Generation and Identification of Dynamics in Ensembles of Co-evolving Modules

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Danil Koryakin

aus Krasnojarsk / Russland

Tübingen

2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	30.01.2026
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr. Martin V. Butz
2. Berichterstatter/-in:	Prof. Dr. Andreas Zell

Abstract

Recent research has focused considerably on modular structures for data modeling due to their inherent benefits, such as high performance and computational efficiency. These architectures are particularly suitable for modeling structured data, with each module dedicated to a specific data segment. While the primary emphasis of modular architectures has traditionally been on achieving high model accuracy, their potential to generate comprehensible data representations and integrate a priori knowledge offers significant opportunities for the efficient identification and understanding of the modeled data. Nevertheless, these capabilities have largely remained unexploited within the domain of dynamic data.

The primary objective of this thesis is to demonstrate the applicability of modular architectures for the identification of the underlying content in time series data. To this end, this work investigates the identification of hidden dynamic components within various types of time-dependent data using ensembles of co-evolving modules. The research employs semi-automatic modular decomposition to discern hidden dynamics in time-dependent data. This approach effectively integrates available a priori knowledge about the problem with automated module adjustments. To tackle the combinatorial complexity arising from a large number of potential dynamic combinations, this thesis introduces a novel population-based approach for identifying dynamics, based on the online synchronization of co-evolving module dynamics. Evolutionary algorithms and gradient-based methods were utilized to tune the internal dynamics of recurrent neural and parametric modules online. Evaluations involved diverse datasets, including compositional time series, chaotic attractors, and handwritten symbol curves.

The experimental results demonstrate the effectiveness of the proposed approach. The increased likelihood of successful synchronization observed with neural modules using smaller weights suggests the necessity of regularization techniques and scaled-down driving signals. The faster convergence exhibited by neural modules indicates that their inherent plasticity facilitates more effective tuning and underscores the critical role of balancing module compactness with accuracy to effectively address the plasticity-stability dilemma in module design. Furthermore, the reliable identification achieved through evolutionary tuning, despite its higher computational cost compared to gradient-based methods, highlights the importance of a priori knowledge in guiding the selection of the most suitable tuning approach. The consistent behavior of the method across diverse datasets suggests that the principles of co-evolving module dynamics are likely applicable to systems exhibiting inherent modularity, where distinct sub-processes significantly contribute to the overall system behavior.

Zusammenfassung

Die Forschung zu modularen Strukturen für die Datenmodellierung hat in letzter Zeit erheblich an Aufmerksamkeit gewonnen, insbesondere aufgrund ihrer Vorteile wie hoher Leistung und Recheneffizienz. Solche Strukturen eignen sich besonders für die Modellierung strukturierter Daten, wo jedes Modul einen spezifischen Teil der Daten abbilden kann. Während modulare Architekturen hauptsächlich zur Erzielung genauer Modelle eingesetzt wurden, bieten andere Vorteile der Modularität, darunter die Fähigkeit, verständliche Datenrepräsentationen zu entwickeln und Vorwissen über das Problem einzubeziehen, ein beträchtliches Potenzial für die effektive Identifizierung und das tiefere Verständnis der modellierten Daten. Dieses Potenzial blieb jedoch im Kontext dynamischer Daten bislang weitgehend unerforscht.

Das Hauptziel dieser Dissertation ist der Nachweis, dass modulare Architekturen zur Identifikation des Inhalts von Zeitreihen geeignet sind. Hierfür wird die Identifikation versteckter dynamischer Komponenten in zeitabhängigen Daten verschiedener Typen durch den Einsatz von Ensembles co-operierender Module untersucht. Die Analysen basieren dabei auf der semi-automatischen modularen Dekomposition zeitabhängiger Daten, wobei Vorwissen über das Problem effektiv mit automatischen Anpassungen der Dynamiken integriert wird. Um die hohe kombinatorische Komplexität zu bewältigen, die durch zahlreiche mögliche dynamische Kombinationen entsteht, wird ein innovativer populationsbasierter Ansatz vorgestellt. Dieser Ansatz beruht auf der Online-Synchronisation parallel ablaufender Teildynamiken und setzt dabei evolutionäre sowie gradientenbasierte Verfahren ein.

Die Experimente zeigen die Wirksamkeit des vorgeschlagenen Ansatzes. Die erhöhte Wahrscheinlichkeit erfolgreicher Synchronisationen, die bei neuronalen Modulen mit kleineren Gewichten beobachtet wurde, deutet auf die Notwendigkeit von Regularisierungstechniken hin. Die schnellere Konvergenz mit neuronalen Modulen zeigt, dass ihre Plastizität eine effektivere Anpassung ermöglicht und somit eine entscheidende Rolle beim Ausgleich von Kompaktheit und Genauigkeit im Kontext des Plastizitäts-Stabilitäts-Dilemmas im Modulentwurf spielt. Zudem unterstreicht die zuverlässige Identifikation, die durch evolutionäre Optimierung erreicht wird – trotz ihrer höheren Rechenkosten im Vergleich zu gradientenbasierten Methoden –, die Bedeutung von Vorwissen bei der Auswahl des effektivsten Optimierungsansatzes. Die beobachteten Ähnlichkeiten im Verhalten der Methode über verschiedene Datensätze hinweg legen nahe, dass die Prinzipien der parallel ablaufenden Moduldynamiken auf Systeme anwendbar sind, die eine innere Modularität aufweisen, bei der isolierte Teilprozesse zum Gesamtverhalten des Systems beitragen.

Acknowledgments

First, I would like to express my sincere gratitude to Prof. Dr. Martin V. Butz, who graciously agreed to be the scientific supervisor of my PhD. His suggestions were crucial in determining both the research area and direction. I appreciated his keen interest in my work and his willingness to make time for reading my progress reports and paper drafts, as well as his patience in correcting them, despite a demanding schedule, other research interests, and the other PhD students. Our fruitful discussions provided valuable insights and helped focus my investigations, leading to publishable results. His advice was very helpful in achieving high-quality papers.

Furthermore, I would like to thank Dr. Johannes Lohmann for his software package ESNJava, which was an excellent tool for my initial experiments and served as a solid basis for its subsequent expansion with the necessary additional features for later experiments. Naturally, publication of the experimental data would not have been possible without strong collaboration. At this point, I would also like to acknowledge his support, as well as the invaluable support from Prof. Dr. Sebastian Otte, who provided advice and contributions that were crucial in improving our papers to a top-level standard. I am also grateful to Prof. Dr. Andreas Zell for his challenging yet motivating questions and constructive criticism of the intermediate results during their presentation. These insights prompted me to reassess the scope of my investigations and to consider parametric modules.

I would also like to acknowledge those people who guided and supported me during my student years, and who had a significant positive effect on the constructive way of approaching the problems, the effective way of working and on the optimistic way of life perception. I express my gratitude to Dr. Ulrich Kreßel, who was the supervisor of my master thesis and the colleague with whom we worked at the Daimler research center in Ulm (formerly the DaimlerChrysler research center in Ulm), and to Prof. Dr. Eugene S. Semenkin, my academic advisor and mentor during my time as a student at the former Siberian Aerospace Academy in Krasnoyarsk. It was a great pleasure working with them.

Additionally, I would like to express my sincere gratitude to my parents for their lifelong support. My special thanks go to my wife Tatjana for her endless love and understanding throughout the years of this work.

Dedication

To my beloved wife Tatjana and our dear sons, German and David.

Contents

List of Tables

List of Figures

Abbreviations

AI Artificial Intelligence

AE Artificial Evolution

ANN Artificial Neural Network

BP Error Backpropagation

BPTT BackPropagation Through Time

DE Differential Evolution

DESN Decoupled Echo State Network

EA Evolutionary Algorithm

EM Expectation-Maximization

ESMoN Evolutionary Synchronization of Modular Network

ESN Echo State Network

FT Fourier Transform

FF-ANN Feed-Forward Artificial Neural Network

FFNM Feed-Forward Neural Module

FFT Fast Fourier Transform

GSMoN Gradient-based Synchronization of Modular Network

HHT Hilbert-Huang Transform

HME Hierarchical Mixture of Experts

IR Identification Rate

LSTM Long Short Term Memory

ME Mixtures of Experts

mESN modular ESN

MG Mackey Glass

MLP Multi Layer Perceptron

MSE Mean Squared Error

MSO Multiple Superimposed Oscillators

MST Mixtures of Sine waves and Triangular oscillatory signals

NE Neuroevolution

NRMSE Normalized Root Mean Squared Error

OFB Output Feed-Back

RBF Radial Basis Functions

RLS Recursive Least Squares

RMSE Root Mean Squared Error

RNM Recurrent Neural Modules

RNN Recurrent Neural Network

SR Spectral Radius

STM Short Term Memory

WSA Winner Score Accumulator

1. Introduction

1.1. Motivations

Modular networks are structures composed of distinct entities, called modules, each performing dedicated tasks to achieve a common goal for the entire network. Modular networks are well-suited for modeling structured data, as each module can represent a corresponding portion of that data. Training a modular network involves adjusting the parameters of individual modules in a way to minimize interference between them. This may improve the computational efficiency and performance of the resulting data model, leading to higher modeling accuracy. Several studies have taken advantage of this characteristic to develop solutions that outperform monolithic models. Furthermore, modular structures are quite suitable for understanding complex systems ([Aza00; BBE11]), where the constituent parts are represented by the corresponding modules. However, this potential of modular networks remains poorly explored, although it can be applied for solving particularly challenging problems such as understanding the dynamics of the human brain ([Pas98; JMP07]) and developing combinatorial generalization [Bat+18; FOB21], which may form the basis for Artificial Intelligence (AI) in the future.

The absence of effective training algorithms makes modular networks less attractive for the identification of dynamic components in sequence data. These training procedures are typically complex, as they adjust module parameters based on signal flow between modules ([Jae+07; XYH07]). Population-based adjustment of dynamic modules offers the potential to overcome this limitation by making the training procedure largely independent of the modular structure and more effective, as it is not tied to a specific training method. The increasing complexity of sequence data can be addressed directly by increasing the number of available modules. While co-evolving the dynamics in ensembles of modules, the most likely combination of module dynamics is identified and progressively harmonized. Consequently, the sequence data is decomposed into a set of relevant modules, where each module unambiguously identifies its corresponding dynamic component through precise reproduction.

The necessity of modeling dynamic components justifies the utilization of Recurrent Neural Network (RNN)s. Focusing on population-based training suggests a need for compact modules to efficiently utilize computational resources. The study [RI09] demonstrated that the Echo State Network (ESN) is a type of RNN that combines compactness with the ability to generate complex oscillatory signals even with a small

number of neurons. Increasing the ESN size significantly enhances their capability to model more complex data, such as chaotic attractors like the Mackey-Glass (MG) time series [JH04; VS09] and the Lorenz attractor [RI09; JH04]. Given that smaller neural networks tend to promote better generalization, module compactness is essential for achieving a meaningful decomposition of sequence data. This highlights the central challenge of module design: finding an acceptable solution to the plasticity-stability dilemma ([HM94; Fre99; Kha06]), which supposes that overly compact modules may lack sufficient accuracy, while the excessive flexibility of large modules may lead to low stability and poor generalization ([Jae+07; WSS08; KLB12; WS09; OXP07]).

A significant challenge for dynamic data decomposition using modular networks is that it is an underconstrained problem: *"function decomposition is an underconstrained problem, and, thus, different modular architectures may decompose a function in different ways"* ([JJB91], p.219). This implies that the desired decomposition cannot be guaranteed by module compactness alone but also requires limiting the variety of induced dynamics within the network. Therefore, the research is not bounded to the question of identifying and harmonizing module dynamics but must also address how to restrict excessive flexibility in the modular structure. This can be achieved by imposing explicit constraints on the generated signals or implicitly through module design and by connecting the modules according to an assumed relationship between the hidden dynamic components within the sequence data. Assumptions about the types of dynamic components and their interrelationships within the sequence data introduce a structural bias that guides the convergence of the training procedure toward the global optimum and facilitates meaningful data decomposition.

The scarcity of available sequence data presents a common challenge for both module design and the identification of dynamic components. Regarding module design, the limited length of available sequence data hinders deep training, preventing modules from achieving high accuracy in signal generation. Concerning component identification, short sequences preclude a trial-and-error approach and prevent data decomposition from benefiting from the potential presence of dynamic attractors. (Dynamic attractors rely on the dynamics of RNN settling into a stable state after a sufficiently long time [Pas98; Gra13]). Therefore, this thesis aims to address how to utilize low-accuracy modules for the reliable identification of individual dynamic components in temporal data. A promising solution lies in combining population-based adjustment with statistical decision-making, which is expected to boost the overall quality of identification by using a multitude of relatively inaccurate individual modules.

To address the aforementioned research challenges, this thesis will explore the concept of population-based identification of dynamic components within given sequence data through their precise generation. The challenges are expected to have potential alternative solutions, which will be investigated in the experimental studies. The results obtained are anticipated to demonstrate both the advantages and potential limitations of the proposed identification approach across various types of sequence data.

1.2. Research goals

The central goal of the research presented in this thesis is to provide evidence that modular architectures are suitable for identifying hidden components in sequence data, thus addressing the question: *"For what kind of tasks are modular architectures favorable?"* [HIT02] To demonstrate the advantages of modularity for identifying hidden components in sequence data, this thesis proposes an approach to decompose given sequence data into a set of modules capable of modeling individual hidden dynamic components, thus enabling their unambiguous identification. The viability of the proposed approach will be evaluated using various types of sequence data.

Assuming that the given sequence is a mixture of hidden dynamic components, the proposed approach considers the modeling entities as modules contributing to the output of a modular network, which reproduces the given sequence. [Kha06] demonstrated that feed-forward neural modules can be evolved solely based on their contribution to the accuracy of the resulting model. This thesis will investigate whether the same principle applies to decomposing given sequence data using recurrent neural modules.

The behavior of dynamic modules is determined by their internal states and potential static parameters. Because decomposition is achieved through reconstructing the given sequence data, the method for tuning module states and parameters influences the quality of the decomposition and is therefore crucial for reliable identification of hidden dynamic components. If the modules' internal states are encoded using real-valued vectors, then gradient-based approaches and differential evolution are suitable candidates for the tuning procedure. However, increasing the number and potentially the size of the modules exacerbates the optimization problem, making the objective function more difficult to manage. This thesis will investigate the effect of increasing complexity on the identification results obtained with both optimization methods to determine their respective advantages and potential drawbacks.

Furthermore, this thesis will demonstrate the flexibility of the proposed approach by using different module types for the identification of dynamic components, accommodating the diverse properties of the given sequence data . This should demonstrate the potential to compete with even the Fourier transform, which uses only sine waves as basis functions and is unsuitable for short sequences and non-stationary dynamics. The flexibility afforded by module selection permits the extension of the identification scope beyond the mere determination of the presence of specific dynamic components to encompass the computation of their parameters. In particular, artificial evolution is expected to provide even greater flexibility, as it does not rely on assumptions about the objective function's properties.

Given the ability to train neural networks without relying on prior knowledge, they appear quite flexible for designing modules suitable for modeling a variety of potentially present dynamic components. Larger neural modules are typically able to model more complex data with greater accuracy. This question is theoretically studied within the

context of the plasticity-stability dilemma: *"the tension between the two counteracting constraints of a sufficiently plastic structure to accommodate new patterns, while simultaneously ensuring the stability of existing representations"* ([HM94], p.993). This dilemma suggests that larger neural networks exhibit higher accuracy but poorer generalization. That is, the same large neural network is likely to precisely model different components of the underlying structure. Consequently, it is expected that module size will influence the quality of data decomposition. This thesis will consider this aspect in module design and investigate the effect of module plasticity on the resulting quality of component identification.

A further advantage of modularity is the ability to adapt the model to non-stationary changes in sequence data, where online training approaches are particularly well-suited ([Jae03]). This thesis will demonstrate the ability of online synchronization to decompose non-stationary sequence data and track changes in the underlying dynamics over time, ensuring the modular network remains interpretable.

1.3. Contributions

This thesis presents a novel online co-evolutionary approach for the identification of hidden dynamic components in time series, which is schematically illustrated in Figure 1.1, and demonstrates its viability in analyzing oscillatory dynamics as well as in classifying non-oscillatory data sequences. The oscillatory time series are composed of different types of oscillatory signals: sinusoidal, triangular, and rectangular oscillations. In the non-oscillatory sequences, the components are aperiodic dynamics used to model handwritten symbols. The identification concept involves representing the given sequence data as a modular network, where each module is responsible for a corresponding data component. In this research, the modular architecture is an ensemble of linearly combined dynamic modules tuned by synchronizing their internal dynamics. During synchronization, the given sequence data is distributed as the total activity among the available modules. For the experimental studies, separate dynamic neural modules are implemented using ESN, thus forming a modular ESN. Although there are numerous papers on ESNs and their variants, their modularization is still underrepresented in the literature.

This thesis analyzes the effect of various module types on the method's ability to find a proper decomposition of sequence data and demonstrates their advantages with respect to the resulting accuracy, identification rate, and descriptive power of the obtained model. All considered module types – parametric, monolithic, and composite neural modules – provided high modeling accuracy and high identification rates, thus demonstrating the method's independence from the chosen module type and its significant flexibility. Moreover, the good results obtained on short and non-stationary sequences show this approach as a time series analysis method with significant potential to compete even with well-established analysis methods such as the Fourier transform. Additionally, the thesis experimentally demonstrates that linear combinations of oscillator modules can

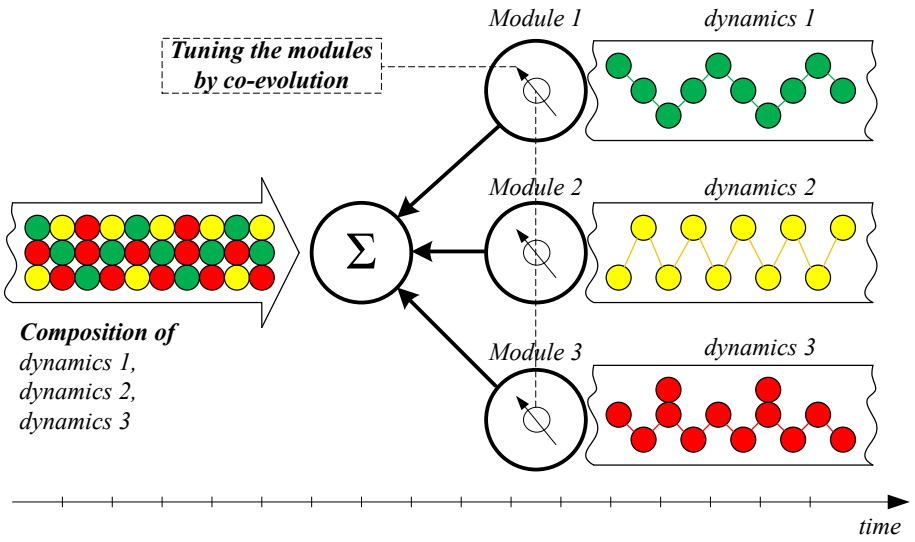

Figure 1.1.: Schematic representation of the explored online co-evolution for identifying hidden dynamic components in time series. Co-evolution tunes the internal dynamics of available modules for the precise reconstruction of a given compositional time series, thereby associating hidden dynamic components with the modules that most accurately generate them.

provide a quite accurate short-term decomposition of chaotic attractor dynamics, such as the MG time series.

Gradient-based tuning of the modules' internal dynamics is presented as an alternative approach to evolutionary tuning. The gradient-based method relies on a set of formulas derived for recurrent neural modules as well as for differentiable parametric modules. I analyze convergence under various conditions to determine the strengths and weaknesses of both methods. The evolutionary approach is generally slower but appears to be more powerful and applicable with various module types for the decomposition of time series, irrespective of the properties of the dynamic data components. The gradient-based approach provides much faster convergence but ensures reliable identification of dynamic components only in combination with neural oscillators that exhibit attractor characteristics. Identification of signals with non-differentiable shapes, such as triangular oscillatory signals, works only with neural oscillator modules and for a small number of dynamic components. In this respect, this thesis provides an explanation for these limitations by analyzing changes in the populations of states along the trajectory of the global optimum in the state space. Furthermore, the discovered dependency between the deployed module types and the parameters of evolutionary tuning is investigated further to provide recommendations for optimal parameter settings.

When dealing with non-stationary data, this thesis demonstrates the ability of synchronization to catch up with the changes in the underlying dynamic structure and reflect

these changes in the model's structure. The investigations reveal that the presence of a built-in attractor in the neural oscillator modules is beneficial for preserving residual diversity within the populations, providing potential for further evolutionary development in non-stationary environments. This eliminates the need to apply techniques such as hypermutation or niching, which have been previously recommended elsewhere (e.g., [Pol+07], [DS11]) to increase population diversity, typically required in non-stationary environments. Thus, the results obtained with non-stationary data should be viewed as a contribution to the limited range of artificial evolution applications to dynamic and non-stationary environments ([CD02; TM99; Win00]).

In addition to the compositions of continuous oscillatory signals, this thesis explores the possibility of applying the identification approach to recognize handwritten symbols represented as short, non-oscillatory data sequences. To deal with them, evolutionary tuning is combined with statistical decision-making based on collected class statistics. Experiments with this hybrid approach demonstrate the advantage of boosting class statistics by the statistics from multiple sequence iterations for reliable recognition. This thesis describes a method for tuning the rejection behavior of the recognition algorithm. Unlike the typical problem setting of classifying a single symbol, this approach is also benchmarked by identifying multiple symbols in a mixture.

This thesis demonstrates the importance of imposing proper constraints to prevent premature convergence of synchronization, regardless of the type of given sequence data. These constraints limit potential excessive activity in the deployed modules. This proves indispensable for dynamic problem decomposition when the modules are implemented with neural networks whose activity does not decay over time, such as Multi Layer Perceptron (MLP) or ESN. Consequently, the range of deployable module types is not restricted to Radial Basis Functions (RBF) networks with their Gaussian-style activity but can also include RNNs, which exhibit a global character of their output signals.

As demonstrated by the experimental data, the quality of identification generally depends on the properties of the deployed modules, such as compactness and plasticity. For successful identification, it is necessary to ensure the presence of these properties independently of the properties of the given sequence data. Nevertheless, specific characteristics of the data must be considered in the module design. This thesis reveals that the magnitude range of the driving signals is a crucial factor for obtaining ESN modules capable of modeling oscillatory signals. This discovery disproves the previous argument that ESNs are unsuitable for mixtures of oscillatory signals, enabling the design of compact ESN modules for modeling dynamic components in oscillatory dynamics. These results were published in [KLB12]. Furthermore, this thesis presents an approach for designing compact, yet sufficiently plastic and recurrent modules for modeling non-oscillatory data sequences. This includes augmenting the given training sequences of handwritten symbols, designing the module's structure, regularization, and learning parameterization of recurrent modules. The design of the module's structure reveals the particular usefulness of unstable dynamics and dynamics at the limit of stability for obtaining compact neural recurrent modules for symbol curves with strong

form asymmetry. The conducted experiments demonstrate the advantages of learning parameterization compared to the alternative with explicit rotation of handwritten symbol curves and propose it as a method for realizing multi-tasking with ESNs.

1.4. List of Publications

The research for this thesis resulted in the following publications.

[Gue+12] J. Guetschow, J. Lohmann, D. Koryakin, and M.V. Butz. "Learning Motor Primitives with Echo State Networks". In: *Machine Learning Reports*. Vol. 3. 2012, pp. 20–33.

[KB12] D. Koryakin and M.V. Butz. "Reservoir Sizes and Feedback Weights Interact Non-linearly in Echo State Networks". In: *Artificial Neural Networks and Machine Learning (ICANN)*. Vol. 1. 2012, pp. 499–506.

[KLB12] D. Koryakin, J. Lohmann, and M.V. Butz. "Balanced echo state networks". In: *Neural Networks* 36 (2012), pp. 35–45.

[KSB15] D. Koryakin, F. Schrodt, and M.V. Butz. "Ensembles of Neural Oscillators". In: *New Challenges in Neural Computation*. Vol. 3. 2015, pp. 57–64.

[KOB21] D. Koryakin, S. Otte, and M.V. Butz. "Inference of time-series components by means of online co-evolution". In: *Genetic Programming and Evolvable Machines* (2021), pp. 1–29. DOI: 10.1007/s10710-021-09408-6.

1.5. Organization of the Thesis

This thesis is structured into eight chapters. The current paragraph presents an outline of the thesis and provides a short summary for each chapter.

Chapter 1 provides the motivations behind the conducted research, the research goals, and lists the contributions resulting from the research.

Chapter 2 presents an overview of relevant studies and different aspects of modular networks – their properties, topologies, modules, modularization approaches, and advantages over non-modular models. The most relevant concepts and features of related approaches are briefly described to position the research subject within the field and to highlight the potential of the proposed identification approach for the thesis goals.

Chapter 3 introduces the basic concept of identifying hidden dynamic components through synchronization of a modular network with given sequence data. This chapter provides an algorithmic description of the synchronization method, defines the module types and modular topology considered in the experiments, and outlines the properties

that justify their use for identification. To specifically underscore the efficiency of modular ESNs for modeling time series composed of dynamic components with significantly different characteristics, the results from our paper [KSB15] are mentioned in subsection 3.3.1. Furthermore, the results from [KSB15], demonstrating the superiority of modular networks over monolithic networks, are referred to in subsection 3.3.2.

Chapter 4 provides a detailed description of the identification approach and analyzes its behavior during the decomposition of oscillatory dynamics. It defines the accuracy measure used for evaluating candidate modular decompositions and explains its effects on tuning a single module in the context of the other modules in the ensemble. The chapter also specifies how the update rules of the evolutionary and gradient-based methods are applied for module tuning. Furthermore, it presents the derivation of formulas for the gradient-based update rules for recurrent and differentiable parametric modules. Additionally, the chapter defines the types of imposed constraints and details the stages of harmonizing module dynamics within the ensemble.

Chapter 5 reports the achieved identification performance and modeling accuracy on oscillatory time series. The description of the design procedure for recurrent neural modules emphasizes the main factors to be considered to obtain the smallest modules that provide exceptionally high modeling accuracy. This design procedure is based on the findings published in our paper [KLB12], which are reiterated in subsection 5.2.1. Subsection 5.2.2 applies the balancing condition from [KLB12] to analyze and explain the differences in the performance of the obtained ESN modules. The performance comparison with Fast Fourier Transform (FFT) from our paper [KOB21] is given in subsection 5.3.2, and the performance on non-stationary time series in section 5.4. The chapter concludes with key findings on the identification of oscillatory components.

Chapter 6 details the combination of population-based tuning of module internal dynamics with statistical decision-making methodologies for identifying aperiodic dynamic components in discontinuous sequence data. Sequences of handwritten symbols are used to illustrate the challenges posed by discontinuous sequences for the design and identification of dynamic components. This chapter describes the design steps taken to address these challenges and to obtain compact composite neural modules with an optimal balance between size and accuracy for modeling the symbol curves.

Chapter 7 reports the recognition performance for handwritten symbols in single-class and multi-class sequences, where each sequence presents a mixture of several symbol curves. The chapter compares alternative module designs based on their performance and describes an approach to adjust the rejection behavior. The chapter concludes by summarizing the findings on the identification of aperiodic dynamic components.

Chapter 8 draws general conclusions from the research in this thesis, outlines potential extensions of the identification approach, and suggests ideas for future research.

More detailed information about related work and specific implementations is provided in the appendix.

2. Modularity: Types, Properties, Methods

This chapter presents a survey of the literature on modularity, relevant to the idea of applying modularity to identify dynamic components within the underlying structure of given sequence data. Initially, this survey helped establish the foundation for the conducted research, identify novel directions through a critical review of related studies, and understand the potential of applying modular networks for the automated analysis of dynamic data. The requirement for precise modeling of dynamic components established a connection to methods for designing modules and tuning their ensembles using available data. The described properties of various modular networks and the resulting conclusions, published in relevant scientific papers, revealed the advantages and disadvantages of modularity compared to non-modular representations. These findings highlighted its significant potential while also providing insights into its limitations for realizing the proposed idea.

The majority of the literature on modularity concerns Artificial Neural Network (ANN)s, which can almost be considered universally applicable for modeling both static data and dynamic signals of various types, while their complexity can be easily controlled by the Artificial Neural Network (ANN) size. Nevertheless, some studies of parametric models, such as Bayesian networks, have also yielded relevant results and are included in this survey. Although the universality of ANNs is quite attractive, other module types may be more preferable because they satisfy specific limiting conditions in a particular application.

In addition to the choice of module type, successful modular decomposition of application data requires selecting a suitable relationship between the modules in the network, i.e., their interconnectivity. Bearing in mind the goal of obtaining an understandable representation, fully automatic, unrestricted decomposition of the data is generally undesirable, as it often produces performance-oriented models with a tangled topology. The chosen module type and modular topology define the elements in the final representation's structure, thus properly restricting it in types. This introduces a structural bias that must be considered in the decomposition method to discover the hidden structure according to predefined performance criteria. The following paragraphs describe the most prominent modular architectures, previously presented methods for identification of modular structures, and their merits. Moreover, they highlight their relation to the proposed method for the identification of the hidden dynamic components.

2.1. Alternative views of modularity

Not only do the properties of modular networks depend on the application domain, but also the notion of modularity itself is viewed differently in various studies and largely depends on the goals of applying a particular modular model. The majority of authors rely on understanding modularity as structural or topological modularity, as defined in [BBE11]: *"A modular network is a network which is well divided into modules such that there are dense internal connections between nodes within modules but only sparse connections between different modules."* (p.2) The presence of connections within modules refers mainly to neural modules, where the neurons of the same module are densely connected to each other. Separate modules can accept signals from or send signals to other modules through input or output connections, respectively. Sparse connections between the hidden neurons of different modules are also conceivable. Dense connectivity of signals implies intensive signal processing within modules, whereas inter-modular connections serve purely for exchanging the results of signal processing between modules. Similarly, ensembles of non-neural modules, such as probabilistic rules in [KS13] or generalized linear models in [CT06], fit the definition of structural modularity, as intensive signal processing occurs within the modules as well.

Alternatively, complex systems can exhibit functional modularity, which focuses on distinct functionalities within the system rather than its structural elements. Functional modularity should be viewed as a higher level of abstraction compared to structural modularity, because the realization of a single functionality can involve several topological modules within the complex system. As applied to biological organisms, according to [HM94]: *"Apart from a layered, hierarchical structure in the primate brain, we find multiple parallel processing streams or pathways that constitute another major neural construction principle"* (p.986); that is, each module encompasses all information flows related to a specific functionality. Besides involving structural elements, the realization of separate functionalities can be based on the modularity of parameters responsible for the emergence of those functionalities in the network, such as Dynamic Bayesian Networks, as investigated in [RH10]. In this study, modularity of likelihoods and the modularity of parameter sets of mixture components were assumed for modeling non-stationary processes in neural information flow in biological organisms. In [MC13], a modular ESN was presented, where a module is defined through the elements of the output vector responsible for a particular region of the modeled output dynamics.

In addition to the qualitative views of structural and functional modularity, there have also been attempts to quantify them, such as in the thesis [Kha06] through the introduction of Structural and Functional Modularity Indices (*SMI* and *FMI*), respectively. Another study, [HIT02], defined two other indicators, $M^{(arch)}$ and $M^{(weight)}$, which differentiated between modularity of architecture, where the actual weights between neurons were neglected, and modularity of weights, which considered the weights. Such measures are quite valuable for the automatic decomposition of data, allowing for a more precise description of the modular decomposition goal.

Although the use of functional modularity is conceivable for developing understandable representations, unambiguously identified relevant structural elements offer greater expressive power in delineating rigid boundaries between dynamic components in given sequence data. Accordingly, networks exhibiting traits of structural modularity appear to be a more suitable descriptive form for their identification.

2.2. Advantages of modularity

Modularity offers several advantages that distinguish modular networks from their non-modular counterparts, making them valuable for applications beyond simply achieving high data modeling accuracy. Numerous studies have analyzed the behavior of modular networks and highlighted the benefits of modularity within their respective application domains.

Biological plausibility of the artificial modular networks supposes their similarity to the brains in biological organisms with respect to the ability to distinguish separate structural or functional entities within their internal organization. This has a dual benefit for research. On the one hand, the development of artificial modularity is facilitated by replicating the properties of brains, which results in acquiring their advantages, such as optimal reaction to external stimuli and highly efficient learning of the environment by adapting the functionality of the separate modules and the information exchange between them to environmental changes. On the other hand, the research of brains is supported by modeling their behavior with artificial modular networks.

The human brain is most frequently referenced in this context. Its performance, the richness of its provided functionalities, and its adaptive ability have fascinated many scientists: *"Modularity develops in the brain to be able to flexibly relate particular predictive encodings across space and time."* ([But16], p.15) The presence of modularity in the brain, as well as in artificial modular networks, makes them suitable for understanding the brain when they are used for modeling a particular functionality to simulate the brain's reaction under different conditions. [Pas98] analyzed the dynamics in combined modules to derive design principles for such models: *"We claim that experiences with complex dynamical properties of artificial neural networks can provide guiding principles for modeling biological brain functions."* (p.17) The simplified relations between artificial modules can support the analysis of the internal unfolding processes inside the model to shed light on the relation between similar processes in the brain and its cognitive abilities, with the significant potential to foster further development of AI.

Ability to develop interpretable representations. In the context of the stated objectives, the ability to generate interpretable representations is the most pertinent advantage of modularity in my research. This ability is achieved through network training, which aims to precisely align modules with corresponding fractions of the training data, thereby enabling the interpretation of the resulting module configuration. In [JJB91], one of the earliest works on modularity, this was noted as: *"Modular architectures tend to develop*

representations that are more easily interpreted than the representations developed by single networks." (p.6) Later studies proposed more advanced concepts that mimic information processing in the human brain, such as the concept of graph network (GN) modules in [Bat+18], which also concluded their ability to produce interpretable models: *"Human cognition makes the strong assumption that the world is composed of objects and relations, and because GNs make a similar assumption, their behavior tends to be more interpretable. The entities and relations that GNs operate over often correspond to things that humans understand (such as physical objects), thus supporting more interpretable analysis and visualization."* (p.24).

Analyzing the modules within a network allows for the interpretation of data's underlying dynamics by examining signal processing. Modularity facilitates a hierarchical understanding of the model, specifically at the intra-module signal processing and inter-module information exchange levels. Various methods exist for analyzing modular network operations. One approach, as suggested by [JJB91], involves comparing individual module activity to both the network's overall activity and other modules' actions. Another technique, demonstrated by [BBE11], tracks changes in dynamic network structure to infer process evolution. However, a straightforward method for understanding network internals is to focus on modules with the strongest outputs, as these modules largely define temporal activity patterns, aiding in dynamic component identification.

In addition to the ability to develop understandable representations, modularity brings other advantages that have been discussed in several studies and are potentially relevant for the precise identification of dynamic components in given sequence data.

Extendibility and ability of incremental learning. This feature allows adding new knowledge to the modular network without disturbing the already available knowledge. Since new knowledge is often incorporated into new modules, the completion of the network with these modules does not affect the functionality of the existing modules. The modular network has the *"ability to learn patterns in a sequential manner without much interference with existing representations"* ([HM94], p.23). In [Aza00], the approach relied on extendibility to enlarge the hierarchical organization of the modules with more entities, with the benefit that it was not necessary to expend computational effort to update the modules already contained in the hierarchical network: *"modular neural networks present an architecture which is suitable for incremental addition of modules that can store any incremental addition to the already existing learned knowledge of the modular neural network structure without having to retrain all of the modules"* (p.29).

The capacity to integrate novel knowledge without disrupting existing one enables modular networks to effectively mitigate catastrophic forgetting, a phenomenon where previously acquired knowledge is lost upon learning new information ([Fre99]). This issue typically arises from the network's limited memory capacity and representational overlap, when a single module is tasked with multiple distinct functions. By assigning each module a single task and ensuring learning occurs without spatial and temporal crosstalk, as defined in [JJB91], representational overlap is minimized. Consequently, the overall memory capacity of the modular network is enhanced, and catastrophic forgetting

is prevented. In [Jac+91; JJB91], the mixtures-of-experts approach was employed in static data applications to partition the feature space, facilitating the identification and assignment of relevant data to one or more modules. They utilized a modular ANN with one module per identified region and asserted that *"modular architecture consisting of a separate network for each output unit is immune to spatial crosstalk."* ([JJB91], p.222).

Furthermore, *"a functional advantage of the anatomical separation of different functions might be the minimization of mutual interference between simultaneous processing and execution of different complex tasks"*. ([HM94], p.3) This implies that optimizing individual modules can maximize their task execution performance without impacting other modules: *"Independent functions are coded independently using modularity, and as such each function can be optimized separately with little interference with other already optimized functions"* ([RSM04], p.2). Given that module parameters only affect the performance of their respective module, multi-criteria optimization becomes unnecessary for maximizing overall network performance. This optimization can be achieved more simply with a single criterion, in contrast to the complexity of multi-tasking non-modular architectures. In addition, reducing mutual interference between modules helps eliminate superfluous recurrent dynamics, leading to improved performance: *"There are instances where modular architectures are shown to outperform fully-connected architectures primarily by minimizing modular interference."* ([Kha06], p.3) For the identification of dynamic components, the extendibility of modular representations offers a pathway to enhance the identification method's capabilities by incorporating additional modules as soon as new insights into the nature of the given sequence data suggest their potential relevance.

Flexible module deployment. The possibility to flexibly vary the ensemble of available modules extends beyond simply augmenting the network with new components. It also encompasses scenarios where a specific module within the network is substituted with its advanced version, offering superior performance, or even with a module of a different type which is deemed more appropriate according to particular metrics, such as reduced module size, and has a compatible interface for integration in the network. In essence, the adaptability to deploy alternative modules is a hallmark of genuinely modular systems, irrespective of their application domain. For instance, in software development, a suboptimal software module can be replaced with a more efficient and compatible counterpart.

My research benefited from this ability in experiments involving the recognition of handwritten symbols, where the symbol-related modules were designed and optimized independently of the symbol recognition algorithm. Should an external optimizer identify a superior module for a certain class, it could directly substitute its predecessor within the dynamic module ensemble. It is important to note, however, that this strategy for deploying alternative modules is most effective within networks composed of independent modules that exhibit no representational overlap.

At the same time, the presence of representational overlap does not invariably constitute a disadvantage for modular networks. Representational overlap is essential for developing

reusable modules that are shared across multiple tasks or contribute to modeling distinct fractions of data. In this context, a reused module need not necessarily execute an entire task independently; rather, it can be integrated with various module groups to achieve the desired composite outcome. For instance, in the hierarchical classifier proposed by [Kor01], a high-level classifier module can ascertain the group class membership of an input pattern, while leaf classifier modules determine the precise class label. On this way, reuse of the high-level module makes the overall modular network compact and facilitates its rapid adaptation to changes in external stimuli, as observed in [Kha06]: *"in dynamic environments, primarily, because of the possibility of reuse of modules, modular structures are at an advantage irrespective of learning conditions"*. In this study, module reuse was demonstrated on tasks exhibiting characteristic similarities within a gated non-hierarchical network, where sub-modules remained unchanged while the gating module adapted to environmental changes.

Ability to incorporate a priori knowledge. *"A strength of the modular architecture is that its structure is well suited for incorporating domain knowledge"*. ([JJB91]) This ability enhances the efficiency of the identification approach by enabling the selection of specific relationships between modules (e.g., a hierarchy of the temporal dependencies in [EB95; Jae07]) and by selecting a module type which aligns with the characteristics of the data from the application domain (e.g., the design of an analytical knowledge-based module in [Pat+18b]). The integration of domain knowledge implicitly imposes constraints, thereby preventing the decomposition algorithm from exploring combinations of features that are a priori unacceptable. As [JJB91] notes, *"finding appropriate restrictions is possible through the application of domain knowledge"*. Consequently, the identification of data components goes faster, with a reduced risk of misidentification. Furthermore, domain knowledge can provide explicit rules that are weakly represented in the training data, the consideration of which improves the generalization of the resulting model: *"learning in largely unstructured networks may often result in good specific solutions which incorporate rules that effectively describe the regularities and characteristics of the training set. Such specific solutions, however, usually lack generality. In order to find good general solutions, knowledge about the total task domain needs to be incorporated in the network."* ([HM94], p.21)

The choice of a suitable parametric module type also facilitates understanding of the underlying dynamics. In this sense, an appropriate parametric module type can be more preferable than universally applicable but less interpretable neural modules, whose coefficients can hardly be associated with the properties of the modeled processes.

Computational efficiency of modularity has been recognized since the inception of machine learning, a time when computational resources and data were significantly limited: *"a key reason why structured approaches were so vital to machine learning in previous eras was, in part, because data and computing resources were expensive, and the improved sample complexity afforded by structured approaches' strong inductive biases was very valuable."* ([Bat+18], p.2) Consequently, modularity introduced a structural bias, enabling the scaling down of learning problems to fit within available computational resources and facilitating efficient problem-solving. Specifically, the human brain's modular organization, which

achieves remarkable efficiency in addressing highly complex challenges, has been a source of inspiration for many researchers. As noted in [HM94]: *"only because of its highly organized architecture that the brain manages to execute a myriad of functions and yet maintains a compact size"* (p.2) Analogously, modularity promotes compactness in artificial modular systems, which achieve enhanced performance with reduced memory requirements compared to non-modular counterparts, as demonstrated in [MC13; KSB15]. These studies showed that modular RNNs required significantly fewer neurons than monolithic neural networks to replicate time series with equivalent accuracy. Furthermore, as proposed by [Pat+18a], the challenge of increased problem dimensionality can be effectively addressed by integrating additional modules in parallel.

Compactness and reduced complexity are related to other features of modular networks, such as reduced representational overlap, where dynamic modules do not need to include additional structural elements to compensate for potential mutual influence between concurrently unfolding dynamics. Nonetheless, compactness of individual modules does not compromise their operational efficacy: *"already small artificial neural networks with recurrent connectivity inherit complex dynamical features"* ([Pas+01], p.2). Incorporation of a priori knowledge can also significantly curtail the total size of the modular constellation ([Pat+18b]).

Besides, the compactness of modular networks facilitates a more rapid computation of output signals in response to external stimuli, both in environmental ANNs ([NP95; Pas+01]) and in their hardware-embedded implementations ([SPK98]). Beyond the compactness of individual modules, structural modularization provides a significant advantage in the parallelization of training, optimization, and operation of distinct modules. This has motivated numerous studies on modular networks within the context of artificial evolution ([HM94; HIT02; Kha06; GSM08; Yan+09; BBE11]). In this domain, [RSM04] highlighted the beneficial impact of modularity on the efficiency of neuro-evolutionary algorithms: *"With the added level of abstraction that modules provide, more complex solutions are searched with fewer operations than standard NE"* (p.71); and *"developmental modularity allows for reuse of complex phenotypic structures without correspondingly complex mutations in the genotype."* (p.70). Additionally, computational efficiency can be achieved through a non-uniform distribution of module updates across the network, as demonstrated by the hierarchical network in [CAB17], where reduced computational effort was allocated to less frequent updates of high-level modules. This strategy aimed to capture long-term dependencies and mitigate the vanishing-gradient problem during sequence data modeling.

High performance. Beyond the compactness of modular networks, the structural segregation of their modules significantly enhances modeling accuracy. Research on modularity has demonstrated (i) a substantial reduction in modeling error compared to non-modular models, (ii) the potential of modularity to improve generalization, and (iii) the ability to solve the given problem at all.

The primary objective for performance enhancement is to minimize modeling error.

A direct approach to achieving this involves reducing representational overlap by constructing the model from several smaller, yet more precise, modules. As noted in [Aza00], p.48, *"[The task] could be accomplished with a monolithic neural network, but a better performance is achieved when the neural network is broken down into a number of specialist modules."* This principle was exemplified in [KSB15] using modular ESNs on mixtures of dynamic components, where the accuracy improvement was more pronounced with an increased number of dynamic components hidden within the sequence data. [HIT02] demonstrated that *"modular networks are indeed advantageous for modular problems"* (p.225) in classification tasks. The second way for enhancing model accuracy is through the indirect estimation of unknown, performance-critical parameters by integrating modules that operate with varying parameter values. For example, [BFC96] employed time-delayed modules, each with distinct sliding window widths, to estimate the optimal width for time series modeling. Similarly, [BP11] combined ESN modules with diverse memory lengths within a single network, resulting in significantly improved accuracy compared to standard ESNs across all evaluated time series. The third method for accuracy improvement was illustrated in [Pat+18b], where one of two linked modules compensated for output signal imprecision in the other.

Beyond enhancing accuracy on available sample data, several researchers have recognized the potential of modularity to improve the generalization abilities of resulting models. [HM94] explored the design of modular neural networks through artificial evolution and applied them to handwritten digit recognition. In their experimental results, the authors observed superior performance on previously unseen samples and concluded that *"modular architecture can induce a system to better generalize its learned behaviour to instances never encountered before"*. (p.23) [JJB91] discussed possible reasons to the expected strong generalization of modular networks, hypothesizing that the structure of a well-generalizing network must align with the structure of the task being modeled: *"If a modular architecture were able to decompose a complex function into a set of simpler functions and allocate an appropriately structured network to each simpler function, then one would expect good generalization."* (p.224) Another reason is the reduction of temporal and spatial crosstalk: *"modular architectures perform local generalization in the sense that each network [each module] of the architecture only learns patterns from a limited region of the input space"*. (p.224)

The ability to solve the give problem at all is a another strength of modular networks, a feat not consistently achievable with monolithic models: *"it might not be possible to accomplish a task in question unless the problem is first simplified by decomposing it into simpler subtasks."* ([Aza00], p.48) This embodies the divide-and-conquer principle, which *"generally forms the basis for the modular neural network design"*. ([Aza00], p.48) The feasibility of solving otherwise intractable problems is also highlighted as a primary motivation for focusing on modular networks in [Sha97a]: *"a modular approach might be used to solve a problem which could not have been solved through the use of a unitary net"*. (p.4) For instance, while a monolithic ESN proved incapable of modeling the Mixtures of Sine waves and Triangular oscillatory signals (MST), a modular ESN in [KSB15] achieved a

highly precise representation.

About some risks. Despite the aforementioned advantages of modularity, it presents certain limitations that can diminish the quality of the resulting model and need to be aware of. Firstly, in specific applications, monolithic models may surpass modular approaches: *"It is observed that a non-modular solution, trained in a particular fashion, can sometimes outperform the modular solution on the basis of the generalization performance."* ([Kha06], p.66) This observation was corroborated by the study [KOB21], where smaller non-modular ESNs exhibited significantly superior modeling accuracy compared to their modular counterparts. Furthermore, on non-separable problems, the performance gains may be marginal, hardly justifying the additional algorithmic and programming effort required for implementing a specific modular decomposition strategy: *"For simpler, static problems modularity does not provide a clear cut advantage."* ([Kha06], p.68)

The added value of modularity in identifying data components is contingent upon the challenge of establishing an appropriate granularity for modular decomposition, one that aligns with the inherent structure of the problem. Overly large modules result in a decomposition that is too coarse, risking the assignment of distinct data components to a single module. Conversely, excessively small modules lead to an excessively fine decomposition, potentially fragmenting a single data component across multiple modules. In both scenarios, the fidelity of identification would suffer. Leveraging expert knowledge from the application domain to guide modular decomposition can mitigate this risk. However, the likelihood of improper granularity increases during "blind" automatic modular decomposition, when a priori information is absent. [RSM04] employed artificial evolution to develop neural modules and observed the influence of evolutionary search granularity on the quality of the solutions obtained, stating: *"The trade-off (cost of increasing level of abstraction of mutation) is that the search becomes coarser. For example, mutations represent combinations of modular building blocks instead of weights, making it difficult or even impossible to find some solutions."* (p.70)

2.3. Modules in modular decomposition

The properties of the given application data mostly prescribe the choice of the suitable module type to model the separate components in the given data. This choice is crucial for the quality of the final model and needs to match the characteristics of the data in the application domain. Presence of dynamicity in the given data, temporal dependencies, or latency suggests the use of recurrent modules like RNNs, which were taken for modeling the sequential data in [XYH07; MC13; Jae07; BP11], whereas feed-forward neural modules are suitable for modeling non-latent data, like in [BFC96; Aza00; Jac+91; HIT02]. Other module types, like RNNs with spiking neurons in [BPM12] or generalized linear models in [CT06], can also be used to realize the particular requested functionalities in the application domain. Although the scope of each module is limited to modeling the corresponding data component, the choice of the module type predefines the scalability

of the entire network, having possible consequences for modularity and its virtue for understandable data representation.

2.3.1. Types of modules

The definition of the module stands in direct relation to the applied concept of modularity, whether structural or functional. According to the declared goals, the current thesis relies on structural modularity and, correspondingly, on the perception of the module as an entity with intensive signal processing inside it, sharing the produced signals with other modules via rare inter-modular connections. The modules, which have been described in numerous papers on modular networks, can be categorized by the form of their implementation into three main groups:

- *Feed-forward neural modules*,
- *Recurrent neural modules*, and
- *Parametric modules*.

Most studies try to benefit from the large potential and advantages of artificial neural networks for their modularization. In their presented modular architectures, separate neural networks are usually used for the implementation of individual modules. Thereat, the ANN type is chosen from among the variety of different neural networks to match the properties of the given data. The dynamicity of the data is the decisive attribute for considering either non-recurrent FF-ANNs or RNNs and offers a logical separation between recurrent and non-recurrent neural modules into two categories. Besides neural modules, there are studies that do not rely on neural networks. Instead, they define the behavior of the module explicitly, using analytical expressions, which suggests grouping them separately as parametric modules.

Feed-Forward Neural Modules. In Feed-Forward Neural Module (FFNM)s, the signal propagates strictly from the neurons of the input layer to the output neurons. In most studies about modular networks, FFNMs have been realized in the form of MLPs. The absence of recurrent connections in their structure makes them suitable for modeling non-sequential data. Having no temporal dependency between consecutive input vectors allows individual distribution of separate data samples among the available modules, with no need for prior segmentation of the given data into groups or sub-sequences. This leads to a straightforward reduction of spatial cross-talk and, subsequently, to an improvement in the total performance of the model. Nevertheless, FFNMs turned out to be applicable also to dynamic data, as demonstrated in [BFC96], where the input vector of an FFNM was a time-delayed sliding window to model sunspot-related time series. However, the potential of this modularization approach is rather limited because of the need to increase the ANN size for a deeper prediction horizon.

FFNMs are often deployed in heterogeneous modular architectures that consist of modules of different types and have different roles. In [GGK09], a gating FFNM was

directing the incoming data to the corresponding expert modules. As it was found in this study, the design of the gating FFNM had a significant effect on the total performance of the network. Therefore, the structure of the modules was optimized on the given validation sample to reach optimal performance. Differently, in [Jac+91; JJ92], the gating FFNM was directing the output signals in the other direction, from the expert modules towards the total output of the modular network. Besides, FFNMs can also serve as preprocessors of the data for other modules in the modular network, like in [Fab+20; FOB21], where an FFNM was used as the latent code layer for preprocessing the incoming data before feeding it to the main Long Short Term Memory (LSTM) module. An ensemble of independent FFNM modules was maintained and extended with new modules in [GOB17], where an FFNM was implemented as a compact one-layer FF-ANN to ensure fast online learning of event taxonomies in the simulated environment. In [GBM19], FFNMs were not directly linked to each other in the ensemble but had their own roles as behavioral, transitional, and error models in learning behavioral primitives by simulated robots.

The global nature of MLP output signals typically results in a globally consistent model when trained on representative data, exhibiting minimal deviations on unseen samples. However, this global presence can impede network extensibility, necessitating retraining of existing modules upon the addition of new ones. Conversely, employing FFNMs with RBF neural networks ([Kha06]) enhances modular architecture extensibility. The localized response of RBF networks simplifies the decomposition process, allowing module additions without disrupting existing modules. Alternatively, defining validity ranges for specific modules, as demonstrated in [GGK09], where a gating module directs dynamic data to dedicated experts, also ensures extensibility.

Recurrent Neural Modules. Recurrent Neural Modules (RNM)s, implemented with recurrent neural networks, are designed to process temporal and sequential information for dynamic data modeling. Their recurrent structure enables the storage of past dynamic signal evolution in a compressed hidden state vector, offering greater compactness compared to alternative time-delayed FFNM implementations: *"The combination of a high-dimensional multivariate internal state [of RNN] and nonlinear state-to-state dynamics offers more expressive power than conventional sequential algorithms such as hidden Markov models. In particular RNNs are better at storing and accessing information over long periods of time."* ([Gra12], p.1) This structural compactness of RNMs facilitates efficient encoding of dynamic information, leading to improved generalization and enhanced computational efficiency of the overall modular network.

While RNMs are powerful tools for representing dynamic data, their design is challenging. It necessitates the selection of an appropriate RNN type, the determination of a suitable module structure, and effective training. Any RNN type can be utilized for RNM implementation, provided its capabilities align with the nature and complexity of the given data. Designing the module structure typically requires substantial effort and is often automated through artificial evolution [Pas+01; RSM04] or random generation [Luk10; KOB21]. Subsequently, the weights of the RNM structure are trained for the

specific task. My research focused on recurrent networks capable of learning a variety of dynamics and reproducing any of them for a given parameter set. The stable reproduction of dynamics by a recurrent module depends on the presence of the *attractor* property. This property ensures that the module's dynamics remain on the target trajectory, regardless of its initial state. *Stable RNM*s, possessing either a *fix-point attractor* or a *periodic attractor* [Pas98], can reproduce their dynamics over extended periods without significant deviations from the learned target trajectory. In contrast, poorly designed RNMs exhibit instability, wherein their internal dynamics deviate from the learned trajectory, resulting in unexpected output signals. Measures to enforce stability can be implemented during the module's structural design or training and are typically specific to a particular RNN type. For instance, the stability of ESNs is ensured when the spectral radius of the hidden layer weight matrix is smaller than "1" ([Jae02]), and training with noise injection into the hidden neuron states further enhances ESN stability ([Jae01]).

All studies of modular RNN known to me dealt with stable RNM, realized with small RNN of standard sum-and-nonlinearity neurons for robot control [Pas+01], with RNN of leaky-integrator neurons for time series modeling [AT17], or with RNN of spiking neurons for controlling the muscles of a 2-dimensional robot arm [BPM12]. In my research, I utilized such modules for the identification of oscillatory dynamics within given time series data. While most studies ([WSS08; RS12]) considered instability in recurrent dynamics an undesirable property to be eliminated, this likely explains the limited exploration ([BN04; OP05]) of its potential benefits. In my thesis, unstable RNM, and those with dynamics at the edge of stability, were employed for modeling handwritten symbols. Despite their inherent instability, they exhibited locally stable dynamics sufficient to generate the necessary temporally-limited, aperiodic signals for drawing symbol curves.

Inspired by the remarkable performance of ESN in modeling chaotic time series, such as MG sequences [GM09; VS09; Jae01] or the Lorenz attractor [JH04], researchers explored the benefits of their modularization. [Jae07] trained a hierarchical modular ESN to model non-stationary artificial data with dynamics switching between three sources. In [XYH07], ESN modules were linked via a lateral inhibitory mechanism to decouple them. This modular ESN was applied to model mixtures of dynamic components and stochastic time series from turbojets. This time series was also modeled in [BP11], which combined multiple ESN modules to implicitly estimate unknown memory lengths: *"The prediction error achieved by this approach was substantially smaller compared to prediction error achieved by standard Echo State neural networks."* This approach resembled that of [BFC96], which also aimed to implicitly estimate the unknown sliding window length. [Pat+18b] presented a junction of two modules of different types: an ESN module and a knowledge-based module, designed to compensate for potential imprecision of the ESN module.

Instead of connecting separate ESN modules in a modular network, some researchers pursued the concept of *sub-reservoirs*, which involves dividing a single dynamic reservoir into groups of reservoir neurons that exhibit distinct dynamics with minimal or no signal

exchange between them, and are connected to a common output layer. In [Pat+18a], a given spatiotemporally chaotic Kuramoto-Sivashinsky time series was distributed among several reservoirs, each responsible for modeling a local region of the data range; the output signals of adjacent reservoirs overlapped. In an attempt to determine the proper topology of the dynamic reservoir, it was incrementally enlarged by groups of reservoir neurons in [Yan+18]. The resulting network achieved optimal size and performed well on chaotic attractor time series, as well as in predicting chemical substance concentration in wastewater treatment processes.

Parametric modules. Parametric modules are those whose output is computed according to parameterized mathematical formulae, which can be defined using analytical expressions, logical rules, or parameterized probability density functions. In essence, an entire network of parametric modules can be represented as a composite mathematical expression, comprising simpler parameterized forms of the individual modules. However, depending on its complexity, its topological representation can provide a better basis for analyzing the observed behavior of the network's output signal and for understanding the dependencies within the model.

In the literature, parametric modules are frequently considered within a probabilistic framework to define parameterized probability density functions that model the probabilistic behavior of observed processes. In [JJ93; JX95], the Hierarchical Mixture of Experts (HME) was employed to model the total probability density of observation data for solving regression and classification problems. The model aimed to decompose the input space into a set of nested regions and assumed a specific class of probability density functions, such as multidimensional Gaussian distributions. Within the hierarchy, expert and gating modules were defined as generalized linear models, producing parameters for the chosen probability density functions. [PJT95] also utilized generalized linear models as parametric components in HME, presented a comprehensive Bayesian analysis, and applied it to vowel recognition with multiple speakers. An advanced version of the generalized linear model with recurrence was used in [CT06] for predicting *switching processes*, whose underlying dynamics abruptly changed over time while remaining stationary between transitions. [ZMA99] proposed using local autoregressive models for modeling time series in the form of mixture distribution, where the behavior of each local autoregressive model was parameteric with respect to the covariance matrix and mean vector.

2.3.2. Training and adaptation of modules

A survey of the literature on modular networks revealed several approaches to preparing modules for deployment in modular networks. Here, they are presented in groups categorized by module type, training and adaptation methods, and the dynamicity of available training data. In most cases, the selected module type is intrinsically linked to its commonly used training approach. This typically involves training connection weights between neurons in feed-forward neural modules. For recurrent modules, the

module update extends to neural activities responsible for precise generation of target output signals.

Training of FFNM. Gradient-based training with Error Backpropagation (BP) is the most widely used method for training FFNM. In [JJB91], it was used for simultaneous training of expert modules in the Mixture-of-Experts architecture, where the error function was defined as the Mean Squared Error (MSE) of the discrepancy between the given training data and the network output. [Jac+91] defined the error function as a sum of weighted errors per module, encouraging competition among local expert modules rather than cooperation. A further modification of the error criterion, including both the total error and the error of the currently trained module, was proposed in [BFC96] and yielded competitive results on the Sunspot time series prediction task. To strengthen competition among FFNM modules compared to [Jac+91], studies [JJ91; JJ92] presented a training procedure based on gradient ascent for maximizing the log-likelihood function, which resulted in a tendency for only a single module to learn each training sample. However, maximizing the likelihood function did not clearly outperform the BP method, providing performance gains only on small and medium-sized tasks. [KS13] reported a significant speedup of the training procedure by introducing the sibling-descendant cascade-correlation method for greedy learning of neural modules, achieving convergence orders of magnitude faster than standard BP.

Adaptation of FFNM. Changes in the environment or in the structure of neural modules necessitate adaptation of module weights to maintain or enhance model accuracy. [HIT02] addressed non-stationary problems requiring model adaptation. This was accomplished through gradient-based batch learning to fine-tune the weights of evolved neural networks. Based on defined modularity indicators, the authors concluded that fine-tuning influenced the final model accuracy but was inconsequential for the emergence of modularity. [Kha06] employed gradient-based training to adjust weights following structural changes in modules, such as deletion of some neurons, and for multiple training of the modules *"with different weight initializations, before fitness evaluation to average out the fluctuations in fitness."* This study indicated that the choice of training algorithm was inconsequential for modular network performance on simple static and dynamic tasks, whereas a strong dependency was observed for incrementally complex dynamic tasks. [GOB17; GBM19] presented a model that was regularly augmented with small, single-layer FF-ANN modules during online learning of continuous information flow, and subsequently applied stochastic gradient descent and Recursive Least Squares (RLS) for weight training after each model extension. Comparison of both methods revealed that both were applicable for rapid learning, but *"but RLS-based learning quickly reaches a much better prediction"*.

For the realization of Bayesian inference, [KS13] presented a modular topology with neural modules for modeling posterior and prior probability distributions. By defining output-to-input connections, this topology enabled incremental adaptation of priors using posterior values from the preceding time step. In [RSM04], mutation of neural weights with a small normally-distributed value was employed for their adaptation

to structural changes in modules during artificial evolution. A genetic algorithm was applied for training a dual ANN in [NP95].

Training of RNM. Recurrent modules are trained using standard procedures typically applied to monolithic neural networks of the same type. Thus, training of LSTM modules is performed with the iterative gradient-based BackPropagation Through Time (BPTT) method ([Fab+20; FOB21]). In contrast to other RNN types, training of ESN modules ([MC13; Pat+18a]) employs a non-iterative linear regression algorithm, which does not rely on gradient information ([Jae01]). Augmenting this linear regression procedure with a regularization term ([WSS08; Pat+18b; Yan+18]) results in smaller weights, preventing potential ESN instability.

Despite the faster convergence of gradient-based methods, their greedy nature can lead to suboptimal weight combinations. The potential to enhance the performance of trained modules motivated several researchers to consider artificial evolution for training of recurrent modules. Similar to gradient-based training, it iteratively searches for optimal weight combinations but can mitigate suboptimal weights by maintaining a population of candidate weight vectors and by relying on randomness when generating new weights by modifying current ones with random values drawn from a probability distribution, such as the Cauchy distribution in [Sch+07; GSM08]. In [Pas+01], new weights were generated by perturbing them with uniformly distributed random numbers. This study featured interactive evolution involving human participation to improve the quality of the solution. However, artificial evolution does not consistently guarantee superior performance; as concluded in [Sch+07], the quality of the final solution is problem-dependent and not always significantly better than that found by gradient-based methods.

Synchronization of dynamic modules involves online adjustment of constant parameters or internal transient states of dynamic modules to compel the modular network to produce the desired target output dynamics. The necessity for adjustment arises from either environmental non-stationarity or new goals for the dynamic system, as observed, for example, in [INS02] for the presented control of a humanoid robot: *"interactions with the environment may require an on-line modification of the [control] policy."* (p.3) This adaptation can leverage available error signals: *"resulting error signals can yield both neural activity adaptation and structural weight adaptation"* ([But16], p.9). For RNM, this entails either adaptation of their weights and neural activity or solely the adaptation of neural activity, assuming the weights were appropriately trained beforehand. Adaptation of constant parameters is then required for non-recurrent modules that do not possess internal storage but receive time-dependent input from external sources.

The stochastic gradient descent method was employed for adapting internal states in a hierarchy of ESN modules in [Jae07]. In this study, the error signal did not solely rely on the model's prediction error but involved outputs from modules at all hierarchical levels. In [AT17], the internal states of RNM in a modular multiple-timescales RNN were tuned online to minimize prediction error through error regression (self-synchronization) on a sliding window of recent past data. In *closed-loop generation* mode, outputs from

preceding time steps were fed back as inputs to the modular network, equivalent to the autoregressive mode of neural oscillator modules. In experiments with humanoid robotics about imitative interaction with human subjects, high performance was reported but exhibited significant sensitivity to parameter selection and experimental conditions. Self-synchronization was also demonstrated in [ORB19] with an ensemble of neural oscillator modules on compositional dynamics. The modules were implemented with standard ESN, enabling the use of teacher-forcing (see Appendix A) to induce the required recurrent dynamics within the modules. During teacher-forcing with a sequence of estimated target values, the dynamics of each pre-trained neural oscillator settled on the desired trajectories. Since the actual target sequence was a priori unknown for any module, the estimated values were inherently imprecise. Nevertheless, due to the presence of attractor characteristics, the modules successfully converged to their target dynamics and precisely reproduced the compositional time series over extended periods. Although greatly benefiting from the properties of ESN modules, self-synchronization was not observed with LSTM modules.

Expectation-Maximization for neural and parametric modules. [Jac+91] proposed the Expectation-Maximization (EM) algorithm for iterative parameter estimation in Mixtures of Experts (ME) and experimentally demonstrated its efficiency compared to gradient-based methods on the vowel discrimination task. In [JJ93], the parameters of the probabilistic model were incrementally updated using the EM algorithm, whose efficiency benefited from the relatively small number of adjustable parameters, and convergence could be analytically proven due to the rigid mathematical form of the modules in the network. Further theoretical analysis of this algorithm, including convergence analysis, was provided in [JX95]. Several subsequent works proposed modifications to the original EM approach and conducted analyses under different assumptions. [Aza00] proposed the evidence maximization framework as a more robust variant of the EM algorithm, with explicit reward assignment to the experts for training a modified HME using alternate RBF. [PJT95] defined mixture coefficients and mixture components as generalized linear models for full Bayesian analysis of ME and HME models. EM achieved high classification rates but required multi-start initialization because of the multimodality of posterior distributions based on mixture models. Beyond ME and HME, [ZMA99] applied EM for training mixtures of local autoregressive models to mitigate excessive complexity in training criteria. To ensure convergence of training procedures for Dynamic Bayesian Networks within a reasonable timeframe, the use of Markov Chain Monte Carlo methods instead of EM was suggested in [RH10].

2.4. Topology of modular decomposition

Clear benefits of modularity are evident in domains where available data possesses an underlying structure and can be decomposed such that modules become responsible for modeling specific components within the application data. In general, relations between modules in the network must replicate dependencies between components in the data

and are encoded by the topology of a modular network, which subsequently represents structural information about the problem. *"The more we are able to exploit this structural information the clearer the advantages of modularization should be."* ([Kha06], p.66) Any a priori knowledge about the problem, including knowledge of relations between data components, is crucial for selecting a modular topology with respect to the type of dependency among modules and their impact on the overall network output. A survey of the literature on modularity identified the following primary variants of modular network topologies:

- *Gated combinations of modules*
- *Linear combinations of modules*
- *Stacking of modules*
- *Hierarchies of modules*
- *Networks of interdependent modules*
- *Dynamic topology*

Gated and linear combinations of modules presume independent data fractions within the underlying data structure, which can be represented with a single-level tree topology without direct connections between modules. Consequently, this presumption is also significant for selecting relevant application areas for experimentation, as in [HIT02], which considered modularity within the scope of problems decomposable solely into independent sub-problems: *"We call a problem modular if it can be solved by a number of non-interacting subsystems."* (p.221).

Topologies involving stacking or hierarchies of modules implement subordinate dependencies between modules. The ability of stacking topologies to smooth out fine details in internal signals propagating over layers from input to output ("bottom-to-top" direction) suggests their applicability for differential modeling of data features at different resolutions or time scales [Jae07; AT17]. Stacking of modules has also been employed for input data preprocessing by low-level modules, providing suitable data representations for high-level modules [LKK04; FOB21]. Hierarchies of modules expand the concept of stacking to encode the dependency of a single high-level feature on multiple lower-level features. Such dependencies imply the simultaneous existence of multiple features in the data at the same resolution, necessitating the definition of corresponding modules at the same hierarchical level with connections to a higher-level module. Their application for decomposing input space into nested regions was demonstrated in [JJ93].

The presence of non-stationarity in the modeled underlying data structure necessitates the ability to vary modular topology over time, a capability provided by networks of interdependent modules and networks with dynamic topology. In networks of interdependent modules, actual signal propagation paths are contingent on the current activity of modules, which can change over time. Decreased activity in certain modules weakens signal conductivity through them, effectively disabling the corresponding

part of the modular network. Dynamic topology networks implement time-varying dependencies between modules to directly disable superfluous connections within the network.

Gated combinations of modules. This modular topology posits that not all available modules are consistently required for accurate decomposition of given data. Their contribution to the overall network output can vary across the feasibility region and is regulated by an additional gating unit, which computes the necessary module weights based on the input signal. The ME architecture of [JJB91; Jac+91] was designed for the decomposition of non-sequential data. It comprised an integrating unit, a gating module, and expert FF-ANN modules. The integrating unit computed the total network output as a weighted sum of the independent expert modules' outputs; the corresponding weight coefficients were determined by the gating module upon receiving the external input signal.

Gated combinations of modules were also utilized for the decomposition of dynamic data in [BFC96] with contributions from FFNM and in [BP11] with contributions from RNM implemented using standard ESN. Both studies aimed to average the performance of the combined modules to enhance the overall network accuracy compared to the accuracy of individual expert modules, which were deemed insufficient for modeling the given data independently. In [Pat+18a], multiple parallel-operating ESN dynamic reservoirs were combined with output weights in the ESN output layer, which functioned as a gating entity.

The study [GGK09] presented an alternative variant of gated topology, where gating was applied to the externally provided input signal, instead of applying it to the outputs of expert modules as implemented in ME. Based on heuristic rules, the gating module exclusively activated the most relevant expert module and forwarded the input signal to this selected expert for processing and computation of the total network output. Thus, the gating module implemented a split of the input data space, where each expert module was responsible for modeling its respective data fraction. Gating with exclusive activation of a single most relevant expert module is also applicable in ensembles of independent modules ([GOB17]). In this scenario, the gating unit is not explicitly present in the modular architecture but is algorithmically implemented to activate the module according to a predefined criterion. In [GOB17], a new module was created, trained, and activated when none of the currently available modules could sufficiently model the incoming data flow, i.e., if all modules produced an unexpectedly large error.

Linear combinations of modules. Linear combinations of modules are a specific case of gated combinations of expert modules. In contrast to the latter, there is no gating network controlling the contributions of modules to the overall network output; contributions are permanent, with strengths defined by weight coefficients or module responsibilities. Linearly combined modules are particularly suitable for describing data whose components coexist within the underlying data structure and are independent of each other. The absence of direct connections between modules is advantageous for decomposing

dynamic data using dynamic modules, as it eliminates the risk of forming looping circuits with destructive positive feedback, which can lead to network instability. However, the lack of direct dependency between modules does not imply diminished representational power of the entire network. Potential needs for higher complexity can be addressed through significant non-linearity within individual modules and their deep recurrence. The simplicity of this topology has demonstrated advantages in model performance and compactness in several previous studies. [Sol02] combined a set of FF-ANN for decomposing time series with frequency range filters. The weighted sum of generalized linear models was investigated in [CT06] for modeling switching processes. A compound probability model was defined in [ZMA99] as a mixture of local autoregressive models with probabilistic weights, also for modeling non-stationary switching time series. In my thesis, such time series were modeled using linear combinations of parametric and recurrent neural modules. Similar to the ESN modules in this thesis, [MC13] utilized recurrent modules for decomposing given sequence data with Modular State Space ESNs, which, however, defined modules in terms of output dimensionality rather than as separate ESNs.

Stacking of modules. Stacking of modules entails serially connecting modules, where the output of the first module is chained to the input of the second, which in turn provides the input signal for the third module, and so forth. Given that the input signal of the first module undergoes transformations as it propagates through the modules' chain, the purpose of stacking modules is either to decompose the input data into fractions or to transform it into a required form for subsequent processing. The ability to decompose data into fractions was utilized to identify different time scales in given time series in [Jae07], where multi-scale dynamic features were discovered in an unsupervised manner, and in [AT17], where sequences with fluctuations were predicted for interaction with a humanoid robot. In both studies, algorithm convergence significantly benefited from the attractor characteristics of the stacked RNM.

In [FOB21], the task of handwritten character trajectory recognition was distributed across two serially connected layers: a feed-forward neural layer and a recurrent LSTM layer. The feed-forward layer learned exemplar trajectories, which were subsequently processed by the LSTM. The preceding feed-forward layer proved highly beneficial *"to recombine attractor dynamics in the RNN layer to quickly learn to generate untrained characters"*. The distribution of the overall task across three neural modules was proposed in [Sha97b] for robot control, specifically to address the disembodied arm problem (where the robot arm and the mobile robot were decoupled). For this purpose, classifier and arm modules were designed to develop and perform suitable data transformations for a third module to further learn decentering of transformationally invariant representations.

Furthermore, hybrid variants such as [Pat+18b] combined a stacking topology with a gated combination of analytical knowledge-based and ESN modules for time series prediction. Along one information processing pathway, they were stacked such that the knowledge-based module provided suitable representations for the ESN module. Along the other pathway, their outputs were combined to compensate for potential imprecision

in each, resulting in a drastic reduction of total model size and a smaller prediction error than if the models had been applied to the data sequence independently.

Hierarchies of modules. Hierarchies of modules can be considered as an extended variant of gated module combinations for implementing nested "several-to-one" dependencies between modules. Hierarchies are represented by tree-structured formations, where modules are positioned at nodes and arranged in layers. Modules within the same layer operate independently. Dependent modules reside in different layers; the dependency direction aligns with the signal propagation direction within the tree. In [Kor01], visual patterns were input to a hierarchical pattern classifier at the tree's root and propagated for finer classification towards classifier modules at the leaves. This implied the dependency of leaf modules on the root module. The reverse direction was implemented in HME, which are based on the gated ME topology but feature multiple layers. Similar to ME, integrating units in HME compute linear combinations of module contributions from the preceding (lower) layer, with weights provided by the gating module. Various studies have explored different types of expert modules. For instance, [JJ92; JJ93; JX95] demonstrated the use of generalized linear models and conducted extensive studies across diverse applications. [Aza00] implemented gating modules using RBF networks and highlighted the suitability of hierarchies for system comprehension, as hierarchical levels explain the layered information processing within the modeled system.

Few studies have been published on hierarchical topologies that incorporate recurrent dependencies between layers or utilize recurrent modules. One such study is [CAB17], which introduced the Multiscale Hierarchical RNN. It was motivated by the persistent challenge of RNNs to learn both hierarchical and temporal representations with increasing levels of abstraction, akin to the human brain. The proposed modular network outperformed standard LSTM on handwriting sequence generation. In [KS13], a hierarchical topology was explored to implement Bayesian inference. Neural modules were hierarchically organized and dedicated to distinct terms in Bayes' rule. The top-level module aggregated outputs from lower-level modules to compute posterior probabilities and update prior probabilities via a feedback connection. Weight decay was observed during learning and used to gain insight into Bayesian reasoning in the human brain.

Networks of interdependent modules. In these networks, modules exchange information among themselves and are, therefore, well-suited for modeling complex humanoid robot behavior, representing intricate social relations, or developing cognitive models. These networks frequently comprise modules of diverse types, each dedicated to a specific function and providing information to other modules. [HM94] defined a modular network with interconnected Categorization and Learning Modules (CALM) to explore the relationship between structure and function in artificial and natural networks. [BPM12] demonstrated how complex behavior can emerge from the interaction of a small number of simple spiking neuron modules for the realization of complex movements. [Pas98] investigated general discrete-time dynamics and concluded that modules can acquire their unique functionalities through interaction with other modules: *"They [interacting modules] receive their specific functional properties only during cooperative or*

competitive interactions with other modules of the system." (p.16).

The advantage of interdependent modules for modeling comprehensible entities and diverse logical relations among them is of significant interest in cognitive modeling for handling structured knowledge and behaviors. [GBM19] implemented the interplay of modules of diverse types – behavioral, transition, and error modules – with the aim of learning modular structures for goal-directed behavioral control. [Bat+18] introduced the Graph Network (GN) for appropriate description and handling of relational reasoning as graph-structured representations of compositional knowledge. Similar to the modular structure explored in this thesis, GN supports generality without adhering to a specific form of module implementation, whether neural or parametric, and enables the integration of a priori knowledge about the environment by adopting specific architectural assumptions to achieve proper structuring of knowledge and behaviors.

However, the interdependency of modules contributes to increased complexity in these networks, which is necessary for many problems. As stated in [Kha06]: *"many problems can only be decomposed into subcomponents with complex interdependencies"* (p.4). Nevertheless, high complexity is undesirable as it compromises the advantages of modularity for comprehending the underlying data structure. This necessitates limiting network complexity by imposing constraints. In most studies, this is implicitly achieved by defining the roles of interacting modules within the network, resulting in improved overview, simplified tuning of individual modules, and enhanced understanding of the overall network.

Dynamic topology. To extend the abilities of modular networks beyond the limitations offered by various static topologies, some studies have explored the more flexible, yet more complex, concept of data-driven dynamic topologies, where modular content and topology vary based on incoming data. This concept expands upon the definition of dynamic structures initially proposed in [Hay94]. According to this definition, a modular network is considered dynamic if it includes an integrating unit that combines the outputs of individual modules and regulates the total network output in response to external input signals, as implemented in ME or HME.

Flexibility in modular topology is essential for non-stationary applications where dynamic data content can vary over time, as seen in recent studies. [RH10] introduced Non-stationary Dynamic Bayesian Networks with time-varying dependencies between modules, applicable for identifying transient neural information flow networks in biological organisms. A Bayesian-based method was presented in [BBE11] to segment piecewise stationary time series by incorporating modular network structure for complex system dynamics identification. Their idea for understanding of complex dynamics was based on identifying a sequence of relevant models, as *"if we were able to find the highest probable sequential states of the random finite sets Net_1, ..., Net_k, given the data, where Net_k shows the state of the underlying network at time k, we could understand the dynamics of a complex system."* (p.58). However, excessive dynamicity in incoming data can lead to high volatility in dynamic networks, making it difficult to track structural changes

and potentially diminishing the models' utility for identifying constituent dynamic components. Constraining variations in network topology is a straightforward method to maintain traceability of changes in modular representation. In this thesis, this constraint is implemented by maintaining a predefined topology that connects independent modules, whose number and parameters can still vary over time based on incoming data.

2.5. Approaches to modular decomposition

The objective of modular decomposition is to define a modular architecture either through reliance on available expert knowledge or automatically according to predefined criteria. As stated in [Aza00]: *"task decomposition is the most important step in designing modular neural networks."* (p.41). In [Sha97a], decomposition methods were categorized into two groups: explicit methods involving data partitioning by experts, and automatic methods performing data partitioning according to predefined criteria. However, this categorization does not account for the varying degrees to which expert knowledge can be leveraged in decomposition. Therefore, I propose a refinement of explicit decomposition, resulting in the following three categories:

- *Expert knowledge-based decomposition*
- *Semi-automatic decomposition*
- *Automatic decomposition*

In this refined categorization, expert knowledge-based methods are entirely dependent on data partitioning by experts, aiming to derive a modular network that satisfies application domain requirements. Semi-automatic methods perform automatic decomposition of incoming data based on initially provided expert knowledge. In automatic decomposition methods, data partitioning and model learning are executed in parallel, as originally proposed in [Sha97a].

2.5.1. Expert knowledge-based decomposition

In expert knowledge-based modular decomposition, training data is explicitly decomposed by experts who select the module type and modular topology. Module parameters are then adjusted using external tools and deployed within the designed modular architecture to achieve high performance in the application domain. In [Kor01; LKK04], training data was externally prepared to train feed-forward neural networks outside the hierarchical classifier of visual data patterns. The resulting system achieved high recognition rates while also providing real-time capabilities for traffic sign recognition.

The study [GGK09] explored the feasibility of constructing a modular neural network using financial data related to option pricing. Training data was partitioned into distinct

subsets – one subset per option type – based on available expert knowledge concerning moneyness and time-to-maturity. Subsequently, these subsets were employed for independent training of feed-forward neural modules within the modular architecture, aiming to enhance the model's generalization properties by addressing the issue of spatial crosstalk. Spatial crosstalk occurs when *"unit receives inconsistent training information at a single instant in time"* ([JJB91], p.222), such as after presenting contradictory training patterns from different data regions to adjust the same weight.

A deeper expert understanding of the application domain is necessary for designing dedicated module functionalities, like frequency range filtering entities in [Sol02], for explicitly defining relations between module pairs, or for satisfying potential physical constraints, e.g., hardware limitations in embedded realizations of modular networks, as noted in [JJB91]: *"suitably designed modular architectures can reduce the number of units and the lengths of connections."* (p.226).

2.5.2. Semi-automatic decomposition

Semi-automatic decomposition leverages the opportunity to incorporate available knowledge from the application domain to facilitate more efficient modularization. In the initial phase, knowledge is provided by experts through (1) defining rules [KS13], (2) predefining modular topology, often based on assumptions about the properties of given training data [RH10], (3) defining relationships between modules [Bat+18; BFC96; CT06], or (4) implicitly integrating knowledge into the modular structure via embedded attractors [AT17] or explicitly by incorporating dedicated knowledge-based modules [Pat+18b]. Subsequently, the decomposition method extracts latent modularity from the given data by learning trainable parameters of the selected modular architecture. This learning phase is a distinguishing characteristic of semi-automatic decomposition, differentiating it from pure expert knowledge-based decomposition.

Relationships between modules can be more complex than simple weighted connections. [XYH07] introduced Decoupled Echo State Network (DESN), where modules were separated by a lateral inhibition mechanism requiring careful expert design to prevent degrading changes in DESN operation. This involved setting critical parameters, whose variation had a significantly stronger impact on model performance than, for example, variations in neural module size. Upon designing the modular architecture, training was conducted using the standard ESN training procedure, described in Appendix A.

In the context of imitation learning [AT17], a priori knowledge was implicitly provided to the system in the form of sequences representing prior-learned prototypical movement patterns, each with its own local attractors and transient regions. These patterns were subsequently learned during training and utilized to predict fluctuations in later observed sequences. This network architecture implemented functional modularity with a number of stacked layers, each encapsulating local attractors and transient regions at different time scales.

The modular architecture in [Jae07] was employed for the automatic discovery of dynamic components within the input signal. A hierarchy of stacked ESN modules decomposed the given data into a set of decorrelated non-linear signals. Instead of trainable real-valued output weights, the output layer of the higher-level ESNs implemented a voting scheme with votes defined as functions. Compared to standard ESNs, this provided greater flexibility in responding to changes in incoming data but required expert knowledge to design suitable ESN modules, conduct their training, and appropriately set training parameters.

Decomposition of temporal sequences also motivated the development of the Multiscale Hierarchical RNN based on LSTM ([CAB17]). In this approach, expert knowledge was integrated into the designed hierarchical modular architecture. An embedded boundary detector identified segment boundaries of different lengths, triggering the update of available modules for learning. This captured multi-scale temporal features within the given sequence data and presented them in the form of a latent hierarchy. In general, an expert's decision regarding deployed module types and chosen relationships between them establishes a structural bias for learning, enabling more efficient data decomposition than if modules and modular topology were learned from scratch. Additional constraints can be imposed to speed up component separation within the given data and minimize the risk of unsatisfactory solutions.

In the field of cognitive science, [Bat+18] relied on the implicit incorporation of expert knowledge in the form of specific architectural assumptions. Based on these assumptions, the approach derived a modular representation of the surrounding environment, structured into categories with logical relations among Graph Network (GN) modules. The approach distinguished itself through effective scaling, which maintained model interpretability despite increasing environmental complexity: *"The entities and relations that GNs operate over often correspond to things that humans understand (such as physical objects), thus supporting more interpretable analysis and visualization."* ([Bat+18], p.24). It demonstrated the advantages of modularizing deep learning architectures for cognitive modeling, compared to pure "end-to-end" learning, and argued that this would lead to the emergence of compositional generalization as an integral feature of AI.

2.5.3. Automatic data-driven decomposition

Data-driven automatic decomposition is the most user-independent approach, as it assumes no expert knowledge is explicitly or implicitly available for extracting structure from the given data. The number of required modules [New06], the functionality of individual modules, and the modular topology [Aza00; RSM04] are automatically determined by the decomposition method to satisfy predefined performance criteria and constraints [JJB91; HM94].

No automatic modularization method can guarantee the retrieval of the true underlying data structure; as stated in [HM94]: *"We know of no comprehensive analytical solution to the*

problem of relating architecture to function." (p.16). Nevertheless, the appeal of automatic modularization lies in minimizing human effort for configuring the modular architecture. Users are only required to define criteria for evaluating the quality of the derived decomposition and to constrain the search space of candidate decompositions. These constraints pertain to specific properties of the resulting model, such as its complexity or stability, and do not necessitate knowledge of application domain laws or rules. For instance, limiting the maximum model size is advisable when the decomposition method has the potential to inflate the modular topology for increased granularity in the resulting decomposition [RSM04; Yan+18].

Conversely, each imposed constraint must be justified to avoid hindering algorithm convergence and to ensure a satisfactory quality of the resulting decomposition. Specifically, these constraints must not contradict the principles of modularity to leverage its advantages. For instance, imposed constraints should not impede module compatibility, thereby preserving the ability to flexibly replace currently linked modules with newly trained alternatives that more accurately represent hidden data components. In recent decades, ANNs have gained popularity and been utilized as modules for problem decomposition in numerous studies [HIT02; BPM12; FOB21]. Their capability to model diverse data types without requiring expert knowledge has made them appealing for automatic decomposition; thereat, the complexity of individual modules can be easily controlled by varying neuron count. The ability to train distinct modules for each data partition enhances the flexibility of fully automatic decomposition. Although automation minimizes human effort, the concurrent search for modules and decomposition topology increases algorithmic complexity, which can become computationally expensive and, under certain conditions, exhibit poor convergence. Automatic decomposition methods can be categorized into two groups: *evolutionary* and *non-evolutionary* methods.

Evolutionary methods. These modular decomposition methods are sometimes referred to as selective methods because artificial evolution selects suitable candidate solutions from a population for further propagation. Since evolutionary algorithms primarily rely on given data and do not require expert knowledge, they have been considered in numerous studies for the automatic design of modular networks [HM94; Pas+01; HIT02; RSM04; Yan+09] and also for automatic non-modular network design [YL98; PD00; Pol+07]. The diversity of evolutionary approaches for network design stems from the variety of possible evolutionary schemes explored across various scientific and engineering disciplines. [Jun07] presented a study on an evolutionary method for designing modular ANNs using a descriptive encoding language. [Pas+01] introduced interactive evolution for developing modularity in robot control for non-stationary environments. The co-evolutionary method of [Kha06] was applied to decompose diverse data types using networks of feed-forward radial basis function modules. Co-evolution involves species from multiple populations adapting to each other during the search for improved candidate solutions. Evolutionary development motivated the architecture in [BPM12], featuring inflexible peripheral synergies between modules. In [HM94], genetic algorithms were employed to search for optimal modular network

topologies, which underwent successive tuning for visual categorization of drawn objects. [RSM04] demonstrated the extraction of reusable modules by a co-evolution algorithm, which decomposed problems into simpler sub-problems during evolution in a scalable board game domain.

Evolutionarily designed networks must satisfy a predefined optimality criterion, which guides the evolutionary algorithm towards the desired final decomposition. The criterion minimizes the discrepancy between the given data and the model's output. However, relying solely on this criterion leads to the automatic design of complex, irregular topologies that are difficult for humans to interpret, particularly when individual modules are also subject to automatic design. This indicates that achieving interpretable representations necessitates the imposition of additional criteria or the appropriate restriction of the search space. In this respect, automatic data-driven decomposition methods get the traits of expert knowledge-based decomposition methods.

Imposing constraints is an efficient method to ensure convergence of the modularization method towards a desired solution. A literature analysis revealed the following constraint types:

- *Initial constraints* define ranges and distributions for initializing the modular representation ([YL98; Kha06]).
- *Operational constraints* define limits on network signals and dynamics or their parameter values during decomposition ([Sch+07; PD00; YM07]).
- *Architectural constraints* define limits on the structure of the modular network, specifying available resources for network construction, such as module types and connection types ([Pol+07; Jin07]).

Regardless of the application domain, constraints facilitate improved generalization and desired data decomposition. As stated in [HM94], function decomposition is an underconstrained problem, leading to *"networks with too many degrees of freedom [that] may not generalize as desired."* Thus, appropriate constraints are essential. *"Such an architecture should generalize in the desired manner."* (p.26)

Non-Evolutionary methods. Non-evolutionary methods derive modular representations of given data through techniques that do not select suitable solutions from candidate populations or modify them using genetic operators like mutation or crossover. Instead, they develop specialized operators to alter modular topology, as demonstrated by [Aza00], who introduced a constructive vertical scaling method for building module hierarchies.

The objective of most non-evolutionary approaches is to distribute given data among available modules, ensuring each module is responsible for its own data partition, as stated in [JJ91]: *"The technical issues addressed by the modular architecture are twofold: (a) detecting that different training patterns belong to different tasks and (b) allocating different networks to learn the different tasks."* (p.768). In this study, available modules competed

for distinct data samples from a static sample set. The “won” data samples were then automatically allocated to corresponding “winner” modules for training. Similarly, this principle is applicable to dynamic data, as demonstrated in [GOB17], where incoming information flow was automatically split into subsequences, which were then assigned to separate learning entities. Segmentation was performed based on an available model set, which was expanded by a new independent feed-forward model if none of the existing models provided an accurate prediction.

In [MC13], the decomposition of dynamic data was performed implicitly. Their modular architecture consisted of a linear combination of a predefined number of ESN modules; each defining a hyperplane within the reservoir’s state space. Consequently, the combination of hyperplanes partitioned the high-dimensional state space into distinct regions. The resulting modular network exhibited high modeling accuracy, even on chaotic attractor sequences of the MG and the Lorenz attractor. However, it represented functional modularity, which was not readily applicable for identifying potentially existing hidden dynamic components.

2.5.4. Evolutionary-pruning decomposition

This thesis proposes and investigates a data component identification approach that can be categorized as semi-automatic decomposition, where expert knowledge is incorporated into the decomposition by manually designing available modules and predefining the modular topology. When applied to given sequence data, it automatically prunes the initial modular representation by identifying and removing irrelevant modules. This process maintains the predefined topology, preserving the overall interpretability of the resulting model. The initially over-sized network can be viewed as a template from which the algorithm "presses" irrelevant modules out of the scope while retaining relevant ones. Somewhat similar concepts were explored in [Kha06], where multiple potential templates were used to initialize the input vector, and in [RSM04], which utilized a population of blueprints. In contrast, constructive methods increase model size to enhance performance; for example, in [Qia+16], the size of a growing modular ESN was augmented by connecting additional sub-reservoirs.

Similar to evolutionary methods, this approach maintains a population of candidate structures that encode diverse combinations of dynamics of available modules. However, the incorporation of available expert knowledge allows for the use of alternative operators, rather than relying solely on genetic operators, which may be more suitable for the given application data.

3. Modular problem decomposition through synchronization

This chapter introduces a framework for the online synchronization of dynamics within an ensemble of co-evolving modules. This includes an algorithmic description of the method and a mathematical formalization of the synchronization goal: to distribute given sequence data among the modules to achieve its decomposition, where each module models a distinct data segment. The ensemble's content is expected to asymptotically converge to the true, yet unknown, content of the time series. The dynamics of irrelevant modules will attenuate and become inactive by the end of synchronization. Consequently, the active modules in the finally tuned ensemble will identify hidden dynamic components of the time series and indicate their parameters. The general synchronization framework relies on an optimization method but is not limited to a specific one. This thesis analyzes the impact of incorporated optimization methods – both evolutionary and gradient-based – on convergence of synchronization under various conditions to highlight their respective advantages.

Possessing some a priori knowledge about the analyzed sequence data should be regarded as a precondition for applying the synchronization approach. This includes assumptions about the nature of components within the time series and their interrelationships. Knowledge about the nature of hidden dynamic components is expected to be available in the form of small dynamic modules that possess some degree of flexibility and are capable of varying their dynamics within a defined range. Similar concepts for data decomposition based on a priori knowledge, in the form of initially available modules, were previously presented in [GGK09; Sol02] with FFNM and in [HM94] with data stream modules. Crucially, proper module design and training are paramount; as emphasized in [FOB21], *"showing how important the training was and supporting our hypothesis that sequence components are learned that can later on be recombined in a compositional manner."* (p.529). Assumptions about sequential data characteristics support the selection of appropriate module types, while assumptions about time series complexity determine the maximum ensemble size. This information can be provided by application domain experts or derived from preliminary data analysis. Given that modules are defined up to their parameters, such as phase and frequency in parametric oscillator modules, the challenge for synchronization is in identifying relevant dynamic modules, their quantity, and their unknown parameters.

This chapter is structured as follows. Initially, it mathematically defines the goal of synchronization, explains the necessity of imposing additional constraints, and formalizes

the synchronization algorithm. Subsequently, it describes the modular architecture, including the network topology and the properties of pertinent module types to be utilized in the experiments afterwards. Finally, an explanation of handling the modules in the co-evolutionary framework concludes the chapter.

3.1. Objective of synchronization

The concept of synchronization is formalized as follows. Given a time series $\mathbf{y}(t)$ and an ensemble of dynamic modules $\mathcal{M}$, the method must tune the ensemble such that its modules reproduce the given sequential data with maximum precision. Specifically, the objective of tuning the modules within $\mathcal{M}$ is to minimize the discrepancy $e(t)$ between $\mathbf{y}(t)$ and the aggregate output $\mathbf{o}(t)$ of the ensemble. Mathematically, the discrepancy $e(t)$ defines the objective function as the following Euclidean norm.

$$e(t) = \|\mathbf{y}(t) - \mathbf{o}(t)\|_2 \tag{3.1}$$

Unlike conventional model tuning, synchronization must also refine the ensemble $\mathcal{M}$, which initially possesses an excess size for the given data; specifically, $\mathcal{M}$ contains more modules than are necessary for modeling this data. Thus, concurrently with tuning latent parameters, the method must identify and eliminate superfluous modules. The challenge for synchronization lies in the fact that the time series $\mathbf{y}(t)$ does not reveal its constituent components, their respective contributions, or their temporal alignment. Nonetheless, some a priori knowledge about the characteristics of $\mathbf{y}(t)$ should allow the selection of appropriate module types, ensuring that, to a certain extent, the ensemble is structurally compatible with the time series, as previously discussed in [HM94; HIT02]. Since the relevant modules and their time-varying internal states $\mathbf{x}(t)$ are not known in advance, synchronization must discover them automatically by minimizing the discrepancy $e(t)$ from Eq. (3.1) with respect to the module states $\mathbf{x}(t)$ on the given synchronization sequence $\mathbf{y}(t)$.

$$x^*(t) = \arg\min_x(e(t)) \tag{3.2}$$

This search will determine the set of internal states $\mathbf{x}^*{}_m(t)$ that define the dynamic behavior of modules for replicating the hidden time series components of $\mathbf{y}(t)$. In the case of oscillator modules, their outputs would then oscillate synchronously with the hidden dynamics of $\mathbf{y}(t)$ at identical frequency, amplitude, and phase. Expression (3.2) casts synchronization as an optimization problem where the set of vectors $\mathbf{x}^*{}_1(t)$, ..., $\mathbf{x}^*{}_M(t)$ defines the global optimum $\mathbf{x}^*(t)$ to be discovered.

As previously mentioned, the presented experiments concentrate on linear combinations of modules $\mathbf{o}_m(t)$ within the ensemble, assuming that the given sequential data $\mathbf{y}(t)$ is also composed of linearly combined, yet a priori unknown, primitive dynamics $\mathbf{y}^*(t)$,

i.e.,

$$e(t) = \left\| \sum_{m=1}^{L} \mathbf{y}^*{}_m(t) - \sum_{m=1}^{M} \mathbf{o}_m(x^*{}_m(t)) \right\|_2 \tag{3.3}$$

where the symbol $*$ at $\mathbf{y}^*{}_m(t)$ denotes the unknown hidden dynamics of $\mathbf{y}(t)$ to be identified during synchronization, and L represents their unknown quantity. Given an initial ensemble size M greater than or equal to the presumed quantity L, the individual terms in Eq. (3.3) can be regrouped to indicate the contribution of modules to the total error as

$$e(t) = \left\| \sum_{m=1}^{L} \left(\mathbf{y}^*{}_m(t) - \mathbf{o}_m(x^*{}_m(t)\right) - \sum_{m=1}^{M-L} \mathbf{o}_m(x^*{}_m(t)) \right\|_2 . \tag{3.4}$$

Here, the first term represents the contribution from the discrepancy between the sought-after hidden dynamics and the outputs of corresponding modules. Accordingly, to unambiguously identify the structure of the given time series $\mathbf{y}(t)$, synchronization must find $\mathbf{x}^*{}_1(t)$, ..., $\mathbf{x}^*{}_L(t)$ such that each module reproduces its corresponding hidden component with minimal error. The second term in (3.4) is the cumulative contribution from superfluous modules, which must be minimized to ensure unambiguous identification of the contents in $\mathbf{y}(t)$. The first term in (3.4) casts synchronization as a decomposition problem, where the discrepancy is minimized when synchronization distributes the given $\mathbf{y}(t)$ data among the most suitable relevant modules while suppressing activity in superfluous modules. Consequently, modules exhibiting non-zero activity at the end of synchronization are deemed relevant, and their quantity represents the sought-after number of hidden dynamics in $\mathbf{y}(t)$.

Considering only the total error (3.1) for tuning the ensemble $\mathcal{M}$ renders the synchronization problem underconstrained. Indeed, the absence of information regarding the content of $\mathbf{y}(t)$ precludes the direct computation of individual discrepancies between $\mathbf{y}^*{}_m(t)$ and $\mathbf{o}_m(t)$ for relevant modules in (3.4). The synchronization method lacks direct knowledge of the extent to which $\mathbf{y}(t)$ should be utilized for tuning a specific module. As stated in [JJB91]: *"function decomposition is an underconstrained problem, and, thus, different modular architectures may decompose a function in different ways... A desirable decomposition can be achieved if the architecture is suitably restricted in the types of functions that it can compute."* (p.219). This necessitates the imposition of additional boundary conditions on the feasible domain of unfolding dynamics to prevent the tuning procedure from considering dynamics that are definitively invalid for a particular module. Since the tuning aims to determine the internal state vector $\mathbf{x}^*(t)$, condition (3.5) constrains each module state to the range observed during the design phase.

$$x_j^{min} \leqslant x_j(t) \leqslant x_j^{max} \tag{3.5}$$

In general, the type of applied constraints depends on the properties of available modules. In [CT06], where mixtures of generalized linear models were investigated, constraints were imposed on the ordering of parameters for individual experts: *"Because of the*

mixture structure, we have to impose some order constraints for the experts parameters according to some order relation, so there is no invariance caused by the permutation of expert indices." (p.5). When applied to recurrent neural modules, condition (3.5) defines the intervals within which the states of hidden, output, and input neurons varied during the training phase. Thus, it also restricts the module's output to ignore states that induce unstable dynamics. Such unstable dynamics can arise even for a vector $\mathbf{x}(t)$ where condition (3.5) is satisfied for the individual states of hidden neurons. Although boundary conditions do not directly guide the search algorithm to the global optimum $\mathbf{x}^*(t)$, they regularize the problem [Bis09] (p.144) by preventing combinations of module dynamics that are definitively undesirable.

3.2. Synchronization algorithm

The idea of synchronization is implemented through the online synchronization of modular networks. The algorithm incorporates an optimization method for tuning co-evolving module states but is not limited to a specific approach. This thesis investigates synchronization using evolutionary tuning, Evolutionary Synchronization of Modular Network (ESMoN), and gradient-based tuning, Gradient-based Synchronization of Modular Network (GSMoN).

While synchronization is not inherently tied to a specific optimization algorithm, it maintains multiple sets of candidate solutions, analogous to populations in an evolutionary context. When candidate solutions are updated using a gradient-based optimization method, this procedure is equivalent to multi-start approaches in global optimization ([TT22; Mar03]), which mitigate the risk of premature convergence by initiating searches from diverse locations within the multi-modal search space. Regardless of the optimization method employed, the algorithm evaluates module behavior after tuning on a sliding window at each time step. Based on this evaluation, the tuning result is either accepted or rejected, akin to the handling of offspring in Evolutionary Algorithm (EA)s. Unlike EAs, the performance evaluation of the dynamic module network considers not only modeling accuracy but also the validity of module dynamics. In this context, the use of the term "fitness" would be inappropriate, as it is conventionally defined as a value inversely proportional to modeling error. To avoid this misleading simplification, a more nuanced perspective is adopted throughout this thesis to describe the quality of generated solutions, addressing validity alongside modeling accuracy.

Synchronization is performed online, shifting a sliding window along the given synchronization sequence $\mathbf{y}(t)$. Initially, M populations P_1, ..., P_M of identical size are allocated – one population per available module. Populations are initialized by assigning individuals random values, uniformly distributed across their respective valid ranges. The structure of the i^{th} individual $\mathbf{p_{mi}}$ in population P_m is defined as a vector encoding the internal states $\mathbf{x}(t)$ of the m^{th} module and its responsibility $\mathbf{r_m}$, represented as $\mathbf{p_{mi}} = \{\mathbf{r_m}, \mathbf{x}_m(t)\}$. The valid range for responsibilities $\mathbf{r_m}$ is consistently [0, 1], where "0" indicates complete

irrelevance of the module and "1" signifies that the model requires a 100% contribution from that module. Internal states $\mathbf{x}(t)$ are initialized from their respective valid ranges (3.6) for each module m and each state $x_{mj}(t)$

$$x_{mj}(t) \in [\gamma_{mj}^{min}, \gamma_{mj}^{max}]. \tag{3.6}$$

The limits γ_{mj}^{min} and γ_{mj}^{max} are defined during the design phase as the minimum and maximum activation values observed during module training, specifically for recurrent neural modules, or set to naturally valid ranges for other module types; for example, the phase range is $[0, 2\pi]$ for parametric sine wave modules. Restrictions are also imposed on module outputs – initialized individuals are forced to produce module outputs within the valid range.

$$o_{mj}(t) \in [\omega_{mj}^{min}, \omega_{mj}^{max}] \tag{3.7}$$

Similar to $x_{mj}(t)$ in (3.6), the valid range of $o_{mj}(t)$ is statistically determined during training. This necessitates training sequences that span the entire feasibility region of the dynamics, ensuring the minimum ω_{mj}^{min} and maximum ω_{mj}^{max} values are encountered. Following initialization, the valid ranges (3.6) and (3.7) remain relevant during evolution.

Algorithm 1 outlines the common parts of the synchronization procedure, independent of any specific optimization approach, whether evolutionary or gradient-based. As depicted, the algorithm progresses along the synchronization sequence $\mathbf{y}(t)$. At each time step, the sliding window W_t is updated, and the selected optimization method is subsequently applied to update individuals within the M populations. To mitigate a greedy search, populations are updated concurrently. Specifically, after a single update of each individual in the current population, individuals of the subsequent population are updated, and so forth. This method of consecutive updating the populations resembles coordinate descent optimization ([Wri15]), where each population corresponds to a single coordinate in the search space. The produced new individuals are evaluated using data from the sliding window. Permutation reshuffles them to enhance the match between states of different modules. Following the update of the last population, the next iteration commences with the first population, continuing until each individual has been updated $N(t)$ times. Irrelevant, low-activity modules are discarded post-tuning at the current time step, thereby excluding their populations from co-evolution in subsequent steps. Upon completion of synchronization at the final time step T_{synch}, the algorithm returns the solution – the combination of the fittest individuals from all populations, $\mathbf{g}(T_{synch})^* = \{\mathbf{p}_{best,1}(T_{synch}), ..., \mathbf{p}_{best,M}(T_{synch})\}$, representing the resulting state of the modular network. In this state, active (non-discarded) modules and their discovered parameters identify the content of the given time series $\mathbf{y}(t)$.

Algorithm 1 Synch($\mathcal{M}, \mathbf{y}(t)$)

Initialize $\{P_1, \ldots, P_M\}$ ▷ apply the valid ranges (3.6) and (3.7)
Assign W_t from $\mathbf{y}(1)$ to $\mathbf{y}(T_W)$ ▷ sliding window over time series
for $t = T_W$ *to* T_{synch} **do** ▷ progress along the sequence
    Adapt $\omega^{\mathbf{min}}$ and $\omega^{\mathbf{max}}$ ▷ handle adaptive constraint (4.50)
    for $n = 1$ *to* $N(t)$ **do** ▷ number of population updates (4.53)
        for $m = 1$ *to* M **do** ▷ optimizing individual populations
            Tuning of P_m on W_t ▷ apply Algorithm 2
            Permutation of P_m
        end for
    end for
    Rejection of modules ▷ apply (4.51) or (4.52)
    Update W_t with $\mathbf{y}(t+1)$ ▷ advance the sliding window
    Advance $\{P_1, \ldots, P_M\}$ ▷ update the module states for $t+1$
end for
Find $\mathbf{g}^*(T_{synch}) = \{\mathbf{p}_{best,1}(T_{synch}), ..., \mathbf{p}_{best,M}(T_{synch})\}$ ▷ solution is the fittest individual

3.3. Modular representation

This section details the method for combining co-evolving modules in the ensemble and introduces the various module types investigated in this thesis. Recurrent neural modules and parametric modules with simple, predefined analytic forms can be deployed in the ensemble, depending on the amount of available a priori knowledge. Conducted experiments showed neural modules to be more flexible and usable in diverse applications, while parametric modules demonstrated the algorithm's ability to discover and provide deeper insights into processed sequence data.

3.3.1. Linear ensemble of modules

In this thesis, the investigations conducted focused on the generation and identification of sequential data in ensembles of linearly combined dynamic modules. The model is schematically illustrated in Figure 3.1. The total output of the model is computed as the weighted sum of individual module outputs, as follows.

$$\mathbf{o}(t) = \sum_{m=1}^{M} \mathbf{r}_m \cdot \mathbf{o}_m(t) \tag{3.8}$$

where M denotes the number of modules, $\mathbf{o}_m(t)$ represents the output of the m^{th} module at time t, and $\mathbf{r}_m$ is the responsibility vector of the m^{th} module. Responsibilities $\mathbf{r}_m$ enable control over the contribution of each corresponding module to the ensemble's total output. Thus, setting a responsibility to "0" deactivates the module, while setting it to a value greater than "1" allows for scaling the module's contribution beyond the maximum

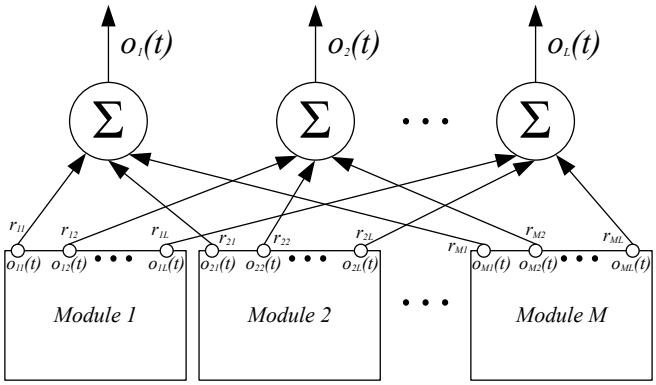

Figure 3.1.: Ensemble of M modules. The module outputs $\mathbf{o}_m(t)$ are linearly combined to form the total output vector $\mathbf{o}(t)$ of the network. The modules contribute to the total output with their respective responsibilities $r_{ml}(t)$ at each output.

achievable magnitude of its output. As responsibility is applied externally to the module, it does not affect possible output feedback, ensuring the module's internal dynamics remain unaltered.

The simplicity of this linear model, the ability to arbitrarily expand the ensemble with additional modules, and its independence from the type of deployed module render it attractive for diverse application areas, including finance and economics, modeling, and time series analysis. In [GGK09], S&P-500 European call option prices were predicted using a weighted combination of FF-ANNs. [Sol02] performed time series decomposition using filters, followed by modeling each component with FF-ANNs. [MC13] introduced the concept of a Modular State Space Echo State Network, where the output is computed as the sum of its module outputs. A weighted sum of experts is employed to model switching processes in regular supermarket customers' purchasing decisions in [CT06]. In cognitive science, summing modules that encode basic concepts facilitates the transition from primitive knowledge to composite concepts. For instance, the composite concept "fruit salad" can be represented as a sum of modules encoding the basic concepts "apple" and "orange." Thus, this method of aggregating separate dynamic components to form more complex dynamics proves applicable even in AI for representation and understanding of abstractions, as highlighted in [Epp+22]: *"Cognitive linguistics and related fields suggest that abstractions benefit from compositionality."* (p.2).

Despite the simplicity of the linear combination, the complexity of the ensemble's output signal can be increased as needed by enhancing the modeling capabilities of the individual modules. A straightforward approach to achieve this is to expand the modules, particularly if they are implemented as monolithic neural networks. This allows for the flexible replacement of any module with a more powerful variant, thereby extending the modeling capabilities of the entire network. However, excessive module expansion for the sake of modeling accuracy is undesirable, as it would compromise the benefits of network modularity. Furthermore, confining high complexity to individual

modules offers a significant advantage in terms of overall model interpretability. This contrasts with even simple non-linear monolithic recurrent networks, where the analysis of internal dynamics is non-trivial and becomes intractable for larger networks.

Previously, the linear combination of independent modules was presented in [KSB15], where it was termed modular ESN (mESN), consisting of compact ESN modules. Its advantages over the standard monolithic ESN were also discussed in that work. Notably, it allows the independent selection of optimal module parameters for each module. This is particularly beneficial for modeling time series composed of dynamic components with significantly different characteristics. Such time series were investigated in the same study, where sinusoidal oscillations were combined with oscillatory triangular signals. Modeling sine waves required only neurons operating in the linear region of the sigmoid function. Conversely, neurons operating in the strongly non-linear region were necessary for triangular signals, but would have compromised the modeling accuracy of the modules for sine oscillations. The structural independence of the deployed modules provides a natural and simple way for decoupling module dynamics, eliminating the need for a dedicated decoupling mechanism as proposed in [XYH07]. With this decoupling, the modules occupy distinct niches within the state space, allowing their dynamics to evolve without mutual interference, thus ensuring optimal performance of the entire ensemble.

The independence of modules offers an additional advantage in tuning module dynamics within the ensemble by mitigating the adverse effects of irrelevant module rejection on the convergence of the tuning algorithm. Rejection of irrelevant modules is treated as a disruption within the ensemble, leading to an abrupt loss of their contribution. Typically, this disruption impacts the ensemble's total output, resulting in an error increase. However, due to module independence, this disruption does not induce any signal loss for the remaining modules; consequently, there is no interruption of the evolving dynamics within them. Given that the module dynamics are not directly affected by the potential removal of other modules from the ensemble, module independence facilitates gradual convergence of module dynamics with a reduced risk of failure.

Dynamic stability is another characteristic that benefits from the independence of dynamic modules. Assuming the stability of individual modules, the absence of inter-module connections precludes the formation of looping paths for positive feedback signal flow and signal paths where modules could interfere with each other unpredictably. This ensures high modeling accuracy after tuning the ensemble for long-term prediction.

The linear combination of modules is not restricted to the type of deployed modules; the module type can be freely chosen, provided that the output vectors $\mathbf{o}_m(t)$ maintain equal dimensionality. In the simplest scenario, each module within the ensemble is a monolithic neural network. More generally, a module can encapsulate a more intricate structure, such as a hierarchy of other modules. This may implement nested modular structures, where modules themselves are composed of modular subcomponents. This concept aligns with biological plausibility, as complex organisms exhibit a range of

modularity levels reflecting the hierarchical granularity of their structure: a biological body comprises organs, each organ consists of diverse cell types, and each cell is further decomposable into its constituent parts, and so forth. In this thesis, nested modularity was implemented through compositional recurrent modules to replicate the aperiodic dynamics of handwritten symbols, where each module was a hybrid of a primary monolithic RNN and a secondary parameterizer sub-module.

3.3.2. Recurrent neural modules

For the presented experiments, recurrent neural modules were implemented with ESNs. The principal characteristics of this RNN type are detailed in Appendix A. Initially, ESNs were successfully employed for modeling complex time series from chaotic attractors, requiring substantial hidden layers to achieve high prediction accuracy. For instance, in [GM09], the MG time series were modeled with reservoir sizes of approximately 1000 neurons, and the Lorenz attractor with reservoir sizes of around 500 in [JH04]. Subsequent studies [RI09; KLB12; KSB15] demonstrated that complex oscillations can be modeled with ESNs comprising only a few reservoir neurons. This is highly advantageous for online synchronization, as it results in compact modules and ensures low computational overhead for their updates. Furthermore, ESNs have exhibited exceptional stability, reproducing periodic dynamics for extended periods with minimal error – orders of magnitude smaller than unity – in auto-generation mode without external input (refer to the results in [KLB12; Ott+15]). The modeling capabilities of ESNs are further enhanced by their integration into ensembles. Their superiority over monolithic ESNs was shown in [KSB15], where increased time series complexity led to only a modest growth of the ensemble.

The independence of modules within the ensemble allows independent training of ESN modules, as well as the independent selection of ESN parameters. These parameters include the size of the dynamic reservoir, its spectral radius, and the initialization interval for the feedback weights $\mathbf{W}_{OFB}$. As detailed in Appendix A, the characteristics of the modeled time series are crucial for determining these parameter values. This indicates that ESN ensembles are well-suited for modeling time series composed of distinct dynamic components with significantly differing characteristics, such as when there is a substantial disparity in the time scales of the constituent dynamics. The capability of both monolithic and modular ESNs to model such time series was empirically examined and documented in [KSB15].

3.3.3. Parametric modules

Parametric modules are those whose behavior is defined by parameterized mathematical expressions, derived from expert knowledge about the nature of the modeled process within the application domain. Their deployment is appropriate for processes where the dynamic components of sequential data can be represented using a parameterized

form. If this requirement is met, and a suitable parameterized expression is available, the task of the synchronization algorithm is to determine the unknown parameter values. Given their typically shorter parameter lists compared to alternative neural modules, parametric modules require fewer computational resources. Furthermore, the underlying parametric form can often be mathematically analyzed to provide deeper insights into the interdependencies between parameters. This makes parametric modules more suitable for understanding the modeled dynamics, a benefit further enhanced by the known semantics of the parameters, such as amplitude and phase in the case of sinusoidal modules.

In this thesis, the capabilities of the synchronization method were investigated using compound oscillatory time-series data, employing the following three types of parametric modules.

- *Sinusoidal module*
- *Triangular signal module*
- *Rectangular signal module*

The output of the sinusoidal module is defined by the sine wave formula

$$o_m(t) = A_m \cdot \sin \lambda_m \cdot t + \phi_m \tag{3.9}$$

where λ_m and ϕ_m represent the frequency and phase of the sine wave, respectively, and t denotes the current time step. In the experiments, the amplitude A_m of the module's output signal was fixed at 1 and appropriately scaled by the module responsibility r_m outside the module. The differentiability of sinusoidal modules is a key advantage, facilitating the derivation of update rules for gradient-based parameter tuning during synchronization. Decomposition using sinusoidal parametric modules aligns with the theoretically grounded Fourier Transform (Fourier Transform (FT)), which relies on sine signals to identify the frequency content of a given sequence data.

In contrast to the FT, the synchronization method is not constrained to a specific type of oscillatory signal and permits the use of triangular and rectangular signal modules. Despite the simplicity of their temporal patterns, these signals are considerably more challenging to model and tune than sine waves. The mathematical expressions for both signals are not differentiable across the entire domain of the argument; specifically, the derivative is undefined at certain values of t. Consequently, gradient-based tuning methods are inapplicable for these types of parametric modules. The outputs and parameters of the triangular and rectangular signal modules are illustrated in Figure 3.2. The output of the triangular signal parametric module m is computed using the following formulas.

$$\begin{cases} \frac{4A_m}{T_m} \cdot t_T & if\ t_T < \frac{T_m}{4} \\ -\frac{4A_m}{T_m} \cdot t_T + 2A_m & if\ \frac{T_m}{4} \leqslant t_T < \frac{3}{4}T_m \\ \frac{4A_m}{T_m} \cdot t_T - 2A_m & if\ t_T \geqslant \frac{3}{4}T_m \end{cases} \tag{3.10}$$

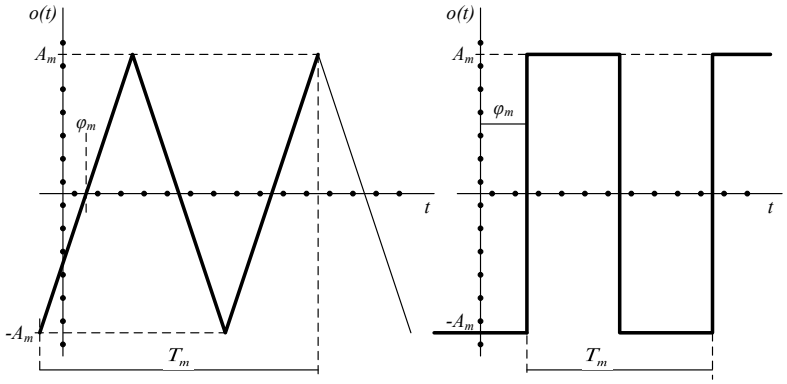

Figure 3.2.: The outputs of the triangular signal parametric module (left) and the rectangular signal parametric module (right) with the amplitude A_m, phase ϕ_m, and period T_m.

where $t_T = (t + \phi_m) \mod T_m$ represents the time index within a single period T_m of the triangular signal, which has a phase ϕ_m and amplitude A_m. The output of the rectangular signal parametric module m is determined using the following formulas.

$$\begin{cases} A_m & if\ t_T < \frac{T_m}{2} \\ -A_m & if\ t_T \geqslant \frac{T_m}{2} \end{cases} \tag{3.11}$$

where $t_T = (t + \phi_m) \mod T_m$ represents the time index within a single period T_m of the rectangular signal, which has a phase ϕ_m and amplitude A_m. Compared to triangular signal modules, rectangular signal modules pose a significantly greater challenge because of their intervals of constant magnitude. Synchronization must accurately capture both the period T_m and the phase ϕ_m to prevent an abrupt increase in the error signal. Assuming an equal amplitude A for the target signal and the module output, the error signal abruptly increases by $2A$ at each time step where the actual and determined periods diverge.

The synchronization method shares its flexibility in the utilization of diverse module types with Hilbert-Huang Transform (HHT) ([HSL99; YG06]). The HHT does not prescribe a specific type of basis signal, though it requires symmetry with respect to the time axis. In both approaches, the appropriate parametric form can be selected based on a priori knowledge from the application domain.

3.4. Co-evolving modules

Topological separation among the modules in the ensemble suggests the application of a co-evolutionary approach for tuning the available modules. This approach presumes that the problem under consideration is decomposable into simpler sub-problems.

The objective of cooperative co-evolution, as studied in [Sch+07; PD00; JW03; MM97; YM07; GSM08], is to identify optimal solutions for these sub-problems, such that their composition yields a solution for the original problem. Competitive co-evolution is an alternative approach to co-evolving solutions. It does not require the original problem to be decomposable but necessitates a reference problem to assess the quality of trial solutions, which should outperform the solution of the reference problem [FP00; FP01].

For the synchronization of module dynamics, the combination of module outputs to form the total output was the decisive factor in favoring cooperative co-evolution, where each population is assigned to a corresponding module. This facilitates the implementation of a divide-and-conquer strategy, wherein the original problem is decomposed into simpler sub-problems with solutions in search sub-spaces of lower dimensionality. Each of these sub-spaces is more efficiently explored by its own small population. The dimensionality of individuals within these populations is less than the compound dimensionality of the overall problem. This enhances the search's ability to escape local optima, an effect previously observed in [GSM08]: *"As the number of species (sub-genotypes) increases, the selection of subgenotypes for reproduction becomes less greedy, causing the search points that are evaluated each generation to converge more slowly, providing more paths toward better solutions."* (p.942) Furthermore, the independence of modules within the ensemble appears to be advantageous for co-evolution as well – according to [Fic04]: *"The less the sub-problems interact with each other, the more effective cooperative coevolution will be."* (p.10)

As in any EA, individuals are evaluated and assigned fitness values. However, the subjectivity of fitness distinguishes co-evolution from evolution with a single population. This subjectivity implies that the fitness of an individual is evaluated in the context of individuals from other populations. Consequently, the computed fitness reflects the efficacy of an evaluated combination of individuals in solving the original problem. Since synchronization aims to produce ensembles with maximally accurate reproductions of the target sequence, individuals that minimize errors within the ensembles receive high fitness:

$$F(i_m) = \frac{1}{e(i_m, i_{C(m)})} \tag{3.12}$$

where $C(m) = [m' |\ m' \in \mathcal{M} \cap m \neq m']$ denotes the contextual modules, and $i_{C(m)}$ specifies the respective individuals from the contextual modules' populations, which are combined with i_m for its fitness evaluation. Throughout the experiments in this thesis, the discrepancy between the ensemble output and the true time-series value was computed as MSE, though other measures are also usable. As evident, this error's dependence on all module states within the ensemble requires cooperative evolution among their populations, similar to the approach in [Sch+07; PD00; JW03; MM97] for maximizing $F(i_m)$.

In practice, evaluating all possible combinations of individuals is infeasible; therefore, a judicious selection of contextual individuals $i_{C(m)}$ is crucial, as an "unlucky" context may result in the loss of a potentially valuable individual. The objective of context

selection approaches for $i_{C(m)}$ is to mitigate this risk and ensure steady convergence of the algorithm towards the global optimum. An elitist strategy is employed in [PD00], where an individual's fitness is computed by combining it with the best individuals from other populations. A less stringent elitism is utilized for neuro-evolution in [MM97], where each neuron is assigned the average fitness from the five best ANNs in which it participated. The online mode of the presented synchronization method precludes the analysis of multiple individual combinations for a single fitness evaluation. Here, contextual individuals are selected to have the same rank as the evaluated individual, based on their current fitness estimates. A similar approach was presented for Neuroevolution (NE) in [GSM08], where context individuals were chosen probabilistically according to their fitness.

4. Synchronization of co-evolving oscillator modules

The capabilities and behavior of the synchronization approach are initially investigated using continuous oscillatory time series. These time series are composed of multiple primitive oscillatory dynamics, which can be modeled using either compact recurrent neural modules or appropriate parametric modules. Synchronization identifies relevant modules through two concurrent processes – permutation of co-evolving modules and tuning their internal dynamics – which harmonize the modules to become precise sequence data generators. In this context, imposed constraints play a crucial role in discarding implausible candidate solutions and guiding synchronization towards the desired decomposition of the given sequence data.

Because of the lack of generally decomposable underlying dynamics in chaotic attractor time series, they were not intensively used for analyzing the synchronization method's behavior. Nevertheless, to evaluate the approach's performance, several runs were conducted on the MG chaotic attractor time series, revealing surprising results regarding dynamics that typically lack a decomposable structure.

4.1. Periodic dynamics

The continuous oscillatory dynamics considered were composed of sine waves, as well as triangular and rectangular oscillatory signals. Although the synchronization algorithm did not have access to the true composition of the given time series, this provided a valuable opportunity for automated analysis of the experimental data. This facilitated the drawing of reliable conclusions after processing a large number of sequences, eliminating the need for manual, time-consuming verification of the obtained decomposition's correctness after each sequence.

4.1.1. Sine waves

Combinations of sine waves are the so-called MSO. These time series have been previously considered as a benchmark problem in various studies, including [HH10; Ott+16; Sch+07; XYH07]. These studies considered MSO time series with varying numbers of oscillations. As shown in Eq. (4.13), MSO time series are defined as a linear combination of several

simple sine waves. Despite their simple analytical form, their extremely long period renders prediction challenging, particularly with a larger number of sine waves.

$$y(t) = \sum_{k=1}^{K} a_k \cdot \sin(\lambda_k t + \phi_k) \tag{4.1}$$

where K is the number of superimposed sine waves; a_k, λ_k, and ϕ_k denote their amplitudes, frequencies, and phases, respectively; and t is an integer index of a time step. The frequencies $\lambda_1 = 0.2$, $\lambda_2 = 0.311$, $\lambda_3 = 0.42$, $\lambda_4 = 0.51$, $\lambda_5 = 0.63$, $\lambda_6 = 0.74$, $\lambda_7 = 0.85$, and $\lambda_8 = 0.97$ were utilized in previous studies on MSO time series and were also employed here. Their floating-point values extend the period of the resulting signal $y(t)$ and minimize the likelihood of its repetitions within the sequence data.

In the standard configuration, the amplitudes a_k of the sinusoidal signals are set to unity, resulting in so-called balanced time series. Alternatively, imbalanced time series are generated by individually weighting the components with unequal, randomly assigned amplitudes, uniformly distributed over the interval $[0, 1]$. These amplitudes are set to distinct random values, which may vary by orders of magnitude. The significant imbalance between minor oscillations with small amplitudes and major oscillations with large amplitudes substantially increases the problem's complexity. Only a few studies present experiments with varying amplitudes. In [Aza00], the data indicated the presence of two components with amplitudes of "0.75" and "2.25". In [Ott+16], different combinations of several sine waves were considered, analyzing optimized ESN reservoirs on MSO sequences with components having random amplitudes within the interval $[0.1, 1.0]$.

Analogous to the amplitudes of the oscillations, their phases ϕ_k were varied across different MSO sequences. Within each sequence, the phases ϕ_k of the oscillations were randomly assigned values, uniformly distributed over the interval $[0, 2\pi]$. To demonstrate the algorithm's scalability, experiments were conducted on time series of varying complexity, modulated by the number of oscillations within the series. The MSO dynamics ranged from the simplest MSO1, comprising a single sine wave, to the most complex MSO8, consisting of eight sine waves. Exemplary MSO sequences are illustrated in Figure 4.1. Comparable sequence data were utilized in [Ott+16] for the optimization of monolithic ESNs. However, the majority of other studies consider MSO time series with zero phases, $\phi_k = 0$.

4.1.2. Triangular and rectangular oscillations

In addition to the MSO dynamics, the synchronization abilities were investigated using more complex triangular and rectangular oscillatory signals. A literature review on recurrent neural networks revealed no prior studies addressing triangular oscillatory dynamics. The requirement for neurons operating in the saturated range of the TANH activation function renders triangular oscillations particularly challenging for modeling

by recurrent neural networks, where significant variations in the neuron's input signal elicit only a minimal change in the error signal. Conversely, rectangular oscillatory dynamics present a distinct challenge; minor inaccuracies in modeling their period and phase result in a steep increase in the error signal. Reliable identification of both types of oscillatory signals demonstrates the synchronization's ability to address these challenges and decompose the given time series independently of the oscillation type.

For the conducted experiments, sequences of triangular and rectangular oscillations were generated using the mathematical formulae (3.10) and (3.11), respectively. Similar to sine waves, these signals are characterized by the oscillation period T and the phase ϕ. To enhance the complexity of the time series, up to six signals with varying periods T were linearly combined as

$$y(t) = \sum_{k=1}^{K} a_k \cdot Triangle(T_k, \phi_k) \tag{4.2}$$

for the triangular signals, and as

$$y(t) = \sum_{k=1}^{K} a_k \cdot Rectangle(T_k, \phi_k) \tag{4.3}$$

for the rectangular signals. In both equations (4.2) and (4.3), K represents the number of signals presented, while a_k, T_k, and ϕ_k denote the amplitude, period, and phase of the corresponding signal, respectively. The individual signals were assigned periods of $T_1 = 32$, $T_2 = 20$, $T_3 = 16$, $T_4 = 12$, $T_5 = 8$, and $T_6 = 4$. Figure 4.1 illustrates exemplary sequences of the simplest and most complex dynamics, providing insight into the high complexity of sequences with $K = 6$. As observed, the combination of six triangular signals ($K = 6$) resembles a chaotic attractor but, unlike the latter, possesses an internal structure composed of separable dynamic components. The phase ϕ_k was randomly selected from a uniform distribution over the interval $[0, T_k]$. In balanced dynamics, all combined components had equal amplitudes, a_k, set to 1. Conversely, in imbalanced dynamics, the amplitudes a_k were randomly selected from a uniform distribution over the interval $[0, 1]$, akin to Multiple Sine Oscillation (MSO) time series. The synchronization method is expected to be amplitude-invariant to identify minor and major dynamic components independently of their amplitudes. It was further challenged by MST that combined sine waves and triangular signals as

$$y(t) = \sum_{k=1}^{K} a_k \cdot \sin(\lambda_k t + \phi_k) + \sum_{k=1}^{K} a_k \cdot Triangle(T_k, \phi_k) \tag{4.4}$$

where K equals 1, 2, 3, or 4, corresponding to increasing complexity of the time series. As described in [KSB15], modeling such time series necessitates recurrent modules with independently selected critical parameters to maintain neuron operation purely in the

Figure 4.1.: Exemplary sequences of the compositional time series are ordered according to hardness of their modeling from the top to the bottom: MSO2 and MSO8 (top), Triangle2 and Triangle6 (middle), Rectangle2 and Rectangle6 (bottom). The Triangle time series are not steadily differentiable signals. The Rectangle time series have intervals of the constant signal magnitude.

saturation range for triangular signals and on the linear interval of the TANH activation function for modeling sine waves.

4.2. Tuning of the internal dynamics

Module tuning is accomplished by minimizing the discrepancy between the given time series $\mathbf{y}(t)$ and the ensemble's total output $\mathbf{o}(t)$. To achieve this, the internal states and responsibilities of the modules are adjusted using either an evolutionary or a gradient-based approach, as shown in Algorithm 2. The algorithm demonstrates that either the module's internal states or its responsibilities are tuned at any given time. This separation stems from the significantly stronger impact of responsibility changes on the error signal. Responsibility defines the scaling of the module output signal and,

Algorithm 2 Tuning of P_m on W_t

for p_i *in* P_m **do** ▷ over individuals of the m^{th} module
  Choose between $\mathbf{r}_i$ and $\mathbf{x}_i$ for update ▷ randomly with probability μ_r
  if $\mathbf{r}_i$ is chosen **then**
    Produce $\mathbf{p}_i^{new}$ by updating $\mathbf{r}_i$ ▷ evolutionary or gradient-based
  else
    Produce $\mathbf{p}_i^{new}$ by updating $\mathbf{x}_i$ ▷ evolutionary or gradient-based
  end if
  Verify validity of $\mathbf{p}_i^{new}$ ▷ handle static constraints (4.47) and (4.48)
  Compute accuracy of $\mathbf{p}_i^{new}$ ▷ apply Eq. (4.5)
  Evaluate $\mathbf{p}_i^{new}$ ▷ accept or reject $\mathbf{p}_i^{new}$ based on validity and accuracy
end for

consequently, possess the potential for rapid error reduction, which can lead to premature convergence during synchronization. This is mitigated by less frequent updates of the module responsibilities compared to the module states. As shown in Algorithm 2, this frequency is controlled by the probability μ_r, set to 20% based on preliminary experiments, meaning the module states are updated, on average, five times more frequently than the module responsibilities. Following each update, constraints are enforced, and the updated value is either accepted or rejected based on the resulting ensemble accuracy.

4.2.1. Modeling accuracy

Synchronization decomposes the time series into a combination of modules, with their dynamics and parameters encoded by the most successful individuals from different populations, akin to the approach in [PD00]. The combination of these individuals, similar to the pairing of individuals within each ensemble throughout the process, is based on their performance ranking. Specifically, individuals in each population are ordered according to their modeling accuracy, with the highest-performing individuals positioned at the beginning of the population. This method facilitates the linking of fitter individuals from different populations according to their modeling accuracy rank: individuals with identical indices across populations are linked, as depicted in Figure 4.2. This pairing strategy aligns the highest-performing individuals, the second-highest, and so forth. Maintaining equal population sizes ensures that each individual in every population is uniquely linked to one individual in each other population at any given time.

Paired individuals with identical indices cooperate within an ensemble. Their successful cooperation is validated by the high modeling accuracy of the resulting ensemble. The computation of modeling accuracy is schematically illustrated in Figure 4.2. Here, accuracy is evaluated prospectively as the modules evolve over a sliding window W_t, initiating at the current time step t_c. Module dynamics commence at t_c with states

defined by the evaluated individuals and progress until the end of the sliding window, while module responsibilities remain constant. The average error $e_{T_W}(t_c)$ represents the ensemble's modeling accuracy and is computed as MSE over the sliding window as

$$e_{T_W}(t_c) = \frac{1}{T_W} \sum_{t=t_c}^{t_c+T_W-1} (\mathbf{y}(t) - \mathbf{o}(t))^2 \tag{4.5}$$

where T_W is the length of the sliding window, and $\mathbf{y}(t)$ and $\mathbf{o}(t)$ represent the given time series and the ensemble's total output, respectively. The average error $e_{T_W}(t_c)$ serves as an estimate of the objective function $e(t)$ (see Eq. (3.1)). The latter cannot be directly computed because the sequence $\mathbf{y}(t)$ is not provided all at once, and the internal states $x^*_m(t)$ of the modules, which replicate the hidden components of $\mathbf{y}(t)$, are unknown a priori. This accommodates the potential for dynamic changes in the given data, rendering the performance measure (4.5) suitable for non-stationary environments, such as those in [Pas+01] and [Jin07]. The inherent uncertainty of the internal states $x^*_m(t)$ implies that the computed output $\mathbf{o}(t)$ remains suboptimal during synchronization. It is anticipated that steady convergence to the global optimum will yield individuals who progressively refine the estimates $e_{T_W}(t_c)$ of the hidden objective function $e(t)$. Empirically, experiments confirmed that the best individuals maintained consistently low values of $e_{T_W}(t_c)$ even after the sliding window's shift during synchronization.

Cooperation among individuals yields an accuracy level that is not solely dependent on any single individual, but rather on the entire ensemble, where each individual defines the dynamic state of its corresponding module. Given the uncertainty regarding each individual's contribution to the ensemble's success, all individuals share the same $e_{T_W}(t_c)$ value. While the co-evolving modules in Figure 3.1 seem independent, their cooperative interaction introduces an indirect dependency, substantially influencing module tuning, as determined by $e_{T_W}(t_c)$. To grasp the intuition behind this effect, it is essential to recall that synchronization tunes one module at a time, and its dynamic states are updated and evaluated within the non-volatile context of its cooperators. The dependency of $e_{T_W}(t_c)$ on the current context, as derived from incorporating (3.8) into (4.5), is shown by the following expression.

$$e_{T_W}(t_c) = \frac{1}{T_W} \sum_{t \in W_t} (\mathbf{y}(t) - \mathbf{r}_{p_i} \cdot \mathbf{o}_{m,p_i}(t) - \mathbf{C}_{m,p_i}(t))^2 \tag{4.6}$$

where T_W is the size of the sliding window, $\mathbf{C}_{m,p_i}(t)$ denotes the context of individual p_i from population P_m, $\mathbf{o}_{m,p_i}(t)$ is the output of the m^{th} module whose state $\mathbf{x}_{m,p_i}(t)$ and responsibility $\mathbf{r}_{m,p_i}(t)$ are defined by individual p_i. As the linkages between individuals remain undisturbed during the tuning of a single module, the context $\mathbf{C}_{m,p_i}(t)$ remains constant despite any transformations of p_i.

The constancy of $\mathbf{C}_{m,p_i}(t)$ is an indispensable condition for tuning the entire ensemble through the consecutive tuning of its modules. This prevents the algorithm from

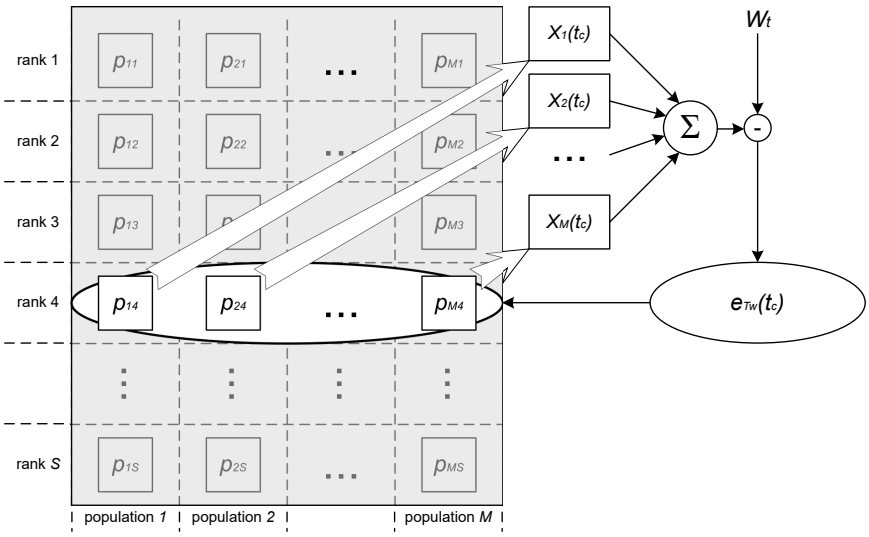

Figure 4.2.: Rank-based computation of the modeling accuracy, $e_{T_W}(t_c)$, for the ensemble of individuals sharing the same rank (here, rank 4) across different populations, as previously illustrated in [KOB21]. Prior to evaluation on the current sliding window, W_t, the states and responsibilities are transferred from each individual to its respective module. The resulting prediction error, $e_{T_W}(t_c)$, is shared among the individuals.

becoming trapped in a local optimum, which is typically observed as a rapid and abrupt improvement of $e_{T_W}(t_c)$ during the state updates of the current module. Abrupt improvements of $e_{T_W}(t_c)$ within a single time step are undesirable because the sliding window W_t provides only a limited perspective of the entire time series. A sudden improvement of $e_{T_W}(t_c)$ would suggest that the module has been tuned excessively to match its context, indicating that the algorithm has converged to a local optimum. Conversely, requiring modules of compact size restricts their plasticity and, consequently, a single module's ability to fully replicate the output dynamics of its context $\mathbf{C}_{m,p_i}(t)$ for the global reduction of the error $e_{T_W}(t_c)$. This ensures that the dynamics of individual modules remain suboptimal at each step until final synchronization is achieved. In turn, the suboptimality of $\mathbf{C}_{m,p_i}(t)$ limits the magnitude of potential accuracy improvements attainable through updates to the current module. Therefore, the constancy of $\mathbf{C}_{m,p_i}(t)$ reinforces this implicit limitation for the current time step. Subsequently, changes in $\mathbf{C}_{m,p_i}(t)$ between consecutive time steps establish a new implicit limit for potential improvements, depending on the extent of generalization gained by the module before the sliding window shift.

4.2.2. Evolutionary tuning

This thesis investigated Artificial Evolution (AE) as an alternative approach for updating module states. The ability of AE to find solutions with limited prior knowledge of the underlying objective function's properties makes it particularly attractive for online tuning of module dynamics. This ensures flexibility in the synchronization method regarding the selection of initially available modules and their combinations. Unlike Newton's optimization methods ([KC02; Fle70]), AE does not require assumptions about the objective function's properties, such as convexity. Consequently, AE enables the tuning of module combinations with a wider variety of reproduced dynamics properties, which may not be tractable by a single theoretically well-founded optimization method. Furthermore, evolutionary optimization is applicable to all module types, including parametric modules with non-differentiable characteristics, such as those reproducing triangular or rectangular signals. Beyond the benefits of broad applicability, the modularity of the network enhances the effectiveness of evolutionary tuning by reducing the dimensionality of the optimization problem, as stated in [RSM04]: *"With the added level of abstraction that modules provide, more complex solutions are searched with fewer operations than standard NE."* Additionally, modularity reduces the requirements to complexity of mutational operations: *"...modularity allows for reuse of complex phenotypic structures without correspondingly complex mutations in the genotype."*

Update rule for the module states

The requirement to operate directly on real-valued vectors restricted the set of evolutionary optimization methods suitable for implementing the update procedure. This avoids encoding vectors into bit strings and decoding them, as performed in genetic algorithms, thereby enhancing the efficiency of evolutionary tuning. Among the candidate methods, Differential Evolution (DE) was deemed particularly suitable due to its simplicity and low computational resource demands. Its excellent performance in optimizing dynamic reservoirs, even with very small populations of only five individuals, was reported in [Ott+16]. In this thesis, this feature allowed for reduced computational effort while achieving high synchronization accuracy with a small population of only 100 individuals, within the range of $5D$ to $10D$ suggested by [SP95], where D is the dimensionality of a single module's search space. Beyond its compact population size, DE supports contour matching, which facilitates harmonization of module dynamics ([DS11]: *"Contour matching refers to the phenomena of adaptation of the vector population such that promising regions of the fitness landscape are investigated automatically once they are detected."*). Furthermore, the high relevance of DE for dynamic module synchronization is supported by prior studies demonstrating its successful application to the problems of constrained optimization [PSL05; LZ99; TS06; HWH07] and non-stationary objective functions [MM05; Bre+09; AS07].

DE is an evolutionary optimization algorithm that emulates the principles of biological evolution, where recombination and mutation are applied to the current parental

population to generate the next generation. In this process, fitter individuals have a higher probability of survival and transmitting their genetic material to the subsequent generation, enabling the algorithm to iteratively refine candidate solutions. In DE, an offspring is generated for each individual and replaces the parent if it exhibits superior performance. Conversely, if the offspring performs worse, the parent remains unchanged. Offspring generation involves two primary steps: first, mutation is performed; then, the mutated result is recombined with the parent to produce the offspring.

In DE, the mutation scheme distinguishes itself from those in other evolutionary methods. Unlike traditional approaches, it utilizes multiple individuals to generate the mutant vector for a given individual. This involvement of multiple individuals in updating a single individual fosters enhanced communication across the population and mitigates the risk of losing the global optimum in the covered part of the search space. Numerous studies on DE have introduced various mutation scheme variants, differing in complexity and the number of involved individuals. The choice of the most appropriate variant depends on the particular objective function and is typically determined through preliminary screening, i.e., after several initial runs. A concise overview of potential objective function types and suitable DE mutation rules, along with additional background on DE, can be found in [DS11]. The seminal work by [SP95] proposed the following basic mutation scheme, which explains the fundamental principle of DE mutation.

$$\mathbf{v}_i = \mathbf{p}_a + F \times (\mathbf{p}_b - \mathbf{p}_c) \tag{4.7}$$

where $\mathbf{v}_i$ is the obtained *donor vector*, and $\mathbf{p}_a$, $\mathbf{p}_b$, $\mathbf{p}_c$ are randomly selected individuals from the population with distinct indices, all different from the index i of the current *target vector* $\mathbf{p}_i$. F is the scaling factor, also known as the differential weight, applied to the difference between $\mathbf{p}_b$ and $\mathbf{p}_c$. The selection of this DE parameter depends on the specific objective function and typically falls within the interval $(0, 1]$. Notably, there is no universally accepted consensus regarding the correlation between the optimal F value and the characteristics of objective functions. As noted in [DS11]: *"As can be perceived from the literature, several claims and counter-claims were reported concerning the rules for choosing the control parameters"*. Qualitatively, larger F values bias mutation towards exploration by facilitating larger steps around the base vector $\mathbf{p}_a$, while smaller F values favor exploitation, which is particularly effective for unimodal objective functions. Given that the properties of the objective function are often unknown, trial runs are conducted to empirically determine suitable parameter settings. Because of the slow convergence of the basic mutation scheme [QHS09], it is less suitable for online synchronization. Instead, the following mutation rule, commonly referred to in the literature (e.g., [Das+09; DS11; Ott+16]) as the *"DE/target-to-best/1"* mutation rule, is employed.

$$\mathbf{v}_i = \mathbf{p}_i + F \times (\mathbf{p}_{best} - \mathbf{p}_i) + F \times (\mathbf{p}_a - \mathbf{p}_b) \tag{4.8}$$

where $\mathbf{p}_i$ is the target vector, $\mathbf{p}_{best}$ is the best individual in the population, and $\mathbf{p}_a$ and $\mathbf{p}_b$ are randomly selected individuals from the population, with indices a and b distinct from i and *best*. According to [Ott+16], *"For many problems the DE/target-to-best/1 scheme performed*

best because it offers a good trade-off between convergence behavior and the number of fitness evaluations." This scheme also promotes steady convergence and mitigates stagnation in diverse populations that exhibit unexplained lack of progress, a phenomenon reported by [LZ00] for the basic mutation rule (4.7).

The obtained donor vector $\mathbf{v}_i$ contributes the genetic material to modify the target vector $\mathbf{p}_i$, thereby generating the trial vector $\mathbf{o}_i$, which serves as a potential update for $\mathbf{p}_i$. Recombination combines $\mathbf{v}_i$ and $\mathbf{p}_i$ to produce the trial vector, where each element is assigned from $\mathbf{v}_i$ with a predefined crossover probability CR, as follows as

$$o_{ij} = \begin{cases} v_{ij} & if \quad c \leqslant CR \quad or \quad j = j_{rand} \\ p_{ij} & if \quad otherwise \end{cases} \tag{4.9}$$

where o_{ij} is the j^{th} element of the trial vector. For small values of CR, the trial vector predominantly inherits elements from the target vector $\mathbf{p}_i$. The random value c, uniformly distributed over the interval $[0, 1]$, is generated for each o_{ij}. Consequently, the number of elements assigned from $\mathbf{v}_i$ follows a binomial distribution, leading to the designation of operation (4.9) as the binomial crossover. The random index j_{rand} is generated once per vector to guarantee that the initial $\mathbf{p}_i$ and updated $\mathbf{o}_i$ vectors differ by at least one element, ensuring $\Delta\mathbf{p}_i \neq 0$. The magnitude and direction of the update vector $\Delta\mathbf{p}_i$ are determined by the crossover probability and, to some extent, by the chosen mutation rule. Utilizing the target vector $\mathbf{p}_i$ in mutation, as in (4.8), results in a smaller magnitude of $\Delta\mathbf{p}_i$ compared to (4.7), which does not involve $\mathbf{p}_i$. Thus, module states are adapted more conservatively when tuning employs rule (4.8).

Figure 4.3 illustrates the typical performance dependency on the DE parameters F and CR for parametric and recurrent neural modules applied to mixtures of sinusoidal and triangular oscillatory components. As depicted in the figure, a range of parameter combinations yielded the highest performance for each deployed module type and on each time series. These combinations are visualized as areas of minimal test RMSE in these 3D plots, having a different spread. This spread is bigger for less complex mixtures of sinusoidal oscillations, observed for both parametric and neural modules. Conversely, for more challenging mixtures of triangular signal oscillations, a wide spread of suitable parameter values is evident only for triangular parametric modules, which exhibit a piecewise-linear dependence of behavior on module parameters (see Eq. (3.10)). In contrast, the hidden neurons of neural modules must operate in saturation to model triangular oscillatory signals, resulting in a rougher fitness landscape and, consequently, a more complex dependency of performance on the DE parameters. Notably, the highest performance for neural modules was achieved with F parameters from two distinct intervals (see the two distinct parameter areas in Figure 4.3). This suggests a dynamic fitness landscape that could evolve during synchronization according to two potential scenarios: either becoming more complex, requiring more intensive exploration with a larger F value of 0.8, or becoming less complex, favoring moderate exploration with a smaller F value of 0.3. Thereat, evolutionary tuning demonstrated less sensitivity to

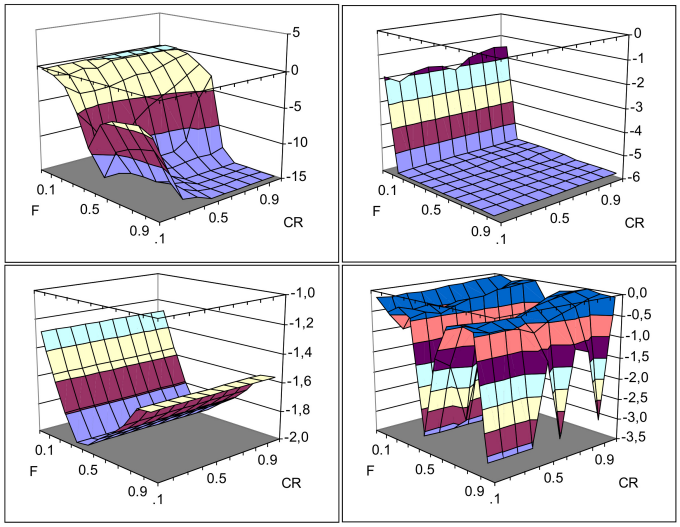

Figure 4.3.: Dependency of the reached test RMSE on the parameters of differential evolution – differential weight (F) and crossover rate (CR) on the MSO4 (top row) and Triangle4 (bottom row) time series with the parametric modules (left column) and the ESN modules (right column).

variations in the crossover rate *CR*, which had suitable values across a broad range.

Update rule for responsibilities

Evolutionary tuning accounts for the semantics of module attributes by employing distinct rules for updating module states and module responsibilities. This separation of updates was necessitated by observed performance degradation in preliminary runs when multiple individuals were simultaneously involved in responsibility updates. This indicated that the DE rule (4.8) was unsuitable for responsibilities and required an alternative rule where the update depends exclusively on the responsibility of the current individual $\mathbf{p}_i$. To uphold the objective of synchronization and to exclude potentially superfluous modules from the ensemble, this thesis proposes updating responsibilities through the suppression of module activity, aiming to reject modules with exceptionally low activity. However, experimental investigations revealed that the method of suppressing module activities must consider module dynamics. This leads to the categorization of module dynamics into two groups. The first group encompasses the dynamics of parametric modules and neural modules operating within the linear region of the TANH activation function. The second group comprises neural modules operating in saturation.

For modules in the first group, responsibilities are updated explicitly through an update rule that reduces the original responsibility r_i^{orig} by multiplying it with a suppression

factor u

$$r_i = r_i^{orig} \cdot u \tag{4.10}$$

where r_i^{orig} is the original responsibility from the individual $\mathbf{p}_i$ being updated, and r_i is the new responsibility value. The update factor u is a random variable, normally distributed over the interval $(0, 1]$ with parameters $m = 1$ and $\sigma = 1$. This parameter configuration typically prevents abrupt suppression of the module being updated. If the update fails to improve modeling accuracy or if the individual violates constraints, the original responsibility is reinitialized with a slightly higher value to enhance the suppression potential for subsequent attempts.

$$r_i = r_i^{orig} \cdot \mathcal{N}(0,\ 0.33 \cdot |1 - r_i^{orig}|) \tag{4.11}$$

The dependence of the variance σ on the original responsibility enables fine-tuning, ensuring that the new value r_i is generated closer to the original value r_i^{orig} as the latter approaches the upper limit of its valid range. The variance σ was scaled to 0.33, a value that demonstrated high performance in preliminary runs and was consistently used throughout all experiments in this thesis.

Multiplying the module output by a constant value does not alter the non-linearity of the module output, and consequently, does not affect the objective function. This implies that the explicit responsibility update does not affect the updating of module states. Consequently, the two updates do not interfere with each other and can be interleaved during algorithm execution. Experimental data demonstrates that evolutionary tuning with the update rule (4.10) exhibited consistent progress for the considered types of parametric modules and for neural modules operating within the linear region of the TANH activation function. Figure 4.6 illustrates the steady convergence of the total ensemble output towards the given time series throughout the synchronization sequence when update rule (4.10) is applied.

To prevent detrimental error progression and identification failures, the explicit update rule (4.10) is not applicable to modules of the second group, whose neurons operate in the saturation region of the TANH activation. This issue stems from the challenging module dynamics, which are difficult to tune because of the neuron states changing only in very low decimal digits despite significant variations in neuron input. This results in the formation of objective function plateaus, leading to DE stagnation and prolonged periods of performance stagnation, as observed in [LZ00]. Consequently, module dynamics tuning fails. The exceedingly slow tuning progress precludes separate updating of responsibilities from module states, rendering the explicit update rule (4.10) ineffective for these dynamics. To ensure algorithm convergence, responsibility updates are coupled with module state updates – module responsibility is implicitly updated through the suppression of module states $\mathbf{x}_m(t)$. This suppression effectively shifts the neuron operating point from the saturation region to the linear region of the TANH

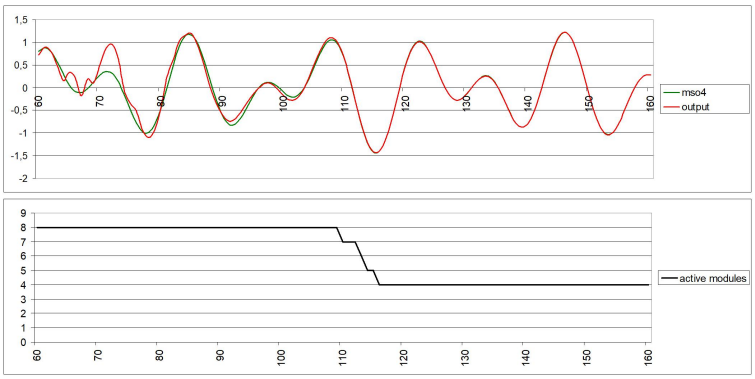

Figure 4.4.: Steady approaching of the total ensemble output to the given MSO4 time series with identification of the four relevant neural modules when the rule (4.10) is used for the responsibility update.

activation; each element of the vector $\mathbf{x}_m(t)$ is updated independently as

$$x_{ij} = x_{ij}^{orig} \cdot u_x \tag{4.12}$$

where x_{ij} is the updated module state, u_x is the update factor, and x_{ij}^{orig} is the original module state. The saturated module states exhibit a wide range of variation, as defined in (3.6), with values spanning several orders of magnitude. This implies that linear suppression through multiplication by a constant, as in the update rule (4.10), would be ineffective for them. Incorporating an additional exponential scaling factor s in Eq. (4.13) achieved the desired suppression effect, albeit at the cost of increased complexity compared to rule (4.10).

$$u_x = s \cdot u \tag{4.13}$$

High performance was achieved in experiments with modules for triangular oscillatory signals when the scaling factor was defined as an exponential function of the number of superfluous modules, as follows

$$s = e^{-\Delta_u} \tag{4.14}$$

where Δ_u is the estimated number of superfluous modules. The dependence of s on Δ_u compels the algorithm to apply significantly stronger suppression to the current module when a majority of modules in the ensemble are superfluous. If the current module is superfluous, the strong suppression substantially reduces its "spoiling" contribution to the total output, thereby not compromising modeling quality. Conversely, if the module is relevant, its suppression degrades modeling accuracy, leading to the rejection of the updated state. On the other hand, when Δ_u is small, the contribution from superfluous modules does not vanish abruptly but gradually diminishes. This accommodates the co-existence of relevant and superfluous modules at preceding time steps, where relevant modules were tuned to partially compensate for the contribution from superfluous ones.

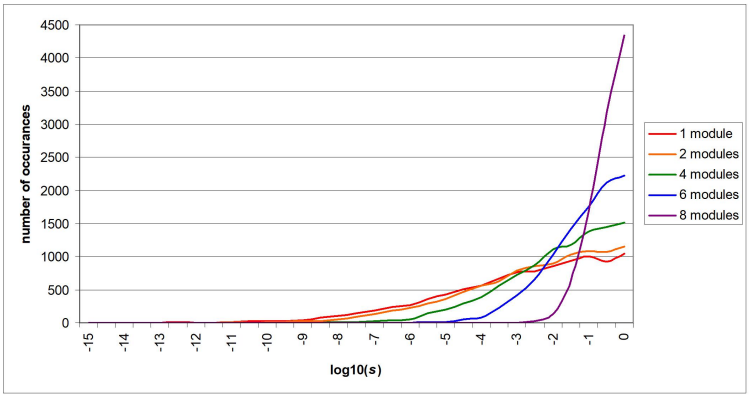

Figure 4.5.: Distributions of the scaling factor s which was randomly generated for effective suppression of irrelevant modules in ensembles of recurrent neural modules as defined by (4.14). The distributions were obtained for mixtures comprising varying numbers of dynamic components. Each distribution was derived from 10^4 accepted module state updates.

Consequently, a gradual reduction in the contribution from superfluous modules is preferred, as it prevents a drastic increase in error, thereby avoiding rejection of the updated state x_{ij}. As illustrated in Figure 4.5, larger changes in module contributions are tolerated for a smaller number of relevant neural modules.

The absence of information regarding the contents of the given time series renders the estimate Δ_u highly uncertain, posing a risk of synchronization failure. To mitigate this issue, it proved advantageous to stochastically diversify the estimate Δ_u across individuals, rather than updating all individuals with the same value. In the proposed approach, each x_{ij} was updated with a unique value of Δ_u, drawn from a normal distribution with a mean of 1 and a variance σ_u representing the estimated number of superfluous modules. Only the positive tail of the normal distribution was considered to ensure the scaling factor s remained within the interval $(0, 1]$. The variance σ_u was calculated as the normalized difference (4.15) between the sum of average contributions from all available modules and the maximum observed value in the sequential data. The normalization factor is the average contribution from a single module.

$$\sigma_u = \frac{\sum_{m=1}^{M} \overline{o_m} - \max(|y(t)|)}{\frac{1}{M}\sum_{m=1}^{M} \overline{o_m}} \tag{4.15}$$

where $\overline{o_m} = 0.5 \cdot (|\omega_m^{min}| + |\omega_m^{max}|)$ is the average magnitude of the contribution from the m^{th} module, with the output range limits ω_m^{min} and ω_m^{max} obtained from (3.7); $y(t)$ denotes the synchronization sequence from the start to the current time step t_c. It is expected that, for a large time step t_c, the value $\max(|y(t)|)$ will accumulate sufficient information to yield a reliable estimate for the number of superfluous modules. Equation (4.15)

assumes balanced module contributions with minimal variance in their responsibilities. Any potential violation of this assumption is mitigated by stochastic diversification, which provides a unique scaling factor s for each update, thereby reducing the risk of synchronization failure. Furthermore, stochastic diversification positively impacts DE convergence. In addition to shifting the vector $\mathbf{x}_m(t)$ into the linear region of the TANH activation, it varies the ratios between elements in $\mathbf{x}_m(t)$, thus preventing DE method stagnation. Experimental data from the MST time series demonstrates that the implicit update rule (4.13) facilitated consistent progress for neural modules operating in saturation, whereas synchronization failures occurred when the explicit update rule (4.10) was applied.

Downgrading of the module responsibilities

Encapsulating distinct module attributes, namely responsibility and states, within a single individual requires consideration of their varying impacts on the module's output signal characteristics. Consequently, their updates exert differing effects on the module output and its contribution to the ensemble's error signal. A module's responsibility, represented by a single value, directly defines the amplitude of its output and needs minimal effort for rapid, albeit localized, error reduction. Conversely, module states, represented as a vector, primarily define the form of the module's output signal. Achieving accuracy improvements necessitates simultaneous variation of multiple states, demanding greater effort for comparable error reduction than adjustments to module responsibilities. Because of the lower effort, module responsibilities exert a more pronounced influence on error tuning. Under these circumstances, uniform handling of both attributes becomes untenable, as the algorithm tends to exploit the stronger influence of module responsibilities for faster error reduction. This predisposition leads to premature convergence and yields poor results on complex time series. Figure 4.6 demonstrates the positive impact of distinct state and responsibility handling on synchronization convergence for the most complex time series considered, composed of either differentiable or non-differentiable dynamic components. As evidenced, distinct handling is indispensable for convergence on mixtures of triangular oscillatory signals and advantageous for mixtures of sine waves to enhance ensemble accuracy.

To prevent the misuse of module responsibilities and mitigate the risk of premature convergence, the synchronization algorithm must update them separately from module states, meaning that only one of these attributes is updated at any given time. The decision regarding which parameter to update is made randomly, with a probability μ_r set to 20% based on observed high performance in preliminary experiments. This 20% probability indicates that module states were updated, on average, five times more frequently than responsibilities.

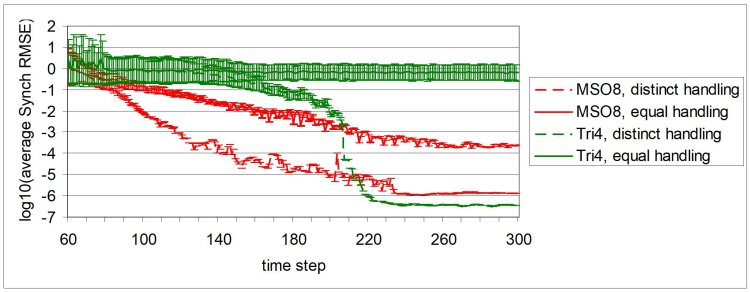

Figure 4.6.: Effect of distinct handling of module responsibilities and module states on the convergence of the synchronization method on the compositions of differentiable sinusoidal signals (red lines, representing the average synchronization RMSE with confidence intervals) and on the compositions of non-differentiable triangular oscillatory signals (green lines).

4.2.3. Gradient-based tuning

The application of evolutionary algorithms for module tuning offers significant flexibility in module type selection and is applicable regardless of the given time series' properties. However, evolutionary tuning relies on stochastic search, which does not utilize gradient information. The incorporation of gradient information could accelerate synchronization, motivating the investigation of gradient optimization methods for tuning module dynamics. Gradient methods seek the optimum along the gradient direction, presenting a potential avenue to further enhance the algorithm's convergence rate. Similar to evolutionary tuning, the gradient-based approach maintains a population of candidate solutions for each module within the ensemble. This enhances resilience against local optima and enables the accumulation of information about the given time series at the population level, without focusing solely on the dynamic module's current state. The dynamic module then acts as a filter, discarding unsuitable module states or favoring suitable ones.

Unlike evolutionary tuning, the gradient-based approach depends on formulas for calculating the error gradient as a function of evolving module dynamics. These formulas are contingent upon the structure of the modules employed and have been analytically derived for each module type considered. The following sections detail the derivation of gradient formulas for ensembles of recurrent neural modules and parametric sine wave modules. Given that the output dynamics of triangular and rectangular oscillatory modules cannot be analytically described using continuously differentiable functions, a rigorous derivation of their gradients is not feasible. Consequently, related time series were excluded from experimental investigations of gradient-based synchronization.

Recurrent neural modules

In the gradient-based procedure, internal module states are adjusted proportionally to the error gradient at the current time step, as follows.

$$\Delta x_n = -\eta \cdot \frac{d}{dx_n} E(x_n) \tag{4.16}$$

where η is the learning rate, and $E(x_n)$ denotes the discrepancy between the given time series and the ensemble's output. Analogous to the evolutionary approach, the adjustment is based on the error signal from all time steps within the current sliding window of size T_W.

$$E(x_n) = E_n(x_n) + E_{n+1}(x_n) + \cdots + E_{n+T_W-1}(x_n) \tag{4.17}$$

Thus, the gradient of the total error is the sum of gradients calculated at each time step within the sliding window.

$$\frac{d}{dx_{ni}} E(x_n) = \frac{d}{dx_{ni}} E_n(x_n) + \frac{d}{dx_{ni}} E_{n+1}(x_n) + \cdots + \frac{d}{dx_{ni}} E_{n+T_W-1}(x_n) \tag{4.18}$$

where i is the index in the internal state $\mathbf{x}_n = \{x_{n1}, x_{n2}, \ldots, x_{nR}\}$ of the module currently being tuned. Upon defining the total error as the sum of squared discrepancies between the ensemble's output and the target values of the time series, the derivative of the total error can be expressed as the sum of their respective derivatives.

$$\frac{d}{dx_{ni}} E(\mathbf{x}_n) = \sum_{t=n}^{n+T_W-1} \frac{d}{dx_{ni}} (\mathbf{y}(t) - \mathbf{o}_t(\mathbf{x}_n))^2 \tag{4.19}$$

where $\mathbf{o}_t(\mathbf{x}_n)$ is the total ensemble output at time step t, expressed as a recurrent function of the state $\mathbf{x}_n$ at the beginning of the sliding window. The subsequent formulas illustrate the dependence of the error derivative on the state $\mathbf{x}_n$ at the initial time step of the sliding window, prior to considering dependencies at subsequent time steps $\mathbf{x}_{n+1}$, $\mathbf{x}_{n+2}$, and so forth.

$$\frac{d}{dx_{ni}} E_n(\mathbf{x}_n) = -2 \cdot (\mathbf{y}(n) - \mathbf{o}_n(\mathbf{x}_n)) \frac{d}{dx_{ni}} \mathbf{o}_n(\mathbf{x}_n) \tag{4.20}$$

Considering the implementation of recurrent neural modules with ESNs and calculating their output using Eq. (A.1), the derivative of $\mathbf{o}_t(\mathbf{x}_n)$ is an L-dimensional vector:

$$\frac{d}{dx_{ni}} \mathbf{o}_n(\mathbf{x}_n) = \begin{pmatrix} r_{m1} \cdot (w^{out}_{11} \frac{d}{dx_{ni}} x_{n1} + w^{out}_{12} \frac{d}{dx_{ni}} x_{n2} + \cdots + w^{out}_{1R} \frac{d}{dx_{ni}} x_{nR}) \\ \vdots \\ r_{mL} \cdot (w^{out}_{L1} \frac{d}{dx_{ni}} x_{n1} + w^{out}_{L2} \frac{d}{dx_{ni}} x_{n2} + \cdots + w^{out}_{LR} \frac{d}{dx_{ni}} x_{nR}) \end{pmatrix} \tag{4.21}$$

where w_{ji} represent the connection weights between the i^{th} internal neuron and the j^{th} output neuron, and r_{mj} denote the responsibilities of the m^{th} module for the j^{th}

output. Due to the independence of modules within the ensemble, derivatives from other modules are zero and thus omitted in Eq. (4.21). The internal states at time step n are adjusted directly, implying that $\frac{d}{dx_{ni}} x_{ni} = 1$, while the derivatives of other state vector elements are zero, that is,

$$\begin{pmatrix} \frac{d}{dx_{ni}} \mathbf{o}_{n1}(\mathbf{x}_n) \\ \vdots \\ \frac{d}{dx_{ni}} \mathbf{o}_{nL}(\mathbf{x}_n) \end{pmatrix} = \begin{pmatrix} r_{m1} \cdot w_{1i}^{out} \\ \vdots \\ r_{mL} \cdot w_{Li}^{out} \end{pmatrix}. \tag{4.22}$$

Consequently, at the initial time step of the sliding window, the error derivative is computed as

$$\frac{d}{dx_{ni}} E_n(\mathbf{x}_n) = -2\mathbf{r}_m \left(\begin{pmatrix} y_1(n) - o_1(n) \\ \vdots \\ y_L(n) - o_L(n) \end{pmatrix}^T \cdot \begin{pmatrix} r_{m1} \cdot w_{1i}^{out} \\ \vdots \\ r_{mL} \cdot w_{Li}^{out} \end{pmatrix} \right). \tag{4.23}$$

In subsequent time steps, the internal dynamics of the modules evolve recursively and depend on the activation function of the reservoir neurons, specifically:

$$\frac{d}{dx_{ni}} \mathbf{x}_{n+1} = f'(\hat{\mathbf{x}}_{n+1}) \cdot \frac{d}{dx_{ni}} \hat{\mathbf{x}}_{n+1} \tag{4.24}$$

where

$$\hat{\mathbf{x}}_{n+1} = \mathbf{W}_{IN}^T \mathbf{u}_{n+1} + \mathbf{W}_{RES}^T \mathbf{x}_n + \mathbf{W}_{OFB}^T \mathbf{o}_n \tag{4.25}$$

is the weighted sum of input signals to the reservoir neurons at time step $n+1$, prior to applying the activation function, and $f'()$ denotes the derivative of the reservoir neuron activation function. In the conducted experiments, the reservoir neurons utilized the TANH activation function, with the derivative

$$f'(\hat{\mathbf{x}}_{n+1}) = 1 - TANH^2(\hat{\mathbf{x}}_{n+1}). \tag{4.26}$$

Due to recurrent signal propagation within the neural modules, the derivatives (4.24) and (4.26) depend on the initial internal state $\mathbf{x}_n$, on the derivatives (4.21) at the preceding time step n, and contribute to the error derivative at the subsequent time step $n+1$ as a function of the initial state $\mathbf{x}_n$.

$$\frac{d}{dx_{ni}} E_{n+1}(\mathbf{x}_n) = -2 \cdot (\mathbf{y}(n+1) - \mathbf{o}_{n+1}(\mathbf{x}_n)) \frac{d}{dx_{ni}} \mathbf{o}_{n+1}(\mathbf{x}_n) \tag{4.27}$$

where

$$\frac{d}{dx_{ni}} \mathbf{o}_{n+1}(\mathbf{x}_n) = \begin{pmatrix} r_{m1} \cdot (w_{11}^{out} \frac{d}{dx_{ni}} x_{(n+1)1} + w_{12}^{out} \frac{d}{dx_{ni}} x_{(n+1)2} + \cdots + w_{1R}^{out} \frac{d}{dx_{ni}} x_{(n+1)R}) \\ \vdots \\ r_{mL} \cdot (w_{L1}^{out} \frac{d}{dx_{ni}} x_{(n+1)1} + w_{L2}^{out} \frac{d}{dx_{ni}} x_{(n+1)2} + \cdots + w_{LR}^{out} \frac{d}{dx_{ni}} x_{(n+1)R}) \end{pmatrix} \tag{4.28}$$

Considering Eq. (4.24) leads to

$$\begin{pmatrix} \frac{d}{dx_{ni}} o_{(n+1)1} \\ \vdots \\ \frac{d}{dx_{ni}} o_{(n+1)L} \end{pmatrix} = \begin{pmatrix} r_{m1} \cdot (w_{11}^{out} f'(\hat{x}_{(n+1)1}) \frac{d}{dx_{ni}} \hat{x}_{(n+1)1} + \cdots + w_{1R}^{out} f'(\hat{x}_{(n+1)R}) \frac{d}{dx_{ni}} \hat{x}_{(n+1)R}) \\ \vdots \\ r_{mL} \cdot (w_{L1}^{out} f'(\hat{x}_{(n+1)1}) \frac{d}{dx_{ni}} \hat{x}_{(n+1)1} + \cdots + w_{LR}^{out} f'(\hat{x}_{(n+1)R}) \frac{d}{dx_{ni}} \hat{x}_{(n+1)R}) \end{pmatrix}. \tag{4.29}$$

Based on (4.25), the derivatives of the elements of $\hat{x}_{n+1}$ with respect to x_{ni} must be computed as follows.

$$\begin{pmatrix} \frac{d}{dx_{ni}} \hat{x}_{(n+1)1} \\ \vdots \\ \frac{d}{dx_{ni}} \hat{x}_{(n+1)R} \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{R} w_{1j}^{res} \frac{d}{dx_{ni}} x_{nj} + \sum_{j=1}^{L} w_{1j}^{ofb} \frac{d}{dx_{ni}} o_{nj} \\ \vdots \\ \sum_{j=1}^{R} w_{Rj}^{res} \frac{d}{dx_{ni}} x_{nj} + \sum_{j=1}^{L} w_{Rj}^{ofb} \frac{d}{dx_{ni}} o_{nj} \end{pmatrix}. \tag{4.30}$$

Applying these equations to compute (4.29) establishes the recursive dependence of the error gradient (4.27) on the initial state $\mathbf{x}_n$ at the beginning of the sliding window, as follows.

$$\frac{d}{dx_{ni}} E_{n+1}(\mathbf{x}_n) = -2\mathbf{r}_m \left(\begin{pmatrix} y_1(n+1) - o_1(n+1) \\ \vdots \\ y_L(n+1) - o_L(n+1) \end{pmatrix}^T \cdot \begin{pmatrix} \frac{d}{dx_{ni}} o_{(n+1)1} \\ \vdots \\ \frac{d}{dx_{ni}} o_{(n+1)L} \end{pmatrix} \right). \tag{4.31}$$

This equation is recursively applicable for computing the error gradient at all subsequent time steps $n+2$, $n+3$, and so forth, until the end of the sliding window. The sum of the error gradient (4.23) at the beginning of the sliding window and the iteratively computed (4.27) at subsequent time steps yields the total error gradient (4.18), serving as the basis for the gradient-based adjustment (4.16) of reservoir states in recurrent neural modules.

Besides, the module ensemble requires adjustment of module responsibilities. Analogous to the adjustment of internal module states in (4.16), responsibilities are adjusted proportionally to the error gradient, as follows.

$$\Delta r_m = -\eta \cdot \frac{d}{dr_m} E(r_m) \tag{4.32}$$

where $E(r_m)$ is the total error gradient, defined as a function of r_m instead of $\mathbf{x}_n$, as in (4.18). In (4.32), the module responsibility is indexed solely by m, indicating its time-independence, as responsibilities remain constant throughout the sliding window, unlike the dynamic module states. Consequently, the total error gradient is time-independent and must be derived as a function of $\mathbf{r}_m$, as follows.

$$\frac{d}{dr_{mi}} E(\mathbf{r}_m) = \sum_{t=n}^{n+T_W-1} \frac{d}{dr_{mi}} (\mathbf{y}(t) - \mathbf{o}(t))^2 \tag{4.33}$$

where i is the index in the responsibility vector $\mathbf{r}_m = \{r_{m1}, \ldots, r_{mL}\}$ of the m^{th} module, which corresponds to the i^{th} element of the ensemble's total output vector $\mathbf{o}(t)$; L represents the number of elements in the output vector.

$$\frac{d}{dr_{mi}} E(\mathbf{r}_m) = -2 \sum_{t=n}^{n+T_W-1} (y_i(t) - o_i(t)) \frac{d}{dr_{mi}} o_i(t) \tag{4.34}$$

where $y_i(t)$ and $o_i(t)$ are the elements of the target time series vector $\mathbf{y}(t)$ and the ensemble's total output vector $\mathbf{o}(t)$ at time step t, respectively. Considering equation (3.8) for computing the ensemble output yields

$$\frac{d}{dr_{mi}} E(\mathbf{r}_m) = -2 \sum_{t=n}^{n+T_W-1} (y_i(t) - o_i(t)) o_{mi}(t) \tag{4.35}$$

where $o_{mi}(t)$ is the i^{th} element of the m^{th} module's output vector $\mathbf{o}_m(t)$ at time step t. As seen, the formula lacks any recurrence, meaning the total error gradient (4.35) requires computation only once for the entire sliding window following each window shift. Nonetheless, the dependence of $\mathbf{o}_m(t)$ on the internal states necessitates that the adjustment of $\mathbf{x}_n$ be taken into account for computation.

Parametric modules

This section presents the formulas for gradient-based tuning of parametric modules. The reliance of this approach on module output derivatives imposes limitations on the types of parametric modules that can be used. For parametric modules representing triangular and rectangular oscillations, as considered in the experiments, these derivatives do not exist at all points along the time axis. This precludes the derivation of explicit formulas for these types of parametric modules. Conversely, gradient-based tuning is applicable for sinusoidal parametric modules, whose output is continuously differentiable and is explicitly defined as

$$o_m(t) = \sin(f_m \cdot t + \phi_m) \tag{4.36}$$

where f_m and ϕ_m are the frequency and phase of the sinusoidal oscillation generated by the m^{th} module. Similar to the tuning of recurrent neural modules, the adjustment of these parameters is based on gradient rules

$$\Delta f_m = -\eta \cdot \frac{d}{df_m} E(f_m) \tag{4.37}$$

and

$$\Delta \phi_m = -\eta \cdot \frac{d}{d\phi_m} E(\phi_m) \tag{4.38}$$

where η is the learning rate, and $E(\ldots)$ is the total error across the entire sliding window. The total error gradients for the respective parameters are the sums of the derivatives of

squared discrepancies at each time step in the sliding window.

$$\frac{d}{df_m}E(f_m) = \sum_{t=n}^{n+T_W-1} \frac{d}{df_m}(y(t) - o(t))^2 \tag{4.39}$$

$$\frac{d}{d\phi_m}E(\phi_m) = \sum_{t=n}^{n+T_W-1} \frac{d}{d\phi_m}(y(t) - o(t))^2 \tag{4.40}$$

This results in

$$\frac{d}{df_m}E(f_m) = -2 \sum_{t=n}^{n+T_W-1} (y(t) - o_t(f_m)) \frac{d}{df_m}o_t(f_m) \tag{4.41}$$

$$\frac{d}{d\phi_m}E(\phi_m) = -2 \sum_{t=n}^{n+T_W-1} (y(t) - o_t(\phi_m)) \frac{d}{d\phi_m}o_t(\phi_m) \tag{4.42}$$

where the gradients of the module output with respect to both parameters can be computed as

$$\frac{d}{df_m}o_t(f_m) = t \cdot \cos(f_m \cdot t + \phi_m) \tag{4.43}$$

and

$$\frac{d}{d\phi_m}o_t(\phi_m) = \cos(f_m \cdot t + \phi_m). \tag{4.44}$$

Given that the m^{th} module typically has different responsibilities for individual elements of the output vector $\mathbf{o}(t)$, the gradients of the total error are dependent on these respective responsibilities, as follows.

$$\frac{d}{df_m}E(f_m) = -2 \sum_{t=n}^{n+T_W-1} t \cdot \cos(f_m t + \phi_m) \begin{pmatrix} y_1(t) - o_1(t) \\ \vdots \\ y_L(t) - o_L(t) \end{pmatrix}^T \begin{pmatrix} r_{m1} \\ \vdots \\ r_{mL} \end{pmatrix} \tag{4.45}$$

$$\frac{d}{d\phi_m}E(\phi_m) = -2 \sum_{t=n}^{n+T_W-1} \cos(f_m t + \phi_m) \begin{pmatrix} y_1(t) - o_1(t) \\ \vdots \\ y_L(t) - o_L(t) \end{pmatrix}^T \begin{pmatrix} r_{m1} \\ \vdots \\ r_{mL} \end{pmatrix} \tag{4.46}$$

Analogous to the gradient-based tuning formula for responsibility in recurrent neural modules (4.35), the gradients of parametric modules lack any recurrent dependency. Consequently, (4.45) and (4.46) require computation only once per parameter adjustment for the entire sliding window following each window shift. Furthermore, computing the total error gradient for adjusting module responsibilities in parametric modules utilizes the same formula, (4.35), as used with recurrent neural modules. This identity stems from the fact that the linear combination of modules with responsibilities as multiplication

coefficients, as defined in (3.8), is independent of module type.

4.3. Constraints and their handling

By definition, synchronization represents a constrained optimization problem where the algorithm must identify the hidden states $\mathbf{x}_m^*(t)$ of the dynamic modules that minimize the discrepancy (3.1) between the model output $\mathbf{o}(t)$ and the given time series data $\mathbf{y}(t)$. The imposed constraints guide the search towards the global optimum by eliminating a priori unsuitable solutions. This enhances the algorithm's convergence and efficiency, as it avoids expending resources on evaluating forbidden areas within the search space. The boundaries of these areas are predefined as static constraints (3.6) and (3.7) to ensure the validity of the model output $\mathbf{o}(t)$. Additionally, adaptive constraints are employed to verify the plausibility of the ensemble's output against the incoming $\mathbf{y}(t)$ sequence data. These two types of constraints complement each other to maximally narrow the relevant search space and to guarantee an acceptable solution by the end of the synchronization sequence.

4.3.1. Static constraints

To satisfy the static constraints, the algorithm must ensure that the module states remain within the valid ranges established during the design phase. The interval (3.6) defines the valid range for each internal state in each available module, and (3.7) defines the valid range for each element of the output vector. Individual updates and evolving module dynamics can yield invalid values that exceed these valid ranges. Such values prevent the module from maintaining the necessary dynamics and must be rejected or corrected by the algorithm. The independence in tuning the internal states $x_{mj}(t)$ allows the correction of potential invalid values to obtain new values that satisfy (3.6). However, the module outputs $o_{mj}(t)$ are not directly tuned but depend on the module states. Consequently, even if the algorithm maintains the module states within the valid range, the evolving internal dynamics can still cause $o_{mj}(t)$ to fall outside of it. In this context, the invalidity of $o_{mj}(t)$ serves as an indicator of an imbalance among the elements in the vector $\mathbf{x}_m(t)$. Given that this imbalance remains unknown, rejecting the entire module state is considered the only viable approach for handling invalid module outputs.

Direct application of the valid ranges (3.6) and (3.7) proved to be largely ineffective for synchronization, which became highly susceptible to premature convergence. The solution was found in relaxing both constraints. The significance of constraint relaxation has been corroborated in numerous studies, such as [TS06], where the ϵDE algorithm demonstrated excellent results on constrained multi-modal problems. Constraint relaxation means "softening" the rigid boundaries of the known feasible region in the optimization problem to a limited extent. Specifically, the final solution must reside within the known feasible region; however, during the optimization process, minor

outliers are tolerated as temporary solutions. This provides the optimization method with greater flexibility in navigating paths towards the desired global optimum. In the context of synchronization, the known feasible regions (3.6) and (3.7) are extended to the following regions, (4.47) and (4.48).

$$x_{mj} \in [\gamma_{mj}^{min} - \Delta\mu_m \cdot |\gamma_{mj}^{min}|, \quad \gamma_{mj}^{min} + \Delta\mu_m \cdot |\gamma_{mj}^{min}|] \tag{4.47}$$

$$o_{mj} \in [\omega_{mj}^{min} - \Delta\mu_m \cdot |\omega_{mj}^{min}|, \quad \omega_{mj}^{min} + \Delta\mu_m \cdot |\omega_{mj}^{min}|] \tag{4.48}$$

where $\Delta\mu_m$ is the margin of the m^{th} module. The margin's extent is proportional to the magnitude of the range's boundary, ensuring a commensurate valid range. The positive effect of relaxation is evident in Figure 4.7, where relaxation enabled significantly improved synchronization accuracy on the MSO time series and proved indispensable for the correct decomposition of triangular oscillatory signal mixtures.

While nearly inconsequential for parametric modules, the setting of $\Delta\mu_m$ significantly impacts the synchronization of recurrent neural modules, whose evolving dynamics can extend far beyond the known valid range. In experiments, performance improvements were typically observed for very small values of $\Delta\mu_m < 0.1$. The parameter's setting controls the selection pressure on the population; decreasing $\Delta\mu_m$ intensifies selection pressure, leading to greedier tuning. The optimal setting depends on the characteristics of the given sequence data $\mathbf{y}(t)$. Larger $\Delta\mu_m$ values are suitable for synchronization on longer sequences of stationary oscillatory dynamics. For discontinuous data, $\Delta\mu_m$ can be reduced to zero to accelerate convergence on their short sequences, albeit with an increased risk of premature convergence. However, this risk is mitigated in tasks such as handwritten symbol sequence recognition. In such tasks, the classification of the given curve takes precedence over long-term synchronization accuracy; rapidly achieving a reasonably good local optimum is generally sufficient for symbol curve recognition, rendering prolonged searches for the global optimum unnecessary.

Handling of static constraints

The literature on constrained optimization offers numerous methods to confine the real-valued internal states $\mathbf{x}_m(t)$ and module outputs $\mathbf{o}_m(t)$ to their respective feasible regions within the search space. These methods span from general constrained optimization techniques [Ber82] to specialized handling employed in differential evolution [PSL05], where invalid values are rejected by assigning them exceptionally high fitness values, ensuring they are never selected. Differently, in [LZ99], invalid values are forced into the valid range by replacing them with random values from within it. This approach results in more frequent objective function evaluations compared to the rejection method, leading to denser exploration of the search space and a reduced risk of missing the global optimum.

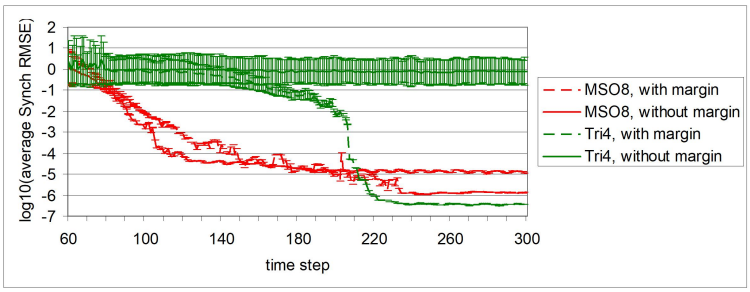

Figure 4.7.: The effect of static constraint relaxation on synchronization convergence on mixtures of differentiable sinusoidal signals (green lines are the average synchronization RMSE with the standard deviation interval) and non-differentiable triangular oscillatory signals (red lines represent the average synchronization RMSE).

The distinct functional roles of $\mathbf{x}_m(t)$ and $\mathbf{o}_m(t)$ necessitate different constraint handling approaches. The state vector $\mathbf{x}_m(t)$ serves as the argument for the module output $\mathbf{o}_m(t)$ and seeds the dynamics of $\mathbf{o}_m(t)$ at the onset of the sliding window. Although all elements of $\mathbf{x}_m(t)$ are valid, the evolution of $\mathbf{o}_m(t)$ dynamics within the sliding window or the progression to the next time step after shifting the window can result in an invalid $\mathbf{o}_m(t)$ value. As the cause of $\mathbf{o}_m(t)$ exceeding its range cannot be determined, $\mathbf{x}_m(t)$ must be rejected alongside $\mathbf{o}_m(t)$. At the same time, if the tuning process yields an invalid element within $\mathbf{x}_m(t)$, it is repaired by replacing it with a random value from the valid range. During population initialization, the random generation of $\mathbf{x}_m(t)$ often requires iterative individual generation until constraint (4.48) is met. Similar to the previously proposed initialization method employing opposite numbers [RTS08], this increases initial computational effort, which is subsequently offset by improved convergence in later time steps due to populations being sampled closer to the global optimum.

The proximity of the original value to the boundaries of the valid range (4.47) increased a risk of generating invalid module state values, particularly when the learning rate or the DE parameter F were high. Given that the original state value x_{mj} resulted from tuning the module state over the preceding time step sequence, repairing a newly generated invalid value with a random value from the vicinity of the original x_{mj} is likely to improve it. The repairing random value is sampled from a normal distribution, $x_{mj}^{new} = \mathcal{N}(x_{mj}^{orig}, \sigma_{repair})$, as is common in evolutionary strategies [HO01; Rec73]. This new value, x_{mj}^{new}, replaces the original x_{mj}^{orig}, as schematically depicted in Figure 4.8. To minimize the probability of generating an invalid x_{mj}^{new} value, the variance can be adaptively chosen, depending on the distance from the original value x_{mj}^{orig} to the nearest boundary of its valid range, as follows.

$$\sigma_{repair} = \frac{1}{3} \times \{|x_{mj}^{orig} - \hat{\gamma}_{mj}^{min}|,\ |x_{mj}^{orig} - \hat{\gamma}_{mj}^{max}|\} \tag{4.49}$$

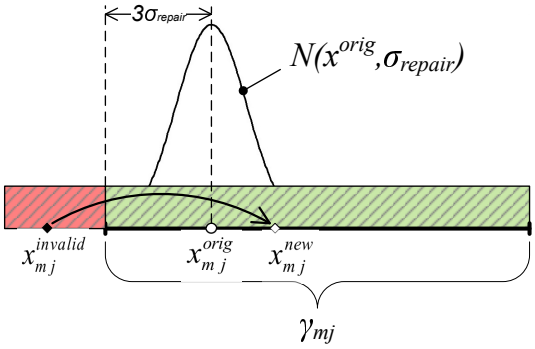

Figure 4.8.: Handling of static constraints for the j^{th} state in the m^{th} module. The curved arrow shows the replacement of the initially generated invalid value $x_{mj}^{invalid}$ with a new, randomly generated value x_{mj}^{new}. The variance σ_{repair} is contingent upon the distance between the original value x_{mj}^{orig} and the nearest boundary of the valid range γ_{mj}.

where $\hat{\gamma}_{mj}^{min}$ and $\hat{\gamma}_{mj}^{max}$ represent the lower and upper boundaries of the valid range (4.47), including the module margin $\Delta\mu_m$. In this formula, the coefficient $\frac{1}{3}$ is included to ensure that the majority of generated values (approximately 97%) remain within the valid range. If the generated value x_{mj}^{new} falls outside the range, then generation must be repeated.

The handling of static constraints also involves defining the valid range. Simply defining the valid range as an interval between the minimum and maximum values disregards the actual distribution of the module state within this interval, assuming it to be uniform. Initially, this seemed unsuitable for recurrent neural modules with neurons operating in the saturation region of the TANH activation function. For instance, such neurons were found in modules modeling triangular oscillation signals. Considering the actual distribution of neuron states appeared to offer potential performance improvements. This suggested representing the valid range (3.6) as a conjunction of two intervals: one for negative saturation at "-1" and the other for positive saturation at "+1". The intervals were separated by a distance to reflect the zero distribution density of neuron states near the value "0". However, experiments did not reveal any advantage in treating neural states as non-uniformly distributed random variables; in fact, experiments with triangular signal mixtures showed even poorer performance. Nevertheless, an advanced implementation of the valid range that reflects the actual distribution of neuron states might still offer benefits in the future. For example, applying relaxation to the intervals, where an invalid value is not immediately discarded after an update but mapped to the other interval with a delay, could be advantageous.

4.3.2. Adaptive constraints

Online module tuning implies that the time series is not available for synchronization all at once, but rather becomes available incrementally through shifting of the sliding window. Initially, this might seem problematic because of an increased risk of premature convergence. However, a closer examination reveals a distinct advantage of the online approach for synchronization: it enables the algorithm to focus on narrow regions in the search space immediately after initialization. These module states generate the ensemble's total output within a range constrained by the sequence data from the sliding window at the first time step, $t = 1$. As the sequence progresses, the algorithm adapts the valid range ω of the total output to the sequence data encountered up to the current time step t_c, i.e.,

$$o_j(t) \in [\omega_j^{min}, \omega_j^{max}] \tag{4.50}$$

where $\omega_j^{min} = \min(y_j(t))$ and $\omega_j^{max} = \max(y_j(t))$ over the range $t = 1, \ldots, t_c$, with j representing an element in the total output vector $\mathbf{o}(t)$. The valid range ω tends to expand incrementally throughout the sequence. Indirectly, this has a consequential effect on the search space, as its volume expands due to the fact that, in general, an output signal is generated by individual module states from a wider spread. The volume of the search space is minimal at $t = 1$. This initially small volume provides the algorithm with an opportunity to enter the attractor of the global optimum, thereby ensuring its convergence.

Handling of adaptive constraints

The adaptive constraint (4.50) is hierarchically superior to the static constraints (4.47) and (4.48), guiding the synchronization at a level above the individual modules. This implies that the adaptive constraint operates on module combinations without altering the internal dynamics of the modules themselves. Assuming (4.47) and (4.48) are satisfied, a violation of the known data range (4.50) by a particular module combination indicates that the module outputs are incompatible for accurate reconstruction of the given time series. In contrast, a combination of modules that satisfies (4.50) is likely to correctly identify the dynamics of the time series. Therefore, maintaining such module combinations is advantageous, and their mutual consistency should be verified in subsequent time steps. Consequently, the algorithm employs the adaptive constraint (4.50) to differentiate between incompatible and compatible module combinations.

Analogously to the violation of the static constraint (4.48) by an invalid output of a single module, a failed ensemble cannot be repaired because of the inability to discern which modules produce improper output dynamics. However, this does not necessitate rejection of the failed module combination. Certain modules within the ensemble may exhibit dynamics that approximates their targets. To mitigate the risk of discarding such modules and to ensure convergence, these modules should be retained within the population, albeit not within the same ensemble. Attempting to replace modules with

those from other ensembles is a viable strategy. Therefore, all failed ensembles undergo permutation, which splits the ensembles and repurposes their modules as building blocks for new ensembles in subsequent time steps. This approach minimizes the risk of losing potentially valuable modules, as corroborated by the study in [GSM08].

4.4. Evaluation of the updated module state

Updating the module states alters the dynamics within the corresponding modules and, consequently, the total output of the ensemble. To achieve the targeted synchronization of module dynamics, the algorithm evaluates these alterations, accepting them if they improve the module's behavior; otherwise, the alterations are rejected. Specifically, accepting an update entails retaining the updated state in the population instead of the original state. Given the synchronization objective defined by (3.1), the primary goal of tuning is to minimize the discrepancy between the network's output $\mathbf{o}(t)$ and the target time series $\mathbf{y}(t)$. However, the evaluation of an updated state is not solely based on reproduction accuracy but must also consider the validity of the updated module states. Incorporating validity constraints restricts the multitude of candidate dynamics that satisfy (3.1), compelling the algorithm to explore alternative search directions. This approach mitigates greediness and reduces the risk of premature convergence, thereby promoting the gradual shaping and harmonization of module dynamics.

The presence of two attributes – (1) validity of the module output and (2) overall modeling accuracy – required the development of a selection procedure that evaluates updated states for improvement according to each attribute. The evaluation procedure is schematically depicted in Figure 4.9. This procedure defines distinct acceptance criteria for each attribute and prioritizes them hierarchically. Validity of the module output takes precedence, meaning that if an updated module state enhances validity, the update is accepted irrespective of its impact on modeling accuracy. Conversely, the updated module state must be rejected if it leads to an anomalous flow of the module output that exceeds the valid range. Indeed, it is logical to focus on improving modeling accuracy only for modules that produce potentially viable output signals. If the update neither enhances nor diminishes the output signal flow, its effect is evaluated in the subsequent stage, where the update is accepted or rejected based on whether modeling accuracy has improved.

The validity of updates is assessed using condition (4.48), which is satisfied when the module output $\mathbf{o}_m(t)$ remains within the valid range over a sliding window. Given the random initialization, populations typically contain numerous module states that generate outputs outside the valid range. Validation-based acceptance rapidly compels all module states within the population to produce outputs within the valid range shortly after the start of synchronization, as illustrated in Figure 4.10. In this figure, the curves representing the number of accepted states exhibit non-zero values solely at the onset of the synchronization sequence, spanning 10-15 time steps post-initialization, until the

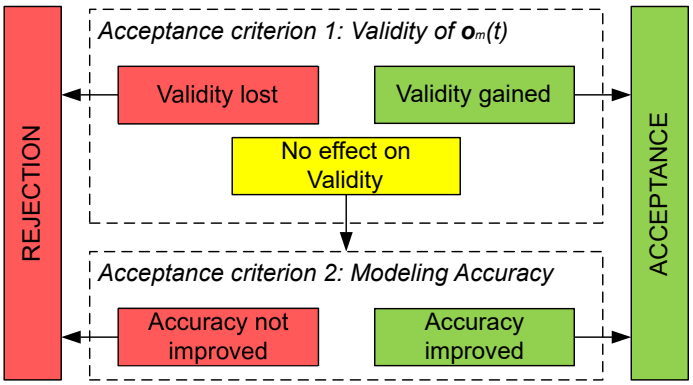

Figure 4.9.: An overview of the two-criterion evaluation of updated module states. In the initial stage, the algorithm verifies the updated module output against the valid range, accepting the update if the module output no longer violates this range. Subsequently, in the following stage, the update is accepted based on improved modeling accuracy, provided there was no deterioration of module dynamics according to the validity criterion.

population converges to the region of valid module states.

The increasing amount of module states with valid $\mathbf{o}_m(t)$ intensifies the search closer to the global optimum, thereby enhancing the likelihood of convergence to the global optimum, which corresponds to the hidden dynamic component to be identified. The growing number of accuracy-improving updates indicates a greater effort in fine-tuning the module output signal. In Figure 4.10, the curves representing accepted error improvements exhibit a rapid increase for MSO4 and MSO8. Conversely, the gradually decreasing trend of the curve for Triangle4 suggests a challenging search problem and its dependency on the characteristics of the considered time series.

The data presented in Table 4.1 underscores the importance of evaluation with respect to validity. It contrasts the achieved performance with and without the application of the acceptance criterion for $\mathbf{o}_m(t)$ validity. In the absence of validity checks, the resulting accuracy deteriorated significantly for the imbalanced MSO4 sequences, and even no convergence was attained for the more complex imbalanced MSO8 and Triangle4 sequences. At the same time, for the balanced time series, the removal of validity checks from the evaluation procedure did not result in a substantial decline in accuracy.

The evaluation of updates with respect to modeling accuracy is performed using the modeling accuracy criterion, which employs the error $e_{T_W}(t_c)$ from Eq. (4.5). An updated module state is accepted if it yields a lower error $e_{T_W}(t_c)$ compared to the error preceding the update. The inherent elitism of the modeling accuracy criterion is a significant weakness, potentially leading to convergence failure if evaluation relies solely on this criterion. This is exemplified by the data in Table 4.1, which highlights the necessity of incorporating the validity criterion and assigning a lower priority to

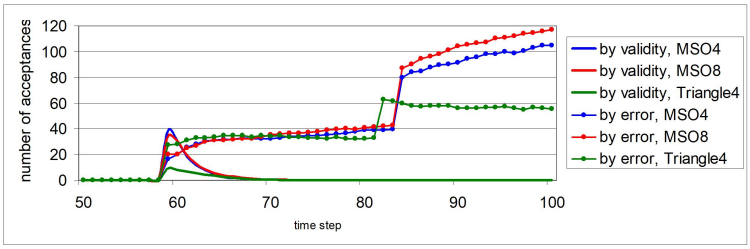

Figure 4.10.: Development of the number of accepted updates generated during the evolutionary tuning of neural ensembles over the synchronization sequence for time series MSO4 (blue line), MSO8 (red line), and Triangle4 (green line). Updates were accepted if all resulting module states fell within the validity range (solid lines), or if the updated state improved the accuracy of time series reproduction over the sliding window (solid lines with circles). The depicted numbers represent averages over 100 independent runs. The noticeable "jump" in the curves at time step 85 was attributed to an increment in the number of generated updates, which gradually increased throughout the synchronization sequence.

the modeling accuracy criterion (see Figure 4.9). This lower priority mitigates elitism, while its combination with the validity criterion ensures sufficient diversity of module dynamics within the population, preventing premature convergence to an erroneous local optimum outside the valid range, thereby safeguarding correct identification.

The evaluation process not only leads to the acceptance of updated states but also enables the rejection of detrimental updates that degrade module performance according to either criterion. The evolution of the number of rejected updates is depicted in Figure 4.11. As observed in the figure, the number of updates rejected due to validity decreased as the algorithm focused on the region proximal to the global optimum. Rejections based on error diminished as modeling accuracy improved successfully for MSO4 and MSO8. Conversely, slightly more rejections occurred for Triangle4, where accuracy improvement proved more challenging. A direct comparison of the number of acceptances and rejections for the same time series in Figure 4.10 and Figure 4.11 explains the factors contributing to the lower convergence rates observed for non-sinusoidal oscillatory time series compared to sinusoidal ones. Despite quantitative variations in the curves representing accepted updates, the two-criterion evaluation renders them qualitatively similar, exhibiting clearly distinguishable phases: initial exploration for an approximate optimum of module states, followed by refinement towards globally optimal module states, irrespective of the processed time series.

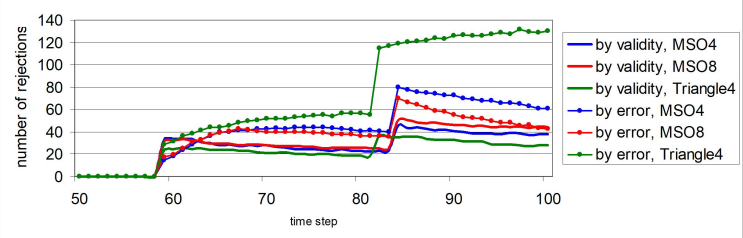

Figure 4.11.: Development of the number of rejected updates generated during the evolutionary tuning of neural ensembles over the synchronization sequence for time series MSO4 (blue line), MSO8 (red line), and Triangle4 (green line). Updates were rejected if any resulting module state fell outside the validity range (solid lines), or if the updated state vector degraded the accuracy of time series reproduction over the sliding window (solid lines with circles). The depicted numbers represent averages over 100 independent runs.

4.5. Identification and harmonization of co-evolving modules

Minimizing the discrepancy between the given time series $\mathbf{y}(t)$ and the ensemble's total output $\mathbf{o}(t)$ necessitates optimal module activity within the ensemble. This implies that the ensemble maintains no irrelevant activity and that the dynamics of co-evolving modules are mutually consistent; that is, the modules accurately reproduce the corresponding co-evolving dynamic components within the given time series. Given that the content of the time series is unknown a priori, the dynamics within initial randomly generated ensembles of the module state are unlikely to align, resulting in a substantial discrepancy between $\mathbf{y}(t)$ and $\mathbf{o}(t)$. As suggested by Eq. (4.6), successful tuning hinges on the contextual dynamics of co-evolving modules within a specific ensemble – a greater mismatch between them diminishes the likelihood of successful module tuning. The search for a suitable context is facilitated by permutation, which involves swapping module states between ensembles. However, improvement of modeling accuracy depends not only on a suitable context but also on mitigating superfluous module activity within the ensemble. Since all available modules are active at the onset of synchronization, initial ensembles of module states are prone to contain superfluous activity from irrelevant modules, which contribute to the error signal and impede further enhancement of modeling accuracy. An iterative process of swapping module states and tuning them aims to achieve significant suppression of activity in irrelevant modules. These modules can then be identified based on their minimal contribution to the ensemble's total output. Eliminating these disruptive modules allows the harmonization of output signals from confirmed relevant modules, leading to further reduction of the discrepancy between the given time series and its model.

Table 4.1.: Average relative amount of accepted and rejected module state updates during the evolutionary synchronization of neural ensembles on sinusoidal and triangular oscillatory time series. The relative amount is defined as the percentage of updates relative to the total number of module states generated in a single run. The data represents averages over 100 independent runs for each time series.

		Acceptance by Validity & Error			Acceptance by Error only		
	Time series	**Test RMSE**	**Accepted / Rejected by Validity (%)**	**Accepted / Rejected by Error (%)**	**Test RMSE**	**Accepted / Rejected by Validity (%)**	**Accepted / Rejected by Error (%)**
Balanced	**MSO4**	1.43×10^{-6}	0.06 / 0.75	99.93 / 99.24	1.43×10^{-6}	- / -	100 / 100
	MSO8	1.86×10^{-6}	0.03 / 0.32	99.96 / 99.67	1.88×10^{-6}	- / -	100 / 100
	Triangle4	5.24×10^{-7}	5.19 / 18.43	94.80 / 81.56	5.60×10^{-7}	- / -	100 / 100
Imbalanced	**MSO4**	2.24×10^{-3}	0.08 / 14.41	99.91 / 85.58	4.95×10^{-3}	- / -	100 / 100
	MSO8	2.95×10^{-3}	0.03 / 32.19	99.96 / 67.80	0.127	- / -	100 / 100
	Triangle4	2.04×10^{-4}	4.41 / 20.68	95.58 / 79.31	0.196	- / -	100 / 100

4.5.1. Permutation of the modules states

A necessary condition for accurate modeling of a given time series by an ensemble of co-evolving modules is the harmonization of module dynamics. This implies that the modules precisely reproduce the true components of the time series. The module outputs are then combined to form the ensemble's total output $\mathbf{o}(t)$, which exhibits minimal deviation from the target sequence $\mathbf{y}(t)$. The linear combination of modules investigated here does not inherently provide a mechanism for the co-adaptation of module dynamics. Consequently, the algorithm must actively facilitate their synchronization. Solely relying on module dynamics tuning is insufficient, as it only fine-tunes each module's dynamics within a constant context $\mathbf{C}_{m,p_i}(t)$ (see Eq. (4.6)), thereby risking convergence to a local optimum. This requires the synchronization algorithm to permute module states, aiming to create combinations of module dynamics that approximate the true dynamics. Modules gain a greater potential for fine-tuning when their states are positioned closer to the global optimum within the search space. This significantly increases the probability of correct identification of the true components. In the developed population-based synchronization, permutation is employed to mix module states. It alters the order of module states $\mathbf{x}_m(t)$ within the m^{th} population, seeking to establish a more suitable conjunction between them and their contexts $\mathbf{C}_{m,p_i}(t)$. Since module states are encoded by individuals, permuting the module states is equivalent to permuting the individuals in the population. Ideally, every permutation brings together an increasing number of mutually compatible module states. However, the rate of progress is contingent upon

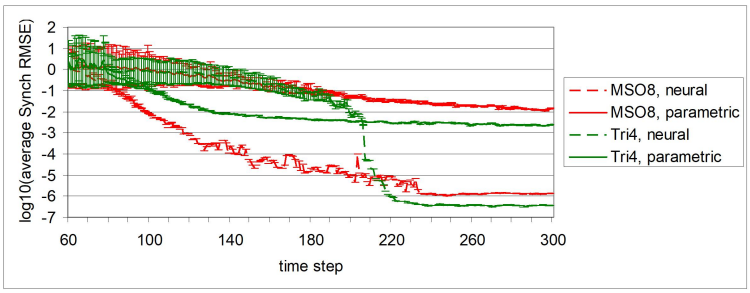

Figure 4.12.: Error curves for ensembles of neural modules (dashed lines) and parametric modules (solid lines) during synchronization on balanced time series: MSO8 (red lines) and Triangle4 (green lines). Note that Triangle4 is more challenging than MSO1, as originally shown in Figure 11 of [KOB21]. For each time series, the vertical lines indicate the narrowing spread of errors observed across multiple runs at each time step.

various factors, including the complexity and composition of the given time series, as well as the types of deployed modules. Differences in the impact of these factors on convergence result in variations in the rate of error reduction across different time series, as demonstrated in Figure 4.12.

Despite the intention to progressively construct more accurate ensembles from individuals within less accurately modeling ensembles, permutation does not recombine all accurately modeling ensembles to yield improved combinations. This is because recombining the best-performing ensembles does not guarantee the creation of new ensembles with superior performance compared to the originals. Depending on the synchronization progress, populations can exhibit high diversity, with module dynamics significantly deviating from the true dynamics. This is invariably the case for a period following initialization. Recombining individuals from highly diverse populations is unlikely to directly enhance modeling accuracy and poses a risk to overall convergence. To prevent synchronization failure because of the combination of incompatible module dynamics, permutation is restricted to a subset of the least promising ensembles.

The selection of ensembles for recombination is performance-dependent. At first glance, the maintenance of multiple populations suggests a natural approach to combine individuals as building blocks based on their performance scores, as implemented in other co-evolutionary methods. For instance, in [MM97], building blocks were combined based on their average performance in earlier generations within the top five combinations. In [Sch+07; GSM08], neurons accumulated fitness scores that reflected their performance across various RNNs in which they participated. Similarly, in [YM07], fitness was computed as the average performance over nine evaluations. However, the volatility of dynamic states and the online mode preclude the synchronization algorithm from accumulating performance statistics for individual building blocks. Instead, selection is based on their current modeling accuracy. A comparable permutation

method was employed for co-evolution in [GSM08], where current performance was used to probabilistically select building blocks. In contrast, this synchronization method employs rank-based selection on a restricted set of permutation candidates. This approach accommodates the dynamic nature of modules and applies the same acceptance criteria used for accepting updated individuals. These criteria aim to propagate only the module states that produce module outputs within the valid range. This strategy entails the disintegration of only undesirable ensembles, those with total outputs outside the valid range, violating the adaptive constraint (4.50). These ensembles are the candidates for permutation, and their evaluation prioritizes the validity of the total output over modeling accuracy.

Once the permutation candidates are identified, their individuals are reordered based on the rank of their modeling accuracy – individuals exhibiting higher modeling accuracy are positioned closer to the beginning of the population. The process of exchanging individuals between different ensembles is illustrated in Figure 4.13. To achieve this, the relative rank of the permutation candidates is determined. As shown in Figure 4.13, if an ensemble demonstrates superior modeling accuracy but occupies a higher index, it is exchanged with an ensemble that exhibits lower modeling accuracy but occupies a lower index within the population. If a permutation candidate's rank aligns with its modeling accuracy, its ensemble remains intact. It is important to note that the rank of individuals generally does not correspond to their index within the population following the preceding tuning, as tuning alters the original modeling accuracy of updated individuals.

Regardless of the diversity within the populations, the average modeling accuracy of newly formed ensembles is typically lower than that of ensembles unaltered by permutation. This decline in modeling accuracy stems from the inherent discrepancy between the module's output signal and the corresponding hidden component of the given time series $\mathbf{y}(t)$. Consequently, immediate improvement in modeling accuracy is not anticipated. Similarly, new ensembles are not expected to instantly satisfy the validity constraint (4.50) for their total output. Instead, the beneficial effect lies in achieving a potentially stronger alignment between module states $\mathbf{x}_{m,p_i}(t)$ and their contexts $\mathbf{C}_{m,p_i}(t)$ (see Eq. (4.6)), thereby enhancing the likelihood of ultimately satisfying the constraint (4.50). Furthermore, evaluating $\mathbf{x}_{m,p_i}(t)$ in different contexts prevents overfitting, as swapped module states are not permanently tuned within the same fixed context. Thus, these new combinations of module states offer an opportunity for accuracy improvement during subsequent module tuning sessions. Conversely, replacing module dynamics distorts the overall network dynamics, potentially leading to chaotic fluctuations in modeling accuracy and jeopardizing the algorithm's convergence. However, swapping dynamics within a single module at a time minimizes the negative impact of permutation on modeling accuracy, ensuring stable synchronization with steady convergence.

Following the permutation of the current population m, the subsequent population $m+1$ is updated and, thereafter, permuted. These steps are reiterated across all populations, leading to individuals from different populations progressively converging and ultimately

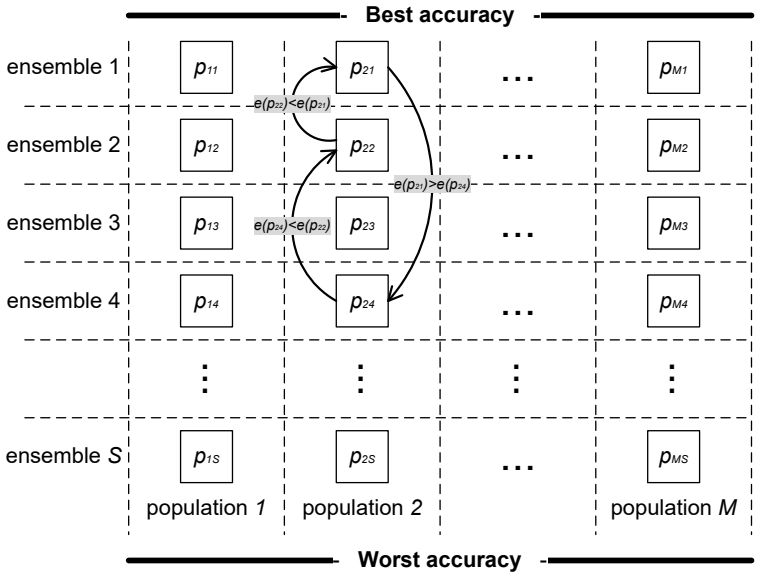

Figure 4.13.: Permutation of population 2 as an example of the current population. During this process, its individuals are exchanged between ensembles. Individuals exhibiting higher modeling accuracy (lower error e) are repositioned towards the beginning of the population (lower indices). Conversely, individuals exhibiting lower modeling accuracy (higher error e) are moved towards the end of the population. During the permutation of the current population, all other populations remain intact.

coalescing within the same ensemble. Their module dynamics become increasingly harmonious, facilitating more accurate reproduction of the given time series. While progressing along the sequence, permutation gradually diminishes in significance as the network's output $\mathbf{o}(t)$ converges towards $\mathbf{y}(t)$. In contrast to the initial stages of synchronization, where a majority of ensembles actively exchange individuals, few or no individuals are swapped by the end of synchronization. Concurrently, as the impact of permutation wanes, module tuning emerges as the dominant factor in enhancing the performance of established ensembles.

4.5.2. Rejection of irrelevant modules

Not all initially available modules are necessary for modeling the given time series $\mathbf{y}(t)$. By the end of synchronization, only the relevant modules significantly contribute to the ensemble's total output, while superfluous modules gradually diminish their activity. The identification of the time series necessitates the algorithm's ability to discriminate between modules based on their relevance. To achieve this, the synchronization algorithm examines each available module's contribution to the ensemble's output. A superfluous module's contribution remains below a predefined threshold ϵ throughout the sliding

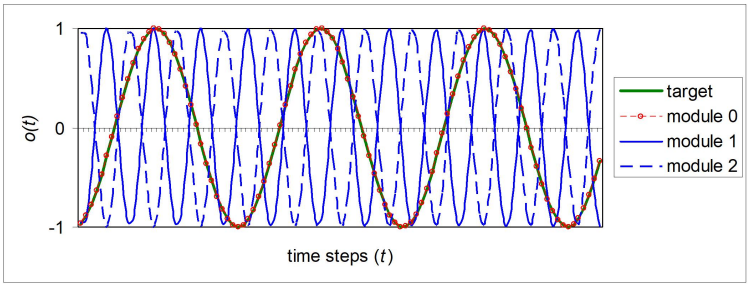

Figure 4.14.: Target sinusoidal oscillation and the outputs of the synchronized parametric sinusoidal modules. The output of module 0 perfectly aligns with the target oscillation. The outputs of modules 1 and 2, although non-zero, do not contribute to the ensemble's total output because of their counter-phase oscillation, resulting in their mutual annihilation. (Here, the frequencies of modules 0 and 2 were tuned to near-identical values of 0.69999999 and 0.70000001, and their phases to 1.749830 and -1.749827, respectively.)

window, specifically:

$$|r_m \cdot o_m(t)| < \epsilon \quad for \quad \forall t \in W_t. \tag{4.51}$$

The algorithm discards such modules, meaning they are excluded from consideration in subsequent time steps, and their corresponding populations are no longer evolved. In the presented experiments, the parameter ϵ was set to 1% of the maximum absolute value within the given time series for both neural and parametric modules.

The rejection of superfluous parametric modules requires an additional rule, which relies on the phenomenon of mutual annihilation occurring when two signals of identical amplitude and frequency oscillate in counter-phase. Consequently, if all conditions specified in (4.52) were satisfied for a given pair of parametric modules i and j, then both modules were discarded.

$$\begin{cases} |r_i \cdot o_i(t) + r_j \cdot o_j(t)| & < \epsilon \\ |\lambda_i - \lambda_j| & < \epsilon \\ ||\phi_i - \phi_j| - \pi| & < \epsilon \end{cases} \tag{4.52}$$

where the threshold ϵ was set to a very small value, specifically 10^{-3} for the experiments. Figure 4.14 illustrates the output of two out of three parametric modules, which were tuned to have equal amplitudes and frequencies but oscillate in counter-phase. As depicted, observing the module outputs independently reveals significant activity within the ensemble. However, when combined, two modules were synchronized to produce mutually canceling signals, leaving only a single oscillation that accurately reproduces the given time series $\mathbf{y}(t)$.

Beyond the identification of the time series, the rejection of superfluous recurrent modules is crucial for the stability of the resulting ensemble. It solves the problem

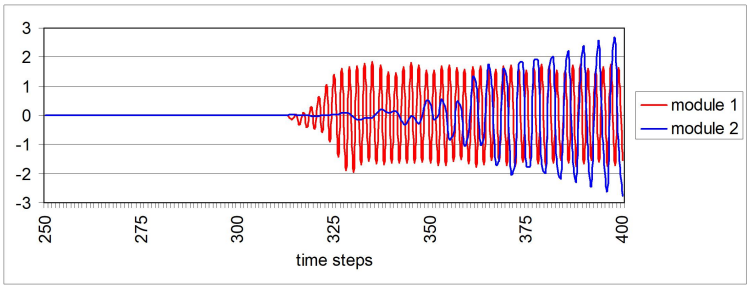

Figure 4.15.: Self-activation of two exemplary superfluous ESN modules. Their outputs, suppressed during synchronization until time step 300, rapidly increased because of internal signal amplification within the ESN reservoir after the completion of synchronization.

of self-activation in superfluous modules post-synchronization. Self-activation occurs when the initially weak internal dynamics of an unstable module are amplified through recurrent connections. During synchronization, module state tuning suppresses activity within these modules to exceptionally low levels. However, subsequently, self-activation can cause the module output to increase in magnitude, thereby overshadowing the contributions from relevant modules. This would corrupt the ensemble's output dynamics after synchronization, as illustrated in Figure 4.15.

In general, the rejection of superfluous modules facilitates achieving higher modeling accuracy and manages the consumption of computational resources. Both aspects are linked to the exclusion of irrelevant populations from co-evolution, which is analogous to dropping a part of the search space. This reduces the compound dimensionality of the search space and enables permutation to produce a more precise alignment between the internal dynamics of the remaining relevant modules. Evolving a smaller number of modules decreases the overall computational effort and enhances synchronization efficiency. Furthermore, removal of unnecessary modules from the ensemble minimizes its size, leading to improved generalization capabilities of the final model.

4.5.3. Harmonization of the modules

The rejection of superfluous modules and the tuning of relevant modules occur concurrently throughout synchronization. Their intensity varies along the sequence – initially, the algorithm identifies and removes superfluous modules; subsequently, it concentrates on harmonizing the dynamics of the relevant modules, aiming to refine alignment of their output signals for increasingly precise modeling of the given time series $\mathbf{y}(t)$. Indeed, fine-tuning the ensemble is largely ineffective in the presence of irrelevant modules, as they would disrupt the ensemble's output and impede the achievement of high accuracy.

The diminishing intensity of permutation throughout the sequence signifies the gradual transition from rejecting superfluous modules to harmonizing relevant ones. As illus-

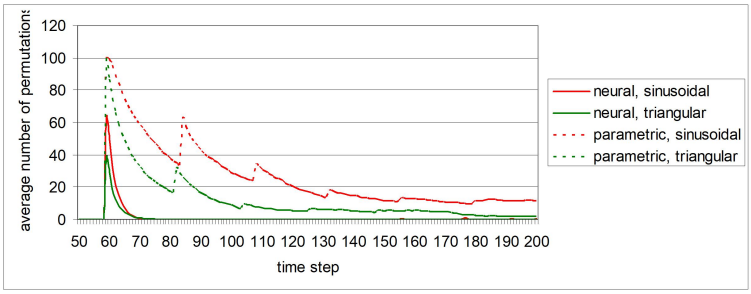

Figure 4.16.: Diminishing frequency of permutations over the synchronization sequence for compositions of sinusoidal and triangular oscillatory signals, using neural and parametric modules. The average frequency was computed from 100 runs. The abrupt increases in the parametric curves resulted from an increased number of generated module state updates.

trated in Figure 4.16, the intensity of permutations peaks at the sequence's onset. At this stage, the disruptive contribution from unneeded modules necessitates compensation through adaptation of the relevant modules' dynamics. This results in suboptimal module dynamics, leading to an increase in deviation between the ensemble's output and the modeled time series $\mathbf{y}(t)$ as the sliding window advances. This deterioration in accuracy induces volatility in combinations of module state, maintaining a high frequency of permutations. Following the removal of superfluous modules, the algorithm is no longer required to compensate for their disruptive contribution at each time step. Consequently, the combinations of relevant module states stabilize, and permutations become less frequent.

The dynamic adjustment of individual update counts facilitates the progressive harmonization of module dynamics. Beginning with a single update per individual in each population at the initial time step, the update count increases linearly to N^{max} by the sequence's end, as follows.

$$N(t) = 1 + N^{max} \cdot \frac{t - T_W}{T_{synch} - T_W} \tag{4.53}$$

where N^{max} is the maximum number of updates, T_W is the sliding window length, and T_{synch} denotes the synchronization sequence length. The application-dependent parameter N^{max} requires larger values for more complex modules that generate intricate output dynamics. The initial term "1" guarantees at least one update during the early synchronization time steps. Every individual update yields a module's trial state, which is evaluated against predefined criteria. Within the framework of DE, this corresponds to the evaluation of generated offspring during selection. At the sequence's onset, the limited number of updates minimizes the impact of local tuning, thereby reducing the risk of premature convergence. Subsequently, more frequent generation of trial states facilitates a denser exploration of the state space at each time step, leading to more precise

tuning of module dynamics. Towards the sequence's end, the intensified fine-tuning of stable module state combinations finalizes the harmonization of modules, resulting in the most accurate alignment of their dynamics.

Beyond adjusting the update count, varying the parameters of the optimization method employed can further enhance final accuracy. Experiments with evolutionary synchronization, conducted without applying the rejection rule (4.51), revealed that the activity of superfluous modules was substantially suppressed to approximately $o_i(t) \approx 10^{-6}$, yet exhibited persistent jittering at this level, failing to converge further towards zero. This behavior likely resulted from oscillations in the search around the global optimum. Consequently, tuning relevant modules might suffer from a similar problem, hindering the achievement of even greater accuracy. In the utilized DE optimization, shifting the balance towards exploitation by reducing the differential weight parameter F could yield a beneficial outcome, as documented in [DKC05]. In this study, F was linearly decreased from 1 to 0.5 throughout the evolutionary process, transitioning from exploration in earlier generations to exploitation in later generations. Smaller values of the scaling factor F reduced the variations in the produced offspring, leading to a more intensive search within the most promising areas of the search space and thus accelerating convergence towards the global optimum.

5. Decomposition of oscillatory time series

This chapter presents the synchronization of ensembles of oscillator modules on time series constructed from oscillatory dynamics. Both evolutionary and gradient-based tuning methods for recurrent dynamics are detailed and evaluated using stationary balanced and imbalanced time series, as well as non-stationary sequences characterized by time-varying content. A comparative analysis highlights the advantages and disadvantages of each approach. The experiments revealed consistent patterns in the method's behavior under diverse conditions. Thereat, both the properties of the input data and the chosen module type have a significant impact on the behavior of the method.

5.1. Performance indicators

Consistent with the majority of studies on time series modeling, the primary performance indicator for the investigated method was the resulting accuracy. In these experiments, it was quantified as the RMSE calculated on the test sequence following the completion of the synchronization process.

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\mathbf{y}(t) - \mathbf{o}(t))^2}, \tag{5.1}$$

where $\mathbf{o}(t)$ is the ensemble's total output, $\mathbf{y}(t)$ is the given time series, and T is the sequence length. Given that the time series was partitioned into synchronization and test sequences, the modeling error was calculated for each. Figure 5.1 illustrates the correlation between these errors, providing insight into the risk of overfitting during synchronization. As depicted, a strong correlation exists between them for both the simple MSO1 time series and the more complex MSO8. This implies that networks exhibiting very low synchronization errors are highly likely to produce similarly low test errors. An approximate equality between the test error and the error computed on the synchronization sequence indicates high generalization quality of the obtained model, with a minimal risk of overfitting. This validates the synchronization error as a reliable indicator of synchronization progress for the considered sequences. This is especially beneficial for evaluating performance on non-stationary time series, where the test error cannot be computed because of the regularly changing characteristics of $\mathbf{y}(t)$.

The complexity of the time series significantly influences the correlation between synchronization and test errors – increased complexity diminishes this correlation. This

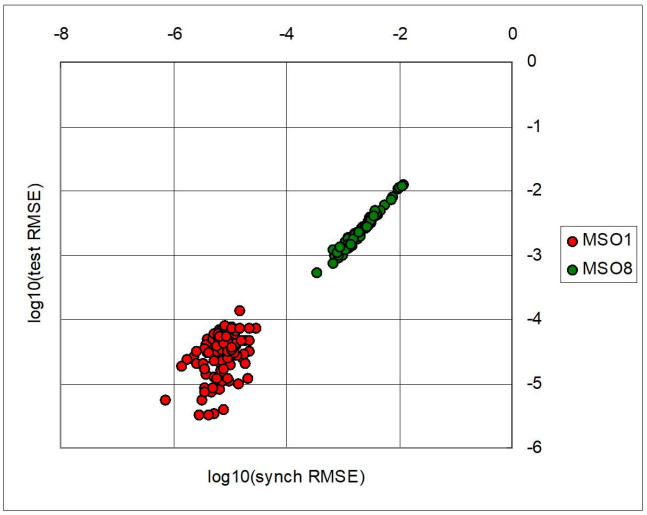

Figure 5.1.: Correlation between synchronization and test errors for the simple MSO1 (red circles) and the complex MSO8 (green circles) sequences. The strong correlation observed indicates a low risk of overfitting.

is attributed to the synchronization algorithm's worse convergence on more intricate dynamics, such as MSO8 (refer to Figure 5.1), which elevates the risk of premature convergence when the algorithm fails to find the global attractor in every sequence run. However, accurate identification of the time series content does not necessarily require locating the global optimum; a satisfactory local optimum can suffice. Here, a satisfactory optimum implies that the synchronized ensemble achieves acceptable modeling accuracy, albeit not the highest. The proportion of sequence runs that yield correct identification is denoted by the Identification Rate (IR), which emphasizes the accuracy of time series decomposition rather than modeling accuracy. This indicator estimates the probability of the algorithm converging with correct decomposition of the given time series. In the experiments, the IR was calculated as follows.

$$IR = \frac{1}{R}\sum_{r=1}^{R}\begin{cases}1 & if\ \mathbf{Y}_r = \mathbf{O}_r \\ 0 & otherwise\end{cases} \tag{5.2}$$

where R is the number of sequence runs, $\mathbf{Y}_r$ is the set of dynamic components within the given time series $\mathbf{y}(t)$, and $\mathbf{O}_r$ is the resulting set of dynamic modules in the synchronized ensemble. The equality of sets $\mathbf{Y}_r$ and $\mathbf{O}_r$ implies that $\mathbf{O}_r$ comprises only relevant modules, whose non-zero output activity uniquely identifies the corresponding components from $\mathbf{Y}_r$. Thereat, the synchronization error must be below a predefined error threshold, which needs to be individually set for each time series based on its complexity – larger thresholds are required for more complex target dynamics. In all experiments, the performance was averaged over 100 independent runs.

Beyond modeling accuracy and the correctness of time series decomposition, which serve as general performance indicators independent of module type, the rigid mathematical structure of parametric modules also allows the evaluation of synchronization performance through the accuracy of determined parameter values.

5.2. Recurrent modules for continuous oscillatory time series

This section describes the methodology for designing the recurrent neural modules for the presented experiments. Additional investigations were conducted to align with the synchronization method's requirements. The primary challenge was to achieve an optimal balance between high modeling accuracy and module size. A straightforward approach involving an increase in ESN reservoir size for improved accuracy inevitably extends the length of individuals, resulting in prohibitively long execution times. Conversely, a direct reduction in reservoir size severely compromises module accuracy, preventing the generation of high-performance individuals within the population. Consequently, the performance of the best individuals remains comparable to the population's average, hindering the search for the global optimum and disrupting algorithm convergence. However, further analysis of ESN behavior revealed that a significant reduction in ESN size, without sacrificing module accuracy, could be achieved, offering a favorable trade-off.

5.2.1. Modeling the sine waves by ESN

An earlier study [Sch+07] asserted that linear combinations of sine signals are challenging to reproduce using monolithic ESNs, stating: *"Echo State Networks are unable to learn functions composed of even two superimposed oscillators, in particular sin(0.2x) + sin(0.311x). The reason for this is that the dynamics of all the neurons in the ESN "pool" are coupled, while this task requires that the two underlying oscillators be represented by the network's internal state."* (p. 19). However, a subsequent study [KLB12] demonstrated the feasibility of modeling sine waves with monolithic ESNs when the unfolding dynamics are balanced with the driving signal at the dynamic reservoir's neurons. This was achieved through a significant reduction of the initialization range of the output feedback weights $\mathbf{W}_{OFB}$. These balanced ESNs yielded exceptionally high accuracy for MSO time series of different complexity. This also facilitated a reduction in ESN size (see Figures 9 and 10 in [KLB12]). Nonetheless, increasing complexity of time series led to a deterioration in test error. As shown in Table 5 in [KLB12], the Normalized Root Mean Squared Error (NRMSE) error degraded by eight orders of magnitude for the more complex MSO8 time series compared to the simpler MSO2. This degradation is attributed to the non-optimality of large, randomly generated reservoirs, related to their weights $\mathbf{W}_{RES}$ and topology, which may contain superfluous neurons that are active as noise and disrupt the unfolding dynamics in the reservoir. Optimization of dynamic reservoirs [Ott+16] resulted in substantially lower errors. It also revealed that increasing time series complexity had

a considerably weaker impact on the modeling accuracy of optimized ESNs than on randomly generated balanced ESNs in [KLB12].

In this thesis, the synchronization method aims to decompose the given time series into a set of basic dynamic components, specifically individual sine waves for MSO time series. Each sine wave was accurately modeled using a compact ESN module with only three reservoir neurons. Given that optimizing such small ESNs does not yield significant improvements over randomly generated reservoirs, these were utilized without further optimization. Table 5.1 presents the test errors of the ESNs employed as recurrent modules in the experiments. As indicated in the table, the small ESNs reproduce the lower frequencies, 0.2 and 0.311, with comparatively larger errors than those observed for other frequencies. The limited dynamic memory of these compact reservoirs is inadequate for memorizing longer periods, resulting in sine wave reproduction with increased distortion.

Table 5.1.: Modeling accuracy of ESN modules on test sequences of the sinusoidal signal. The displayed errors were obtained by running each corresponding module on a 300-time-step test sequence, following training on a 300-time-step training sequence.

Signal	Test RMSE
sin(0.2)	1.81×10^{-6}
sin(0.311)	5.35×10^{-7}
sin(0.42)	3.95×10^{-8}
sin(0.51)	2.68×10^{-8}
sin(0.63)	4.38×10^{-9}
sin(0.74)	8.86×10^{-8}
sin(0.85)	9.35×10^{-8}
sin(0.97)	4.69×10^{-8}

The utilized ESN modules were selected from a set of 100 randomly generated ESNs, adhering to recommendations from [KLB12], and then trained using Eq. (A.4). The selection criterion was the minimal test error on the corresponding sine signal sequence. To prepare a module for each sine wave, the sequence was divided into washout, training, and test intervals. The sequence length was 700 time steps. The first 100 time steps were used for washing out the random states of the randomly generated ESN. Training was performed on the subsequent 300 time steps. The final 300 time steps were used to calculate the neural network's test error. Modules had no input neurons, with their internal dynamics maintained solely by OFB signals. All reservoir neurons employed TANH activation. The output neuron used an identity activation function and possessed no additional non-linearity.

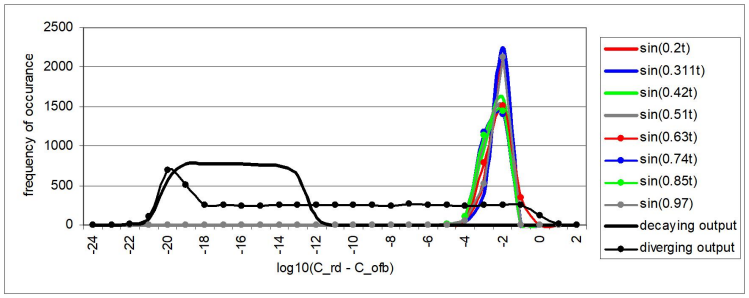

Figure 5.2.: Distribution of the difference between the contributions from output-feedback (C_{OFB}) and from reservoir dynamics (C_{RD}) at reservoir neurons for sine wave sequences with frequencies ranging from 0.2 to 0.97. The distributions of stable ESN modules are represented by colored lines, while those of unstable ESN modules are shown with black lines.

5.2.2. Balanced dynamics in ESN modules

Given that reservoir neurons must operate within the linear region of the TANH activation function to accurately model sine waves, the prepared ESN modules were analyzed for the balance between the contributions from output feedback (C_{OFB}) and from reservoir dynamics (C_{RD}). As per [KLB12], achieving balance requires similarity between these contributions across all reservoir neurons. To verify the balancing condition (7) from [KLB12] for the ESN modules, Figure 5.2 compares the distributions of $(C_{RD} - C_{OFB})$ values in deployed stable and precise ESNs with examples of poorly performing ESNs that exhibited large test errors. As illustrated in the figure, effective ESN modules maintain a minimal difference between these contributions, limiting the distribution's spread to four orders of magnitude. On the contrary, poorly performing ESNs demonstrate imbalance, characterized by a substantial difference between C_{OFB} and C_{RD} contributions and a wide distribution spread. According to [KLB12], prolonged dominance of neuron states by either C_{OFB} or C_{RD} negatively impacts neural oscillator performance. Dominance of C_{OFB} diminishes the influence of reservoir dynamics on the network output and restricts the reservoir's short-term memory. Conversely, dominance of reservoir dynamics over the OFB signal can induce instability in dynamic reservoirs, leading to abrupt increases in module output signals.

5.2.3. Modeling the triangular signals by ESN

Triangular signals present a particular challenge for modeling with compact ESNs, primarily because of the corner points at minima and maxima, where the linear signal flow abruptly changes direction. To comprehend the properties of ESN modules for modeling triangular signals, it is beneficial to consider ESN modules for sinusoidal signals as a reference. As demonstrated in [Jae02], the unfolding temporal patterns of reservoir dynamics must exhibit similarity to the temporal patterns of the modeled

signal. Consequently, the smooth signal flow of sine waves is accurately modeled when the output of reservoir neurons also changes smoothly within the linear region of the TANH activation. This suggests that ESN reservoirs must maintain internal dynamics that exhibit a form of periodic switching to reproduce triangular oscillatory signals. In contrast to ESN modules for sinusoidal oscillations, this indicates the necessity of employing reservoir neurons that operate in saturation. The output of such neurons can transition rapidly between extreme values close to "-1" and "+1," which is essential for realizing the sharp change in signal direction at corner points.

The ESN's capacity to maintain periodically switching dynamics was investigated to design compact modules for modeling the triangular signals. Experimental data confirmed that the highest-performing ESNs were capable of producing triangular signals with high accuracy over extended periods (see Table 5.3). Analysis of the dynamic reservoirs highlighted the utility of saturated neurons in sustaining switching internal dynamics. Figure 5.3 exemplifies the switching internal dynamics within an ESN reservoir containing saturated neurons. This analysis also revealed a functional division of reservoir neurons into three following groups.

- *Neurons operating purely within linearity of TANH*, e.g. Neurons 0 and 1 in Figure 5.3.
- *Neurons operating within moderate non-linearity of TANH*, e.g. Neuron 4 in Figure 5.3.
- *Neurons operating in saturation*, e.g. Neurons 2, 3, and 5 in Figure 5.3.

Neurons in the first group are better suited for modeling smoothly changing signals, whereas those in the third group are more suitable for modeling abruptly changing dynamics. Considering the modeled triangular signal as a combination of corner points and linearly changing segments, the ratio between the neuron counts in these groups varied according to the triangular oscillatory signal's period. As shown in Table 5.2, a longer period of 32 time steps necessitated a higher switching demand – the corresponding ESN reservoirs were larger, with more neurons operating in saturation. In contrast, the shortest periods of eight and four time steps were modeled as periodic sequences of discrete values, computed with repeating combinations of sufficiently large values within the moderate non-linearity range of TANH. Similar to sine waves, triangular oscillatory signals with shorter periods are inherently less challenging to model than long-period oscillations. As evidenced by comparing test errors for both signal types in Table 5.1 and Table 5.3, modeling accuracy for shorter periods was consistently higher.

5.3. Decomposition of stationary time series

Experimental data for the performance evaluations were obtained by running the synchronization algorithm on a variety of stationary oscillatory time series. While the algorithm itself lacked knowledge of the time series' true contents, this information enabled automated performance evaluation, eliminating the need for manual inspection of synchronized modules after each sequence processing.

Table 5.2.: Number of reservoir neurons used for modeling triangular oscillatory signals. Neurons are classified into three groups based on their operating range within the TANH activation function: linear, moderate non-linearity, and saturation. Oscillatory signals with longer periods require larger reservoirs with an increased number of neurons operating in saturation.

Period of triangular signal	Number of neurons on linear interval of TANH	Number of neurons with moderate non-linearity	Number of neurons that come into saturation	Total number of reservoir neurons
32	2	0	4	6
20	2	0	2	4
16	2	0	2	4
12	2	0	2	4
8	0	2	2	4
4	0	4	0	4

Table 5.3.: Modeling accuracy of ESN modules on test sequences of triangular oscillatory signals. The displayed errors were obtained by running each corresponding ESN module on a 300-time-step test sequence, following training on a 300-time-step training sequence.

Period of triangular signal	Test RMSE
32	7.87×10^{-7}
20	3.60×10^{-10}
16	1.30×10^{-15}
12	5.94×10^{-16}
8	3.75×10^{-16}
4	1.56×10^{-16}

The modeling accuracy, quantified by the average test RMSE and its standard deviation, along with the IR, for both gradient-based and evolutionary synchronization methods, are presented in Table 5.4 and Table 5.5 for balanced time series, and in Table 5.6 and Table 5.7 for imbalanced time series. Modeling mixtures of rectangular signals with compact ESNs proved infeasible, thereby precluding the use of neural modules. Additionally, given that the parametric modules of the triangular and rectangular signals exhibit non-differentiability, performance data are not obtainable for the respective parametric modules.

The presented performance results are followed by an analysis, which contrasts the behavior of gradient-based and evolutionary approaches (subsection 5.3.1 Gradient-based vs. evolutionary) and parametric and neural modules (subsection 5.3.2 Parametric vs. neural modules).

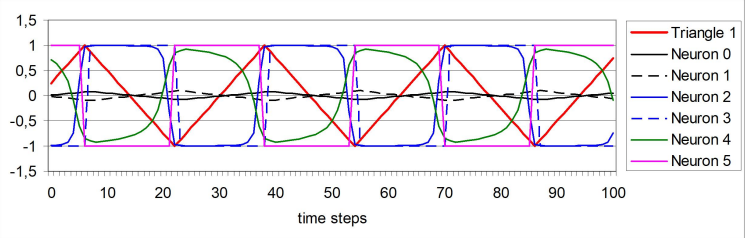

Figure 5.3.: The output of the ESN module (red line) reproducing a triangular oscillatory signal. The corner points of the target signal were generated by combining the minima and maxima of reservoir states from Neurons 0 and 1, along with simultaneous sharp transitions in the dynamics of saturated Neurons 2, 3, and 5.

5.3.1. Gradient-based vs. evolutionary

The experimental data presented in Tables 5.4 to 5.7 allowed a comparison between gradient-based and evolutionary synchronization, focusing on final synchronization accuracy and convergence behavior on compositions of smooth sinusoidal and non-smooth triangular oscillations, as illustrated in Figure 5.4. As seen in this figure, the error curves of evolutionary synchronization exhibit a non-monotonic trend because of the stochastic nature of the approach. Instead, the curves show fluctuations, with error variations of up to an order of magnitude observed in the MSO8 data. The sudden error increases indicate temporary convergence of the population to local optima, which were subsequently overcome by the evolutionary process, leading to significantly improved final synchronization accuracy. Notably, evolutionary synchronization achieved up to four orders of magnitude lower final errors compared to the gradient-based method on the same time series, as evidenced by a comparison of Tables 5.4 and 5.5. The only time series where the gradient-based method performed marginally better were on imbalanced MSO sequences, where the ensemble of sinusoidal oscillators created a strong attractor with a few local optima.

Overall, the experiments revealed limitations of the gradient-based method. Firstly, as demonstrated in Tables 5.4 and 5.6, the method is largely inapplicable to parametric modules of non-differentiable signals. Besides, evolutionary synchronization consistently outperformed gradient-based synchronization on more complex time series with a large number of dynamic components. The increased complexity of these time series leads to more complex objective functions with a greater number of local optima, posing significant challenges for gradient-based optimization. The gradient-based methods' tendency for rapid convergence towards local optima becomes a pitfall when the search is trapped in a poor local optimum, leading to premature convergence. To illustrate the rapid stagnation of the gradient-based method, 5.5 presents the distributions of the time steps at which the last significant improvement in synchronization accuracy was observed on the Triangle3 and Triangle4 time series. These time series were selected for

Table 5.4.: Performance of gradient-based synchronization on balanced compositional time series using ensembles of neural and parametric modules. The symbol "-" indicates no convergence across all runs. "n/a" denotes cases where the method was not applicable because of the unavailability of either modules or their gradients for the oscillatory signals.

	Time series	Average test RMSE	Standard deviation	IR (%)
Neural	**MSO1**	1.85×10^{-6}	1.04×10^{-6}	100
	MSO2	2.00×10^{-5}	1.61×10^{-5}	100
	MSO4	5.79×10^{-5}	3.08×10^{-5}	100
	MSO6	1.91×10^{-4}	1.16×10^{-4}	100
	MSO8	2.71×10^{-4}	1.38×10^{-4}	100
	Triangle1	1.35×10^{-2}	8.82×10^{-3}	95
	Triangle2	1.20×10^{-2}	8.11×10^{-3}	41
	Triangle3	-	-	-
	MST11	4.29×10^{-2}	4.41×10^{-2}	80
	MST22	4.84×10^{-2}	5.94×10^{-2}	7
	MST33	-	-	-
	Rectangles	n/a	n/a	n/a
Parametric	**MSO1**	3.84×10^{-2}	2.63×10^{-2}	100
	MSO2	2.91×10^{-2}	1.94×10^{-2}	100
	MSO4	4.13×10^{-2}	2.85×10^{-2}	100
	MSO6	3.59×10^{-2}	2.19×10^{-2}	100
	MSO8	3.70×10^{-2}	2.62×10^{-2}	93
	Triangle1 / MST11	n/a	n/a	n/a
	Triangle2 / MST22	n/a	n/a	n/a
	Triangle3 / MST33	n/a	n/a	n/a
	Rectangles	n/a	n/a	n/a

comparison because the gradient-based approach failed on them. As shown in the figure, the gradient-based method frequently became trapped in local optima at much earlier time steps (approximately time step 90) compared to the evolutionary method, which continued to significantly improve modeling accuracy until around time step 210. This extended improvement period afforded the evolutionary method a higher probability to correctly identify the hidden dynamic components with greater precision.

Attractor characteristics and convergence

Figure 5.6, Figure 5.7, Figure 5.8, and Figure 5.9 demonstrate convergence of the population to the global optimum of the dynamic states. Specifically, convergence is observed on the imbalanced MSO8 time series, where both evolutionary and gradient-based methods converged, and on the Triangle3 time series, where convergence was achieved only by the evolutionary method. A comparative analysis of the population's behavior on the Triangle3 time series elucidates why synchronization was successful for both methods with sinusoidal oscillations and reveals the reasons for the gradient-based

Table 5.5.: Performance of evolutionary synchronization with neural and parametric modules on balanced time series. The Triangle and MST data complete the MSOs previously presented in [KOB21]. The equal amplitudes of the rectangular signals render the problem ill-posed, making unique decomposition of the Rectangle sequences impossible; therefore, this data is unavailable.

	Time series	Average test RMSE	Standard deviation	IR (%)
Neural	**MSO1**	1.62×10^{-6}	3.03×10^{-6}	100
	MSO2	1.38×10^{-6}	4.06×10^{-8}	100
	MSO4	1.43×10^{-6}	3.51×10^{-8}	100
	MSO6	1.83×10^{-6}	6.21×10^{-8}	100
	MSO8	1.86×10^{-6}	6.82×10^{-8}	100
	Triangle1	4.51×10^{-7}	2.02×10^{-9}	100
	Triangle2	5.36×10^{-7}	1.15×10^{-7}	100
	Triangle3	5.24×10^{-7}	1.03×10^{-7}	100
	Triangle4	5.24×10^{-7}	1.80×10^{-7}	100
	Triangle5	0.627	7.60×10^{-2}	97
	Triangle6	0.849	0.157	100
	MST11	1.32×10^{-5}	1.88×10^{-5}	88
	MST22	6.06×10^{-4}	7.87×10^{-4}	77
	MST33	7.55×10^{-4}	6.89×10^{-4}	18
	MST44	1.55×10^{-3}	1.69×10^{-3}	58
Parametric	**MSO1**	3.97×10^{-3}	3.58×10^{-3}	100
	MSO2	8.38×10^{-3}	7.22×10^{-3}	100
	MSO4	3.78×10^{-2}	2.45×10^{-2}	100
	MSO6	7.65×10^{-2}	4.38×10^{-2}	92
	MSO8	9.88×10^{-2}	4.16×10^{-2}	52
	Triangle1	1.03×10^{-2}	6.07×10^{-3}	100
	Triangle2	9.15×10^{-3}	6.87×10^{-3}	100
	Triangle3	8.40×10^{-3}	4.98×10^{-3}	100
	Triangle4	1.20×10^{-2}	5.98×10^{-3}	100
	Triangle5	2.57×10^{-2}	9.98×10^{-3}	100
	Triangle6	6.14×10^{-2}	2.05×10^{-2}	100

method's failure with triangular oscillatory signals. In each figure, the population of the first neural oscillator module is projected onto the plane defined by the module states x_1 and x_2, which exhibited the highest average variance throughout the synchronization process. The progressive reduction in the spread of the module states within the population is evident across the snapshots, from initialization to the advanced state at time step 200.

Due to the dynamic nature of the recurrent neural modules, their global optimum was moving along its trajectory over time. These optima and trajectories are depicted in each figure. As observed for the MSO time series, the neural oscillator possessed an inherent attractor with a family of alternative trajectories for stable evolvement of its dynamics. These trajectories are visualized as a series of nested concentric ellipses (exemplified by

Table 5.6.: Performance of gradient-based synchronization on imbalanced compositional time series using ensembles of neural and parametric modules. The symbol "-" indicates no convergence across all runs. "n/a" denotes cases where the method was not applicable because of the unavailability of either modules or their gradients for the oscillatory signals.

	Time series	Average test RMSE	Standard deviation	IR (%)
Neural	**MSO1**	4.05×10^{-6}	6.94×10^{-6}	100
	MSO2	4.01×10^{-5}	5.26×10^{-5}	100
	MSO4	8.56×10^{-5}	4.91×10^{-5}	100
	MSO6	1.95×10^{-4}	1.62×10^{-4}	100
	MSO8	2.92×10^{-4}	1.47×10^{-4}	100
	Triangle1	6.60×10^{-3}	5.84×10^{-3}	97
	Triangle2	2.12×10^{-2}	1.53×10^{-2}	53
	Triangle3	-	-	-
	Rectangles	n/a	n/a	n/a
Parametric	**MSO1**	9.67×10^{-3}	1.21×10^{-2}	100
	MSO2	1.80×10^{-2}	2.24×10^{-2}	100
	MSO4	5.87×10^{-2}	9.52×10^{-2}	98
	MSO6	8.81×10^{-2}	0.123	94
	MSO8	0.104	0.143	90
	Triangle1 / MST11	n/a	n/a	n/a
	Triangle2 / MST22	n/a	n/a	n/a
	Triangle3 / MST33	n/a	n/a	n/a
	Rectanges	n/a	n/a	n/a

two trajectories in Figure 5.6 and Figure 5.7). While remaining on any of these trajectories, the module maintained high modeling accuracy. Given that different module states yielded similarly high accuracy at the end of synchronization, the population did not converge to a single point but exhibited a relatively elongated, narrow spread, radially oriented towards the center of the coordinate plane (see the snapshots at time step 200 in Figure 5.6 and Figure 5.7). Further evolvement of any module state resulted in an elliptic trajectory concentric to the two shown exemplary trajectories.

In contrast, on the Triangle time series, the dynamic module lacked an inherent attractor and possessed a single globally optimal trajectory for accurate reproduction of the time series data. This trajectory is depicted as a single green dotted line in Figure 5.8 and Figure 5.9, featuring two distinct points at the lower-left and upper-right corners where the dynamics reverses direction. These points are characteristic for the trajectory and correspond to the maximum and minimum magnitudes of the triangular oscillatory signal, where it sharply changes the direction of its propagation. As shown in Figure 5.8, the population remained largely dispersed across the search space for an extended period (until time step 50) during evolutionary synchronization, while the gradient-based method rapidly identified the optimal trajectory by time step 10. However, comparing the population's location relative to the global optimum at later time steps

Table 5.7.: Performance of evolutionary synchronization with ensembles of neural and parametric modules on imbalanced compositional time series. The Triangle and Rectangle data complete the MSOs previously presented in [KOB21].

	Time series	Average test RMSE	Standard deviation	IR (%)
Neural	**MSO1**	4.55×10^{-5}	3.43×10^{-5}	100
	MSO2	1.93×10^{-4}	1.43×10^{-4}	100
	MSO4	2.68×10^{-4}	1.78×10^{-4}	100
	MSO6	5.47×10^{-4}	2.67×10^{-4}	100
	MSO8	6.06×10^{-4}	2.20×10^{-4}	100
	Triangle1	2.30×10^{-6}	3.43×10^{-6}	99
	Triangle2	4.36×10^{-5}	3.99×10^{-5}	97
	Triangle3	3.90×10^{-4}	1.80×10^{-3}	97
	Triangle4	2.09×10^{-4}	1.20×10^{-4}	91
	Triangle5	0.330	0.204	66
	Triangle6	0.514	0.256	62
	Rectangles	n/a	n/a	n/a
Parametric	**MSO1**	3.39×10^{-3}	2.68×10^{-3}	100
	MSO2	1.38×10^{-2}	1.16×10^{-2}	91
	MSO4	6.18×10^{-2}	0.110	46
	MSO6	0.523	0.358	15
	MSO8	0.217	5.95×10^{-2}	6
	Triangle1	3.18×10^{-3}	2.93×10^{-3}	99
	Triangle2	6.14×10^{-3}	5.14×10^{-3}	97
	Triangle3	5.43×10^{-3}	3.30×10^{-3}	95
	Triangle4	1.20×10^{-2}	1.93×10^{-2}	86
	Triangle5	2.60×10^{-2}	4.40×10^{-2}	73
	Triangle6	8.11×10^{-2}	6.89×10^{-2}	21
	Rectangle1	5.24×10^{-3}	8.96×10^{-3}	100
	Rectangle2	1.26×10^{-2}	1.30×10^{-2}	100
	Rectangle3	1.64×10^{-2}	1.11×10^{-2}	100
	Rectangle4	4.58×10^{-2}	2.04×10^{-2}	31
	Rectangle5	4.60×10^{-2}	1.05×10^{-2}	8
	Rectangle6	6.11×10^{-2}	1.87×10^{-2}	7

revealed weaknesses in the gradient method; it experienced premature convergence, with the entire population becoming trapped in various local optima along the trajectory, preventing further tracking of the global optimum. In Figure 5.9, the population is distributed along the globally optimal trajectory without any change after time step 50. This observation aligns well with the data presented in Figure 5.5 for the gradient method. Conversely, the evolutionary method successfully overcame the local optima and tracked the global optimum more and more closely, evidenced by the decreasing spread of the population at the global optimum and near-point convergence by time step 200 (see Figure 5.8).

Therefore, the presence of an inherent attractor with a family of alternative trajectories facilitates synchronization, leading to a dynamic optimization problem of reduced

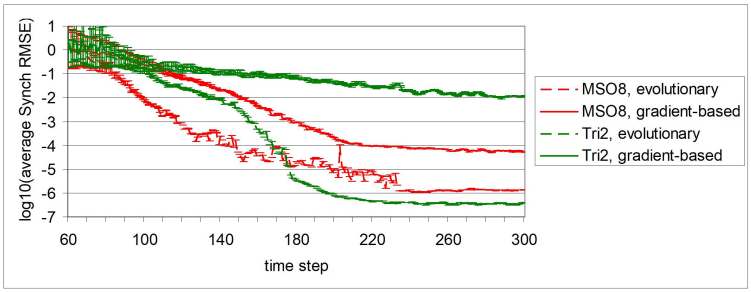

Figure 5.4.: Average error curves for gradient-based and evolutionary synchronization on MSO8 (red lines) and Triangle2 (green lines). Triangle2 was selected for comparison as the most complex triangle time series on which gradient-based synchronization converged.

complexity that can be successfully addressed using the methods of global as well as local optimization. Conversely, the increased compound dimensionality of the problem, coupled with the absence of an attractor, can be effectively managed only by the methods of global optimization.

Computational effort

Leveraging gradient information, gradient-based methods typically require less computational effort than their evolutionary counterparts to find solutions. Consequently, gradient-based synchronization was also faster than evolutionary synchronization; in the worst case, the average sequence processing time was halved, as shown in Table 5.8. The most significant time advantage, exceeding 12 times (!), for the gradient-based method was observed on the least complex imbalanced MSO1 sequences. However, the moderate time advantage observed on the balanced time series was unexpected and likely resulted from search oscillation around the global optimum with a target responsibility of "1.0", which is located at the boundary of the feasible region defined by (4.47).

Table 5.8.: Average processing time in seconds per sequence for evolutionary and gradient-based synchronization on the least and most complex time series considered, MSO1 and MSO8 respectively. Using ensembles of eight neural modules and a population of size 100, the experiments were performed on a PC with an Intel Core i7-9750H processor and 16GB RAM.

Time series	Sequence type	Evolutionary	Gradient-based
MSO1	balanced	105.21	46.80
	imbalanced	106.66	8.40
MSO8	balanced	722.21	314.40
	imbalanced	400.18	64.80

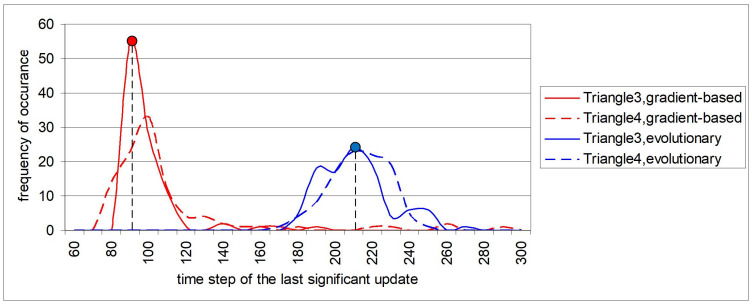

Figure 5.5.: Distributions of the time steps at which the gradient-based (red lines) and evolutionary (blue lines) synchronization methods achieved their last significant (five-fold) error improvement during synchronization on Triangle3 (solid lines) and Triangle4 (dashed lines). A tendency for premature convergence was observed in the gradient method, as indicated by the substantially earlier peak of the red distribution relative to the blue distribution of the evolutionary method.

In an attempt to reduce computational effort, one might consider decreasing the population size as a potential solution. However, two risks must be kept in mind. First, there is the risk of premature convergence. With a small population, initialization distributes only a few individuals across a limited number of attractor basins. Given that gradient methods have limited capacity to escape local attractors, the search will likely converge to one of these local optima. Second, smaller populations reduce diversity for potential pairings between individuals of different populations, thereby restricting permutation capabilities.

Figure 5.10 and Figure 5.11 demonstrate the advantage of using larger populations, which ensure both higher modeling accuracy and identification rates for both synchronization methods, regardless of the time series. However, the type of sequence data significantly influences the extent to which the population size can be reduced without substantial performance loss. As observed, the MSO time series allows the population of the evolutionary method to be reduced from an initial 100 module states to approximately 30 while maintaining a 100% identification rate. Thereat, the reduced population size affected the final accuracy, which largely plateaued above a population size of 80. Interestingly, the performance, both IR and error, of the gradient method remained independent of population size. This suggests a significant potential for accelerating the gradient-based method beyond what is shown in Table 5.8. In contrast, the identification performance on the Triangle time series shows a stronger, nearly linear dependence on population size. Along with this, the average test error on converged runs remained relatively constant; that is, regardless of population size, the algorithm converged to the same optimum, which was a local one. This optimum was significantly inferior to the one found during evolutionary synchronization, which was likely the global optimum for population sizes above 50 (see Figure 5.10). This also implies significant potential for

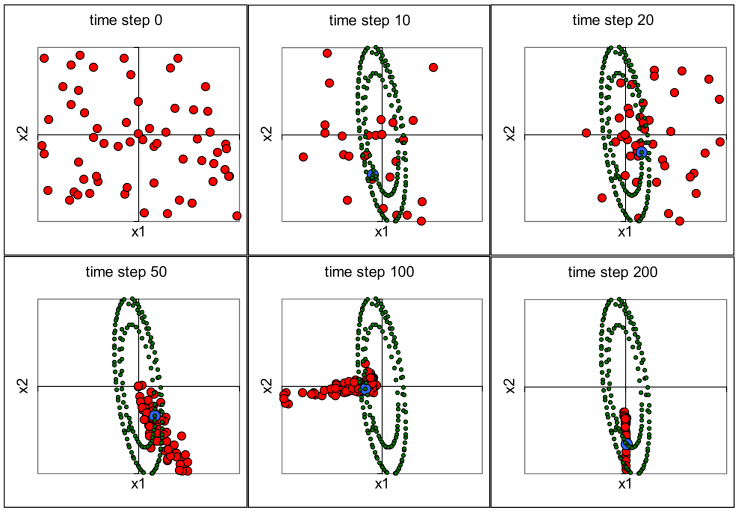

Figure 5.6.: Evolution of the population during evolutionary synchronization of neural oscillator 0 on the imbalanced MSO8. The red circles represent the projection of the population onto the plane defined by the states x_1 and x_2, which exhibited the highest variance across the population. The green ellipses are the alternative trajectories of the global optimum (blue circle).

accelerating evolutionary synchronization.

Despite the increased computational effort, a wider sliding window would provide more local information about the objective function and could mitigate the issue of local optima. But such an approach is limited by vanishing gradients, a phenomenon reported in [Hoc+01]. This phenomenon suggests that earlier samples within the sliding window have negligible impact on the computed gradient value when the window exceeds a certain size. Along with this, reducing the sliding window would make gradient-based tuning more susceptible to local optima while decreasing computational effort. Therefore, careful consideration is required when selecting the sliding window size. Given the dependencies observed in Figure 5.10 and Figure 5.11, population size should be regarded as the key parameter, directly influencing the expected computational effort, final accuracy, and identification performance of the synchronization methods.

5.3.2. Parametric vs. neural modules

As shown in Table 5.4, Table 5.5, Table 5.6, and Table 5.7, the decomposition ability of the synchronization method is generally independent of the deployed module type. Specifically, the identification rates were similarly high for both module types, and both allowed tuning the modules to highly accurate models of the corresponding dynamic components. However, performance was slightly higher with neural recurrent modules that exhibit greater plasticity, resulting in a smoother objective function and,

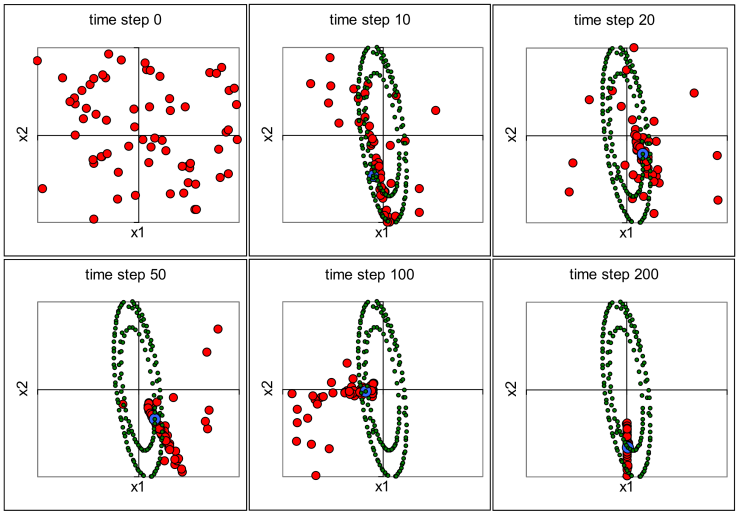

Figure 5.7.: Evolution of the population during gradient-based synchronization of neural oscillator module 0 on the imbalanced MSO8. The red circles represent the states in the population. The green ellipses are the alternative trajectories of the global optimum (blue circle).

consequently, improved accuracy and marginally higher identification rates. Additionally, performance was influenced by the characteristics of the mixed signals. The abruptly changing triangular and rectangular signals produce an unsmooth error signal, increasing the number of local optima in the objective function and thus the risk of premature convergence. Indeed, on mixtures of triangular and rectangular signals, accuracy was comparatively lower than on the MSO time series, confirming the particular difficulty of these signals for decomposition. Regardless of the type of mixed signals, increasing their number made the problem more challenging, reducing the likelihood of correct identification.

Applying synchronization to ensembles of parametric modules revealed additional advantages of network modularity for time series decomposition. Compared to the black-box nature of neural modules, the rigid mathematical expressions of parametric modules render the resulting modular network more amenable to detailed analysis of the modeled dynamics and the physical processes underlying the given sequence data. However, this analysis requires precise determination of the hidden true parameter values. As shown in Table Table 5.9, the achieved accuracy was high, with the determined frequency and phase values of the sinusoidal modules exhibiting only minor deviations from the corresponding hidden parameter values. The parameter accuracy of the triangular and rectangular modules was slightly lower, consistent with the larger test error of their models.

The observed high performance of synchronization with both considered module types, neural and parametric, confirmed the method's independence from the module type used

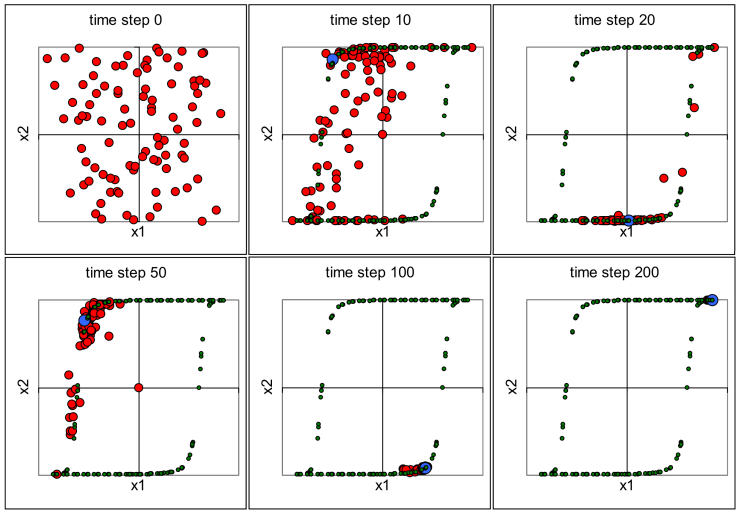

Figure 5.8.: Evolution of the population during evolutionary synchronization of neural oscillator module 0 on the imbalanced Triangle3 time series. The red circles represent the states in the population, and the trajectory of the global optimum (blue circle) is indicated by the green dots.

Table 5.9.: Accuracy of the determined parameter values in mixtures of sinusoidal oscillations.

Time series	Average deviation of frequency	Average deviation of phase
MSO1	2.70×10^{-5}	7.19×10^{-3}
MSO2	3.62×10^{-5}	8.24×10^{-3}
MSO4	1.09×10^{-4}	2.69×10^{-2}
MSO6	1.66×10^{-4}	3.99×10^{-2}
MSO8	2.02×10^{-4}	4.89×10^{-2}

and provided an opportunity to extend the declared general advantages of modularity with those offered by a specific module type.

The use of sinusoidal parametric modules raises a reasonable question regarding the synchronization method's performance compared to FFT, which also employs sinusoidal oscillations as elementary signals for sequence decomposition. To address this, we must examine the exemplary frequency spectra of the MSO4 and MSO8 time series in Figure 5.12, obtained via FFT and published in our earlier work [KOB21]. As observed, the resolution of the frequency spectra is highly dependent on the chosen window length. As stated in [KOB21]: *"But even with up to 300 samples (which cover the full amount of data that ESMoN requires for full convergence with a 60-time-step sliding window), the peaks do neither exactly align with the true frequencies nor do they allow deducing the correct amplitudes of the involved basic oscillations directly."* (p.20) This confirms the synchronization method

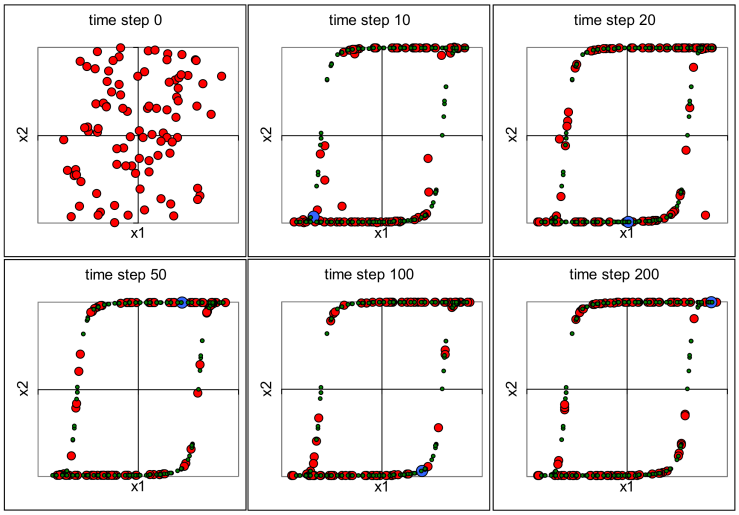

Figure 5.9.: Evolution of the population during gradient-based synchronization of neural oscillator module 0 on the imbalanced Triangle3 time series. The red circles represent the states in the population, and the trajectory of the global optimum (blue circle) is indicated by the green dots.

as a more favorable option for component identification on short sequences of a few dozen time steps, although the high computational cost of synchronization must be taken into consideration.

5.3.3. Decomposition of the chaotic attractor dynamics

To further evaluate the capabilities of the synchronization approach, I applied it to decompose the chaotic attractor dynamics of the MG time series. Given that this time series lacks a generally decomposable underlying dynamics, most studies, such as [JH04; GM09], model the MG time series using very large monolithic recurrent networks, typically around 1000 neurons, to achieve accurate predictions up to 84 time steps ahead. Modular neural networks were rarely used to model this time series. For instance, [Kha06] presented an evolutionary approach that relied solely on feed-forward RBF modules. To account for the dynamic nature of this time series, a two-level modular network topology was augmented with feedback connections, providing input neurons with time-delayed values from the sequence. The resulting modular network was relatively compact and exhibited weak recurrence, potentially explaining its limited prediction horizon of only 3 time steps ahead. Additionally, modularity was explored with the MG time series in [Sol02; MC13].

In the experimental setup, the set of available module comprised ten sinusoidal and ten triangular oscillatory signals. The frequency range of the sinusoidal modules was constrained to the interval [0.1, 1.0]. The time range of the triangular parametric modules

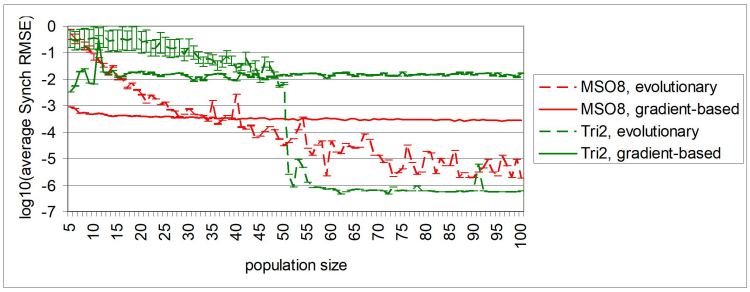

Figure 5.10.: Dependency of achieved modeling accuracy on population size for evolutionary (dashed lines) and gradient-based (solid lines) synchronization for compositions of sinusoidal (red lines) and triangular (green lines) oscillatory signals.

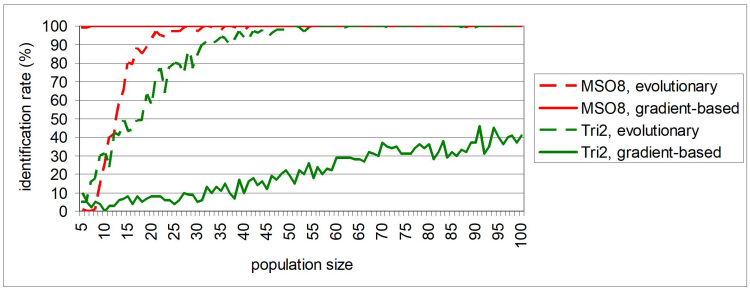

Figure 5.11.: Dependency of the identification rate on population size for evolutionary (dashed lines) and gradient-based (solid lines) synchronization for compositions of sinusoidal (red lines) and triangular (green lines) oscillatory signals.

was limited to the interval [2, 38] time steps. The MG sequences were generated with parameters $\beta = 0.1$, an exponent of 10, and $\tau = 17$. Each sequence was generated with an own value of the parameter α, random value uniformly distributed within the interval [0.05, 0.5]. As shown in Table Table 5.10, the algorithm successfully reconstructed some of the given MG sequences as a linear combination of the parametric modules within a limited time horizon of ten time steps. This is a noteworthy result, as chaotic attractors, unlike mixtures of oscillatory signals, are highly nonlinear and lack a known decomposable dynamic structure. Figure 5.13 illustrates exemplary MG sequences and their reconstruction using a linear combination of oscillator modules.

As indicated in the caption of Figure 5.13, the reconstruction of different MG sequences required different numbers of oscillator modules. In practice, the total number of identified oscillators could potentially serve as a complexity measure, characterizing the difficulty of modeling the chaotic attractors.

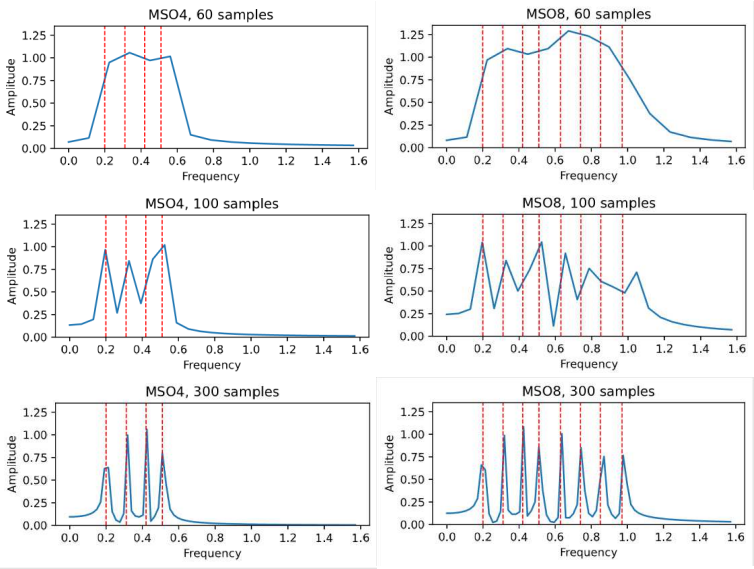

Figure 5.12.: Power spectra with several window lengths for balanced MSO4 (left column) and balanced MSO8 (right column). The dotted red vertical lines indicate the true frequencies within the respective time series. (This figure was previously published in our work [KOB21].)

5.3.4. Critical parameters of synchronization

The investigations revealed the parameters that have the greatest impact on the balance between decomposition quality and computational effort. These are: (i) population size, (ii) the number of individual updates, and (iii) the sliding window size. Larger populations enable a more comprehensive evaluation of the objective function, exploring a wider range of module state combinations across the search space. Given that the search space can be divided into attraction basins of different optima, a random initialization of more individuals increases the likelihood of hitting the attraction basins of a greater number of optima. This enhances the probability of converging to a superior optimum, as demonstrated in Figure 5.10 and Figure 5.11, where larger population sizes improve modeling accuracy. Conversely, larger populations place a greater demand on computational resources and should not be excessively large to ensure synchronization can be performed on the available hardware platform.

The next parameter is the maximum number of individual updates N^{max} (from Eq. (4.53)), which defines the intensity of module tuning at the current time step. As previously described, this parameter controls the expected synchronization accuracy – a higher number of updates is required for greater accuracy. However, excessively large values of N^{max} increase the risk of premature convergence because of the limited width of the sliding window, which provides only restricted information about the time series $\mathbf{y}(t)$. It is essential to avoid intensive tuning of module dynamics during the initial synchronization

Table 5.10.: Prediction accuracy of the linear combination of sinusoidal and triangular parametric modules on the MG time series. The median error is reported due to a significant number of non-converged outlier runs.

Horizon of prediction (in time steps)	Median test RMSE	Standard deviation
1	6.46×10^{-2}	0.135
2	8.47×10^{-2}	0.216
3	8.22×10^{-2}	0.288
4	8.15×10^{-2}	0.273
5	8.49×10^{-2}	0.298
6	9.09×10^{-2}	0.350
7	9.79×10^{-2}	0.333
8	9.43×10^{-2}	0.374
9	9.29×10^{-2}	0.413
10	9.24×10^{-2}	0.404

phase, as irrelevant modules have not yet been discarded. Their intensive tuning misleads the synchronization process, resulting in poor decomposition quality. This explains why premature convergence was observed less frequently with smaller values of N^{max} compared to larger ones.

In contrast, while large values of the parameter N^{max} are not prohibited, they are necessary for tuning larger modules. This increases the risk of premature convergence during synchronization of dynamic modules lacking built-in attractors, such as those used for compositions of triangular oscillatory signals. To mitigate this, the number of individual updates is gradually increased during synchronization, as defined by Eq. (4.53). A beneficial side effect is that this prevents the algorithm from unnecessarily consuming computational resources at the beginning of the sequence and distributes resource consumption more evenly across subsequent time steps. Table 5.11 compares the test errors obtained with constant and variable numbers of updates on compositions of imbalanced triangular oscillatory signals. As shown, maintaining a constant number of updates resulted in an approximate order-of-magnitude decrease in average accuracy. For compositions of sinusoidal oscillations, where neural modules exhibited strong attractors, the variable number of individual updates offered no advantage, as indicated in the same table.

For tuning module dynamics, the sliding window provides information about $\mathbf{y}(t)$, and each module state vector in the population is evaluated after its update. The data content of the sliding window directly influences the module performance, computed as the average error $e_{T_W}(t_c)$ in Eq. (4.5). Moving the window along $\mathbf{y}(t)$ alters the data content and, consequently, the resulting performance value. Tracking this performance over time yields the modeling error curve, whose smoothness is crucial for guiding synchronization towards the identification of the true, hidden dynamic components. Smoothness minimizes the risk of premature convergence and depends on the amount of

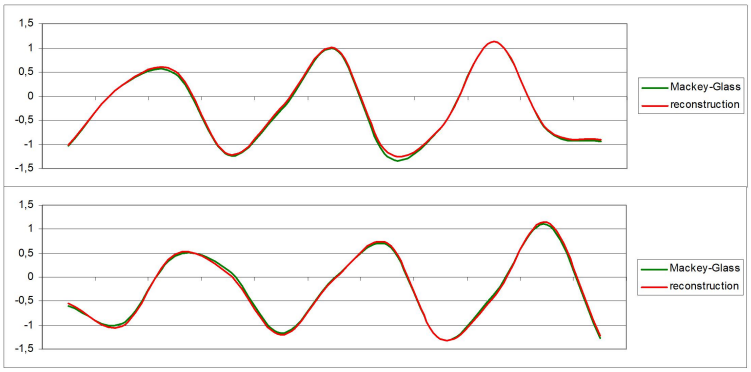

Figure 5.13.: MG sequences and their reconstruction as a linear combination of parametric modules. The upper sequence was reconstructed using a combination of ten sinusoidal and ten triangular oscillators. The lower sequence was reconstructed using a combination of three sinusoidal and four triangular oscillators.

Table 5.11.: Comparison of the average test RMSE after evolutionary synchronization using increasing and constant $N(t)$ for each module state in the population. The errors were obtained from an ensemble of neural recurrent modules on imbalanced time series with the highest number of sinusoidal and triangular recurrent signals, where convergence was observed for the most of totally 100 runs.

Time steps	**increasing** $N(t)$	$N_{const} = 10$	$N_{const} = 5$
MSO8	6.06×10^{-4}	4.60×10^{-4}	6.63×10^{-4}
Triangle4	2.09×10^{-4}	2.75×10^{-3}	2.29×10^{-3}

new data injected into the window after a single time step shift. To understand the impact of this new data, its amount must be considered relative to the data retained in the sliding window after the shift. This ratio is small in larger sliding windows, resulting in only minor changes to the contained data after a single time step movement. Consequently, $e_{T_W}(t_c)$ exhibits smooth reductions throughout the synchronization sequence. In contrast, with smaller sliding windows, the newly introduced data constitutes a significant portion of the window's content, exerting a more substantial influence on $e_{T_W}(t_c)$, which is prone to drastic changes after a single window shift. This leads to an unsmooth error curve, increasing the risk of local optima and premature convergence. Therefore, larger sliding windows are more suitable for ensuring the correct identification of the time series.

However, the enlargement of the sliding window should be judicious. An excessively large sliding window is undesirable either, as it can degrade the quality of the time series decomposition and significantly increase computational effort. The experiments revealed a degradation in decomposition quality when the sliding window size was comparable

to the synchronization sequence length. Considering that the leading time steps of the sequence are already incorporated in the sliding window, the principal disadvantage of a large window is that it diminishes the time steps remaining for permutation. This reduces the likelihood of identifying matching counterparts for the evolving modules in the ensemble. Besides, a large window size requires greater computational resources for evaluating updated module states using Eq. (4.5). Since this does not guarantee improved performance, the enlargement of the sliding window must be justified. Consequently, the window size should be carefully determined through preliminary runs to estimate the potential performance gains associated with a larger window.

5.4. Decomposition of non-stationary time series

In contrast to stationary time series, the parameters and structure of dynamic components in non-stationary time series evolve over time. This exacerbates the problem, demanding that the algorithm detects changes within a limited time frame, thereby significantly emphasizing the importance of rapid convergence. The algorithm is usually devoid of direct information about relevant changes to the underlying data structure. However, simulated environments offer the unique opportunity to record the precise time at which a significant change occurred in the time series' structure. This recorded information can then be leveraged in performance analysis to evaluate the algorithm's decomposition ability to recover from such changes.

To demonstrate the capabilities of the developed approaches, several studies explicitly incorporated non-stationarity into the utilized data. [AT17] evaluated a Multi-Timescales RNN on sine wave sequences, whose period and amplitude were regularly perturbed by normally distributed random values. [ZMA99] analyzed data generated by threshold autoregressive models, which simulate the abrupt activation and deactivation of noisy linear models. In [Kha06], non-stationarity was simulated by switching the underlying data-generating processes. Specifically, the same process generated the data for two distinct settings before and after the switch. The objective was to see whether the modular learning system could effectively reuse previously acquired information following the change.

Non-stationarity can also be intrinsic to the environment in which the developed approach is intended to operate. [BBE11] worked with multivariate piecewise stationary time series to predict temporal changes in gene networks, where conditional dependencies between parameters could change at specific time points. In [Pas+01], a mobile robot had to survive in the environment that was regularly altered by the introduction of new obstacles. The environment of the mobile agent was artificially modified in [GOB17] by randomly introducing a new object of interest after processing the previous one. In vowel recognition with multiple speakers, as studied in [PJT95], the data characteristics switched depending on the current speaker. In handwritten text generation, as explored in [CAB17], non-stationarity arose from the variable length of written words and required

the detection of separations between consecutive words. [RH10] assumed gradually changing characteristics in sequence data obtained from evolving networks, such as those found in developing biological organisms or changing traffic patterns during a day.

Several studies on modular networks have proposed various approaches to detect data non-stationarity. In most of these, a dedicated change detector augmented the capabilities of the adaptation algorithm. [GOB17] developed an event transition detector based on statistical evaluations of the prediction error to segment the information flow in a sequence of models. In this architecture, event boundaries were identified by a significant increase in the prediction error exceeding a predefined uncertainty threshold. In another study, [CAB17] incorporated a parameterized boundary detector into a hierarchical multi-scale LSTM model for handwriting sequence generation to detect transitions between consecutive words. This boundary detector aimed to identify boundaries by optimizing the overall target objective, facilitating the updating of low- and high-level modules at different rates. [Kha06] utilized knowledge about the time of changing the data structure to examine the co-evolutionary decomposition algorithm's ability to reuse previously learned information. This study confirmed information reuse and demonstrated that the network learned more rapidly than if it had learned from scratch.

No explicit detector of data non-stationarity was implemented in, for example, [PJT95], where the output of the HME network was controlled by the output signal from the gating networks. Similarly, switching between regimes was realized through dynamic weighting of autoregressive models in [ZMA99]. In [AT17], a Multi- Timescales RNN leveraged the attractor characteristics of RNNs to handle parameter changes in the underlying data dynamics. [RH10] proposed Dynamic Bayesian Networks tailored to address data non-stationarity by identifying the conditional dependencies within the given time series.

Properties of non-stationary data

In the considered time series, non-stationarity was implemented as a regular change in the parameters of sinusoidal signals, where the amplitudes α_k and phases ϕ_k abruptly changed every 100 time steps. Each new amplitude α_k was selected as a random value uniformly distributed within the interval $[0, 1]$, and each new phase ϕ_k was a random value uniformly distributed within the interval $[0, 2\pi]$. A similar approach to generating the non-stationary sequence data was used in [AT17], where the RNN performance was analyzed on a single sine wave time series with abrupt amplitude changes. The amplitude for each subsequent interval was a normally distributed random value with a mean of 1 and a standard deviation of 0.2, limiting the amplitude difference between consecutive intervals.

The regular abrupt changes make the considered non-stationary oscillatory dynamics comparable to the discontinuous sequence data discussed later. In both cases, the

preceding flow of the given sequence is interrupted and resumes from a completely different point. Consequently, the synchronization algorithm must reconfigure the module state populations to catch up with the evolving dynamic components for continued tracking.

Performance on non-stationary data

In contrast to stationary data, the parameters of non-stationary dynamic components change over time, making the computation of test error challenging. However, a strong correlation between errors on the synchronization and test sequences (see Figure 5.1) indicates that synchronization error serves as a reliable performance indicator even for non-stationary data. This is particularly advantageous for the automatic evaluation of decomposition quality – a low synchronization error strongly suggests that synchronization has accurately identified the underlying structure of the time series. Table 5.12 presents the achieved accuracy and identification rates for non-stationary MSO time series using both neural and parametric modules. Due to the neural modules' capacity to generate a broader range of dynamics, population diversity was higher, leading to a more thorough exploration of the module state space and consequently, superior synchronization performance. Conversely, the constant amplitude and phase of the parametric modules within the sliding window resulted in lower diversity and diminished identification rates for more complex time series.

Table 5.12.: Synchronization performance on non-stationary MSO time series with randomly changing amplitudes and phases every 100 time steps. (These data were published in our earlier paper [KOB21]).

Module type	Time series	Average test RMSE	Standard deviation	IR (%)
Neural	**MSO1**	1.45×10^{-3}	4.19×10^{-3}	94.33
	MSO2	7.07×10^{-3}	2.96×10^{-2}	89.33
	MSO4	9.09×10^{-3}	1.70×10^{-2}	87.33
	MSO6	1.05×10^{-2}	1.57×10^{-2}	89
	MSO8	1.29×10^{-2}	1.30×10^{-2}	92.33
Parametric	**MSO1**	2.05×10^{-3}	3.87×10^{-3}	97.33
	MSO2	7.00×10^{-3}	2.09×10^{-2}	93.33
	MSO4	7.11×10^{-2}	0.108	49.66
	MSO6	0.176	0.180	2.66
	MSO8	0.411	0.374	1

Besides, the presence of non-stationarity necessitates the evaluation of the synchronization convergence rate to determine the minimum interval between consecutive shifts in the target dynamics that the algorithm can tolerate. Figure 5.14 depicts the error reduction between consecutive changes in the parameters of the dynamic components. As shown in this figure, the algorithm automatically begins tracking the changed dynamic components immediately after a non-stationary change, without relying on an explicit

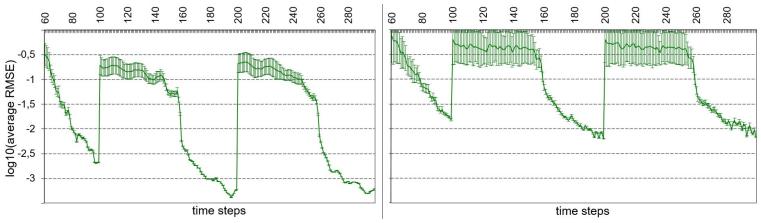

Figure 5.14.: Average error curves of neural ensembles on non-stationary MSO1 (left) and MSO8 (right). Abrupt changes in the time series result in error increases. Following each change, synchronization reduces the error to a low level, which is lower for the less complex MSO1 time series. (This figure was published in our earlier paper [KOB21]).

non-stationarity detector, unlike other approaches (e.g., [CAB17]). Due to the growing discrepancy between the ensemble's total output $\mathbf{o}(t)$ and the given time series $\mathbf{y}(t)$, the error $e_{T_W}(t_c)$ (see Eq. (4.5)) increases. This disrupts the performance ranking within the population, compelling permutation to intensify the reshuffling of module states across ensembles. Concurrently, the newly combined module states are rigorously tuned for mutual consistency. The onset of rapid accuracy improvement is observed following a time interval equal to the sliding window's width, when outdated information is purged from it. Thus, the sliding window's length reduces the number of time steps available for synchronization between consecutive abrupt changes in the time series.

Convergence is numerically quantified by the average time required to reduce the initial error by two orders of magnitude. Table 5.13 presents this metric for both stationary and non-stationary data, categorized by module type. Generally, higher time series complexity correlates with slower convergence for both stationary and non-stationary data. However, for the simpler time series MSO1 and MSO2, convergence times were shorter for non-stationary data. The observed speed-up was due to the populations' ability to gather information concerning the target dynamics, including the number and frequencies of the constituent oscillations, which did not change over time. Compared to parametric modules, neural modules exhibited shorter convergence times for the previously outlined reasons.

The algorithm maintains its capacity to resume synchronization following each non-stationary change due to the residual diversity within the populations, even after achieving a steady state (see the population spread in Figure 5.6). Indeed, in the converged state, the populations contain a significant proportion of alternative module states that deviate from the globally optimal ones. These states arise not only from less effective updates but also from their dependence on the linked context modules $\mathbf{C}_{m,p_i}(t)$ (from Eq. (4.6)). This dependence implies that, during synchronization, the states of modules within the same ensemble become increasingly interconnected, culminating in a strong linkage by the end of synchronization, as evidenced by the decreasing intensity of permutation. The residual diversity of module states becomes advantageous

Table 5.13.: Average time required to reduce the initial error by two orders of magnitude for neural and parametric modules on stationary and non-stationary data.

	Time series	Neural modules	Parametric modules
Stationary	**MSO1**	64.25	55.75
	MSO2	66.37	67.65
	MSO4	69.38	81.24
	MSO6	69.89	90.98
	MSO8	70.23	98.73
Non-Stationary	**MSO1**	59.78	58.94
	MSO2	63.77	79.91
	MSO4	75.13	92.81
	MSO6	76.95	96.87
	MSO8	78.66	98.91

when the increasing error $e_{T_W}(t_c)$ implicitly triggers intensive exploration of the search space through the exchange of module states between previously established ensembles. Therefore, techniques to artificially increase population diversity, like hypermutation [COY21; SC02], are rendered unnecessary.

5.5. Concluding summary for oscillator modules

This chapter presented the studies on time series decomposition through online tuning of dynamics in ensembles of independent oscillator modules, organized in a single-layer modular network. Leveraging the network's inherent redundancy and the modules' capacity to model assumed dynamic components within specified variation ranges, the proposed solution employs synchronization of the internal dynamics of relevant modules and the exclusion of irrelevant ones. The conducted experiments demonstrated that the variation of module dynamics must be appropriately constrained to achieve successful time series decomposition via online tuning of the modules. Thereat, the probability of accurate decomposition increases by narrowing the variation ranges of module parameters and reducing the plasticity of individual modules, thereby minimizing functional overlap.

The approach was evaluated on diverse compositional time series to analyze its scalability with increasing data complexity. In the experiments, time series complexity was varied by increasing the number of mixed dynamic components. Furthermore, the approach was challenged by incorporating non-differentiable oscillating signals. Experiments with such signals demonstrated the approach's flexibility, enabling it to operate with any suitable module type – a feature particularly beneficial when signals cannot be effectively modeled using compact neural modules. Both observed identification rates and modeling accuracy exhibited a gradual decline with increasing complexity across all time series. A

more pronounced performance degradation was noted following the inclusion of non-differentiable oscillating signals, indicating that the problem became more challenging and would necessitate larger populations of module states and longer sequences for accurate identification of the dynamic components. Good generalization was indicated by a strong correlation consistently observed between test and synchronization errors across all time series, suggesting that their similarity can be utilized as a reliable indicator for the correctness of the obtained data decomposition.

The thesis defined the synchronization of module dynamics as an optimization problem, to be resolved with respect to module states, to maximize the reconstruction quality of the time series by the entire model. This prompted the research question regarding the optimization method suitable for the given compositional time series and the deployed module types. Investigations revealed a significant influence of these factors on the objective function's smoothness and modality, motivating the consideration of two alternative optimization methods: differential evolution and gradient descent. Differential evolution is known for its weak dependence on objective function properties, while gradient descent is recognized for its rapid convergence.

As observed in the experiments, the gradient-based tuning's faster convergence was distinctly advantageous on simpler time series. However, its greedy nature increased the risk of premature convergence and resulted in significantly lower identification rates compared to differential evolution, which proved more favorable for complex time series with a greater number of dynamic components. Furthermore, due to its enhanced flexibility, the evolutionary method was applicable across all considered module types, as it does not rely on the derivatives of the module's output signal. Nevertheless, its performance also experienced premature convergence on the most complex time series. This suggests that more complex time series necessitate larger populations and sliding windows, regardless of the chosen optimization procedure. The performance's dependency on the methods' parameters, such as the learning rate and crossover probability, varied across the time series and would require individual consideration to achieve optimal performance on other datasets.

On non-stationary time series, a short interval between abrupt changes in the underlying data structure posed a significant challenge to convergence, particularly due to the absence of a non-stationarity detector. Such a detector could have provided information about potential changes in the target dynamics, facilitating the restart of synchronization or triggering hypermutation to enhance population diversity, as implemented in other studies. Nevertheless, the algorithm successfully tracked the changes without explicit notification. This suggests that a non-stationarity detector is not essential for decomposing non-stationary compositional time series. The observed acceleration of convergence over time indicates that the populations accumulate information about changes, enabling faster adaptation to subsequent changes in the time series' underlying structure. However, the sliding window's size limits tracking capabilities and requires consideration of the frequency of non-stationarity changes and time series complexity.

The design of the deployed neural oscillator modules revealed that their structural properties determined their modeling capabilities and the characteristics of the objective function. Consequently, appropriate module design is a crucial factor influencing decomposition success and must align with the properties of the modeled oscillating signals. Recognizing the variations in the characteristics of the oscillating signals under consideration, this thesis explored diverse design approaches. The investigation's findings indicated that modeling sinusoidal oscillations necessitates neural modules where the driving OFB signal is balanced with the internal recurrent signal, as proposed in [KLB12]. Specifically, modeling sinusoidal signals requires stringent limitation of the OFB signal through reduction of the OFB weights in recurrent neural oscillators operating auto-regressively. Differently, modeling of triangular oscillating signals required neural modules with internal recurrent signals that abruptly alter the signal gradient at switching points. This suggests that modeling the triangular oscillating signals necessitates neural modules with unbalanced hidden neurons, operating in saturation of the TANH activation and synchronously swapping their states, dependent on the modeled oscillation's period.

The experiments demonstrated significant flexibility of the synchronization method regarding the deployed module type – the algorithm successfully identified components in the given sequences using both recurrent neural and parametric modules. Among these, the neural modules exhibited higher plasticity, resulting in a smoother objective function and, consequently, improved accuracy and somewhat higher identification rates. Conversely, the parametric modules' known parameter semantics render them more suitable for understanding the underlying data-generating process's nature. The method's ability to identify frequency components in the given time series prompted an question about its competitiveness compared to FFT analysis, which produces a frequency spectrum and can also be used for sinusoidal oscillation identification in sequence data. Indeed, the experimental data revealed that the synchronization method required significantly shorter sequences for successful sinusoidal component identification, albeit with a considerably higher computational cost. This suggests that, for very short data sequences, the synchronization method can achieve even greater decomposition accuracy than FFT, making its application in such cases worthwhile.

Beyond compositional time series, the synchronization method's identification capability was further examined using MG time series. In contrast to compositional time series, this chaotic attractor lacks an a priori known decomposable underlying dynamic structure that can be partitioned into a set of independent dynamic components; thus, one cannot generally expect chaotic attractors to be representable as a linear combination of a few simple modules. Nevertheless, the method obtained a quite precise reconstruction of the chaotic attractor sequences using a linear combination of periodic signals for a number of time intervals. This finding demonstrates that the proposed synchronization method is applicable to the decomposition and identification of dynamic components in time series, even without a confident assumption about their compositional structure.

6. Synchronization of co-evolving aperiodic modules

Previous chapters detailed the synchronization approach and investigated its application to the decomposition of continuous oscillatory dynamics. The experiments demonstrated that high synchronization accuracy of oscillator modules was achieved within a relatively short time. This suggested the method's applicability to time-limited, discontinuous sequence data, which are confined to a specific time span and do not exist beyond it. This type of data does not demand uninterrupted contributions from the relevant recurrent modules, in contrast to oscillator modules, which produce output signals with periodic temporal patterns. Instead, discontinuous sequence data can be modeled using aperiodic dynamic modules. Their output signals do not repeat periodically and are applicable for sequence modeling solely within a specified time interval. Outside this interval, the modules' output signals are irrelevant and can be disregarded. Aperiodic modules offer the advantage of modeling locally complex dynamic patterns that need not be repetitive. This reduces the complexity requirements for individual modules, eliminating the need for excessively large modules that are unsuitable for synchronization. The synchronization method's task then becomes identifying the given dynamics by establishing a suitable match between the given sequence data and the aperiodic module outputs. This chapter details the properties of discontinuous sequences of handwritten symbols, elaborates on the design methodology for aperiodic dynamic modules, underscores the key differences in their synchronization compared to oscillator modules, and presents the experimental results from applying aperiodic modules for handwritten symbol recognition.

6.1. Different approaches to the recognition of handwriting

The goal of handwriting recognition is to determine the meaning of handwritten text or individual symbols. In most studies, the classifier is required to identify the class labels of text segments, i.e., the meaning of isolated symbols or even separate words. The handwritten text is presented to the classifier either as a sequence of coordinates of the pen trajectory or as a vector encoding the visual image of the written symbol. Therefore, a diverse range of classification techniques has emerged.

Competitive results in handwritten text recognition have been achieved using LSTM networks [Gra+08; GS08; Gra08] for word recognition and [Fab+20; FOB21] for symbol

recognition. In these approaches, the pen trajectory coordinates were sequentially fed into the trained model, which then identified the most probable class from a set of known classes. These models were trained on extensive datasets, comprising approximately 20,000 frequently occurring words or covering various writing styles for all 26 alphabetic symbols. Consequently, the models were large to memorize the substantial training data, with the number of trained weights ranging from 10^5 in the offline recognition setup [GS08] to about 1.2×10^5 in the online recognition setup [Gra+08]. Despite achieving a high recognition rate of approximately 97% for individual symbols, the word recognition rate was somewhat lower, around 70%, likely because of the significant variation in writing styles. This would likely necessitate even larger networks to achieve higher recognition rates.

Statistical pattern recognition is another machine learning field where handwritten character recognition has been approached using techniques such as Support Vector Machines [Vap95; Bur98; Aro+10] or polynomial classifiers [Sch96; KS97]. Each written symbol is represented as an image pattern and is fed at once as a vector to the classifier's input. The models in statistical pattern recognition implicitly encode the posterior probability densities and return the most probable class label for the given input pattern.

Additionally, there are approaches that analyze the drawn curves of symbols and compare them with available templates for recognition. In most of these methods, the symbol curve analysis involves preliminary preprocessing and is coupled with an ANN classifier. The preprocessing stage aims to extract characteristic features of the drawn symbol's curve, such as the distance from the curve's start, the angle, and the arc-to-chord ratio, as demonstrated in [MMD17], for feeding them to the neural classifier. In contrast, my synchronization method identifies the most suitable module and provides parameters of the recognized symbol curve as output. Another group, template-based methods, exemplified by [CY98], relies on comparing the shapes of available symbol models with the given handwritten symbol curves. [RWH96] presented a symbol recognition method based on matching generative models to images of submitted symbols. Here, the analytical form of elastic splines was used not only for recognition but also for extracting additional information, such as the instantiation parameters of the submitted symbols. This extraction of additional information and the locality of the match make this method similar to the synchronization approach, although there are principle differences between them, including the RNN model representation versus analytical representations, and the representation of written symbols as pen trajectories versus their representation as image patterns.

6.2. Discontinuous sequence data

The representation of handwritten symbols as discontinuous sequences made handwritten symbol recognition a suitable application for investigating the synchronization of aperiodic modules. However, the characteristics of these symbol sequences, specifically

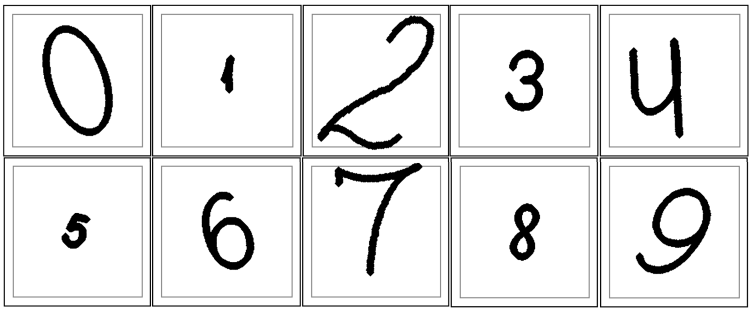

Figure 6.1.: Examples of single-class sequences for symbol classes “0” through “9” from the database created as described in subsection 6.2.2.

their discontinuity and reduced information content, demand particular attention to both the algorithm’s flow and the architecture of the recurrent modules. Given that short symbol sequences limit the achievable modeling accuracy, class decisions must be made by accumulating class statistics, unlike the approach used in oscillator module synchronization. Moreover, these short sequences are primarily sufficient only for training compact aperiodic modules, which must still be able to handle a wide variety of handwritten symbol curves. The subsequent paragraphs detail the symbol sequences used, their properties, and the challenges addressed.

6.2.1. Sequence data of the handwritten symbols

Handwritten symbols are represented as time-limited, discontinuous two-dimensional sequences, where each sample is an (X, Y)-coordinate of a drawn symbol’s point on the plane. Unlike the one-dimensional sequences discussed earlier, these two-dimensional sequences are not composed of periodic oscillators and exist only within a finite time interval, defined by the start and end positions of the pen drawing the symbol. The sequence length is physically constrained by the sampling resolution of the drawn curve, imposing an additional limitation on synchronization. This constraint prevents arbitrarily prolonging the sequences to achieve more precise module synchronization. The proposed method was evaluated on both single-class and multi-class sequences. In the single-class scenario, each sequence comprises only the (X, Y)-coordinates of points from the drawn curve of a single symbol. Examples of single-class symbol sequences are shown in Figure 6.1. The algorithm is required to identify the correct class of the drawn symbol, irrespective of its orientation and scaling.

In addition to the single-class sequences, this thesis analyzed the method’s behavior on more complex multi-class sequences. These sequences were generated by combining multiple single-class sequences, each with distinct class labels, orientation angles, and symbol scaling. Their parameter values were selected stochastically and independently of each other. Examples of these multi-class sequences are shown in Figure 6.2. Upon examination, it becomes apparent that humans struggle to discern the symbols combined

Figure 6.2.: Examples of multi-class sequences. Each sequence is a mixture of three symbols from distinct classes, randomly selected from the set of all considered symbol classes, ranging from "0" to "9". The shown examples illustrate mixtures of the symbols (from left to right): ("0", "7", "9"); ("1", "2", "4"); ("8", "5", "6"), ("0", "1", "8") and ("2", "5", "7").

within each sequence. Nonetheless, the algorithm is required to accurately classify the composite symbols.

The single-class sequence dataset was automatically generated by transforming several original template sequences that were handwritten beforehand. A single template sequence was recorded for each symbol class. The transformation of these template sequences introduced the necessary data variation and was achieved by scaling the template by a scaling factor randomly selected from the interval $[0.1, 1]$ and tilting it by an angle randomly selected from the interval $[60^\circ,\ 120^\circ]$. Generating the sequences by varying these parameters preserved the basic form of the symbol's curve, allowing a single compact module to reproduce all sequences of a single class. Introducing more variety to the sequence dataset by varying the basic form of the symbols' curves would necessitate additional modules for each symbol class. This was intentionally avoided in the presented experiments, as their objective was to demonstrate the method's principle ability for handwritten symbol recognition, rather than to construct a high-performance recognition system competitive with advanced systems in this field. The synchronization algorithm must be able to discriminate between the curves of different classes and should be regarded as a promising alternative approach.

6.2.2. Challenges of the handwritten symbols data

To achieve reliable identification, the sequence must provide sufficient information about the recognized symbol. This imposes a lower bound on the resolution for recording the drawn symbols. The applied resolution is directly related to the sequence length – higher resolution yields longer sequences. The requirement for longer sequences precluded benchmarking the algorithm on open datasets like MNIST, leading to the creation of an own database for experiments on single-class (see Figure 6.1) and multi-class sequences (see Figure 6.2). The database was populated with variations of originally handwritten symbols, generated by modifying the orientation angle and scaling factor. (A similar automated approach for constructing a synthetic handwriting sequence database was previously employed in studies such as [Luk10] to investigate self-organizing reservoirs of RBF neurons.) In the created database, the symbol "1" had the shortest sequence,

comprising 190 time steps (see Table 6.1). As shown in the table, sequence lengths varied across different symbols. The inherent limitation on freely selecting the sequence lengths for discontinuous sequence data distinguishes synchronization from, and renders it more challenging than, synchronization on continuous oscillatory time series, as presented in the preceding chapter.

Table 6.1.: Sequence lengths corresponding to each symbol class, "0" through "9".

Symbol	**"0"**	**"1"**	**"2"**	**"3"**	**"4"**	**"5"**	**"6"**	**"7"**	**"8"**	**"9"**
Sequence length	300	190	209	357	336	354	402	230	300	402

The substantial similarity between drawn symbol curves of different classes within certain sequence intervals presents a major challenge to synchronization on these two-dimensional sequences. As illustrated in Figure 6.1, a vertical line segment is present in the symbols "1", "4", and "7". A clear similarity is also observed in the curvature of the drawn symbols "0", "6", and "9'". This increases the risk of misidentification. Conversely, the greater the dissimilarity between available primitive dynamics, the lower the risk of confusing them within the sliding window. This implies that changes in the states of an individual module are likely to result in corresponding changes in the overall output characteristics, with a reduced risk of premature convergence. For instance, a larger difference between frequencies of two oscillator modules compels the algorithm to concentrate on the relevant module and refine the mismatching frequency, resulting in improved overall output.

The convexity of time series curves can be used to quantify similarity between the symbol curves. As the second derivative of a function reflects the curve's convexity, the average convexity of the symbol curves is estimated by the average magnitude of the second derivative across the sequence, as follows.

$$\hat{y}''(t) = \frac{1}{T_{synch}} \sum_{t=1}^{T_{synch}-2} \left| \frac{y(t+2\Delta t) - y(t+\Delta t) - y(t)}{\Delta t} \right| \tag{6.1}$$

where T_{synch} is the synchronization sequence length, and Δt is the time interval between two consecutive samples in the sequence. In the experiments, time was measured in integers; therefore, Δt was set to 1. Table 6.2 and Table 6.3 present the estimated average convexity for the one-dimensional and two-dimensional sequences, respectively. Due to variations in phase and orientation angle across the available sequences, the minimum and maximum values were calculated for each symbol curve. As observed, the intervals do not overlap for the one-dimensional time series, suggesting clearer discrimination between the modules within the ensemble. In contrast, for the two-dimensional time series, the intervals are very close and even overlap. This indicates more challenging discrimination and a higher risk of confusion between the modules during time series

identification.

Table 6.2.: Average estimated convexity per sequence in the considered one-dimensional time series. The values vary between the minimum (*min*) and maximum (*max*) values because of different phases for generating the sequences. The $[min, max]$ intervals do not overlap.

Primitive dynamics	Min / Max average convexity per sequence
sin(0.2)	2.532×10^{-2} / 2.559×10^{-2}
sin(0.311)	6.098×10^{-2} / 6.135×10^{-2}
sin(0.42)	0.1102 / 0.1110
sin(0.51)	0.1615 / 0.1622
sin(0.63)	0.2428 / 0.2456
sin(0.74)	0.3316 / 0.3335
sin(0.85)	0.4312 / 0.4333
sin(0.97)	0.5511 / 0.5537

Compared to the values in Table 6.2, the average convexity of the considered two-dimensional dynamics is significantly lower, indicating a very gradual change in the gradient along the curves. This numerically confirms the statement made in [KOB21] for two-dimensional oscillators: *"In contrast to the MSO dynamics, the unfolding of the considered 2D curves is slow and provides scarce information for discrimination between the curves within a single sliding window."* (p.23) In this scenario, a single-step move along the slowly changing curve yields either no or minimal new information for synchronization, increasing the risk of converging to an undesirable local optimum.

Table 6.3.: Average estimated convexity per sequence in the two-dimensional curves of the handwritten digits. The values vary between the minimum (*min*) and maximum (*max*) values because of the different orientations of the symbol curves across the sequences. The $[min, max]$ intervals are closely spaced and overlap.

Primitive dynamics	Min / Max average convexity per sequence
Symbol "0"	1.076×10^{-4} / 1.228×10^{-4}
Symbol "1"	1.044×10^{-3} / 1.665×10^{-3}
Symbol "2"	2.305×10^{-3} / 2.720×10^{-3}
Symbol "3"	1.041×10^{-3} / 1.249×10^{-3}
Symbol "4"	1.007×10^{-3} / 1.430×10^{-3}
Symbol "5"	1.155×10^{-3} / 1.348×10^{-3}
Symbol "6"	1.330×10^{-3} / 1.521×10^{-3}
Symbol "7"	1.140×10^{-3} / 1.608×10^{-3}
Symbol "8"	1.515×10^{-4} / 2.162×10^{-4}
Symbol "9"	1.330×10^{-3} / 1.521×10^{-3}

6.3. Evolutionary synchronization for symbols recognition

The inherent differences in the properties of oscillatory and aperiodic dynamics need careful consideration in the synchronization method. These differences encompass the sequence length and the higher dimensionality of sequence data. The low variability of discontinuous sequence data along relatively short sequences precludes the precise long-term tuning of the modular network. Conversely, the classification of handwritten symbols does not require parameter identification of the drawn curves; instead, the algorithm must identify the dynamic modules best suited for modeling the given two-dimensional sequence, without necessitating precise module tuning within the ensemble. However, the high similarity between symbols within certain sequence intervals increases the risk of confusion. This motivated the integration of an evolutionary approach with statistical decision-making to achieve reliable recognition, as realized by the collection of class statistics in Algorithm 3. The higher dimensionality of the sequence data demands longer individuals $\mathbf{p}_i = \{\mathbf{r}_m, X_0, Y_0, \mathbf{x}_m(t), \alpha\}$, which additionally encode symbol tilting via the orientation angle α and the X_0 and Y_0 coordinates for the center of the two-dimensional curve. The module responsibilities $\mathbf{r}_m$ encode the normalized size of the symbol.

Significant disparities in the complexities of dynamic components within the considered discontinuous sequence data result in a substantial imbalance among the available modules in the ensemble, further complicating synchronization. The intricate curves of certain symbols need larger modules, producing more complex objective functions. Their dimensionality, corresponding to the number of hidden neurons in the RNN modules, ranges from five for the smaller modules of simpler symbols like “0” and “1” to nine for symbols “3” and “5”, which exhibit more complex curves. In addition to the differences in search space dimensionality, the inherent non-linearity of the objective function, dependent on the movement dynamics used to draw the symbols and generally distinct for each symbol, further influences problem difficulty. The more complex objective functions demand greater computational effort. This must be accounted for during synchronization to prevent the unintended rejection of relevant modules because of insufficient tuning. The necessary balance in module tuning is achieved by more frequent state updates for larger modules; specifically, evolutionary tuning generates more generations for larger modules at each time step. The module margin $\Delta\mu_m$ in the static constraints (4.47) and (4.48) provides an additional mechanism for balancing the tuning effort across search spaces of varying complexity. Unlike synchronization on continuous oscillatory time series, individual tuning of module margins is crucial for balancing the non-oscillatory modules of the symbols to ensure optimal rejection. Alongside the algorithmic nuances of synchronization described below, the need for balancing among available modules distinguishes symbol sequence recognition from the identification of oscillatory dynamic component and emphasizes the importance of effective module design.

A comparison of Algorithm 1 and Algorithm 3 reveals that, for discontinuous sequence

Algorithm 3 Synch($\mathcal{M}, \mathbf{y}(t)$)

```
Initialize {P_1, ..., P_M}                          ▷ apply the valid ranges (3.6) and (3.7)
Initialize {A_1 = 0, ..., A_M = 0}                  ▷ initialization of WSA
Assign W_t from y(1) to y(T_W)                       ▷ sliding window over time series
while until termination criterion is not fulfilled do
    for t = T_W to T_synch do                        ▷ progress along the sequence
        Adapt ω^min and ω^max                        ▷ handle adaptive constraint (4.50)
        for n = 1 to N(t) do                         ▷ number of population updates (4.53)
            for m = 1 to M do                        ▷ optimizing individual populations
                if A_m was not reset then
                    Tuning of P_m on W_t             ▷ apply Algorithm 2
                    Permutation of P_m
                end if
            end for
        end for
        Update A from {P_1, ..., P_M}                ▷ increment A for winning P_m
        Update W_t with y(t + 1)                     ▷ advance the sliding window
        Advance {P_1, ..., P_M}                      ▷ update the module states for t + 1
    end for
    Rejection of modules                             ▷ apply Algorithm 4
    Reinitialize {P_1, ..., P_M}                     ▷ whole p_i except r_m
end while                                            ▷ until termination criterion is not fulfilled
Determine the class of y(t)                          ▷ Search for maximum in A
```

data, the given sequence $\mathbf{y}(t)$ is iteratively processed until the termination criterion is satisfied. To integrate evolutionary tuning with statistical decision-making, a winner score accumulator was introduced to gather statistics on the available classes in the given sequence. A detailed description of the algorithmic extensions and an analysis of the observed behavior are presented and analyzed in the subsequent subsections. Given the superior performance of the evolutionary approach over the gradient-based approach in experiments on oscillatory time series, the synchronization behavior for the more challenging aperiodic dynamics was investigated exclusively within the evolutionary framework.

6.3.1. Initialization

The disparities between the properties of components in continuous oscillatory time series and those in the considered discontinuous sequence data necessitate distinct initialization approaches. In the case of oscillatory dynamics, the potentially unlimited sequence lengths, coupled with the unlimited autoregression of available modules, ensure a seamless evolution of module states from initialization to precise dynamic state identification. In contrast, the relatively short lengths of the drawn symbols' curves

limit the operational duration of corresponding modules. With the exception of symbols “0” and “8”, all considered symbols are non-symmetric, non-closed-loop curves. This implies that only the modules representing symbols “0” and “8” can maintain oscillatory dynamics and autoregressively reproduce the symbols’ curves. Modules for other symbols require external intervention to initiate the drawing of their corresponding symbol curves upon subsequent requests after previous drawing requests have been completed. The dynamics of these modules are non-oscillatory, meaning the module states at the end of the drawn curve differ significantly from the initial states. The requirement to reset module states to their starting point presents a challenge in handling aperiodic dynamics. A seemingly straightforward solution – assigning pre-stored initial module states – is impractical, as the parameters of the symbol curve are unknown a priori. Combined with the low information content of the symbol sequences, this motivated repeated sequence processing with module state re-initialization at the beginning of each iteration, leveraging the increasingly reliable information about the given symbol curve.

The absence of information regarding the current state of dynamic components in the sequence data restricts initialization of module state to uniformly distributed random values. This represents a worst-case scenario for algorithm convergence, potentially leading to failure on the relatively short two-dimensional dynamic sequences if populations are "unluckily" initialized with individuals that either sparsely populate or completely omit the attraction region of the global optimum. As a result, convergence failure results in erroneous statistics, which are inadequate for the recognition of symbol curves. Generally, the collected class statistics fluctuate across sequence iterations when the algorithm starts with different initial conditions. To reduce the dependency of recognition on initial conditions, the obtained class statistics must be accumulated over multiple sequence iterations before the algorithm begins rejecting irrelevant modules. This mitigates the adverse effects of erroneous statistics and enhances recognition reliability. The issue of "unlucky" initialization has previously attracted research attention, as exemplified in [Kha06]: *”In every generation, each System needs to be trained multiple times, with different weight initializations, before fitness evaluation to average out the fluctuations in fitness.”* (p.89), aiming to minimize the negative impact of stochastic effects in EAs on model performance.

Reliable identification and rejection of irrelevant modules after several sequence iterations reduces the compound dimensionality of the search space, thereby gradually decreasing problem complexity. Synchronization then focuses on tuning only the relevant modules, whose parameter values progressively approach the true values of the corresponding hidden dynamic components – the algorithm gains deeper insights into the dynamic components. Consequently, it becomes advantageous to reuse the collected information for partial initialization in subsequent sequence iterations. This involves preserving the values of non-transient attributes – module responsibilities and the orientation angle in individuals from the previous sequence iteration, rather than resetting them to random values. Unlike full initialisation, information reuse accelerates synchronisation by distributing initial module states closer to the searched global optima, reducing

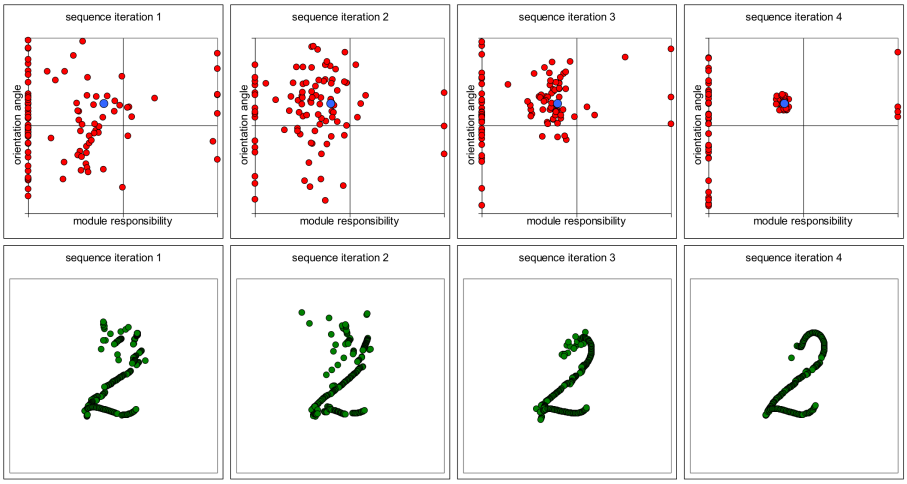

Figure 6.3.: Convergence of orientation angle and responsibility within the population (red points) of the symbol "2" neural module over four sequence iterations (upper row) towards the optimum (blue circle), demonstrating simultaneous improvement in symbol curve reproduction quality (lower row).

failure risk and minimizing relevant module rejection. In the experiments, the modules for oscillatory dynamics "0" and "8" were treated identically to other modules to avoid exacerbating the inherent imbalance among available modules, even though re-initialization is technically redundant for oscillatory dynamics.

Figure 6.3 illustrates how the time-invariant parameters, module responsibility and the orientation angle of the drawn symbol curve, progressively converge towards their target values across sequence iterations for the symbol "2". This demonstrates the advantage of retaining their attained values from the preceding sequence iteration. The same figure also reveals a consistent improvement in the reproduced symbol curve's quality, which aids in the accurate recognition of the target symbol class. Consequently, insufficient convergence of module responsibility and orientation angle, coupled with the low quality of the symbol curve after a single sequence, suggests that correct recognition of the target symbol class after a single sequence iteration remains uncertain. Furthermore, Figure 6.3 shows a high concentration of the population at a module responsibility of 0. These are module combinations dominated by modules with incorrect classes, potentially distorting accurate recognition.

6.3.2. Winner score accumulator

Given the short sequence lengths of symbol curves and the non-oscillatory dynamics of available modules, it is improbable that the synchronization algorithm attains a steady state by sequence's end. This necessitates reliance on statistical indicators rather than the performance of individual ensembles for dynamic component identification in the given

time series. The concept involves gathering statistics from the diverse module states within the population – relevant module populations are expected to generate suitable module states more frequently than irrelevant ones. Consequently, the synchronization algorithm must record how often each module accurately models the sequence data. These collected statistics represent the likelihood of available modules being relevant for modeling the given time series. Since each module corresponds to a specific class, the statistics over modules equate to class statistics, usable for symbol curve recognition. However, these statistics are influenced by initialization randomness and the stochastic nature of evolutionary update. Combined with the objective function's high complexity and the sequences' short lengths, these factors induce variations in collected data across sequence iterations. This advocates for accumulating statistics over several sequence iterations to establish a more reliable basis for distinguishing between relevant and irrelevant modules. The desired effect is reached through the partial compensation of potentially distorted class statistics in certain sequence iterations by the accurate statistics obtained from other iterations. This enhances recognition reliability but delays the final decision because of repeated sequence processing.

Basic idea of statistical decision making

Variations in the accuracy of total output make ensembles unequally suitable for updating class statistics. Clearly, ensembles with poor modeling accuracy do not accurately represent the true relationship between dynamic components in the time series. Their inclusion in the update must be avoided, as it introduces disruptive variations in the class statistics. Selecting only appropriate ensembles reduces variance in the basis for the update, thereby dampening variations in class statistics across sequence iterations and enhancing their reliability for decision-making. This raises the question of how to select the appropriate ensembles for the update. One potential approach is to select them based on modeling accuracy, requiring the ensembles' total output error to be below a predefined error level. However, this method has a significant drawback: the explicitly defined error level is overly restrictive, resulting in few or no ensembles meeting this criterion at the sequence's beginning, when the network's total output significantly deviates from the given time series. Consequently, the collected statistics are sparse and inadequate for decision-making. Instead of explicitly predefining the error level, specifying a small number of the best-performing ensembles for the update proves more efficient, regardless of their actual error. This provides a broader basis for collecting class statistics, with the taken ensembles' accuracy automatically improving over the sequence as populations converge to the global optimum.

Upon selecting the appropriate ensembles for update, their contents are analyzed to identify irrelevant modules. These modules contribute zero to the ensemble's total output. Algorithmically, they are recognized by their contribution satisfying the condition (4.51), indicating their candidacy for rejection. Conversely, the modules exhibiting the lowest rejection frequency are designated as winners, and their bins $\{A_1, \ldots, A_M\}$ are

incremented in the WSA at each time step. Initially, all scores are set to zero ($A_m = 0$). Subsequently, the WSA values $\{A_1, \ldots, A_M\}$ are updated until the end of synchronization. At this point, the highest scores indicate the modules whose dynamics are most prevalent in the given sequence data. A single-class sequence is then classified as the symbol corresponding to the highest-scoring module's class. For multi-class sequences, the recognition result is the combination of the classes of all non-rejected modules.

Selecting the highest accumulator score aligns with the maximum likelihood principle, which aims to determine unknown parameters that maximize the conditional probability (see related remarks in [Bis09], p.23). WSA effectively integrates template matching [CY98; RWH96] and statistical-based classification within the evolutionary synchronization framework. Template matching seeks to identify similarities between a given object contour and known template contours. In the context of synchronization, the object contour is represented by a two-dimensional sequence, potentially reproducible by available dynamic modules, which generate the template contours. In my investigations, module parameters (orientation angle and symbol size) and capabilities to combine the modules provided flexibility for the template contours. To determine the similarity between symbol curves and available modules, synchronization adjusts the recurrent modules, thereby shaping the template contours in various ways. The module offering the closest match to the given curve is designated as the winner class and its score is incremented in the WSA. At the end of synchronization, the final class decision is based on the WSA statistics: the processed curve is assigned the class label of the module whose output most frequently matched the curve during synchronization. The proposed WSA approach is similar to the tracking classification in [LKK04], which employed majority voting for robust traffic signal recognition by accumulating single classification results over a sequence of camera images. At the sequence's end, the most frequent classes were identified as the classes of the detected objects. In contrast to the WSA update, which analyzes module contributions, the tracking classifier in [LKK04] utilized a FF-ANN classifier to obtain single-step class statistics.

Winner score for module rejection

The accumulated class statistics fulfill a dual role: final identification of the time series content and rejection of irrelevant modules during synchronization. Given that the discontinuous sequences of symbol curves are too short for the algorithm to reach a steady state, the current module states cannot be analyzed using condition (4.51) to determine module rejection. Instead, the WSA status is analyzed at the end of each symbol curve pass to reject modules with the lowest scores, which are likely irrelevant.

Rejecting multiple irrelevant modules simultaneously offers the potential to accelerate symbol recognition, but must ensure safe rejection with minimal risk of discarding relevant modules. The developed rejection method operates on the assumption that all modules can be categorized into two groups based on their scores: promising modules and likely irrelevant modules. Promising modules represent symbols with curves similar

to the currently processed symbol's curve, resulting in relatively high scores. In contrast, likely irrelevant modules represent symbols with dissimilar curves, thus accumulating only low scores. The objective is to identify the boundary between the scores of these two groups. Thereafter, low-score modules beyond this boundary can be discarded as likely irrelevant, while promising high-score modules are retained for the next sequence iteration.

The decision on module membership in either group is implemented in Algorithm 4. This algorithm analyzes the WSA status by examining score differences and identifies modules with scores close to the highest-score module as promising. The remaining modules, with significantly lower scores, are classified as likely irrelevant and are rejected. Algorithm 4 begins by sorting the WSA bins $\{A_1, \ldots, A_M\}$ in descending order, creating the array $\hat{\mathbf{A}} = \{\hat{A}_1, \ldots, \hat{A}_M\}$. The promising module group *PM* is initialized with the module having the highest winner score, $\hat{A}_1$. If this score exceeds three times the second-highest score, $\hat{A}_2$, then $\hat{A}_1$'s module is the sole promising module, and all others are rejected. (The factor "three" was determined experimentally.) Otherwise, the module with $\hat{A}_2$ is added to the *PM* group, and the average difference, $\overline{A_{PM}}$, between the *PM* module scores is calculated. Subsequently, this average is compared to the difference between $\hat{A}_2$ and the next lower score, $\hat{A}_3$. If this difference surpasses the computed average, all modules outside the *PM* group are rejected. These steps are repeated until all scores in $\hat{\mathbf{A}}$ are analyzed. As shown in Figure 6.4, the WSA status after sequence iterations 2 and 3 reveals that the high scores of the *PM* modules constituted a cluster of similarly high values. Consequently, the objective of Algorithm 4 was to determine this cluster's boundary and remove low-score outliers.

The objective of lines 6 and 10 in Algorithm 4 is to identify the boundary between the two groups, as illustrated in Figure 6.4. Line 6 detects a clear winning highest-score module characterized by an extremely large difference between the first and second largest scores (see the red line after iteration 4 in Figure 6.4). If a clear winner cannot be identified, line 10 detects a significant step in the average scores between the promising and likely irrelevant module groups (see the red line after iterations 2 and 3 in Figure 6.4). When the condition in line 12 is met, the weaker module is rejected, and the high-score module's class is the result of identification. In Figure 6.4, this condition was simultaneously satisfied with code line 6. A problematic outcome can arise when an unfavorable constellations of stochastic factors precludes a clear separation between the groups, preventing boundary detection and module rejection before the next sequence iteration. Along this line, Algorithm 4 could be further enhanced for future applications.

While rejecting multiple modules simultaneously accelerates recognition, it risks discarding relevant modules that have not yet accumulated sufficient scores. Alternatively, rejecting a single module per sequence iteration offers more time for relevant modules to accumulate scores, thereby mitigating the risk of accidental rejection. The basic assumption behind single-module rejection is that it is improbable for any relevant module to have the lowest winner score and thus be rejected. However, single-module rejection is relatively slow, as eliminating all irrelevant modules requires more sequence

Algorithm 4 Module rejection($\mathbf{A} = \{A_1, \ldots, A_M\}$)

Obtain $\hat{\mathbf{A}}$ by sorting of $\mathbf{A}$ ▷ sort WSA in descending order
Initialize PM with the module index of $\hat{A}_1$ ▷ PM is the group of promising modules
for $i = 1$ to M **do**
    Compute $\overline{A_{PM}}$ ▷ $\overline{A_{PM}}$ is average difference between scores of PM modules
    if i = 1 **then** ▷ progress along the sequence
        if $\hat{A}_1 > 3 \cdot \hat{A}_2$ **then** ▷ Module of $\hat{A}_1$ is a clear winner
            Reject modules ▷ Keep only the module of $\hat{A}_1$
        end if
    else
        if $\overline{A_{PM}} < (\hat{A}_{i-1} - \hat{A}_i)$ **then**
            Reject modules ▷ Reject all modules not in PM
        else if only 2 modules left **then**
            Reject modules ▷ Reject the small-score module
        else
            Add module of $\hat{A}_i$ to PM
        end if
    end if
    i =i + 1
end for
Forward modules of PM to the next sequence iteration

iterations, significantly slowing module identification. To balance the risk of module loss and identification speed, the appropriate rejection approach was selected based on the application: single-module rejection was used for multi-class sequence experiments, while multi-module rejection was employed for recognition on the single-class sequences.

Alternative error-based symbol recognition

As an alternative to winner-score-based decision-making, this thesis explored the feasibility of distinguishing between relevant and irrelevant modules by analyzing the test error level after teacher-forcing the available ESN modules (see Appendix A). The objective of this approach was to teacher-force each available module with the given symbol sequence, inducing reservoir dynamics suitable for modeling the sequence's future flow. Subsequently, the achieved modeling accuracy was compared across the ESN modules. The module with the lowest error was selected, and its class then assigned as the class of the given symbol sequence.

Despite its apparent simplicity, this approach revealed several deficiencies during experimental investigations. Table 6.4 and Table 6.5 present the error confusion matrix and error variances obtained using the error-based method. Ideally, only the main diagonal elements should possess the lowest values. However, as shown in Table 6.4,

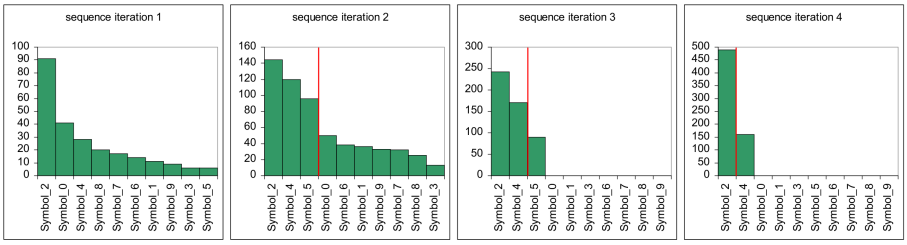

Figure 6.4.: Evolution of the WSA status over sequence iterations for the symbol "2". Module relevance could not be determined after iteration 1. The red line after iterations 2, 3, and 4 denotes the boundary between promising modules (to the left from the red line) and likely irrelevant modules (to the right from the red line).

comparably low values also exist outside the main diagonal. Furthermore, Table 6.5 shows that a significant portion of the error variances is relatively high. Their combination with the relatively low errors suggests a substantial risk of confusion, indicating the method's limited ability to discriminate between relevant and irrelevant modules. The small differences between diagonal and non-diagonal elements in the error confusion matrix imply that some ESN modules can reproduce both own-class and other-class sequences with similar accuracy. For instance, the module of symbol "1" exhibits a low error on the sequences of symbol "4," which features a similar vertical line. Although Table 6.4 displays the lowest values on the main diagonal, their difference from most other values is less than one order of magnitude.

The obtained experimental data indicated that error-based symbol recognition is unreliable, unsuitable for dynamic combinations of multiple classes, and highly dependent on the properties of the applied sequence. Furthermore, implementing this method would require an additional algorithmic effort to determine unknown parameters such as the symbol's orientation angle and size for recognition.

6.3.3. Termination criterion

As previously described, synchronization terminates at the end of the synchronization sequence. This approach is generally applicable to both oscillatory and aperiodic dynamics, ensuring high model accuracy and reliable data content identification once the algorithm reaches a steady state on a sufficiently long sequence. To achieve this, the sequence length should be regarded a critical parameter that must be selected based on the total number of modules, their size, and the information content within the sliding window. The sequence length must be extended with a higher initial module count or larger module size. Similarly, time series with low information content, i.e., low entropy [Bis09], provide limited information for model tuning and thus also require longer sequences.

Table 6.4.: Error confusion matrix of the average test RMSE for symbol-specific ESN modules. Each matrix element was computed from 10 test sequences per symbol class. The main diagonal elements are highlighted in gray, and classes with a high risk of confusion are marked in magenta.

		Sequence classes									
		"0"	**"1"**	**"2"**	**"3"**	**"4"**	**"5"**	**"6"**	**"7"**	**"8"**	**"9"**
Module classes	**"0"**	0.042	0.105	0.080	0.244	0.201	0.253	0.142	0.163	0.207	0.130
	"1"	0.160	0.058	0.153	0.229	0.181	0.296	0.231	0.101	0.136	0.204
	"2"	0.293	0.245	0.118	1.574	1.164	2.641	1.750	0.678	0.363	2.593
	"3"	0.324	0.177	0.172	0.124	0.296	0.205	0.304	0.194	0.319	0.338
	"4"	0.774	0.020	0.386	5.356	0.034	0.165	0.117	0.049	4.869	8.411
	"5"	0.254	0.110	0.140	0.253	0.248	0.113	0.225	0.191	0.225	0.303
	"6"	0.155	0.112	0.087	0.206	0.200	0.217	0.096	0.148	0.206	0.220
	"7"	0.730	0.095	0.195	0.530	0.611	0.408	0.457	0.131	0.741	0.715
	"8"	0.257	0.215	0.214	0.277	0.285	0.259	0.218	0.230	0.156	0.190
	"9"	0.153	0.172	0.136	0.291	0.235	0.299	0.898	0.206	0.283	0.097

Selecting an appropriate synchronization sequence length is typically constrained by the total available sequence length and the necessity to reserve a portion for model testing. This poses no issue for stationary time series, where characteristics remain constant throughout the sequence, allowing arbitrary segmentation into synchronization and test sequences. However, it becomes critical for non-stationary time series. As demonstrated by experiments on the non-stationary MSO, the synchronization sequence length is limited by the interval between consecutive parameter switches in the time series. Since the synchronized model must be tested on data with identical dynamic properties, the synchronization sequence length directly affects the test sequence length: extending the synchronization sequence shortens the subsequent test sequence interval.

The module synchronization problem for two-dimensional symbol curves, constrained by their limited sequence lengths, resembles that of one-dimensional non-stationary time series. However, processing short symbol curve sequences is significantly more challenging because of several factors. Firstly, the increased number of modules is a major consideration: symbol recognition requires ten modules compared to the eight in the most complex one-dimensional time series, MSO8. These larger ensembles require longer sequences for reliable module separation and reduced class confusion. Secondly, the size of symbol modules can be up to twice that of oscillator modules for one-dimensional dynamics. This results in an expanded search space and demands longer sequences for precise module state tuning. And the third factor is the low information content of symbol sequences, which consist of intervals with minimal data variation. Consequently, the presence of these factors requires the complete available sequence, thereby precluding the reservation of any data for testing. Even when utilizing the entire sequence for synchronization, preliminary experiments have demonstrated that achieving accurate ensemble synchronization in a single pass is infeasible.

Table 6.5.: Variance of the test RMSE for symbol-specific ESN modules across symbol-specific test sequences. The main diagonal elements are highlighted in gray, and classes with a high risk of confusion are marked in magenta.

		Sequence classes									
		"0"	**"1"**	**"2"**	**"3"**	**"4"**	**"5"**	**"6"**	**"7"**	**"8"**	**"9"**
Module classes	**"0"**	0.016	0.040	0.035	0.070	0.044	0.079	0.011	0.055	0.034	0.043
	"1"	0.036	0.057	0.036	0.041	0.025	0.100	0.020	0.085	0.033	0.039
	"2"	0.052	0.119	0.086	2.874	2.346	3.747	2.799	1.558	0.088	3.486
	"3"	0.024	0.077	0.054	0.079	0.013	0.113	0.055	0.038	0.015	0.048
	"4"	1.752	0.005	0.346	4.549	0.019	0.104	0.054	0.016	4.319	8.200
	"5"	0.042	0.021	0.054	0.051	0.037	0.088	0.025	0.046	0.030	0.065
	"6"	0.030	0.038	0.017	0.018	0.016	0.027	0.058	0.030	0.024	0.016
	"7"	0.055	0.043	0.050	0.076	0.035	0.109	0.121	0.116	0.063	0.036
	"8"	0.041	0.051	0.086	0.015	0.013	0.042	0.016	0.024	0.040	0.020
	"9"	0.064	0.071	0.052	0.100	0.064	0.086	2.185	0.068	0.060	0.056

Consequently, the algorithm must process the same sequence iteratively to achieve a reliable symbol class decision. During each iteration, the algorithm updates the class probability estimates within the WSA. This iterative accumulation of information mitigates the uncertainty arising from the stochastic nature of the evolutionary search, thus stabilizing the bin ordering in the WSA. Following each iteration, the algorithm assesses the maxima within the accumulator and terminates if their order remains unchanged from the preceding iteration. The bins with the highest accumulated values indicate the classes most likely present in the given symbol curve sequence. To prevent possible algorithm oscillations, termination occurs after ten iterations, regardless of WSA stability. (The value “ten” corresponds to the number of available modules.)

6.4. Design of aperiodic recurrent modules

Beyond algorithmic considerations, the properties of the modules are crucial for the recognition abilities of synchronization. As observed on the continuous oscillatory time series, the synchronization behavior varies significantly depending on whether neural or parametric modules are employed. Similarly, variations in the properties of the aperiodic modules substantially impact the complexity of the objective function $e_{T_W}(t_c)$, and consequently, the performance of symbol recognition. Overly large and overfitted modules result in high-dimensional objective functions with numerous local optima, thereby increasing the computational effort required to find a suitable solution. Conversely, properly designed compact modules maintain a flat objective function landscape with fewer local optima, enabling the efficient determination of indisputable solutions with reduced computational effort.

The limited length of the discontinuous sequences poses a significant challenge for module

design, which encompasses both the structural selection and the training of dynamic modules. Balancing compactness and power in these modules is essential, demanding a precise trade-off between module size and modeling accuracy. Furthermore, training the recurrent neural modules on such short, discontinuous sequences is problematic. Allocating a portion of the symbol's curve sequence to establish the initial transient of the randomly initialized neural network, while reserving another portion for testing the trained network, severely reduces the available training data, making it particularly critical for larger networks.

Building upon previous studies [KLB12; RI09], compact ESN models can be trained to achieve high accuracy on complex, oscillatory time series. The ability to adjust their STM, as defined in [Jae02], by modifying parameters without increasing size, makes them particularly attractive as potential modules for synchronization. This thesis proposes a methodology for designing ESN modules with compact structures, selecting their parameters, and effectively training them on short, non-oscillatory sequences.

6.4.1. Augmentation of discontinuous sequences

The development of suitable compact dynamic modules is challenged not only by the high non-linearity inherent in symbol curves but also by the non-symmetry and non-closure of the contour lines found in most drawn symbols. These characteristics significantly influence the parameters of the resulting ESN modules. Notably, symbols "0" and "8" are exceptions, as their symmetric closed curves can be relatively easily modeled using compact neural modules composed of a few sigmoid neurons with symmetric TANH activation functions. However, modeling the non-symmetric curves of other symbols necessitates non-symmetric signals within the dynamic reservoir. This asymmetry is introduced by incorporating neurons that operate in the saturation region of the TANH function. While their inclusion effectively "distorts" the symmetry of other neurons as required, it inevitably increases the module size. Therefore, careful design of the neural network structure is crucial to avoid an undue enlargement of the ESN.

The non-closure of drawn contour lines presents another challenge to the design complexity. This raises the question of how to reconcile the discontinuity of symbol sequence data with the continuous recurrent dynamics within the dynamic reservoir. This issue is relevant for both symbol curve generation and neural module training. Although a finite-length curve must be derived from the module's continuous dynamics, this qualitative difference does not imply that the module's structure requires a time-limited operational mode for drawing the curve from the given start point to the end point. In the absence of anchoring the curve at a specific starting point, the drawn curve will generally deviate from the given curve. Although this discrepancy indirectly affects recognition, the underlying statistical approach does not require a perfect match between generated and target curves because it relies solely on the ordering of module states within the population for updating the class statistics. Consequently, the magnitude of this discrepancy has no direct impact and only a minor, diminishing negative effect

Figure 6.5.: Examples of training sequences for symbol classes "2", "4", "5", "6", and "7". Augmenting data (orange pixels) were appended to the original curves (black pixels) to form closed-loop curves with a smooth transition between the original and augmented segments.

on recognition. This effect further attenuates as the discrepancy is minimized over the synchronization sequence.

Besides, the discontinuity and relatively short length of the available symbol sequences pose significant challenges for training neural modules, necessitating a suitable training approach to ensure the viability of synchronization on discontinuous sequence data. This thesis proposes and investigates the augmentation of symbol sequences for training recurrent modules. The basic idea is to artificially complete the available data to create closed-loop curves, as illustrated in Figure 6.5. The shape of the augmenting sequence interval is arbitrary, provided it is significantly shorter than the original curve's length and ensures a smooth transition between the original and augmented data. Serial replication of the resulting closed-loop curves generates a pseudo-continuous sequence from the original discontinuous sequence data. This pseudo-continuous sequence then represents periodic dynamics, which enables straightforward training. Its period corresponded to the number of time steps in the closed-loop symbol curve.

A smooth transition between the original sequence and the augmented data is crucial for designing compact ESN modules capable of achieving high modeling accuracy on symbol curves. The resulting smooth, closed-loop curves, devoid of abrupt signal flow changes, exhibit relatively low information content, making compact ESNs sufficient for precise modeling. The efficacy of compact ESNs in modeling smooth signals was demonstrated in the aforementioned experiments with neural oscillators, where sinusoidal oscillation models required fewer neurons compared to triangular signal models, which featured sharp signal flow changes. Considering that irregular and highly asymmetrical symbol curves pose even greater challenges, demanding more complex ESNs with increased neuron counts than triangular signals, smoothing the training curves becomes particularly relevant for enhancing compactness of the designed ESN modules and reducing the modeling errors. Practically, the original sequence and augmentation segment can be smoothly joined using spline smoothing [Pol99], i.e., with a curve slightly bent at connection points to avoid creases. This ensures the resulting curve's smoothness, enabling the trained module to reproduce the original curve with minimal error, including at the start and end points.

Training the ESN modules on short sequences necessitates accounting for the portion of

the sequence used for washout. This effectively diminishes the available training data. Augmenting the original symbol curve to form a closed-loop curve mitigates this issue. The augmented data itself is excluded from both training and performance evaluation. Its role is to facilitate continuous teacher-forcing of internal network states, essential for training. Thereat, a smooth junction between the augmented data and the original curve restricts the variation of reservoir states between consecutive time steps. This results in smoothed internal dynamics, allowing reservoir states to be free of initial random transience and usable for training directly at the start of the symbol curve.

Augmentation reveals its advantages during the cyclic processing of the generated closed-loop sequence, as it enables a seamless transition of the module dynamics from the end to the start of the symbol curve, as depicted in Algorithm 5. Notably, the transition from iteration 1 to iteration 2 makes the reservoir states at the first T_{wash} time steps suitable for training. Because of random initialization, the initial reservoir states are unsuitable for training and require washout at the beginning of sequence iteration 1. Teacher forcing washes out the initial randomness of the reservoir states, allowing the reservoir dynamics to converge towards a dynamic attractor. This teacher forcing continues throughout the augmented closed-loop sequence, evolving the reservoir states until the start of iteration 2, where they are no longer influenced by initial randomness. Continued teacher forcing maintains the module states aligned with the sequence, making the entire symbol curve sequence available for training. During iterations 1 and 2, the reservoir state vectors are collected in matrix $\mathbf{X}$ for subsequent computation of trainable weights $\mathbf{W}_{OUT}$. As shown in Algorithm 5, the augmentation interval is used solely for continuous teacher forcing and is irrelevant for both training and modeling accuracy evaluation during iteration 3. In iteration 3, the discrepancy $\mathbf{o}(t) - \mathbf{y}(t)$ is stored in matrix $\mathbf{\Delta}$ to assess the ESN's ability to memorize the learned symbol curve. However, not all generated ESNs can effectively memorize non-linear symbol curves due to their limited dynamic memory and possible deficiencies in the recurrent structure.

The flexibility in the augmenting sequence interval's shape promotes compactness in the designed ESN modules by eliminating the need for additional neurons, which might be required for the periodic reproduction of potentially complex resulting curves. In turn, the inherent limitation of the compact modules' dynamic memory minimizes the implicit delayed impact of the augmented data via the internal neuron states, thereby facilitating accurate replication of the original symbol curve. Disregarding the augmentation interval during training leads to output variations among different ESNs within that interval, resulting in a notable laxity effect in the output dynamics as they transition from the end to the start of the symbol curve through different paths. This laxity mirrors the varied pen trajectories human writers might use when repeatedly drawing a symbol, lifting the pen between the symbol's end and start.

Algorithm 5 Processing of ESN on augmented sequence $\mathbf{Y}$

```
Initialize X = ∅                          ▷ empty array of module states for training
Initialize Δ = ∅                          ▷ empty array of discrepancies between o(t) and y(t)
for i = 1 to 3 do                         ▷ do 3 sequence iterations: washout, train, test
    for t = 1 to T_Y do                   ▷ progress along the sequence Y
        if i = 1 or i = 2 then            ▷ iteration 1 and 2 for washout and training
            Set o(t) to y(t)              ▷ teacher-forcing of ESN
            Update ESN                    ▷ advance x(t) of ESN by 1 time step
            if (i = 2 or t > T_wash) and t ∉ augmeting interval then
                Add x(t) to X             ▷ store ESN state for training
            end if
        else
            Add o(t) − y(t) to Δ          ▷ store the discrepancy for ESN evaluation
        end if
    end for
    if i = 2 then                         ▷ after iteration 2
        Train ESN using X and Y
    else if i = 3 then                    ▷ after iteration 3
        Evaluate ESN by using Δ
    end if
end for
```

6.4.2. Generation of the module structure

Owing to the required compactness of the ESN modules, no sophisticated optimization of the module structure is necessary to achieve sufficient modeling accuracy. In the conducted investigations, promising ESN candidates were readily obtained through simple random generation of 1000 small-sized neural networks, which consistently yielded at least a few ESNs with sufficiently high modeling accuracy. Their limited size generally offered no opportunity for additional error reduction. Instead of pursuing this, it proved more beneficial for recognition performance to design a module structure that balanced compactness with the flexibility to maintain consistent accuracy across a wide range of symbol curve variations.

This thesis introduces the concept of modulating the desired symbol curve shape by varying the trainable read-out weights $\mathbf{W}_{OUT}$, wherein the symbol's basic shape is encoded by the dynamic reservoir's structure. Retraining the read-out weights $\mathbf{W}_{OUT}$ enables the ESN module to be reused for modeling variations of the symbol shape. Consequently, the ESN structure design aimed to develop a module comprising a reusable dynamic reservoir and a retrainable read-out layer, allowing the ESN module to reproduce any symbol curve variation within a predefined range with satisfactory modeling accuracy. This latter condition suggested the test error $e_{test}(\eta)$ as an accuracy evaluation metric across candidate ESNs. The selected candidate η^* was the one exhibiting

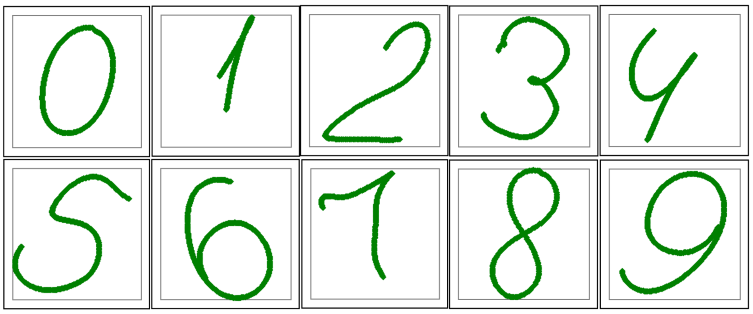

Figure 6.6.: Examples of symbol curves generated by the ESN modules for all considered symbol classes, from "0" to "9".

the highest modeling accuracy across the entire predefined range of symbol curve variations. However, evaluating the ESN on sequences for each orientation angle α generated numerous test errors, requiring aggregation for inter-candidate comparison. To avoid overlooking ESNs with poor performance at certain angles α, because of averaging the test errors, the min-max criterion was applied across available test errors to select a more reliable candidate. Specifically, the chosen ESN candidate η^* should possess a test error satisfying the following criterion.

$$e_{test}(\eta^*) = \min_{\eta \in H}(\max_{\alpha \in A}(e_{test,\,\alpha}(\eta))) \tag{6.2}$$

where η^* is the selected ESN candidate from the set of generated ESN candidates H, and α denotes the orientation angle within the considered angle range A. This criterion imposes a constraint on the maximum anticipated error, assuming that if the worst observed accuracy $e_{test}(\eta^*)$ is acceptable, then the module can reliably model all other variations of the symbol curve with satisfactory accuracy. Figure 6.6 shows the output of the chosen ESN modules, providing insight into their modeling quality for each considered symbol class.

At this juncture, it is relevant to address the method of encoding point coordinates in two-dimensional sequences. Experimentation with various encoding schemes revealed that absolute X and Y coordinates were more appropriate for symbol modeling, yielding significantly higher symbol reproduction quality compared to relative coordinates (compare the reproduced symbol curves in Figure 6.6 and Figure 6.7). The deficiency of encoding with relative X and Y coordinates lies in their inherent imprecision, which accumulates over the sequence, resulting in a substantial deviation of the drawn symbol from the desired curve by the sequence's end. While increasing the module size from several to dozens of neurons improved modeling accuracy to some extent, it led to prohibitively long synchronization times. Furthermore, the enhanced plasticity of larger modules compromised their discrimination capability, ultimately degrading recognition performance.

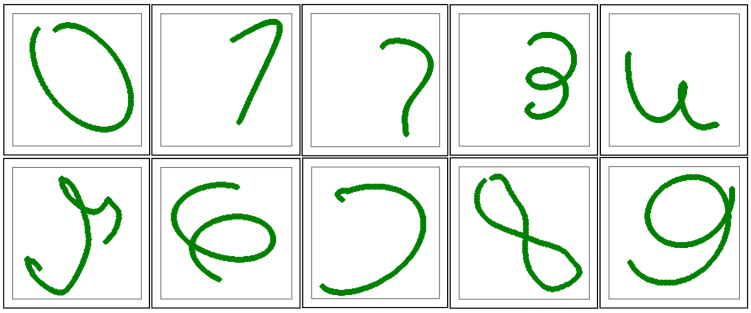

Figure 6.7.: Examples of the best symbol curves generated by the ESN modules, which were trained on sequences with relative X and Y coordinates of the points on the symbol curves.

6.4.3. Relevant parameters for the design

The conducted investigations provided evidence of the dependency between module properties and recognition performance. They also identified several ESN parameters, including module size, SR, and leakage rate, as most relevant for module design. These parameters govern the discrimination power and STM capacity of the modules, the complexity of objective functions, and the computational effort required for synchronization. Despite their significance, suitable values for these parameters are not known in advance. This necessitates varying the ESN parameters within a suitable range to generate ESN candidates. This approach proves appropriate, given that a range of ESN parameter values typically ensures optimal performance across time series. [Jae02] emphasizes this regarding the SR of the dynamic reservoir: "*the range of optimal settings is relatively broad, so if an ESN works well with* $\alpha = 0.8$*, it can also be expected to work well with* $\alpha = 0.7$ *and with* $\alpha = 0.85$" (p.43). Consequently, suitable ESN parameter ranges must be defined prior to generating candidate ESN structures.

Among the relevant ESN parameters, module size is paramount, controlling the dimensionality of the module's state space and requiring minimization, while still ensuring sufficient accuracy in modeling the symbol curve. Given the limited number of points featuring sharp changes in drawing direction within handwritten symbol curves, even small recurrent networks with few neurons can reproduce them with notable precision. While larger networks might offer improved accuracy during module preparation, smaller networks significantly enhance synchronization efficiency and performance. Furthermore, tuning smaller modules progresses more rapidly within their lower-dimensional state spaces. Additionally, compact neural modules exhibit better generalization, thereby improving discrimination between relevant and irrelevant classes. In contrast, the high plasticity of larger neural networks enables them to reproduce the curves of different symbols with comparable accuracy, increasing the risk of class confusion.

To identify compact yet sufficiently accurate neural networks, a constructive algorithm for the automated design of appropriately sized networks is employed. Similar to the

approaches in studies [YC12; Wan+15], this algorithm incrementally builds the network by adding reservoir neurons one at a time, while monitoring the achieved modeling accuracy. The process terminates when the most accurate candidate network attains a predefined minimum accuracy level, which must be established prior to initiating ESN candidate generation. If none of the generated ESNs satisfies this condition, the size is incremented by one neuron, and a new set H of candidate ESNs is generated.

$$e_{test}(\eta^*) < e_{test}^{min} \tag{6.3}$$

where $e_{test}(\eta^*)$ is the test error of the most accurate ESN candidate, and e_{test}^{min} is the maximum tolerated test error, determined empirically. On the one hand, an excessively high error threshold e_{test}^{min} fails to compel the module to model the class-specific attributes of the symbol curves, thereby increasing class confusion. On the other hand, an overly stringent error threshold e_{test}^{min} promotes larger modules with weak discrimination power, also leading to higher confusion. Thus, the accuracy threshold governs the discrimination power of the resulting modules. The smallest neural network that meets the predefined limit e_{test}^{min} offers the optimal balance between size and accuracy, and should be selected as the module for the currently considered symbol. In preliminary profiling, an optimal error threshold e_{test}^{min} of 10^{-4} was observed to be appropriate.

Standard ESNs employ conventional sigmoid neurons, updated according to the rule in (A.2). Combining the previous neuron state $\mathbf{x}(t-1)$ with the input signal $\mathbf{u}(t)$ and the output feedback signal $\mathbf{o}(t-1)$, followed by applying the sigmoid activation $\mathbf{f}(.)$, enables only fractional and indirect information propagation from the past ([Jae01]). This imposes a significant limitation on reservoir dynamics memorization, making sigmoid neurons unsuitable for modeling very slow dynamics; as noted in [Jae02]: *"it is, for instance, almost impossible with standard sigmoid networks to obtain ESN generators of very slow sinewaves"* (p.39). Given the very slow evolution of symbol curves, leaky-integrator neurons are the natural choice (see Appendix A). The module state update via Eq. (A.3) preserves a portion of the previous module state with an additive term, without applying nonlinear activation. This extends the STM of the dynamic reservoir, thus, reaching the necessary memory capacity without increasing module size. Consequently, utilizing leaky-integrator neurons promotes ESN module compactness. However, despite general recommendations, no definitive rule exists for selecting the appropriate leakage rate; the selection of a suitable parameter value remains a matter of empirical investigation.

Given the differences in the dynamic characteristics of the considered symbol curves, the required leakage rates may also differ. Generating candidate ESNs with diverse leakage rates and evaluating their performance provides a practical approach to determine the appropriate leakage rate, unique to each symbol curve. In the conducted experiments, ESN candidates were generated with leakage rates in a broad range, from a very low 0.1 (longer memory) to 1.0 (full leakage). The latter corresponds to the standard sigmoid neuron, which retains no information from the previous time step. The leakage rates and sizes of the resulting ESN modules are detailed in Table 6.6. As observed, symbols "0", "1", and "3" are effectively modeled by sigmoid neurons without leakage. Symbols

"1", "2", "4", and "7" exhibit lengthy lines in their drawn patterns, necessitating longer memory. The reduced leakage rate successfully compensates for the short memory of small-sized modules "6" and "9", which share identical parameter values due to the similarity of their symbol curves.

Table 6.6.: ESN parameters employed in the experiments with the recognition of handwritten symbols. The modules for symbols "1", "2", "4", and "7", which exhibit lengthy lines in their drawn patterns, necessitate longer STM. Larger module sizes, smaller leakage rates, and higher SR values support increased memory capacity. Conventional memoryless sigmoid neurons proved sufficient for modeling the curves of "0" and "8".

Module	**"0"**	**"1"**	**"2"**	**"3"**	**"4"**	**"5"**	**"6"**	**"7"**	**"8"**	**"9"**
Size	5	5	10	9	9	9	5	9	9	5
Leakage rate	1.0	0.2	0.5	1.0	0.4	0.7	0.2	0.3	1.0	0.2
Spectral radius	0.9	0.9	0.85	0.95	0.8	0.9	0.99	0.9	0.95	0.99

According to [Jae02], a large SR extends the STM of an ESN, which is essential for modeling the slow dynamics of drawn symbol curves. This effect mirrors that of small leakage rates. Unlike the partial retention of module states in leaky-integrator neurons, the memory effect from a larger SR stems from the slower decay of reservoir dynamics among reservoir neurons. Increasing the SR value is beneficial for module compactness, as it allows for STM extension without increasing module size. Given that the optimal SR value is typically unknown in advance, candidate ESNs must be generated with varying SR values approaching the upper limit of "1", which ensures the crucial echo-state property (see Appendix A). In the experiments, candidate ESNs were generated with SR values from the set $\{0.80, 0.85, 0.90, 0.95, 0.99\}$. As evidenced in Table 6.6, modules with SR values between 0.9 and 0.99 demonstrated optimal performance for the majority of symbol curves.

6.4.4. Regularization of the ESN modules

While high modeling accuracy is crucial for the designed module, it is not the sole criterion for evaluating generated candidate ESNs. Experimental observations revealed that deploying highly accurate modules did not consistently yield high recognition rates. Besides, recognition performance significantly improved when employing somewhat less accurate modules of the same size. A more in-depth analysis of the relationship between module structure and recognition performance discovered another influential factor for synchronization: the norms of the trainable output weights $\mathbf{W}_{OUT}$ of the ESN modules. The most accurate ESNs included networks with $\mathbf{W}_{OUT}$ norms as large as 10^6, which reproduce the learned time series with minimal error only when the

reservoir dynamics are optimal, corresponding to the module states $\mathbf{x}^*(t)$. According to Eq. (A.1), any deviation of the actual module states $\mathbf{x}(t)$ from the optimum $\mathbf{x}^*(t)$ is strongly amplified by such large $\mathbf{W}_{OUT}$ norms. Since the unknown $\mathbf{x}^*(t)$ must be determined during synchronization, the updated module states $\mathbf{x}(t)$ will inevitably deviate from optimality, resulting in a rapid deterioration of modeling accuracy due to the substantial values of $e_{T_W}(t_c)$ (see Eq. (4.5)), even for minor deviations of $\mathbf{x}(t)$ from $\mathbf{x}^*(t)$. This non-optimality of $\mathbf{x}(t)$ results in a significant number of updated module states being rejected, both in terms of module output $\mathbf{o}_m(t)$ validity and modeling accuracy $e_{T_W}(t_c)$, thereby severely affecting synchronization convergence. Consequently, these ESN module ensembles fail to score for the correct class in the winner score accumulator, leading to misclassification of the given symbol curve.

Reduction of the trainable output weights $\mathbf{W}_{OUT}$ is achieved through their regularization using ridge regression, also known as Tikhonov regularization. It enforces a reduction in the $\mathbf{W}_{OUT}$ norm without significantly compromising modeling accuracy. The benefits of this technique over the standard ESN training rule, as well as the rationale for employing the l_2 norm instead of the l_1 norm, have been highlighted in [Yan+18]. It asserts that *"the use of Tikhonov regularization could achieve better prediction accuracy than the traditional least square methods by making a bias–variance trade-off"* (p.3) and emphasizes the advantage of the differentiable l_2 norm for the choice of the regularization parameter. As presented in [WSS08]: *"Tikhonov regression is based on the idea of adding an additional cost term to the least squares optimization so that the norm of the parameters is also kept small."* (p.2). The predefined cost term in the training rule effectively makes the training procedure less sensitive to fine details in the sequence data, thereby enhancing the generalization capabilities of the resulting neural network. This proves advantageous for recognition performance, as minor variations in symbol curves of the same class do not impact the tuned module's modeling accuracy and, consequently, do not affect the update of class statistics in WSA.

Figure 6.6 shows the average error gradient behavior near the optimal states of both regularized and non-regularized ESN modules for the symbol "6" over the synchronization sequence. (To mitigate a bias from the sequence type, the gradient was averaged over 100 sequences encompassing all symbols.) As depicted in this figure, minor variations in module states led to a substantial increase in the error signal, spanning several orders of magnitude. The error gradients of the non-regularized ESN either remained consistently high throughout the synchronization sequence (module state 2) or sharply increased as spikes. In contrast, the regularized ESN modules displayed only minimal changes in the error signal after variations in the module states. This demonstrates the advantage of regularization in flattening the objective function, effectively reducing the risk of premature convergence.

The effectiveness of Tikhonov regression in preventing overfitting of trained networks was further validated by the promising results of the study [Pat+18b], which combined a dynamic reservoir with a knowledge-based module. The findings in [Yan+18] demonstrated that employing Tikhonov regression promotes the desired compactness of ESNs, enabling them to outperform other ESN models, such as those in [JH04; Ott+16]. The

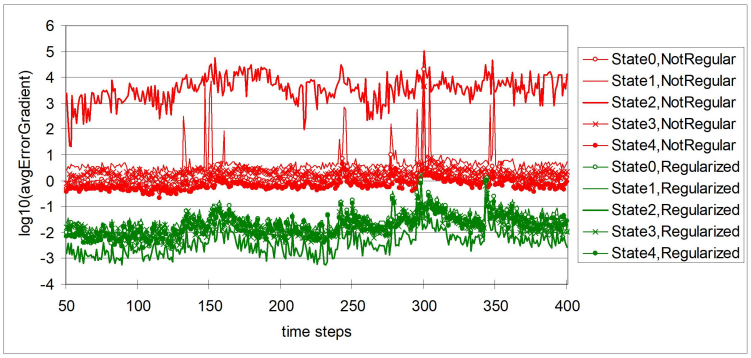

Figure 6.8.: Average error gradient of non-regularized and regularized ESN modules for the symbol "6" during synchronization. The gradient was calculated as the error change following a small variation (gradient step of 10^{-2}) of the respective module state near the optimum. The impact of regularization varied across module states, depending on their contribution to the module output. Averaging over 100 sequences of diverse symbols mitigated any potential bias from sequence type.

derivation of formulas in [Yan+18] resulted in the following training rule for regularized ESNs.

$$\mathbf{W}_{OUT} = (C\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{T}, \tag{6.4}$$

where C is the regularization parameter ($C > 0$), $\mathbf{X}$ is the matrix of module states obtained from Algorithm 5, $\mathbf{I}$ is the identity matrix of the same dimensionality as $\mathbf{X}^T\mathbf{X}$, and $\mathbf{T}$ is the matrix of target ESN outputs from the training sequence. This formula is recognized as a special case of the training rule for standard ESNs (Eq. (A.4) in Appendix A). The selection of the regularization parameter represents a trade-off between the magnitude of $\mathbf{W}_{OUT}$ and the modeling accuracy of the trained neural network. To prevent excessive loss of modeling accuracy, the regularization parameter was set to 10^{-5}. Larger values resulted in unacceptable distortions of the drawn symbol curves.

6.4.5. Adaptive parameterization

Considering the potential variations in handwritten symbols concerning orientation and size, the modules must possess sufficient flexibility to accommodate these variations without retraining. However, the compactness of the modules limits their structural plasticity, resulting in each recurrent module primarily reproducing only a single variation of the symbol curve. This suggests either increasing module size or combining modules to enhance their modeling capability. Nevertheless, depending on the complexity of the modeled dynamics, the module size can grow significantly, potentially negating the compactness advantage of a modular network over a monolithic one and rendering the synchronization-based tuning of the entire modular network infeasible. This issue can

be addressed through adaptive parameterization of the recurrent modules.

The core principle of adaptive parameterization is to enable the activation of optimal parameter values for recurrent modules based on the provided sequence data. Let us consider this in the context of modeling handwritten symbols. As previously described, each recurrent module in the presented approach is designed to reproduce the basic form of a specific symbol class; for instance, one module for the symbol "0", another for "1", and so forth. Despite variations in symbol writing, the basic form of its curve is typically preserved. Following the principle of parameterization, these variations can be modeled through additional network parameters.

The experiments described herein addressed variations in the size and tilt of the drawn symbol curves. In the network, symbol size is explicitly defined by the scaling coefficients of the modules, specifically, their responsibilities $\mathbf{r}_m$. Tilting the drawn symbol curves requires their rotation. Unlike scaling, the network architecture depicted in Figure 3.1 does not include a parameter for this. Consequently, the modules themselves must incorporate the capability to rotate the symbol curves.

Explicit rotation of the symbol curve

Explicit rotation of the symbol curve was achieved through a non-linear transformation that mapped each point $(x,\ y)$ of the symbol curve to a corresponding point $(x',\ y')$ on the rotated curve. This transformation preserved the distance to the origin of the coordinate plane while adjusting the orientation angle α as required. Consequently, the elements of the original output vector $(\hat{o}_{m1}(t),\ \hat{o}_{m2}(t))$ of the m^{th} module were transformed into the elements of the "rotated" output vector $(o_{m1}(t),\ o_{m2}(t))$ for a given orientation angle α_m as

$$\begin{cases} o_{m1}(t) = \hat{o}_{m1}(t) \cdot cos(\alpha_m) - \hat{o}_{m2}(t) \cdot sin(\alpha_m) \\ o_{m2}(t) = \hat{o}_{m1}(t) \cdot sin(\alpha_m) + \hat{o}_{m2}(t) \cdot cos(\alpha_m) \end{cases} \tag{6.5}$$

Within the module's structure, this was implemented as an additional node positioned above the recurrent reservoir inside the module (see Figure 6.9). Given that the angle α was unknown a priori, the synchronization algorithm had to determine its value while aligning the module outputs with the synchronization sequence.

Learning parameterization of recurrent dynamics

Explicit rotation is straightforward and applicable for handwritten symbol recognition, where the basic form is known a priori and the symbol variations are limited to rotations of this basic form. However, in general, extracting the basic form from a multitude of its variations can be difficult or even impossible. Furthermore, the transformation formulas for deriving a particular variation from the basic form may be unknown. An example of such a transformation is elastic morphing, as described in [BZS09].

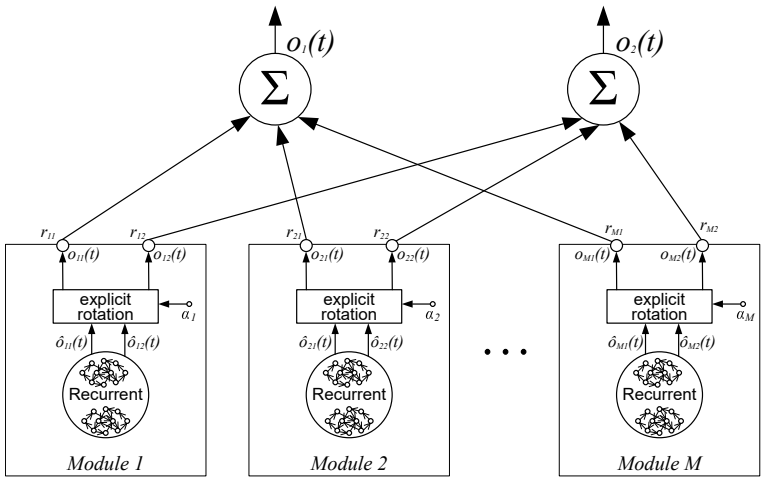

Figure 6.9.: Modular network with explicit rotation of the basic form for handwritten symbol recognition. The recurrent part of the m^{th} module generates the sequence $(\hat{o}_{m1}(t),\ \hat{o}_{m2}(t))$ representing the X- and Y-coordinates of the basic form of a symbol. It is then rotated by an orientation angle α using the formulae in (6.5) to produce the sequence $(o_{m1}(t),\ o_{m2}(t))$ at the module output. The angle α of each module must be determined during synchronization.

This thesis proposes a method for learning the parameterization of recurrent dynamics that avoids reliance on explicit formulas for transforming a basic form. Instead, the required transformation is learned from variations in the recurrent network's parameters, where the network is trained independently on each variation of the basic form. It is assumed that a subset of the recurrent network remains constant across all considered variations. In this case, successful learning of all known variations implies that the constant part of the RNN can effectively encode the *class characteristic attributes*. For instance, the constant part of the module could be the topology of the connections between the RNN neurons, which is not subject to training. Here, the class characteristic attributes represent the dynamic properties of the basic form that are common to all its variations. Simultaneously, the *specific attributes* are encoded in the learned parameters for each individual curve. A representative collection of these parameter sets provides implicit information about the transformation of the basic form and enables learning of the required transformation. This concept allows modeling each symbol using a *compositional recurrent module*, which comprises two sub-modules: a *primary recurrent* part, encoding the class characteristic attributes, and a *secondary parametrizer* sub-module, encoding the specific attributes. Figure 6.10 shows the modular network designed for handwritten symbol recognition, where each module incorporates two parameterizers – one for each output signal.

The basic idea of learning parameterization is to establish the dependency between the module output and the *transformation parameters*. In the presented architecture, the

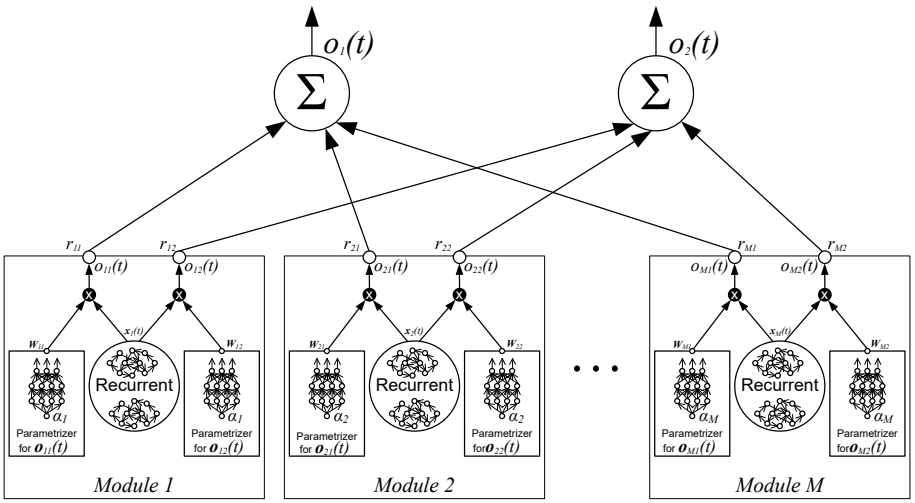

Figure 6.10.: Ensemble of compositional modules consisting of parameterizers and recurrent parts. The state $\mathbf{x}_m(t)$ of the recurrent part of the m^{th} module encodes the class characteristic attributes of the corresponding symbol. The parameterizers generate the parameter vector required for the recurrent part to draw the symbol curve at an orientation angle α_m. The outputs $o_{m1}(t)$ and $o_{m2}(t)$ represent the X- and Y-coordinates of the drawn curve's points. The values of α_m and $\mathbf{x}_m(t)$ are determined during synchronization.

transformation parameter is the orientation angle α of the tilted handwritten symbols. Variations in α alter the coordinates of the drawn symbol and influence the connection weights of the RNN, which is trained to generate the sequence of drawn points. Modeling this dependency yields the parameterizer sub-module, which receives the transformation parameter as input and outputs the necessary parameter values for the primary recurrent part. These parameter values modulate the non-linear output of the module, and their accuracy is crucial for the precision of the modulated signal. Training a sufficiently large parameterizer model on a representative, densely sampled training set improves its accuracy and limits unexpected deviations in the module output.

The combination of the recurrent part and the parameterizer sub-module resembles the Jumpnet architecture [Rue93], where a processing network (analogous to the primary recurrent part) was coupled with a control network that modulated the weights of the processing network. Another example of a learned transformation in a modular network is Feature-wise Linear Modulation ([Per+17]), which learns to influence the output of neural modules by applying an affine transformation during image processing.

Learning parameterization with ESNs

The ability to generate compact ESNs for modeling complex dynamics makes them highly suitable for applying the method of learning parameterization. Their dynamic

reservoir, including the randomly generated topology and reservoir weights $\mathbf{W}_{RES}$, encodes the class characteristic attributes, while the trainable read-out weights $\mathbf{W}_{OUT}$ encode the specific attributes. As shown in Figure 6.6, compact ESNs with only a few neurons accurately reproduced the handwritten symbols. In these experiments, the same dynamic reservoir was used to generate symbol curves within a specific class; its redesign was unnecessary, with only ESN retraining required after changes in the orientation angle. Thus, retraining yielded a distinct set of read-out weights, $\mathbf{W}_{OUT}$, for each rotated curve. Subsequently, each new $\mathbf{W}_{OUT}$ is treated as a single vector in the training sample set for the parameterizer sub-module. The small size of the $\mathbf{W}_{OUT}$ matrix promotes compactness of the parameterizer sub-module and, consequently, of the entire conjunction of the recurrent part and the parameterizers, resulting in a size significantly smaller (see Table 6.6) than that of a single universal ESN, similar to those explored in [WSS08; WS09], where ESN sizes ranged from 100 to 2000 neurons. This makes learning parameterization particularly advantageous for online synchronization.

Moreover, a simple linear dependency of the ESN output on the read-out weights $\mathbf{W}_{OUT}$ is a distinct advantage of this RNN type. This linearity reduces the non-linearity of the modeled parameterization dependency, thereby promoting compactness of the parameterizer sub-module. Indeed, the weak non-linearity can be modeled with a relatively compact model, sufficiently precisely to mitigate potential inaccuracies in the parameterizer sub-module when modulating the module's output signal. In other words, a smaller and less complex parameterizer model can be employed without significantly increasing the risk of its inaccuracies being amplified by strong non-linearities. In the conducted experiments, the weak non-linearity of the parameterization dependency was seen as a smooth function with a bounded gradient, indicating that small changes in the orientation angle α result in only minor changes in the parameterized weights $\mathbf{W}_{OUT}$.

Figure 6.11, Figure 6.12, and Figure 6.13 illustrate the dependency of the parameterized weights $\mathbf{W}_{OUT}$ on the orientation angle for the symbols "0", "1", and "9", which possess the smallest modules and, consequently, the fewest weight values depicted in the diagrams. A comparison of these figures reveals that more complex symbol curves exhibit more complex parameterization dependencies, with extrema present in some elements of the vector $\mathbf{W}_{OUT}$. The number of extrema is greater for symbol "9" than for "1", while the dependencies for symbol "0" are simply increasing or decreasing linear functions of the angle. In all cases, the smoothness of the parameterization dependency is maintained. Notably, certain elements of the vector $\mathbf{W}_{OUT}$ do not influence the transformation of the ESN module's output dynamics and remain constant across the range of the orientation angle α, as exemplified by values 2 and 5 in Figure 6.13.

The question now arises: how does one identify an appropriate dynamic reservoir for the recurrent part? Given the absence of a definitive rule for designing ESN reservoirs, the search typically involves an iterative process. In this process, high reproduction accuracy serves as an indicator of a suitable dynamic reservoir for the given sequence data. For this search, a series of ESNs were generated and trained on symbol sequences across a range of orientation angles. A dynamic reservoir that consistently yielded sufficiently

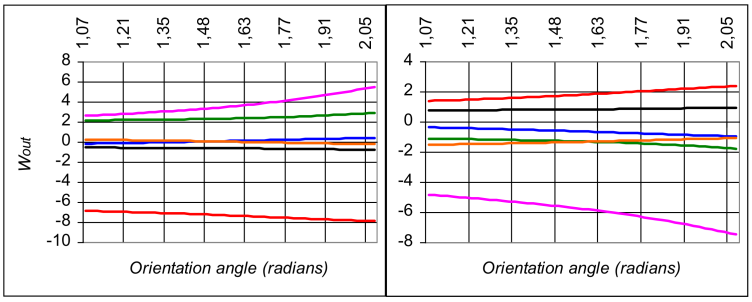

Figure 6.11.: Dependency of individual output weights ($W_{out,0}$ - black, $W_{out,1}$ - blue, $W_{out,2}$ - green, $W_{out,3}$ - orange, $W_{out,4}$ - magenta, and $W_{out,5}$ - red line) on the orientation angle α in the ESN module of the symbol "0". The $\mathbf{W}_{OUT}$ weights for $o_1(t)$ are shown on the left, and those for $o_2(t)$ on the right.

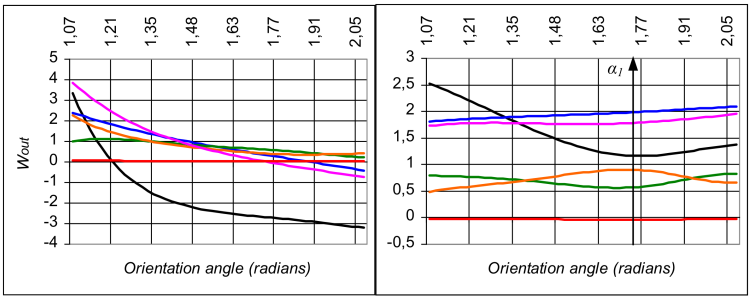

Figure 6.12.: Dependency of individual output weights ($W_{out,0}$ - black, $W_{out,1}$ - blue, $W_{out,2}$ - green, $W_{out,3}$ - orange, $W_{out,4}$ - magenta, and $W_{out,5}$ - red line) on the orientation angle α in the ESN module of the symbol "1". The $\mathbf{W}_{OUT}$ weights for $o_1(t)$ are shown on the left, and those for $o_2(t)$ on the right. The increased complexity of the symbol "1" curve results in extrema in the dependencies at a specific angle α_1.

high accuracy for all orientations of a symbol was deemed to effectively encode the class characteristic attributes of the basic form. Such a reservoir was then selected as the recurrent part.

Although there is no definitive rule for designing ESNs, theoretical and experimental findings must be considered to achieve an optimal balance between the size and accuracy of the resulting ESN modules. On one hand, larger ESN reservoirs exhibit a richer variety of recurrent dynamics and support longer short-term memory (STM), enabling precise modeling of complex time series [Jae01]. However, the computational effort makes large ESNs impractical as recurrent modules for online synchronization. Instead of increasing the reservoir size, dynamic memory can be enhanced through alternative methods, such as increasing SR of the dynamic reservoir or utilizing leaky-integrator neurons. Given that ESN size must be minimized for online synchronization, reservoirs with a very large

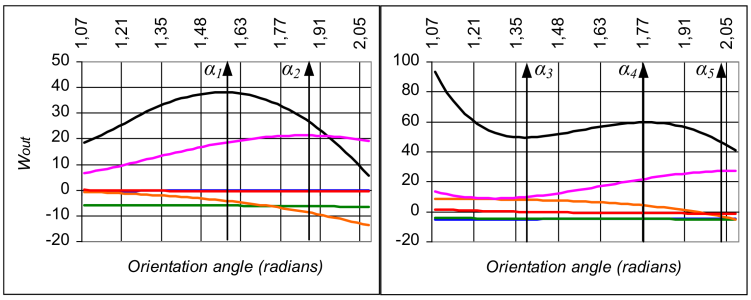

Figure 6.13.: Dependency of individual output weights ($W_{out,0}$ - black, $W_{out,1}$ - blue, $W_{out,2}$ - green, $W_{out,3}$ - orange, $W_{out,4}$ - magenta, and $W_{out,5}$ - red line) on the orientation angle α in the ESN module of the symbol "9". The $\mathbf{W}_{OUT}$ weights for $o_1(t)$ are shown on the left, and those for $o_2(t)$ on the right. The complexity of the symbol "9" curve is even greater than that for "1", resulting in extrema in the dependencies at multiple angles, from α_1 to α_5, while the smoothness of the dependency remains.

SR, close to “1”, were generated. Simultaneously, the leakage rate of leaky-integrator neurons is highly dependent on the specific time series, making it difficult to estimate appropriate values. Therefore, for each randomly generated ESN, the leakage rate was randomly selected from the interval [0.1, 1.0]; setting it to 1.0 implied full leakage, meaning the neurons did not retain their previous states at all.

6.5. Concluding summary for handling discontinuous sequence data

This chapter presented an advanced synchronization method for the recognition of handwritten symbols. These symbols are represented by relatively short sequences, posing challenges for the design of compact recurrent neural modules and their synchronization. Such discontinuous data exist only within a specific time interval and not outside it. This presents a particular challenge for training recurrent neural modules that evolve continuous recurrent dynamics within the dynamic reservoir. Moreover, the short sequence length hinders the achievement of low error levels during synchronization, and their low information content impacts the resulting identification.

The proposed solutions were presented and studied in this chapter through a series of experiments, which mostly demonstrated reliable recognition of individual handwritten symbols despite the challenging nature of the problem. It described an approach involving the augmentation of discontinuous sequences to generate pseudo-continuous sequences that contain sufficient data for training recurrent ESN modules. A smooth transition between original and augmented sequence data is important for the compactness of these modules. However, training the modules solely on these pseudo-continuous sequences

is insufficient for designing appropriate modules. Analysis of performance variations revealed their dependence on the module's $\mathbf{W}_{OUT}$ weights. Based on this dependency, reducing the norms of these weights through regularization smooths the objective function, consequently suppressing excessive rejection of the updated module states during synchronization and thereby improving performance.

A further analysis of the module structure uncovered a dependency between the trainable $\mathbf{W}_{OUT}$ weights and characteristics of the generated symbol curve. This dependency allows direct manipulation of the symbol curve's orientation by changing the $\mathbf{W}_{OUT}$ weights. This confirms the reusability of the ESN modules, allowing the same module structure to reproduce a symbol curve with varying characteristics. The existence of this dependency presents an opportunity to learn a parametrization of the ESN modules and to utilize it for online modification of symbol curves, thus avoiding the need for retraining the modules.

Furthermore, investigations revealed that selecting an appropriate encoding scheme for the symbol curves is crucial for the module design. Absolute X and Y coordinates proved to be more suitable, yielding significantly higher symbol reproduction quality with more compact modules. In contrast, the inherent imprecision of relative X and Y coordinates accumulates over the sequence, resulting in a substantial deviation of the drawn symbol from the desired curve by the sequence's end. This deficiency cannot be compensated even by a significant increase in module size, which additionally leads to prohibitively long synchronization times and ultimately causes high confusion between symbol classes.

Despite the possibility of obtaining sufficient training data through augmentation, this approach is not applicable to the synchronization sequences. Consequently, the recognition of the symbol curve must rely solely on the original data. My experiments indicated that the limited availability and low information content of the data hinder the identification of the data content through precise synchronization of the modules, unlike in the case of oscillatory time series. Instead, the relevant dynamic components are reliably identified based on statistics from a pool of low-accuracy modules. Subsequently, the parameters of these identified components can be read out from the corresponding parameters of the module structure. However, it was observed that the performance decreased with an increasing alphabet size.

Building upon the insights from this chapter, future research could explore alternative module structures for discontinuous sequence data. Such structures might enhance the accuracy of the generated symbol curves while maintaining module compactness. Furthermore, the proposed decision-making method employing the Winner Score Accumulator could be refined for more efficient handling of class statistics, particularly in cases where no clear winner is distinguishable. This enhancement should enable earlier detection of irrelevant classes, consequently reducing the number of required sequence iterations.

7. Evolutionary synchronization for handwritten symbol recognition

This chapter presents the achieved recognition performance for single- and multi-class symbol sequences. An analysis of the factors influencing recognition performance revealed its dependency on both synchronization parameters and the designed structure of the modules. Based on this dependency, the chapter outlines a method for tuning the rejection behavior to ensure accurate symbol recognition.

7.1. Recognition of single-class sequences

The brevity of the symbol curve sequences poses a challenge in evaluating the method's performance. Specifically, the low information content significantly impedes convergence, making it infeasible to divide the symbol sequences into sufficiently long segments for both synchronization and testing. This necessitates the use of the entire available sequence for synchronization, precluding the calculation of a test error – a situation analogous to experiments with non-stationary continuous oscillatory time series, where test errors were similarly dropped. However, unlike non-stationary dynamics, the synchronization error cannot be considered a reliable performance metric for symbol sequences because of their brevity relative to the problem's complexity. With the moderate-sized recurrent modules employed in these experiments, the short, low-information sequences do not allow the algorithm to fully optimize the modules into highly accurate predictors. Consequently, achieving a very low synchronization error is not expected. Nonetheless, the algorithm's ability to reliably identify dynamic components in the time series suggests the identification rate as a viable performance indicator. This rate can be further analyzed in detail using confusion matrices.

Identification rate

The complexity of the problem significantly affects IR, with complexity being strongly influenced by the number of considered classes. To understand the impact of complexity on recognition performance, experiments were conducted on single-class sequences for varying numbers of known classes, ranging from two to ten. The resulting IR dependency on the number of available modules is shown in Table 7.1, where IR was calculated using formula (5.2). As shown in this table, algorithm performance decreased with an

increasing number of modules in the ensemble; that is, a larger alphabet reduced the number of correctly recognized symbols. For alphabets with more than five symbols, IR fell below 100%. Each additional symbol increased the likelihood of encountering similarly drawn symbols; for instance, a vertical line was present in symbols "1", "4", and "7". This increased the risk of misclassification during recognition. In addition to problem complexity, the stochastic nature of EA negatively impacted recognition quality. Thus, unfavorable initialization and the progressive loss of population diversity became more critical in more complex search spaces with a greater number of available modules.

Table 7.1.: Identification rate using learned piecewise linear parameterization on single-class sequences, for varying numbers of known classes. When two classes were considered, they were "0" and "1"; when three classes, "0", "1", and "2"; and so forth.

Number of classes	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
IR (%)	100	100	100	100	96.92	94.73	87.35	83.50	82.0

Confusion matrix

The identification rate serves as a cumulative performance indicator, offering a comprehensive overview of the method's overall recognition performance across all available target classes. Table 7.1 and Table 7.5 demonstrate its applicability to the sequences with both single and multiple target classes. However, IR alone is insufficient for a detailed analysis of the algorithm's behavior on sequences of different symbols. Confusion matrices prove significantly more valuable for a granular evaluation of recognition performance than the overall classification rate. Widely utilized in statistical pattern recognition [Kor01; Sch96], confusion matrices provide insight into the algorithm's ability to discriminate between available target classes. Specifically, a confusion matrix indicates the frequency with which each class is correctly classified, as well as the frequency and nature of misclassifications. Typically, a confusion matrix is a square matrix, with rows representing recognized classes and columns representing true classes of the processed sequences. Prior to processing a batch of symbol sequences, all elements of the confusion matrix are initialized to zero. Upon processing each sequence, the algorithm increments the element at the intersection of the row corresponding to the recognized class and the column corresponding to the true class. Consequently, the elements along the main diagonal reflect the statistics of correctly recognized symbols, while off-diagonal elements quantify misclassified sequences.

Figure 7.1 and Figure 7.2 show the confusion matrices for an increasing number of classes, ranging from five (the maximum number of classes with 100% IR) to ten (the total number of digit symbols). As shown, for the smallest number of available classes, there are no non-zero values outside the main diagonal, indicating that the method

Classes		Processed sequences					
		“0“	“1“	“2"	“3“	“4"	“5“
Result of identification	“0“	**10**	0	0	0	0	0
	“1“	0	**10**	0	0	0	0
	“2“	0	0	**10**	0	0	0
	“3“	0	0	0	**10**	0	0
	“4“	0	0	0	0	**10**	0
	“5"	0	0	0	0	0	**10**

Figure 7.1.: Confusion matrix resulting from the synchronization of the ESN modules on single-class sequences with 5 classes.

accurately recognized all submitted symbols. In the ten-class configuration, however, not all submitted sequences were correctly recognized. Several isolated "1" values outside the main diagonal indicate a small proportion of misclassified sequences, attributable to the stochastic nature of the synchronization algorithm and similarities in the trajectories of the confused symbols. Conversely, this phenomenon positively impacts the classification of different trajectories that semantically belong to the same symbol, as previously observed in [Fab+20]: *“The network and inference mechanisms were furthermore able to classify different variants of a character as belonging to the same one, as long as the presented trajectory variants were closely related.”* (p.153) The increasing number of misclassified sequences demonstrates the growing negative impact of the number of available classes on convergence.

7.1.1. Adjustment of rejection behavior

The rejection behavior of the synchronization algorithm is implemented by discarding modules that achieve only low A_i values in the WSA. The algorithm parameters – the maximum number of generations N^{max} from Eq. (4.53) and the tolerance margin of the module $\Delta\mu_m$ from Eq. (4.47) – influence the accumulation of the winner score and are therefore rejection-relevant. The confusion matrix enables a class-by-class analysis of recognition performance, facilitating the empirical determination of optimal parameter values to maximize correct classifications and minimize misclassifications. Similar to any optimization process, the adjustment of rejection-relevant parameters typically begins with an arbitrary set of initial values. Experiments have demonstrated that a suitable initial guess is to set N^{max} for each module equal to the number of hidden states $\mathbf{x}_m(t)$ in that module. The tolerance margins $\Delta\mu_m$ for all modules can initially be set to zero. Using these initial parameter values, the algorithm produces a confusion matrix that requires analysis before updating the parameter values. This iterative process must be repeated until satisfactory recognition performance is achieved.

Classes		Processed sequences									
		“0“	**“1“**	**“2"**	**“3“**	**“4"**	**“5“**	**“6“**	**“7“**	**“8“**	**“9“**
Result of identification	**“0“**	**8**	0	0	0	0	0	1	0	0	2
	“1“	0	**7**	0	0	1	0	0	1	0	0
	“2“	0	0	**9**	1	0	0	0	0	0	0
	“3“	0	0	1	**9**	0	0	0	1	0	0
	“4“	0	1	0	0	**8**	1	0	0	0	0
	“5"	0	1	0	0	0	**8**	0	0	1	0
	“6"	1	0	0	0	0	0	**8**	0	0	0
	“7"	0	1	0	0	1	1	0	**8**	0	0
	“8"	0	0	0	0	0	0	0	0	**9**	0
	“9"	1	0	0	0	0	0	1	0	0	**8**

Figure 7.2.: Confusion matrix resulting from the synchronization of the ESN modules on single-class sequences with 10 classes.

The analysis of the confusion matrix aims to identify *dominant* and *recessive* classes. Dominant classes are those whose labels are most frequently indicated in cases of misclassification. In contrast, recessive classes are those whose symbols are least frequently classified correctly. For example, the dominance of classes "0" and "4" is illustrated in Figure 7.3 and Figure 7.4, respectively. Similarly, the recession of classes "2" and "1" is evident in both figures. As observed in the presented confusion matrices, the rows corresponding to dominant classes contain several non-zero values outside the main diagonal, representing the number of misclassifications of other classes. On the contrary , the columns corresponding to recessive classes contain several non-zero values, and their diagonal values are smaller than the total number of processed sequences for those classes. The remaining classes, not highlighted in yellow in Figure 7.3 and Figure 7.4, are neither dominant nor recessive. Optimal recognition performance is achieved when all known classes are neither dominant nor recessive, as shown in the confusion matrices with green-highlighted rows and columns in both figures. Such matrices contain only a few isolated, small non-zero values outside the main diagonal. If dominant or recessive classes are present, recognition performance is improved by adjusting the rejection: tightening the rejection for dominant classes and loosening it for recessive classes.

As detailed in subsection 6.3.2 Winner score accumulator, irrelevant classes are rejected after sequence processing if their modules accumulate only low winner scores. In contrast, the algorithm retains classes whose modules accurately reproduce the given symbol curve most frequently across time steps. This highlights the importance of expected accuracy for rejection, which is controlled by module-specific search effort. Increasing search effort significantly enhances the likelihood of finding more accurate solutions by optimizing relevant modules, thereby increasing their presence among the most

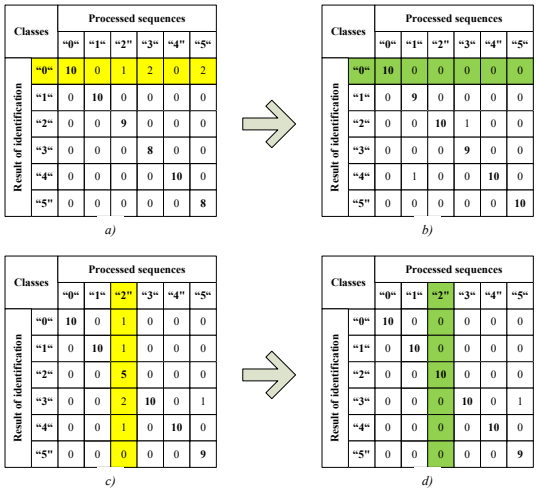

Figure 7.3.: Adjustment of rejection behavior by modifying the number of generations N^{max}. The row corresponding to the dominant class "0" is highlighted in yellow in a), and the column corresponding to the recessive class "2" is highlighted in yellow in c). Reducing N^{max} for module "1" from 5 to 4 resolves its dominance. The resulting confusion matrix, b), has no non-zero values outside the main diagonal in the row of "0" (highlighted in green). For class "2" in c), increasing N^{max} for module "2" from 5 to 6 resolves the recession of its class. The resulting confusion matrix, d), has no non-zero values outside the main diagonal in the column of class "2" (highlighted in green).

accurate ensembles counted in the WSA. Conversely, optimizing irrelevant modules has a negligible impact on the reduction of the ensemble's total error and does not significantly alter the WSA statistics.

To enhance synchronization accuracy, a greater number of trial solutions must be generated during evolutionary module tuning. When the module's class matches the class of the processed symbol sequence, a sufficiently large N^{max} results in a sustained improvement in accuracy, remaining at a low level over subsequent time steps. This sustained improvement is not observed when the module's class differs from the sequence's class. However, due to the inherent plasticity of recurrent modules and the local similarity between different symbols, a short-term error reduction can still occur even when tuning irrelevant modules. Therefore, to mitigate the risk of misclassification, excessively large values of N^{max} should be avoided.

The tolerance margin $\Delta\mu_m$ is another parameter that should be considered for adjusting the rejection behavior. This parameter is useful for tightening the rejection after a previous increase in N^{max} for a recessive class. Occasionally, a recessive class can become dominant upon increasing N^{max} just by "1" (the smallest possible increment). This necessitates a slight loosening of the rejection, which cannot be achieved solely by manipulating the

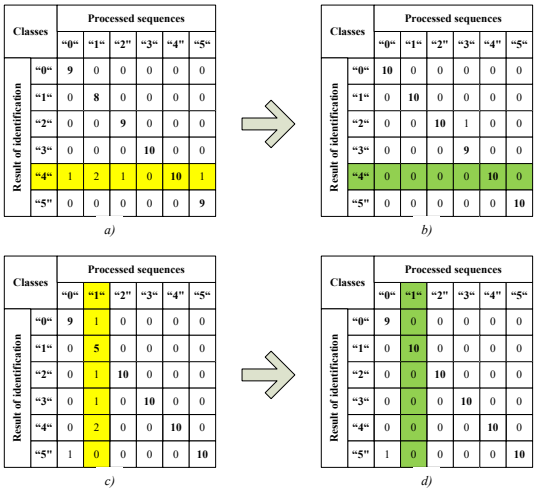

Figure 7.4.: Adjustment of the rejection behavior by modifying the tolerance margin $\Delta\mu_m$. The row for dominant class "4" is marked yellow in a), and the column for recessive class "1" is marked yellow in c). Increasing the tolerance margin of module "4" from 0.03 to 0.05 effectively resolves its dominance. The resulting confusion matrix b) shows no non-zero values outside the main diagonal for the row of class "4" (marked green). Similarly, for class "1" in c), tightening the tolerance margin of module "1" from 0.1 to 0.08 resolves recessiveness of its class. The resulting confusion matrix d) exhibits no non-zero values outside the main diagonal for the column of class "1" (marked green).

integer-valued N^{max}. Instead, a subtle tightening is accomplished by expanding the valid range of the tunable states $\mathbf{x}_m(t)$ by a sufficiently small value of the module tolerance margin $\Delta\mu_m$. This process is schematically illustrated in Figure 7.4. The lower limit is shifted towards the negative end, while the upper limit is shifted towards the positive end. Viewing the elements of the vector $\mathbf{x}_m(t)$ as dimensions of the search space, expanding the valid ranges effectively enlarges the search space. As demonstrated in the example in Figure 7.5, the lower-error region is likely to extend into the extension of the complex search space, potentially concealing local optima and making the global optimum more difficult to locate. Consequently, a portion of the population spreads into this extended area, diminishing the accuracy gains achieved from tuning the module in question. This equates to a reduction in module-specific search effort. Furthermore, comparing the diagrams in Figure 7.5 highlights that the effect of varying $\Delta\mu_m$ is highly dependent on the error landscape. This landscape depends on the dynamic characteristics of the recurrent module and is hard to know in advance. Therefore, the impact of the tolerance margin $\Delta\mu_m$ on adjusting the rejection behavior is less predictable. Consequently, $\Delta\mu_m$ should be regarded as a secondary parameter compared to N^{max}.

Ultimately, appropriate adjustment of the rejection-related parameters should yield balanced recognition of all classes, eliminating both dominance and recessiveness in the

confusion matrix.

7.1.2. Explicit rotation vs. learning parameterization

The experimental data demonstrates a significant advantage of the learning parameterization over explicit rotation (compare the substantial difference in IR from Table 7.1 and Table 7.2). The learning parameterization achieves a higher recognition rate with a greater number of available modules. This performance disparity is attributable to two key factors. Firstly, it allows adjusting the read-out weights $\mathbf{W}_{OUT}$ during synchronization. Secondly, it implicitly realizes the rotation operation through the network structure. Thereat, the module's output undergoes explicit rotation after computation, whereas learning parameterization influences the output formation itself. Consequently, in the network structure, explicit rotation appears serially after the recurrent module, and learning parameterization appears in parallel. A comparative representation of both models is provided in Figure 7.6.

The ability to adjust the read-out weights $\mathbf{W}_{OUT}$ is crucial, particularly immediately after initialization, when the recurrent module states are typically unstable. Evolutionary tuning prioritizes values within $\mathbf{W}_{OUT}$ that effectively limit the OFB signal, thereby preventing significant network errors. At this stage, tuning the orientation angle is of secondary importance. Once the module states stabilize, $\mathbf{W}_{OUT}$ is fine-tuned to align with the curve's orientation. In contrast, in the case of explicit rotation, the static $\mathbf{W}_{OUT}$ fails to match the initialized recurrent module states, frequently causing the algorithm to oscillate and preventing stable convergence to the global optimum.

The positioning of the rotation operator significantly influences the complexity of the optimized objective function. According to the explicit rotation equations (6.5), the module output is multiplied by sine and cosine functions of the orientation angle. Since these are functions of independent arguments, this results in a complex multiplicative objective function, increasing the risk of premature convergence. Conversely, learning parameterization avoids such multiplication, thereby transforming the synchronization into a less complex optimization problem, leading to substantially improved synchronization performance.

Table 7.2.: Identification rates by ESN modules achieved with explicit rotation of the symbol curve for the recognition of single-class sequences, across varying numbers of classes. Specifically, classes "0" and "1" were used for the two-class scenario, classes "0", "1", and "2" for the three-class scenario, and so forth.

Number of classes	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
IR (%)	100	71.87	55.81	33.33	24.61	23.68	18.39	16.49	12.96

7.1.3. Piecewise linear parameterization vs. FF-ANN

The potential benefits of alternative learning parameterization concepts for recognition performance were investigated by analyzing experimental data from piecewise linear parameterization and a model using FF-ANN parameterizer modules. The simplicity of the piecewise linear model is a distinct advantage. Additionally, it yielded accuracy several orders of magnitude higher in computing the primary recurrent part's parameters compared to the FF-ANN parameterizer (see Table 7.4). Consistent with [Aza00], the superior accuracy of the piecewise linear method is anticipated for this specific relationship between ESN output weights $\mathbf{W}_{OUT}$ and the orientation angle α: *"in the case of function approximation, piecewise continuous functions cannot in general be accurately modeled by monolithic artificial neural networks."* (p.28) On the contrary, FF-ANN modules are universal and potentially more effective for parameterization dependencies that deviate from the smooth, linearly interpolable functions depicted in Figure 6.11, Figure 6.12, and Figure 6.13.

Table 7.3 presents the recognition performance achieved using the FF-ANN parameterization modules. As expected, the imprecise computation of $\mathbf{W}_{OUT}$ negatively impacted the recognition performance, resulting in lower accuracy compared to piecewise linear parameterization for a larger number of known classes (compare to Table 7.1). Nonetheless, the FF-ANN parameterizers significantly outperformed the explicit rotation method in terms of IR (compare to Table 7.2). However, the negligible difference in achieved IR confirms the recognition performance's independence from the choice of parameterizer, highlighting the reliability of the general identification framework. Specifically, despite the lower accuracy of the FF-ANN parameterization modules, the recognition system effectively identified the majority of submitted symbol curves. These results reaffirm the advantage of modularity, allowing for flexible deployment where modules of one type can be replaced with more suitable alternatives.

Table 7.3.: Identification rate achieved by the ESN modules coupled with FF-ANN parameterizers for single-class sequence recognition for varying numbers of known classes. For two classes, "0" and "1" were considered; for three classes, "0", "1", and "2" were considered, and so forth.

Number of classes	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
IR (%)	100	100	100	100	90.0	88.15	81.60	75.25	61.11

7.2. Recognition of multi-class sequences

In experiments involving multi-class sequences, the presence of multiple target classes precludes the generation of a standard confusion matrix, which requires a unique pairing of target and actual classes for each processed sequence. Consequently, identification

Table 7.4.: Accuracy of piecewise linear parameterization modules and FF-ANN parameterizers coupled with the ESN modules for all considered symbol classes. The displayed error represents the maximum absolute error among all elements of the vector $\mathbf{W}_{OUT}$ in the respective ESN module.

ESN module	Error of piecewise linear parametrizer	Error of FF-ANN parametrizer
symbol "0"	4.15×10^{-8}	2.14×10^{-3}
symbol "1"	1.20×10^{-8}	1.92×10^{-2}
symbol "2"	3.15×10^{-8}	7.37×10^{-4}
symbol "3"	3.55×10^{-8}	7.60×10^{-4}
symbol "4"	5.04×10^{-10}	5.07×10^{-3}
symbol "5"	9.45×10^{-9}	6.20×10^{-4}
symbol "6"	2.15×10^{-8}	7.77×10^{-3}
symbol "7"	1.70×10^{-15}	6.28×10^{-4}
symbol "8"	3.22×10^{-8}	3.45×10^{-4}
symbol "9"	1.19×10^{-14}	2.47×10^{-2}

performance was assessed using cumulative indicators for different numbers of target classes, as presented in Table 7.5. IR statistics were calculated using Eq. (5.2) for different counts of correctly identified target classes: IR_1 represents cases where at least one target class was correctly identified, IR_2 indicates runs with two correctly recognized classes, and IR_3 corresponds to runs with three correctly recognized classes. As shown in Table 7.5, IR_1 remains close to 100% because one of the true classes was almost always among the top three classes with the highest WSA scores, regardless of the number of available modules. However, IR_2 and IR_3 values below 100% demonstrate that the second and third target classes were not consistently ranked within the top three after processing each sequence.

In general, the IR trend observed in Table 7.5 mirrors that seen for the single-class sequences presented in Table 7.1. In both scenarios, IR decreases as the number of modules in the ensemble increases. However, the identification problem became considerably more challenging with the increase in target classes.

Table 7.5.: Identification rates on multi-class sequences. IR_1 indicates the percentage of sequences with at least one correctly identified class, IR_2 indicates the percentage of at least two correctly identified classes, and IR_3 indicates the percentage of all mixed classes correctly identified.

Number of combined symbol curves	5	6	7	8	9	10
IR_1 **(%)**	100	100	99	99	97	98
IR_2 **(%)**	96	86	72	73	55	47
IR_3 **(%)**	52	35	19	13	7	7

7.3. Concluding summary for aperiodic modules

This chapter provided a detailed description of the study, presented the experiments, and outlined the main results obtained from applying the co-evolutionary synchronization approach to classify discontinuous sequence data through the synchronization of an ensemble of compact recurrent modules. The successful identification of dynamic components within compositional time series motivated further investigation into module synchronization for error-based classification of handwritten symbol sequences. However, it became clear from the initial results that the short sequences limited both the design of compact neural modules and the precise reconstruction of entire symbol curves. This prompted the question of how to classify given symbol sequences using only their imprecise reconstructions. The proposed classification method employs statistical decision-making based on the collective states of modules, which offer only coarse symbol reconstructions. Seeing that these states are generated during population progression towards an optimum, the winning symbol class is determined by the population that generated the highest number of module states yielding the most accurate reconstruction of the corresponding symbol curve.

Experiments on single-class sequences of handwritten digits demonstrated the viability of the proposed method, achieving reliable recognition in alphabets containing up to seven symbols. Expanding the alphabet beyond this size increased the difficulty of the task, resulting in classification performance dropping below 90%. The algorithm also effectively decomposed mixtures of multiple symbols, ensuring the recognition of the most prominent symbol in each sequence. This suggests that the population-based approach is suitable for probabilistic identification of dynamic components, where identified hidden components are indicated along with their likelihood of presence in the data. In both cases, the ensemble size proved to be a critical parameter, influencing recognition performance such that larger alphabets increased the likelihood of confusion between similar-looking symbols. This implies that increasing the alphabet size must be accompanied by appropriate adjustments to the rejection behavior, the design of modules with enhanced discrimination capabilities, and potentially, a revision of the module rejection strategy within the winner score accumulator.

The short lengths and low entropy of the symbol sequences posed challenges not only for the synchronization of recurrent neural modules but also for their design that produced compact, yet sufficiently flexible, modules for potential dynamic components. To address this, design was considered broadly, encompassing augmentation of training sequences, regularization of ESN modules, and constructing modules as a combination of reusable primary sub-modules and learning parameterizers. This module structure proved superior to the alternative concept of explicit rotation of symbol curves, which was also presented and studied in this thesis. The results indicated that learning parameterization enhances the resilience of module outputs, reducing susceptibility to premature convergence and improving recognition performance. Beyond achieving high recognition performance for handwritten symbols, the design of recurrent neural

modules as a combination of unique dynamic reservoirs and parameterizers offers a practical approach for realizing multi-tasking with ESN.

Furthermore, the limited length of the symbol curves shifted the focus of their decomposition towards identifying hidden dynamic components, rather than obtaining a stable long-term predictor, as was the case with oscillatory dynamics. Nevertheless, experiments demonstrated that the synchronization approach's capabilities extended beyond classifying dynamic components. To achieve precise reconstruction of each sequence, the method also determined the orientation angle and symbol size, parameters explicitly represented in the modular structure. Thereat, iterative sequence processing improved the accuracy of these determined parameter values. Thus, selecting an appropriate modular structure with known parameter semantics is crucial, broadening the identification capabilities of the network-based decomposition method. This allows it to function effectively with short sequences and contributes to the understanding of dynamic components in both single-class and multi-class sequences.

Rejection of low-confidence classifier outputs is essential for correct identification of dynamic components. However, in the context of synchronization, this presents a significant challenge that cannot be directly addressed using a rejection threshold. This is due to stochastic effects that cause substantial variations in population composition and accumulated class statistics over time. Consequently, this thesis proposed and explored an alternative rejection method based on factors that are implicitly controlling population search potential and are suitable for adjusting the rejection behavior, specifically the number of individual module updates and the module tolerance margin. The advantages of using these factors to adjust the rejection behavior were evident in experiments. The declining recognition performance with larger alphabets also highlighted the limitation of relying solely on these factors for rejection behavior, making it necessary to also consider the module's capacity to reproduce the symbol curves of other classes. Viewing this ability as a result of the balance between module plasticity and stability, appropriate module design becomes crucial for adjusting the rejection behavior and, consequently, for overall identification performance.

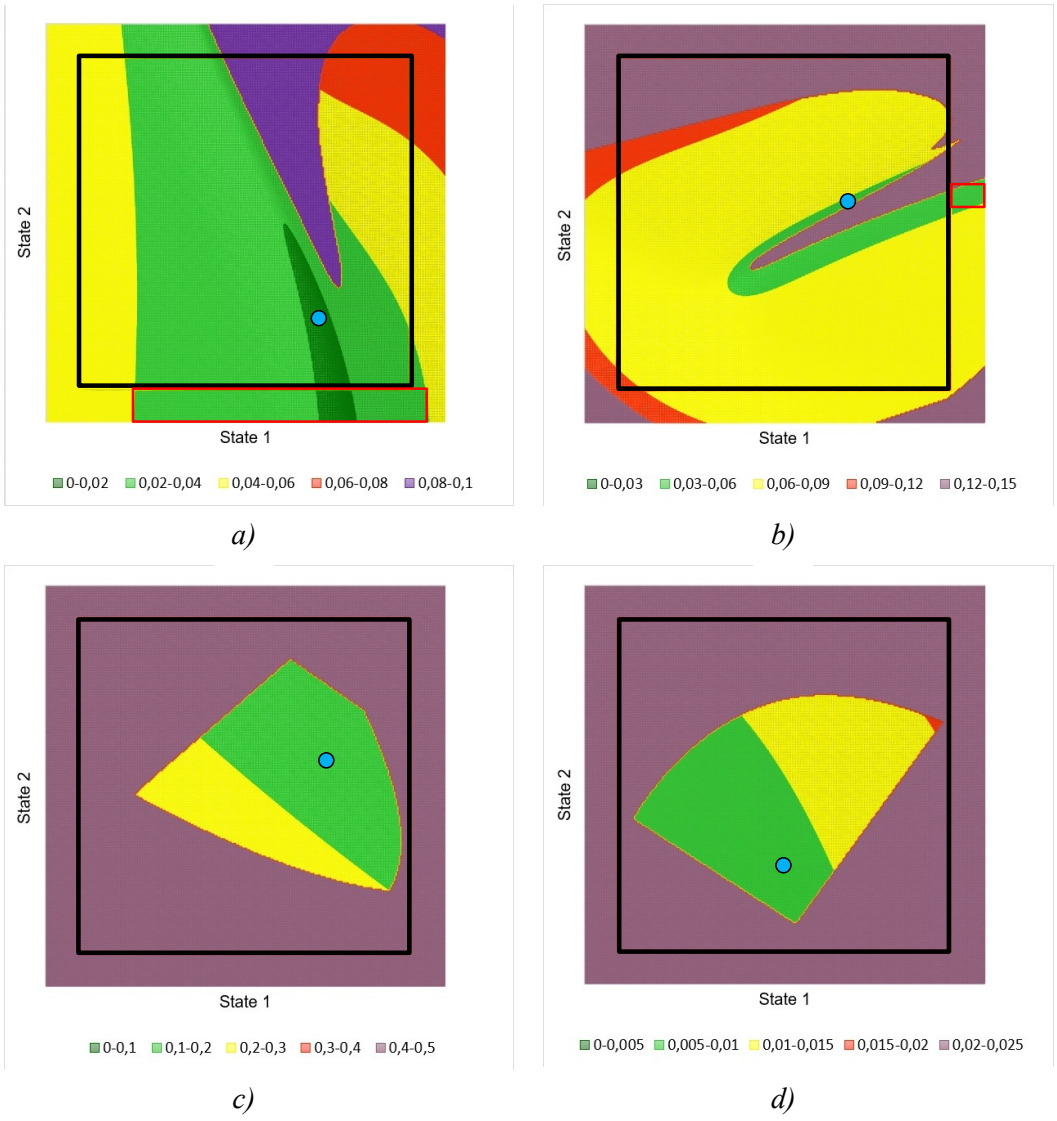

Figure 7.5.: Examples of the search space and its expansion by the small tolerance margin $\Delta\mu_m$ (beyond the black square). Here, $\Delta\mu_m$ is 0.1. The colors encode the magnitude of the error $e_{T_W}(t_c)$, calculated using Eq. (4.5), following variations in states 1 and 2 of the ESN modules. The blue points indicate the locations of the global optima. In figures a) and b), the low-error green region extends beyond the border (black square), resulting in a portion of the population being dispersed within this low-error extension (red rectangles). This dispersion diminishes the accuracy gain achieved from tuning the irrelevant modules. In figures c) and d), the low-error region is entirely contained within the valid range of states 1 and 2. Thus, increasing $\Delta\mu_m$ for these modules has no impact on adjusting their rejection.

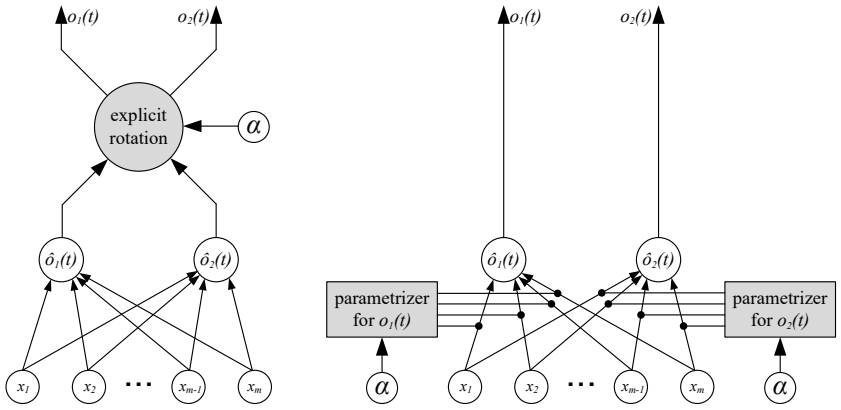

Figure 7.6.: Connections of the modules for explicit rotation of the symbol curve (left) and learning parameterization (right) are illustrated. In both configurations, the orientation angle α is determined during synchronization and used to compute the output signals $o_1(t)$ and $o_2(t)$.

8. General conclusions and promising research directions

Building upon the detailed summaries of the investigations in section 5.5 Concluding summary for oscillator modules and section 7.3 Concluding summary for aperiodic modules, this chapter outlines the main results and proposes ideas for potential future research.

8.1. General summary and conclusions

This thesis has proposed and explored a concept for the identification of dynamic components in the underlying structure of given sequence data. The concept relies on generating the hidden components using an ensemble of co-evolving dynamic modules. These modules' internal dynamics are adjusted through a proposed and studied synchronization method, aiming to achieve a precise reconstruction of the given data. This is accomplished by distributing the data among the available dynamic modules, forming a compact, single-layer modular network that provides an interpretable decomposition of the sequence into a set of dynamic components. To analyze the approach's properties, the synchronization behavior was examined under various conditions, including variations in module type (parametric and recurrent neural), optimization method (evolutionary and gradient-based), and data type (compositional time series, chaotic attractors, and sequences of handwritten symbol curves).

The conducted experiments have demonstrated that the characteristics of the time series and the deployed modules significantly influence the properties of the dynamic state space and, consequently, the convergence of the synchronization algorithm. The observed resilience of evolutionary optimization across tested combinations indicates its suitability within the synchronization framework for complex time series, particularly when modules must replicate non-differentiable or highly non-linear signals.

The inherent incompleteness of a priori information regarding the given dynamic data renders the adjustment of the modular network an under-constrained problem, indicating that relying solely on the overall quality of time series reconstruction is insufficient. This necessitates imposing appropriate constraints to ensure that the synchronization algorithm converges towards the desired distribution of activity among the modules. The experimental data indicate that successful synchronization is achieved by imposing

explicit static and adaptive constraints, while the module structure and network topology introduce a structural bias that further constrains the search space.

An analysis of the impact of the deployed neural modules on the convergence of the decomposition method revealed their clear advantages. The flexible variation of internal dynamics within their structure effectively prevents premature convergence, facilitating steady convergence to the global optimum, which corresponds to a meaningful data decomposition. However, designing the neural modules is challenging, as it requires finding an optimal solution to the plasticity-stability dilemma to prevent the decomposition algorithm from sliding towards trivial and meaningless solutions. Its optimality implies a proper balance between excessively plastic, large modules with low discrimination power and overly constrained, small modules with insufficient expressiveness.

In addition to other factors, the correct identification of hidden dynamic components requires adjusting the rejection of irrelevant modules during synchronization. This adjustment involves two key aspects. First, it must be addressed implicitly by maximizing the discrimination ability in the module design. This design should minimize similarity between replicated dynamic components, thereby preventing confusion. Second, it must be addressed explicitly through an appropriate selection of the rejection threshold. A too restrictive threshold increases the risk of premature convergence, while a too loose threshold increases the consumption of computational resources because of prolonged evaluation of irrelevant modules. The optimal threshold setting depends on the characteristics of the dynamic modules and the similarity between the replicated dynamic components.

The proposed solutions address the following key questions that must, in some form, be considered in any work dealing with modular networks. Firstly, regarding how to properly constrain the modular network to achieve the desired decomposition ([JJB91]), this thesis demonstrates the feasibility of semi-automatically decomposing dynamic data by leveraging assumptions about the nature of the dynamics and employing a combination of offline and online constraints. The offline constraints exclude a priori implausible decompositions, whereas the online constraints narrow the range of possible decompositions dynamically, incorporating information from incoming data regarding the dynamics' development. Given the vast range of potential dynamics, expert assumptions about their nature and interdependencies are crucial for coarsely constraining the network by selecting appropriate module types and topology. Subsequently, the module design aims to exploit this knowledge to implicitly restrict the variation of dynamics inside the modules.

Secondly, addressing how to partition the given data for training the modular network ([Jac+91]), this thesis proposes and explores the approach of dividing the data into fractions corresponding to the subprocesses that contributed to the overall time series generation. Consequently, the decomposition aims to identify these latent subprocesses by recognizing the associated modules. During data decomposition, the modular network

learns to generate the time series through the production of corresponding data fractions. Thus, the optimal data partitioning is achieved implicitly by tuning the dynamics inside the modules during their synchronization.

And thirdly, regarding how to select appropriate structures for individual modules ([JJ92; OXP07]), given that these modules encode distinct components of the overall solution, their design must facilitate a unique correspondence between each latent dynamic component and its dedicated module. This is achieved by minimizing functional overlap between modules, ensuring that each module accurately generates only a specific family of dynamics within a relatively narrow variation range. Consequently, this reduces confusion between dynamic components and enhances the modeling performance of the entire network.

Finally, reflecting on the question regarding appropriate tasks for modular networks ([HIT02]), the reported performance across diverse datasets showcased the strength of the identification approach and the potential and attractiveness of modular networks for identifying dynamic components within temporal data.

8.2. Promising research directions

Extending beyond linear combinations of dynamic modules can significantly enhance the method's abilities, making it particularly relevant for application areas such as AI, which aims to imitate information processing in the human brain, where mental representations are flexibly composed: *"for sensorimotor abstractions, the rules of composition need to be flexibly inferred by our minds to form plausible environmental scenarios in accordance with our world knowledge."* ([Epp+22], p.2) Therefore, the introduction of non-linear compositions of dynamic modules would represent a straightforward advancement in addressing time series of greater complexity.

Another approach to managing increased complexity involves introducing alternative topologies that reflect intricate dependencies among components within the sequence data. These topologies could account for potential inter-module dependencies, hierarchical organization, and non-linear contributions of dynamic components in the time series data. To align with data decomposition objectives, implementing such topologies may necessitate additional constraints to filter out a priori implausible solutions that can emerge during synchronization. Specifically, these constraints would likely be crucial for addressing stability concerns if the network topology assumes direct interaction between modules, which could sustain undesirable positive feedback loops within the network.

Modeling more complex dependencies also carries the risk of exceeding the model complexity threshold, beyond which modularity loses its advantages, such as understandability and flexibility. Oversized, irregular structures are extremely difficult for human comprehension. In this context, the decomposition method should prioritize compactness of the data model and regularity in the selected topology, ensuring an

interpretable data decomposition in the end. This is particularly relevant for automatic decomposition, where the topology pattern is not predefined. However, achieving interpretable decomposition in such cases requires defining appropriate criteria, likely based on model size constraints, similar to Occam's razor in [ZM93], and mathematically formalized measures of irregularity and symmetry ([RSM04; Sta07]); thereat, a limited degree of irregularity and asymmetry in the topology can be tolerated.

Depending on the module type deployed, one must address the interface between modules and their interconnection within the topology. For instance, recurrent neural ESN modules do not necessarily require direct output-to-input connections. Alternative configurations, such as connecting the output of one module to a subset of hidden neurons in another, can enrich the reservoir dynamics of the second module, thereby enhancing the representational capacity of the entire modular network. The Hyper-Network [Tra+24] serves as a relevant example of a dedicated module for controlling dynamics within a secondary module to enhance the overall network's capabilities in image segmentation.

Non-linear dependencies of the network output on module contributions would undoubtedly complicate the objective function landscape, potentially introducing more local optima, plateaus, and other optimization challenges. This would necessitate the exploration of alternative EA methods that outperform the currently employed DE. However, the potential of DE is not fully realized. According to [DS11], numerous DE modifications have demonstrated superior performance for objective functions with specific characteristics. Certain DE variants have shown promising results in constrained optimization problems and could offer performance improvements if module margin dynamics are adaptively adjusted, as demonstrated by the ϵ-DE algorithm [TS06], which is powerful due to its dynamic control of the tolerated constraint violation. Another promising direction is the online adaptation of the differential weight, which governs the exploration-exploitation trade-off. This suggests a progressive reduction of this parameter to shift the balance over time from exploration of the search space at the beginning of synchronization towards exploitation at the end of the sequence, when harmonization of the module dynamics becomes increasingly important.

While experiments have demonstrated the method's ability to adapt to non-stationary changes in the underlying data structure, incorporating a non-stationarity detector could further enhance tracking speed during frequent non-stationarity events, leading to improved accuracy on complex non-stationary time series. Alternatively, accelerating the identification of switching non-stationary dynamics can be achieved by employing synchronization for the evolution of redundant dynamic modular structures, similar to those in [RH10]. During the learning phase on switching non-stationary data, these modular structures would accumulate information about potential switching modes by developing new modules that were previously absent for identifying the current mode. Subsequently, during the exploitation phase, this accumulated knowledge can be utilized on demand. Assuming a compact overall size for the redundant structure, dynamic programming techniques ([BBS95; JJS94]) could potentially be applied for rapid

identification of relevant components and the topology of their communication.

To demonstrate synchronization's potential for achieving higher accuracy in experiments, data decomposition was terminated at the end of the synchronization sequence. However, this termination criterion appears overly conservative for practical engineering applications. Given that high prediction accuracy is often attained within the initial time steps of the synchronization sequence, it would be more efficient to halt synchronization once the prediction error falls below a predetermined threshold.

This thesis demonstrated the synchronization approach as a technique for time series analysis, capable of identifying individual components within the data generating process, including their parameterization. This holds significant potential for applications in domains with compositional dynamics, such as speech recognition, audio source extraction, and robotics. Audio files with multiple sources may require the identification of individual sources. In humanoid robotics, the modularity of complex movement patterns can be observed in trajectories generated by central pattern generators [WS09], where remarkably complex robotic movements can be generated by linear combinations of simple oscillators.

The aforementioned variety of candidate applications poses new challenges that would require scaling the presented synchronization framework, including efficient methods for the automatic or interactive design of compact but flexible modules. I hope that the reported findings and the presented answers to the research questions will serve as a solid basis for further exploration and will be useful for tackling these more challenging tasks by enhancing the synchronization approach and integrating it with other state-of-the-art systems for more powerful analysis of dynamic data in the near future.

A. Echo-States Neural Networks

Despite the relative novelty of ESNs, numerous publications have comprehensively explored their properties and presented variants tailored to specific problems. For instance, ESNs with filter neurons were introduced in [HH10; WSS08], self-organizing ESNs with RBF neurons were described in [Luk10], and ESNs with non-linear output stages were investigated in [GM09]. While these variations exist, the performance of all ESN types remains dependent on several key features inherent to the standard ESN initially proposed in [Jae01], and further detailed in the tutorial [Jae02], which provides comprehensive information on the standard ESN's properties, parameters, and their training.

Similar to classical MLPs, the standard ESN is composed of "weighted-sum-and-non-linearity" neurons. The general structure of a standard ESN is schematically shown in Figure A.1. It comprises hidden neurons, output neurons, and optionally, input neurons. Each hidden neuron can be connected to multiple or all other hidden neurons, forming a recurrent network known as the dynamic reservoir. In ESNs, these hidden neurons are referred to as reservoir neurons. In a fully connected reservoir, the output of any neuron is directly connected to the inputs of all other reservoir neurons. The dynamic reservoir serves as an environment where a signal, once injected, can propagate through the network from one reservoir neuron to another over an extended period. These propagating signals follow diverse paths, resulting in a variety of reservoir neuron states over time. These signals are considered echoes of the initially injected input signal when their temporal patterns resemble the injected one.

The states of the reservoir neurons form the basis for computing the ESN output. In a standard ESN, these states are transmitted to the output neurons via simple weighted output connections, the readout. Essentially, at each time step, the output neurons read out the current state of the dynamic reservoir. The weighted sum of these reservoir states determines the corresponding ESN output, as follows.

$$\mathbf{o}(t) = \mathbf{W}_{OUT} \cdot \mathbf{x}(t) \tag{A.1}$$

where $\mathbf{o}(t)$ is the state of the ESN output neurons with a linear activation function, $\mathbf{x}(t)$ is the vector containing the states of all reservoir neurons, and $\mathbf{W}_{OUT}$ is the matrix of output weights. The greater the variation of reservoir states, the more complex dynamics the ESN can produce. This capability is dependent on the reservoir size, that is, on the number of neurons in the dynamic reservoir. Generally, larger reservoirs offer larger variety of reservoir states and, consequently, greater capacity for modeling complex time series.

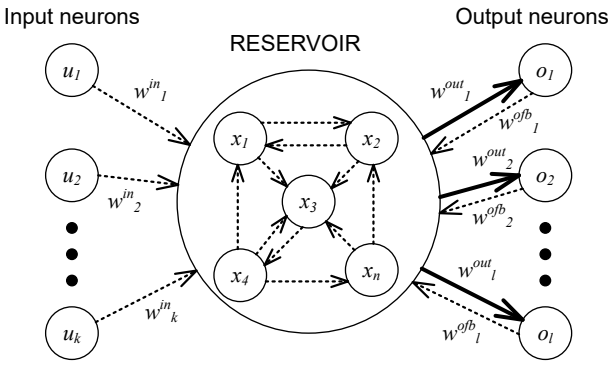

Figure A.1.: Structure of the standard ESN, as previously illustrated in [KOB21]. Dashed lines represent input and OFB connections with randomly initialized weights, similar to the connections within the dynamic reservoir. Solid lines indicate output connections from the reservoir neurons to the output neurons. The weights of these output connections are trainable.

However, maximizing reservoir size is not universally beneficial. As demonstrated in [KLB12], for certain dynamics, an optimal reservoir size exists that minimizes superfluous disruptive dynamics, thereby maximizing modeling accuracy.

The ESN weights comprise the reservoir weights $\mathbf{W}_{RES}$, the input connection weights $\mathbf{W}_{IN}$, the OFB connection weights $\mathbf{W}_{OFB}$, and the trainable output weights $\mathbf{W}_{OUT}$. The weights $\mathbf{W}_{IN}$ and $\mathbf{W}_{OFB}$ are initialized within intervals selected based on the characteristics of the modeled time series. The reservoir weights are also initialized but require additional scaling by SR α. This ESN parameter represents the largest absolute eigenvalue of the scaled weight matrix $\mathbf{W}_{RES}$. To ensure the vital echo-state property of the ESN, α must be less than 1. The appropriate value must be selected based on the characteristics of the modeled time series. Larger values result in slower attenuation of the reservoir dynamics and are, therefore, recommended for slowly changing time series. This slower attenuation is equivalent to an increased dynamic memory of the ESN reservoir.

All these weights contribute to updating the reservoir neuron states according to the following equation.

$$\mathbf{x}(t) = \mathbf{f}(\mathbf{W}_{IN}\mathbf{u}(t) + \mathbf{W}_{RES}\mathbf{x}(t-1) + \mathbf{W}_{OFB}\mathbf{o}(t-1)) \tag{A.2}$$

where t is the current time step, $\mathbf{x}(t)$ is the vector of reservoir states at the current time step, $\mathbf{f}(.)$ is the activation function of the reservoir neurons, $\mathbf{u}(t)$ is the vector of input values, and $\mathbf{o}(t-1)$ is the vector of the ESN output at the preceding time step. The vectors $\mathbf{u}(t)$ and $\mathbf{o}(t-1)$ are optional, depending on the chosen ESN structure. However, at least one of them must be present to provide a driving signal for the dynamic reservoir. Consistent with many ESN studies, the experiments in this thesis, which involve symbol sequences, utilized both $\mathbf{u}(t)$ and $\mathbf{o}(t-1)$ to drive the recurrent modules. Differently, the neural oscillator modules for continuous oscillatory time series did not incorporate the

input signal $\mathbf{u}(t)$. Publications on ESNs without input neurons are rare, exemplified by [XYH07; HH10; WSS08; RS11].

Incorporating the leaky-integration update rule (A.3) into the reservoir neurons offers the potential to further extend the dynamic memory of the ESN reservoir. These neurons are known as *leaky-integrator neurons*. According to the leaky-integration update rule, a fraction of the reservoir state from the previous time step is utilized to compute the state at the current time step, effectively simulating a gradual leakage of the previous state.

$$\mathbf{x}'(t) = (\mathbf{I} - \mathbf{A})\mathbf{x}(t-1) + \mathbf{x}(t) \tag{A.3}$$

where $\mathbf{A}$ is the diagonal matrix containing the individual leakage rates a_i of the reservoir neurons, and $\mathbf{I}$ is the identity matrix. The state $\mathbf{x}(t)$ is computed using Eq. (A.2). The leakage rates a_i must be selected from the range $0 < a_i \leqslant 1$, where $a_i < 1$ is necessary to ensure the forgetting of previous activity, which, in turn, is required for the stability of the dynamic reservoir. A value of 1 indicates complete leakage, meaning the reservoir state retains no previous activity. Leaky-integrator neurons are typically employed in applications where even large values of SR α fail to provide sufficient dynamic memory for modeling the given time series.

Training a standard ESN supposes adjusting the matrix $\mathbf{W}_{OUT}$ to enable the ESN to generate the target sequence with minimal error. Given a target sequence T, the method for calculating the required output weights $\mathbf{W}_{OUT}$ is derived from Eq. (A.1) as

$$\mathbf{W}_{OUT} = \mathbf{X}^{-1} \cdot \mathbf{T}, \tag{A.4}$$

where $\mathbf{X}^{-1}$ is the inverse matrix of the reservoir states $\mathbf{X}$ corresponding to the training sequence $\mathbf{T}$. This matrix $\mathbf{X}$ is obtained by feeding the target values $\mathbf{T}$ into the dynamic reservoir through the output feedback connections. In essence, the states $\mathbf{X}$ are computed as if the target values $\mathbf{T}$ were produced by the ESN itself. Computing the reservoir states $\mathbf{X}$ by feeding the target sequence T into the dynamic reservoir, instead of using the actual computed ESN outputs $\mathbf{o}(t)$, is known as teacher-forcing. The matrix equation (A.4) is solved using linear algebra methods, yielding the globally optimal solution in a single step, rather than through iterative approximations. Further details on the training procedure and additional properties of the standard ESN can be found in [Jae02].

Bibliography

[Aro+10] S. Arora, D. Bhattacharjee, M. Nasipuri, L. Malik, M. Kundu, and D.K. Basu. "Performance Comparison of SVM and ANN for Handwritten Devnagari Character Recognition". In: *International Journal of Computer Science Issues* 7.3 (2010), pp. 1–10.

[AS07] R. Angira and A. Santosh. "Optimization of dynamic systems: A trigonometric differential evolution approach". In: *Computational Chemical Engineering* 31.9 (2007), pp. 1055–1063.

[AT17] A. Ahmadi and J. Tani. "How can a recurrent neurodynamic predictive coding model cope with fluctuation in temporal patterns? Robotic experiments on imitative interaction". In: *Neural Networks* 92 (2017), pp. 3–16.

[Aza00] F. Azam. "Biologically Inspired Modular Neural Networks". PhD thesis. Virginia Polytechnic Institute, 2000.

[Bat+18] P. Battaglia, J. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, and R. Pascanu. "Relational inductive biases, deep learning, and graph networks". In: *arXiv* (June 2018), pp. 1–40. DOI: 10.48550/arXiv.1806.01261.

[BBE11] N. Bazzazzadeh, B. Brors, and R. Eils. "Reconstructing Evolutionary Modular Networks from Time Series Data". In: *The 14th International Conference on Information Fusion*. 2011, pp. 57–64.

[BBS95] A.G. Barto, S.J. Bradtke, and S.P. Singh. "Learning to act using real-time dynamic programming". In: *Artificial Intelligence* 72 (1995), pp. 81–138.

[Ber82] D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. .Academic Press Inc., 1982.

[BFC96] S. Bengio, F. Fessant, and D. Collobert. "Use of modular architectures for time series prediction". In: *Neural Processing Letters* 3.2 (1996), pp. 101–106.

[Bis09] Ch.M. Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag, 2009.

[BN04] N. Bertschinger and T. Natschlaeger. "Real-time computation at the edge of chaos in recurrent neural networks". In: *Neural Computation* 16.7 (2004), pp. 1413–1436.

[BP11] S. Babinec and J. Pospichal. "Modular Echo State Neural Networks in Time Series Prediction". In: *Computing & Informatics* 30.2 (2011), pp. 321–334.

[BPM12] K.V. Byadarhaly, M.C. Perdoor, and A.A. Minai. "A modular neural model of motor synergies". In: *Neural Networks* 32 (2012), pp. 96–108.

[Bre+09] J. Brest, A. Zamuda, B. Boskovic, M.S. Maucec, and V. Zumer. "Dynamic optimization using self-adaptive differential evolution". In: *IEEE Congress on Evolutionary Computing*. 2009, pp. 415–422.

[Bur98] C.J.C. Burges. "A tutorial on support vector machines for pattern recognition". In: *Knowledge discovery data mining* 2 (1998), pp. 1–43.

[But16] M.V. Butz. "Toward a Unified Sub-symbolic Computational Theory of Cognition". In: *Frontiers in Psychology* 925.7 (2016), pp. 1–19. DOI: 10.3389/fpsyg.2016.00925.

[BZS09] M. Bulacu, A. Brink T. Zant, and L. Schomaker. "Recognition of handwritten numerical fields in a large single-writer historical collection". In: *10th International Conference on Document Analysis and Recognition (ICDAR)*. 2009, pp. 808–812.

[CAB17] J. Chung, S. Ahn, and Y. Bengio. "Hierarchical Multiscale Recurrent Neural Networks". In: *ICLR*. arXiv:1609.01704v7, 2017, pp. 1–13.

[CD02] A. Carlisle and G. Dozier. "Tracking changing extrema with adaptive particle swarm optimizer". In: *The 5th Biannual World Automation Congress*. 2002, pp. 265–270.

[COY21] D. Corus, P.S. Oliveto, and D. Yazdani. "Fast immune system-inspired hypermutation operators for combinatorial optimization". In: *IEEE Transactions on Evolutionary Computation* 25.5 (2021), pp. 956–970.

[CT06] A. X. Carvalho and M. A. Tanner. "Modeling Nonlinearities with Mixtures-of-Experts of Time Series Models". In: *International Journal of Mathematics and Mathematical Sciences* 19423 (2006), pp. 1–22.

[CY98] D. Cheng and H. Yan. "Recognition of handwritten digits based on contour information". In: *Pattern Recognition* 31.3 (1998), pp. 235–255.

[Das+09] S. Das, A. Abraham, U.K. Chakraborty, and A. Konar. "Differential evolution using a neighborhood based mutation operator". In: *IEEE Transationas on Evolutionary Computation* 13.3 (2009), pp. 526–553.

[DKC05] S. Das, A. Konar, and U. K. Chakraborty. "Two improved differential evolution schemes for faster global search". In: *GECCO*. 2005, pp. 991–998.

[DS11] S. Das and P.N. Suganthan. "Differential Evolution: A Survey of the State-of-the-Art". In: *IEEE Transactions on Evolutionary Computation* 15.1 (2011), pp. 4–31. DOI: 10.1109/TEVC.2010.2059031.

[EB95] S. ElHihi and Y. Bengio. "Hierarchical recurrent neural networks for long-term dependencies". In: *Advances in Neural Information Processing Systems* (1995), pp. 493–499.

[Epp+22] M. Eppe, Ch. Gumbsch, M. Kerzel, Ph. Nguyen, M.V. Butz, and S. Wermter. "Intelligent problem-solving as integrated hierarchical reinforcement learning". In: *Nature Machine Intelligence* 4 (Jan. 2022), pp. 1–10. DOI: 10.1038/s42256-021-00433-9.

[Fab+20] S. Fabi, S. Otte, J.G. Wiese, and M.V. Butz. "Investigating Efficient Learning and Compositionality in Generative LSTM Networks". In: *ICANN*. Ed. by I. Farkas et al. Springer, 2020, pp. 143–154. DOI: 10.1007/978-3-030-61609-0_12.

[Fic04] S.G. Ficici. "Solutions concepts in Coevolutionary Algorithms". PhD thesis. Brandeis University, 2004.

[Fle70] R. Fletcher. "A new approach to variable metric algorithms". In: *Computer Journal* 13.3 (1970), pp. 317–322.

[FOB21] S. Fabi, S. Otte, and M.V. Butz. "Fostering Compositionality in Latent, Generative Encodings to Solve the Omniglot Challenge". In: *ICANN*. Ed. by I. Farkas et al. Springer Nature Switzerland AG, 2021, pp. 525–536. DOI: 10.1007/978-3-030-86340-1_42.

[FP00] P. Funes and J.B. Pollack. "Measuring progress in coevolutionary competition". In: *From Animals to Animats: 6th Sixth International Conference on Simulation of Adaptive Behavior*. Ed. by J. Meyer et al. MIT Press, 2000.

[FP01] S.G. Ficici and J.B. Pollack. "Pareto optimality in coevolutionary learning". In: *Advances in Artificial Life: 6th European Conference*. Ed. by P.S.J. Kelemen. Springer Verlag, 2001.

[Fre99] R.M. French. "Catastrophic Forgetting in Connectionist Networks: Causes, Consequences and Solutions". In: *Trends in Cognitive Sciences* 3.4 (1999), pp. 128–135.

[GBM19] C. Gumbsch, M.V. Butz, and G. Martius. "Autonomous Identification and Goal-Directed Invocation of Event-Predictive Behavioral Primitive". In: *IEEE Transactions on Cognitive and Developmental Systems* (2019), pp. 1–21. DOI: 10.1109/TCDS.2019.2925890.

[GGK09] N. Gradojevic, R. Gencay, and D. Kukolj. "Option Pricing With Modular Neural Networks". In: *IEEE Transactions on Neural Networks* 4.20 (2009), pp. 626–637.

[GM09] C. Gallicchio and A. Micheli. *On the Predictive Effects of Markovian and Architectural Factors of Echo State Networks*. Tech. rep. TR-09-22. University of Pisa, 2009.

[GOB17] C. Gumbsch, S. Otte, and M. V. Butz. "A Computational Model for the Dynamical Learning of Event Taxonomies". In: *ResearchGate* (2017), pp. 452–457.

[Gra+08] A. Graves, S. Fernandez, M. Liwicki, H. Bunke, and J. Schmidhuber. "Unconstrained online handwriting recognition with recurrent neural networks". In: *Advances in Neural Information Processing Systems* 20 (2008).

[Gra08] A. Graves. "Supervised Sequence Labelling with Recurrent Neural Networks". PhD thesis. Technical University of Munich, 2008.

[Gra12] A. Graves. "Sequence Transduction with Recurrent Neural Networks". In: *ArXiv:1211.3711v1* (2012), pp. 1–9.

[Gra13] A. Graves. "Generating Sequences With Recurrent Neural Networks". In: *ArXiv:1308.0850* (2013), pp. 1–43.

[GS08] A. Graves and J. Schmidhuber. "Offine handwriting recognition with multidimensional recurrent neural networks". In: *Advances in Neural Information Processing Systems* 21 (2008).

[GSM08] F. Gomez, J. Schmidhuber, and R. Miikkulainen. "Accelerated neural evolution through cooperatively coevolved synapses". In: *The Journal of Machine Learning Research* 9 (2008), pp. 937–965.

[Hay94] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan Co., 1994.

[HH10] G. Holzmann and H. Hauser. "Echo State Networks with Filter Neurons and a Delay&Sum Readout". In: *Neural Networks* 23.2 (2010), pp. 244–256.

[HIT02] M. Hüsken, C. Igel, and M. Toussaint. "Task-dependent evolution of modularity in neural networks". In: *Connection Science* 14.3 (2002), pp. 219–229.

[HM94] B.L.M. Happel and J.M.J. Murre. "The Design and Evolution of Modular Neural Network Architectures". In: *Neural Networks* 7 (1994), pp. 985–1004. DOI: 10.1016/S0893-6080(05)80155-8.

[HO01] N. Hansen and A. Ostermeier. "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary Computation* 9.2 (2001), pp. 159–195.

[Hoc+01] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In: *A Field Guide to Dynamical Recurrent Neural Networks* (2001).

[HSL99] N.E. Huang, Z. Shen, and S.R. Long. "A New View of Nonlinear Water Waves: The Hilbert Spectrum". In: *Annual Review of Fluid Mechanics* 31 (1999), pp. 417–457. DOI: 10.1146/annurev.fluid.31.1.417.

[HWH07] F. Huang, L. Wang, and Q. He. "An effective co-evolutionary differential evolution for constrained optimization". In: *Applied Mathematical Computing* 186.1 (2007), pp. 340–356.

[INS02] A.J. Ijspeert, J. Nakanishi, and S. Schaal. "Movement Imitation with Nonlinear Dynamical Systems in Humanoid Robots". In: *International Conference on Robotics and Automation (ICRA)*. 2002, pp. 1–6.

[Jac+91] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. "Adaptive mixtures of local experts". In: *Neural Computation* 3.1 (1991), pp. 79–87. DOI: 10.1162/neco.1991.3.1.79.

[Jae+07] H. Jaeger, M. Lukosevicius, D. Popovici, and U. Sewert. "Optimization and Applications of Echo State Networks with Leaky Integrator Neurons". In: *Neural Networks* 20 (2007), pp. 1–40.

[Jae01] H. Jaeger. *The "echo state" approach to analysing and training recurrent neural networks*. GMD Report 148. German National Research Center for Information Technology, 2001.

[Jae02] H. Jaeger. *A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*. GMD Report 159. German National Research Center for Information Technology, 2002.

[Jae03] H. Jaeger. "Adaptive nonlinear system identification with echo state networks". In: *Advances in Neural Information Processing Systems* 15 (2003), pp. 593–600.

[Jae07] H. Jaeger. *Discovering multiscale dynamical features with hierarchical Echo State Networks*. Tech. rep. 10. Jacobs University Bremen, 2007.

[JH04] H. Jaeger and H. Haas. "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication". In: *Science* 304.5667 (2004), pp. 78–80.

[Jin07] N. Jin. "Constraint-based Co-evolutionary Genetic Programming for Bargaining Problems". PhD thesis. University of Essex, 2007.

[JJ91] M.I. Jordan and R.A. Jacobs. "A competitive modular connectionist architecture". In: *Advances in Neural Information Processing Systems* 3 (1991), pp. 767–773.

[JJ92] M.I. Jordan and R.A. Jacobs. "Hierarchies of adaptive experts". In: *Advances in Neural Information Processing Systems* 4 (1992), pp. 985–992.

[JJ93] M.I. Jordan and R.A. Jacobs. "Hierarchical mixtures of experts and the EM algorithm". In: *International Joint Conference on Neural Networks*. 1993, pp. 1339–1344.

[JJB91] R.A. Jacobs, M.I. Jordan, and A.G. Barto. "Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks". In: *Cognitive Science* 15.2 (1991), pp. 219–250. DOI: 10.1207/s15516709cog1502_2.

[JJS94] T. Jaakkola, M.I. Jordan, and S.P. Singh. "On the convergence of stochastic iterative dynamic programming algorithms". In: *Neural Computation* 6.6 (1994), pp. 1185–1201.

[JMP07] H. Jaeger, W. Maass, and J. Principe. "Special issue on echo state networks and liquid state machines (editorial)". In: *Neural Networks* 20.3 (2007), pp. 287–289. DOI: 10.1016/j.neunet.2007.04.001.

[Jun07] J.-Y. Jung. "Evolutionary DEsign of ARtificial Neural Networks Using a Descriptive Encoding Language". PhD thesis. University of Maryland, 2007.

[JW03] T. Jansen and R.P. Wiegand. "Exploring the explorative advantage of the cooperative coevolutionary (1+1) EA". In: *International Conference on Genetic and Evolutionary Computation (GECCO)*. Vol. 2723. Springer-Verlag, 2003, pp. 310–321.

[JX95] M.I. Jordan and L. Xu. "Convergence results for the EM approach to mixtures of experts architectures". In: *Neural Networks* 8.9 (1995), pp. 1409–1431.

[KC02] D. Kincaid and W. Cheney. *Numerical Analysis*. Brooks/Cole, 2002.

[Kha06] V.R. Khare. "Automatic Problem Decomposition Using Co-Evolution and Modular Neural Networks". PhD thesis. University of Birmingham, 2006.

[KLB12] D. Koryakin, J. Lohmann, and M.V. Butz. "Balanced echo state networks". In: *Neural Networks* 36 (2012), pp. 35–45.

[KOB21] D. Koryakin, S. Otte, and M.V. Butz. "Inference of time-series components by means of online co-evolution". In: *Genetic Programming and Evolvable Machines* (2021), pp. 1–29. DOI: 10.1007/s10710-021-09408-6.

[Kor01] D. Koryakin. "Adaptation, Evaluation and Optimization of Recognition Systems for Traffic Signs". MA thesis. University of Ulm, 2001.

[KS13] M. Kharratzadeh and T. Shultz. "Neural-network modelling of Bayesian learning and inference". In: *The 35th Annual Conference of the Cognitive Science Society*. 2013, pp. 2686–2691.

[KS97] U. Kreßel and J. Schürmann. "Pattern classification techniques based on function approximation". In: *Handbook of Character Recognition and Document Image Analysis*. Ed. by H. Bunke and P.S.P. Wang. 1997, pp. 49–78.

[KSB15] D. Koryakin, F. Schrodt, and M.V. Butz. "Ensembles of Neural Oscillators". In: *New Challenges in Neural Computation*. Vol. 3. 2015, pp. 57–64.

[LKK04] F. Lindner, U. Kreßel, and S. Kaelberer. "Robust recognition of traffic signals". In: *IEEE Intelligent Vehicles Symposium*. 2004, pp. 49–53.

[Luk10] M. Lukosevicius. *On self-organizing reservoirs and their hierarchies*. Tech. rep. 25. Jacobs University Bremen, 2010.

[LZ00] J. Lampinen and I. Zelinka. "On stagnation of the differential evolution algorithm". In: *6th International Mendel Conference on Soft Computing*. 2000, pp. 76–83.

[LZ99] J. Lampinen and I. Zelinka. "Mixed integer-discrete-continuous optimization with differential evolution". In: *5th International Mendel Conference on Soft Computing*. 1999, pp. 71–76.

[Mar03] R. Marti. "Multi-start methods". In: *Handbook of metaheuristics* (2003), pp. 355–368.

[MC13] Q.-L. Ma and W.-B. Chen. "Modular state space of echo state network". In: *Neurocomputing* 122 (2013), pp. 406–417.

[MM05] R. Mendes and A.S. Mohais. "DynDE: A differential evolution for dynamic optimization problems". In: *IEEE Congress on Evolutionary Computing*. Vol. 2. 2005, pp. 2808–2815.

[MM97] D.E. Moriarty and R. Miikkulainen. "Forming neural networks through efficient and adaptive coevolution". In: *Evolutionary Computation* 5.4 (1997), pp. 373–399. DOI: 10.1162/evco.1997.5.4.373.

[MMD17] T.K. Mishra, B. Majhi, and R. Dash. "A Contour Descriptors-Based Generalized Scheme for Handwritten Odia Numerals Recognition". In: *Information Processing Systems* 13.1 (2017), pp. 174–183. DOI: 10.3745/JIPS.02.0012.

[New06] M.E.J. Newman. "Modularity and community structure in networks". In: *ArXiv:physics/0602124v1* (2006), pp. 1–7.

[NP95] S. Nolfi and D. Parisi. *Learning to adapt to changing environments in evolving neural networks*. Tech. rep. 95-15. Institute of Psychology C.N.R. - Rome, 1995.

[OP05] M.C. Ozturk and J.C. Principe. "Computing with transiently stable states". In: *IEEE International Joint Conference on Neural Networks*. Vol. 3. 2005, pp. 1467–1472.

[ORB19] S. Otte, P. Rubisch, and M. V. Butz. "Gradient-Based Learning of Compositional Dynamics with Modular RNNs". In: *Artificial Neural Networks and Machine Learning (ICANN)* 11727 (2019), pp. 484–496. DOI: 10.1007/978-3-030-30487-4_38.

[Ott+15] S. Otte, F. Becker, M.V. Butz, M. Liwicki, and A. Zell. "Learning Recurrent Dynamics with Differential Evolution". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2015.

[Ott+16] S. Otte, M.V. Butz, D. Koryakin, F. Becker, M. Liwicki, and A. Zell. "Optimizing Recurrent Reservoirs With Neuro-Evolution". In: *Neurocomputing* 192 (2016), pp. 128–138.

[OXP07] M.C. Ozturk, D. Xu, and J. Principe. "Analysis and design of echo state network". In: *Neural Computation* 19.1 (2007), pp. 111–138.

[Pas+01] F. Pasemann, U. Steinmetz, M. Huelse, and B. Lara. "Robot Control and the Evolution of Modular Neurodynamics". In: *Theory in Biosciences* 120 (2001), pp. 311–326.

[Pas98] F. Pasemann. "Structure and Dynamics of recurrent Neuromodules". In: *Theory in Biosciences* 117 (1998), pp. 1–17.

[Pat+18a] J. Pathak, B. Hunt, M. Girvan, Zh. Lu, and E. Ott. "Model-Free Prediction of Large Spatiotemporally Chaotic Systems from Data: A Reservoir Computing Approach". In: *Physical Review Letters* 120 (2018), pp. 1–5. DOI: 10.1103/PhysRevLett.120.024102.

[Pat+18b] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. Hunt, M. Girvan, and E. Ott. "Hybrid Forecasting of Chaotic Processes: Using Machine Learning in Conjunction with a Knowledge-Based Model". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 28.4 (2018), p. . DOI: 10.1063/1.5028373.

[PD00] M.A. Potter and K.A. DeJong. "Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents". In: *Evolutionary Computation* 8.1 (2000), pp. 1–29. DOI: 10.1162/106365600568086.

[Per+17] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A.C. Courville. "FiLM: Visual Reasoning with a General Conditioning Layer". In: *CoRR* abs/1709.07871 (2017). arXiv: 1709.07871. URL: http://arxiv.org/abs/1709.07871.

[PJT95] F. Peng, R.A. Jacobs, and M.A. Tanner. "Bayesian Inference in Mixtures-of-Experts and Hierarchical Mixtures-of-Experts Models With an Application to Speech Recognition". In: *Journal of the American Statistical Association* (1995), pp. 1–26. DOI: 10.1080/01621459.1996.10476965.

[Pol+07] R. Poli, W.B. Langdon, N.F. McPhee, and J.R. Koza. *Genetic Programming An Introductory Tutorial and a Survey of Techniques and Applications*. Tech. rep. CES-475. University of Essex, 2007.

[Pol99] D.S.G. Pollock. "Smoothing with Cubic Splines". In: *Handbook of Time Series Analysis, Signal Processing, and Dynamics* (1999), pp. 1–25. DOI: 10.1016/B978-012560990-6/50013-0.

[PSL05] K. Price, R. Storn, and J. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*. Springer Verlag, 2005.

[QHS09] A.K. Qin, V.L. Huang, and P.N. Suganthan. "Differential evolution algorithm with strategy adaptation for global numerical optimization". In: *IEEE Transactions on Evolutionary Computation* 13.2 (2009), pp. 398–417.

[Qia+16] J. Qiao, F. Li, H.-G. Han, and W. Li. "Growing Echo-State Network With Multiple Subreservoirs". In: *IEEE Transactions on Neural Networks and Learning Systems* 28 (2016), pp. 1–14. DOI: 10.1109/TNNLS.2016.2514275.

[Rec73] I. Rechenberg. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, 1973.

[RH10] J.W. Robinson and A.J. Hartemink. "Learning Non-Stationary Dynamic Bayesian Networks". In: *Journal of Machine Learning Research* 11 (2010), pp. 3647–3680.

[RI09] B. Roeschies and C. Igel. "Structure optimization of reservoir networks". In: *Logic Journal of IGPL* (2009), pp. 1–35. DOI: 10.1093/jigpal/jzp043.

[RS11] F. Reinhart and J. Steil. "Reservoir regularization stabilizes learning of Echo State Networks with output feedback". In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. 2011, pp. 59–64.

[RS12] F. Reinhart and J. Steil. "Regularization and stability in reservoir networks with output feedback". In: *Neurocomputing* 90 (2012), pp. 96–105.

[RSM04] J. Reisinger, K.O. Stanley, and R. Miikkulainen. "Evolving Reusable Neural Modules". In: *Genetic and Evolutionary Computation Conference (GECCO)*. Ed. by K.Deb et al. Springer, 2004, pp. 69–81.

[RTS08] S. Rahnamayan, H.R. Tizhoosh, and M.M.A. Salama. "Opposition based differential evolution". In: *IEEE Transactions on Evolutionary Computation* 12.1 (2008), pp. 64–79.

[Rue93] J. Rueckl. "Jumpnet: A multiple-memory connectionist architecture". In: *The 15th Annual Conference of the Cognitive Science Society*. 1993, pp. 1–10.

[RWH96] M. Revow, C.K.I. Williams, and G.E. Hinton. "Using Generative Models for Handwritten Digits Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.6 (1996), pp. 592–606.

[SC02] A. Simoes and E. Costa. "Using genetic algorithms to deal with dynamic environments: A comparative study of several approaches based on promoting diversity". In: *Genetic and Evolutionary Computation Conference (GECCO)*. Vol. 2. 2002, pp. 698–707.

[Sch+07] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. "Training Recurrent Neural Networks by Evolino". In: *Neural Computation* 19.3 (2007), pp. 757–779.

[Sch96] J. Schuermann. *Pattern classification: a unified view of statistical and neural approaches*. Wiley Interscience, 1996.

[Sha97a] A.J.C. Sharkey. "Modularity, Combining and Artificial Neural Nets". In: *Connection Science* 9.1 (1997), pp. 3–10. DOI: 10.1080/095400997116702.

[Sha97b] N.E. Sharkey. "Neural networks for coordination and control: The portability of experiential representations". In: *Robotics and Autonomous Systems* 22 (1997), pp. 345–359.

[Sol02] S. Soltani. "On the use of the wavelet decomposition for time series prediction". In: *Neurocomputing* 48 (2002), pp. 267–277.

[SP95] R. Storn and K.V. Price. *Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces*. Tech. rep. 95-012. ICSI USA, 1995.

[SPK98] Ch. Scheier, R. Pfeifer, and Y. Kunyioshi. "Embedded neural networks: Exploiting constraints". In: *Neural Networks* 11 (1998), pp. 1551–1569.

[Sta07] K.O. Stanley. "Compositional Pattern Producing Networks: A Novel Abstraction of Development". In: *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems* (2007), pp. 1–36.

[TM99] K. Trojanowski and Z. Michalewicz. "Searching for Optima in Non-stationary Environments". In: *IEEE Congress on Evolutionary Computation (CEC)*. Ed. by Angeline et al. Vol. 3. 1999, pp. 1843–1850.

[Tra+24] M. Traub, F. Becker, A. Sauter, S. Otte, and M.V. Butz. "Loci-segmented: improving scene segmentation learning". In: *International Conference on Artificial Neural Networks*. Springer, 2024, pp. 45–61.

[TS06] T. Takahama and S. Sakai. "Constrained optimization by the e-constrained differential evolution with gradient-based mutation and feasible elites". In: *IEEE Congress on Evolutionary Computing*. 2006, pp. 308–315.

[TT22] I.G. Tsoulos and A. Tzallas. "An improved multi-start-based method for global optimisation problems". In: *International Journal of Computational Intelligence Studies* 11.2 (2022), pp. 73–93. DOI: 10.1504/IJCISTUDIES.2022.126905.

[Vap95] V. Vapnik. *The nature of statistical learning theory*. Springer, 1995.

[VS09] G.K. Venayagamoorthy and B. Shishir. "Effects of spectral radius and settling time in the performance of echo state networks". In: *Neural Networks* 22.7 (2009), pp. 861–863.

[Wan+15] J.H. Wang, H.Y. Wang, Y.L. Chen, and C.M. Liu. "A constructive algorithm for unsupervised learning with incremental neural network". In: *Journal of applied research and technology* 13.2 (2015), pp. 188–196.

[Win00] M. Wineberg. "Improving the Behavior of the Genetic Algorithm in a Dynamic Environment". PhD thesis. Carleton University of Ottawa, 2000.

[Wri15] S.J. Wright. "Coordinate descent algorithms". In: *Mathematical Programming* 151.1 (2015), pp. 3–34. DOI: 10.1007/s10107-015-0892-3.

[WS09] F. Wyffels and B. Schrauwen. "Design of a central pattern generator using reservoir computing for learning human motion". In: *IEEE Advanced Technologies for Enhanced Quality of Life* (2009), pp. 118–122.

[WSS08] F. Wyffels, B. Schrauwen, and D. Stroobandt. "Stable Output Feedback in Reservoir Computing Using Ridge Regression". In: *ICANN*. Ed. by V. Kurkova, R. Neruda, and J. Koutnik. Springer, 2008, pp. 808–817.

[XYH07] Y. Xue, L. Yang, and S. Haykin. "Decoupled echo state networks with lateral inhibition". In: *Neural Networks* 20.3 (2007), pp. 365–376.

[Yan+09] Z. Yang, J. Zhang, K. Tang, X. Yao, and A.C. Sanderson. "An Adaptive Coevolutionary Differential Evolution Algorithm for Large-scale Optimization". In: *IEEE Congress on Evolutionary Computation (CEC)*. 2009, pp. 102–109.

[Yan+18] C. Yang, J. Qiao, L. Wang, and X. Zhu. "Dynamical regularized echo state network for time series prediction". In: *Neural Computing and Applications* (2018), pp. 1–14. DOI: 10.1007/s00521-018-3488-z.

[YC12] Sh.-H. Yang and Y.-P. Chen. "An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications". In: *Neurocomputing* 86 (2012), pp. 140–149.

[YG06] R. Yan and R.X. Gao. "Hilbert-Huang Transform-Based Vibration Signal Analysis for Machine Health Monitoring". In: *IEEE Transactions on Instrumentation and Measurement* 55.6 (2006), pp. 2320–2329. DOI: 10.1109/TIM.2006.887042.

[YL98] X. Yao and Y. Liu. "Making Use of Population Information in Evolutionary Artificial Neural Networks". In: *IEEE Transactions on Systems and Cybernetics-Part B: Cybernetics* 28.3 (1998), pp. 417–425.

[YM07] Ch.H. Yong and R. Miikkulainen. *Coevolution of Role-Based Cooperation in Multi-Agent Systems*. Tech. rep. AI07-338. University of Texas at Austin Austin, 2007.

[ZM93] B.-T. Zhang and H. Muelenbein. "Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor". In: *Complex Systems* 7.3 (1993), pp. 199–220.

[ZMA99] A. Zeevi, R. Meir, and R.J. Adler. "Non-Linear Models for Time Series Using Mixtures of Autoregressive Models". In: *www.researchgate.net* (1999), pp. 1–49.