Report for Final Project from Group C

Tasks addressed: 5
Authors: Aleksi Kääriäinen (03795252)
Danqing Chen (03766464)
Julia Xu (03716599)
Joseph Alterbaum (03724310)

Last compiled: 2024–09–27

The work on tasks was divided in the following way:

Aleksi Kääriäinen (03795252)	Task 1	0%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	0%
Danqing Chen (03766464)	Task 1	0%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	100%
Julia Xu (03716599)	Task 1	0%
	Task 2	33.3%
	Task 3	33.3%
	Task 4	33.3%
	Task 5	0%
Joseph Alterbaum (03724310)	Task 1	100%
	Task 2	0%
	Task 3	0%
	Task 4	0%
	Task 5	0%

Remarks about the projects task & extent

The project was **originally planned** with **5 tasks for 5 people**. Since we are **only 4 people in the group**, we split up **one task per person** and additionally did **task 5 as an extra effort**.

Learning dynamical systems from data: Neural networks

Report on task 1, Summary of papers & small literature study

1 Summary of Provided Approaches & Relation to New Developments

The first goal of this section is to summarize and explore the publications by Rico-Martínez et al. underlying the implementation in the following chapters. Afterwards, we are going to elaborate on further works by Rico-Martínez and his colleagues around the time of 1995 and finally we are going to look at newer approaches from the last years.

1.1 2 Publications underlying the implementation

The following publications are the foundation for our implementation in the next chapters. Therefore, we want to summarize them thoroughly and look at their concepts in-depth, which include the combination of iterative solution methods for ordinary differential equations (ODEs), e.g., Runge-Kutta, and Neural Networks (NNs) as well as identifying the ODEs of a dynamical system from experimental time series data.

1.1.1 Rico-Martínez 1994 [1]

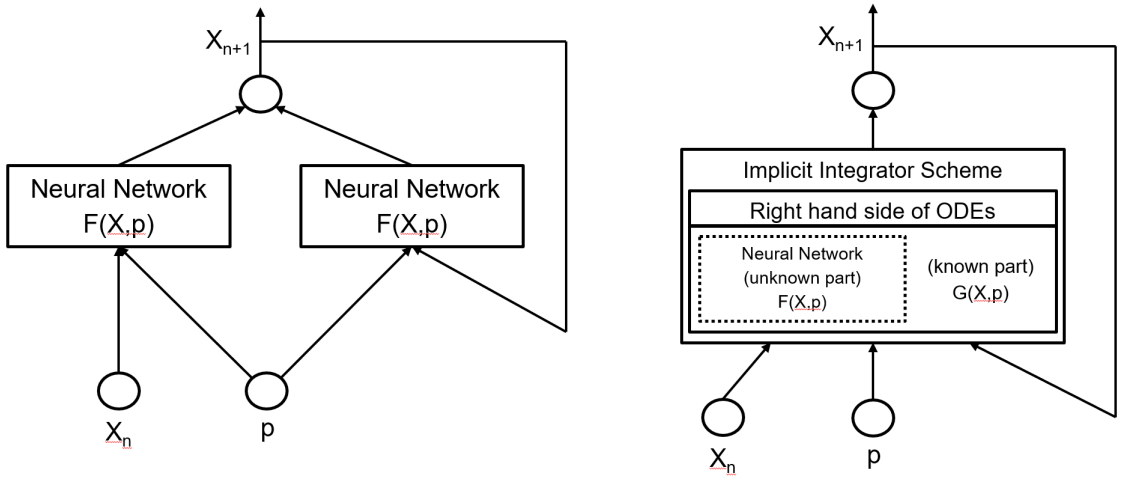
The main idea of the paper [1] is to use continuous-time models, i.e., sets of ODEs, to approximate the behaviour of non-linear dynamical systems, including long-term dynamics & bifurcations. While in a previous paper they developed a black-box approach with NNs, here they extend it to "gray-box identification" incorporating known parts of the underlying dynamical model.

The **"Black-Box approach"** is based on an approximation F, e.g., a NN, of autonomous ODEs

$$\dot{X} = F(X, p)$$

where X is the n-dimensional system state and p describes the system parameters, cp. Eq. 1 in [1]. The framework devised in [2] now integrates the NN-training from data with a numerical integrator scheme, which is in this case the trapezoidal rule, i.e., an **implicit 2nd-order Runge-Kutta method**. The framework is shown in Fig. 1a from [1] and can also handle other types of numerical integrators. The approximation of the ODEs is also possible if the time-series data is only observed for a single dimension or a subset of state dimensions. One noticeable detail is that with the use of an implicit integrator the network turns into a recurrent network because of the input of X_{n+1} on the right side. For explicit formulas normal feed forward training can be applied.

Figure 1: Two approaches shown by Rico-Martinez et al. in [1]



(a) Black-box approach

(b) Gray-box approach

The new concept of a **"Gray-Box approach"** in the paper at hand, is the combination with first principle model excerpts, illustrated in Fig. 1b from [1]. Here, the known part G of the right-hand side of the ODE is calculated explicitly while the NN-block F approximates unknown dynamical behaviour. Accordingly, the approximated ODE can be split up in the following way:

$$\dot{X} = G(X, p) + F(X, p)$$

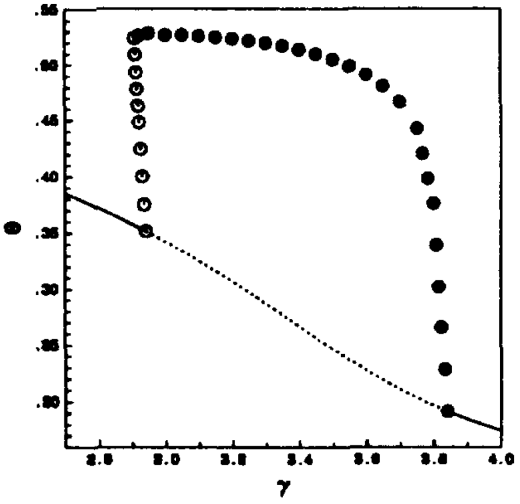
where the new function G is the available first-principle model, cp. Eq. 3 in [1]. Of course also other combinations than addition of known and unkown parts are possible. Again the underlying implicit trapezoidal rule induces a recurrent network architecture which has to be accounted for in the training of the NN.

To illustrate the approach, Rico-Martínez et al. provide an **example of a well-stirred reactor** with an irreversible reaction A to B on a catalytic surface ([1]). The system can be described by two differential equations in the state space dimensions – θ, the fractional coverage of the catalytic surface, and Π, the partial pressure of the reactant in the gas phase. The system is supposed to be one of the simplest dynamical system having oscillations in the field of catalytic reactions. Time series data was obtained for several settings of the **operational parameter** γ, which is the dimensionless temperature. Based on this parameter the system's long-term behaviour shows either steady states, oscillations or one of the two depending on the initial condition. The transitions between this behaviour are a subcritical Hopf-bifurcation (steady state to oscillation) at $\gamma = 2.941$ and a supercritical Hopf-bifurcation (oscillation to steady state) at $\gamma = 3.841$. Therefore, this system is suitable to test if the NN-architecture can approximate the dynamic behvaiour & corresponding bifurcations.

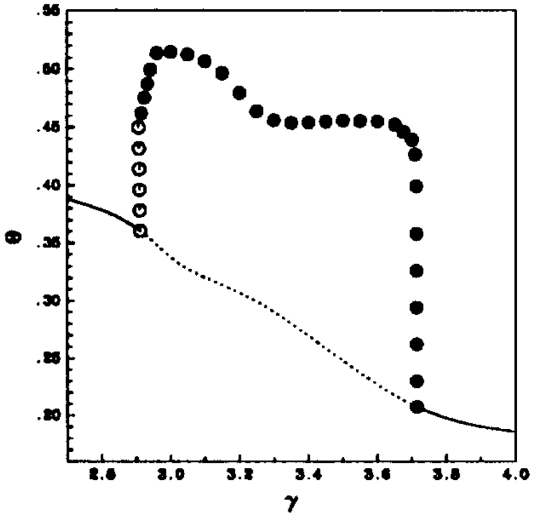
The **network architecture** was based on time-series data for θ & Π against the time τ for different regions of the parameter γ. To show the capability of the approach, one of the terms in the known pair of ODEs was then replaced by the approximation function F, in this case the desorption rate was chosen appearing in both $\frac{d\Pi}{d\tau}$ & $\frac{d\theta}{d\tau}$. The neural network itself had two input dimensions, θ & Π, 1 output dimension and 2 hidden layers of 6 neurons each with tanh-activations. For training, the derivative of the error was computed by the chain rule and the implicit function theorem, since the used integrator is implicit and the network therefore recurrent. The implicit trapezoidal rule was chosen because the underlying chemical kinetic equations are expected to be stiff. Afterwards, training of the network was conducted with a conjugate gradient algorithm on a dataset of 2950 points, with a higher point density in the multi-stability region and an integrator time-step of 0.06.

The **results** in [1] show that the neural network F successfully learns the influence of the desorption rate with regard to θ & γ from the given dataset, which was assumed to be unknown in the test setting. Furthermore, the framework correctly reproduces both Hopf-bifurcations in the system and the system's long-term dynamical behaviour, which can be seen in Fig. 2. Furthermore, the NN-approximation can be used to extract mechanistic information about the fitted part of the model, in order to for example choose from possible dynamical models for this part. In the example at hand the linear dependence of the logarithm of the desorption rate vs. $\frac{1}{\gamma}$ at constant θ was correctly predicted.

Figure 2: Bifurcation diagrams for the system in [1]



(a) Underlying bifurcation diagram of the system



(b) Bifurcation diagram predicted with the NN-approach

Own Conclusion
This paper presents a very interesting approach on identification of dynamical systems by combining NNs and existing numerical integrators. For our final project, we want to build on this approach and reconstruct their positive results with this framework. Since we do not have access to the data and experiments of this paper, we are going to use a different underlying dynamical system but the results should nevertheless reproduce the accurate reconstruction power displayed in this publication.
Furthermore, it is evident that NN-architectures popular today, like **Recurrent Neural Networks**, were already regarded 30 years ago. Because of the lack of computational power in these times, the networks were small and covered only a very limited number of neurons, e.g., 12 in this case. Nowadays, the available processing power is much bigger and the depth of NNs extended significantly leading to the field of actual Deep Learning. Accordingly, we would expect significant improvements in accuracy and complexity of underlying models when looking at current publications from this field in a later section.

1.1.2 Rico-Martínez 1995 [3]

In this very extensive journal publication, Rico-Martínez et al. again used NNs to identify models of non-linear dynamical systems from experimental time-series data [3]. To quantify their performance, three requirements are formulated – accurate short-term (1) & long-term (2) behaviour prediction and good prediction of bifurcations (3), i.e., the dependence on varying operational parameters. One described architecture-type is "discrete-time delay-based methods" [3] where the NN performs an accurate short-term prediction which is then repeatedly applied to achieve long-term prediction of behaviour. For these models, the point-wise deviations from the real & predicted time-series can be quite significant in the long-term. While it is still seen as successful if the set of steady states or attractors is similar, there are drastic shortcomings when looking at the comparison of attractors & qualitative changes in behaviour, e.g., bifurcations & instabilities. For this reason later the use of time-continuous models is recommended.

A first part of the paper elaborates on **discrete-time dealy-based modelling** where again the experimental behaviour follows a model:

$$\dot{X} = G(X, p)$$

as previously seen in [1]. The goal is now to predict a future state based on observed states $x(t_i, p)$, parameters p and a short history of states. The model can be described as the following:

$$X(t + \tau) = F(X(t), X(t - \tau), ..., X(t - (m - 1)\tau), p)$$

where F again is a discrete map in form of a NN, cp. Eq. 2 in [3]. In a deterministic system this means that the next state at time $t + \tau$ is the value of a non-linear function of measured single state variables and

enough previous measurements which is very closely **related to the time-delay embeddings** we encountered **in Exercise 5**. Overall Rico-Martínez et al. leverage a 4-layer NN-architecture devised for time-discrete modelling by Lapedes and Farber, which can be seen in Fig. 3. This architecture showed accurate short-term prediction and attractors close to the real data. However, the predicted bifurcations did not perform well.

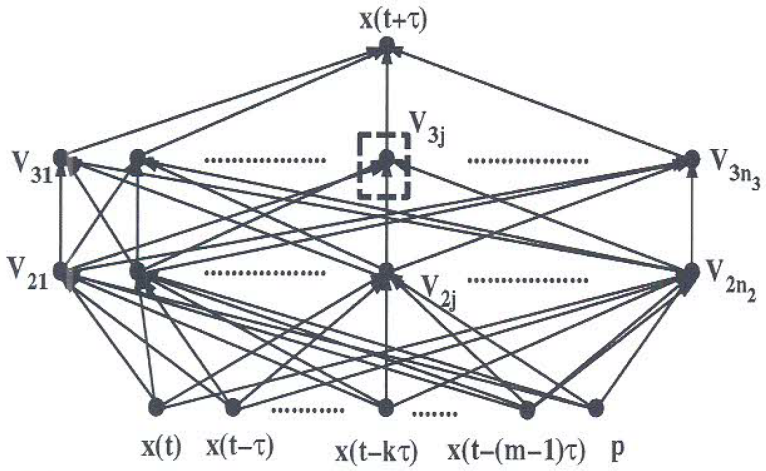


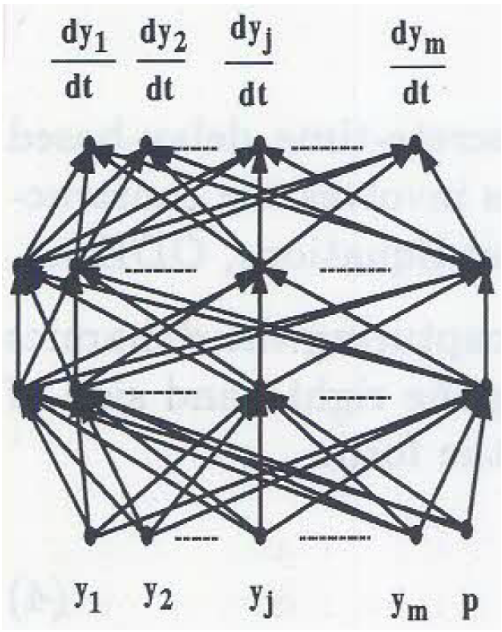
Figure 3: Discrete-time delay-based NN-architecture from [3]

In a second step, Rico-Martínez et al. show the **continuous-time modelling** approach which now searches directly for a model f to approximate:

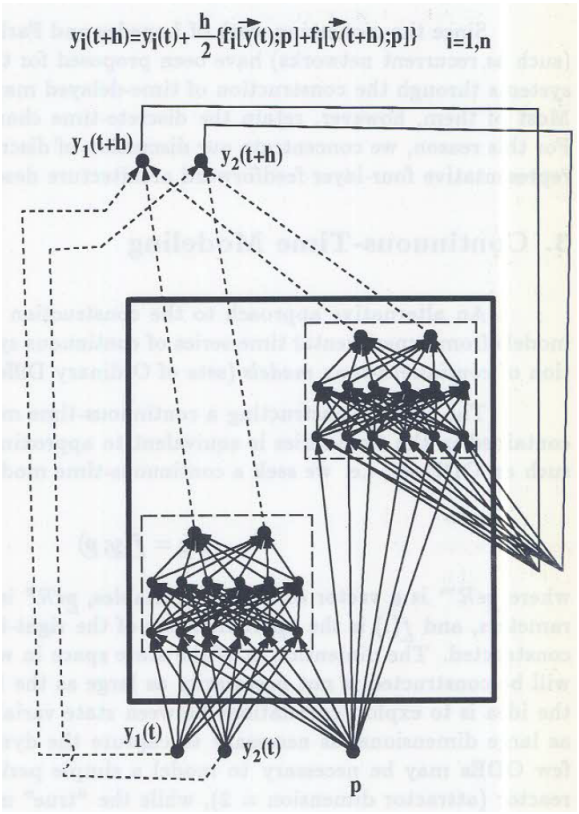
$$\dot{y} = f(y, p)$$

where f estimates the right-hand side of the underlying ODEs of the system, cp. Eq. 4 in [3]. In contrast, the dimension of the model y can be smaller than the actual data state space since correlations in the dimensions can be exploited within the ODEs. Again, the training of the NN f is integrated with numerical integrators, such as Runge-Kutta-methods, but the long-term prediction of the dynamic behaviour is now obtained by integrating the approximated ODEs instead of repeatedly applying a short-time mapping as in the discrete-time model. The standard block devised by the authors is again a 4-layer feed-forward NN which takes a state variable y at time t as input along the parameter settings p and outputs the right-hand side of the ODE $\dot{y}$. The NN-architecture as well as its embedding in an implicit trapezoidal integration rule can be seen in Fig. 4. The implicit integration rule also results in a recurrent network architecture in this case. As in [1], the training is based on a Least-Squares optimization of the predicted & real time-series values.

Figure 4: Continuous-time modelling in [3]



(a) NN-architecture of the continuous-time identification model



(b) NN-model integrated with an implicit trapezoidal rule

A new approach in the paper at hand concerns the **Pre-Processing** in the case where data is only available from a small subset or a single state variable of the experimental system. Here, the goal is to "reconstruct the phase space" [3] with a high enough dimension. This would be possible through a time-delay-based embedding with numerical differentiation but is sensitive to noise & measurement errors and therefore not desirable. Instead Principal Component Analysis based on NNs, called **"Nonlinear Principal Component Analysis"**, is used in an NN-architecture visualised in Fig. 5. While the bottleneck layer has linear activations, the encoder & decoder use non-linear sigmoidal activation functions. Furthermore, the networks goal is to approximate it's inputs which is in combination with this architecture **today better known as an Auto-Encoder**. For this, the authors use segments of the time series as input with a time θ in between and then slide a time-window interval over the data for new data points. This interval between two data points is later used as the integrator time-step h. The encoding of a data point in the bottleneck layer, or **latent space**, is the searched principal component and later used as input in the continuous-time modelling NN from the previous section. Sometimes there can be consistency problems in overlapping regions when inverting the time-series from NLPC-space to real space again.

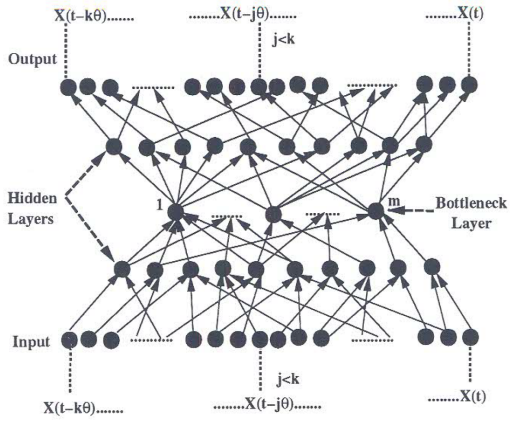


Figure 5: Non-Linear Principal Component NN-architecture from [3]

To prove the enhanced capabilities of continuous-time models over discrete-time delay-based models, Rico-Martínez et al. conduct two case studies. The **first case study** uses data of a **"heterogeneous catalytic oxidation of CO on Pt(110) single crystals"** [3] which shows a wide variety of oscillations & spatio-temporal pattern formation. In this case, the experimental data is images of spatio-temporal patterns for one fixed parameter setting. Before using a continuous-time model for identification, the images are pre-processed via "Proper Orthogonal Decomposition" (POD) [3] to decouple spatial & temporal evolution and achieve data reduction. This technique uses the normalised eigenvectors ϕ_i of a 2-point-correlation matrix and disregards the smallest eigenvalues which correspond to the probability of occurrence in the data. Here, the first 4 eigenvectors yielded good results meaning that the dynamic behaviour exists on a low-dimensional subspace of the initial 256x256 = 65536-dimensional phase space of the image data. Projection of the attractors in the first 4 coefficients shows noisy but periodic limit cycles as expected.
By **Taken's theorem, which we also saw in Exercise 5**, it is evident that the limit cycles, i.e., 1D-closed curves, only need 3 embedding dimensions. Therefore, the NLPC-procedure explained before is applied with the 4 eigenvectors at a time-step t & a time $t - \tau$, i.e., an 8D-vector, as input and a 3-dimensional bottleneck layer to obtain the three embedding dimensions. This yielded a much smoother projection of the experimental attractors compared to the POD-results.
Finally, the 3 non-linear principal component (NLPC) time-series are taken as input to a continuous-time NN which uses explicit 4th-order Runge-Kutta method as the numerical integrator. The results show that the long-term (attractor) as well as the short-term prediction of the model are very accurate. Moreover, a comparison of the prediction in NLPC-space & real image space show that the model yields good results. The predicted image was reconstructed for the comparison by first using the Decoder and afterwards the POD-eigenvectors. When compared to the prediction of a discrete-time delay-based model it was evident that physically inconsistent "bumps" in the discrete prediction could be avoided. Furthermore, the attractors were quite accurate but problems with bifurcations and varying parameter settings were solved by the continuous-time approach [3].

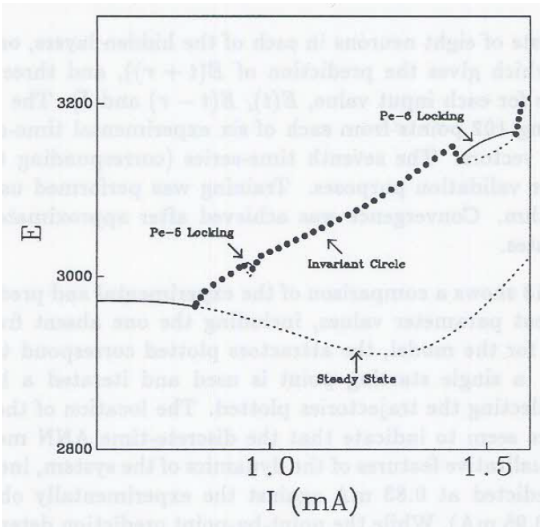
A **second case study** on **"Electrochemical Oxidation of H2"** [3] also included varying operating parameters and therefore several attractors as well as transitions between them. The operating parameter for this specific time-series data is the **applied current I**, where 7 different settings with oscillatory behaviour were observed with a Hopf-bifurcation at $I = 0.95$ mA. The goal of this case study is then to successfully detect the Hopf-bifurcation and the growth of the limit cycle amplitude with growing current. Firstly, a discrete-time model of the following form is devised:

$$E(t + \tau) = F(E(t), E(t - \tau), I)$$

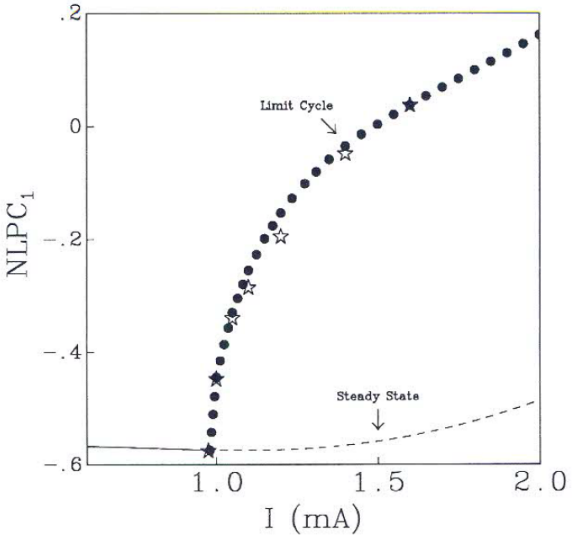
where F is a 4-layer feed-forward NN, cp. Eq. 9 in [3]. Rico-Martínez et al. train this model on 6 time-series and use 1 for validation. The results show that the model successfully captures qualitative dynamic features such as the location & appearance of the attractors and a rather accurate position of the Hopf-bifurcation in the system. However, the point-by-point prediction deteriorates quite strongly and the model also predicts impossible features such as **attracting invariant circles** instead of the limit cycles seen in ODEs. While a continuous-time oscillation visits every point on the limit cycle, the discrete map visits only a finite number of points. Beyond this property, the appearance of **"phase locks"** is observed which are a finite number of points

instead of a closed curve as the attractor prediction. The phase locks appear with a **separate saddle-node bifurcation** composed of stable & unstable periodic points which is **not part of the expected underlying dynamical system behaviour**. Therefore, the discrete-time model is not suited to predict the bifurcations & instabilities of the system, cp. Fig. 6a.
Another elaborated challenge is the **non-invertibility of discrete-time NNs** due to their non-linear activations. In principle, any non-invertible NN is inconsistent with continuous-time dynamical systems, but if only 1 possible previous state is physically relevant it might be "valid in a restricted region of the phase space" [3]. Rico-Martínez et al. formalise this notion of non-invertibility with Jacobians & points where its determinant vanishes. In the following, we only want to summarize a small example: If multiple pre-images of a state exist but only one of them is on the attractor itself we might restrict the pre-images to a close neighbourhood of the attractor and classify the map as uniquely invertible. For the case study at hand the discrete-time model is free of non-invertibilities in the parameter range of the training data. However, other ranges of the parameter I show these non-invertibilities and therefore a significantly reduced prediction quality. Furthermore, it is possible that the **interaction** of the **non-invertibilities & the previously discussed phase-locks** in the dynamic behaviour lead to **self-intersections** of the predicted attractor which is a physically impossible behaviour. For this reason, discrete-time-based models are not favourable since they can even predict behaviour impossible in continuous-time systems.
Finally, the authors test a **continuous-time model for Case Study II** based on pre-processing with an NLPC-NN with a 2D-bottleneck. The overall framework is the same as in the first case study but now with a 20-dimensional input vector and again 6 of the 7 time-series used for training the model. The NN takes 3 inputs – the 2 non-linear principal components & the current setting of the parameter I – and uses an implicit trapezoidal rule, i.e., implicit 2nd-order Runge-Kutta, as the numerical integrator. Once the NN is trained it is also possible to use any other integrator or time-step available. The results of the long-term prediction including the growing amplitude of the limit cycle are very good. Furthermore, the prediction of the bifurcation diagram is incredibly accurate, cp. Fig. 6b, and shows no physical inconsistencies such as the saddle-node bifurcation predicted with the discrete-time model.

Figure 6: Predicted Bifurcation Diagrams for Case Study II in [3]



(a) Bifurcation diagram prediction of discrete-time model (incl. phase-locks)



(b) Bifurcation diagram prediction of continuous-time model

In summary, Rico-Martínez et al. showed the "long-term dynamics approach" [3] for spatio-temporal behaviour of non-linear systems, its dependence on operating parameters & NN-models for their identification. Several problems with discrete-time delay-based models were shown, especially the phase-locking phenomena & non-invertibility in Case Study II. These led to the prediction of spurious transitions & even physically impossible attractors. They showed that continuous-time NN-approaches perform much better for experimental data from two oscillating chemical systems. These revised models could ultimately also improve control schemes & responses for such non-linear dynamical systems.

Own Conclusion
The focus of the second publication by Rico-Martínez et al., underlying our implementation, is on the evaluation of the Black-Box approach. In our final project we want to reproduce these successful results. While a few parts of the paper are of a more theoretic nature, e.g., the non-invertibility of NNs & phase-locking in the second case study, it shows that the NN-approach is capable of approximating the non-linear dynamical system.

Considering a first look at current trends, we can again see the computation performance limitations the authors had to manage in the 1990s. Therefore, all presented models use a rather small number of neurons and the training datasets are limited in size. A second observation is the predominant use of **tanh-activation functions** which are nowadays not as popular due to known issues with **vanishing gradients** for high & low input values. In newer publications we would therefore expect the use of different activation functions & much larger networks. Again, architectures that are well-known today can be seen in their first versions, like the Non-Linear Principal Component Analysis NN. The same concept of **Auto-Encoders** is very popular today and a widely used method to find latent space representations, of course with much deeper architectures in place. We would therefore expect that these first versions are replaced with their modern equivalents in current publications.

1.2 Other literature from Rico-Martínez et al. (1992 - 2000)

Along the first two publications in the previous chapter, Rico-Martínez et al. published several other papers in this research direction. In the following, we want to give a short overview of three provided publications from that time period and their concepts.

1.2.1 Rico-Martínez 1992 [2]

Short Summary
This early paper by Rico-Martínez et al. performs the groundwork of the continuous-time models we discussed earlier. After examining discrete-time NN-based models for short-term prediction again, they find that these models are unable to reproduce proper long-term behaviour & bifurcations. One possible explored solution is the usage of Poincaré maps of the data which could help with periodic and temporally complex systems. However, they find that these maps still do not correctly predict the bifurcations. The main contribution is the framework combining pre-processing by an NLPC-NN with another NN for system identification embedded in a numerical integrator. This continuous-time model successfully predicts long- & short-term behaviour as well as the bifurcations in the system which Poincaré & discrete-time models did not. The example process is the "potentiostatic electrodissolution of Cu in phsophoric acid solution" [2] where the applied potential is characterized as a varied operating parameter. The data pre-processing, i.e., finding a lower-dimensional embedding & rejecting noise without loosing the underlying dynamical structure, was an achieved with an NLPC-NN, as similarly shown in previous sections.

Especially interesting parts
While concepts like discrete-time models & NLPC-NNs were discussed extensively already, we want to elaborate three interesting areas of this publication. First of all, no underlying chemically-based kinetic equations are known for the **chosen experiment**. However, the experimental data shows a wide range of dynamic behaviour, including steady, periodic & chaotic regions and also period doubling, with bifurcations between them when varying the applied current. The details of data collection are specified in the paper but all time-series were retrieved after initial transients faded out. The NN successfully extrapolates the stability of attractors but for quantification more data including these initial transients would be required.
The concept of **Poincaré maps** is used for transforming the continuous trajectories to discrete maps which should improve the performance of a time-discrete model. A reasonable three-dimensional embedding space is found in this case with two-dimensional Poincaré sections. Afterwards, the now discretised data can be fitted with a discrete model.
Lastly, regarding the **continuous-time approach** most concepts were explained earlier, although the paper provides a good description of the framework including training of the NN-architecture. Interestingly, the authors chose an **explicit 4th-order Runge-Kutta-integrator** in this approach, which can be seen in Fig. 7. While an explicit method would fail if the underlying ODE was stiff, it worked in this case leaving the assumption that the unknown chemical kinetic equations are not stiff. However, an implicit integrator for stiff

ODEs would be much more expensive computationally.

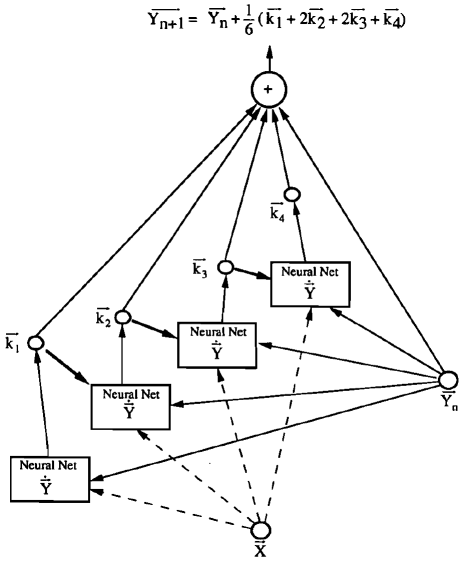


Figure 7: NN-based model embedded in explicit 4th-order Runge-Kutta integrator from [2]

Own Conclusion
This work by Rico-Martínez is the initial paper proposing the black-box approach and as such the basis for its enhancements in 1994 [1] and the two further case studies in 1995 [3]. Compared to the two examples in [3] the experimental system is much more complex regarding the bifurcations and different dynamical behaviour. The higher order but explicit nature of the integrator shows that recurrent training ideas are only integrated in later publications. For our implementation later on, the specific explanation of embedding a NN into a 4th-order Runge-Kutta method is quite valuable.
Considering newer approaches, many of the described challenges, e.g., computational effort & need for recurrent training with implicit integrators, were either resolved shortly after (recurrent training in [1]) or in the long-term by technical advancements (computational power).

1.2.2 Rico-Martínez 1996 [4]

Short Summary
Pre-Processing of data with NLPC-NN is needed for successful time-continuous models as seen in earlier discussions. However, while linear Principal Component Analysis (PCA) has a "self-consistency property" [4], i.e., iterative application gives the same results or the principal component coordinates stay the same, NLPC does not have this property. Therefore, training an NLPC-NN on original data vs. on a previous NLPC-representation does not yield the same latent space representation, which they call the "self-consistency problem" [4]. The main contribution of this work is a modified method introducing self-consistency to NLPC-NNs and thereby reducing the representation error.

Especially interesting parts
The **illustration of self-consistency** is done with data from a Lorenz-attractor and 4 linear principal components vs. 3 non-linear principal components. Iterative application of standard linear PCA shows that the representation stays the same, i.e., linear PCA is self-consistent. However iterative application of an NLPC-NN, i.e., taking the bottleneck-layer embedding as input for the next NLPC-NN training, shows that the obtained 3D-embedding varies strongly.
For the **modification of the NN-architecture** the NLPC-NN is first formalised as an optimisation problem and the self-consistency condition is stated as well. The combination of these two formalisations yields a modified optimisation problem which can be realised as a NN with an Elman-type architecture & recurrent connections, cp. Fig. 8 from [4]. The weighting factor w can vary between 0 & 1, where 1 retrieves the original NLPC-NN-architecture and 0 ensures self-consistency but yields bad representation results.

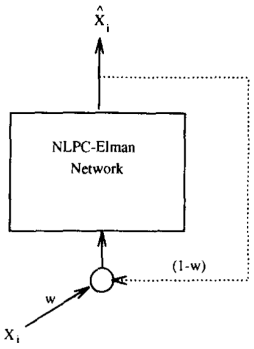


Figure 8: Modified NLPC-architecture with Elman-type architecture from [4]

Finally, **results** were obtained by training the network with standard recurrent backpropagation and a setting of $w = 0.5$. The reconstruction shows that the representation is still accurate with 3 NLPCs while the self-consistency was massively improved, although still not perfect. Several difficulties were encountered during training the recurrent network where more rigorous methods are stated as a possible solution.

Own Conclusion

This paper describes an improvement of a rather detailed step of the other approaches with a theoretic method. However since NLPC-NN were a basis for the success of all other previously discussed papers, the effect of its improvement on the performance of all other approaches is quite big. Even other tasks using NLPC-NN as pre-processing for their data have benefitted from this publication. In our following implementation we are not going to include it since we do not use NLPC-NNs for pre-processing our data.

Regarding current developments, Auto-Encoders yield good results without applying recurrent connections. This posts the interesting question if self-consistency was not as big of an issue or if there are other techniques in use to account for it, which we are not going to answer in this report. Furthermore, regarding the issues with recurrent training it can be assumed that modern training algorithms were optimised towards efficiency which solves the described challenges with it.

1.2.3 Rico-Martínez 2000 [5]

Short Summary

In this paper Rico-Martínez et al. elaborate the possible non-invertibility of time-discrete NN-based models, whereas tim-continuous dynamical systems are invertible by integrating the state back in time. Non-Invertibilities can result in quantitatively different model predictions & qualitatively inconsistent behaviour. Their main contribution is a study on "how non-invertibility arises", key analytical concepts involved & the related phenomenology [5]. Therefore, 2 examples, with experimental & purely computational data, for invalid NN-predictions are shown and 2 pathologies described in detail for each system. Finally, the remarks can be used to test the validity of NNs & as guidelines for collection of additional training data.

Especially interesting parts

First of all, the first example of an **electrochemical reaction** is interesting. Here, non-invertibilities within models from purely experimental data with noise are described, where the same data is used as in Case Study II of [3]. Therefore, the discussion of prediction results from an earlier section can be referenced here. The first pathology is the relation of **J_n-curves & multiple pre-images** which was previously summarised in [3] and is here shown in-depth. **Distorted attractors** are described as a second pathology which occurs if a predicted attractor crosses a J_0-curve. In this case excess pre-images in- & outside of the attractor appear in addition to the regular one on the attractor itself, which is physically impossible and leads to the distortion of the initially round attractor. In this example it is easy to blame the non-invertibilities on the noise in the data.

However, for the second case of data from a **surface reaction** this cannot be done since it is obtained fully computational, and therefore noise-free, from two given ODEs. The third pathology are **resonant (frequency-locked) solutions** which also occur in the previous example and were described in Case Study II of [3]. Like summarised in an earlier section the prediction exhibits phase-locks and corresponding saddle-node-bifurcations which are not part of the underlying dynamical system, cp. Case Study II of [3] again. The last pathology in this context are occuring **global bifurcations & self-intersections** that were already discussed for the previous example in the summary of [3]. Self-inconsistencies were there explained by the interaction of phase-locks &

non-invertibilities.
Finally for the **discussion**, the authors state that non-invertibilities & their pathologies in predicted behaviour cannot be explained by noise in the data, which is evident in the second example. Furthermore, the quality & validity of a NN-prediction could be quantified not only by classical accuracy measure but in addition by the absence of these pathologies. While non-invertibilities appear inherently in time-discrete models, they can be either displaced out of relevant regions or avoided by the use of time-continuous models.

Own Conclusion
The examples used in this work were already present in previous publications: the electrochemical oxidation of H2 was used in [3] and we extensively summarised the catalytic surface reaction in the context of [1]. While a first discussion of non-invertible time-discrete mappings was done in [3] this paper provides an extended discussion with a second example system. While only the "global bifurcation" & the "distorted attractors" [5] were really new pathologies, all non-invertible features were discussed in a new depth. Since our implementation already focuses on time-continuous models which do not exhibit non-invertibility challenges, we are not further leveraging this paper. However, it showed that discrete-time NN-based models come with significant disadvantages and endorsed the use of time-continuous models.

1.3 Newer approaches

This chapter shows three approaches from recent years, gives a short overview over their concepts and shows the connections to the work by Rico-Martínez et al. that we showed previously.

1.3.1 Meta-Learning ODE-integrators (Guo, 2021 [6])

Short Summary
In this approach, the authors describe the "meta-learning of numerical integrators" [6] as the integration of manually designed algorithm structures with data-driven adaption to specific tasks. The main concept here, is to learn ODE-solvers based on a Runge-Kutta architecture & apply Taylor-series-based regularisation to improve their performance. The NN-approximation and meta-learning in the framework then yield high-order integrators tuned on specific differential equation without further manual adjustment necessary, which is especially for high-order Runge-Kutta methods a big improvement because of the many coefficients. The performance is tested against a standard explicit 3rd-order Runge-Kutta integrator in general & specifically for the Van der Pol oscillator as well as the Brusselator system. The results finally reveal that it performs better than classic RK-methods for specific systems because the NN is capable of learning relevant ODE-properties.

Especially interesting points
The first point to further elaborate is the **Problem Formulation**, where also ODE-problems & explicit Runge-Kutta methods are discussed in-depth, which we summarised already, e.g., in the context of the explicit 4th-order method in [2]. The main question is if one is somehow able to obtain better integrators adapted to a specific family of tasks F [6] which is further formalised as an optimisation problem. Within the optimisation a loss-function comparing numeric & analytic integration result is combined with a regularisation term catering for a specific order of accuracy. The final goal is then to learn an Runge-Kutta-like method of a fixed order m with a NN-architecture.
The **RK-like NN-architecture** uses the basic Runge-Kutta-type method, i.e., a linear combination of function values & parameters k_i, with submodels N_i that are later merged with the k_i's to a combined model N_c. The whole framework is depicted in Fig. 9 taken from [6]. The learnable parameters are here the coefficients before each k_i which have to add up to 1 in the end. The parametric construction ensures consistency of the integrator, i.e., the local transaction error approaches 0 as the integrator time-step h approaches 0 [6]. The loss function to train the network architecture is chosen as a scaled squares loss function while the regulariser based on a Taylor-series expansion controls the desired order of convergence for the global truncation error.
The **performance tests** of the architecture were done on **specific ODEs**. First of all two RK-NN with $m = 3$, 1 with & 1 without the regulariser in place, were tested against a standard 3rd-order Runge-Kutta on linear & squared task families. In this case, the NN-architectures outperformed the standard 3rd-order Runge-Kutta for all time-steps h in the training range of $(0.01, 0.1)$ but the performance was only good for the trained task. For example, a NN-integrator trained on the linear task performed very poorly on the quadratic task. Significant further improvements were achieved by setting the regulariser to achieve accuracy orders even higher than the initial Runge-Kutta order m specified beforehand. Outperformance of standard Runge-Kutta was also observed when testing multi-dimensional task families like the Van der Pol-oscillator & the Brusselator, which are both

2-dimensional dynamical systems. Again, the performance of the NN-integrators outside of the training range of h varied quite strongly.

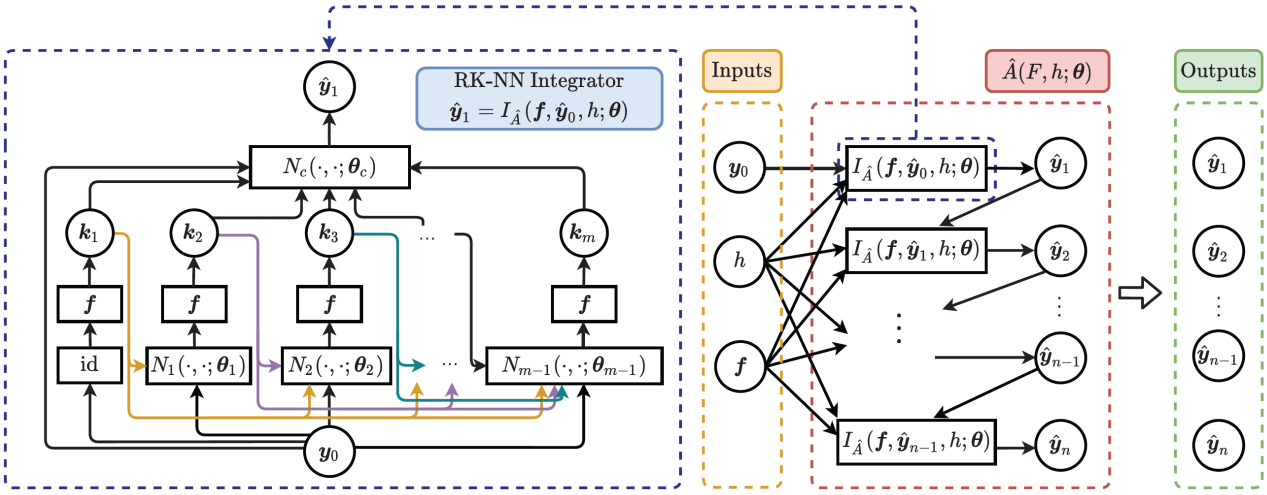


Figure 9: Runge-Kutta-like NN-architecture from [6]

Relation to other publications & Remarks

The presented framework can be seen as a complementary part to the previous work by Rico-Martínez et al., e.g., [3], we discussed: While Rico-Martínez uses a fixed numerical integration method and approximates the underlying ODEs of the system, Guo et al. work with fixed underlying ODEs and approximate a well-performing numerical integration method. While both use primarily Runge-Kutta-type integrators in this paper Guo et al. constrain themselves to explicit methods. In previous sections primarily implicit methods were used, cp. [1], to deal with stiffness and to achieve a good integration outcome needed for accurate approximation of the ODEs. So overall, the basic idea of embedding ODEs in an integrator & combining it with NNs is equal although the modern NN-frameworks, e.g., TensorFlow, & methods improve the performance significantly.
In summary, we think that this approach shows well how existing numerical integrators & algorithms can be greatly improved by including Machine Learning methods and data. The concept of constraining generalisation capability to a family or subset of tasks to improve performance within this subset is certainly interesting as well. Lastly, the question remains how the combination of previous approaches for identification of dynamical systems and the concept at hand would impact its performance. Here, one could for example examine if first learning an integrator for a suspected family of dynamical systems and afterwards identifying the specific ODEs from that family for an experimental system brings accuracy improvements in the identification.

1.3.2 Recursively Recurrent Neural Networks (Doncevic, 2024 [7])

Short Summary

The main topic of this paper is enhancing the approach to apply meta-learning to numerical algorithms as started in [6]. Therefore, the previous approach with Runge-Kutta NNs is expanded to a "recursively recurrent NN architecture (R2N2)" [7]. The forward pass is computed in 2 steps – the generation of information and its assembly to a solution. In the first step, the "subordinate, inner, iteration of the recurrent function" [7] starts at the current value of the second outer iterate and computes its values. Afterwards, the next outer iterate is calculated as a linear combination of the inner ones. The results are learned algorithms similar to Krylov & Newton-Krylov solvers for linear & non-linear equations and Runge-Kutta integrators for ODE-systems. Moreover, the framework could be further advanced for more general algorithms, e.g., based on Tayler-series expansions. The overall goal is described by the authors as the question if "meta-learning applied to (...) R2N2 can discover old & new algorithms personalized to certain problem classes" [7].

Especially interesting points

A first contribution is the **problem formulation for iterative algorithms**: Here, a generic definition of tasks for a solver is given as a function being equal to 0 at a solution $\tilde{x}$, e.g., for an IVP this means that the deviation

between a predicted solution $\tilde{x}$ & an analytical solution is zero. The problem solver is now a function mapping onto the space of possible solutions $\tilde{x}$, this map should then be learned by the framework. Iterative algorithms are formalised as adding a linear combination of certain v_i to a previous iterate x_k to obtain the next, outer, iterate x_{k+1}. The v_i are function evaluations at the current outer iterate x_k and their calculation is seen as the "inner iterate" [7]. The iterative algorithm then loops until x_{k+1} converges to a solution $\tilde{x}$.
One forward pass of the described **R2N2-architecture** equals one iteration of the numerical algorithm. The provided R2N2-superstructure, cp. Fig. 10 from [7], represents this algorithm using skip-connections from the previous x_k and recurrency by taking v_i as input for v_{i-1} as well. To obtain a series of states, X_1 to X_t, the forward pass is then applied multiple times, as seen in Fig. 11 from [7]. The relation to Krylov & Newton-Krylov solvers based on Krylov-subspaces is not further summarized here since we focus on ODE-problems in the following. For training & implementation the authors apply an MSE-loss but no specific regularizer like in [6]. Furthermore, frameworks like PyTorch are used for training with an Adam-optimizer for 25000 epochs and a train-test-split of 70/30.

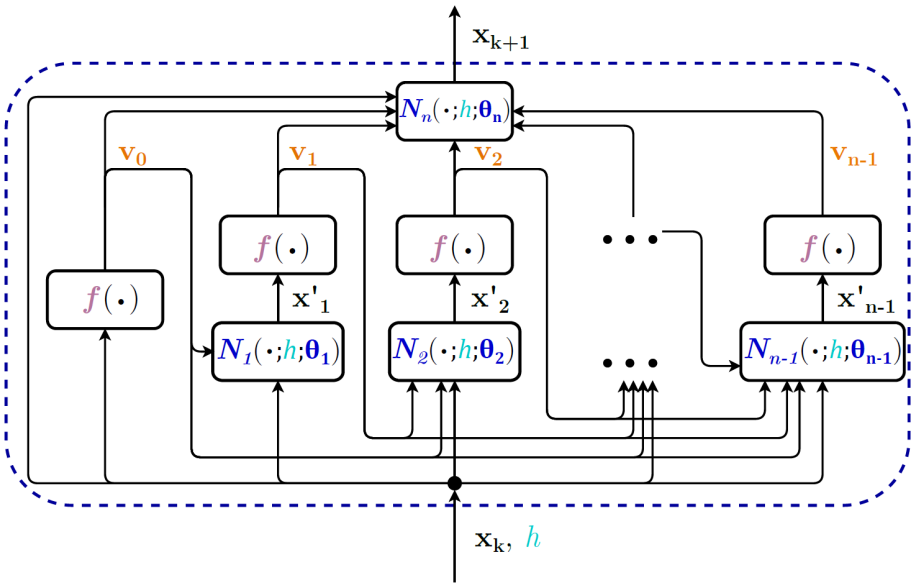


Figure 10: R2N2-superstructure from [7]

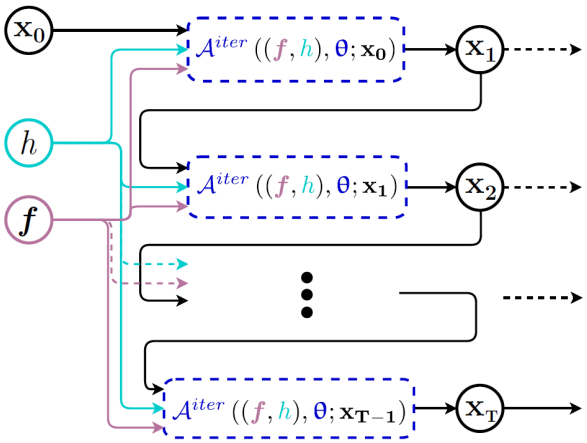


Figure 11: Iterative application of the R2N2-superstructure from [7]

The architecture returned good **results** for linear & non-linear equations as well as for IVPs focussed in our project. As an example again the Van der Pol oscillator was used and the NN-architecture showed similar results to [6] but without the need for specialised regularisation in R2N2. A performance comparison was based on an n-layer R2N2 vs. a standard n-th order Runge-Kutta method. For $n = 3$ the first three iterations performed better than standard Runge-Kutta over a range of integration steps h_i because R2N2 is able to learn task-specific coefficients. However, the accuracy dropped with the number of iterations increasing and after 5 iterations it performed worse than standard 3rd-order Runge-Kutta.

Relation to other publications & Remarks

The work by Rico-Martínez et al. is today recalled as "neural ODEs" and is stated as one of the motivations for this paper. The framework of Runge-Kutta with neural ODEs by Rico-Martínez is very similar to R2N2, except for the learnable part which is in this case the coefficients of the algorithm instead of the ODE itself. Both work with inner function evaluations (Rico-Martínez: the NN for ODEs) that are later combined in a weighted sum. This approach is however much more generalised compared to Rico-Martínez et al. and Guo [6] because it can not only solve IVPs but is also capable of learning algorithms for linear & non-linear equation solving. Again it would be interesting to combine the approach by Rico-Martínez, i.e., learning dynamical equations, with this algorithm approximation to a specific task to see if it improves performance on specific scenarios.

We think that this work is especially impressing because of its general capabilities to learn algorithms for IVPs. It shows again that good results can be obtained with modern NN-frameworks like PyTorch and large networks. This could also serve as possible future work for our project (not within the Practical but afterwards) since re-implementation & combination of approaches promises good results on real-life applications.

1.3.3 Model Order Reduction with Runge-Kutta NNs (Zhuang, 2021 [8])

Short Summary

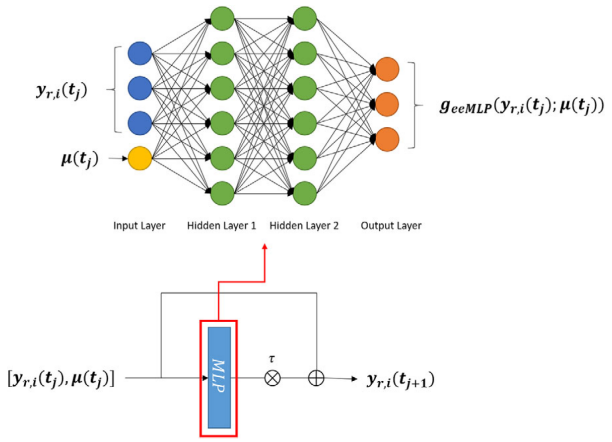
Zhuang et al. propose a framework for "Model Order Reduction", i.e., finding low-dimensional dynamical models independently of the numerical solver used for the full-dimensional ODEs [8]. For this, Machine Learning techniques & data are used to cope with non-linearities in the model. The approach is divided into two steps: Firstly a dimension reduction, e.g., with a projection-based method, and secondly a model reconstruction with a NN. The modifications proposed for both parts were then tested on three simulation examples. "Proper Orthogonal Decomposition" (POD) was used as the dimension reduction algorithm and further tested with constant & time-dependent input parameters. On the the other hand, three different NN-architectures served as the reconstruction method: a standard Multi-Layer Perceptron (MLP), an Explicit-Euler-NN and a Runge-Kutta-NN. The results showed that the latter outperformed the others because of the advantages of a higher-order integrator.

Especially interesting points

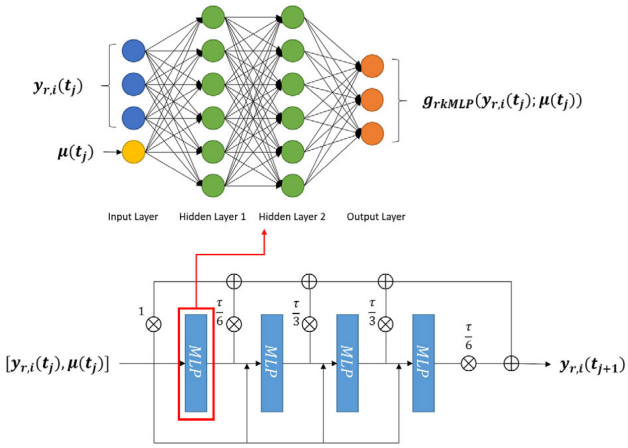
The **Proper Orthogonal Decomposition** is a variation of PCA for Model Order Reduction [8] and obtains a lower-dimensional ($dim = N_r$) representation from snapshots of the fully dimensional model ($dim = N$). The variations imposed by this paper are concerning the snapshots, i.e., data, of the full order model (FOM). On the one hand, they can be obtained with constant parameters μ ("static parameter sampling" – SPS) [8]. Alternatively, the parameters can be chosen from a function space, e.g., a sinusoidal function in this case, before obtaining the snapshots ("dynamical parameter sampling" – DPS) [8]. Afterwards, a Singular Value Decomposition on the matrix of snapshots is performed and disregarding the singular vectors with small singular values leads to the transformation of the data to a new basis.

The simplest **NN-architectures** tested in this paper is a standard MLP that maps from all current available information, e.g., previous states, etc., to the next state value y_{t+1}. This is conceptually related to the idea of discrete-time models summarized earlier in Rico-Martínez' works. Secondly, an Explicit-Euler NN learns the right-hand side of the ODEs in the reduced dimension state and embeds the approximated ODEs in an explicit Euler integrator, cp. Fig. 12a from [8]. For the Runge-Kutta NN the only difference is that an explicit 4th-order Runge-Kutta integrator is used where the number of neurons depends on the dimensionality of the reduced model, cp. Fig. 12b from [8]. Both NN-architectures with numerical integrators are very similar to the frameworks shown by Rico-Martínez et al. which were described earlier. Furthermore, all three NNs use ReLU-activations, an Adam-optimiser & L_2-regularisation.

Figure 12: Different NN-architectures in [8]



(a) Explicit-Euler NN-architecture from [8]



(b) Runge-Kutta NN-architecture from [8]

Performance results were obtained on three dynamical systems: "Heat Sink", "Gap radiation" & "Heat Exchanger", with specific ODEs (mostly thermal equations) & parameters for each one. The two different sampling-methods for POD displayed the same singular values and mostly similar reconstruction errors, apart from one case for each where it outperformed the other. This shows that a hybrid method sampling statically & dynamically is probably the best choice, while both methods still impact NN-training in a certain way. A first test of the NN-architectures was conducted with a fixed integrator step size h for the MLP & the RK-NN. The latter did not perform more accurately than the MLP which was even better for models with a large N_r. One possible explanation provided by the authors was that the RK-NN structure is deeper and might need more training data [8]. Secondly, the EE-NN & RK-NN were trained on varying step sizes. In this scenario, the RK-NN consistently outperformed the EE-NN which shows that training on a higher-order integrator improves the quality of approximated dynamics, especially if the input data is sparse [8].

Relation to other publications & Remarks

This approach concerns the same topics & ideas as the publications by Rico-Martínez that were summarized earlier. Zhuang et al. use deeper models and modern frameworks but constrain themselves to explicit numerical integrators like Rico-Martínez in his early works, cp. [2]. Within this paper, the focus additionally lies on Model Order Reduction but overall the same methods, like POD in [3], are used. Furthermore, an improvement for POD by different sampling methods is proposed. Another point is that discrete-time models do not seem to be in focus anymore since a standard MLP can be used instead. Additionally, more modern pre-processing methods are available. Another interesting development compared to Rico-Martínez is the use of Rectified Linear Units (ReLUs) as activations instead of sigmoidal functions, e.g., tanh. This is most certainly due to findings in the domain of Deep Learning and NN that suggest ReLUs as solutions to several training problems, e.g., vanishing gradients.

We think that this is a very valuable paper in the context of this project since it shows modern approaches for the same topics as Rico-Martínez. Additionally, there is much room for further studies such as the realisation of implicit methods of different orders. However, we want to focus on re-implementation of the papers by Rico-Martínez for now. In that context, it would also be interesting to test prediction capabilities for different dynamical states & bifurcations within this framework.

1.4 Conclusion

In this part of the report, we summarised the underlying approaches by Rico-Martínez et al. and showed some ambient publications by their group. Since we do not have access to the data from the experiments used in [1] & [3] we are going to train our NNs on data from a standard dynamical system – the Hopf-bifurcation normal form. Apart from this change, we will try to reproduce the positive results obtained by Rico-Martínez et al.: Successfully learning the dynamical system with NNs approximating the right-hand side of the ODEs and embedding them in fixed numerical integrator schemes for training and inference. Here, we also want to show that the frameworks are capable of learning the bifurcations evident in a system, in our case the Hopf

bifurcation at a certain parameter α. As for the numerical integrators, we chose a mixture of explicit & implicit Runge-Kutta methods of different orders to hopefully see different performance between them. In particular, we will implement an explicit Euler-method (explicit 1st-order RK), an implicit trapezoidal rule (implicit 2nd-order RK), cp. [1] and [3], & an explicit 4th-order Runge-Kutta, cp. [2]. More modern approaches were illustrated to give an overview of current developments and the relations between the work by Rico-Martínez et al. and their "descendants".
The report is structured as follows: In chapter 2 we are showing the implementation of NNs with explicit Euler & implicit trapzoidal rule, while in the third part the realisation with explicit 4th-order Runge-Kutta is covered. Task 4 illustrates the testing of the NNs on the Hopf-bifurcation normal form and in Task 5 we are going to retrieve the predicted bifurcation diagram of the system.

Report on task 2, Implementation of NN for Euler's method and Trapezoidal rule

2 Implementation of NN for Euler's method and Trapezoidal rule

In this task, the first part will be explaining **Euler's method** followed by a description and reconstruction of the neural network (NN) implemented by Rico-Martinez et al. in [3]. The second part will be taking this concept and applying it with the **trapezoidal rule** instead. To do that the definition has to be stated as well as another NN desgin concept.

We introduce two implementation approaches which deal differently with the integrator scheme. This interesting concept is due to our model implementations done by two different students. The second implementation was reused and slightly modified to fit RK1 and RK2. While the first one is thoroughly described in this chapter, the second approach is only briefly touched since the implementation was done by another and mainly for the RK4.

Finally, to train the created NN, training data is naturally needed. Since the original training data from Rico-Martinez et al. is not provided to us we **generated our own data** with the process described in the last part of this chapter.

2.1 Euler's method (Frist order Runge-Kutta, explicit)

The purpose of this part of the task is to solely extract relevant information from [3] to reconstruct the NN they used in 1995. The definition of Euler's method is quickly stated first followed then by the actual implementation approaches.

2.1.1 Definition

Euler's method is a straightforward numerical procedure used to approximate solutions to ordinary differential equations (ODEs). In the context of dynamical systems, it provides a means of estimating the evolution of a system over time. The method works by taking a known initial condition and iteratively applying a simple update rule to generate subsequent points. Specifically, given an initial value problem of the form $\frac{dy}{dt} = f(t, y)$ with an initial condition $y(t_0) = y_0$, Euler's method updates the solution in small steps Δt using the formula shown below in (1):

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n) \tag{1}$$

Despite its simplicity and ease of implementation, Euler's method is limited by its accuracy and stability, especially for stiff equations or when large step sizes are used. Nevertheless, it provides a foundational technique for understanding more advanced numerical methods in the study of dynamical systems.

2.1.2 NN Designs: first approach

(Corresponding notebook: `code/task2/approach1/RK1.pynb`)

There are two approaches of NN implementation. The first one is found in the folder `code/task2/approach1` and infuses the integrator scheme inside the actual NN. The architecture defines a *training block* and *head operation* where the first corresponds to the four layer feed forward (4L-FF) with trainable weights and the

latter correlating with the integrator scheme with fixed weights. The forward pass deals with adding the *head operation* at the end of the 4L-FF to generate the output layer.

Therefore, the general architecture of this NN summarized is as follows:

There is the training block (consisting of a 4L-FF) and a head operation (integrator scheme). According to [3], the 4L-FF has **2 hidden layers** with each **3 neurons**. The activation function used is **tanh** and since we are dealing with a FF-NN linear layers are stacked. The head operation for Euler's method is simply defined by multiplying the intermediate output of the training block with the fixed weights (here: `h/2`). Figure 13 shows a simple visualization of the complete NN architecture [1].

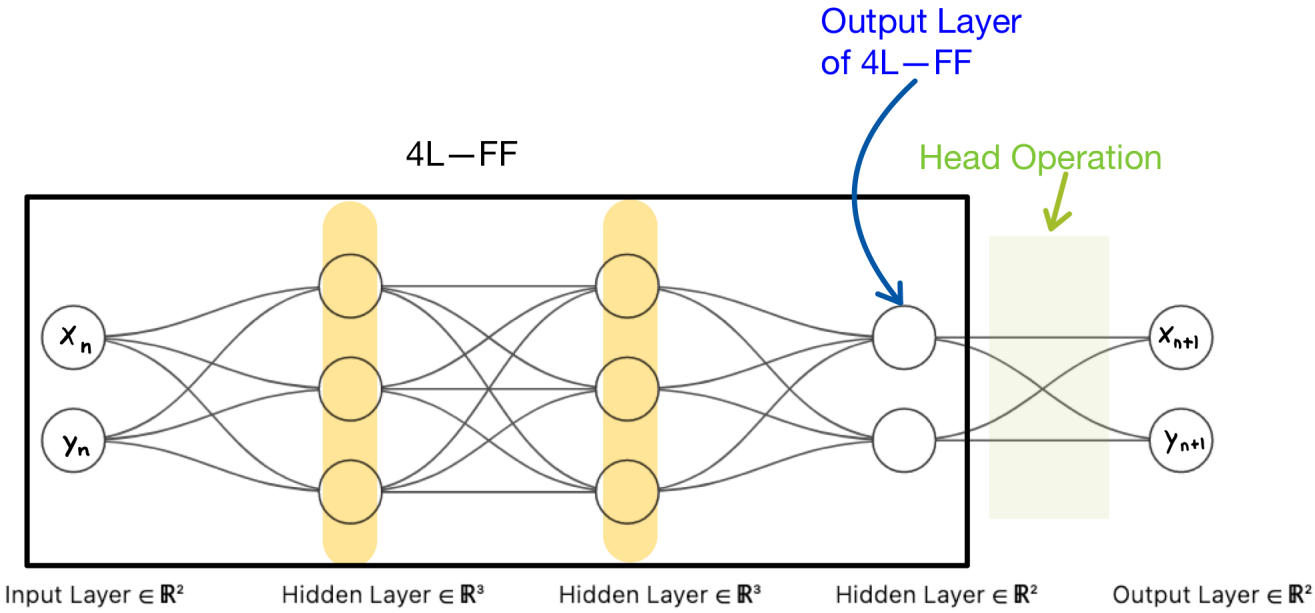


Figure 13: Colored EulerNN model visualization with (a) training block (black box, 4L-FF), (b) two hidden layers with tanh activation (yellow, 3 neurons each), and (c) head operation (green, rk1 step). The x and y values stand for the coordinates of the 2D state space.

One difficulty occurring with this approach was the loss calculation. With the integrator scheme (calculating the predicted next state) infused into the actual NN the loss function (and therefore affecting the weight updates) should compare the predicted next state with the actual next state from our data. For this the actual data needs to be present in the loss function. An error could have occurred in our training step implementation or in the data preprocessing which could not be fixed due to the restricted time frame of this project. With further time and better understanding/knowledge this problem could probably have been solved properly.

The model is found in `code/task2/approach1/nn_models.py`.

One note that is immediately recognized is that the age of the paper is evident. The NN consisting a hidden size of only 6 neurons resulting in a rather small and shallow network. Compared to nowadays where NNs easily built up layers in a much larger scale being it in numbers of neurons or layers. This could be naturally the result of higher computation power (and resulting computation time) in the modern era compared to roughly 40 years ago. Furthermore, the activation function **tanh** is considered outdated now since better performing activation functions have been found such as variants of ReLU which have advantaging properties like zero-centering and more.

The fact that the network here is very small and using a rather sub optimal way of adding non-linearity is a possible inidcation why this network performs rather unwell compared to our other implementations. Further discussion on the evaluation and testing of the NN implementations is found in the later chapters.

[1] generated with the help of a NN-architecture schematic tool: `http://alexlenail.me/NN-SVG/index.html`

2.1.3 NN Designs: second approach

(Corresponding notebook: `code/task2/approach2/RK1_2.pynb`)

The second implementation approach uses the concept of moving the integrator scheme outside the actual NN architecture and setting it place inside the training process. This way the calculation comparison of predicted next states with actual next states can be done easier and result in better performances (see chapter 4). To briefly describe the second approach: In the training process, the NN goes through a forward pass and takes its output into a method which predicts the next state through the integrator scheme. Then the loss is calculated and the weights are updated. The method for the integrator scheme can be seen in `rk1_step()` in `code/task2/approach2/model.py`.

2.2 Bonus: Trapezoidal rule (Second order Runge-Kutta, implicit)

After reconstructing the NN from Rico-Martinez et al. with Euler's method, we want to test the performance on a Runge-Kutta method of second order (RK2): the **trapezoidal rule** before the RK4 is implemented and thoroughly tested.

2.2.1 Definition

The trapezoidal rule is a numerical method used to approximate solutions to ordinary differential equations (ODEs), offering improved accuracy over simpler methods like Euler's method. In the context of dynamical systems, it provides a way to estimate the system's evolution by averaging the slopes at the beginning and end of each time step. The formula for trapezoidal rule (RK2) is shown below:

$$k_1 = f(t_n, y_n) \tag{2}$$

$$k_2 = f(t_n + \frac{dt}{2}, y_n + \frac{dt}{2} * k_1) \tag{3}$$

$$y_{n+1} = y_n + \frac{dt}{2} * (k_1 + k_2) \tag{4}$$

This involves solving an implicit equation, as y_{n+1} appears on both sides. The method requires either an iterative approach or solving a system of equations to find y_{n+1}. The trapezoidal rule is a form of the implicit Runge-Kutta method and is known for its increased stability and accuracy compared to explicit methods.

2.2.2 RK2: First approach (implicit)

(Corresponding notebook: `code/task2/approach1/RK2.pynb`)

The first approach corresponds with the same concepts as described above with the only dissimilarities of an additional training block and a different implementation of the head operation. The additional training block consisting of another 4L-FF takes in a recurrent connection from the output of the whole NN (implicit). Figure 14 shows a visualization of the NN architecture by modifying (coloring) the model found in [3].

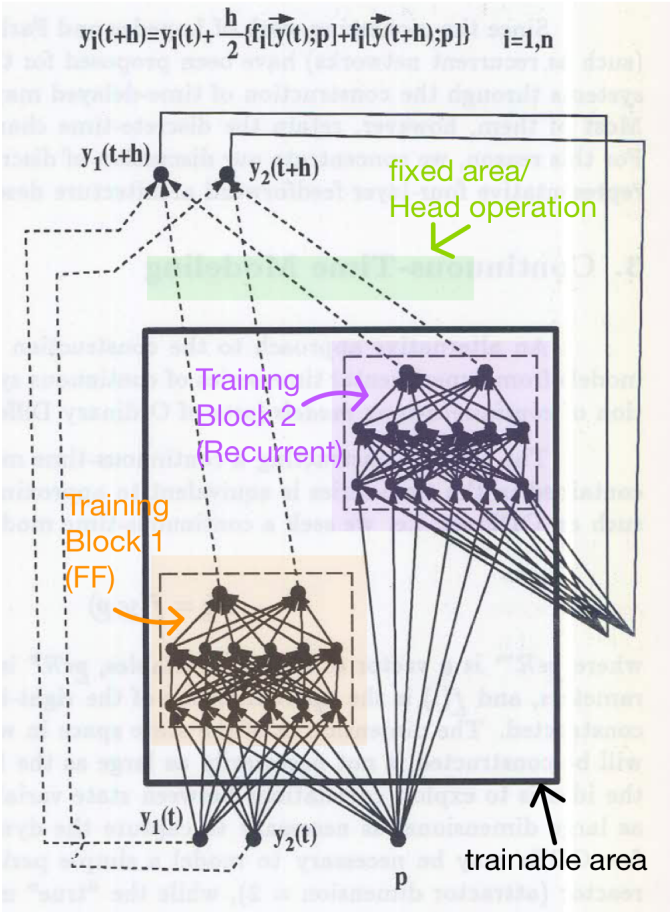


Figure 14: Colored TrapezoidalNN model visualization with (a) training block 1 (orange, 4L-FF), (b) training block two (purple, 4L-Recurrent), and (c) head operation (green, rk2 step).

2.2.3 RK2: Second approach

(Corresponding notebook: `code/task2/approach2/RK1_2.pynb`)

Identical to the description before, the second approach extracts the integrator scheme to the training step making the loss calculation easier to compute. The `rk2_step()` method in the `code/task2/approach2/model.py` implements the **trapezoidal rule** shown in (4).

The second approach will be applied to the rest of this project and a more detailed description about this approach is found in the next chapter (task 4).

2.3 Data Generation

The script `gen_data.py` (found in `/code/task4/utilities`) is run to generate data with different parameter values. The given script takes in different arguments such as α, initial state values, and value ranges to create a data set of two-dimensional states. The columns are defined with the first corresponding to x value and the second to y value with the first row being the initial state. The first part in the notebooks of our first approach automated the data generation given certain parameters (e.g. see `code/task2/approach1/RK1.pynb -> Create Training Data`).

Report on task 3, Implementation of NN for Runge-Kutta method

3 Predicting Bifurcation Behavior in Dynamical Systems Using Runge-Kutta 4th order method

Objective The objective of this experiment is to train a neural network (NN) model capable of predicting the evolution of a 2D state in a dynamical system given the system's parameter (α). The goal is to demonstrate the model's ability to capture bifurcation behavior by:

1. Training the model on datasets with different fixed α values.
2. Testing the model on new α values to observe different dynamical behaviors and the bifurcation phenomenon.

3.1 Neural Network Design

- The neural network consists of 4 layers.
- Inputs to the network include a 2D state at time t_1 and a parameter value α.
- The output of the network is the 2D state at time $t_1 + \tau$.

3.2 Layers

1. **Input Layer**: Accepts a 2D state and parameter α.
2. **Hidden Layers**: Two fully connected layers with non-linear activation functions.
3. **Output Layer**: Produces the predicted 2D state at $t_1 + \tau$.

3.3 Data Generation

In our project, we aim to build a predictive model using time series data. The data sets are generated for different alpha values, which play a crucial role in the dynamics of the system we are studying. The following section details the process of data generation and loading for both training and testing phases.

The training data consists of multiple datasets, each corresponding to a different alpha value. The alphas used for training are [0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9, 1.1, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8], and the resulting tensors are concatenated to form the final training dataset.

The testing data is prepared in a similar manner, but with specific alpha values chosen to test different aspects of the model's performance. The chosen alpha values for testing are 0.4 (low alpha), 1.0 (bifurcation point), and 1.2 (high alpha).

- **Training Datasets**: Generated for a range of α values (e.g., from 0.1 to 1.0).
- **Testing Datasets**: Generated for specific α values representing low, bifurcation, and high points (e.g., 0.4, 1.0, 1.2).

3.4 Training Process

The goal of this section is to describe the training methodology employed for our predictive model, specifically detailing the implementation of the Runge-Kutta 4th Order (RK4) integration method and the overall training loop. The RK4 method is crucial for numerical integration, providing high accuracy for our time series predictions.

Runge-Kutta 4th Order (RK4) Integration Method The RK4 method is used to solve ordinary differential equations (ODEs) numerically. It is particularly useful due to its balance between accuracy and computational cost. The general formula for a single step of the RK4 method is given by:

$$
\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f\left(t_n + \frac{dt}{2}, y_n + \frac{dt}{2} \cdot k_1\right) \\
k_3 &= f\left(t_n + \frac{dt}{2}, y_n + \frac{dt}{2} \cdot k_2\right) \\
k_4 &= f\left(t_n + dt, y_n + dt \cdot k_3\right) \\
y_{n+1} &= y_n + \frac{dt}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)
\end{aligned} \tag{5}
$$

The training loop involves iterating over the training data, using the RK4 method to predict the next state, and updating the model parameters based on the loss between the predicted and actual next states, a pseudo code for the training process is given below:

Algorithm 1 Training Process

function TRAIN(model, optimizer, scheduler, X_{train}, y_{train}, X_{val}, y_{val}, epochs, dt)
    Set model to training mode
    for each epoch in epochs **do**
        Initialize epoch loss to 0
        for each training example in X_{train} **do**
            Reset gradients in optimizer
            Extract current state (x, y) and parameter alpha from the training example
            Form the state tensor with current state and alpha
            Predict the next state using `rk4_step` function with the model, state tensor, alpha, and dt
            Extract the actual next state from y_{train}
            Calculate loss between predicted next state and actual next state
            Perform backpropagation to compute gradients
            Update model parameters using optimizer
            Accumulate the loss for the current epoch
            Update progress bar with current average loss per example
        end for
        Adjust learning rate using scheduler
        if epoch % 5 == 0 **then** ▷ Every 5 epochs, validate the model
            Predict next states for each example in X_{val}
            Calculate validation loss between predicted and actual next states in y_{val}
            Print training and validation losses for the current epoch
        end if
    end for
end function

1. **Dataset Preparation**: Generate synthetic datasets simulating the dynamical system for a variety of α values.
2. **Loss Function**: Use a suitable loss function to measure the prediction error, such as Mean Squared Error (MSE).
3. **Optimization**: Train the model using gradient descent-based optimization techniques.
4. **Training**: Train the model on the generated datasets to learn the dynamics of the system across different α values.

3.5 Testing Process

This section describes the testing methodology employed to evaluate the performance of our predictive model. A key aspect of our testing approach is the use of the predicted state as the input for the next prediction, creating a sequential dependency that closely mimics real-world applications of time series forecasting.

Sequential Dependency and Its Importance The use of the predicted state as the input for the next prediction is a critical aspect of our testing methodology. This approach ensures that each prediction depends on the previous one, mirroring real-world scenarios where future states are influenced by prior predictions. This sequential dependency tests the model's robustness and ability to handle the propagation of prediction errors over time.

By evaluating the model in this manner, we can better understand its performance in practical applications, where the accuracy of each prediction affects subsequent predictions. This testing strategy provides a more realistic measure of the model's effectiveness and its potential impact when deployed in time series forecasting tasks.

1. **Dataset Preparation**: Prepare new datasets with α values that were not seen during training.
2. **Evaluation**: Test the model on these new datasets.
3. **Observation**: Evaluate the model's performance and observe the dynamical behaviors for low, bifurcation, and high α values.

Visualization

- Visualize the output over a series of time steps to demonstrate the actual dynamical behavior.
- Show that for different α values during inference, the model captures the bifurcation phenomenon.

Report on task 4, Testing of NNs on 2-dimensional dynamical system

---

4 Testing of NNs on 2-dimensional dynamical system

4.1 Hopf-Bifurcation Normal Form

In this task, we tested the architectures we implemented on a simple two-dimensional dynamical system. The goal is to evaluate the accuracy and performance of the networks in capturing the dynamics of such systems. The dynamical system we chose to use is a system in Hopf-Bifurcation Normal Form. The Hopf bifurcation is defined as the appearance or the disappearance of a periodic orbit through a local change in the stability properties of a fixed point. The system we used is defined as:

$$\begin{cases} \frac{dx}{dt} = \alpha x - \omega y - x(x^2 + y^2) \\ \frac{dy}{dt} = \omega x - \alpha y - y(x^2 + y^2) \end{cases},$$

or in polar coordinates:

$$\begin{cases} \frac{dr}{dt} = \alpha r - r^3 \\ \frac{d\theta}{dt} = \omega \end{cases}.$$

The system fits our task, since it is very simple. It's equilibrium point is at the origin, and it's stability is dependent only on the bifurcation parameter α, which can be seen clearly from the polar form of the system. The system's steady states can be found when setting $\frac{dr}{dt} = 0$:

$$\begin{aligned} 0 &= \alpha r - r^3 \\ 0 &= (\alpha - r^2)r. \end{aligned}$$

When $\alpha < 0$, the only solution to the equation is $r = 0$, which indicates that the system has a stable fixed point at the equilibrium. When $\alpha > 0$, there is a stable limit cycle with radius $r = \sqrt{\alpha}$. When $\alpha = 0$, the linear part, i.e. the Jacobian of the system, only has purely imaginary eigenvalues $\pm i\omega$ near the equilibrium

point, which is indicative of a Hopf bifurcation. Let us show this behavior. Let $f_1 = \alpha x - \omega y - x(x^2 + y^2)$, $f_2 = \omega x - \alpha y - y(x^2 + y^2)$. The Jacobian is thus:

$$\begin{aligned} J &= \begin{bmatrix} \frac{df_1}{dx} & \frac{df_1}{dy} \\ \frac{df_2}{dx} & \frac{df_2}{dy} \end{bmatrix} \\ &= \begin{bmatrix} \alpha - 3x^2 - y^2 & -\omega - 2xy \\ \omega - 2xy & \alpha - x^2 - 3y^2 \end{bmatrix} \end{aligned}$$

Set $\alpha = 0$, and since we are in the equilibrium point, set $x, y = 0$ as well.

$$J = \begin{bmatrix} 0 & -\omega \\ \omega & 0 \end{bmatrix}$$

Calculating the eigenvalues:

$$\begin{aligned} \det(J - \lambda I) &= 0 \\ (-\lambda)^2 - \omega(-\omega) &= 0 \\ \lambda^2 &= -\omega^2 \\ \lambda &= \pm i\omega \end{aligned}$$

Thus we have showed that when α increases from negative to positive, the fixed stable point changes into a stable limit cycle. In the next part, we attempt to capture the dynamics with the neural network architectures we constructed in the previous sections.

4.2 Testing the NNs

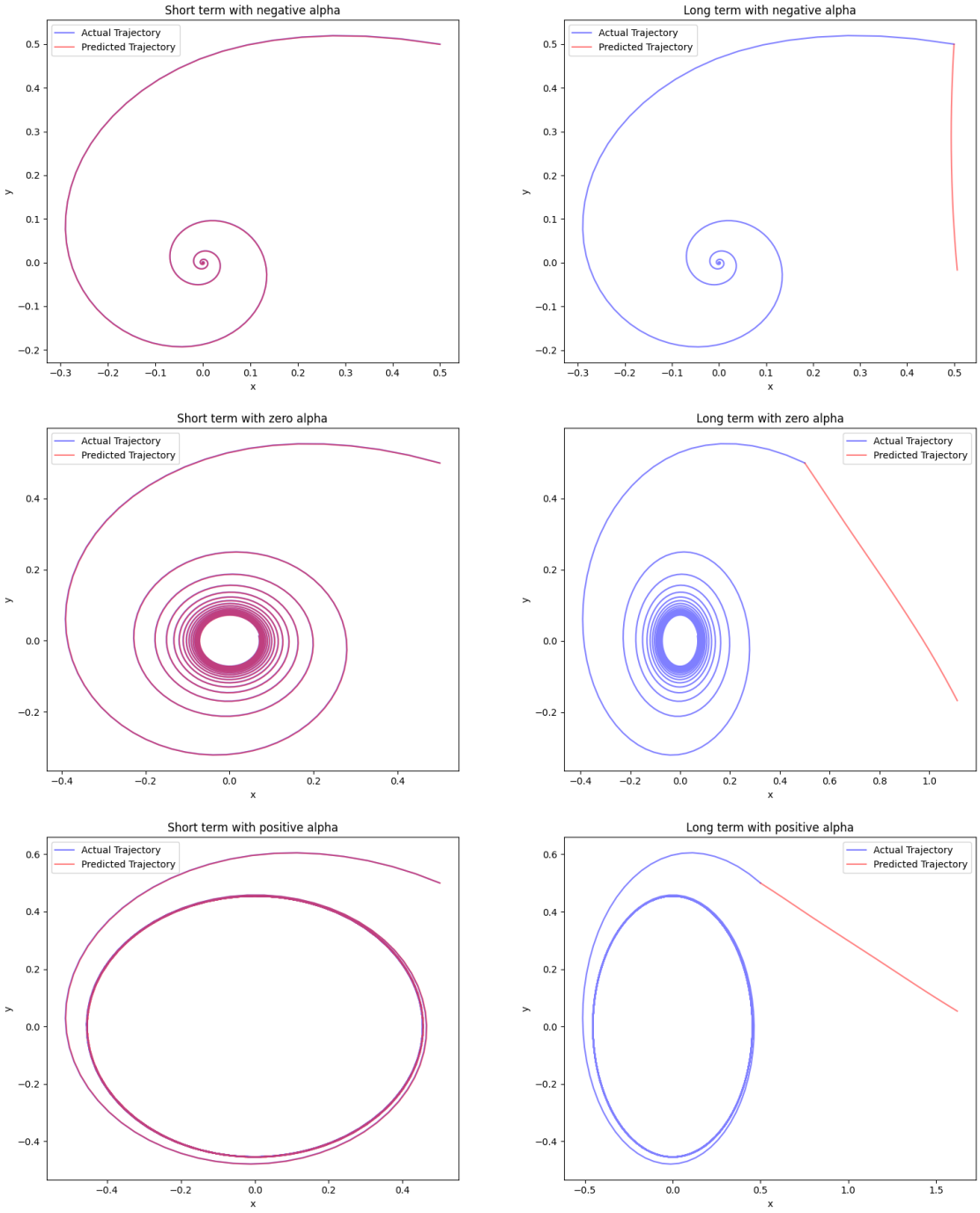
To be able to test the neural networks, we first needed data to train the networks. We used the `scipy.integrate.solve_ivp` method to generate data with our dynamical system and an initial state as the input. We used varying α-values, hoping to be able to capture the changing behavior of the system. Each dataset consists of 1000 datapoints, where a datapoint represents the state of the system (x, y) at a timestep t.

Then, we used the generated data to train our neural networks. Our accuracy metric was the visual correspondence to the actual evolution of the system state compared to the system state produced by our neural networks. We trained our networks with a fixed α. Our initial plan was to make the neural networks also learn the bifurcation parameter α, but we were not able to achieve that due to difficulties in the architecture construction and time constraints.

4.2.1 RK4-method

As stated previously, we trained each model on a dataset with a given α that shows the evolution of the system from a given initial state, hoping to be able to capture the system dynamics in the initial state. After training the models, we used the model to predict the system state's evolution with short and long term predictions.

Figure 15: Phase portraits



As can be seen from the figure 15, the short term predictions are really close to the actual evolution, but the long term predictions are completely off the mark. We think this is due to our problems we encountered when implementing the neural network. The network did not learn properly, i.e. the weights did not update in the training process and the training loss of the network stayed really high, even when increasing the number of epochs. Unfortunately, we were not able to fix these problems, and had to use subpar results in the report.

4.2.2 Trapezoidal method

In this section, we test the trapezoidal method implementation. Like in the previous section, we trained 3 models, one each with a dataset with negative, zero and positive *alpha*-values.

Figure 16: Phase portraits

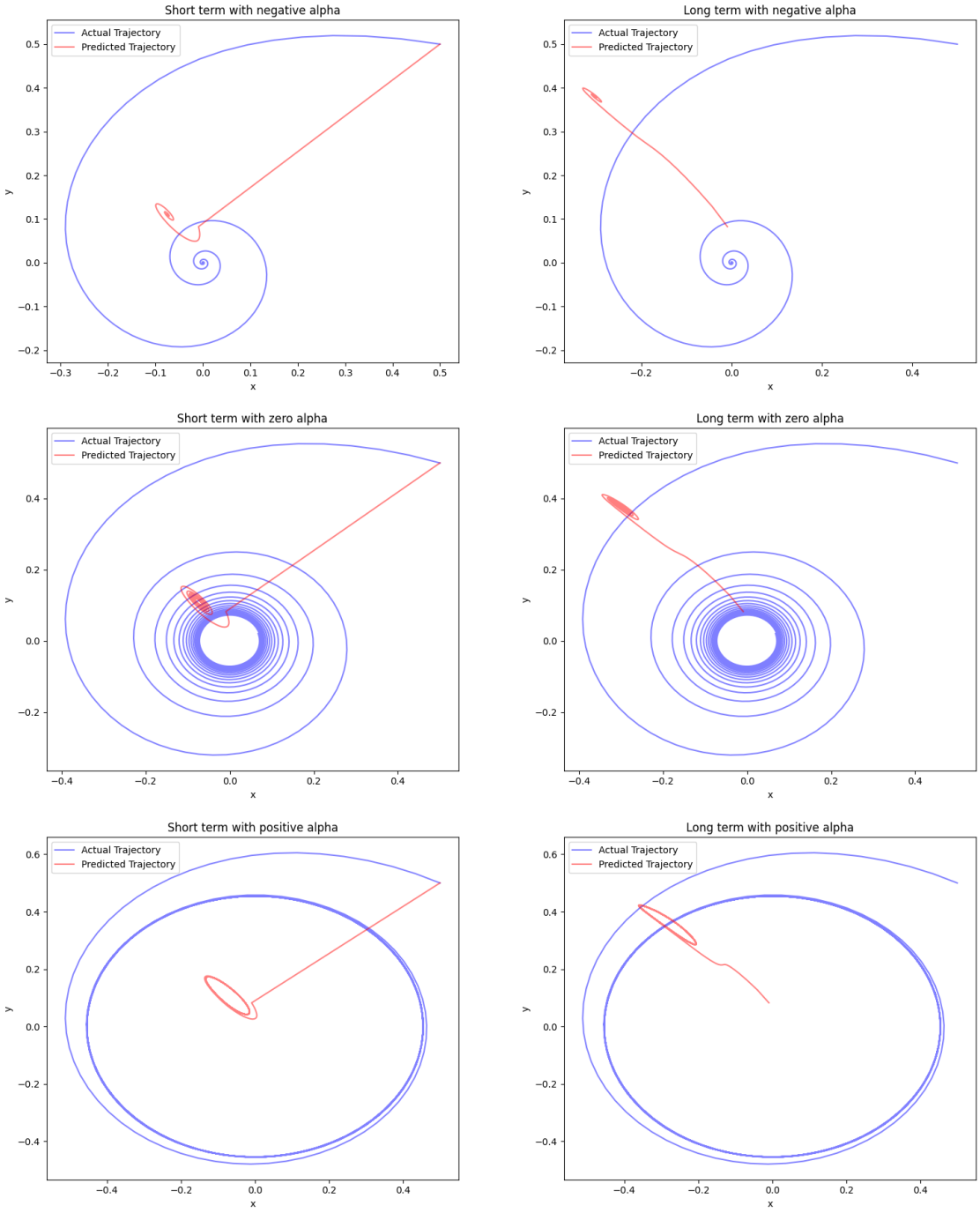


Figure 16 shows the results of the system the neural network learned during the training process. The results are even poorer than in the previous section, suggesting problems in the implementation of our architecture. The long term prediction plots are missing the plot from initial state (0.5, 0.5) to the actual start of the plot. One thing worth noting is that the network kind of learns the stability of the system, even though it does not follow the actual trajectory of the system at all. You can see in the plots that the network learns that the system has a stable fixed point when α is negative, even though the equilibrium point is not at the origin. When α is 0, the system has no stable points, and when α is negative, it kind of shows a stable cycle, even though skewed and malformed.

We had many hardships and difficulties with the implementation of the neural network architectures, and

the results show us, that we could not properly implement the architectures as they were described in the paper. With more careful planning and usage of time, we might've been able to achieve better results.

Report on task 5, Bifurcation Diagram

5 Constructing Bifurcation Diagrams Using Dynamics from Neural Networks

This task presents a study on constructing bifurcation diagrams using the dynamics extracted from neural networks. Bifurcation diagrams are instrumental in understanding the qualitative changes in the behavior of dynamical systems as parameters vary. By identifying critical points where system behavior transitions, these diagrams provide deep insights into the stability and transformation of the modeled systems. The use of neural networks for this purpose leverages their powerful capability to approximate complex functions and capture intricate dynamical patterns. This work aims to illustrate the process and highlight the significance of bifurcation analysis in the context of neural network dynamics.

This task outlines the methodology used to train neural networks for dynamical systems analysis, describes the process of constructing bifurcation diagrams, and discusses the insights gained from these diagrams regarding the stability and transitions of the modeled systems.

6 Methodology

To retrieve and visualize the predicted bifurcation diagram of the system using the trained neural network, we follow these steps:

6.1 Generate Data for a Range of Parameters

1. Create a grid of initial conditions and parameter values.
2. Use the trained model to predict the system's behavior over time for each initial condition and parameter value.

6.2 Plot the Bifurcation Diagram

1. Collect the long-term behavior (e.g., steady states or periodic orbits) of the system for each parameter value.
2. Plot these behaviors against the corresponding parameter values to create the bifurcation diagram.

The plotted bifurcation diagram can be found in 17.

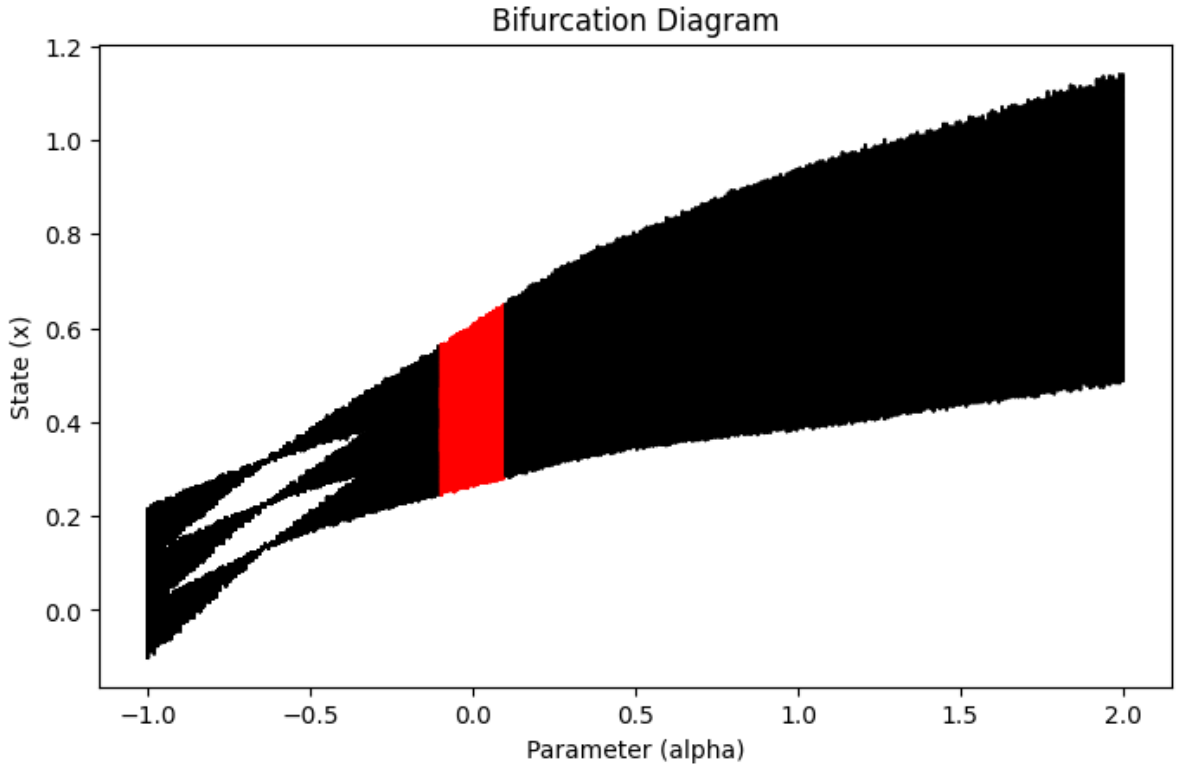


Figure 17

7 Conclusion and Observations

The bifurcation diagram obtained from the neural network dynamics provides significant insights into the behavior of the modeled system as the parameter α is varied. Below are the key observations and conclusions drawn from the analysis:

- **Single Stable State:** For $\alpha < -0.5$, the system exhibits a single stable state, indicating that the system's behavior is predictable and stable within this range of α.
- **Bifurcation Points:** As α increases from -0.5, the diagram shows the emergence of multiple branches. These branches represent bifurcations where the system transitions from a single state to multiple coexisting states. This behavior suggests increased complexity in the system's dynamics as α is varied.
- **Chaotic Behavior:** For $\alpha > 1.0$, the diagram becomes densely populated with states, signifying chaotic behavior. In this regime, the system's state x fluctuates unpredictably, and the dynamics become highly sensitive to initial conditions.
- **Critical Region:** The region highlighted in red around $\alpha = 0$ marks a significant transition in the system's dynamics. This critical region likely represents a parameter range where the system undergoes a major bifurcation or transition to chaos, indicating an area of interest for further detailed study.

In summary, the bifurcation diagram effectively visualizes how the qualitative behavior of the system evolves as the parameter α is varied. The transitions from stable states to periodic and chaotic behaviors provide a comprehensive understanding of the system's dynamics. These insights are crucial for identifying stability and predicting transitions in the modeled system.

References

[1] R. Rico-Marínez, J. S. Anderson, and I. G. Kevrekidis. Continuous-time nonlinear signal processing: A neural network based approach for gray box identification. *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pp. 596-605, 1994.

[2] R. Rico-Martínez, K. Krischer, I.G. Kevrekidis, M.C. Kube, and J.L. Hudson. Discrete- vs. Continuous-Time Nonlinear Signal Processing of Cu Electrodissolution Data. *Chemical Engineering Communications*, 188: 25-48, January 1992.

[3] R. Rico-Martínez, I. G. Kevrekidis, and K. Krischer. Nonlinear system identification using neural networks: dynamics and instabilities. *Neural Networks for Chemical Engineers*, 1995.

[4] R. Rico-Martínez, J. S. Anderson, and I. G. Kevrekidis. Self-Consistency in Neural Network-based NLPC Analysis with Applications to Time-Series Processing. *Computers & Chemical Engineering*, 20(supplement 2): S1089-S1094, 1996.

[5] R. Rico-Martínez, R. A. Adomaitis, and I. G. Kevrekidis. Noninvertibility in neural networks. *Computers & Chemical Engineering*, 24(11): 2417-2433, November 2000.

[6] Y. Guo, F. Dietrich, T. Bertalan, D. T. Doncevic, M. Dahmen, I. G. Kevrekidis, and Q. Li. Personalized Algorithm Generation: A Case Study in Meta-Learning ODE Integrators. *arXiv-preprint*: 2105.01303, May 4, 2021. Available under: https://arxiv.org/abs/2105.01303

[7] D. T. Doncevic, A. Mitsos, Y. Guo, Q. Li, F. Dietrich, M. Dahmen, and I. G. Kevrekidis. A Recursively Recurrent Neural Network (R2N2) Architecture for Learning Iterative Algorithms. *SIAM Journal on Scientific Computing*, 46(2): A719-A743, 2024.

[8] Q. Zhuang, J. M. Lorenzi, H.-J. Bungartz, and D. Hartmann. Model Order Reduction Based on Runge-Kutta Neural Networks. *Data-Centric Engineering*, 2(13): July 2021.