model can then be solved numerically, similar to the analytical equations in Sec. 3.2.1. Figure 4.12 (a) shows a comparison of the latent space coordinates of an original trajectory of the validation set and a trajectory generated by solving the network's description numerically with the same initial position. For the first six oscillation periods of the NHIM, there is hardly any deviation of the generated trajectory from the original one, which is a vast improvement to the results presented in Figure 4.11. This is simply due to the much more accurate description of the system's dynamics given by the network compared to the analytical equations found with gplearn. The solution only starts to visibly deviate from the original data at around 15 oscillation periods, as it can be seen in Figure 4.12 (b). Just like with the interpolated equations of motion method, although not as strongly, this deviation grows over time.

Nevertheless, by using this method, it is possible to quickly generate new trajectories starting from anywhere in the sampled reduced space of the NHIM for approximately 15 oscillation periods.

4.2.3 Neural network for prediction – results

Unlike the approach using gplearn and the interpolated equations of motion method, this method has no predisposition to growing deviations of the generated trajectories from the original ones for longer times. As described in Section 3.2.3, a feed forward neural network is trained to predict all positions of trajectories starting from their respective initial position and the time delta for which the next position ought to be predicted. Again, the dataset consists out of the 90 trajectories with a length of 40 oscillation periods sampled in the latent space of the autoencoder described above. The model used

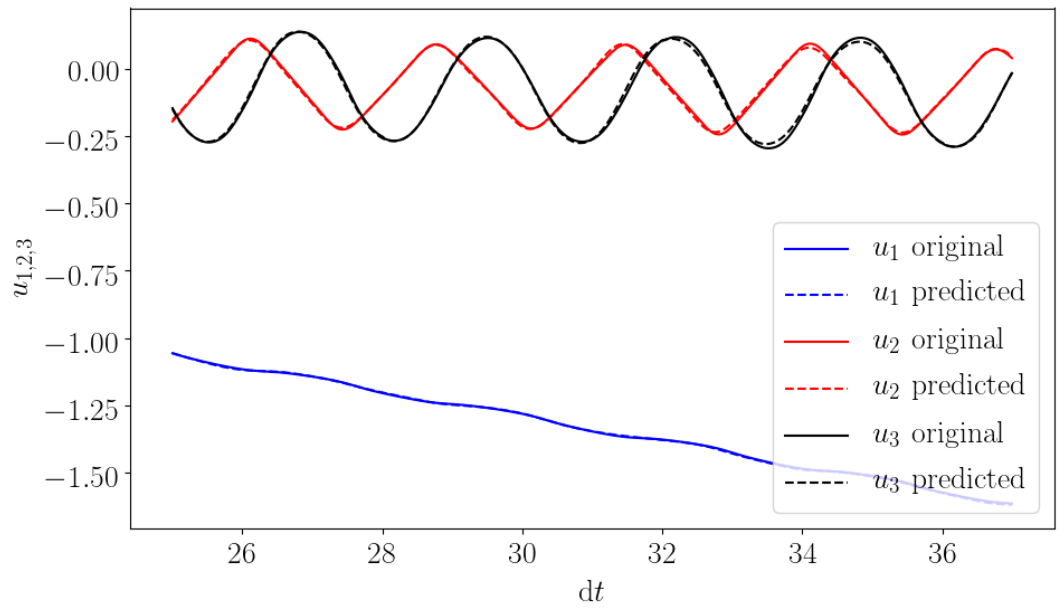


Figure 4.13: Comparison of LS coordinates (u_1, u_2, u_3) for six oscillation periods of the NHIM, predicted by the neural network described in Section 3.2.3 (dashed lines) and by transforming an original trajectory on the NHIM into the LS via the encoder (solid lines). There is hardly any visible deviation between the courses of the original and the predicted trajectory and small deviations like the ones at around d$t = 34$ do not necessarily propagate in time.

for the interpolation of the given data has a configuration of $[4 \to 50 \to 100 \to 100 \to 100 \to 50 \to 3]$ nodes per layer and arrives at a mean squared error of MSE $= 2.7 \times 10^{-3}$ for the validation set and MSE $= 5.8 \times 10^{-5}$ for the training data. This clear overfitting of data was tried to be prevented by using various methods such as decreasing the model's size, increasing the size of the dataset and introducing a weight decay to the training process. However, none of these methods leads to better results for the validation set. It should also be noted that, since this is an interpolation method, the time scale that can be predicted is limited to that of the training data.

Nevertheless, it is found that even though the model is overfitting, the trajectories of the validation set are still predicted with a sufficient accuracy, as it can be seen in Figure 4.13. The plot shows the comparison of a predicted and a original trajectory that is part of the validation set for the same time scale as in Figure 4.12 (b), where the deviation starts to grow over time. Here, a deviation in one time step does not cause an even bigger deviation for the subsequent one because they do not explicitly depend on each other. Consequently, the slight deviation in latent space coordinate u_3 at time step d$t = 34$ that can be seen in Figure 4.13 does not continue for the following time steps.

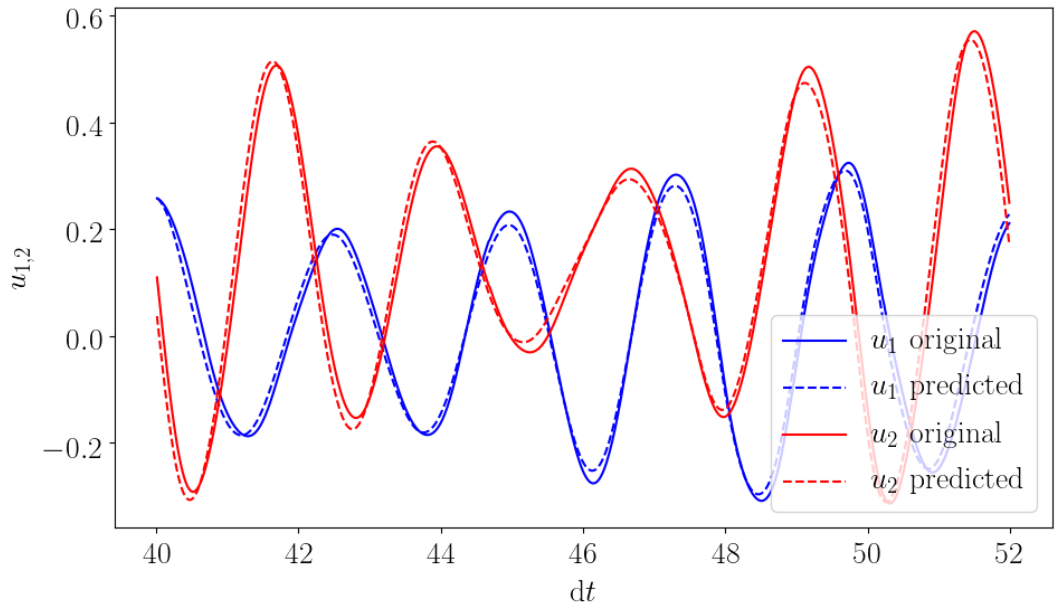


Figure 4.14: Comparison of LS coordinates (u_1, u_2) for six oscillation periods of the NHIM, obtained by numerically solving the description given by a recurrent neural network model of the type described in Section 3.2.4 (dashed lines) and by transforming an original trajectory on the NHIM into the LS by the AE (solid lines). There are small deviations between the courses of the original and the predicted trajectory. However, a deviation at one point does not necessarily propagate in time as it does in Figure 4.12.

With this method, it is possible to predict trajectories starting from anywhere in the reduced space for the time scale that the model was trained on. In addition and in contrast to the results presented in Sections 4.2.1 and 4.2.2, the predicted trajectories are stable over time.

4.2.4 Recurrent neural network model for prediction – results

All of the previous methods used to predict trajectories in the reduced space are not ideally suited for predicting large time scales due to a growing inaccuracy over time or the limitation to the time scale given by the training data. Like any iterative approach, this method, described in Section 3.2.4, can in theory be used to predict trajectories for any time scale.

In contrast to all othe methods, this method is tested for the latent space of an autoencoder trained on the phase information of the NHIM rather than the absolute time.

The resulting latent space does not have to be artificially extended in time in order to capture trajectories of arbitrary length due to its inherently periodic nature.

As shown in the sketch in Figure 3.11, the model used for the prediction consists out of three units: two feed forward neural networks and one recurrent neural network in combination with a feed forward network. For the results presented in this section, the feed forward neural network encoding the initial position of a trajectory has a configuration of $[4 \rightarrow 64 \rightarrow 128 \rightarrow 64 \rightarrow 8]$ nodes per layer, while the one predicting the subsequent position consists of $[4 \rightarrow 32 \rightarrow 128 \rightarrow 128 \rightarrow 32 \rightarrow 4]$ nodes per layer. The third unit consists of a recurrent neural network with three layers of 128 nodes each and two subsequent feed forward layers with 32 and 4 nodes. The network is trained for a total length of seven oscillation periods of the external driving for a total of 400 different trajectories, following the procedure described in Sec. 3.2.4.

A new trajectory can be generated by applying the model iteratively to an arbitrary initial position in the latent space. By doing so, the trajectory is built step by step for any desired length. Normally, an iterative approach is prone to the propagation of errors. However, as can be seen in Figure 4.14, deviations of the predicted trajectory do not cause a growing error in the following time steps. This is most likely due to the recurrent neural network unit of the model. It does not make predictions based solely on the previous time step of the trajectory, but rather keeps a memory of the overall course, encoded in the hidden state of the recurrent neural network. It should also be noted, that the network performs well on predicting trajectories for significantly larger time scales than it was trained on. Figure 4.14 shows a trajectory for periods 20 to 26, while the model itself is only trained on data up to seven oscillation periods.

By using this iterative method, it is possible to generate new trajectories starting from anywhere in the reduced space and for an arbitrary length of time. Rapidly growing deviations in time are prevented by the use of a recurrent neural network, however, it can not be completely avoided for long time scales.

4.2.5 Validation of predicted trajectories

In order to validate the methods used to predict trajectories, it is necessary to transform them from the reduced space of the respective autoencoder back into the original space of the system first. This transformation can easily be done by using the decoder part of the autoencoder. Back in the original space, the trajectories can be validated in regards to the deviation from the NHIM, the deviation from the original trajectories coming from the simulation, as well as the Floquet rate of each original and predicted trajectory. All of the properties just mentioned are analyzed for all methods presented in Sec. 4.2 except for the symbolic regression analysis due to its insufficient performance for larger time scales.

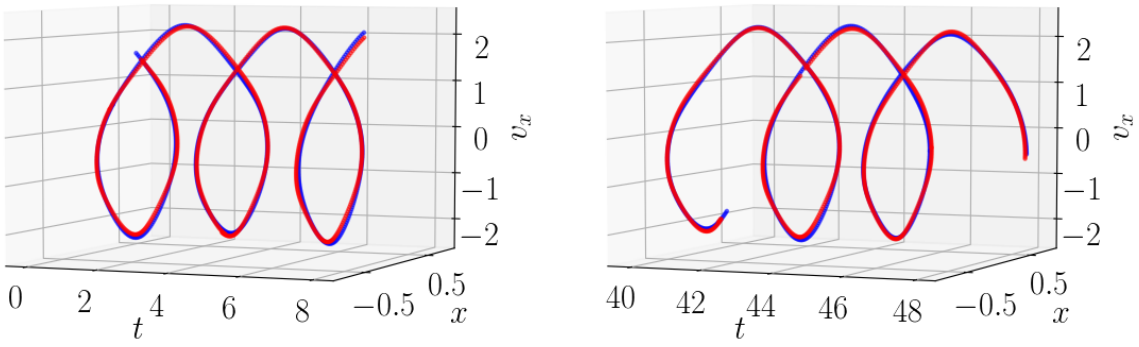
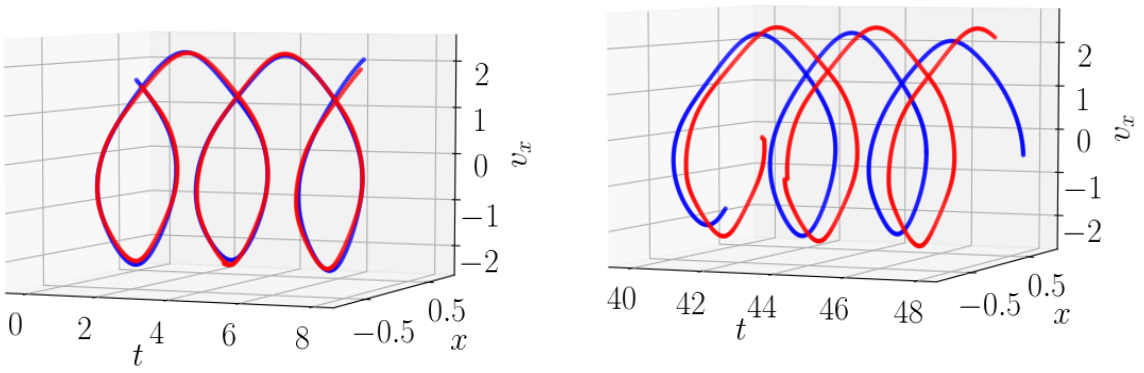


Figure 4.15: Comparison of a trajectory originating directly from the simulation and a trajectory that is processed by an AE of the type described in Section 3.1.1 once. Both plots show three of the in total five dimensions of the original space (t, x, y, v_x, v_y). Left: Trajectories for the first four periods of the external driving. Right: Trajectories for system times $t = 40$ to $t = 48$. For both time intervals, the course of the predicted trajectory closely follows that of the original one, indicating that the processing of trajectories by the AE is stable over time.

Autoencoder – absolute time input

When transforming the trajectories, predicted in an artificially extended latent space, back into the original space via the decoder, it is necessary to first transform them back into the ranges the autoencoder was trained on. In other words, the artificial extension of the latent space, as described in Section 3.1.1, has to be reversed first. This is done by successively subtracting the same shifts that were once added to the data in order to extend it in time. Afterwards, the trajectories are no longer contiguous and instead all sections lay in the same spatial ranges suited for a transformation by the decoder. Once they are transformed back to the original space, each section is extended in time again in order to form a contiguous trajectory of the same time scale as in the latent space.

(a)



(b)

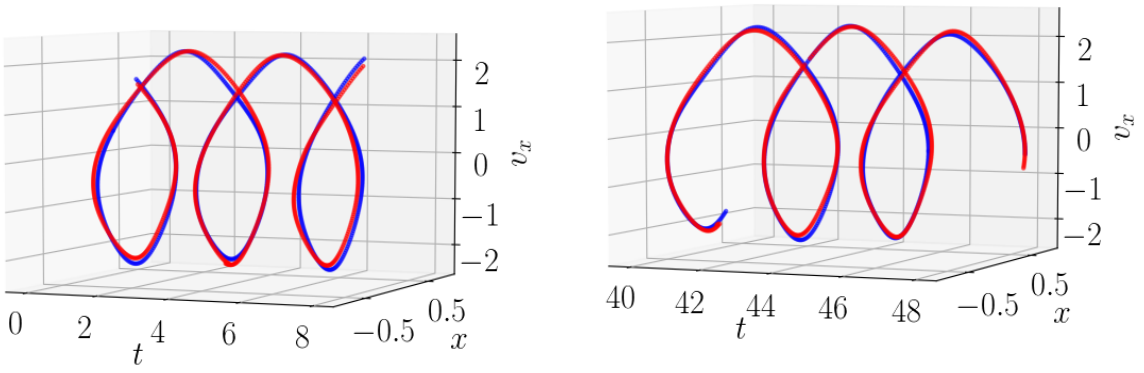


Figure 4.16: Comparison of a trajectory originating directly from the simulation and a decoded trajectory predicted by numerically solving the differential equation system given by a neural network of the type described in Section 3.2.2 (a) and by a neural network of the type described in Section 3.2.3 (b). Similar to Figure 4.15, the plots show three of the in total five dimensions of the original space (t, x, y, v_x, v_y). Left: Trajectories for the first four periods of the external driving. Right: Trajectories for system times $t = 40$ to $t = 48$. (a) The course of the predicted trajectory closely follows that of the original one for the first four periods. For periods 20 to 24, the predicted trajectory no longer matches that of the original one and is shifted in time, indicating that the prediction of trajectories given by the procedure described in Section 3.2.2 is not stable over time. (b) By contrast, the course of the predicted trajectory follows that of the original one rather closely for both time intervals, indicating that the prediction of trajectories given by the procedure described in Section 3.2.3 is stable over time.

Figures 4.15-4.16 show coordinates t, x and v_x of the original space of an exemplary trajectory originating from the simulation compared to the corresponding trajectory originating from the methods presented in Sections 3.2.2 (EOM-NN method) and 3.2.3 (prediction method), as well as from being run through the autoencoder once which serves as a reference for the performance of the AE. All Figures show this comparison for the first four oscillation periods as well as the 20th to 24th ones in order to capture the course over time. As expected, the output of the autoencoder in Figure 4.15 as well as the resulting trajectory of the prediction method in Figure 4.16 (b) show a rather high accuracy and are stable over time, while the one resulting from the EOM-NN method in Figure 4.16 (a) shows a good accuracy in the beginning but is severely shifted in phase from the original trajectory for later times. The stability over time is further analyzed in Figure 4.17 showing the mean deviations of the trajectories processed by the autoencoder from the original ones. While the ones that are run through the autoencoder once have very little deviation that does not change in time at all, the ones resulting from the prediction method show a small increase in deviation that stagnates at around ten oscillation periods. Only the EOM-NN method leads to a rapidly growing deviation in time with seemingly no upper bound.

As mentioned above, the resulting trajectories are validated in terms of deviation to the NHIM, deviation to the original trajectories and their respective Floquet rate. These properties are calculated for trajectories with 20 different initial positions sampled throughout the latent space. Table 4.1 shows a comparison between trajectories generated directly by the simulation, ones that are processed by the autoencoder once, ones that are generated by the prediction method and ones that are generated by the method of interpolating gradients.

Coming straight from the simulation, the trajectories lay on the NHIM with a high accuracy of approximately $\overline{\Delta}_{\mathrm{NHIM}} = 10^{-6}$. When processed by the autoencoder, either as an input or created in the latent space, the trajectories lay on the NHIM with a similar and much lower accuracy of approximately $\overline{\Delta}_{\mathrm{NHIM}} = 0.13$. This accuracy is simply limited by the autoencoder's performance and may be increased by using a better model. However, what is striking about this result is the similarity of the mean deviations from the NHIM for trajectories processed by the autoencoder regardless of the method that was used to create them. This can be explained by the autoencoder's characteristic of always mapping data to the space that it is trained on, which is analyzed in Section 4.1.2. Once a trajectory is either mapped to the known latent space or created in it, regardless of it being physically plausible for the system or not, it will lead to a trajectory on the NHIM in the original space with an accuracy determined by the autoencoder's general performance. This is because the latent space is nothing but a representation of the NHIM.

As can be seen in the forth column of Table 4.1, the mean deviations of trajectories processed by the autoencoder from those generated by the simulation shows some variance

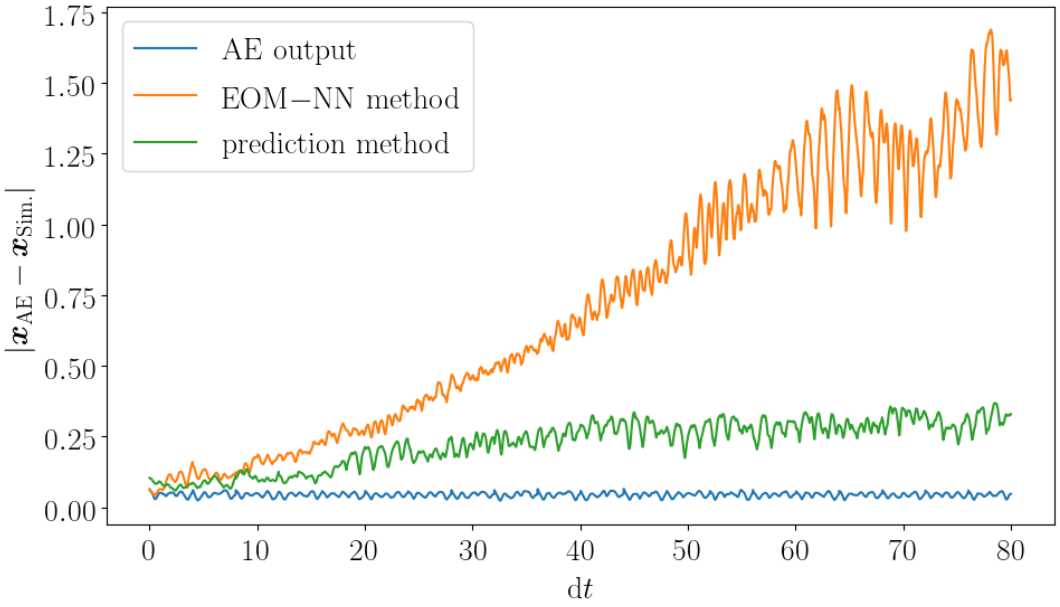


Figure 4.17: Comparison of the mean absolute deviation of trajectories $\boldsymbol{x}_{\mathrm{Sim.}}$ coming directly from the simulation and those coming from the AE $\boldsymbol{x}_{\mathrm{AE}}$, either by processing them once or by predicting them via the two prediction methods of Section 3.2.2 and 3.2.3 over time. Trajectories run through the AE once have very little deviation that stays the same over time. Those resulting from the prediction method show a small increase in deviation which then stagnates at around ten oscillation periods. Only the EOM-NN method leads to a rapidly growing deviation in time with seemingly no upper bound.

depending on how they are generated or processed. This deviation is the mean absolute difference averaged over all validation set trajectories of the prediction and the EOM-NN methods, respectively. It is lowest for trajectories that were simply ran through the autoencoder once and highest for the ones predicted by solving the network's description of their differential equation system in the EOM-NN method. The latter is mostly due to the growing deviations over time that occur in this method.

However, despite of the noticeable decrease in proximity to the NHIM and the respective original trajectories, the resulting trajectories from the autoencoder methods still fit the system with a sufficient accuracy that enables the calculation of Floquet rates with the standard procedure. Furthermore, these rates have only little deviation from the respective original ones, as can be seen in Table 4.1. It should be noted that, in the case of the EOM-NN method, this deviation is expected to grow over time and as mentioned above, this method is not suited for large time scales. Nevertheless, most trajectories coming from the autoencoder lead to a Floquet rate with an acceptable accuracy, validating the methods used to generate them.

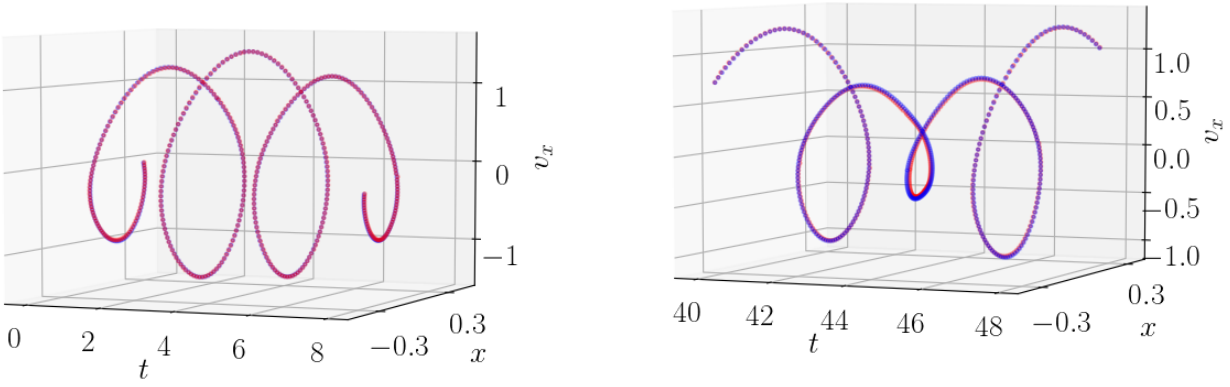
Table 4.1: Comparison of trajectories calculated via different methods in regards to various properties that are a measure of the validity of a trajectory on the NHIM. These properties include the mean deviation to the NHIM $\overline{\Delta}_{\mathrm{NHIM}}$ and standard deviation σ_{NHIM}, the mean deviation from the original trajectories $\overline{\Delta}_{\mathrm{Traj.}}$ and standard deviation $\sigma_{\mathrm{Traj.}}$, as well as the mean Floquet rate $\overline{k_{\mathrm{F}}}$ and the deviation from the rates of the original trajectories $\overline{\Delta}_{k_{\mathrm{F}}}$. The methods presented are the calculation via the simulation, the processing of such trajectories via the AE of Sec. 3.2 and LS-dim. three, the prediction of trajectories via the EOM-NN method, as well as the generation of trajectories via the prediction method. The values are averaged over 20 trajectories of 40 oscillation periods.

Method	$\overline{\Delta}_{\mathrm{Nhim}}$	σ_{Nhim}	$\overline{\Delta}_{\mathrm{Traj.}}$	$\sigma_{\mathrm{Traj.}}$	$\overline{k_{\mathrm{F}}}$	$\overline{\Delta}_{k_{\mathrm{F}}}$
Simulation	5.54×10^{-6}	3.576×10^{-5}	–	–	5.058	–
AE	0.131	0.077	0.044	0.020	4.973	0.070
EOM-NN	0.130	0.0842	0.716	0.798	5.016	0.123
NN predict	0.130	0.076	0.226	0.372	4.945	0.098

Autoencoder – oscillation phase input

Similar to Sec. 4.2.5, this section deals with the validation of trajectories created in the LS. However, the trajectories of interest are predicted with the RNN model of Sec. 4.2.4 which used an iterative approach to generate trajectories in the LS of the autoencoder presented in Sec. 4.1.3. Since this AE is trained on the oscillation phase of the system rather than the absolute time, quasi-periodic trajectories can be fully represented in its LS for any time scale without having to artificially extend it in time like it is done for the AE with the absolute time input. Therefore, trajectories created in this LS can simply be run through the decoder as they are in order to transform them back into the original space. Lastly, the phase information has to be translated into the absolute time again. Back in the five-dimensional space (t, x, y, v_x, v_y), the trajectories are analyzed in regards to their stability over time, deviation from the NHIM, deviation from the respective trajectories coming straight from the simulation and the Floquet rates.

(a)



(b)

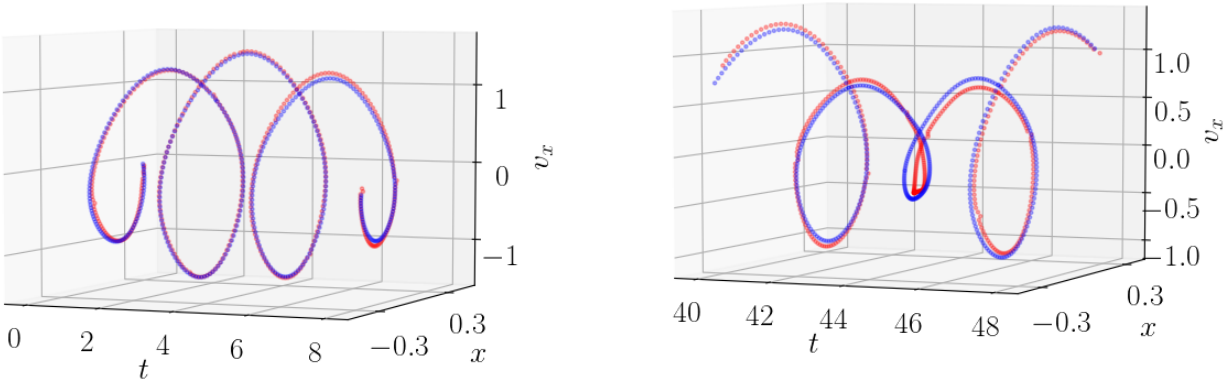


Figure 4.18: Comparison of trajectories similar to Figure 4.15. The course of a trajectory coming from the simulation is compared to that processed by the AE of Section 4.1.3, either as an input (a) or predicted by the RNN method (b) of Section 4.2.4. (a) The trajectory that is run through the AE once shows almost no visible deviation from the one coming directly from simulation for both time intervals. (b) The trajectory predicted by the RNN model matches the one from the simulation quite well for the first four periods, but starts to slightly deviate for later times. However, this deviation is not as severe as the phase shift observed for the EOM-NN method in Figure 4.15 (a).

Figure 4.14 shows comparisons of a trajectory that was processed by the AE once (a), as well as predicted by the RNN model (b) to the respective original trajectory coming directly from the simulation. The comparison is done for the first four periods of the external driving, as well as at a later time from 20 to 24 periods in order to observe their stability over time. The trajectory that is run through the AE once has almost no visible deviation to the original trajectory for both time intervals, meaning its transformation of trajectories through the LS is stable over time. By contrast, the trajectory predicted by the RNN model shows some deviations from the original one in both time intervals. For the first four oscillation periods this deviation is minimal but it becomes more visible for later times. However, these deviations are rather small and there is no shift in phase between the predicted and original trajectory like it is observed in Figure 4.16 for the EOM-NN method. This stands out since the RNN model, like the numerical solving in the EOM-NN method, predicts trajectories iteratively and errors are expected to propagate and cause a more severe shift over time. Hence, even though the deviation grows in time, the recurrent building block of the RNN model seems to mitigate the propagation of errors.

The course of the deviation of trajectories from the RNN model is shown in Figure 4.19. The deviation starts to grow linearly after approximately $\mathrm{d}t = 14$, which is the time span the RNN model was trained on. Even though this linear growth has similarities to the one of the EOM-NN method in Figure 4.17, it should be noted that the deviation is reduced by factor 10. In part, this is also due to the higher performance of the AE used to transform the trajectories from the LS into the original space. However, when comparing the RNN model to the EOM-NN method, the AEs' influences on the trajectories' deviations is small compared to the influence of the methods themselves. Consequently, the RNN method has a significantly higher performance than the other iterative approach of the EOM-NN method and could potentially be used for longer time scales until the deviation is too high.

As mentioned above, the predicted trajectories are validated in regards to deviation from the NHIM, deviation from the respective original trajectory from the simulation, as well as their Floquet rates. All of these properties are presented in Table 4.2.

Similar to the Sec. 4.2.5, the deviation from the NHIM is very small for trajectories from the simulation itself and lays in the same order of magnitude for trajectories coming from the AE, either by running them through the model once or by predicting them in the AE's LS via the RNN method. However, in contrast to the values of Table 4.1, the values for the trajectories coming from the AE are not as similar to each other and the value for the trajectories from the RNN method is twice as high. This might be due to the nature of the AE's latent space, where two of the four dimensions hold the phase information in form of a $\sin(\omega_x\, t)$ and $\cos(\omega_x\, t)$. Since these values depend on each other, whenever the RNN model predicts phase variables that do not exactly match, this position deviates

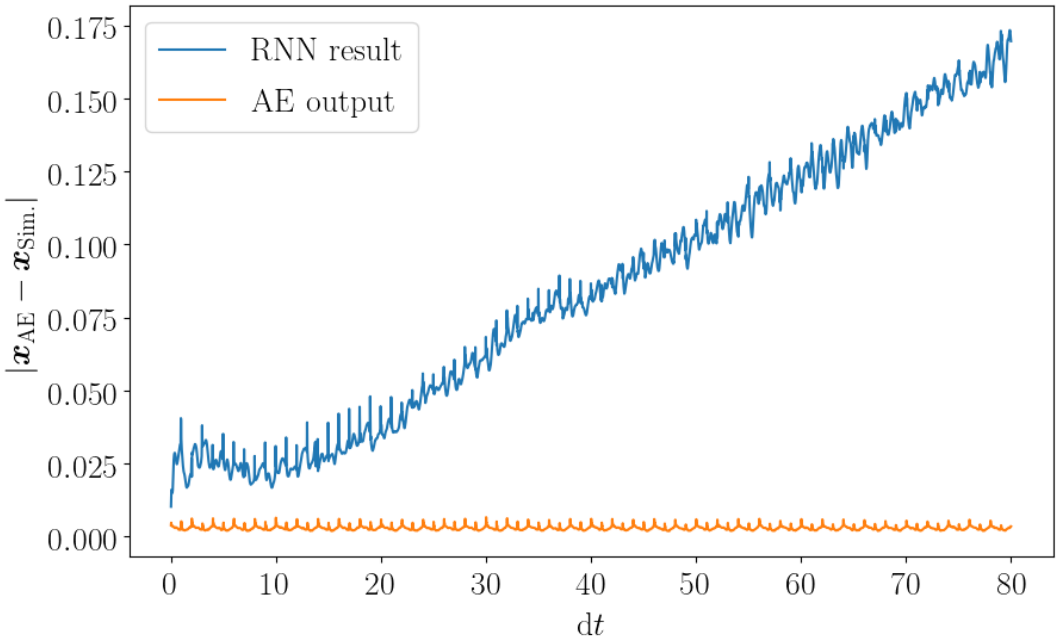


Figure 4.19: Mean absolute deviation of the trajectories $\vec{x}_{\mathrm{AE}}$ processed by the AE of Section 4.1.3, either as an input or predicted by the RNN method of Section 4.2.4 from the respective trajectories $\vec{x}_{\mathrm{Sim.}}$ given by the simulation in the original space over time. Trajectories run through the AE show no increase of the deviation at all. Trajectories predicted by the RNN model show a linearly increasing deviation after about seven oscillation periods of the NHIM, which is also the time scale the model was trained on. However, the deviation here is minimal compared to the other iterative approach of the EOM-NN method in Figure 4.17.

Table 4.2: Comparison of the same validation properties for trajectories on the NHIM as in Table 4.1. The results are averaged over 100 trajectories, all part of the validation set of the RNN model, for 40 oscillation periods for trajectories originating directly from the simulation, processed once by the AE presented in Sec. 4.1.3, as well as predicted by the RNN model of Sec. 4.2.4.

Method	$\overline{\Delta}_{\mathrm{Nhim}}$	σ_{Nhim}	$\overline{\Delta}_{\mathrm{Traj.}}$	$\sigma_{\mathrm{Traj.}}$	$\overline{k_{\mathrm{F}}}$	$\overline{\Delta}_{k_{\mathrm{F}}}$
Simulation	6.54×10^{-6}	3.99×10^{-5}	–	–	5.066	–
AE	0.004	0.003	0.003	0.001	5.066	0.004
RNN	0.008	0.008	0.101	0.102	5.040	0.027

slightly from the learned LS and consequently also deviates from the NHIM back in the original space.

As can be seen in Figure 4.19, the deviation of the trajectories from the original trajectory from the simulation grows over time. Despite this behavior, the RNN model leads to trajectories that have a mean deviation from the original ones of only $\overline{\Delta}_{\text{Traj.}} = 0.101$ for the time scale of 40 oscillation periods, compared to $\overline{\Delta}_{\text{Traj.}} = 0.716$ of the EOM-NN method and $\overline{\Delta}_{\text{Traj.}} = 0.226$ for the prediction method.

The calculated mean Floquet rates of trajectories coming from the AE match those coming from the simulation extremely well, which is a very important measure of validation when it comes to trajectories on the NHIM.

Since the calculation of rates for the dynamics on the NHIM is one of the ultimate goals in TST, this shows that by using an autoencoder to first parametrize the NHIM and then analyzing and reproducing the dynamics in this reduced space, it is possible to quickly calculate new valid trajectories without doing any simulation. Especially in higher-dimensional systems, where simulating trajectories and stabilizing them on the NHIM might become extremely time consuming, making use of the presented methods could be of great advantage.

5 Conclusion and outlook

In this thesis, it was successfully shown how the dimension reduction of a critical manifold, the *Normally Hyperbolic Invariant Manifold* (NHIM), of a rank-1 saddle system can provide a better interpretability and analysis of the dynamics on it. For this task, various autoencoder (AE) neural network architectures were developed and examined on their performance on processing the oscillating and multidimensional NHIM.

Two main AE architectures were developed. Both architectures processed the NHIM's positions in the phase space, however, one model was designed to incorporate the absolute time of the system and the other one processed the phase information of the NHIM. While the AE with the absolute time input is also suited for parameterizing non-periodic objects, the other model is specifically designed for periodic manifolds like the NHIM.

In order to gain more insight into how an AE processes data and how its hyperparameters effect performance, the AE model with the absolute time input was evaluated for various different architectures such as different latent space dimensions, number of hidden layers, number of nodes per layer, as well as for different activation functions.

Section 4.1.2 demonstrated how the AE can be used for stabilizing points on the NHIM, since, once it is fully trained on NHIM data with high precision, it maps noisy input points back onto the known space. In future work it might be interesting to develop a *denoising AE* that is specifically trained on reducing the deviation of noisy input points to the NHIM. This could be achieved by comparing the model's output to noise-free data and also calculating the respective deviation of the model's output to the NHIM and adding it to the loss function as an additional term.

The AE model with the oscillation phase input lead to an essentially two-dimensional LS, which made the analysis of the dynamics on the NHIM in form of Poincaré maps pretty straight forward. The Poincaré maps revealed a bifurcation in the LS when tuning the angular frequency of the NHIM. Most importantly, unlike the ones in the original space of the system, the resulting Poincaré maps of the LS were unique, since there is only one option for selecting the intersection for the PSOS in a two-dimensional space.

Another key focus of this thesis was the analysis of the dynamics on the NHIM in its compressed form, meaning in the LS of the AEs mentioned above. The mostly quasi-periodic trajectories on the NHIM were examined with further machine learning

techniques. The goal was to capture their dynamics well enough to be able to reproduce it and generate new trajectories starting from anywhere in this reduced space.

One of the methods used a genetic programming algorithm to perform a symbolic regression analysis and found an analytical differential equation system describing the trajectories. However, the solution of this equation system deviated from the original solution after just one oscillation period of the NHIM. Nonetheless, with this method is was possible to calculate trajectories starting from anywhere in the LS with a sufficient accuracy for the first oscillation period.

Another method used a feed forward neural network for interpolation in order to find a more accurate description of a differential system of equations of the trajectories. Indeed, this approach lead to a much more accurate description of the data. After solving the neural network based differential system of equations numerically, the solution remained stable for up to 15 oscillation periods.

In order to predict stable trajectories over time, another feed forward neural network was used to predict the positions of trajectories at any given time step solely from its initial position as well as the time difference. This method lead to stable solutions with high accuracy compared to the previous methods.

Another iterative approach for predicting trajectories was developed by combining feed forward neural networks and a recurrent neural network (RNN). Even though this was an iterative approach, the effects of growing deviations over time steps were mitigated by the special architecture of the RNN model.

Trajectories generated in the LS were transformed back into the original space and validated in terms of deviation to the NHIM, deviation from the original trajectory and deviation of their respective Floquet rate. The results showed that, within the limitations of the AE model itself, the trajectories matched the system well. Most importantly, the trajectories lead to Floquet rates that were in good agreement with those of the original trajectories. Since the calculation of decay rates of trajectories on the NHIM is one of the ultimate goals in Transition State Theory (TST), the methods presented are proven to be useful tools to quickly generate valid trajectories starting from anywhere on the NHIM.

In this thesis, all methods were applied to a rather simple, well understood model system, for which already a lot of efficient tools for calculating certain properties exist. For future work it would be interesting to apply the methods developed in this thesis to higher-dimensional systems, where interpretation of the dynamics, as well as propagating and stabilizing particles on the NHIM might become increasingly difficult. Here, AE models could help to interpret and stabilize the dynamics on the NHIM, while the trajectory generation methods could help to quickly predict trajectories in the LS of these AEs. This could potentially replace a lot of computationally expensive calculations.

In future work it might also be interesting to compare the parameterization via the AEs with the method presented in Ref. [6], where the NHIM is simply given by (t, v, v_y), while the missing coordinates (x, v_x) are reconstructed via a feed forward neural network.

6 Zusammenfassung in deutscher Sprache

Diese Arbeit befasst sich mit der Dimensionalitätsreduktion einer speziellen Mannigfaltigkeit in einem Rang-1-Sattel-System, das die Übergänge von klassischen Objekten zwischen zwei Zuständen im Rahmen der Transition State Theory (TST) modelliert. Diese Dimensionsreduktion erfolgt durch die Anwendung einer Methode des maschinellen Lernens (ML), einer künstlichen neuronalen Netz Architektur namens Autoencoder (AE). Der zweite Fokus dieser Arbeit liegt auf der Analyse und Reproduktion der Dynamik auf dieser Mannigfaltigkeit in ihrer, durch die AE Modelle gegebene, reduzierten Form mittels weiterer ML-Techniken.

Wie oben erwähnt, modellieren wir die Übergänge von klassischen Objekten zwischen zwei Zuständen. Diese Zustände sind in der Regel durch lokale Minima auf einer Potentialenergielandschaft charakterisiert und energetisch durch eine Barriere getrennt. Die reaktiven Objekte müssen diese Energiebarriere überwinden, um in den Produktzustand überzugehen. Das spezielle System, welches Gegenstand dieser Arbeit ist, wird durch eine instabile Reaktionskoordinate und einen stabilen Freiheitsgrad, die orthogonale Mode, beschrieben. Außerdem ist das System zeitabhängig und die Barriere oszilliert entlang der Reaktionskoordinate. Zwei Mannigfaltigkeiten, eine stabile und eine instabile, können im Phasenraum eines solchen Systems identifiziert werden. Teilchen, die auf der stabilen Mannigfaltigkeit initialisiert werden, nähern sich dem Maximum der Barriere asymptotisch schnell, während Teilchen, die auf der instabilen Mannigfaltigkeit initialisiert werden, dasselbe Verhalten zeigen, wenn sie in der Zeit rückwärts propagiert werden. Diese beiden Mannigfaltigkeiten überschneiden sich in einer Region, die eine weitere Mannigfaltigkeit definiert, die Normally Hyperbolic Invariant Manifold (NHIM). Diese spezielle Mannigfaltigkeit ist Gegenstand der Dimensionalitätsreduktion durch die AE-Modelle. Wenn Teilchen auf der NHIM initialisiert werden, sind sie an sie gebunden und bleiben für immer auf ihr, wenn sie sowohl vorwärts als auch rückwärts in der Zeit propagiert werden. Die NHIM und die Dynamik auf ihr sind von großem Interesse, da sie Einblicke in die allgemeine Reaktionsdynamik des Systems geben und die Berechnung von Zerfallsraten ermöglichen, welche ein Maß für die Stabilität des Systems sind.

Die Interpretation der Struktur der mehrdimensionalen und zeitabhängigen NHIM, sowie die Propagation von Teilchen auf ihr kann, insbesondere bei höherdimensionalen Systemen, sehr schwierig und rechenaufwändig werden. In dieser Arbeit wird gezeigt, wie

die Dimensionsreduktion der NHIM eines zweidimensionalen Modellsystems durch AE eine niedriger dimensionale Darstellung liefern kann, die einfacher zu interpretieren ist und dennoch alle relevanten Eigenschaften der NHIM erfasst. Außerdem wird die Rolle des AE als Stabilisator von Teilchen auf dem NHIM untersucht. Darüber hinaus werden vier ML-Methoden entwickelt, mit denen sich die Trajektorien auf der NHIM in ihrer niedrigdimensionalen Darstellung für verschiedene Zeitskalen erfolgreich vorhersagen lassen. Mit Hilfe dieser Methoden ist es möglich, Trajektorien von Teilchen zu generieren, die an beliebigen Positionen des niedrigdimensionalen Raums der NHIM initialisiert wurden. Diese können dann über die AEs zurück in den ursprünglichen Raum des Systems transformiert werden.

Die im Zuge dieser Arbeit entwickelten Methoden wurden auf ein recht einfaches, gut verstandenes Modellsystem angewandt, für das es bereits eine Reihe effizienter Werkzeuge zur Berechnung bestimmter Eigenschaften gibt. Für zukünftige Arbeiten wäre es daher interessant, die entwickelten Methoden auf höherdimensionale Systeme anzuwenden, bei denen die Interpretation der Dynamik sowie die Propagation und Stabilisierung von Teilchen auf der NHIM zunehmend schwieriger werden könnte. Hier könnten AE-Modelle dabei helfen, die Dynamik auf der NHIM zu interpretieren und zu stabilisieren, während die Methoden zur Trajektoriengenerierung helfen könnten, Trajektorien in der, durch die AE Modelle gegebenen, reduzierten Darstellung der NHIM schnell vorherzusagen. Dies könnte möglicherweise viele kostspielige Propagationen ersetzen.

Bibliography

1. Kassel, L. S. Statistical Mechanical Treatment of the Activated Complex in Chemical Reactions. *J. Chem. Phys.* **3,** 399–400. eprint: doi:10.1063/1.1749687 (1935).
2. E. P. Wigner. Calculation of the Rate of Elementary Association Reactions. *J. Chem. Phys.* **5,** 720–725. eprint: doi:10.1063/1.1750107 (1937).
3. Truhlar, D. G., Garrett, B. C. & Klippenstein, S. J. Current Status of Transition-State Theory. *J. Phys. C* **100,** 12771–12800. eprint: doi:10.1021/jp953748q (1996).
4. Mullen, R. G., Shea, J.-E. & Peters, B. Communication: An existence test for dividing surfaces without recrossing. *J. Chem. Phys.* **140,** 041104. eprint: doi:10.1063/1.4862504 (2014).
5. Wiggins, S. The role of normally hyperbolic invariant manifolds (NHIMS) in the context of the phase space setting for chemical reaction dynamics. *Regul. Chaotic Dyn.* **21,** 621–638. eprint: doi:10.1134/S1560354716060034 (2016).
6. Tschöpe, M., Feldmaier, M., Main, J. & Hernandez, R. Neural network approach for the dynamics on the normally hyperbolic invariant manifold of periodically driven systems. *Phys. Rev. E* **101,** 022219. eprint: doi:10.1103/PhysRevE.101.022219 (2020).
7. De Oliveira, H. P., Ozorio de Almeida, A. M., Damião Soares, I. & Tonini, E. V. Homoclinic chaos in the dynamics of a general Bianchi type-IX model. *Phys. Rev. D* **65,** 083511/1–9. eprint: doi:10.1103/PhysRevD.65.083511 (2002).
8. Jaffé, C., Farrelly, D. & Uzer, T. Transition State Theory without Time-Reversal Symmetry: Chaotic Ionization of the Hydrogen Atom. *Phys. Rev. Lett.* **84,** 610–613. eprint: doi:10.1103/PhysRevLett.84.610 (2000).
9. Komatsuzaki, T. & Berry, R. S. Regularity in chaotic reaction paths. I. Ar_6. *J. Chem. Phys.* **110,** 9160–9173. eprint: doi:10.1063/1.478838 (1999).
10. Komatsuzaki, T. & Berry, R. S. Chemical reaction dynamics: Many-body chaos and regularity. *Adv. Chem. Phys.* **123,** 79–152. eprint: doi:10.1002/0471231509.ch2 (2002).
11. Huepe, C., Tuckerman, L. S., Métens, S. & Brachet, M. E. Stability and decay rates of nonisotropic attractive Bose-Einstein condensates. *Phys. Rev. A* **68,** 023609. eprint: doi:10.1103/PhysRevA.68.023609 (2003).

12. Junginger, A., Main, J., Wunner, G. & Dorwarth, M. Transition state theory for wave packet dynamics. I. Thermal decay in metastable Schrödinger systems. *J. Phys. A: Math. Theor.* **45,** 155201. eprint: doi:10.1088/1751-8113/45/15/155201 (2012).

13. Junginger, A., Dorwarth, M., Main, J. & Wunner, G. Transition state theory for wave packet dynamics. II. Thermal decay of Bose-Einstein condensates with long-range interaction. *J. Phys. A: Math. Theor.* **45,** 155202. eprint: doi:10.1088/1751-8113/45/15/155202 (2012).

14. Junginger, A., Kreibich, M., Main, J. & Wunner, G. Transition states and thermal collapse of dipolar Bose-Einstein condensates. *Phys. Rev. A* **88,** 043617. eprint: doi:10.1103/PhysRevA.88.043617 (2013).

15. Wiggins, S. *Normally Hyperbolic Invariant Manifolds in Dynamical Systems* (Springer, New York, 1994).

16. Feldmaier, M., Schraft, P., *et al.* Invariant manifolds and rate constants in driven chemical reactions. *J. Phys. Chem. B* **123,** 2070–2086. eprint: doi:10.1021/acs.jpcb.8b10541 (2019).

17. Feldmaier, M. Phase-space resolved decay rates of driven systems near the transition state (2020).

18. Pollak, E. & Pechukas, P. Transition States, Trapped Trajectories, and Classical Bound States Embedded in the Continuum. *J. Chem. Phys.* **69,** 1218–1226. eprint: doi:10.1063/1.436658 (1978).

19. Pechukas, P. & Pollak, E. Classical Transition State Theory is Exact if the Transition State is Unique. *J. Chem. Phys.* **71,** 2062–2068. eprint: doi:10.1063/1.438575 (1979).

20. Hernandez, R. & Miller, W. H. Semiclassical Transition State Theory. A New Perspective. *Chem. Phys. Lett.* **214,** 129–136. eprint: doi:10.1016/0009-2614(93)90071-8 (1993).

21. Uzer, T., Jaffé, C., Palacián, J., Yanguas, P. & Wiggins, S. The geometry of reaction dynamics. *Nonlinearity* **15,** 957–992. eprint: doi:10.1088/0951-7715/15/4/301 (2002).

22. Çiftçi, Ü. & Waalkends, H. Reaction Dynamics Through Kinetic Transition States. *Phys. Rev. Lett.* **110,** 233201. eprint: doi:10.1103/PhysRevLett.110.233201 (2013).

23. Allahem, A. & Bartsch, T. Chaotic dynamics in multidimensional transition states. *J. Chem. Phys.* **137,** 214310. eprint: doi:10.1063/1.4769197 (2012).

24. Li, C.-B., Shoujiguchi, A., Toda, M. & Komatsuzaki, T. Definability of No-Return Transition States in the High-Energy Regime above the Reaction Threshold. *Phys. Rev. Lett.* **97,** 028302. eprint: doi:10.1103/PhysRevLett.97.028302 (2006).

25. Junginger, A., Garcia-Müller, P. L., Borondo, F., Benito, R. M. & Hernandez, R. Solvated molecular dynamics of LiCN isomerization: All-atom argon solvent versus a generalized Langevin bath. *J. Chem. Phys.* **144,** 024104. eprint: doi : 10.1063/1.4939480 (2016).

26. Feldmaier, M., Reiff, J., *et al.* Influence of external driving on decays in the geometry of the LiCN isomerization. *J. Chem. Phys.* **153,** 084115. eprint: doi:10.1063/5.0015509 (2020).

27. Craven, G. T. & Hernandez, R. Deconstructing field-induced ketene isomerization through Lagrangian descriptors. *Phys. Chem. Chem. Phys.* **18,** 4008–4018. eprint: doi:10.1039/c5cp06624g (2016).

28. A. Junginger. *Transition state theory for wave packet dynamics and its application to thermal decay of metastable nonlinear Schrödinger systems* PhD thesis (University of Stuttgart, 2014).

29. Mancho, A. M., Wiggins, S., Curbelo, J. & Mendoza, C. Lagrangian Descriptors: A Method for Revealing Phase Space Structures of General Time Dependent Dynamical Systems. *Commun. Nonlinear Sci. Numer. Simul.* **18,** 3530–3557. eprint: doi:10.1016/j.cnsns.2013.05.002 (2013).

30. Bardakcioglu, R., Junginger, A., Feldmaier, M., Main, J. & Hernandez, R. Binary contraction method for the construction of time-dependent dividing surfaces in driven chemical reactions. *Phys. Rev. E* **98,** 032204. eprint: doi : 10 . 1103 / PhysRevE.98.032204 (2018).

31. H. A. Kramers. Brownian motion in a field of force and the diffusion model of chemical reactions. *Physika* **7,** 284 (1940).

32. Hänggi, P., Talkner, P. & Borkovec, M. Reaction-rate theory: Fifty years after Kramers. *Rev. Mod. Phys.* **62.** and references therein, 251–341. eprint: doi : 10 . 1103/RevModPhys.62.251 (1990).

33. Craven, G. T., Bartsch, T. & Hernandez, R. Communication: Transition State Trajectory Stability Determines Barrier Crossing Rates in Chemical Reactions Induced by Time-Dependent Oscillating Fields. *J. Chem. Phys.* **141,** 041106. eprint: doi:10.1063/1.4891471 (2014).

34. Feldmaier, M., Bardakcioglu, R., Reiff, J., Main, J. & Hernandez, R. Phase-space resolved rates in driven multidimensional chemical reactions. *J. Chem. Phys.* **151,** 244108. eprint: doi:10.1063/1.5127539 (2019).

35. Verlet, L. Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules. *Phys. Rev.* **159,** 98–103 (1 1967).

36. Kuchelmeister, M. Untersuchung der Dynamik auf der normal hyperbolisch invarianten Mannigfaltigkeit eines periodisch getriebenen Systems mit Rang-1-Sattel. *Bachelor's Thesis (2019).*

37. Schraft, P., Junginger, A., *et al.* Neural network approach to time-dependent dividing surfaces in classical reaction dynamics. *Phys. Rev. E* **97,** 042309. eprint: doi:10.1103/PhysRevE.97.042309 (2018).

38. Mishra, P. Predicting instantaneous decay rates at the transition state using neural networks and investigating their usefulness for the reaction rates of arbitrary ensembles. *Master Thesis (2021).*

39. F.R.S., K. P. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2,** 559–572 (1901).

40. Kramer, M. A. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* **37,** 233–243 (1991).

41. Caron, M., Touvron, H., *et al. Emerging Properties in Self-Supervised Vision Transformers* 2021. arXiv: 2104.14294 [cs.CV].

42. Bardes, A., Ponce, J. & LeCun, Y. *VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning* 2021. arXiv: 2105.04906 [cs.CV].

43. Assran, M., Caron, M., *et al. Semi-Supervised Learning of Visual Features by Non-Parametrically Predicting View Assignments with Support Samples* 2021. arXiv: 2104.13963 [cs.CV].

44. Rosenblatt, F. *The perceptron, a perceiving and recognizing automaton Project Para* (Cornell Aeronautical Laboratory, 1957).

45. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Networks* **2,** 359–366. ISSN: 0893-6080 (1989).

46. Nair, V. & Hinton, G. E. *Rectified Linear Units Improve Restricted Boltzmann Machines* in *Proceedings of the 27th International Conference on International Conference on Machine Learning* (Omnipress, Haifa, Israel, 2010), 807–814. ISBN: 9781605589077.

47. Matsugu, M., Mori, K., Mitari, Y. & Kaneda, Y. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks* **16.** Advances in Neural Networks Research: IJCNN '03, 555–559. ISSN: 0893-6080 (2003).

48. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323,** 533–538 (1986).

49. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* 2017. arXiv: 1412.6980 [cs.LG].

50. Van der Maaten, L. & Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research* **9** (2008).

51. An, J. & Cho, S. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* **2,** 1–18 (2015).

52. Chaitanya, C. R. A., Kaplanyan, A. S., *et al.* Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. **36** (2017).

53. Elman, J. L. Finding Structure in Time. *Cognitive Science* **14,** 179–211 (1990).

54. Hochreiter, S. & Schmidhuber, J. Long Short-term Memory. *Neural computation* **9,** 1735–80 (Dec. 1997).

55. Cho, K., van Merrienboer, B., *et al.* Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR* **abs/1406.1078.** `http://arxiv.org/abs/1406.1078` (2014).

56. `https://gplearn.readthedocs.io/en/stable/intro.html`.

57. Udrescu, S.-M. & Tegmark, M. AI Feynman: A physics-inspired method for symbolic regression. *Science Advances* **6** (2020).

Danksagung

Zuletzt möchte ich mich bei allen bedanken, die mich beim Schreiben meiner Masterarbeit unterstützt haben. Mein Dank gilt insbesondere

- Prof. Dr. Jörg Main für die Möglichkeit meine Masterarbeit über dieses spannende Thema am ITP1 zu schreiben.
- Dr. Andrej Junginger für die Inspiration dieser Arbeit und das hervorragende Mentoring während dieser Zeit.
- Der TST-Forschungsgruppe am ITP1, bestehend aus Johannes Reiff und Micha Schleeh, die mich in ihre interessanten Diskussionen miteinbezogen haben und mir stets mit Rat und Tat zur Seite standen.
- Allen Besuchern unserer Kaffeerunde, auf die ich mich gerade im Homeoffice immer sehr gefreut habe.
- Meiner tollen Lerngruppe, ohne die das Studium nur halb so schön gewesen wäre.
- Meiner Schwester Jessica und meinem Freund Jan für das Korrektur lesen und das Interesse an meiner Arbeit, welches mir oft zu neuen Ideen verholfen hat.

Ehrenwörtliche Erklärung

Ich erkläre,

- dass ich diese Masterarbeit selbständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
- dass ich die Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe, es sei denn, der Prüfungsausschuss hat die Veröffentlichung vorher genehmigt
- und dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

Stuttgart, den 10. August, 2021 *Melissa Jacqueline Lober*