Inference of time series components by online co-evolution

Danil Koryakin¹ · Sebastian Otte¹ · Martin V. Butz¹

Received: 23 August 2020 / Revised: 25 June 2021 / Accepted: 5 July 2021 /
Published online: 21 July 2021
© The Author(s) 2021

Abstract

Time series data is often composed of a multitude of individual, superimposed dynamics. We propose a novel algorithm for inferring time series compositions through evolutionary synchronization of modular networks (ESMoN). ESMoN orchestrates a set of trained dynamic modules, assuming that some of those modules' dynamics, suitably parameterized, will be present in the targeted time series. With the help of iterative co-evolution techniques, ESMoN optimizes the activities of its modules dynamically, which effectively synchronizes the system with the unfolding time series signal and distributes the dynamic subcomponents present in the time series over the respective modules. We show that ESMoN can adapt modules of different types. Moreover, it is able to precisely identify the signal components of various time series dynamics. We thus expect that ESMoN will be useful also in other domains—including, for example, medical, physical, and behavioral data domains—where the data is composed of known signal sources.

Keywords Time series identification · Modular recurrent neural networks · Dynamic modules · Non-stationary processes · Online co-evolution · Synchronization

1 Introduction

Most time series data originates from sequential measurements, which typically reflect the dynamics of several, concurrently unfolding, hidden processes. Time series analyses attempt to identify the contents of the time series and, ideally, their exact causes. That is, an analysis ideally discloses which individual

Area Editor: Sebastian Risi.

✉ Martin V. Butz
martin.butz@uni-tuebingen.de

Danil Koryakin
danil.koryakin@uni-tuebingen.de

¹ Neuro-Cognitive Modeling, Department of Computer Science, Faculty of Science, University of Tübingen, Sand 14, 72076 Tübingen, Germany

processes generate the time series, how big their individual contributions are, their current respective parameterization, and their relative alignment. In this study, we assume that the considered time series are indeed produced by a finite number of underlying processes, which we model either by dedicated parameterized equation modules or recurrent neural network (RNN) modules.

Modularized RNNs have been used to model sequential data in various previous studies (cf. e.g. [3, 16, 22, 36]). For example, a modularized spiking neural network has been used to model motor synergies [5]. Using evolutionary algorithms, modularized ANNs were optimized for the generation of biologically plausible motion [10]. Related modularizations of feed-forward networks to various tasks can be found in [2, 4, 12, 14], whereas generalized linear models were used in [6]. Sometimes the individual modules are not connected to each other [3, 6, 22]. Elsewhere, they are hierarchically organized in order to foster system understanding [2, 16]. One highly relevant advantage of modularity is explainability, particularly when the modules explicitly identify the underlying processes. Additionally, a priori knowledge can be integrated easily [27] and the resulting model is typically rather compact [21, 26]. Application options for modularized network architectures cover a wide range, including the modeling of static [13, 14] and dynamic [4, 17, 27] data, robot control problems [5, 12], or even playing board games [29]. In all cases, the individual modules' activities of the network need to be tuned to the data at hand.

In this paper we focus on temporally dynamic processes that can be represented by linear combinations of independent module dynamics. The modules' parameters and internal activities are tuned to the data stream on-the-fly, aiming at synchronizing the modules' activities online in order to accurately simulate the perceived dynamics. As a result, the resulting activities can be used in a closed-loop manner to simulate the time series dynamics further into the future. The main assumption is that the hidden dynamic processes that generate the time series data can be modeled by means of the available modules, whereby each module produces rather simple dynamics, such as a sine wave with a particular frequency and amplitude range.

We propose evolutionary synchronization of modular networks (ESMoN), a novel algorithm that analyzes, decomposes, and tunes its predictive activities to the time series dynamics online. We equip ESMoN with particular dynamic modules, which model components of the target dynamics. ESMoN assumes that combinations of the available, suitably parameterized and synchronized modules can model the target dynamics. It tunes both the internal dynamics and the parameter values of its dynamic modules online, minimizing a time series reconstruction error by means of co-evolution. We will show that the algorithm is largely indifferent to the types of modules it is equipped with, which allows a flexible module choice dependent on the target application at hand. As suggested elsewhere, any tuning algorithm of a modular network needs proper constraints to ensure convergence towards the required solution [6, 13]. In ESMoN, the concurrent tuning of the modules' activities and the parameters through co-evolving populations using differential evolution techniques fosters convergence. Moreover, the types of modules and their possible parameterizations constrain the tuning process, where we either use parameterized equation modules or small echo state network (ESN) modules as individual dynamic models.

We now first give short introductions to co-evolution and differential evolution—the two main evolutionary principles used in ESMoN. We then detail the ensemble of dynamic modules, the parameters, and involved activities, which ESMoN strives to optimize online. Finally, we introduce the employed ESN modules. Section 3 describes the ESMoN’s structure and algorithm, while Sect. 4 analyzes its performance on time series of different complexity, including imbalanced and non-stationary data, using different types of dynamic modules. An outlook concludes the paper.

2 Theoretical background

This section gives short descriptions of the basic concepts used in ESMoN. The details on ESMoN itself are then provided in Sect. 3.

2.1 Co-evolution

ESMoN co-evolves M populations simultaneously. Each such population $m \in \{1, \dots, M\}$ contains individuals, which generate (simple) time series predictions. The individual predictions are then combined to yield actual (complex) joint time series predictions. Due to the consequentially simpler individual genotype structures, co-evolutionary search has been shown to become less greedy and to overcome local optima easier [8]. The independence of the modules offers further advantages such as decoupling of the unfolding co-evolutionary dynamics and simpler regrouping of the modules.

Each individual i_m in a particular module’s population m is assigned a fitness in a co-evolutionary manner, aiming at minimizing joint error:

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

where $C(m) = \{m' | m' \in M \wedge m' \neq m\}$ refers to all other (contextual) modules and $i_{C(m)}$ specifies the respective individuals from the other modules’ populations, which are combined with i_m for its fitness calculation. We apply the standard mean squared error (MSE) to determine $e()$, but certainly other error measures may be applied.

Note that the particular individual fitness supposes cooperation between populations for maximizing $F(i_m)$ [18, 23, 28, 31]. The choice of the context individuals $i_{C(m)}$ in (1) induces fitness subjectivity, which is a critical challenge for co-evolutionary approaches, because an “unlucky” context choice may lead to a loss of useful individuals. We use a rank-based ordering of the individuals in each population and combine individuals of the same rank across populations to estimate individual current fitness. A similar but slightly more probabilistic approach has been used, for example, in [8].

2.2 Differential evolution (DE)

All candidate solutions are encoded by a population of real-valued vectors. We used differential evolution (DE) [7] to reduce computational effort while fostering fast online convergence. DE generates new candidate solutions by means of a multi-parent mutation rule:

$$\mathbf{v}_i = \mathbf{p}_a + F \cdot (\mathbf{p}_b - \mathbf{p}_c), \quad (2)$$

where $\mathbf{v}_i$ is a mutant genotype, which is produced for each parental individual $\mathbf{p}_i$ in a population. $\mathbf{p}_a$, $\mathbf{p}_b$, $\mathbf{p}_c$ denote the genotypes of randomly chosen individuals, which must differ from each other as well as from $\mathbf{p}_i$. $F \in (0, 1]$ is the differential weight, which essentially determines the strength of modifying genotype $\mathbf{p}_a$. Mutation is followed by recombining $\mathbf{v}_i$ with its *direct parent* $\mathbf{p}_i$, yielding the child $\mathbf{c}_i$. Recombination applies uniform crossover, replacing each parental gene with probability CR . We additionally ensure that at least one gene is actually replaced.

After recombination, $\mathbf{c}_i$ replaces $\mathbf{p}_i$ if its fitness is not worse than the one of its parent.

2.3 Ensemble of dynamic modules

ESMoN consists of a number of modules, each of which is designed to model a particular range of dynamics. Together, we refer to the modules by an ensemble of dynamic modules $\mathcal{M}$. Each module is operating entirely independently of the others. There are no direct connections between them. As a result, the modules can be designed independently, which enables the flexible combination of highly differing dynamics, such as the ones studied in [21].

We compute the output of an ensemble of modules by a weighted sum (cf. the schematic visualization in Fig. 1), but other combinations are generally possible as well:

$$\mathbf{o}(t) = \sum_{m=1}^M \mathbf{r}_m \cdot \mathbf{o}_m(t), \quad (3)$$

where M denotes the number of modules $\mathcal{M}$, $\mathbf{o}_m(t)$ is the output of the m th module at time t , and $\mathbf{r}_m$ is a responsibility parameter, which weighs the impact of the m th module's output. The product of the module output $\mathbf{o}_m(t)$ and its responsibility $\mathbf{r}_m$ is the *contribution* of the m th module to the ensemble's output $\mathbf{o}(t)$. Setting the responsibility $\mathbf{r}_m$ to zero effectively deactivates the respective module. Previously, similar ensembles of dynamic modules were presented in [21] as a modular Echo-State Network (mESN), where each module consisted of a simple ESN. Here, we show performance of ESMoN with such neural modules as well as with parametric modules, whose behavior is defined by a parameterized equation. Note that other module combinations, including the mixing of neural and parametric ensembles are certainly possible.

Fig. 1 Ensemble of M modules, whose individual outputs $\mathbf{o}_m(t)$ produce the total output $\mathbf{o}(t)$ of the ensemble. The module outputs are weighted with their responsibilities $\mathbf{r}_m$

2.4 Echo state networks

ESNs offer a possibility to encode particular neural dynamics compactly. It has been shown multiple times that only relatively few neurons and low computational effort is needed to model rather complex oscillating dynamics [20, 21, 24]. Moreover, ESNs were shown to be able to generate periodic dynamics for an extended period of time with extremely small error. The abilities of ESNs were further improved by means of modularizations in multiple studies [3, 21, 26]. Figure 2 shows the structure of a standard ESN. It consists of hidden neurons, output neurons, and, optionally, input neurons. Each hidden neuron is recurrently connected with some or all other hidden neurons. Here we neglect input neurons, which may further modify the internal dynamics. As a consequence, the internal recurrences combined with the external recurrences via the output neurons determine the reservoir's dynamics.

The activities $\mathbf{x}(t)$ in the reservoir at time point t determine the reservoir's output:

$$\mathbf{o}(t) = \mathbf{f}_{out}(\mathbf{W}_{out}\mathbf{x}(t)), \quad (4)$$

where $\mathbf{o}(t)$ denotes the ESN output, $\mathbf{x}(t)$ the reservoir's neural activities, $\mathbf{f}_{out}$ the activation functions of the output neurons, and $\mathbf{W}_{out}$ the output weight matrix, which is sequence-optimized directly via:

$$\mathbf{W}_{out} = \mathbf{X}^{-1} \cdot \mathbf{T}, \quad (5)$$

where $\mathbf{X}^{-1}$ denotes the inverted matrix of the reservoir states on the training sequence and $\mathbf{T}$ the corresponding sequence of target outputs.

Besides $\mathbf{W}_{out}$, an ESN contains a reservoir weight matrix $\mathbf{W}_{res}$, an input weight matrix $\mathbf{W}_{in}$, and an output feedback weight matrix $\mathbf{W}_{ofb}$. While $\mathbf{W}_{out}$ is determined during training, the other weights are initialized at random and rely on properly scaled value ranges [15, 20]. Each time step, the reservoir's state activities are updated as follows:

Fig. 2 Structure of the standard Echo-State Network. Small circles depict individual neurons. Dashed lines show weighted connections that are not subject to training. Thick solid lines denote the output connections, which are optimized during training

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}_{in} \cdot \mathbf{u}(t+1) + \mathbf{W}_{res} \cdot \mathbf{x}(t) + \mathbf{W}_{ofb} \cdot \mathbf{y}(t)), \quad (6)$$

where t denotes the current time step, $\mathbf{f}$ the activation function of the reservoir neurons, and $\mathbf{u}(t+1)$ a vector of input values. Note that input and output feedback connections are optional. Here, we only include output feedback connections. More details on the training procedure and properties of standard ESNs can be found elsewhere [15].

3 Evolutionary synchronization of modular networks (ESMoN)

ESMoN co-evolves the activities in its ensemble of dynamic modules $\mathcal{M}$ online in order to accurately predict and identify the components of the unfolding time series composition. Co-evolution is synchronized with a sliding window that moves along the time series. The data from the window is used to compute the fitness of each module's individuals. In order to tune the internal dynamics of the individual modules into the unfolding time series maximally accurately, the respective modules' activities have to synchronize while participating in a co-evolutionary competition for their respective overall utility. We now specify the individual components of the ESMoN algorithm, which tackle this challenge, specifying first the overall objective followed by an algorithmic description of ESMoN and further details on its evolutionary search and permutation routines¹.

¹ The code including the experimental setup used to run the experiments detailed below is available online: <https://github.com/CognitiveModeling/esmon>

3.1 Synchronization objective

Given a time series and a particular set of modules $\mathcal{M}$, ESMoN needs to find module activities that simulate the data with highest possible accuracy. Let us denote the time series target values, which will have a particular synchronization sequence length T_{synch} with $\mathbf{y}(t)$ and the output of the ensemble with $\mathbf{o}(t)$. Synchronization has the objective to minimize the discrepancy between them, denoted as the Euclidean norm:

$$e(t) = ||\mathbf{y}(t) - \mathbf{o}(t)||_2, \quad (7)$$

where $\mathbf{o}(t)$ is determined as a responsibility-weighted sum over the individual modules' outputs $\mathbf{o}_m(t)$ as specified in (3). The challenge for ESMoN is to identify the hidden time series components, their contribution, and their alignment in time. Since the output $\mathbf{o}_m(t)$ of each dynamic module m is a function of the respective time-varying internal states $\mathbf{x}_m(t)$, ESMoN searches for a set of internal states $\mathbf{x}^*(t)$ which minimize the discrepancy (7):

$$\mathbf{x}^*(t) = \arg \min_{\mathbf{x}} e(t). \quad (8)$$

In order to foster convergence, ESMoN restricts the amplitude and internal state activity ranges of each module, as detailed below.

3.2 ESMoN algorithm

ESMoN synchronizes its population via co-evolution while progressing along the time series $\mathbf{y}(t)$. A precise algorithmic description is provided in Algorithm 1.

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

1: Initialize $\{P_1, \dots, P_M\}$	▷ consider the valid ranges (10) and (11)
2: Assign W_t from $\mathbf{y}(1)$ to $\mathbf{y}(T_W)$	▷ sliding window over time series
3: for $t = T_W$ to T_{synch} do	▷ progress along the sequence
4: for $n = 1$ to $N(t)$ do	▷ number of generational optimizations
5: for $m = 1$ to M do	▷ optimizing individual populations
6: Differential Evolution of P_m on W_t	▷ Algorithm 2
7: Permutation of P_m	
8: end for	
9: end for	
10: Advance $\{P_1, \dots, P_M\}$	▷ using (6) or applying a parametric module step
11: Update W_t with $\mathbf{y}(t+1)$	▷ advance the sliding window
12: end for	
13: Find $\mathbf{g}^* = \{\mathbf{p}_{best,1}(T_{synch}), \dots, \mathbf{p}_{best,M}(T_{synch})\}$	

The modules' populations $P_1, \dots, P_M$ are first initialized, assigning uniformly-distributed random values over the corresponding valid ranges. At a particular point in time t , each individual i in population P_m encodes a particular internal state activity $\mathbf{x}_m(t)$ and a responsibility estimate $\mathbf{r}_m(t)$:

$$\mathbf{p}_{mi}(t) = \{\mathbf{r}_m(t), \mathbf{x}_m(t)\}. \quad (9)$$

The valid range of responsibilities $\mathbf{r}_m$ is always $[0, 1]$, where 0 indicates module irrelevance. The internal states have their own valid ranges for each internal state j in each module m :

$$x_{mj} \in [\gamma_{mj}^{\min}, \gamma_{mj}^{\max}], \quad (10)$$

where the limits $\gamma_{mj}^{\min}$ and $\gamma_{mj}^{\max}$ are set to the smallest and largest activation values that were encountered during training in the case of neural modules, or to the naturally valid ranges, e.g. the phase of a sine wave is $\in [0, 2\pi]$, in the case of parametric modules. Similar restrictions are imposed onto the module outputs:

$$o_{mj} \in [\omega_{mj}^{\min}, \omega_{mj}^{\max}], \quad (11)$$

where the valid ranges ω_{mj} are again determined during training similar to γ_{mj} in (10).

At every time step t the algorithm produces $N(t)$ generations for each population and prospectively evaluates the produced offspring on the sliding window W_t of size T_W , starting at current time step t reaching backwards in time. To counteract premature convergence, we linearly increase the number of generations per synchronization window from 1 to $N^{\max}$:

$$N(t) = 1 + N^{\max} \frac{t - T_W}{T_{\text{synch}} - T_W}, \quad (12)$$

where T_{synch} denotes the overall synchronization sequence length and $N^{\max}$ specifies the maximum number of generations. As a result, early on few generations avoid premature convergence, while later a larger number of generations tunes the genotypes even more exactly into the time series dynamics.

While progressing along the sequence, ESMoN optimizes each population P_m , interleaving the differential evolution-based determination of the next generation of an individual population with respective rank-based permutations (lines 5–9 in Algorithm 1). Details of the implemented differential co-evolution are given in Sect. 3.3. Permutation reorders the individuals of a population according to their fitness values, where the best comes first. This reordering does not necessarily lead to instant fitness improvements, but it assures that similarly-potent individuals are mixed while partially fostering diversity. As a result, co-evolutionary dynamics unfold where each population's next generation intricately depends on the evolution of the other populations. Our interleaving method resembles so-termed coordinate descent [33], where the populations play the role of coordinates in the search space. Once all modules' $\mathcal{M}$ underwent $N(t)$ differential evolution generations, both the module states $\mathbf{x}_m(t)$ and the sliding window advance one time step. In the case of the module states, essentially one simulation step is run by means of the respective models (e.g. one ESN-controlled recurrent step is processed).

After reaching the end of the synchronization sequence, ESMoN combines the best individuals $\mathbf{g}^* = \{\mathbf{p}_{\text{best},1}(T_{\text{synch}}), \dots, \mathbf{p}_{\text{best},M}(T_{\text{synch}})\}$ from all populations, yielding the final performance:

$$\bar{e}(\mathbf{g}(t_{\text{Synch}})) = \frac{1}{T_W} \sum_{t \in W_{t_c}} (\mathbf{y}(t) - \mathbf{o}(t))^2. \quad (13)$$

3.3 Differential co-evolution

Differential evolution aims at optimizing the modules' internal states $\mathbf{x}(t)$ and module responsibilities $\mathbf{r}$ with the objective to minimize the average error (13). An algorithmic description of its implementation in ESMoN is provided in Algorithm 2. Each module contains a population P_m of individuals i , each of which specifies a particular internal state $\mathbf{x}_i(t)$ and responsibility $\mathbf{r}_i(t)$ pair, denoting its genotype at time t . Due to the distinct semantic roles of state and responsibility, the differential evolution operations differ, evolving either $\mathbf{r}_i(t)$ with probability P_r (here set to 20%) or $\mathbf{x}_i(t)$ otherwise.

Algorithm 2 Differential Evolution(P_m, W_t)

```

1: for  $p_i$  in  $P_m$  do                                ▷ over individuals
2:   Choose a part of  $\mathbf{p}_i$  for mutation                    ▷  $\mathbf{r}_i$  or  $\mathbf{x}_i$ 
3:   if responsibility is chosen then
4:     Produce  $\mathbf{o}_i$  through mutation of  $\mathbf{r}_i$ 
5:   else
6:     Produce  $\mathbf{o}_i$  through mutation of  $\mathbf{x}_i$ 
7:     Ensure offspring values are in range
8:   end if
9:   Compute performance of  $\mathbf{o}_i$                           ▷ using (17)
10:  Select offspring or keep parent                       ▷  $\mathbf{o}_i$  or  $\mathbf{p}_i$ 
11: end for
  
```

Mutation of responsibilities $\mathbf{r}_i(t)$ aim at potentially disabling irrelevant modules through fast reduction of their amplitudes while fostering local neighborhood search meanwhile. We thus implemented a single parental value-based mutation:

$$\mathbf{r}_i^{\text{offspring}} = \mathbf{r}_i^p(t) \cdot N(1, 1), \quad (14)$$

where $\mathbf{r}_i^p(t)$ is the parental responsibility value and $N(1, 1)$ denotes a normally distributed value with both mean and standard deviation set to 1.

Mutation of the individuals' states $\mathbf{x}_i(t)$ apply proper differential evolution. We apply the “DE/target-to-best/1” mutation scheme:

$$\mathbf{v}_i = \mathbf{p}_i + F \cdot (\mathbf{p}_{\text{best}} - \mathbf{p}_i) + F \cdot (\mathbf{p}_a - \mathbf{p}_b), \quad (15)$$

where $\mathbf{v}_i$ is the mutant individual, $\mathbf{p}_i$ is the direct parent, $\mathbf{p}_{\text{best}}$ is the current best individual in the respective module's population, $\mathbf{p}_a$ and $\mathbf{p}_b$ are randomly chosen and must differ from $\mathbf{p}_{\text{best}}$, F is the differential weight. Note that in contrast to the standard differential evolution rule, this rule includes the best individual to determine the differential gradient, which has been shown to often yield orders of magnitude better final accuracy [24].

Fig. 3 Rank-based fitness computations are calculated by combining the individuals with the same rank index from each population module (here index 4 is highlighted) to compute the combined output prediction. The inverse of the resulting mean squared error determines the fitness

Note that this differential evolution procedure may generate invalid offspring values v_{ij} that lie outside of their respective valid ranges. In this event, the invalid offspring value is replaced by a value that is drawn from a normal distribution with the parental value p_{ij} as its mean and standard deviation:

$$\sigma_{\text{repair}} = \frac{1}{3} \times \min\{|p_{ij} - \gamma_{mj}^{\min}|, |p_{ij} - \gamma_{mj}^{\max}|\}, \quad (16)$$

which focuses the repaired value around the parental value relative to the distance of the closer boundary.

Offspring is evaluated on the sliding window W_t , which has length T_W . Like in all co-evolutionary algorithms, fitness dynamically depends on the paired individuals from the co-evolving modules. We pair individuals fitness-rank-based as shown in Fig. 3. According to Eqs. (1) and (13), we compute the fitness of an individual with rank i in the population of module m as follows:

$$F_{mi}(t) = \frac{T_W}{\sum_{t \in W_t} (\mathbf{y}(t) - \sum_{m \in \{1, \dots, M\}} r_m(t) \mathbf{o}_{mi}(t))^2}, \quad (17)$$

where index i indicates the rank of individuals in the respective population modules m . Note how the fitness $F_{mi}(t)$ actually only depends on the rank. It is equal for all modules $m \in \{1, \dots, M\}$. The assignment, however, is only applied to the population that is currently evolved. Nonetheless, the subsequent reordering of population

P_m (cf. Line 7 in Algorithm 1) indirectly affects the co-evolving fitness values of the individuals in the other populations.

The obtained fitness $F_{mi}(t)$ is then compared to the fitness of the direct parent, replacing the parent if (i) the offspring's fitness is equal or better than the parent's, (ii) the overall output $\sum_{m \in \{1, \dots, M\}} r_m(t) \mathbf{o}_{mi}(t)$ stays in the valid range, and (iii) the error is not steadily increasing over the fitness window, which would correspond to a small but slowly and steadily diverging error. Involving the conditions (ii) and (iii) punishes the individuals that encode instable dynamics, on the one hand. On the other hand, selection of the individuals is not solely dependent on the elitist criterion (i). Randomness in the behavior of the instable recurrent modules in the beginning of synchronization counteracts the elitism. This helps maintaining the diversity in the populations and promotes exploration of the search space.

4 Experiments

We evaluate the algorithm first on one-dimensional time series data. We thereby increase the data complexity by varying the number of basic dynamics as well as by changing their individual characteristics statically and dynamically. Here we focus on the multiple superimposed oscillators challenge [31, 36]. We show that ESMoN is applicable to ensembles of neural modules ($\mathcal{M}_N$) as well as to directly parameterized ensembles ($\mathcal{M}_p$), synchronizing either one well and identifying the present dynamic components. The comparison to the Fast Fourier Transform (FFT) baseline shows that ESMoN performs competitively on mimicking the target sequence but clearly outperforms FFT when autoregressively simulating the unfolding sequence dynamics in a closed-loop manner further into the future. The results thus underline ESMoN's great potential for both approximating and forecasting modular dynamics. Furthermore, ESMoN may generally be applied to other compositions of dynamics. To underline this potential, we finally evaluate ESMoN's performance on two-dimensional time series data.

4.1 Multiple superimposed oscillators

In the one-dimensional case, the target dynamics consisted of linearly combined sine waves, that is, multiple superimposed oscillators (MSO). MSO complexity can be easily modified by varying the number and parameters of the basic oscillations, including their respective amplitudes.

4.1.1 Experimental setup

MSO time series have already been considered as a benchmark problem with different numbers of sine waves in several studies [11, 24, 31, 36]. Despite of their simplicity, MSOs are a challenging problem for typical gradient-based sequence

learners such as fully-connected LSTMs, which tend to fail already at predicting two superimposed waves for more than just a few time steps [25]. Their combined activity is computed by:

$$y(t) = \sum_{k=1}^K a_k \cdot \sin(\lambda_k t + \varphi_k), \quad (18)$$

where K is the number of sine waves; a_k , λ_k , φ_k are their amplitudes, frequencies, and phases, respectively; t is an integer time step. We used the following frequencies, which were considered in earlier studies on MSOs: $\lambda_1 = 0.2$, $\lambda_2 = 0.311$, $\lambda_3 = 0.42$, $\lambda_4 = 0.51$, $\lambda_5 = 0.63$, $\lambda_6 = 0.74$, $\lambda_7 = 0.85$, and $\lambda_8 = 0.97$. Thereat, only λ_1 was present in the easiest time series MSO1. λ_1 and λ_2 were present in MSO2, λ_1 to λ_4 — in MSO4, λ_1 to λ_6 — in MSO6, and all λ 's in the most challenging MSO8. In the standard setting, all amplitudes a_k are equal to unity, which we refer to as *balanced time series*. Unequal amplitudes, where we set $a_k \in [0, 1]$ yielding *imbalanced time series*, increase complexity. Related variations for two components with amplitudes of 0.75 and 2.25 were considered in [2]. In [1], amplitude of a single sine wave was abruptly changed as a random value normally distributed with the mean of 1 and the standard deviation of 0.2. In [24], sine waves had random amplitudes from $[0.1, 1.0]$.

While most related work only considers zero phases ($\varphi_k = 0$), we varied phases randomly in the interval $[0, 2\pi]$ similar to [24]. Each time series was split into 300 synchronization time steps and 300 subsequent test time steps, where autoregressive forecasting unfolded in closed-loop without further input considerations or co-evolutionary adaptations.

Besides the Root Mean Squared Error (RMSE), we also use an “identification rate” (IR) as a performance indicator, which measures the percentage of runs in which the basic oscillations were identified sufficiently correctly:

$$IR = \frac{1}{M} \sum_{m=1}^M \begin{cases} 1, & Y_m = o_m \\ 0, & \text{otherwise} \end{cases}, \quad (19)$$

where Y_m are the basic oscillations, whose presence was supposed to be properly identified by the corresponding module's output activity o_m ². RMSEs and IRs are reported as averages over 100 independent runs.

ESMoN was evaluated using either ensembles of neural modules ($\mathcal{M}_N$) or parametric modules ($\mathcal{M}_P$). In the case of $\mathcal{M}_P$, each module m was defined as a sine function with parameters λ_m and φ_m . The algorithm had to determine these values as well as amplitudes a_m . Each module was responsible for a particular frequency range $[\lambda^{\min}, \lambda^{\max}]$ covering two target frequencies. Accordingly, the first module had the

² The module was seen as active if its contribution to the total output of the ensemble was larger than 1% of the largest value in the synchronization sequence.

Table 1 Performance of ESMoN with $\mathcal{M}_N$ and $\mathcal{M}_P$ with eight-modules on balanced MSOs

Ensemble	Time series	Average test RMSE	Standard deviation	IR (%)	Average deviation of freq.	Average deviation of phase
$\mathcal{M}_N$	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	—	—
$\mathcal{M}_P$	MSO1	3.97×10^{-3}	3.58×10^{-3}	100	2.70×10^{-5}	7.19×10^{-3}
	MSO2	8.38×10^{-3}	7.22×10^{-3}	100	3.62×10^{-5}	8.24×10^{-3}
	MSO4	3.78×10^{-2}	2.45×10^{-2}	100	1.09×10^{-4}	2.69×10^{-2}
	MSO6	7.65×10^{-2}	4.38×10^{-2}	92	1.66×10^{-4}	3.99×10^{-2}
	MSO8	9.88×10^{-2}	4.16×10^{-2}	52	2.02×10^{-4}	4.89×10^{-2}

IR below 100% shows that the overlapping frequency ranges of the parametric modules affected the algorithm's convergence. The not-converged runs produced extremely big errors and were not considered in the accuracy statistics

interval [0.1, 0.35] with the target frequencies 0.2, 0.311; the second—[0.3, 0.45] with 0.311, 0.42, and so on.³

In the case of $\mathcal{M}_N$, the separate dynamics were modeled with compact ESNs of three fully connected reservoir neurons and with no input neurons. While reservoir neurons had tanh activation functions, all output neurons had the identity activation. Each module was randomly generated and pre-trained for one of the basic oscillations using (5) on the sequence of 700 time steps (similar to [20]). Since each neural module was trained only for its own frequency, they were less flexible than the parametric ones.

In the experiments we used 10 for the number of generations N^{max} , 100 for the population size S , and 60 for the width of the sliding window T_w .

4.1.2 Balanced dynamics

In the first series of experiments, ESMoN was evaluated on the balanced time series data. In all evaluations, the ensembles consisted of eight modules, one for each of the eight frequencies λ_i specified above. Table 1 shows the reached accuracy for both dynamic module types. In all runs, the ensembles consisted of eight modules. Thus, on MSO1 ESMoN had to identify the relevant oscillation and deactivate seven irrelevant modules; whereas on MSO8 all available modules were active. Since the contribution from the irrelevant modules prevents the algorithm from reaching higher accuracy, it must detect and deactivate them while, concurrently, improving the performance of the relevant modules.

³ It did not make any difference if non-centered intervals were chosen, as long as the target frequencies were covered by the frequency range of a particular module.

Fig. 4 Exemplary progress of ESMoN on MSO4. ESMoN with $\mathcal{M}_N$ modules (top) deactivates the irrelevant modules (red lines) faster than ESMoN with $\mathcal{M}_P$ modules (bottom) in this case. As seen for the neural module 4 at time step 70, evolving outputs of the neural modules can violate the valid range $[-1, +1]$ at some time steps shortly after the start until EA forces them to stay in the range at the later time steps (Color figure online)

Fig. 5 Dependency of the ESMoN performance on the DE parameters F and CR eight neural modules ($\mathcal{M}_N$) on MSO8 (left), MSO6 (middle) and MSO4 (right). An increase in the number of irrelevant dynamic modules increases the risk of premature convergence. As a result, the area of extreme accuracy with the test RMSE of about 10^{-10} becomes smaller

As can be seen in Table 1, the final accuracy of ESMoN with $\mathcal{M}_N$ does not depend on the complexity of the time series and is restricted only by the accuracy of the available dynamic modules. In contrast, ensembles of the parametric modules reach high accuracy only on the less complex time series. The overlapping frequency ranges of these modules affect their tuning on the most complex MSO8 time series and reduce accuracy of the determined values λ_m and φ_m accordingly.

Table 2 The performance of ESMoN with an ensemble of eight neural or parametric modules ($\mathcal{M}_N$ and $\mathcal{M}_p$, respectively) on imbalanced MSOs

Time series	$\mathcal{M}_N$			$\mathcal{M}_p$		
	Avg test RMSE	Standard deviation	IR (%)	Avg test RMSE	Standard deviation	IR (%)
MSO1	3.52×10^{-5}	2.24×10^{-5}	100	3.39×10^{-3}	2.68×10^{-3}	100
MSO2	5.08×10^{-4}	4.58×10^{-4}	99	1.38×10^{-2}	1.16×10^{-2}	91
MSO4	2.24×10^{-3}	2.47×10^{-3}	99	6.18×10^{-2}	0.110	46
MSO6	2.62×10^{-3}	2.08×10^{-3}	99	0.523	0.358	15
MSO8	2.95×10^{-3}	2.49×10^{-3}	100	0.217	5.95×10^{-2}	6

The varying amplitudes of the basic oscillations made the problem much more complex such that the algorithm had worse convergence (lower IR) than on the balanced time series

Fig. 6 The average error of the neural ensembles on the non-stationary MSO1 (left) and MSO8 (right). As expected, abrupt parameter changes cause large error increases. Effectively, the length of the sliding window reduces the number of the time steps, which are available for synchronization, to 40. On these time steps, the reached accuracy is much higher on MSO1 than on MSO8. The error curves on MSO1 and MSO8 represent the smallest and largest errors, respectively; the error curves on MSO2 and MSO4 would be located in-between

Figure 4 shows exemplary progress on MSO4. While a smooth decay of irrelevant modules can be observed in the case of $\mathcal{M}_N$, parametric module amplitudes behave more erratically. Nonetheless, in both cases irrelevant modules are eventually successfully deactivated, while the relevant ones get in-tune.

Dependencies of the performance on the evolutionary parameters F (differential weight) and CR (crossover probability) are shown in Fig. 5. As can be seen, within the analyzed parameter ranges, the test error varies by several orders of magnitude: from 0.1 at $F=0.1$ to the smallest error of 10^{-10} at the point $F=0.7$ and $CR=0.7$. The area of highly accurate performance decreases with an increasing number of irrelevant dynamic modules, underlining the difficulty to fully deactivate individual modules.

Table 3 The performance on non-stationary MSOs where amplitudes and phases of the basic oscillations were randomly changed every 100 time steps

Time series	$\mathcal{M}_N$			$\mathcal{M}_P$		
	Avg test RMSE	Standard deviation	IR (%)	Avg test RMSE	Standard deviation	IR (%)
MSO1	1.45×10^{-3}	4.19×10^{-3}	94.33	2.05×10^{-3}	3.87×10^{-3}	97.33
MSO2	7.07×10^{-3}	2.96×10^{-2}	89.33	7.00×10^{-3}	2.09×10^{-2}	93.33
MSO4	9.09×10^{-3}	1.70×10^{-2}	87.33	7.11×10^{-2}	0.108	49.66
MSO6	1.05×10^{-2}	1.57×10^{-2}	89	0.176	0.180	2.66
MSO8	1.29×10^{-2}	1.30×10^{-2}	92.33	0.441	0.374	1

Since the not-converged runs could not be clearly identified, the error was computed on all 100 runs. We assumed that the algorithm converges shortly before the next parameter change and, therefore, computed the statistics from the errors averaged over the last 10 time steps before a change

Table 4 The errors and the sizes of different models on the MSO data. Except ESMoN and ESN-DE, most approaches only considered balanced MSO

Approach	Test RMSE			Size		
	MSO2	MSO4	MSO8	MSO2	MSO4	MSO8
ESMoN with $\mathcal{M}_N$ (EA; on balanced)	1.38×10^{-6}	1.43×10^{-6}	1.86×10^{-6}	8	16	32
ESMoN with $\mathcal{M}_N$ (EA; on imbalanced)	5.08×10^{-4}	2.24×10^{-3}	2.95×10^{-3}	8	16	32
Balanced ESN [20] (non-EA)	2.52×10^{-12}	8.15×10^{-8}	5.50×10^{-4}	5	9	68
Filter ESNs [11] (non-EA)	1.86×10^{-7}	1.42×10^{-5}	—	100	100	—
Evolino [31] (EA)	4.17×10^{-3}	0.157	—	40	80	—
ESN-DE [24] (EA; on balanced)	$\approx 10^{-13}$	$\approx 10^{-10}$	1.24×10^{-7}	20	20	40
ESN-DE [24] (EA; on imbalanced)	—	—	min 2.56×10^{-7} max 8.93×10^{-4}	—	—	40
DESN [36] (non-EA)	1.73×10^{-2}	—	—	400	—	—

The evolutionary approaches (EA) are computationally more intensive than the non-evolutionary (non-EA) ones

4.1.3 Imbalanced dynamics

As a second set of analyses, we varied the amplitudes individually for each basic oscillation. Table 2 shows that the reached accuracy is still high, albeit lower than those reached for the balanced dynamics. Remarkable is that the neural modules provided more accurate synchronization than the parametric ones. This is due to

Fig. 7 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. Note the significant deviations of the computed amplitudes from their target values of 1 even at the window samples size of 300 (Color figure online)

higher flexibility of the neural modules whose responsibility $\mathbf{r}_m$ and the variable amplitude of the own signal $\mathbf{o}_m(t)$ gradually co-adapt. In contrast, for the parametric modules with the constant amplitude of “1”, the small errors are observed only for close-to-optimal responsibilities such that the algorithm can easily get stuck to a local optimum. In Table 2 this is seen as a fast degradation of the error with an increasing number of the basic oscillations, while it stays quite stable when neural ensembles $\mathcal{M}_N$ are used.

4.1.4 Non-stationary dynamics

Non-stationarity, i.e. changing internal parameters of the process over time, is an inherent characteristic of most real-world applications. Here, we consider abrupt changes of amplitudes and phases every 100 time steps. Thereat, the algorithm was not informed about the changes.

Figure 6 shows the typical error behavior. After the start and after every change, the error is steadily decreasing until the next change. The reached value is small and depends on complexity of the time series. On the simpler dynamics, the algorithm needs to re-tune fewer values and, therefore, reaches smaller errors.

Fig. 8 Average power spectra with sliding windows of length 60 over 300 samples for balanced MSO4 (left) and balanced MSO8 (right). The dotted red vertical lines indicate the true frequencies within the respective time series (Color figure online)

Fig. 9 FFT-based signal reconstructions (first 300 time steps) and predictions (last 300 time steps). In the two top rows the full spectral range was used. For the two bottom rows only the known frequencies of the respective MSO dynamics were considered. The dashed curves indicate the ground truth, the blue curves represent the FFT-based signals (Color figure online)

Table 5 The average run time per sequence and the total number of tuned values in the populations at the start and at the end of the sequence

Ensemble	Data Type	MSO1		MSO8	
		Time (s)	Number of values (at start/at end)	Time (s)	Number of values (at start/ at end)
$\mathcal{M}_N$	Balanced	105.21	(25600 / 3200)	722.21	(25600/25600)
	Imbalanced	106.66	(25600 / 3200)	400.18	(25600/25600)
	Non-Stationary	669.45	(25600 / 25600)	710.53	(25600/25600)
$\mathcal{M}_P$	Balanced	71.29	(19200 / 2400)	292.57	(19200/19200)
	Imbalanced	67.20	(19200 / 2400)	258.48	(19200/19200)
	Non-Stationary	233.81	(19200 / 19200)	259.20	(19200/19200)

On the balanced and imbalanced MSO1, the time is shorter as the algorithm stops tuning the superfluous modules and thus reduces the number of tuned values. On MSO8, occasional misidentifications of the superfluous modules shorten the time. On the non-stationary data, all populations are evolved until the end of the sequence; hence, the time does not depend on complexity of the time series. Synchronization of $\mathcal{M}_P$ is faster because each ensemble has 24 tuned values vs. 32 in $\mathcal{M}_N$. The number of tuned values is given for the population size 100 and the ensemble size 8. The runs were done on PC with Intel Core processor i7-9750H with 16Gb RAM

Overall performance is shown in Table 3. As can be seen, the error is increasing with time series complexity. But IR behaves differently for $\mathcal{M}_N$ —it is the worst on the MSO4 sequences of medium complexity. On such time series, ESMoN yields the highest initial uncertainty about the relevant modules.

4.1.5 Performance comparisons

Table 4 summarizes the performance and the RNN size for ESMoN and several other studies of RNNs on the MSO time series⁴. Note that standard LSTMs yield a test RMSE of .980 for MSO2 and are thus not considered any further in this work [25].

Similar to our approach, DESN [36] is modular and Evolino [31] is based on co-evolution. But their performance and especially the network size is inferior to those of ESMoN. Among the studies, ESMoN and the optimized monolithic ESN-DE show superior performance on the more complex dynamics. Thus, evolutionary model parameter optimization in the monolithic RNN [24] and the co-evolution of the modules' activities as well as the integration of a priori knowledge about target dynamics in our modular networks have the strongest positive effect on the resulting accuracy.

Given the nature of the MSO time series, it seems reasonable to contrast the ESMoN approach with traditional frequency transformation techniques such as FFT. The exemplar spectra of MSO4 and MSO8 (Fig. 7) show that the resulting

⁴ Note that we report the number of tunable parameters (number of neurons plus responsibility) as the respective modules' sizes.

Fig. 10 Exemplary closed-loop signal predictions (last 300 time steps) with ESMoN after 300 time steps with ESN modules (two top rows) and sine modules (two bottom rows). The dashed curves indicate the ground truth, the blue curves represent the FFT-based signals (Color figure online)

frequency resolution is insufficient for exact frequency identification when considering the window of 60 samples. FFT requires more samples in order to form higher resolved and more accurate spectra. But even with up to 300 samples (which cover the full amount of data that ESMoN requires for full convergence with the sliding window of 60 time steps), the peaks do neither exactly align with the true frequencies nor do they allow deducing the correct amplitudes of the involved basic oscillations directly. This is also the case when computing the average spectra density based on a sliding window of length 60 over 300 samples with hop size 1 (cf. Fig. 8).

Table 6 Percentage of the converged runs with eight parametric modules $\mathcal{M}_p$ whose frequency ranges overlap to a different extent

Data type	Time series	No Overlap	Overlap with 1 shared frequency	Overlap with 2 shared frequencies
Balanced	MSO1	100%	100%	100%
	MSO2	100%	100%	100%
	MSO4	100%	100%	100%
	MSO6	100%	92%	27%
	MSO8	100%	52%	32%
Imbalanced	MSO1	100%	100%	100%
	MSO2	94%	91%	91%
	MSO4	62%	46%	53%
	MSO6	50%	15%	3%
	MSO8	23%	6%	2%
Non-Stationary	MSO1	98.66%	97.33%	98%
	MSO2	94.33%	93.33%	92.66%
	MSO4	64.66%	49.66%	40.66%
	MSO6	6.33%	2.66%	0.33%
	MSO8	6%	1%	1%

More shared target frequencies lead to weaker separation between the modules in the search space and result in significant weakening of convergence on MSO8

Fig. 11 The error curves of ESMoN with eight parametric modules ($\mathcal{M}_p$, solid lines) and neural modules ($\mathcal{M}_N$, dashed lines) from the runs on the balanced MSO1 and MSO8. The parametric modules have slower but steady convergence. In the neural modules, tuning the recurrent dynamics is accompanied with their instability which causes big variations in the error convergence curves. For both module types, the confidence intervals are steadily decreasing over the sequence

In Fig. 9 exemplary FFT-based reconstructions and predictions are presented for MSO4 and MSO8. From the first 300 time steps of a time series the spectrum was calculated using the FFT.

This becomes even more evident with the FFT-based predictions shown in Fig. 9. The superimposition of all sine waves that are obtained from the FFT spectrum (150

non-redundant frequencies on the time window of 300 samples), results in a perfect reconstruction of the signal within the time window (RMSE of 3.62×10^{-16} and 4.96×10^{-16} for MSO4 and MSO8, respectively). However, the FFT-based predictive performance is very low: continuing these waves for the next 300 time steps does not generate a useful signal prediction (RMSE of 2.01 and 2.98 for MSO4 and MSO8, respectively). This also holds true when only the known frequencies of the respective MSO dynamics are considered.

Seeing that ESMoN and FFT are not really comparable, because ESMoN can in principle be applied to any type of modular time series, this investigation underlines the strength of ESMoN to quickly converge towards a clean and accurate signal decomposition given relatively few samples.

Despite its performance advantages, though, it should be noted that, like other EAs, ESMoN is computationally relatively expensive. As can be seen in Table 5, the running time varies with the complexity of the task. The high computational load lies in the relatively big number of fitness evaluations. For the chosen parameter values, the estimated number of fitness evaluations is about 105,000 per sequence on MSO8, which needs all eight available modules. The most relevant parameters to reduce the running time are the modules' population sizes, the number of generations N^{max} , as well as the size of the individual modules.

4.1.6 Convergence

As shown in [30], convergence of EAs is guaranteed only in the probabilistic sense over an infinite period of time. Given the limited synchronization sequence and the limited number of fitness evaluations, increasing complexity reduces the probability of convergence to the global optimum. Table 6 illustrates this dependency for an increasing number of the basic oscillations when the modules' frequency ranges overlap to a different extent. Interestingly, the larger overlap has a significant effect on convergence only on MSO6 and MSO8 (Fig. 10).

Furthermore, Fig. 11 shows convergence with the modules of both types on the time series of different complexity. The parametric modules provide steady although slower convergence. This slowness is due to higher universality of the parametric modules, which maintain a range of frequencies unlike a single frequency per neural module. This is also valid for the confidence interval which is shrinking for them noticeably slower as seen on the more complex MSO8. The inherent instability of the recurrent neural modules makes their error convergence curve somewhat rough. The biggest variation of the curve is observed on the simpler MSO1 sequence where the error jumps by orders of magnitude shortly before the dynamics reach the global attractor.

4.2 2D dynamics

The results for MSOs show that the ESMoN is very well suited to either tune parameters or latent neural activities for approximating time series dynamics online. In fact, it was possible to reliably identify the involved components and

Table 7 The performance on the 2D curves

Noise level	Ellipse			8-curve		
	Avg test RMSE	Standard deviation	IR (%)	Avg test RMSE	Standard deviation	IR (%)
0	1.31×10^{-2}	2.53×10^{-3}	100	4.16×10^{-2}	5.76×10^{-2}	88
10^{-6}	1.33×10^{-2}	4.19×10^{-3}	99	3.61×10^{-2}	4.50×10^{-2}	81
10^{-3}	6.40×10^{-3}	1.73×10^{-3}	98	1.18×10^{-2}	1.52×10^{-2}	76
0.05	3.24×10^{-2}	5.16×10^{-3}	49	8.21×10^{-2}	7.80×10^{-2}	30

For each strength of noise, the performance stays higher on the simpler ellipse curve. In contrast, the resulting accuracy and IR are somewhat lower for the more complex 8-curve

tune in the relevant modules concurrently. However, it may be argued that MSOs are rather regular and can also be analyzed with standard techniques, such as FFT.

To evaluate the more general applicability of ESMoN, we conducted further experiments on two-dimensional trajectories. 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. These experiments were conducted on noiseless as well as on noisy sequences.

4.2.1 Experimental setup

We considered two types of 2D dynamic modules. One module was trained to generate an ellipse, the other one to draw an 8-curve, which is non-trivial for RNNs [35]. The training and test sequences of both dynamic trajectories were generated for different angular orientations with the random angles uniformly distributed in the range between $\frac{\pi}{4}$ and $\frac{3}{4}\pi$. We considered four sets of sequences: noise-free as well as noisy with noise strengths of 10^{-6} , 10^{-3} , and 0.05. In all sequences, coordinates of the curves were normalized to the interval $[-0.5, +0.5]$. More detail on the ellipse and 8-curve dynamics can be found in Appendix A.

Each module was realized by an ESN reproducing periodic dynamics in the X-Y coordinate plane, i.e. its outputs were interdependent periodic functions of time. The modules were run in the auto-generation mode and received the orientation angle as its input. The module outputs were scaled with responsibilities and summed.

To prepare the modules, 450,000 ESNs were randomly generated for different combinations of the ESN parameters such as the reservoir size, $\mathbf{W}_{in}$ and $\mathbf{W}_{ofb}$. Every generated ESN was trained using (5) on 100 training sequences — each sequence for a certain orientation angle. The sequences were split into 50 washout, 300 training and 250 test time steps. For the runs with ESMoN, we chose the ESNs that provided the smallest average test error over the considered range of orientation angles. Their reservoirs had 7 and 9 neurons for the ellipse and the 8-curve, respectively. Two ESN modules were prepared on the sequences at each noise level, except for

Fig. 12 Examples of successful (upper row) and mediocre (lower row) synchronization on the ellipse (left column) and 8-curve (right column) at noise strength of 0.05. The green dots mark the test time series and the red ones the output of ESMoN's ensemble of neural dynamics $\mathcal{M}_N$. ESMoN effectively filters out noise and matches the time series, though slightly loses track in the lower row (Color figure online)

level 0.05. At this noise strength, none of the generated ESNs could reproduce the 8-curve. Therefore, on the synchronization sequences with noise strength of 0.05, we used the ESN modules that were trained with the weaker noise of 10^{-3} .

The evolutionary search needed longer chromosomes $\mathbf{p}_{mi}$ than those defined in the expression (9) for the one-dimensional time series. Every individual was encoded as $\mathbf{p}_{mi} = \{r, X, Y, \mathbf{x}_m(t), \alpha\}$ with responsibility r of the module outputs, and X- and Y-coordinates for the center of the 2D dynamics, $\mathbf{x}_m(t)$ the latent activity states, and also the orientation angle α . To evaluate the individuals, the average error $\bar{e}$ was computed on the sliding window separately for the X- and Y-coordinates. The largest of both error values was then used in (1). ESMoN was run on 100 sequences of each type of 2D dynamic. Each sequence was split into 300 synchronization time steps and 300 test time steps.

4.2.2 Results

Table 7 shows the performance on the 2D curves. In general, the differences between the experimental data on both curves reflect the differences in their complexity — lower complexity of the ellipse favors its more precise and reliable identification

with a smaller standard deviation for the same level of noise. As can be seen, noise has some positive effect on accuracy of the converged runs as accuracy is slightly better on the sequences with noise 10^{-3} than on the noise-free data. Nevertheless, stronger noise increases the risk of premature convergence and affects identification rate for both curves where worsening of identification rate goes significantly faster for the more complex 8-curve.

Figure 12 shows examples of successful and mediocre synchronization. The output of the well-synchronized module approximately focuses on the middle of the area, which is covered by the noisy sequence points. This means that the algorithm effectively filters out the noise while minimizing the error of reconstruction. In the other case, despite lower accuracy, the algorithm provided correct identification of both considered target curves.

5 Discussion

The conducted experiments have shown ESMoN's potential but also current challenges. The evolutionary nature of the method makes it generally applicable, because it does not rely on differentiable predictive models or a convex optimization landscape. ESMoN allows encodings of basic oscillations by any model type, such as neural network modules, parametric modules, or even module combinations. Our results have shown that generative, autoregressive neural network modules can yield additional stabilization benefits, improving stability and accuracy of the online tuning process. We believe that this is the case because the neural module's activities tend towards latent attractor states, which are encoded in their hidden state dynamics. Critically, any included module should be designed as an expert for generating a particular type of time series dynamics—possibly considering a range of variations thereof.

The results have furthermore shown that relatively little data is needed for convergence. In many cases ESMoN converged on rather short sequences and the obtained reconstruction performance was competitive even with respect to well-established techniques, such as FFT. FFT is mainly applicable to a sufficiently long sequence of time series dynamics with stationary components for frequencies identification and signal reconstruction. It fails for prediction purposes in the MSO context. In contrast, ESMoN is even applicable to dynamically changing time series dynamics and those that contain other, possibly non-periodic, dynamic regularities.

The additional capacity and adaptive flexibility of ESMoN comes with additional computational overhead for maintaining and processing multiple candidate neural activity distributions across the different modules in parallel. Thereat, synchronization of smaller modules is less demanding. Thus, as long as the number of modules' tunable states stays reasonably small, ESMoN will be applicable.

Another consideration lies in the fact that ESMoN's modules are essentially required to be able to model the available signals in the data. Thus, a priori knowledge about the problem is needed and more accurately-tuned modules will foster convergence and adaptation speed. Additionally, as shown in Table 6, better separation between the valid ranges of the module parameters reduces the risk of

premature convergence. Hence, we think that the presence of a priori information makes ESMoN particularly useful for on-the-fly identification of modular dynamics in the sequential data.

6 Conclusions and outlook

In this paper we presented evolutionary synchronization of modular networks (ESMoN) as an algorithm for modeling time series with a compact modularized architecture, which consists of simple dynamic modules. ESMoN was evaluated on time series of different complexity including standard, imbalanced, and non-stationary one-dimensional benchmark sequences, as well as two-dimensional time series. The algorithm identified the basic components of the applied time series and determined their parameters. Analyses of the resulting models confirmed their compactness and competitive accuracy.

In order to show its generality, ESMoN was used with different module types. In combination with the neural modules, it resembles known neuroevolutionary approaches. Moreover, ESMoN adds another facet to neuroevolution where the internal dynamics are evolved, instead of the weights or topology of a neural network [32]. Our algorithm is able to explain the present components of time series data by determining their respective frequencies, phases, and amplitudes online. The results also emphasize the importance of choosing a suitable module type with respect to the primary purpose of the application.

The challenge to synchronize non-linear combinations of the dynamic modules may need to be met when aiming towards real-world processes, where the individual data sources may interact non-linearly. For non-stationary data, the algorithm may be complemented with an event detector, which could monitor prediction error dynamics over time [9]. We expect that such a mechanism will enable much quicker adaptations to changes in the time series composition.

ESMoN thus has potential for applications in domains with modular dynamics, for example, for hand-written text analysis, source extraction in audio files, or for robotics. In the experiments on the 2D time series, the algorithm identified the 2D curves that are similar to hand-written symbols. It might also be able to identify the individual sources in audio files. Finally, in robotics, modularity of complex patterns is inherent in trajectories that are generated by central pattern generators (CPG), whereby complex robot movements can be encoded by a combination of simple periodic oscillations [34]. Regardless of the chosen application, ESMoN extends the basic techniques of time series analysis with the ability to identify individual data sources, including their parameterization, and their currently unfolding dynamic activity online by means of co-evolution.

Appendix 1: 2D dynamic curves

The ellipse is a periodic curve with the points (x, y) being computed as:

$$\begin{cases} x(\alpha) = a \cdot \cos(\alpha + \phi_x) \\ y(\alpha) = b \cdot \sin(\alpha + \phi_y) \end{cases}, \quad (20)$$

where α is the angle of a point (x, y) with respect to the X-axis going from 0 to 2π ; and ϕ_x and ϕ_y are the phases. The coefficients a and b are the major and minor semi-axes. Every point (x, y) corresponded to a certain time step in the sequences that were generated as a concatenation of several periods. Each period had length L of 300 points, which were computed while the angle α was changing from 0 to 2π with the step $\frac{2\pi}{L}$. The phases ϕ_x and ϕ_y were set to 0. In order not to get too stretched or too circular ellipse, the semi-axes a and b were set to 1 and 0.528, respectively.

The periodic 8-curve was defined as the Booth lemniscate [19]. Similarly to the ellipse, coordinates of its points (x, y) depend on the angle α as

$$\begin{cases} x(\alpha) = \rho \cdot \cos(\alpha) \\ y(\alpha) = \rho \cdot \sin(\alpha) \end{cases} \quad (21)$$

with

$$\rho^2 = |2m + n| \cdot \cos^2(\alpha) - |2m - n| \cdot \sin^2(\alpha), \quad (22)$$

where α is the angle of a point (x, y) with respect to the X-axis going from 0 to 2π , and m and n are coefficients of the Booth lemniscate. The 8-curve sequences were also concatenation of several periods. Each period consisted of 300 points. In order to produce the 8-curve smoothly approaching the point $(0, 0)$, the coefficients m and n were set to 1 and 40, respectively. Finally, the curves were rotated at the point $(0, 0)$ by the orientation angle ϕ via converting the coordinates of every point (x, y) to (x', y') as follows.

$$\begin{cases} x' = x \cdot \cos(\phi) - y \cdot \sin(\phi) \\ y' = x \cdot \sin(\phi) + y \cdot \cos(\phi) \end{cases} \quad (23)$$

For the sequence generation, ϕ was varied in the range between $\frac{\pi}{4}$ and $\frac{3}{4}\pi$.

Acknowledgements Martin Butz is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. Additional funding from the German Research Foundation (DFG) within Priority-Program SPP 2134 - project “Development of the agentive self” (BU 1335/11-1, EL 253/8-1) is acknowledged.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. A. Ahmadi, J. Tani, How can a recurrent neurodynamic predictive coding model cope with fluctuation in temporal patterns? robotic experiments on imitative interaction. *Neural Netw.* **92**, 3–16 (2017)
2. F. Azam, Biologically inspired modular neural networks. Ph.D. thesis, Virginia Polytechnic Institute (2000)
3. S. Babinec, J. Pospichal, Modular echo state neural networks in time series prediction. *Comput. Inform.* **30**(2), 321–334 (2011)
4. S. Bengio, F. Fessant, D. Collobert, Use of modular architectures for time series prediction. *Neural Process. Lett.* **3**, 101–106 (1996)
5. K. Byadarhaly, M. Perdoor, A. Minai, A modular neural model of motor synergies. *Neural Netw.* **32**, 96–108 (2012)
6. A.X. Carvalho, M.A. Tanner, Modeling nonlinearities with mixtures-of-experts of time series models. *Int. J. Math. Math. Sci.* 019423 (2006)
7. S. Das, P.N. Suganthan, Differential evolution: A survey of the state-of-the-art. *IEEE Trans. Evol. Comput.* **15**(1), 4–31 (2011)
8. F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.* **9**, 937–965 (2008)
9. C. Gumbsch, M.V. Butz, G. Martius, Autonomous identification and goal-directed invocation of event-predictive behavioral primitives. *IEEE Trans. Cogn. Dev. Syst.* **13**(2), 298–311 (2019)
10. B.L.M. Happel, J.M.J. Murre, The design and evolution of modular neural network architectures. *Neural Netw.* **7**(6–7), 985–1004 (1994)
11. G. Holzmänn, H. Hauser, Echo state networks with filter neurons and a delay&sum readout. *Neural Netw.* **23**(2), 244–256 (2010)
12. M. Hüsken, C. Igel, M. Toussaint, Task-dependent evolution of modularity in neural networks. *Connect. Sci.* **14**, 219–229 (2002)
13. R.A. Jacobs, M.I. Jordan, A.G. Barto, Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cogn. Sci.* **15**(2), 219–250 (1991)
14. R.A. Jacobs, M.I. Jordan, S.J. Nowlan, G.E. Hinton, Adaptive mixtures of local experts. *Neural Comput.* **3**(1), 79–87 (1991)
15. H. Jaeger, A tutorial on training recurrent neural networks, covering bppt, rtll, ekf and the “echo state network” approach. GMD Report 159, German National Research Center for Information Technology (2002)
16. H. Jaeger, Discovering multiscale dynamical features with hierarchical echo state networks. Tech. Rep. 10, Jacobs University Bremen (2007)
17. H. Jaeger, H. Haas, Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science* **304**, 78–80 (2004)
18. T. Jansen, R.P. Wiegand, Exploring the explorative advantage of the cooperative coevolutionary (1+1) ea. In: GECCO’03 Proceedings of the 2003 international conference on Genetic and evolutionary computation: Part I, pp. 310–321. Springer-Verlag Berlin, Heidelberg (2003)
19. R. Kargar, A. Ebadian, J. Sokol, On booth lemniscate and starlike functions. *Anal. Math. Phys.* **9**, 143–154 (2017)
20. D. Koryakin, J. Lohmann, M.V. Butz, Balanced echo state networks. *Neural Netw.* **36**, 35–45 (2012)
21. D. Koryakin, F. Schrödt, M.V. Butz, Ensembles of neural oscillators. In: *New Challenges in Neural Computation*, pp. 57–64 (2015)
22. Q.L. Ma, W.B. Chen, Modular state space of echo state network. *Neurocomputing* **122**, 406–417 (2013)
23. D.E. Moriarty, R. Miikkulainen, Forming neural networks through efficient and adaptive coevolution. *Evol. Comput.* **5**(4), 373–399 (1997)
24. S. Otte, M.V. Butz, D. Koryakin, F. Becker, M. Liwicki, A. Zell, Optimizing recurrent reservoirs with neuro-evolution. *Neurocomputing* **192**, 128–138 (2016)
25. S. Otte, P. Rubisch, M.V. Butz, Gradient-based learning of compositional dynamics with modular rnns. *Artif. Neural Netw. Mach. Learn. ICANN* **2019**(11727), 484–496 (2019)
26. J. Pathak, B. Hunt, M. Girvan, Z. Lu, E. Ott, Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Phys. Rev. Lett.* **120**, 1–5 (2018)

27. J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. Hunt, M. Girvan, E. Ott, Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model. *Chaos Interdisciplinary J. Nonlinear Sci.* **28**(4). (2018)
28. M.A. Potter, K.A. DeJong, Cooperative coevolution: An architecture for evolving coadapted sub-components. *Evol. Comput.* **8**(1), 1–29 (2000)
29. J. Reisinger, K.O. Stanley, R. Miikkulainen, Evolving reusable neural modules. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)* pp. 69–81 (2004)
30. G. Rudolph, Convergence of evolutionary algorithms in general search spaces. *Proceedings of the IEEE Conference on Evolutionary Computation*, pp. 50–54 (1996)
31. J. Schmidhuber, D. Wierstra, M. Gagliolo, F. Gomez, Training recurrent neural networks by evoluno. *Neural Comput.* **19**(3), 757–779 (2007)
32. A. Soltoggio, K. Stanley, S. Risi, Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Netw.* **108**, 48–67 (2018)
33. S. Wright, Coordinate descent algorithms. *Math. Program.* **151**, 3–34 (2015)
34. F. Wyffels, B. Schrauwen, Design of a central pattern generator using reservoir computing for learning human motion. In: *Advanced Technologies for Enhanced Quality of Life*, pp. 118–122. IEEE (2009)
35. F. Wyffels, B. Schrauwen, D. Stroobandt, Stable output feedback in reservoir computing using ridge regression, in *ICANN*. ed. by V. Kurkova, R. Neruda, J. Koutnik (Springer, Berlin, 2008), pp. 808–817
36. Y. Xue, L. Yang, S. Haykin, Decoupled echo state networks with lateral inhibition. *Neural Netw.* **20**(3), 365–376 (2007)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.