Backpropagation algorithms and Reservoir Computing in Recurrent Neural Networks for the forecasting of complex spatiotemporal dynamics

P.R. Vlachas a, J. Pathak b,c, B.R. Hunt d,e, T.P. Sapsis f, M. Girvan b,c,d, E. Ott b,c,g, P. Koumoutsakos a,*

a Computational Science and Engineering Laboratory, ETH Zürich, Clausiusstrasse 33, Zürich CH-8092, Switzerland
b Institute for Research in Electronics and Applied Physics, University of Maryland, College Park, MD 20742, USA
c Department of Physics, University of Maryland, College Park, MD 20742, USA
d Department of Mathematics, University of Maryland, College Park, MD 20742, USA
e Department of Physical Science and Technology, University of Maryland, College Park, MD 20742, USA
f Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139-4307, USA
g Department of Electrical and Computer Engineering, University of Maryland, MD 20742, USA

A R T I C L E   I N F O

Article history:
Received 8 October 2019
Received in revised form 17 February 2020
Accepted 24 February 2020
Available online 21 March 2020

Keywords:
Time series forecasting
RNN, LSTM, GRU
Reservoir Computing
Kuramoto–Sivashinsky
Lorenz-96
Complex systems

A B S T R A C T

We examine the efficiency of Recurrent Neural Networks in forecasting the spatiotemporal dynamics of high dimensional and reduced order complex systems using Reservoir Computing (RC) and Backpropagation through time (BPTT) for gated network architectures. We highlight advantages and limitations of each method and discuss their implementation for parallel computing architectures. We quantify the relative prediction accuracy of these algorithms for the long-term forecasting of chaotic systems using as benchmarks the Lorenz-96 and the Kuramoto–Sivashinsky (KS) equations. We find that, when the full state dynamics are available for training, RC outperforms BPTT approaches in terms of predictive performance and in capturing of the long-term statistics, while at the same time requiring much less training time. However, in the case of reduced order data, large scale RC models can be unstable and more likely than the BPTT algorithms to diverge. In contrast, RNNs trained via BPTT show superior forecasting abilities and capture well the dynamics of reduced order systems. Furthermore, the present study quantifies for the first time the Lyapunov Spectrum of the KS equation with BPTT, achieving similar accuracy as RC. This study establishes that RNNs are a potent computational framework for the learning and forecasting of complex spatiotemporal systems.

© 2020 Elsevier Ltd. All rights reserved.

1. Introduction

In recent years we have observed significant advances in the field of machine learning (ML) that rely on potent algorithms and their deployment on powerful computing architectures. Some of these advances have been materialized by deploying ML algorithms on dynamic environments such as video games (Ha & Schmidhuber, 2018; Schrittwieser et al., 2019) and simplified physical systems (AI gym) (Brockman et al., 2016; Mnih et al., 2015; Silver et al., 2016). Dynamic environments are often encountered across disciplines ranging from engineering and physics to finance and social sciences. They can serve as bridge for scientists and engineers to advances in machine learning and at the same time they present a fertile ground for the development and testing of advanced ML algorithms (Hassabis, Kumaran, Summerfield, & Botvinick, 2017). The deployment of advanced machine learning algorithms to complex systems is in its infancy. We believe that it deserves further exploration as it may have far-reaching implications for societal and scientific challenges in fields ranging from weather and climate prediction (Gneiting & Raftery, 2005; Weyn, Durran, & Caruana, 2019), medicine analytics (Esteva et al., 2017; Kurth et al., 2018), to modeling ocean dynamics and turbulent flows (Aksamit, Sapsis, & Haller, 2019; Brunton, Noack, & Koumoutsakos, 2020).

Complex systems are characterized by multiple, interacting spatiotemporal scales that challenge classical numerical methods for their prediction and control. The dynamics of such systems are typically chaotic and difficult to predict, a critical issue in problems such as weather and climate prediction. Recurrent Neural Networks (RNNs), offer a potent method for addressing
these challenges. RNNs were developed for processing of sequential data, such as time series (Hochreiter & Schmidhuber, 1997), speech (Graves & Jaitly, 2014), and language (Cho et al., 2014; Dong, Wu, He, Yu, & Wang, 2015). Unlike classical numerical methods that aim at discretizing existing equations of complex systems, RNN models are data driven. RNNs keep track of a hidden state, that encodes information about the history of the system dynamics. Such data-driven models are of great importance in applications to complex systems where equations based on first principles may not exist, or may be expensive to discretize and evaluate, let alone control, in real-time.

Early application of neural networks for modeling and prediction of dynamical systems can be traced to the work of Lapedes and Farber (1987), where they demonstrated the efficiency of feedforward artificial neural networks (ANNs) to model deterministic chaos. In (Krischer, Rico-Martínez, Kevrekidis, Rotermund, Erll, & Hudson, 1993), ANNs were utilized to forecast the principal components of a spatiotemporal catalytic reaction. As an alternative to ANNs, wavelet networks were proposed in Cao, Hong, Fang, and He (1995) for chaotic time series prediction. However, these works have been limited to intrinsically low-order systems, and they have been often deployed in conjunction with dimensionality reduction tools. As shown in this work, RNNs have the potential to overcome these scalability problems and be applied to high-dimensional spatio-temporal dynamics. The works of Takens (1981) and Sauer, Yorke, and Casdagli (1991) showed that the dynamics on a D-dimensional attractor of a dynamical system can be unfolded in a time delayed embedding of dimension greater than 2D. The identification of a useful embedding and the construction of a forecasting model have been the subject of life-long research efforts (Bradley & Kantz, 2015). More recently, in Lusch, Kutz, and Brunton (2018), a data-driven method based on the Koopman operator formalism (Koopman, 1931) was proposed, using feed-forward ANNs to identify an embedding space with linear dynamics that is then amenable to theoretical analysis.

There is limited work at the interface of RNNs and non-linear dynamical systems (Lu, Hunt, & Ott, 2018; Pathak, Hunt, Girvan, Lu, & Ott, 2018; Pathak, Lu, Hunt, Girvan, & Ott, 2017; Vlachas, Byeon, Wan, Sapsis, & Koumoutsakos, 2018; Wan, Vlachas, Koumoutsakos, & Sapsis, 2018). Here we examine and compare two of the most prominent nonlinear techniques in the forecasting of dynamical systems, namely RNNs trained with backpropagation and Reservoir Computing (RC). We note that our RC implementation also uses a recurrent neural network, but according to the RC paradigm, it does not train the internal network parameters. We consider the cases of fully observed systems as well as the case of partially observed systems such as reduced order models of real world problems, where typically we do not have access to all the degrees-of-freedom of the dynamical system.

Reservoir Computing (RC) has shown significant success in modeling the full-order space dynamics of high dimensional chaotic systems. This success has sparked the interest of theoretical researchers that proved universal approximation properties of these models (Gonon & Ortega, 2019; Grigoryeva & Ortega, 2018). In Pathak et al. (2017), Pathak, Wikner et al. (2018) RC is utilized to build surrogate models for chaotic systems and compute their Lyapunov exponents based solely on data. A scalable approach to high-dimensional systems with local interactions is proposed in Pathak, Hunt et al. (2018). In this case, an ensemble of RC networks is used in parallel. Each ensemble member is forecasting the evolution of a group of modes while all other modes interacting with this group is fed at the input of the network. The model takes advantage of the local interactions in the state–space to decouple the forecasting of each mode group and improve the scalability.

RNNs are architectures designed to capture long-term dependencies in sequential data (Bengio, Simard, & Frasconi, 1994; Goodfellow, Bengio, & Courville, 2016; Hochreiter, 1998; Pascanu, Mikolov, & Bengio, 2013). The potential of RNNs for capturing temporal dynamics in physical systems was explored first using low dimensional RNNs (Elman, 1990) without gates to predict unsteady boundary-layer development, separation, dynamic stall, and dynamic reattachment back in 1997 (Faller & Schreck, 1997). The utility of RNNs was limited by the finding that during the learning process the gradients may vanish or explode. In turn, the recent success of RNNs is largely attributed to a cell architecture termed Long Short-Term Memory (LSTM). LSTMs employ gates that effectively remember and forget information thus alleviating the problem of vanishing gradients (Hochreiter, 1998). In recent years (Bianchi, Maiorino, Kampffmeyer, Rizzi, & Jenssen, 2017) RNN architectures have been bench-marked for short-term load forecasting of demand and consumption of resources in a supply network, while in Laptev, Yosinski, Li, and Smyl (2017) they are utilized for extreme event detection in low dimensional time series. In Wan and Sapsis (2018) LSTM networks are used as surrogates to model the kinematics of spherical particles in fluid flows. In Vlachas et al. (2018) RNNs with LSTM cells were utilized in conjunction with a mean stochastic model to capture the temporal dependencies and long-term statistics in the reduced order space of a dynamical system and forecast its evolution. The method demonstrated better accuracy and scaling to high-dimensions and longer sequences than Gaussian Processes (GPs). In Wan et al. (2018) the LSTM is deployed to model the residual dynamics in an imperfect Galerkin-based reduced order model derived from the system equations. RC and LSTM networks are applied in the long-term forecasting of partially observable chaotic chimera states in Neofotistos et al. (2019), where instead of a completely model-free approach, ground-truth measurements of currently observed states are helping to improve the long-term forecasting capability. RNNs are practical and efficient data-driven approximators of chaotic dynamical systems, due to their (1) universal approximation ability (Schäfer & Zimmermann, 2006; Siegelmann & Sontag, 1995) and (2) ability to capture temporal dependencies and implicitly identify the required embedding for forecasting.

Despite the rich literature on both methods there are limited comparative studies of the two frameworks. The present work aims to fill this gap by examining these two machine learning algorithms on challenging physical problems. We compare the accuracy, performance, and computational efficiency of the two methods on the full-order and reduced-order modeling of two prototype chaotic dynamical systems. We also examine the modeling capabilities of the two approaches for reproducing correct Lyapunov Exponents and frequency spectra. Moreover, we include in the present work some more recent RNN architectures, like Unitary (Arjovsky, Shah, & Bengio, 2016; Jing et al., 2017) and Gated Recurrent Units (GRUs) (Cho et al., 2014; Chung, Gulcehre, Cho, & Bengio, 2014) that have shown superior performance over LSTMs for a wide variety of language, speech signal and polyphonic music modeling tasks.

We are interested in model-agnostic treatment of chaotic dynamical systems, where the time evolution of the full state or some observable is available, but we do not possess any knowledge about the underlying equations. In the latter case, we examine which method is more suitable for modeling temporal dependencies in the reduced order space (observable) of dynamical systems. Furthermore, we evaluate the efficiency of an ensemble of RNNs in predicting the full state dynamics of a high-dimensional dynamical system in parallel and compare it with that of RC. Finally, we discuss the advantages, implementation aspects (such as RAM requirements and training time) and
eliminate an internal high-dimensional hidden state denoted by $h_{\hat{}}$. Given the current observable $o_t$ that enables the encoding of temporal dependencies on past state. This forecast is a function of the hidden state. As a consequence, the equations that implicitly define the recurrent mapping $f_h^R$ of the LSTM are given by

$$h_{t+1} = f_h^R(o_t, h_t)$$

where $h_{t+1}$ is the hidden-to-hidden mapping and $h_t$ is the hidden-to-output mapping. All recurrent models analyzed in this work share this common architecture. They differ in the realizations of $f_h^R$ and $f_o^R$ and in the way the parameters of weights of these functions are learned from data, i.e., trained, to forecast the dynamics.

2.1. Long short-term memory

In Elman RNNs (Elman, 1990), the vanishing or exploding gradients problem stems from the fact that the gradient is multiplied repeatedly during back-propagation through time (Werbos, 1988) with a recurrent weight matrix. As a consequence, when the spectral radius of the weight matrix is positive (negative), the gradients are prone to explode (shrink). The LSTM (Hochreiter & Schmidhuber, 1997) was introduced in order to alleviate the vanishing gradient problem of Elman RNNs (Hochreiter, 1998) by leveraging gating mechanisms that allow information to be forgotten. The equations that implicitly define the recurrent mapping $f_h^R$ of the LSTM are given by

$$g_t^i = \sigma_i(W_i[h_{t-1}, o_t] + b_i)$$
$$g_t^c = \sigma_c(W_c[h_{t-1}, o_t] + b_c)$$
$$\tilde{c}_t = \tanh(W_c[h_{t-1}, o_t] + b_c)$$
$$c_t = \tilde{c}_t \odot c_{t-1} + g_t^c \odot \tilde{c}_t$$
$$g_t^g = \sigma_g(W_g[h_{t-1}, o_t] + b_g)$$
$$\tilde{h}_t = \tanh(W_h[h_{t-1}, o_t] + b_h)$$
$$h_t = \tilde{h}_t \odot \tanh(c_t),$$

where $g_t^i, g_t^c, g_t^g, \tilde{c}_t, \tilde{h}_t \in \mathbb{R}^d_h$ are the gate vector signals (forget, input and output gates), $o_t \in \mathbb{R}^d_o$ is the observable input at time $t$, $h_t \in \mathbb{R}^d_h$ is the hidden state, $c_t \in \mathbb{R}^d_c$ is the cell state, while $W_i, W_c, W_g, W_h \in \mathbb{R}^{d_h \times (d_h + d_o)}$, are weight matrices and $b_i, b_c, b_g, b_h \in \mathbb{R}^{d_h}$ biases. The symbol $\odot$ denotes the element-wise product. The activation functions $\sigma_i, \sigma_c$ and $\sigma_g$ are sigmoids.

For a more detailed explanation of the LSTM cell architecture and output gates, we refer to Hochreiter and Schmidhuber (1997). The dimension of the hidden state $d_h$ (number of hidden units) controls the capacity of the cell to encode history information. The hidden-to-output functional form $f_o^R$ is given by a linear layer

$$\hat{o}_{t+1} = W_o h_t,$$

while $W_o \in \mathbb{R}^{d_o \times d_h}$. The forget gate bias is initialized to one according to Joëzefowicz, Zaremba, and Sutskever (2015) to accelerate training. An illustration of the information flow in a LSTM cell is given in Fig. 1(c).

2.2. Gated recurrent unit

The Gated Recurrent Unit (GRU) (Cho et al., 2014) was proposed as a variation of LSTM utilizing a similar gating mechanism. Even though GRU lacks an output gate and thus has fewer parameters, it achieves comparable performance with LSTM in polyphonic music and speech signal datasets (Chung et al., 2014). The GRU equations are given by

$$z_t = \sigma_g(W_c[h_{t-1}, o_t] + b_z)$$
$$r_t = \sigma_g(W_c[h_{t-1}, o_t] + b_r)$$
$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, o_t] + b_h)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t,$$

where $o_t \in \mathbb{R}^{d_o}$ is the observable at the input at time $t$, $z_t \in \mathbb{R}^{d_h}$ is the update gate vector, $r_t \in \mathbb{R}^{d_h}$ is the reset gate vector, $h_t \in \mathbb{R}^{d_h}$ is the hidden state, $W_c, W_h \in \mathbb{R}^{d_h \times (d_h + d_o)}$ are weight matrices and $b_z, b_r, b_h \in \mathbb{R}^{d_h}$ biases. The gating activation $\sigma_g$ is a sigmoid. The output $\hat{o}_{t+1}$ is given by the linear layer:

$$\hat{o}_{t+1} = W_o h_t,$$

where $W_o \in \mathbb{R}^{d_o \times d_h}$. An illustration of the information flow in a GRU cell is given in Fig. 1(d).

2.3. Unitary evolution

Unitary RNNs (Arjovsky et al., 2016; Jing et al., 2017), similar to LSTMs and GRUs, aim to alleviate the vanishing gradients problem of plain RNNs. Here, instead of employing sophisticated gating mechanisms, the effort is focused on the identification of a re-parametrization of the recurrent weight matrix, such that its spectral radius is a priori set to one. This is achieved by optimizing the weights on the subspace of complex unitary matrices. The architecture of the Unitary RNN is given by

$$h_t = \text{modReLU}(W_h h_{t-1} + W_o o_t)$$

$$\hat{o}_{t+1} = W_o \Re(h_t),$$

where $W_h \in \mathbb{C}^{d_h \times d_h}$ is the complex unitary recurrent weight matrix, $W_o \in \mathbb{C}^{d_o \times d_h}$ is the complex input weight matrix, $h_t \in \mathbb{C}^{d_h}$ is the complex state vector, $\Re(\cdot)$ denotes the real part of a complex number, $W_o \in \mathbb{C}^{d_h \times d_o}$ is the real output matrix, and the modified ReLU non-linearity modReLU is given by

$$\text{modReLU}(z) = \frac{z}{|z|} \odot \text{ReLU}(|z| + b_i),$$

where $|z|$ is the norm of the complex number $z$. The complex unitary matrix $W_o$ is parametrized as a product of a diagonal matrix and multiple rotational matrices. The reparametrization used in this work is the one proposed in Jing et al. (2017). The
complex input weight matrix $W_o \in \mathbb{C}^{d_o \times d_h}$ is initialized with $W_{rr}^{o} + j W_{im}^{o}$, with real matrices $W_{rr}^{o}, W_{im}^{o} \in \mathbb{R}^{d_o \times d_h}$ whose values are drawn from a random uniform distribution $\mathcal{U}(-0.01, 0.01)$ according to Jing et al. (2017). An illustration of the information flow in a Unitary RNN cell is given in Fig. 1(b).

In the original paper of Jing et al. (2017) the architecture was evaluated on a speech spectrum prediction task, a copying memory task and a pixel permuted MNIST task demonstrating superior performance to LSTM either in terms of final testing accuracy or wall-clock training speed.

2.4. Back-propagation through time

Backpropagation dates back to the works of Dreyfus (1962), Linnainmaa (1976) and Rumelhart, Hinton, and Williams (1986), while its extension to RNNs termed Backpropagation through time (BPTT) was presented in Werbos (1988, 1990). A forward pass of the network is required to compute its output and compare it against the label (or target) from the training data based on an error metric (e.g., mean squared loss). Backpropagation amounts to the computation of the partial derivatives of this loss with respect to the network parameters by iteratively applying the chain rule, transversing backwards the network. These derivatives are computed analytically with automatic differentiation. Based on these partial derivatives the network parameters are updated using a first-order optimization method, e.g., stochastic gradient descent.

The power of BPTT lies in the fact that it can be deployed to learn the partial derivatives of the weights of any network architecture with differentiable activation functions, utilizing state-of-the-art automatic differentiation software, while (as the data are processed in small fragments called batches) it scales to large datasets and networks, and can be accelerated by employing Graphics Processing Units (GPUs). These factors made backpropagation the workhorse of state-of-the-art deep learning methods (Goodfellow et al., 2016).

In our study, we utilize BPTT to train the LSTM (Section 2.1), GRU (Section 2.2) and Unitary (Section 2.3) RNNs. There are three key parameters of this training method that can be tuned. The first hyperparameter $\kappa_1$ is the number of forward-pass time-steps performed to accumulate the error for back-propagation. The second parameter is the number of previous time-steps for the back-propagation of the gradient $\kappa_2$. This is also denoted as truncation length, or sequence length. This parameter has to be large enough to capture the temporal dependencies in the data. However, as $\kappa_2$ becomes larger, training becomes much slower, and may lead to vanishing gradients. In the following, we characterize as stateless, models whose hidden state before $\kappa_2$ is hard-coded to zero, i.e., $h_{\kappa_2} = 0$. Stateless models cannot learn dependencies that expand in a time horizon larger that $\kappa_2$. However, in many practical cases stateless models are widely employed assuming that only short-term temporal dependencies exist. In contrast, stateful models propagate the hidden state $h_{\kappa_2} \neq 0$ between temporally consecutive batches. In our study, we consider only stateful networks.

Training stateful networks is challenging because the hidden state $h_{\kappa_2}$ has to be available from a previous batch and the network has to be trained to learn temporal dependencies that may span many time-steps in the past. In order to avoid overlap between two subsequent data fragments and compute $h_{\kappa_2}$ for the next batch update, the network is teacher-forced for $\kappa_2$ time-steps between two consecutive weight updates. That implies providing ground-truth values at the input and performing forward passing without any back-propagation. This parameter, has an influence on the training speed, as it determines how often the weights are updated. We pick $\kappa_3 = \kappa_2 + \kappa_1 - 1$ as illustrated in Fig. 2, and optimize $\kappa_1$ as a hyperparameter.

The weights of the networks are initialized using the method of Xavier proposed in Glorot and Bengio (2010). We utilize a stochastic optimization method with adaptive learning rate called Adam (Kingma & Ba, 2015) to update the weights and biases. We add Zoneout (Krueger et al., 2017) regularization in the recurrent weights and variational dropout (Gal & Ghahramani, 2016) regularization at the output weights (with the same keep probability) to both GRU and LSTM networks to alleviate over-fitting. Furthermore, following Vlachas et al. (2018) we add Gaussian noise sampled from $\mathcal{N}(0, \kappa_4 \sigma)$ to the training data, where $\sigma$ is the standard deviation of the data. The noise level $\kappa_4$ is tuned. Moreover, we also vary the number of RNN layers by stacking residual layers (He, Zhang, Ren, & Sun, 2016) on top of each other. These deeper architectures may improve forecasting efficiency by learning more informative embedding at the cost of higher computing times.

In order to train the network on a data sequence of $T$ time-steps, we pass the whole dataset in pieces (batches) for many iterations (epochs). An epoch is finished when the network has been trained on the whole dataset once. At the beginning of every epoch we sample uniformly $B = 32$ integers from the set $I = \{1, \ldots, T\}$, and remove them from it. Starting from these indexes we iteratively pass the data through the network till we reach the last (maximum) index in $I$, training it with BPTT. Next, we remove all the intermediate indexes we trained on from $I$. We
repeat this process, until $I = \emptyset$, proclaiming the end of the epoch. The batch-size is thus $B = 32$. We experimented with other batch-sizes $B \in \{8, 16, 64\}$ without significant improvement in performance of the methods used in this work. As an additional over-fitting counter-measure we use validation-based early stopping, where 90% of the data is used for training and the rest 10% for validation. When the validation error stops decreasing for $N_{\text{patience}} = 20$ consecutive epochs, the training round is over. We train the network for $N_{\text{rounds}} = 10$ rounds decreasing the learning rate geometrically by dividing with a factor of ten at each round to avoid tuning the learning rate of the Adam optimizer. When all rounds are finished, we pick the model with the lowest validation error among all epochs and rounds.

Preliminary work on tuning the hyperparameters of the Adam optimization algorithm apart from the learning rate, i.e. $\beta_1$ and $\beta_2$ in the original paper (Kingma & Ba, 2015), did not lead to important differences on the results. For this reason and due to our limited computational budget, we use the default values proposed in the paper (Kingma & Ba, 2015) ($\beta_1 = 0.9$ and $\beta_2 = 0.999$).

2.5. Reservoir computing

Reservoir Computing (RC) aims to alleviate the difficulty in learning the recurrent connections of RNNs and reduce their training time (Lukoševičius, 2012; Lukoševičius & Jaeger, 2009). RC relies on randomly selecting the recurrent weights such that the hidden state captures the history of the evolution of the observable $o$, and train the hidden-to-output weights. The evolution of the hidden state depends on the random initialization of the recurrent matrix and is driven by the input signal. The hidden state is termed reservoir state to denote the fact that it captures temporal features of the observed state history. This technique has been proposed in the context of Echo-State Networks (ESNs) (Jaeger & Haas, 2004) and Liquid State Machines with spiking neurons (LSM) (Maass, Natschläger, & Markram, 2002).

In this work, we consider reservoir computers with $f^h$ given by the functional form

$$h_t = \tanh(W_{oh}o_t + W_{rh}h_{t-1}),$$

where $W_{oh} \in \mathbb{R}^{d_h \times d_o}$ and $W_{rh} \in \mathbb{R}^{d_h \times d_h}$. Other choices of RC architectures are possible, including Antonik, Haelterman, and Massar (2017), Haynes, Soriano, Rosin, Fischer, and Gauthier (2015) and Larger et al. (2017, 2012) Following Jaeger and Haas (2004), the entries of $W_{oh}$ are uniformly sampled from $[-\omega, \omega]$, where $\omega$ is a hyperparameter. The reservoir matrix $W_{rh}$ has to be selected in a way such that the network satisfies the “echo state property”. This property requires all of the conditional Lyapunov exponents of the evolution of $h_t$ conditioned on the input (observations $o_t$) to be negative so that, for large $t$, the reservoir state $h_t$ does not depend on initial conditions. For this purpose, $W_{rh}$ is set to a large low-degree matrix, scaled appropriately to possess a spectral radius (absolute value of the largest eigenvalue) $\rho$ whose value is a hyperparameter adjusted so that the echo state property holds.\(^1\) The effect of the spectral radius on the predictive performance of RC is analyzed in Jiang and Lai (2019). Following Pathak, Hunt et al. (2018) the output coupling $f^o_t$ is set to

$$\hat{o}_{t+1} = W_{oh}h_t,$$

where the augmented hidden state $\hat{h}_t$ is a $d_h$ dimensional vector such that the $i$th component of $\hat{h}_t$ is $\hat{h}^i_t = h^i_t$ for half of the reservoir nodes and $\hat{h}^i_t = (h^i_t)^2$ for the other half, enriching the dynamics with the square of the hidden state in half of the nodes. This was empirically shown to improve forecasting efficiency of RCs in the context of dynamical systems (Pathak, Hunt et al., 2018). The matrix $W_{oh} \in \mathbb{R}^{d_h \times d_o}$ is trained with regularized least-squares regression with Tikhonov regularization to alleviate overfitting (Tikhonov & Arsenin, 1977; Yan & Su, 2009) following the same recipe as in Pathak, Hunt et al. (2018). The Tikhonov regularization $\eta$ is optimized as a hyperparameter. Moreover, we further regularize the training procedure of RC by adding Gaussian noise in the training data. This was shown to be beneficial for both short-term performance and stabilizing the RC in long-term forecasting. For this reason, we add noise sampled from $\mathcal{N}(0, \kappa \sigma)$ to the training data, where $\sigma$ is the standard deviation of the data and the noise level $\kappa$ a tuned hyperparameter.

3. Comparison metrics

The predictive performance of the models depends on the selection of model hyperparameters. For each model we perform an extensive grid search of optimal hyperparameters, reported

\(^1\) Because of the nonlinearity of the tanh function, $\rho < 1$ is not necessarily required for the echo state property to hold true.
in the Appendix. All model evaluations are mapped to a single Nvidia Tesla P100 GPU and are executed on the XC50 compute nodes of the Piz Daint supercomputer at the Swiss national supercomputing center (CSCS). In the following we quantify the prediction accuracy of the methods in terms of the normalized root mean square error, given by

$$\text{NRMSE}(\hat{o}) = \sqrt{\frac{(\hat{o} - o)^2}{\sigma^2}},$$  

where \( \hat{o} \in \mathbb{R}^d \) is the forecast at a single time-step, \( o \in \mathbb{R}^d \) is the target value, and \( \sigma \in \mathbb{R}^d \) is the standard deviation over time of each state component. In Eq. (10), the notation \( (\cdot) \) denotes the state–space average (average of all elements of a vector).

To alleviate the dependency on the initial condition, we report the evolution of the NRMSE over time averaged over 100 initial conditions randomly sampled from the attractor.

The most basic characterization of chaotic motion is through the concept of Lyapunov exponents (Ort, 2002): Considering two infinitesimally close initial conditions \( U(t = 0) \) and \( U(t = 0) + \delta U(t = 0) \), their separation \( |\delta U(t)| \) on average diverges exponentially in time, \( |\delta U(t)/|\delta U(t = 0)| \sim \exp(\Lambda t) \), as \( t \to \infty \). Note that the dimensionality of the vector displacement \( \delta U(t) \) is that of the state–space. In general, the Lyapunov exponent \( \Lambda \) depends on the orientation \( \langle \delta U(t)/|\delta U(t)| \rangle \) of the vector displacement \( \delta U(t) \). In the \( t \to \infty \) limit, the number of possible values of \( \Lambda \) is typically equal to the state–space dimensionality. We denote these values \( \Lambda_1 \geq \Lambda_2 \geq \Lambda_3 \geq \cdots \) and collectively call them the Lyapunov exponent spectrum (LS) of the particular chaotic system. The Lyapunov exponent spectrum will be evaluated in Section 6.

However, we note that a special role is played by \( \Lambda_1 \), the largest Lyapunov exponent. We refer to the largest Lyapunov exponent as the Maximal Lyapunov exponent (MLE). Chaotic motion of a bounded trajectory is defined by the condition \( \Lambda_1 > 0 \). Importantly, if the orientation of \( \delta U(t = 0) \) is chosen randomly, the exponential rate at which the orbits separate is \( \Lambda_1 \) with probability one. This is because in order for any of the other exponents \( \{ \Lambda_2, \Lambda_3, \ldots \} \) to be realized, \( \delta U(t = 0) \) must be chosen to lie on a subspace of lower dimensionality than that of the state–space; i.e., the orientation of \( \delta U(t = 0) \) must be chosen in an absolutely precise way never realized by random choice. Hence, the rate at which typical pairs of nearby orbits separate is \( \Lambda_1 \), and \( T^{\Lambda_1} = \Lambda_1^{-1} \), the “Lyapunov time”, provides a characteristic time scale for judging the quality of predictions based on the observed prediction error growth.

In order to obtain a single metric of the predictive performance of the models we compute the valid prediction time (VPT) in terms of the MLE of the system \( \Lambda_1 \) as

$$\text{VPT} = \frac{1}{\Lambda_1} \arg\max \{ t_f | \text{NRMSE}(o_i) < \epsilon, \forall t \leq t_f \}$$

which is the largest time \( t_f \) the model forecasts the dynamics with a NRMSE error smaller than \( \epsilon \) normalized with respect to \( \Lambda_1 \). In the following, we set \( \epsilon = 0.5 \).

In order to evaluate the efficiency of the methods in capturing the long-term statistics of the dynamical system, we evaluate the mean power spectral density (power spectrum) of the state evolution over all \( i \in \{1, \ldots, d \} \) elements \( o_i \) of the state (since the state \( o \) is a vector). The power spectrum of the evolution of \( o_i \) is given by

$$\text{PSD}(f) = 20 \log_{10} \left( \hat{U}(f) \right) \text{dB},$$

where \( \hat{U}(f) = \text{FFT}(o_i) \) is the complex Fourier spectrum of the state evolution.

### 4. Forecasting reduced order observable dynamics in the Lorenz-96

The accurate long-term forecasting of the state of a deterministic chaotic dynamical system is challenging as even a minor initial error can be propagated exponentially in time due to the system dynamics even if the model predictions are perfect. A characteristic time-scale of this propagation is the Maximal Lyapunov Exponent (MLE) of the system as elaborated in Section 3. In practice, we are often interested in forecasting the evolution of an observable (that we can measure and obtain data from), which does not contain the full state information of the system. The observable dynamics are more irregular and challenging to model and forecast because of the additional loss of information.

Classical approaches to forecast the observable dynamics based on Takens seminal work (Takens, 1981), rely on reconstructing the full dynamics in a high-dimensional phase–space. The state of the phase–space is constructed by stacking delayed versions of the observed state. Assume that the state of the dynamical system is \( x_t \), but we only have access to the less informative observable \( o_t \). The phase–space state, i.e., the embedding state, is given by \( z_t = [o_t, o_{t-d}, \ldots, o_{t-(d-1)d}] \), where the time-lag \( \tau \) and the embedding dimension \( d \) are the embedding parameters. For \( d \) large enough, and in the case of deterministic nonlinear dynamical chaotic systems, there is generally a one-to-one mapping between a point in the phase–space and the full state of the system and vice versa. This implies that the dynamics of the system are deterministically reconstructed in the phase–space (Kantz & Schreiber, 1997) and that there exists a phase–space forecasting rule \( z_{t+1} = F^\tau(z_t) \), and thus an observable forecasting rule \( \hat{o}_{t+1} = F^\tau(o_t, o_{t-1}, \ldots, o_{t-(d-1)d}) \).

The recurrent architectures presented in Section 2 fit to this framework, as the embedding state information can be captured in the high-dimensional hidden state \( h_t \) of the networks by processing the observable time series \( o_t \), without having to tune the embedding parameters \( \tau \) and \( d \).

In the following, we introduce a high-dimensional dynamical system, the Lorenz-96 model and evaluate the efficiency of the methods to forecast the evolution of a reduced order observable of the state of this system. Here the observable is not the full state of the system, and the networks need to capture temporal dependencies to efficiently forecast the dynamics.

#### 4.1. Lorenz-96 model

The Lorenz-96 model was introduced by Lorenz (1995) to model the large-scale behavior of the mid-latitude atmosphere. The model describes the time evolution of an atmospheric variable that is discretized spatially over a single latitude circle modeled in the high-dimensional state \( x = [x_0, \ldots, x_{J-1}] \), and is defined by the equations

$$\frac{dx_j}{dt} = (x_{j+1} - x_{j-2})x_{j-1} - x_j + F,$$

for \( j \in \{0, 1, \ldots, J-1\} \), where we assume periodic boundary conditions \( x_{J-1} = x_0 \), \( x_{J-2} = x_{J-1} \). In the following we consider a grid-size \( J = 40 \) and two different forcing regimes, \( F = 8 \) and \( F = 10 \).

We solve Eq. (12) starting from a random initial condition with a Fourth Order Runge–Kutta scheme and a time-step of \( \delta t = 0.01 \). We run the solver up to \( T = 2000 \) after ensuring that transient effects are discarded (\( T_{trans} = 1000 \)). The first half \( 10^3 \) samples are used for training and the rest for testing. For the forecasting test in the reduced order space, we construct observables of dimension \( d_o \in \{35, 40\} \) by performing Singular Value Decomposition (SVD) and keeping the most energetic \( d_o \) components. The complete procedure is described in the Appendix. The 35 most energetic modes taken into account in the reduced order observable, explain approximately 98% of the total energy of the system in both \( F \in \{8, 10\} \).

As a reference timescale that characterizes the chaoticity of the system we use the Lyapunov time, which is the inverse of
The evolution of the NRMSE of the model with the largest VPT of each architecture for $F \in \{8, 10\}$ is plotted in Fig. 3 for two values of the dimension of the observable $d_o \in \{35, 40\}$, where $d_o = 40$ corresponds to full state information. Note that the observable is given by first transforming the state to its SVD modes and then keeping the $d_o$ most energetic ones. As indicated by the slopes of the curves, models predicting the observable containing full state information ($d_o = 40$) exhibit a slightly slower NRMSE increase compared to models predicting in the reduced order state, as expected.

When the full state of the system is observed, the predictive performance of RC is superior to that of all other models. Unitary networks diverge from the attractor in both reduced order and full space in both forcing regimes $F \in \{8, 10\}$. This divergence (inability to reproduce the long-term climate of the dynamics) stems from the iterative propagation of the forecasting error. The issue has been also demonstrated in previous studies in both RC (Lu et al., 2018; Pathak, Wikner et al., 2018) and RNNs (Vlachas et al., 2018). This is because the accuracy of the network for long-term climate modeling, depends not only on how well it approximates the dynamics on the attractor locally, but also on how it behaves near the attractor, where we do not have data. As noted in Lu et al. (2018), assuming that the network has a full Lyapunov spectrum near the attractor, if any of the Lyapunov exponents that correspond to infinitesimal perturbations transverse to the attractor phase-space is positive, then the predictions of the network will eventually diverge from the attractor. Empirically, the divergence effect can also be attributed to insufficient network size (model expressiveness) and training, or attractor regions in the state–space that are underrepresented in the training data (poor sampling). Even with a densely sampled attractor, during iterative forecasting in the test data, the model is propagating its own predictions, which might lead to a region near (but not on) the attractor where any positive Lyapunov exponent corresponding to infinitesimal perturbations transverse to the attractor will cause divergence.

In this work, we use $10^5$ samples to densely capture the attractor. Still, RC suffers from the iterative propagation of errors leading to divergence especially in the reduced order forecasting scenario. In order to alleviate the problem, a parallel scheme for RC is proposed in Pathak, Wikner et al. (2018) that enables training of many reservoirs locally forecasting the state. However, this method is limited to systems with local interactions in their state–space. In the case we discuss here the observable obtained by singular value decomposition does not fulfill this assumption. In many systems the assumption of local interaction may not hold. GRU and LSTM show superior forecasting performance in the reduced order scenario setting in Lorenz-96 as depicted in Figs. 3(a)–3(c). Especially in the case of $F = 10$, the LSTM and GRU models are able to predict up to 2 Lyapunov times ahead before reaching an NRMSE of $\epsilon = 1$, compared to RC and Unitary RNNs that reach this error threshold in 1 Lyapunov time. However, it should be noted that the predictive utility of all models (considering an error threshold of $\epsilon = 0.5$) is limited to one Lyapunov time when applied to reduced order data and up to two Lyapunov times in the full state.

In order to analyze the sensitivity of the VPT to the hyper-parameter selection, we present violin plots in Fig. 4, showing a
Fig. 4. Violin plot showing the probability density of the VPT of all hyperparameter sets for each model for observable dimension $d_o = 35$ and $d_o = 40$ and forcing regimes (a) $F = 8$ and (b) $F = 10$ in the Lorenz-96. RC LSTM; GRU; RC Unitary networks, but rather to the difficulty on identifying the right hyperparameters (Griff, Srivastava, Couti, Steunebrink, & Schmidhuber, 2016). In Fig. 4 we observe that the violin plots in the reduced order state are much thinner at the top compared to the ones in the full state. This implies that the identification of hyperparameter sets that achieve a high VPT in the reduced order space is more challenging. This emphasizes that forecasting on the reduced order state is a more difficult task compared to the full state scenario.

In the following, we evaluate the ability of the trained networks to forecast the long-term statistics of the dynamical system. In almost all scenarios and all cases considered in this work, forecasts of Unitary RNN networks fail to remain close to the attractor and diverge. For this reason, we omit the results on these networks.

We quantify the long-term behavior in terms of the power spectrum of the predicted dynamics and its difference with the ground-truth spectrum. In Fig. 5, we plot the power spectrum of the predicted dynamics from the model (hyperparameter set) with the lowest power spectrum error for each architecture for $d_o \in \{35, 40\}$ and $F \in \{8, 10\}$ against the ground-truth spectrum computed from the testing data (dashed black line). In the full state scenario in both forcing regimes (Figs. 5(b), 5(d)), all models match the true statistics in the test dataset, as the predicted power spectra match the ground-truth. These results imply that RC is a powerful predictive tool in the full order state scenario, as RC models both capture the long-term statistics and have the highest VPT among all other models analyzed in this work. However, in the case of a reduced order observable, the RC cannot match the statistics. In contrast, GRU and LSTM networks achieve superior forecasting performance while matching the long-term statistics, even at this challenging setting of a chaotic system with reduced order information.

An important aspect of machine learning models is their scalability to high-dimensional systems and their requirements in terms of training time and memory utilization. Large memory requirements and/or high training times might hinder the application of the models in challenging scenarios, like high-performance applications in climate forecasting (Kurth et al., 2018). In Figs. 6(a) and 6(d), we present a Pareto front of the VPT with respect to the CPU RAM memory utilized to train the models with the highest VPT for each architecture and reduced order the testing data (dashed black line). 

In the full state scenario (Fig. 6(b) and 6(d)), we observe that RC models both capture the long-term statistics and have the highest VPT among all other models analyzed in this work. However, in the case of a reduced order observable, the RC cannot match the statistics. In contrast, GRU and LSTM networks achieve superior forecasting performance while matching the long-term statistics, even at this challenging setting of a chaotic system with reduced order information.

Table 1 Maximum and average Valid Prediction Time (VPT) over all hyperparameter sets averaged over 100 initial conditions sampled from the testing data for each model.

<table>
<thead>
<tr>
<th>Model</th>
<th>Scenario</th>
<th>$F = 8$</th>
<th>$F = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$d_o = 35$</td>
<td>$d_o = 40$</td>
<td>$d_o = 35$</td>
</tr>
<tr>
<td>Unit</td>
<td>MAX</td>
<td>AVG</td>
<td>MAX</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.74</td>
<td>0.37</td>
<td>0.79</td>
</tr>
<tr>
<td>GRU</td>
<td>0.55</td>
<td>0.32</td>
<td>2.31</td>
</tr>
<tr>
<td>RC</td>
<td>0.44</td>
<td>0.34</td>
<td>0.58</td>
</tr>
</tbody>
</table>

In the full state scenario ($d_o = 40$) and forcing regime $F = 8$, RC shows a remarkable performance with a maximum VPT $\approx 2.31$, while GRU exhibits a max VPT of $\approx 1.34$. The LSTM has a max VPT of $\approx 0.97$, while Unitary RNNs show the lowest forecasting ability with a max VPT of $\approx 0.58$. From the violin plots in Fig. 4 we notice that densities are wider at the lower part, corresponding to many models (hyperparameter sets) having much lower VPT than the maximum, emphasizing the importance of tuning the hyperparameters. Similar results are obtained for the forcing regime $F = 10$. One noticeable difference is that the LSTM exhibits a max VPT of $\approx 1.73$ which is higher than that of GRU which is $\approx 1.59$. Still, the VPT of RC in the full state scenario is $\approx 2.35$ which is the highest among all models.

In contrast, in the case of $d_o = 35$ where the models are forecasting on the reduced order space in the forcing regime $F = 8$, GRU is superior to all other models with a maximum VPT $\approx 0.98$ compared to LSTM showing a max VPT $\approx 0.74$. LSTM shows inferior performance to GRU which we speculate may be due to insufficient hyperparameter optimization. Observing the results on $F = 10$ justifies our claim, as indeed both the GRU and the LSTM show the highest VPT values of $\approx 1.22$ and $\approx 1.17$ respectively. In both scenarios $F = 8$ and $F = 10$, when forecasting the reduced order space $d_o = 35$, RC shows inferior performance compared to both GRU and LSTM networks with max VPT $\approx 0.55$ for $F = 8$ and $\approx 0.60$ for $F = 10$. Last but not least, we observe that Unitary RNNs show the lowest forecasting ability among all models. This may not be attributed to the expressiveness of Unitary networks, but rather to the difficulty on identifying the right hyperparameters (Griff, Srivastava, Couti, Steunebrink, & Schmidhuber, 2016). In Fig. 4 we observe that the violin plots in the reduced order state are much thinner at the top compared to the ones in the full state. This implies that the identification of hyperparameter sets that achieve a high VPT in the reduced order space is more challenging. This emphasizes that forecasting on the reduced order state is a more difficult task compared to the full state scenario.
Fig. 5. Predicted power spectrum of the RC, GRU, and LSTM networks with the lowest spectrum error forecasting the dynamics of an observable consisting of the SVD modes of the Lorenz-96 system with forcing $F \in \{8, 10\}$. The observable consists of the $d_o = 35$ most energetic modes or full state information $d_o = 40$. (a) Reduced order observable at forcing $F = 8$. (b) Full state observable at forcing $F = 8$. (c) Reduced order observable at forcing $F = 10$. (d) Full state observable at forcing $F = 10$. RC - GRU - LSTM - Groundtruth. (For interpretation of the reference to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 6. Forecasting results on the dynamics of an observable consisting of the SVD modes of the Lorenz-96 system with forcing $F = 8$ and state dimension 40. The observable consists of the $d_o \in \{35, 40\}$ most energetic modes. (a), (d) Valid prediction time (VPT) plotted w.r.t. the required RAM memory for dimension $d_o \in \{35, 40\}$. (b), (e) VPT plotted w.r.t. total training time for dimension $d_o \in \{35, 40\}$. (c), (f) VPT measured from 100 initial conditions sampled from the test data plotted against the VPT from 100 initial conditions sampled from the training data for each model for $d_o \in \{35, 40\}$. In the reduced order space ($d_o = 35$), RCs tend to overfit easier compared to GRUs/LSTMs that utilize validation-based early stopping. In the full order space ($d_o = 40$), RCs demonstrate excellent generalization ability and high forecasting accuracy. RC - GRU - LSTM - Unit - Ideal. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Due to the fact that the RNN models are learning the recurrent connections, they are able to reach higher VPT when forecasting in the reduced order space without the need for large models. In contrast, in RC the maximum reservoir size (imposed by computer memory limitations) may not be sufficient to capture the dynamics of high-dimensional systems with reduced order information and non-local interactions. We argue that this is the reason why the RC models do not reach the performance of GRUs/LSTMs that utilize validation-based early stopping. In the full order space ($d_o = 40$), RCs demonstrate excellent generalization ability and high forecasting accuracy.

At the same time, letting memory limitations aside, training of RC models requires the solution of a linear system of equations $HW_{out}^T = Y$, with $H \in \mathbb{R}^{d_h \times d_h}$, $W_{out} \in \mathbb{R}^{d_h \times d_h}$, and $Y \in \mathbb{R}^{d_h \times d_o}$ (see Appendix A). The Moore–Penrose method of solving this system, scales cubically with the reservoir size as it requires the inversion of a matrix with dimensions $d_h \times d_h$. We also tried an approximate iterative method termed LSQR based on diagonalization, without any significant influence on the training time. In contrast, the training time of an RNN is very difficult to estimate a priori, as convergence of the training method depends on initialization and various other hyperparameters and are not necessarily dependent on the size. That is why we observe a greater variation of the training time of RNN models. Similar results are obtained for $F = 10$, the interested reader is referred to the Appendix.

In the following, we evaluate to which extend the trained models overfit to the training data. For this reason, we measure the VPT in the training dataset and plot it against the VPT in the test dataset for every model we trained. This plot provides insight on the generalization error of the models. The results are shown in Figs. 6(c), and 6(f) for $d_o = 35$ and $d_o = 40$. Ideally a model architecture that guards effectively against overfitting, exhibits a low generalization error, and should be represented by a point in this plot that is close to the identity line (zero generalization error). As the expressive power of a model increases, the model may fit better to the training data, but bigger models are
more prone to memorizing the training dataset and overfitting (high generalization error). Such models would be represented by points on the right side of the plot. In the reduced order scenario, GRU and LSTM models lie closer to the identity line than RC models, exhibiting lower generalization errors. This is due to the validation-based early stopping routine utilized in the RNNs that guards effectively against overfitting.

We may alleviate the overfitting in RC by tuning the Tikhonov regularization parameter ($\eta$). However, this requires to rerun the training for every other combination of hyperparameters. For the four tested values $\eta \in \{10^{-2}, 10^{-4}, 10^{-5}, 10^{-6}\}$ of the Tikhonov regularization parameter the RC models tend to exhibit higher generalization error compared to the RNN trained with BPTT. We also tested more values $\eta \in \{10^{-4}, 10^{-2}, 10^{-3}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$, while keeping fixed the other hyperparameters, without any observable differences in the results.

However, in the full-order scenario, the RC models achieve superior forecasting accuracy and generalization ability as clearly depicted in Fig. 6(f). Especially the additional regularization of the training procedure introduced by adding Gaussian noise in the data was decisive to achieve this result.

An example of an iterative forecast in the test dataset, is illustrated in Fig. 7 for $F = 8$ and $d_o \in \{35, 40\}$.

5. Parallel forecasting leveraging local interactions

In spatially extended dynamical systems the state–space (e.g., vorticity, velocity field, etc.) is high-dimensional (or even infinite dimensional), since an adequately fine grid is needed to resolve the relevant spatio-temporal scales of the dynamics. Even though RC and RNNs can be utilized for modeling and forecasting of these systems in the short-term, the RC and RNN methods described in Section 2 do not scale efficiently with the input dimension, i.e., as the dimensionality of the observable $\mathbf{o} \in \mathbb{R}^{d_o}$ increases. Two limiting factors are the required time and RAM memory to train the model. As $d_o$ increases, the size $d_n$ of the reservoir network required to predict the system using only a single reservoir rises. This implies higher training times and more computational resources (RAM memory), which render the problem intractable for large values of $d_o$. The same applies for RNNs. More limiting factors arise by taking the process of identification of optimal model hyperparameters into account, since loading, storing and processing a very large number of large models can be computationally infeasible. However, these scaling problems for large systems can be alleviated in case the system is characterized by local state interactions or translationally invariant dynamics. In the first case, as shown in Fig. 8 the modeling and forecasting task can be parallelized by employing multiple individually trained networks forecasting locally in parallel exploiting the local interactions, while, if translation invariance also applies, the individual parallel networks can be identical and training of only one will be sufficient. This parallelization concept is utilized in RC in Pathak, Hunt et al. (2018). The idea dates back to local interactions, the evolution of each element is affected by its spatially neighboring grid points. The elements $\mathbf{o}_i$ are split into $N_g$ groups, each of which consisting of $G$ spatially neighboring elements such that $d_o = G N_g$. The parallel model employs $N_g$ RNNs, each of which is utilized to predict a spatially local region of the system observable indicated by the $G$ group elements $\mathbf{o}_i$. Each of the $N_g$ RNNs receives $G$ inputs $\mathbf{o}_i$ from the elements $i$ it forecasts in addition to $I$ inputs from neighboring elements on the left and on the right, where $I$ is the interaction length. An example with $G = 2$ and $I = 1$ is illustrated in Fig. 8.

During the training process, the networks can be trained independently. However, for long-term forecasting, a communication protocol has to be utilized as each network requires the predictions of neighboring networks to infer. In the case of a homogeneous system, where the dynamics are translation invariant, the training process can be drastically reduced by utilizing one single RNN and training it on data from all groups. The weights of this RNN are then copied to all other members of the network. In the following we assume that we have no knowledge of the underlying data generating mechanism and its properties, so we assume the data is not homogeneous.

The elements of the parallel architecture are trained independently, while the MPI (Dalćin, Paz, Kler, & Cosimo, 2011; Dalćin, Paz, Storti, & D’Elía, 2008; Walker & Dongarra, 1996) communication protocol is utilized to communicate the elements of the interaction for long-term forecasting.

5.2. Results on the Lorenz-96

In this section, we employ the parallel architecture to forecast the state dynamics of the Lorenz-96 system explained in Section 4.1 with a state dimension of $d_o = 40$. Note that in contrast to Section 4.2, we do not construct an observable and then forecast the reduced order dynamics. Instead, we leverage the local interactions in the state–space and employ an ensemble of networks forecasting the local dynamics.

Instead of a single RNN model forecasting the $d_o = 40$ dimensional global state (composed of the values of the state in the 40 grid nodes), we consider $N_g = 20$ separate RNN models, each forecasting the evolution of a $G = 2$ dimensional local state (composed of the values of the state in 2 neighboring grid nodes). In order to forecast the evolution of the local state, we take into account its interaction with $I = 4$ adjacent grid nodes on its left and on its right. The group size of the parallel models is thus $G = 2$, while the interaction length is $I = 4$. As a consequence, each model receives at its input a $2I + G = 10$ dimensional state and forecasts the evolution of a local state composed from 2 grid nodes. The size of the hidden state in RC is $d_h \in \{1000, 3000, 6000, 12000\}$. Smaller networks of size $d_h \in \{100, 250, 500\}$ are selected for GRU and LSTM. The rest of the hyperparameters are given in the Appendix. Results for Unitary networks are omitted, as the identification of hyperparameters leading to stable iterative forecasting was computationally heavy and all trained models led to unstable systems that diverged after a few iterations.

In Fig. 9(a), we plot the VPT time of the RC and the BPTT networks. We find that RNN trained by BPTT achieve comparable predictions with RC, albeit using much smaller number hidden nodes (between 100 and 500 for BPTT vs 6000 to 12000 for RC). We remark that RC with 3000 and 6000 nodes have slightly lower VPT than GRU and LSTM but require significantly lower training times as shown in Fig. 9(c). At the same time, using 12000 nodes for RC implies high RAM requirements, more than 3 GB per rank, as depicted in Fig. 9(b).

As elaborated in Section 4.2 and depicted in Fig. 3(a), the VPT reached by large nonparallelized models that are forecasting
Fig. 7. Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz-96 system with $F = 8$ in the testing dataset with GRU, LSTM, RC and a Unitary network along with the true (target) evolution and the associated NRSE contours for the reduced order observable (a) $d_o = 35$ and the full state (b) $d_o = 40$. The evolution of the component average NRSE (NMRSE) is plotted to facilitate comparison. Unitary networks suffer from propagation of forecasting error and eventually their forecasts diverge from the attractor. Forecasts in the case of an observable dimension $d_o = 40$ diverge slower as the dynamics are deterministic. In contrast, forecasting the observable with $d_o = 35$ is challenging due to both (1) sensitivity to initial condition and (2) incomplete state information that requires the capturing of temporal dependencies. In the full-state setting, RC models achieve superior forecasting accuracy compared to all other models. In the challenging reduced order scenario, LSTM and GRU networks demonstrate a stable behavior in iterative prediction and reproduce the long-term statistics of the attractor (attractor climate). In contrast, in the reduced order scenario iterative predictions of RC diverge frequently from the attractor (refer to the Appendix). GRU ; LSTM ; RC-6000 ; RC-9000 ; Unit . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the 40 SVD modes of the system is approximately 1.4. We also verified that the nonparallelized models of Section 4.1 when forecasting the 40 dimensional state containing local interactions instead of the 40 modes of SVD, reach the same predictive performance. Consequently, as expected the VPT remains the same independently of whether we are forecasting the state or the SVD
In order to spatially discretize Eq. (13) we select a grid size $\Delta x$ with $D = L / \Delta x + 1$ the number of nodes. Further, we denote with $u_i = u(i \Delta x)$ the value of $u$ at node $i \in \{0, \ldots, D - 1\}$. In the following, we select $\nu = 1$, $L = 200$, $\Delta t = 0.25$ and a grid of $d_o = 512$ nodes. We discretize Eq. (13) and solve it using the fourth-order method for stiff PDEs introduced in Kassam and Trefethen (2005) up to $T = 6 \cdot 10^4$. This corresponds to $24 \cdot 10^4$ samples. The first $4 \cdot 10^4$ samples are truncated to avoid initial transients. The remaining data are divided to a training and a testing dataset of $10^5$ samples each. The observable is considered to be the $d_o = 512$ dimensional state. The Lyapunov time $T_{\Delta I}$ of the system (see Section 3) is utilized as a reference timescale. We approximate it with the method of Pathak (Pathak, Hunt et al., 2018) for $L = 200$ and it is found to be $T_{\Delta I} \approx 0.094$.

5.4. Results on the Kuramoto–Sivashinsky equation

In this section, we present the results of the parallel models in the Kuramoto–Sivashinsky equation. The full system state is used as an observable, i.e., $d_o = 512$. The group-size of the parallel models is set to $G = 8$, while the interaction length is $I = 8$. The total number of groups is $N_g = 64$. Each member forecasts the evolution of 8 state components, receiving at the input 24 components in total. The size of the reservoir in RC is $d_R = \{500, 1000, 3000\}$. For GRU and LSTM networks we vary $d_R = \{100, 250, 500\}$. The rest of the hyperparameters are given in the Appendix. Results on Unitary networks are omitted, as the configurations tried in this work led to unstable models diverging after a few time–steps in the iterative forecasting procedure.

The results are summed up in the bar-plots in Fig. 12. In Fig. 12(a), we plot the VPT time of the models. LSTM models reach VPTs of $\approx 4$, while GRU show an inferior predictive performance with VPTs of $\approx 3.5$. An RC with $d_R = 500$ reaches a VPT of $\approx 3.2$, and an RC with 1000 nodes reaches the VPT of LSTM models with a VPT of $\approx 3.9$. Increasing the reservoir capacity of the RC to $d_R = 3000$ leads to a model exhibiting a VPT of $\approx 4.8$. In this case, the large RC model shows slightly superior performance to GRU/LSTM. The low performance of GRU models can be attributed to the fact that in the parallel setting the probability that any RNN may converge to bad local minima rises, with a detrimental effect on the total predictive performance of the parallel ensemble. In case of spatially translational invariant systems, we could alleviate this problem by using one single network. Still, training the single network to data from all spatial locations would be expensive.

As depicted in Fig. 12, the reservoir size of 3000 is enough for RC to reach and surpass the predictive performance of RNNs utilizing a similar amount of RAM memory and a much lower amount of training time as illustrated in Fig. 12(b).

![Fig. 8. Illustration of the parallel architecture for a group size of $G = 2$ and an interaction length of $I = 1$. The network consists of multiple RNNs with different parameters. Each RNN is trained to forecast the evolution of $G$ elements of the observable.](image-url)

![Fig. 9. (a) Valid prediction time (VPT). (b) CPU memory utilization and (c) total training time of RNN parallel architectures with group size $G = 2$ and an interaction length $I = 4$ forecasting the dynamics of Lorenz-96 with state dimension $d_o = 40$ (full state). GRU and LSTM results do not depend significantly on network size. RC with 3000 or 6000 nodes have slightly lower VPT, but require much less training time. Increasing RC size more than 12000 nodes was not feasible due to memory requirements.](image-url)
The evolution of the NRMSE is given in Fig. 13(a). The predictive performance of a small LSTM network with 80 hidden units, matches that of a large RC with 1000 hidden units. In Fig. 13(b), the power spectrum of the predicted state dynamics of each model is plotted along with the true spectrum of the equations. The three models captured successfully the statistics of the system, as we observe a very good match. An example of an iterative prediction with LSTM, GRU and RC models starting from an initial condition in the test dataset is provided in Fig. 14.

6. Calculation of Lyapunov exponents in the Kuramoto–Sivashinsky equation

The recurrent models utilized in this study can be used as surrogate models to calculate the Lyapunov exponents (LEs) of a dynamical system relying only on experimental time series data. The LEs characterize the rate of separation if positive (or convergence if negative) of trajectories that are initialized infinitesimally close in the phase–space. They can provide an estimate of the attractor dimension according to the Kaplan–Yorke
Fig. 12. (a) Valid prediction time (VPT), (b) total training time, and (c) CPU memory utilization of parallel RNN architectures with group size \( G = 8 \) and an interaction length \( I = 8 \) forecasting the dynamics of Kuramoto–Sivashinsky equation with state dimension \( d_o = 512 \).

Fig. 13. (a) The evolution of the NRMSE error (averaged over 100 initial conditions) of different parallel models in the Kuramoto–Sivashinsky equation with state dimension \( d_o = 512 \). (b) The power spectrum. All models capture the statistics of the system. RC-500 \( \cdots \) GRU-80 \( \cdots \); LSTM-80 \( \cdots \); Groundtruth \( \cdots \). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

formulas (Kaplan & Yorke, 1979). Early efforts to solve the challenging problem of data-driven LE identification led to local approaches (Sano & Sawada, 1985; Wolf, Swift, Swinney, & Vastano, 1985) that are computationally inexpensive at the cost of requiring a large amount of data. Other approaches fit a global model to the data (Maus & Sprott, 2013) and calculate the LE spectrum using the Jacobian algorithm. These approaches were applied to low-order systems.

A recent machine learning approach utilizes deep convolutional neural networks for LE and chaos identification, without estimation of the dynamics (Makarenko, 2018). An RC-RNN approach capable of uncovering the whole LE spectrum in high-dimensional dynamical systems is proposed in Pathak, Hunt et al. (2018). The method is based on the training of a surrogate RC model to forecast the evolution of the state dynamics, and the calculation of the Lyapunov spectrum of the hidden state of this surrogate model. The RC method demonstrates excellent agreement for all positive Lyapunov exponents and many of the negative exponents for the KS equation with \( L = 60 \) (Pathak, Hunt et al., 2018), alleviating the problem of spurious Lyapunov exponents of delay coordinate embeddings (Dechert & Gençay, 1996). We build on top of this work and demonstrate that a GRU trained with BPTT can reconstruct the Lyapunov spectrum accurately with lower error for all positive Lyapunov exponents at the cost of higher training times.

The Lyapunov spectrum of the KS equation is computed by solving the KS equations in the Fourier space with a fourth order time–stepping method called ETDRK4 (Kassam & Trefethen, 2005) and utilizing a QR decomposition approach as in Pathak, Hunt et al. (2018). The Lyapunov spectrum of the RNN and RC surrogate models is computed based on the Jacobian of the hidden state dynamics along a reference trajectory, while Gram–Schmidt orthonormalization is utilized to alleviate numerical divergence. We employ a GRU-RNN over LSTM-RNN, due to the fact that the latter has two coupled hidden states, rendering the computation of the Lyapunov spectrum mathematically more involved and computationally more expensive. The interested reader can refer to the Appendix for the details of the method. The identified maximum LE is \( \Lambda_1 \approx 0.08844 \). In this work, a large RC with \( d_h = 9000 \) nodes is employed for LS calculation in the Kuramoto–Sivashinsky equation with parameter \( L = 60 \) and \( D = 128 \) grid points as in Pathak, Hunt et al. (2018). The largest LE identified in this case is \( \Lambda_1 \approx 0.08378 \) leading to a relative error of 5.3%. In order to evaluate the efficiency of RNNs, we utilize a large GRU with \( d_h = 2000 \) hidden units. An iterative RNN roll-out of \( N = 10^4 \) total time–steps was needed to achieve convergence of the spectrum. The largest Lyapunov exponent identified by the GRU is \( \Lambda_1 \approx 0.0849 \) reducing the error to \( \approx 4\% \). Both surrogate models identify the correct Kaplan–Yorke dimension \( KY \approx 15 \), which is the largest LE such that \( \sum \Lambda_i > 0 \).

The first 26 Lyapunov exponents computed the GRU, RC as well as using the true equations of the Kuramoto–Sivashinsky are plotted in Fig. 15. We observe a good match between the positive Lyapunov exponents by both GRU and RC surrogates. The positive Lyapunov exponents are characteristic of chaotic behavior. However, the zero Lyapunov exponents \( \Lambda_1 \) and \( \Lambda_8 \) cannot be captured either with RC or with RNN surrogates. This is also observed in RC in Pathak, Hunt et al. (2018), and apparently the GRU surrogate employed in this work does not alleviate the problem. In Fig. 15(b), we augment the RC and the GRU spectrum with these two additional exponents to illustrate that there is an excellent agreement between the true LE and the augmented LS identified by the surrogate models. The relative and absolute errors in the spectrum calculation are illustrated in Fig. 16.
Fig. 14. Contour plots of a spatio-temporal forecast starting from an initial condition in the testing dataset with parallel GRU, LSTM, and RC networks along with the true (target) evolution and the associated NRSE contours in the Kuramoto–Sivashinsky equation with the full state as an observable $d = 512$. The evolution of the component average NRSE (NMRSE) is plotted to facilitate comparison. RC-1000 ; RC-3000 ; GRU-80 ; LSTM-80 ; Groundtruth . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 15. (a) Estimated Lyapunov exponents $\Lambda_{k}$ of the KS equation with $L = 60$. The true Lyapunov exponents are illustrated with green crosses, red circles are calculated with the RC surrogate, while the blue rectangles with GRU. In (b) we augment the computed spectrum with the two zero Lyapunov exponents $\Lambda_{7}, \Lambda_{8}$. Inset plots zoom in the zero crossing regions. True ; RC ; GRU . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

augmenting with the two zero LE, we get a mean absolute error of 0.012 for RC and 0.008 for GRU. The mean relative error is 0.23 for RC, and 0.22 for GRU. As a conclusion, GRU in par with RC networks can be used to replicate the chaotic behavior of a reference system and calculate the Lyapunov spectrum accurately.

7. Conclusions

In this work, we employed several variants of recurrent neural networks and reservoir computing to forecast the dynamics of chaotic systems. We present a comparative study based on their efficiency in capturing temporal dependencies, evaluate how they
dimensional systems, as the required hidden state size scales to systems with high-dimensional state–space, and how to guard against overfitting. We highlight the advantages and limitations of these methods and elucidate their applicability to forecasting spatiotemporal dynamics.

We considered three different types of RNN cells that alleviate the well-known vanishing and exploding gradient problem in Back-propagation through time training (BPTT), namely LSTM, GRU and Unitary cells. We benchmarked these networks against reservoir computers with random hidden to hidden connection weights, whose training procedure amounts to least square regression on the output weights. The efficiency of the models in capturing temporal dependencies in the reduced order state–space is evaluated on the Lorenz-96 system in two different forcing regimes $F = [8, 10]$, by constructing a reduced order observable using Singular Value Decomposition (SVD) and keeping the most energetic modes. Even though this forecasting task is challenging due to (1) chaotic dynamics and (2) reduced order information, LSTM and GRU show superior forecasting ability to RC utilizing similar amounts of memory at the cost of higher training times. GRU and LSTM models demonstrate stable behavior in the iterative forecasting procedure in the sense that the forecasting error usually does not diverge, in stark contrast to RC and Unitary forecasts. Large RC models tend to overfit easier than LSTM/GRU models, as the latter are utilizing validation-based early stopping and regularization techniques (e.g., Zoneout, Dropout) that guard against overfitting which are not directly applicable to RC. Validation in RC amounts to tuning an additional hyperparameter, the Tikhonov regularization. However, RC shows excellent forecasting efficiency when the full state of the system is observed, outperforming all other models by a wide margin, while also reproducing the frequency spectrum of the underlying dynamics.

RNNs and RC both suffer from scalability problems in high-dimensional systems, as the required hidden state size $d_H$ to capture the high-dimensional dynamics can become prohibitively large especially with respect to the computational expense of training. In order to scale the models to high-dimensional systems we employ a parallelization scheme that exploits the local interactions in the state of a dynamical system. As a reference, we consider the Lorenz-96 system and the Kuramoto–Sivashinsky equation, and we train parallel RC, GRU, and LSTM models of various sizes. Iterative forecasting with parallel Unitary models diverged after a few time–steps in both systems. Parallel GRU, LSTM and RC networks reproduced the long-term attractor climate, as well as the power spectrum of the state of the Lorenz-96 and the Kuramoto–Sivashinsky equation matched with the predicted ones.

In the Lorenz-96 and the Kuramoto–Sivashinsky equation, the parallel LSTM and GRU models exhibited similar predictive performance compared to the parallel RC. The memory requirements of the models are comparable. RC networks require large reservoirs with 1000–6000 nodes per member to reach the predictive performance of parallel GRU/LSTM with a few hundred nodes, but their training time is significantly lower.

Further investigation on the underlying reasons why the RNNs and RC cannot capture the zero Lyapunov exponents is a matter of ongoing work. Another interesting direction could include studying the memory capacity of the networks. This could offer more insight into which architecture and training method is appropriate for tasks with long-term dependencies. Moreover, we plan to investigate a coupling of the two training approaches to further improve their predictive performance, for example a network can utilize both RC and LSTM computers to identify the input to output mapping. While the weights of the RC are initialized randomly to satisfy the echo state property, the output weights alongside with the LSTM weights can be optimized by back-propagation. This approach, although more costly, might achieve higher efficiency, as the LSTM is used as a residual model correcting the error that a plain RC would have.

Although we considered a batched version of RC training to reduce the memory requirements, further research is needed to alleviate the memory burden associated with the matrix inversion (see Appendix A, (15)) and the numerical problems associated with the eigenvalue decomposition of the sparse weight matrix.

Further directions could be the initialization of RNN weights with RC based heuristics based on the echo state property and fine-tuning with BPTT. This is possible for the plain cell RNN, where the heuristics are directly applicable. However, in more complex architectures like the LSTM or the GRU, more sophisticated initialization schemes that ensure some form of echo state property have to be investigated. The computational cost of training networks of the size of RC with back-propagation is also challenging. Another interesting topic is to analyze the influence of the amount of training data, and network size on the predictive efficiency of the methods, under the lens of the
recently discovered “double descent” curve phenomenon (Belkin, Hsu, Ma, & Mandal, 2019), where over-parametrized networks (increasing model capacity beyond the point of interpolation) results in improved generalization.

One topic not covered in this work is invertibility of the models, when forecasting the full state dynamics. Non-invertible models like the RNNs trained in this work, may suffer from spurious dynamics not present the training data and the underlying governing equations (Frouzakis, Gardini, Kevrekidis, Millerioux, & Mira, 1997; Gicquel, Anderson, & Kevrekidis, 1998). Invertible RNNs may constitute a promising alternative to further improve accurate short-term prediction and capturing of the long-term dynamics.

In conclusion, recurrent neural networks for data-driven surrogate modeling and forecasting of chaotic systems can efficiently be used to model high-dimensional dynamical systems, can be parallelized alleviating scaling problems and constitute a promising research subject that requires further analysis.

Data and code

The code and data that support the findings of this study are openly available in the GitHub repository https://github.com/pvlachas/RNN-RC-Chaos to assist reproducibility of the results. The software was written in Python utilizing Tensorflow (Abadi et al., 2016) and Pytorch (Paszke et al., 2017) for automatic differentiation and the design of the neural network architectures.

CRediT authorship contribution statement

P.R. Vlachas: Conceptualization, Methodology, Software, Data curation, Visualization, Writing - original draft. J. Pathak: Conceptualization, Methodology, Software, Data curation, Writing - review & editing. B.R. Hunt: Conceptualization, Supervision, Writing - review & editing. T.P. Sapsis: Conceptualization, Supervision, Writing - review & editing. M. Girvan: Conceptualization, Supervision. E. Ott: Conceptualization, Supervision, Writing - review & editing. P. Koumoutsakos: Conceptualization, Methodology, Supervision, Writing - review & editing.

Acknowledgments

We thank Ivica Kičić and Fabian Wermelinger for useful discussions and support on the cluster setup, and Guido Novati for valuable feedback on this manuscript. Moreover, we would like to acknowledge the time and effort of the Action Editor Herbert Jaeger along with two anonymous reviewers whose insightful and thorough feedback led to substantial improvements of the manuscript. TPS has been supported through the ARO-MURI grant W911NF-17-1-0306. This work has been supported at UMD by DARPA under grant number DMS51813027. PRV acknowledges support by the Collegium Helveticum transdisciplinary Institute for Advanced Study. We are also thankful to the Swiss National Supercomputing Center (CSCS) providing the necessary computational resources under project IDs s929 and s930.

Appendix A. Memory efficient implementation of RC training

In order to alleviate the RAM requirement for the computation of the RC weights we resort to a batched approach. Assuming the hidden reservoir size is given by \( h \in \mathbb{R}^{d_h} \), by teacher forcing the RC network with true data from the system for \( d_h \) time-steps and stacking the evolution of the hidden state in a single matrix we end up with matrix \( H \in \mathbb{R}^{d_h \times d_h} \). Moreover, by stacking the target values, which are the input data shifted by one time-step, we end up in the target matrix \( Y \in \mathbb{R}^{d_x \times d_t} \), where \( d_x \) is the dimension of the observable we are predicting. In order to identify the output weights \( W_{out} \in \mathbb{R}^{d_x \times d_h} \), we need to solve the linear system of \( d_h \cdot d_x \) equations

\[
H Y_{out} = Y.
\]

A classical way to solve this system of equations is based on the Moore–Penrose inverse (pseudo-inverse) computed using

\[
W_{out} = Y^t H (H^t H + \eta I)^{-1}
\]

where \( \eta \) is the Tikhonov regularization parameter and \( I \) the unit matrix. In our case \( d_h \) is of the order of \( 10^3 \) and \( d_h \gg d_x \). To reduce the memory requirements of the training method, we compute the matrices \( \tilde{H} = H^t H \in \mathbb{R}^{d_h \times d_h} \) and \( \tilde{Y} = Y^t H \in \mathbb{R}^{d_h \times d_x} \) in a time-batched schedule.

Specifically, we initialize \( \tilde{Y} = 0 \) and \( \tilde{H} = 0 \). Then every \( d_h \) time-steps with \( d_h \ll d_h \), we compute the batch matrix \( \tilde{H}_b = H^\dagger H_b \in \mathbb{R}^{d_h \times d_h} \) where \( H_b \in \mathbb{R}^{d_h \times d_h} \) is formed by the stacking of the hidden state only for the last \( d_h \) time-steps. In the same way, we compute \( \tilde{Y}_b = Y^t H_b \in \mathbb{R}^{d_h \times d_x} \), where \( Y_b \in \mathbb{R}^{d_x \times d_h} \) is formed by the stacking of the target data for the last \( d_h \) time-steps. After every batch computation we update our beliefs with \( \tilde{H} \leftarrow \tilde{H} + \tilde{H}_b \) and \( \tilde{Y} \leftarrow \tilde{Y} + \tilde{Y}_b \).

In addition, we also experimented with two alternative solvers for the linear system (Eq. (14)) in the Lorenz-96. We tried a dedicated regularized least-squares routine utilizing an iterative procedure (scipy.sparse.linalg.lsqqr) and a method based on stochastic gradient descent. We considered the solver as an additional hyperparameter of the RC models. After testing the solvers in Lorenz-96 systems, we found out that the method of pseudo-inverse provides the most accurate results. For this reason, and to spare computational resources, we used this method for the Kuramoto–Sivashinsky system.

Appendix B. Regularizing training with noise

In our study, we investigate the effect of noise to the training data. In Fig. 17, we plot the Valid Prediction Time (VPT) in the testing data with respect to the VPT that each model achieves in the training data. We find out that RC models trained with additional noise of 5–10% not only achieve better generalization, but their forecasting efficiency improves in both training and testing dataset. Moreover, the effect of divergent predictions by iterative forecasts is alleviated significantly. In contrast, adding noise does not seem to have an important impact on the performance of GRU models.

Appendix C. Dimensionality reduction with singular value decomposition

Singular Value Decomposition (SVD) can be utilized to perform dimensionality reduction in a dataset by identifying the modes that capture the highest variance in the data and then performing a projection on these modes. Assuming that a data matrix is given by stacking the time-evolution of a state \( u \in \mathbb{R}^D \) as \( U = [u_1, u_2, \ldots, u_N] \), where the index \( N \) is the number of data samples. By subtracting the temporal mean \( \mu \) and stacking the data, we end up with the data matrix \( U \in \mathbb{R}^{D \times D} \). Performing SVD on \( U \) leads to

\[
U = M \Sigma V^t, \quad M \in \mathbb{R}^{m \times N}, \quad \Sigma \in \mathbb{R}^{N \times D}, \quad V \in \mathbb{R}^{D \times D},
\]

with \( \Sigma \) diagonal, with descending diagonal elements. The columns of matrix \( V \) are considered the modes of the SVD, while the square \( D \) singular values of \( \Sigma \) correspond to the data variance explained by these modes. This variance is also referred to as
In order to calculate the percentage of the total energy the square of the singular value of each mode has to be divided by the sum of squares of the singular values of all modes. In order to reduce the dimensionality of the dataset, we first have to decide on the reduced order dimension $r_{\text{dim}} < D$. Then we identify the eigenvectors corresponding to the most high-energetic eigenmodes. These are given by the first columns $V_r$ of $V$, i.e., $V = [V_r, V_{-r}]$. We discard the low-energetic modes $V_{-r}$. The dimension of the truncated eigenvector matrix is $V_r \in \mathbb{R}^{D \times r_{\text{dim}}}$. In order to reduce the dimensionality of the dataset, each vector $u \in D$ is projected to $u_r \in r_{\text{dim}}$ by

$$c = V^T_r u, \quad c \in \mathbb{R}^{r_{\text{dim}}}. \quad (17)$$

In the Lorenz-96 system, we construct a reduced order observable with $d_o = 35$ modes of the system. The cumulative energy distribution along with a contour plot of the state and the mode evolution is illustrated in Fig. 18.

**Appendix D. Calculation of Lyapunov spectrum**

The true Lyapunov exponents of the KS equation are computed as in Pathak, Hunt et al. (2018) by solving the KS equations
in the Fourier space with a fourth order time–stepping method called ETDRK4 (Kassam & Trefethen, 2005) and utilizing a QR decomposition approach. The trained RNN model with GRU cell is used as a surrogate to compute the full Lyapunov spectrum of the Kuramoto–Sivashinsky system. Recall that the RNN dynamics are given by

\[ h_t = f^h_t(o_t, h_{t-1}) \]  
\[ o_{t+1} = f^o_t(h_t) \],

(18)

where \( f^h_t \) is the hidden-to-hidden and \( f^o_t \) is the hidden-to-output mapping. \( o \in \mathbb{R}^{d_o} \) is an observable of the state, and \( h_t \in \mathbb{R}^{d_h} \) is the hidden state of the RNN. All models utilized in this work share this common architecture. They only differ in the forms of \( f^h_t \) and \( f^o_t \). More importantly, the output mapping is linear, i.e.,

\[ o_{t+1} = f^o_t(h_t) = W_o h_t \].

The LEs are calculated based on the Jacobian \( J = \frac{dh_z}{dh_{z-1}} \) of the hidden state dynamics along the trajectory. In the following we compute the Jacobian using Eq. (18). By writing down the equations for two consecutive time–steps, we get

\[ \text{Time–step } t - 1 : h_{t-1} = f^h_t(o_{t-1}, h_{t-2}) \]  
\[ o_t = f^o_t(h_{t-1}) = W_o h_{t-1} \]  
\[ \text{Time–step } t : h_t = f^h_t(o_t, h_{t-1}) \].

(20)

(21)

(22)

The partial Jacobians needed to compute the total Jacobian are:

\[ \frac{\partial f^h_t}{\partial o} = f^h_t \in \mathbb{R}^{d_h \times d_o} \]  
\[ \frac{\partial f^h_t}{\partial h} = f^h_t \in \mathbb{R}^{d_h \times d_h} \]  
\[ \frac{\partial f^o_t}{\partial h} = f^o_t \in \mathbb{R}^{d_o \times d_h} \].

(23)

(24)

(25)

In total we can write:

\[ \frac{dh_t}{dh_{t-1}} = \frac{df^h_t(o_t, h_{t-1})}{dh_{t-1}} = \frac{df^h_t(o_t, h_{t-1})}{\partial o_t} \frac{\partial o_t}{\partial h_{t-1}} + \frac{df^h_t(o_t, h_{t-1})}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \]  
\[ \frac{dh_t}{dh_{t-1}} = \frac{df^h_t(o_t, h_{t-1})}{\partial o_t} \frac{\partial o_t}{\partial h_{t-1}} + \frac{df^o_t(o_t, h_{t-1})}{\partial h_{t-1}} \]  
\[ \frac{dh_t}{dh_{t-1}} = f^h_t \left|_{(o_t, h_{t-1})} \right. f^h_t \left|_{(o_t, h_{t-1})} \right. + f^o_t \left|_{(o_t, h_{t-1})} \right. \]  

evaluated at \( t \) \hspace{1cm} \text{evaluated at } t-1 \hspace{1cm} \text{evaluated at } t

(26)

(27)

(28)

A product of this Jacobian along the orbit \( \delta \) is developed and iteratively orthonormalized every \( T_h \) steps using the Gram–Schmidt method to avoid numerical divergence and keep the columns of the matrix \( R \) independent. We check the convergence criterion by tracking the estimated LE values every \( T_h \) time–steps. The input provided to the algorithm is a short time series of length \( T_w \) to initialize the RNN and warm-up the hidden state \( \tilde{o}_{1:T_w+1} \) (where the tilde denotes experimental or simulation data), the length of this warm–up time series \( T_w \), the number of the LE to calculate \( N \), the maximum time to unroll the RNN \( T \), a normalization time \( T_n \) and an additional threshold \( \epsilon \) used as an additional termination criterion. The function ColumnSum(·) computes the sum of each column of a matrix, i.e., \( \text{sum}(\cdot, \text{axis} = 1) \). This method can be applied directly to RNNs with one hidden state like RC or GRUs. An adaptation to the LSTM is left for future research. The pseudocode of the algorithm to calculate the Lyapunov exponents of the RNN is given in Algorithm 1.

Appendix E. Model hyperparameters

The hyperparameter values used to tune the models for forecasting the dynamics of the Lorenz-96 system in the reduced and full order PCA mode (observable) space with \( d_o \in \{35, 40\} \), are reported on Table 2 for RC, Table 3 for the GRU/LSTM models, and Table 4 for Unitary networks. For the parallel architectures in the state–space of Lorenz-96 the hyperparameters are reported on Table 5 for the parallel RC, and on Table 6 for the parallel GRU/LSTM models. For the parallel architectures forecasting the dynamics of the state–space of the Kuramoto–Sivashinsky equation the hyperparameters are reported on Tables 7 and 8 for the parallel RC and GRU/LSTM models respectively. We note here that in the case of the RNNs trained with BPTT, the optimizer used to update the network and its hyperparameters can also be optimized. To alleviate the computational burden we stick to Adam with its default hyperparameter values (\( \beta_1 = 0.9 \) and \( \beta_2 = 0.999 \)).

Appendix F. Additional results – Lorenz-96 – divergence of unitary and RC RNNs

In this section, we try to quantify the divergence effect due to the accumulation of the forecasting error in the iterative prediction. In Fig. 19 we present violin plots with fitted kernel density...
Fig. 20. Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz-96 system with $F = 8$ in the testing dataset with GRU, LSTM, RC and a Unitary network along with the true (target) evolution and the associated NRSE contours for the reduced order observable (a) $d_o = 35$ and the full state (b) $d_o = 40$. The evolution of the component average NRSE (NRMSE) is plotted to facilitate comparison. Unitary networks suffer from propagation of forecasting error and eventually their forecasts diverge from the attractor. Forecasts in the case of an observable dimension $d_o = 40$ diverge slower as the dynamics are deterministic. In contrast, forecasting the observable with $d_o = 35$ is challenging due to both (1) sensitivity to initial condition and (2) incomplete state information that requires the capturing of temporal dependencies. In the full-state setting, RC models achieve superior forecasting accuracy compared to all other models. In the challenging reduced order scenario, LSTM and GRU networks demonstrate a stable behavior in iterative prediction and reproduce the long-term statistics of the attractor. In contrast, in the reduced order scenario RC suffer from frequent divergence. The divergence effect is illustrated in this chosen initial condition. GRU , LSTM , RC-6000 , RC-9000 , Unit . (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

estimates for the number of divergent predictions of each hyperparameter set, computed based on all tested hyperparameter sets for forcing regimes $F \in \{8, 10\}$ and observable dimensions $d_o \in \{35, 40\}$. The annotated lines denote the minimum, mean and maximum number of divergent predictions over the 100 initial conditions of all hyperparameter sets. In the fully observable
Fig. 21. Forecasting results on the dynamics of an observable consisting of the SVD modes of the Lorenz-96 system with $F = 10$ and state dimension 40. The observable consists of the $d_o \in \{35, 40\}$ most energetic modes. (a), (d) VPT plotted w.r.t. the required RAM memory for $d_o \in \{35, 40\}$. (b), (e) VPT plotted w.r.t. training time for $d_o \in \{35, 40\}$. (c), (f) VPT measured from 100 initial conditions sampled from the test data plotted w.r.t. VPT from 100 initial conditions sampled from the training data for each model for $d_o \in \{35, 40\}$. In the reduced order scenario, RCs tend to overfit easier compared to GRUs/LSTMs that utilize validation-based early stopping. RC – GRU – LSTM – Unit – Ideal – – –. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

**Table 2**

Hyperparameters of RC for Lorenz-96.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Explanation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_r$</td>
<td>Reservoir size</td>
<td>{6000, 9000, 12000, 18000}</td>
</tr>
<tr>
<td>$N$</td>
<td>Training data samples</td>
<td>$10^5$</td>
</tr>
<tr>
<td>$d_o$</td>
<td>Degree of $W_h$</td>
<td>{3, 8}</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Radius of $W_h$</td>
<td>{0.4, 0.8, 0.9, 0.99}</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Input scaling</td>
<td>{0.1, 0.5, 1.0, 1.5, 2.0}</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Regularization</td>
<td>{10^{-1}, 10^{-3}, 10^{-5}, 10^{-6}}</td>
</tr>
<tr>
<td>$d_o$</td>
<td>Observed state dimension</td>
<td>{35, 40}</td>
</tr>
<tr>
<td>$n_w$</td>
<td>Warm-up steps (testing)</td>
<td>2000</td>
</tr>
<tr>
<td>$\kappa_n$</td>
<td>Noise level in data</td>
<td>{0, 0.5%, 1%}</td>
</tr>
</tbody>
</table>

**Table 3**

Hyperparameters of GRU/LSTM for Lorenz-96.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Explanation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_h$</td>
<td>Hidden state size</td>
<td>{1, 2, 3} layers of {500, 1000, 1500}</td>
</tr>
<tr>
<td>$N$</td>
<td>Training data samples</td>
<td>$10^5$</td>
</tr>
<tr>
<td>$B$</td>
<td>Batch-size</td>
<td>32</td>
</tr>
<tr>
<td>$k_2$</td>
<td>BPT forward time-steps</td>
<td>{1, 8}</td>
</tr>
<tr>
<td>$k_3$</td>
<td>BPT truncated backprop. length</td>
<td>{8, 16}</td>
</tr>
<tr>
<td>$\eta$</td>
<td>BPT skip gradient parameter</td>
<td>(= \frac{k_2 + k_1}{k_1} - 1)</td>
</tr>
<tr>
<td>$\theta$</td>
<td>Initial learning rate</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>$p$</td>
<td>Zoneout probability</td>
<td>{0.99, 0.995 1.0}</td>
</tr>
<tr>
<td>$d_o$</td>
<td>Observed state dimension</td>
<td>{35, 40}</td>
</tr>
<tr>
<td>$n_w$</td>
<td>Warm-up steps (testing)</td>
<td>2000</td>
</tr>
<tr>
<td>$\kappa_n$</td>
<td>Noise level in data</td>
<td>{0, 0.2%}</td>
</tr>
</tbody>
</table>

In the reduced order scenario, this divergence effect seems to be more prominent in RC and Unitary networks, as indicated by the very thin lower part of their violin plots, for both forcing regimes. In contrast, many hyperparameter sets of GRU and LSTM models lead to stable iterative prediction. This indicates that hyperparameter tuning in RC and Unitary networks when the systems $d_o = 40$, in both forcing regimes $F \in \{8, 10\}$, there are many models (hyperparameter sets) with zero divergent predictions for RC, GRU and LSTM, as illustrated by the wide lower part of the violin plot. In contrast, most hyperparameter sets in Unitary networks lead to models whose iterative predictions diverge from the attractor, as illustrated by the wide upper part of the violin plot. In the reduced order scenario, this divergence effect seems to be more prominent in RC and Unitary networks, as indicated by the very thin lower part of their violin plots, for both forcing regimes. In contrast, many hyperparameter sets of GRU and LSTM models lead to stable iterative prediction. This indicates that hyperparameter tuning in RC and Unitary networks when the
Fig. 22. Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz-96 system with $F = 10$ in the testing dataset with GRU, LSTM, RC and a Unitary network along with the true (target) evolution and the associated NRSE contours for the reduced order observable (a) $d_o = 35$ and the full state (b) $d_o = 40$. The evolution of the component average NRSE (NMSE) is plotted to facilitate comparison. Unitary networks suffer from propagation of forecasting error and eventually their forecasts diverge from the attractor. Forecasts in the case of an observable dimension $d_o = 40$ diverge slower as the dynamics are deterministic. In contrast, forecasting the reduced order observable with $d_o = 35$ is challenging due to both (1) sensitivity to initial condition and (2) incomplete state information that requires the capturing of temporal dependencies. In the full-state setting, RC models achieve superior forecasting accuracy compared to all other models. In the challenging reduced order scenario, LSTM and GRU networks demonstrate a stable behavior in iterative prediction and reproduce the long-term statistics of the attractor (attractor climate). In contrast, in the reduced order scenario iterative predictions of RC diverge frequently from the attractor. GRU $\times$; LSTM $\rightarrow$; RC-6000 $\rightarrow$; RC-9000 $\rightarrow$; Unit $\rightarrow$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)
The RC and the Unitary networks diverge in the reduced order system state is not fully observed, is cumbersome compared to LSTM and GRU networks. One example of this divergence effect in an initial condition from the test dataset is illustrated in Fig. 20. The RC and the Unitary networks diverge in the reduced order state predictions after approximately two Lyapunov times.

### Appendix G. Additional results – Lorenz-96 – \( F = 10 \)

In Fig. 21, we provide additional results for the forcing regime \( F = 10 \) that are in agreement with the main conclusions drawn in the main manuscript for the forcing regime \( F = 8 \). An example of a single forecast of the models starting from an initial condition in the test dataset is given in Fig. 22.

### Appendix H. Temporal dependencies and backpropagation

In our study, in order to train the GRU and LSTM models with back-propagation through time (BPTT), we need to tune the parameters \( \kappa_1 \) and \( \kappa_2 \). The first one denotes the truncated backpropagation length also referred to as sequence length and the second the number of future time-steps used to compute the loss and backpropagate the gradient during training at each batch. In the hyperparameter study considered in this work, we varied \( \kappa_1 \in \{8, 16\} \) and \( \kappa_2 \in \{1, 8\} \). For each of these hyperparameter sets, we varied all other hyperparameters according to the grid search reported in Appendix E.
Table 5
Hyperparameters of Parallel RC for Lorenz-96.

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Explanation</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_r$</td>
<td>Reservoir size</td>
<td>{1000, 3000, 6000, 12000}</td>
</tr>
<tr>
<td>$N_g$</td>
<td>Number of groups</td>
<td>20</td>
</tr>
<tr>
<td>$G$</td>
<td>Group size</td>
<td>2</td>
</tr>
<tr>
<td>$I$</td>
<td>Interaction length</td>
<td>4</td>
</tr>
<tr>
<td>$N$</td>
<td>Training data samples</td>
<td>$10^5$</td>
</tr>
<tr>
<td>Solver</td>
<td>Pseudoinverse/LSQR/Gradient descent</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>Degree of $W_{h,h}$</td>
<td>10</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Radius of $W_{h,h}$</td>
<td>0.6</td>
</tr>
<tr>
<td>$\omega$</td>
<td>Input scaling</td>
<td>0.5</td>
</tr>
<tr>
<td>$\eta$</td>
<td>Regularization</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>$d_0$</td>
<td>Observed state dimension</td>
<td>40</td>
</tr>
<tr>
<td>$n_w$</td>
<td>Warm-up steps (testing)</td>
<td>2000</td>
</tr>
</tbody>
</table>

Fig. 23. Violin plots of the VPT in the testing data for stateful LSTM and GRU models trained with different truncated Backpropagation through time parameters $\kappa_1$ and $\kappa_2$ in the (reduced) SVD mode observable of the Lorenz-96 system. The legend of each plot reports the models along with their $\kappa_1 - \kappa_2$ parameters used to train them. The three markers report the minimum, mean and maximum VPT. We observe that especially in the reduced order observable scenario ($d_0 = 35$), having a large truncated back-propagation parameter $\kappa_1$ (also referred to as sequence length) is vital to capture the temporal dependencies in the data and achieve high forecasting efficiency. In contrast in the full-state scenario ($d_0 = 40$) a model with a small back-propagation horizon suffices. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(a) $F = 8$

(b) $F = 10$

Fig. 24. The average VPT measured from 100 initial conditions sampled from the test dataset is plotted against the average VPT measured from 100 initial conditions sampled from the training dataset for parallel models forecasting the dynamics of (a) the Lorenz-96 system with state dimension $d_0 = 40$ and (b) the Kuramoto–Sivashinsky equation with state dimension $d_0 = 512$. RC □; GRU X; LSTM O. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(a) VPT in test w.r.t. VPT in train

(b) VPT in test w.r.t. VPT in train

In Fig. 23 we present a violin plot, that illustrates the forecasting efficiency of LSTM and GRU models trained with the listed $\kappa_1$ and $\kappa_2$ (legend of the plot), while varying all other hyperparameters. The forecasting efficiency is quantified in terms of the Valid Prediction Time (VPT) in the test dataset (averaged over 100 initial conditions) on the Lorenz-96 system for $F \in \{8, 10\}$. The three bars in each violin plot, denote the minimum, average and maximum performance.

In the reduced order scenario case, we observe that models with a large sequence length $\kappa_1$ and a large prediction length $\kappa_2$ are pivotal in order to achieve a high forecasting efficiency. This implies that there are temporal correlations in the data that cannot be easily captured by other models with smaller horizons. In contrast, in the full order scenario, models with smaller $\kappa_1$ perform reasonably well, demonstrating that the need of capturing temporal correlations in the data in order to forecast the evolution is less prominent, since the full information of the state of the system is available.

Appendix I. Over-fitting of parallel models

In this section, we provide results on the overfitting of the models in the parallel setting in the Lorenz-96 model in Fig. 24(a) and the Kuramoto–Sivashinsky equation in Fig. 24(b). In both cases we do not observe overfitting issues as the predictive performance in terms of the VPT of the models in the test dataset is very close to that in the training dataset, emphasizing that the generalization ability of the models is good.
References


