DISS. ETH NO. 28277

LEARNING AND FORECASTING THE EFFECTIVE DYNAMICS OF COMPLEX SYSTEMS ACROSS SCALES

A dissertation submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

PANTELIS RAFAIL VLACHAS
M.Sc., Electrical Engineering & Information Technology,
Technical University of Munich

born on October 28th 1993
citizen of Greece

accepted on the recommendation of

Prof. Dr. P. Koumoutsakos, examiner
Prof. Dr. Themistoklis Sapsis, co-examiner
Prof. Dr. Matej Praprotnik, co-examiner

2022

DOI: 10.3929/ethz-b-000551130

To my (extended) family.

ABSTRACT

Simulations of complex systems are essential for applications ranging from weather forecasting to molecular systems and drug design. The veracity of the resulting predictions hinges on their capacity to capture the underlying system dynamics. Massively parallel simulations performed in High-Performance Computing (HPC) clusters capture these dynamics by resolving all spatio-temporal scales. The computational cost is often prohibitive for experimentation or optimization, while their findings might not allow for generalization. The design of dimensionality reduction methods and fast data-driven reduced-order models or surrogates have been matters of life-long research efforts.

In the first part of this thesis, we focus on the design and training of data-driven recurrent neural networks for forecasting the spatio-temporal dynamics of high-dimensional and reduced-order complex systems. We propose architectural advances and training algorithms that alleviate the pitfalls of previously proposed methods, whose application was limited to lower-order systems. The designed algorithms extend the arsenal of predictive models for complex systems and spatio-temporal chaos.

In the second part, we present a novel systematic framework that bridges large-scale simulations and reduced-order models to Learn the Effective Dynamics (LED) of complex systems. The framework forms algorithmic alloys between non-linear machine learning algorithms and the Equation-Free approach for modeling complex systems exhibiting spatio-temporal chaos. LED deploys autoencoders to map between fine- and coarse-grained representations and evolves the latent space dynamics using recurrent neural networks. The algorithm is validated on benchmark problems, and we find that it outperforms state-of-the-art reduced-order models in terms of predictability and large-scale simulations in terms of cost. LED is applicable to systems ranging from chemistry to fluid mechanics and reduces the computational effort by up to two orders of magnitude while maintaining the prediction accuracy of the full system dynamics. We argue that LED constitutes a potent novel modality for the accurate prediction of complex systems.

ZUSAMMENFASSUNG

Die Simulation von komplexen Systeme sind für Anwendungen bei Wetterprognosen bis hin zu Entwicklung von Medikamenten unerlässlich. Die Richtigkeit der resultierenden Vorhersagen hängt von der Fähigkeit ab, die zugrunde liegende Systemdynamik vollständig zu erfassen. Massiv parallele Simulationen, die in High-Performance Computing (HPC)-Clustern durchgeführt werden, erfassen diese Dynamik, indem sie alle räumlichen und zeitlichen Skalen auflösen. Die Rechenkosten sind oft zu hoch um Experimente oder Optimierungen durchzuführen und die Ergebnisse dieser Simulationen lassen sich möglicherweise nicht verallgemeinern. Aus diesem Grund ist die Entwicklung von Methoden zur Dimensionalitätsreduktion und schneller, datengesteuerten Modellen der Gegenstand diverser Forschungsbemühungen.

Im ersten Teil dieser Arbeit konzentrieren wir uns auf den Entwurf und das Training rekurrenter neuronaler Netze zur Vorhersage der räumlich-zeitlichen Dynamik hochdimensionaler komplexer Systeme in reduzierter Ordnung. Wir schlagen architektonische Fortschritte und Trainingsalgorithmen vor, die die Probleme früherer Methoden, deren Anwendung auf Systeme niedrigerer Ordnung beschränkt war, beheben. Die entwickelten Algorithmen erweitern das Arsenal an Modellen für die Vorhersage von komplexen Systeme und räumlich-zeitlichem Chaos.

Im zweiten Teil stellen wir einen neuen Rahmen vor um massiv parallele Simulationen und Modelle in reduzierter Ordnung miteinander zu verbinden, um die effektive Dynamik verschiedener komplexer Systeme zu erlernen (LED). Der Rahmen bildet eine algorithmische Verknüpfung von nichtlinearen Algorithmen des maschinellen Lernens und dem gleichungsfreien Ansatz zur Modellierung komplexer Systeme, die ein räumlich-zeitliches Chaos aufweisen. LED setzt Autoencoder ein, um eine Abbildung zwischen fein- und grobkörnigen Repräsentationen zu finden, und entwickelt die so gefundene latente Raumdynamik mithilfe rekurrenter neuronaler Netze. Der Algorithmus wird anhand bekannter Problemen validiert, und wir stellen fest, dass er die existierenden Techniken in Bezug auf die Qualität der Vorhersagen und bei massiv parallele Simulationen in Bezug auf die Kosten übertrifft. LED ist auf Systeme aus der Chemie bis hin zu Systemen aus der Strömungsmechanik anwendbar und reduziert den Rechenaufwand um

bis zu zwei Größenordnungen, wobei die Vorhersagegenauigkeit erhalten bleibt. Wir legen dar, dass LED eine neuartige, wirksame Methode für die genaue Vorhersage komplexer Systeme darstellt.

ACKNOWLEDGEMENTS

First and foremost, I am grateful to my advisor, Prof. Petros Koumoutsakos. He gave me the chance to be part of a dynamic research environment, the CSElab, provided important research ideas and valuable advice, and helped me identify and filter the signal from the noise in the academic environment. His scientific guidance helped me grow scientifically and personally.

I would like to thank Prof. Themis Sapsis, Prof. Edward Ott, Dr. Zhong Yi Wan, Dr. Jaideep Pathak, and Dr. Wonmin Byeon for our fruitful collaborations. Special thanks go to Prof. Julija Zavadlav and Prof. Matej Praprotnik, who provided scientific advice and guidance and ensured an enjoyable collaboration.

I am also thankful to Ivica Kicic, Pascal Weber, Guido Novati, Fabian Wermelinger, Susanne Lewis, and all other members of the CSElab for their support throughout this journey. I will remember some of our exciting discussions about technical and -especially- non-technical topics.

Last but not least, I am incredibly thankful to my family and friends, too many to name here, in Athens, in Ioannina, and Zurich, especially to Aris, Kostas, and Sotiris. All of them constitute a vital supportive social structure, without which any professional or scientific step of mine would be of greater difficulty.

CONTENTS

NOMENCLATURE

Mathematical conventions

$(\cdot)^*$	Complex conjugate of $(\cdot)$
$(\cdot)_{t+1}, (\cdot)_{t+\Delta t}$	State $(\cdot)$ at next sampled timestep
Δt	Sampling interval
δt	Integrator timestep
$\langle \cdot \rangle$	Average of expression $\cdot$ (depending on context, statespace, time, or both)
$\mathbb{C}$	The set of complex numbers
$\mathbb{E}$	Expectation
$\mathcal{O}$	Big O notation for runtime requirements
$\odot$	Element-wise product
$\mathrm{COV}(\cdot_i, \cdot_j)$	Covariance between vectors $\cdot_i$ and $\cdot_j$
$\mathrm{DFT}(\cdot)$	Discrete Fourier transform of $\cdot$
$\mathrm{PSD}(\cdot)$	Power spectral density of signal $\cdot$
$\mathrm{vec}(\cdot)$	Vectorization of vector $\cdot$
$\overline{(\cdot)}$	Average of state $\cdot$ (depending on context, statespace, time, or both)
$\mathbb{R}$	The set of real numbers
tanh	Hyperbolic tangent activation

Latin & Greek symbols

$(\cdot)^w$	Neural network $(\cdot)$ parametrized with weights $\boldsymbol{w}$
$(\cdot)_t$	State vector $(\cdot)$ at timestep t
β_1, β_2	Hyperparameters of Adam optimizer
$\boldsymbol{F}$	Function of the dynamics
$\boldsymbol{I}$	Unit matrix
$\boldsymbol{b}$	Bias of neural network
$\boldsymbol{c}$	Cell state of RNN
$\boldsymbol{g}$	Gate vector signals of RNN
$\boldsymbol{h}$	Hidden state of RNN
$\boldsymbol{o}$	Observable (full state, reduced order, or latent state)
$\boldsymbol{\sigma}$	Vector of standard deviations

$\widetilde{\boldsymbol{o}}$	Prediction of observable
$\widetilde{\boldsymbol{z}}$	Prediction of latent state
$\boldsymbol{W}$	Matrix of weights of the neural network
$\boldsymbol{w}$	Weights of neural network
$\boldsymbol{w}^\star$	Optimized weights of neural network
$\boldsymbol{x}, \boldsymbol{s}$	State vector
$\boldsymbol{z}$	Latent state vector
$\mathcal{N}$	Normal distribution
$\cdot \sim p(\cdot)$	State $\cdot$ is sampled according to probability density p
$\chi^{(s)}$	Characteristic function
$\mathrm{diag}(\cdot)$	Diagonal matrix with diagonal elements given by the vector $\cdot$
ε	User defined threshold
η	Learning rate
exp	Exponentiation to Euler's number
$\boldsymbol{u}$	Velocity field
$\hat{\boldsymbol{h}}$	Augmented hidden state of RNN
$\hat{X}$	Fourier components
$\mathcal{F}_{hh}$	Hidden-to-hidden mapping of an RNN
$\mathcal{H}_h, \mathcal{H}_o$	Contributions of the previous hidden state and the current state to the next hidden state of RNN
$\mathcal{F}_{ho}$	Hidden-to-output mapping of an RNN
κ_1	Prediction horizon of BPTT
κ_2	Truncation (backpropagation) length of BPTT
κ_3	Teacher forcing length of BPTT
κ_B	Boltzmann's constant
Λ	Lyapunov exponent
Λ_1	Maximal Lyapunov exponent
ln	Natural logarithm
$\mathbf{H}, \boldsymbol{Y}$	Data matrices in RC training
$\mathcal{D}$	Decoder network
$\mathcal{E}$	Encoder network
$\mathcal{L}$	Neural network loss
$\mathcal{T}$	Temperature
$\mathcal{Z}$	Low-order manifold of the dynamics
μ	Viscosity
$\widetilde{\rho}$	Flow density field

ν	Viscosity parameter
Ω	Simulation domain
ω	Omega
$\mathrm{Im}(\cdot)$	Imaginary part of complex state $\cdot$
$\mathrm{Re}(\cdot)$	Real part of complex state $\cdot$
π	Mixing coefficients of Mixture Model
ρ	Ratio between macro and micro simulation time in LED
$\rho^{ac}(x,t)$	Activator density in FHN
$\rho^{in}(x,t)$	Inhibitor density in FHN
Σ	Covariance matrix
σ	Standard deviation
$\sigma(\cdot)$	Standard deviation of state $\cdot$
$\sigma_{attractor}$	Standard deviation of the state space
σ_{en}	Ensemble standard deviation
σ_{noise}	Noise standard deviation
$\widetilde{\eta}$	Tikhonov regularization parameter
$\tilde{X}, \tilde{t}$	Scaled state and time of Lorenz 96
ϱ	Spectral radius of a matrix
C_d	Drag coefficient
d_h	Dimension of hidden state
d_r, d_z	Dimension of reduced order state
$d_{(\cdot)}$	Dimension of state vector $(\cdot)$
D_{cyl}	Cylinder diameter
E	Dirichlet energy
E_p	Average energy fluctuation
F	Forcing regime of Lorenz 96
f_{vs}	Vortex shedding frequency
G	Group size of parallel model
I	Interaction length of parallel model
J_o^{hh}, J_h^{hh}, J_h^{oh}	Jacobians of RNN states
L	Boundary size (domain)
L_v^k	Lower-triangular matrix
N_g	Number of network members in parallel model
N_{en}	Ensemble size
$N_{patience}$	Patience in epochs for validation based adaption of the learning rate

N_{rounds}	Number of rounds for validation based adaption of the learning rate
N_{train}	Number of training samples
p	Pressure field
$p(\cdot)$	Probability distributions
S	Speed-up
T	Simulation time
t	Discretized timestep
T^{Λ_1}	Lyapunov time
T_m	Simulation time in macro-scale
T_μ	Simulation time in micro-scale
T_f	Final simulation time
$T_{i \to j}$	MFPT or transition time from metastable state i to j
T_{warm}	Warm-up time of non-Markovian model
$u(x,t), v(x,t)$	Spatiotemporal field
u_x	Partial derivative $\partial u / \partial x$
u_{xx}	Partial derivative $\partial^2 u / \partial x^2$
v_{cyl}	Cylinder speed
x, x_i, x^i	Component i of the state vector

Dimensionless numbers

Re	Reynolds number
St	Strouhal number

Acronyms

1D	One-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
ACC	Anomaly Correlation Coefficient
AE	Autoencoder
AE-RNN	Autoencoder RNN
BPTT	Backpropagation Through Time
BPTT-SA	Scheduled Autonomous BPTT
BPTT-SS	BPTT with Scheduled Sampling
CG	Coarse Graining
CNN	Convolutional Neural Network
CNN-RNN	Convolutional Autoencoder RNN

ConvRNN	RNN with convolutional filters
CPU	Central Processing Unit
CSCS	Swiss National Super-computing Centre
CSPDE	EFF variants identifying PDEs on the coarse-grained state
CVs	Collective Variables
DFT	Discrete Fourier Transform
DiffMaps	Diffusion Maps
DMD	Dynamic Mode Decomposition
EFF	Equation-Free Framework
end2end	Network trained in an end-to-end fashion
EOFs	Empirical Orthogonal Functions
ESN	Echo State Network
FHN	FitzHugh-Nagumo
FLAVOR	FLow AVeraged integatoR
GP	Gaussian Process
GPR	Gaussian Process Regression
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
HMM	Heterogeneous Multiscale Method
HPC	High Performance Computing
KS	Kuramoto-Sivashinsky
Latent-LED	LED macro dynamics (no alteration)
LB	Lattice Boltzmann
LE	Lyapunov Exponent
LED	Learning Effective Dynamics
LS	Lyapunov Exponent Spectrum
LSM	Liquid State Machine
LSTM	Long Short-Term Memory
MBP	Müeller-Brown potential
MD	Molecular Dynamics
MDN	Mixture Density Network
MeSM	Mean Stochastic Model
MFPT	Mean first-passage time
MG	Mackey-Glass
ML	Machine Learning
MLE	Maximal Lyapunov Exponent
MLP	Multilayer Perceptron (Feedforward Neural Network)

MNAD	Mean Normalized Absolute Difference
MPI	Message Passing Interface
MSE	Mean Squared Error
Multiscale-LED	LED with switching between micro and macro dynamics
NAD	Normalized Absolute Difference
NN	Neural Network
NRMSE	Normalized Root Mean Squared Error
ODE	Ordinary Differential Equation
PCA	Principal Component Analysis
PDE	Partial Differential Equation
RAM	Random Access Memory
RAVE	Reweighted Autoencoded Variational Bayes for Enhanced sampling
RC	Reservoir Computing
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SNR	Signal to Noise Ratio
SVD	Singular Value Decomposition
TICA	Time-lagged Independent Component Analysis
Unitary	Unitary evolution RNN cell
VAE	Variational Autoencoder
VPT	Valid Prediction Time

1

INTRODUCTION

1.1 MULTISCALE SYSTEMS

Some of the most important scientific advances and engineering designs are founded on the study of complex systems that exhibit nonlinear dynamics spanning multiple spatio-temporal scales. Examples include protein dynamics (Rackovsky et al., 2020), morphogenesis (Gilmour et al., 2017), brain dynamics (P. A. Robinson et al., 2005), the climate (Council, 2012; Warrach-Sagi et al., 2013), ocean dynamics (A. Mahadevan, 2016) and social systems (Bellomo et al., 2011). Over the last fifty years, simulations have become a key component of these studies thanks to a confluence of advances in computing architectures, numerical methods, and software. Large scale simulations have led to unprecedented insight, acting as in-silico microscopes (E. H. Lee et al., 2009) or telescopes to reveal the dynamics of galaxy formations (Springel et al., 2005). Massively parallel simulations (Dennis et al., 2008; S.-J. Kim et al., 2008; Rasthofer et al., 2017) in High-Performance Computing (HPC) architectures aim at resolving all spatio-temporal scales, often at a prohibitive computational cost. At the same time, these simulations have led to the understanding that in many complex physical phenomena resolving the full range of spatio-temporal scales will remain out of reach in the foreseeable future.

A remedy is found by the design of reduced-order models (ROMs) (Forrester et al., 2006) that reduce the complexity of the physical phenomena under study and resolve only a part, e. g. the most energetic, of the system scales. The computational cost of these models is lower than methods resolving all scales, but their accuracy is limited by the approximations involved, like linearization of the dynamics or heuristic closure schemes to incorporate the effects of unmodelled system scales (Novati, de Laroussilhe, et al., 2021).

In recent years there have been intense efforts to develop efficient simulations that exploit the multiscale character of the systems under investigation (Car et al., 1985; Erban et al., 2006; Kevrekidis, Gear, and Hummer, 2004; Kevrekidis, Gear, Hyman, et al., 2003; Kevrekidis and Samaey, 2009;

Weinan, Engquist, et al., 2003). Multiscale methods rely on prudent approximations of the interactions between processes occurring over different scales, and a number of potent frameworks have been proposed, including the Equation-Free Framework (EFF) (Bar-Sinai et al., 2019; Kevrekidis, Gear, and Hummer, 2004; Kevrekidis, Gear, Hyman, et al., 2003; Laing et al., 2010), the Heterogeneous Multiscale Method (HMM) (Weinan, Engquist, et al., 2003, 2007; Weinan, X. Li, et al., 2004), and the FLow AVeraged integatoR (FLAVOR) (M. Tao et al., 2010). These algorithms distinguish the system dynamics into fine (fully resolved) and coarse (reduced-order) scales or expensive and affordable simulations. Their success depends on the separation of scales inherent to the system dynamics and their capability to capture the transfer of information between scales. Successful applications of multiscale methodologies minimize the computational effort while maximizing the accuracy of the propagated dynamics. Multiscale methodologies rely on three components, dimensionality reduction methods to identify a reduced-order (latent) embedding, timestepping procedures to propagate the reduced-order dynamics, and "lifting" operators to return to the fine-scale descriptions.

A wide range of methods have been proposed in the context of dimensionality reduction, from linear methods (Cunningham et al., 2015), e. g. Principal Component Analysis (PCA), or Dynamic Mode Decomposition (DMD) (Kutz, Brunton, et al., 2016), to nonlinear (J. A. Lee et al., 2007), e. g. Diffusion Maps (Coifman and Lafon, 2006) or kernel based methods (Bittracher, Klus, et al., 2021). The development of efficient timestepping procedures in the reduced-order space and, more importantly, accurate methods to return to the fine-scale description from the reduced-order one have been hindered by the complex nonlinear nature of the reduced-order dynamics and the difficulty in identifying nonlinear operators to recover the full-scale description from reduced-order information. Although previously proposed multiscale frameworks like the EFF, HMM, and FLAVOR, have revolutionized the field, their generalization and scalability to nonlinear, high-dimensional systems is hindered by these two critical issues: the accuracy of the latent integrator and the encoding and lifting operators that significantly affect the accuracy of the methods.

1.2 MOLECULAR SYSTEMS

Over the last 30 years, molecular dynamics (MD) simulations of biological macromolecules have advanced our understanding of their structure and

function (Karplus et al., 2002). Today MD simulations have become an essential tool for scientific discovery in biology, chemistry, and medicine. However, they remain hampered by their limited access to timescales of biological relevance for protein folding pathways, conformational dynamics, and rare-event kinetics.

In order to resolve this bottleneck, two complementary approaches have been pursued. First efforts centered around innovative hardware solutions started with crowdsourcing for computing cycles (Shirts et al., 2000) and have more recently received a boost with the Anton machine (Shaw et al., 2009) enabling remarkable, millisecond-long simulations of bio-molecules. Complementary algorithmic efforts aim to advance timescales by systematic coarse-graining of the system dynamics. One of the first such studies used the principal component or normal mode analysis to simulate the conformational changes in proteins (Balsera et al., 1996; Brooks et al., 1983; Ichiye et al., 1991; Praprotnik and Janežič, 2005; Skjaerven et al., 2011). Several coarse-graining (CG) methods reduce the complexity of molecular systems by modeling several atoms as a single particle (Noid, 2013; Wagner et al., 2016; Zavadlav et al., 2019). Backmapping techniques (Hess et al., 2006; Pezeshkian et al., 2020; Stieffenhofer et al., 2020) can be subsequently utilized to recover the atomistic degrees of freedom from a CG representation. Multiscale approaches combine the atomistic and coarse-grained/continuum models (Ayton et al., 2007; Praprotnik, Site, et al., 2008; Werder et al., 2005) to augment the accessible timescales while significant efforts have focused on enhanced sampling techniques (Dellago et al., 1998; Huber et al., 1994; Inizan et al., 2021; Laio et al., 2002; Maragliano et al., 2006; Van Erp et al., 2003; Voudouris, 1998). Several of these methods exploit the fact that coarse kinetic dynamics on the molecular level are often governed by a few, slow collective variables (CVs), also termed reaction coordinates (Bittracher, Banisch, et al., 2018; Bonati et al., 2020; Peters et al., 2006; Stamati et al., 2010), or by transitions between a few long-lived metastable states (Bittracher, Koltai, et al., 2018; Schütte et al., 2011).

The CVs are typically specified a priori and their choice crucially impacts the performance and success of the respective sampling methods. Like the CG models, the CVs provide a low-order representation of the molecular system, albeit without a particle representation. CVs have much lower dimensionality than CG models, and retrieving atomistic configurations from CVs is a more challenging problem. While many research efforts have addressed the fine to coarse mapping in CG models, the literature is still scarce on retrieving atomistic configurations from CVs.

1.3 CHAOTIC SYSTEMS

Complex systems are typically chaotic and challenging to predict, a critical issue in weather and climate forecasting problems. Minor prediction errors propagate exponentially, rendering long-term forecasts inaccurate. The works of Takens (Takens, 1981) and Sauer, Yorke, and Casdagli (Sauer et al., 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 has been the subject of life-long research efforts (Bradley et al., 2015).

Efforts to comprehend and forecast the dynamics of such complex, chaotic systems have spurred developments in large-scale simulations, dimensionality reduction techniques, and forecasting methods. The goals of understanding and prediction have been complementing each other but have been hindered by the high-dimensionality and chaotic behavior of these systems. We have observed a convergence of these approaches in recent years due to advances in computing power, algorithmic innovations, and data availability. A major beneficiary of this convergence are data-driven dimensionality reduction methods (Arbabi et al., 2017; Kerschen et al., 2005; Kutz, Fu, et al., 2016; Rowley, 2005; Sapsis et al., 2013; Tu, 2013; M. O. Williams et al., 2015), model identification procedures (Bongard et al., 2007; Brunton, Proctor, et al., 2016; Duriez et al., 2017; Farazmand et al., 2016; Krischer et al., 1993; A. J. Majda and Y. Lee, 2014; Milano et al., 2002; Schaeffer, 2017) and forecasting techniques (Abdollahzade et al., 2015; Comeau et al., 2017; Cousins et al., 2014, 2016; Einicke et al., 1999; Julier et al., 1997; Y. Lee et al., 2016; Y. Lin et al., 2011; Marques et al., 2006; Quade et al., 2016) that aim to provide precise short-term predictions while capturing the long-term statistics of these systems. Successful forecasting methods address the highly nonlinear energy transfer mechanisms between modes not captured effectively by the dimensionality reduction methods.

1.4 MACHINE LEARNING

Over the last years, machine learning (ML) algorithms have exploited the widespread availability of data, and powerful computing architectures (Kurth et al., 2018), to provide us with remarkable successes across scientific disciplines from physics (Baldi et al., 2014; Brunton, Proctor, et al.,

2016; Champion et al., 2019; Lusch et al., 2018; Novati, L. Mahadevan, et al., 2019; Raissi et al., 2019; Wan and Sapsis, 2018), biology (Alipanahi et al., 2015; Jumper et al., 2021), fluid dynamics (Brunton, Noack, et al., 2019, 2020; Wan and Sapsis, 2018), climate modeling (Kurth et al., 2018), mathematics (Davies et al., 2021; Han et al., 2018), signal processing (Oord, Dieleman, et al., 2016), image and language processing (Pennington et al., 2014), computer vision (Gregor, Danihelka, Graves, et al., 2015; Krizhevsky et al., 2017), to medicine and drug discovery (H. Chen et al., 2018). ML offers potent tools that can address the challenges in the design of multiscale algorithms for the simulation of multiscale systems from fluid flows to molecules. Moreover, in cases where models based on first-principles (e. g. mathematical equations derived from physics) are not available, but some underlying dynamics are manifested in the form of available data from an observable of the system, data-driven ML methods can be employed to derive surrogate or ROMs that mimick the dynamics.

Recurrent Neural Networks (RNNs) are ML architectures tailored for sequential or time series data (Y. Bengio et al., 1994; Goodfellow et al., 2016; Hochreiter and Schmidhuber, 1997; Pascanu et al., 2013). RNNs are universal function approximators (Schäfer et al., 2006; Siegelmann et al., 1995) and can capture nonlinear and non-Markovian dynamics. The potential of RNNs for capturing temporal dynamics in physical systems was explored first using low-dimensional RNNs (Elman, 1990) with simple cell architectures to predict unsteady boundary-layer development, separation, dynamic stall, and dynamic reattachment back in 1997 (Faller et al., 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 (Ahmad et al., 2004; Gregor, Danihelka, Graves, et al., 2015) is largely attributed to a cell architecture termed Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). LSTMs employ gates that effectively remember and forget information, thus alleviating the problem of vanishing gradients (Hochreiter and Schmidhuber, 1997). In Wan and Sapsis, 2018 LSTM networks are used as surrogates to model the kinematics of spherical particles in fluid flows. In Wan, P. Vlachas, 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. LSTM-RNNs offer a potent alternative on the one hand to model and analyze complex, chaotic spatio-temporal dynamics and on the other hand, to model the reduced-order dynamics in multiscale algorithms. Nevertheless, data-driven forecasting of chaotic or complex dynamics with RNNs is

challenging for four main reasons, (i) partial observability (reduced-order information of the state), (ii) under-resolved dynamics (insufficient training data near the attractor boundaries), (iii) vanishing gradients during training of RNNs (Hochreiter, 1998), (iv) distributional shift in long-term forecasting during testing (as predictions of the model are iteratively propagated).

Autoencoders (AEs) (Kramer, 1991) are powerful nonlinear dimensionality reduction methods that can embed the physical properties of the system under study or properties of the data in the form of a geometric prior, e. g. energy constraints, translation invariance using Convolutional Neural Networks (CNNs) (Gu et al., 2018), or permutation invariance (Noé, Tkatchenko, et al., 2020). AEs have been used to identify a linear latent space based on the Koopman framework (Lange et al., 2021; Lusch et al., 2018), or model high-dimensional fluid flows (Geneva et al., 2020; Milano et al., 2002). AEs coupled with LSTMs are used in Gonzalez et al., 2018; Hasegawa et al., 2020; Maulik et al., 2021 to model fluid flows. Mixture Density Networks (MDN) (Bishop, 1994) are expressive data-driven models of probability density distributions. They are potent algorithms that can be employed to model stochastic dynamics or probabilistic mappings from the reduced-order space to the full-scale description in multiscale algorithms. The utilization of AEs and MDNs to identify accurate operators to recover the full-scale description from reduced-order information in multiscale algorithms has been unexplored.

1.5 CONTRIBUTIONS

This thesis presents an extensive analysis of data-driven machine learning algorithms and novel training procedures for modeling chaotic spatio-temporal dynamics and introduces a versatile framework for accelerating multiscale simulations of complex systems by learning their effective dynamics (LED). The effectiveness of the LED framework is demonstrated in a wide range of applications from fluid flows to molecular systems, accelerating the full-scale solvers by orders of magnitude without sacrificing accuracy. The ML methods developed in this thesis extend the arsenal of powerful computational tools available to the broader scientific community that cope with the challenges of chaos, multiscale problems, or problems where models based on first principles are not available. A list of publications is appended to this thesis.

Chapter 3: Coupling an LSTM-RNN with a Mean Stochastic Model

In this chapter, we demonstrate how LSTM-RNNs can be employed to forecast effectively high-dimensional spatio-temporal chaotic dynamics. We tackle the first two issues of data-driven forecasting with RNNs, namely (i) partial observability and (ii) under-resolved dynamics (insufficient training data near the attractor boundaries). These issues are addressed by proposing an LSTM based prediction framework coupled with an equation-based Mean Stochastic Model (MeSM). The LSTM is learning local dynamics from data. The MeSM is constructed to mimic the global attractor dynamics. A hybrid network is proposed, where the MeSM is utilized on attractor regions where data is not available. The methods are benchmarked against state-of-the-art algorithms based on Gaussian Process Regression (GPR) in the Lorenz-96 system, the Kuramoto-Sivashinsky equation, and a Barotropic climate model. We show that the LSTM-RNN is superior in short-term forecasting, but predictions eventually diverge from the attractor and become unphysical due to the iterative error propagation and undersampled regions near the boundaries. In contrast, we demonstrate that the hybrid method (LSTM-MeSM) exhibits lower error in the long term, and the error eventually converges to the invariant measure.

Chapter 4: Training Algorithms and Scalability of RNNs for Forecasting Dynamical Systems

In this chapter, we concentrate on the vanishing gradients problem of RNNs and scalability to high-dimensional spatio-temporal chaotic systems. We study four prominent RNN cell architectures in terms of their effectiveness in short-term predictability and long-term behavior (capturing attractor statistics) in both cases of full state observability and reduced-order state (partial observability). The methods are benchmarked in the Lorenz-96 system and the Kuramoto-Sivashinsky equation. We find that fixing the hidden internal dynamics of the RNN cell, according to the Reservoir Computing paradigm, can be beneficial in the case of full state information. In the more challenging case of reduced-order information of the state, which is more relevant in practice, gated architectures show superior performance. Moreover, we demonstrate how a trained RNN can be utilized to compute the Lyapunov exponent spectrum of a dynamical system in a data-driven manner. We present a parallel framework to scale these methods to very

high-dimensional spatio-temporal systems in case of a fully observable state with local dependencies.

Chapter 5: Scheduled Autoregressive Backpropagation Through Time for Long-Term Forecasting

In this chapter, we focus on the problem of accumulated prediction errors in iterative forecasting with RNNs, and propose a novel training method that considers the iterative forecasting (autoregressive) error. We propose a scheduled approach, where the standard one step ahead prediction loss is minimized at early training epochs, while the loss is gradually switching to the autoregressive version at later stages. We benchmark our approach to other state-of-the-art training methods demonstrating that it leads to better long-term predictive performance in forecasting the dynamics of the Mackey-Glass time series and the viscous Navier-Stokes past behind a cylinder in a channel at Reynolds number $\mathrm{Re} = 100$.

Chapter 6: Learning Effective Dynamics

In this chapter, we present a framework to accelerate multiscale simulations of complex systems by learning their effective dynamics (LED). The LED resolves two critical issues of previously proposed multiscale simulation frameworks: the accuracy of the timestepper in the reduced-order space and the identification of encoding and accurate lifting operators. This is achieved by ML algorithms that (i) deploy RNNs with gating mechanisms to evolve the coarse-grained dynamics and (ii) employ advanced AEs to transfer in a systematic, data-driven manner the information between coarse and fine-scale descriptions. The framework is benchmarked on the FitzHugh-Nagumo equation, the Kuramoto-Sivashinsky equation, and the Navier-Stokes flow past a cylinder at $\mathrm{Re} \in \{100, 1000\}$. The LED is orders of magnitude faster than the micro-scale solvers while maintaining predictive accuracy. By switching between the reduced-order latent dynamics (LSTM-RNN propagator of LED) and propagation of the full high-dimensional dynamics using the micro-scale solver, the predictive accuracy is further increased at the cost of reduced speed-up. We demonstrate that LED unravels the inertial manifold of the Kuramoto-Sivashinsky equation and reproduces the long-term climate of the dynamics. We show that LED captures the long-term dynamics of the Navier-Stokes flow past a cylinder at the challenging

case of $\mathrm{Re} = 1000$, even though propagating drastically reduced-order latent state dynamics compared to other literature. Moreover, a benchmark with other latent state propagators indicates that the LSTM-RNN is the most prominent, demonstrating high predictive accuracy in all applications.

Chapter 7: LED for Molecular Systems

This chapter extends LED to incorporate probabilistic dynamics on the latent space and apply it to molecular systems. Other state-of-the-art approaches rely on memory-less (Markovian) latent dynamics. LED alleviates this assumption by employing probabilistic RNNs on the latent space and probabilistic Autoencoders to map from a reduced-order description to molecular configurations. We demonstrate the effectiveness of LED to capture the timescales, kinetics, and statistics in Langevin particle dynamics in the Müeller-Brown potential and prototypical molecular systems, i.e., the alanine dipeptide and the Trp Cage miniprotein. LED accurately reproduces the kinetics and statistics in the molecular systems while being three orders of magnitude faster than the molecular dynamics solver.

We summarize the contribution of this work and provide avenues for future research in Chapter 8.

2

PRELIMINARIES

2.1 NEURAL ARCHITECTURES

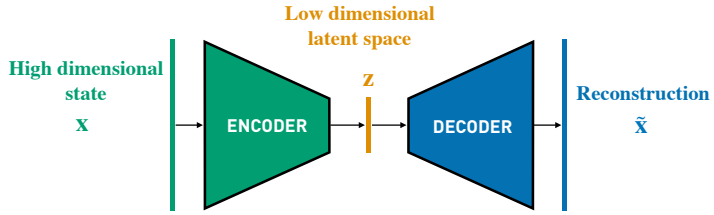
2.1.1 *Autoencoders*

Classical autoencoders are nonlinear neural networks that map an input to a low-dimensional latent space and then decode it to the original dimension at the output, trained to minimize the reconstruction loss $\mathcal{L} = |\boldsymbol{x} - \widetilde{\boldsymbol{x}}|^2$. They were proposed in Kramer, 1991 as a nonlinear alternative to Principal Component Analysis (PCA). An autoencoder is depicted in Figure 2.1(a).

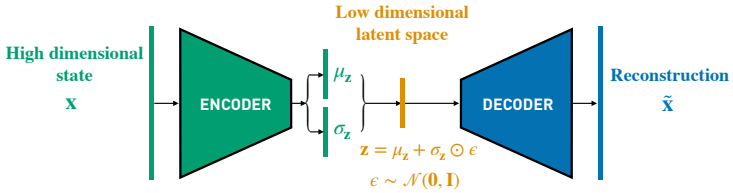
2.1.2 *Variational Autoencoders*

Research efforts on generative modeling led to the development of Variational Autoencoders (VAEs). The VAE, similar to AE is composed of an encoder and a decoder. The encoder neural network, instead of mapping the input $\boldsymbol{x}$ deterministically to a reduced-order latent space $\boldsymbol{z}$, produces a distribution $q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{w}_q)$ over the latent representation $\boldsymbol{z}$, where $\boldsymbol{w}_q$ is the parametrization of the distribution given by the output of the encoder $\boldsymbol{w}_q = \mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{x})$. In most practical applications, the distribution $q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{w}_q)$ is modeled as a factorized Gaussian, implying that $\boldsymbol{w}_q$ is composed of the mean, and the diagonal elements of the covariance matrix. The decoder maps a sampled latent representation to an output $\widetilde{\boldsymbol{x}} = \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}(\boldsymbol{z})$. By sampling the latent distribution $q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{w}_q)$, for a fixed input $\boldsymbol{x}$, the autoencoder can generate samples from the probability distribution over $\widetilde{\boldsymbol{x}}$ at the decoder output. The network is trained to maximize the log-likelihood of reproducing the input at the output, while minimizing the Kullback-Leibler divergence between the encoder distribution $q(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{w}_q)$ and a prior distribution, e. g. $\mathcal{N}(0, \boldsymbol{I})$. VAEs are essentially regularizing the training of AE by adding the Gaussian noise in the latent representation. Here, we consider a Gaussian latent distribution with diagonal covariance matrix is considered, i.e.,

$$q(\boldsymbol{z} \,|\, \boldsymbol{x}; \boldsymbol{\mu}_z, \boldsymbol{\sigma}_z) = \mathcal{N}\big(\boldsymbol{z} \,|\, \boldsymbol{\mu}_z(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}_z(\boldsymbol{x}))\big), \tag{2.1}$$



(a) Classical Autoencoder



(b) Variational Autoencoder

FIGURE 2.1: (a) A schematic diagram of a classical Autoencoder (AE). A high-dimensional state $\boldsymbol{x}$ is mapped to a low-dimensional feature space $\boldsymbol{z}$ by applying the encoder transformation through multiple fully connected layers. The low-dimensional feature space $\boldsymbol{z}$ is expanded in the original space by the decoder. The autoencoder is trained with the loss $\mathcal{L} = ||\boldsymbol{x} - \tilde{\boldsymbol{x}}||^2$, so that the input can be reconstructed as faithfully as possible at the decoder output. (b) A schematic diagram of a Variational Autoencoder (VAE). Instead of modeling the latent space deterministically, the encoder outputs a mean latent representation $\boldsymbol{\mu_z}$, along with the associated uncertainty $\boldsymbol{\sigma_z}$. The latent space $\boldsymbol{z}$ is sampled from a normal distribution $\boldsymbol{z} \sim \mathcal{N}(\cdot|\boldsymbol{\mu_z}, \text{diag}(\boldsymbol{\sigma_z}))$, with diagonal covariance matrix.

where $\boldsymbol{w}_q = (\boldsymbol{\mu}_z, \boldsymbol{\sigma}_z)$ and the mean latent representation $\boldsymbol{\mu}_z$ and the variance $\boldsymbol{\sigma}_z$ vectors are the outputs of the encoder neural network $\mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{x})$. The latent representation is then sampled from $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}_z, \mathrm{diag}(\boldsymbol{\sigma}_z))$. The decoder receives as an input the sample and outputs the reconstruction $\widetilde{\boldsymbol{x}}$. A VAE is depicted in Figure 2.1(b).

2.1.3 *Convolutional Neural Networks*

Convolutional neural networks (CNNs) are tailored to process image data with spatial correlations. Each layer of a CNN is processing a multi-dimensional input (with a channel axis and some spatial axes) by applying a convolutional kernel or filter that slides along the input spatial axes. In other words, CNNs consider the structure in the data in their architecture, which is a form of a geometric prior. Here, CNN layers are used in the autoencoder by introducing a bottleneck layer, reducing the dimensionality. Other dimensionality reduction techniques, like AEs, Principal Component Analysis (PCA) (Shlens, 2014), or Diffusion maps (DiffMaps) (Coifman, Kevrekidis, et al., 2008) that are based on vectorization of input field data do not take into account the structure of the data, i. e. when an input field is shifted by a pixel, the vectorized version will differ a lot, while the convoluted image will not.

2.1.4 *Recurrent Neural Networks (RNNs)*

A Recurrent Neural Network (RNN) is a machine learning architecture designed to process sequential data, i. e. temporal time series. In this work, we consider RNNs for time series forecasting. The models are trained on time series of an observable $\boldsymbol{o}$ (e. g. latent state $\boldsymbol{z}$ with reduced-order information, or full state $\boldsymbol{s}$) sampled at a fixed rate $1/\Delta t$, $\{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T\}$, where we eliminate Δt from the notation for simplicity. The models possess an internal high-dimensional hidden state denoted by $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ that enables the encoding of temporal dependencies on past state history. Given the current input (observable) $\boldsymbol{o}_t$, the output of each model is a forecast $\widetilde{\boldsymbol{o}}_{t+1}$ for the observable at the next time instant $\boldsymbol{o}_{t+1}$. This forecast is a function of the hidden state. In this way, RNNs can capture nonlinear and non-Markovian dynamics.

The forecasting rule of the RNN is given by

$$\boldsymbol{h}_t = \mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}\left(\boldsymbol{o}_t, \boldsymbol{h}_{t-1}\right), \quad \widetilde{\boldsymbol{o}}_{t+1} = \mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}\left(\boldsymbol{h}_t\right), \tag{2.2}$$

where $\mathcal{F}_{hh}$ is the hidden-to-hidden mapping and $\mathcal{F}_{ho}$ is the hidden-to-output mapping. $\boldsymbol{w}_{\mathcal{F}_{hh}}$ and $\boldsymbol{w}_{\mathcal{F}_{ho}}$ are the trainable parameters of $\mathcal{F}_{hh}$ and $\mathcal{F}_{ho}$ respectively. $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is an internal hidden memory state, and $\widetilde{\boldsymbol{o}}_{t+1}$ is a prediction of the observable.

All recurrent models in this work share this common architecture. They differ in the realizations of $\mathcal{F}_{hh}$ and $\mathcal{F}_{ho}$ and how the parameters or weights of these functions are learned from data, i.e., trained, to forecast the dynamics.

The RNN is trained to minimize the forecasting loss $||\widetilde{\boldsymbol{o}}_{t+1} - \boldsymbol{o}_{t+1}||_2^2$, which can be written as

$$||\widetilde{\boldsymbol{o}}_{t+1} - \boldsymbol{o}_{t+1}||_2^2 = ||\mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}\left(\boldsymbol{h}_t\right) - \boldsymbol{o}_{t+1}||_2^2 = \tag{2.3}$$

$$= ||\mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}\left(\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}\left(\boldsymbol{o}_t, \boldsymbol{h}_{t-1}\right)\right) - \boldsymbol{o}_{t+1}||_2^2. \tag{2.4}$$

This leads to

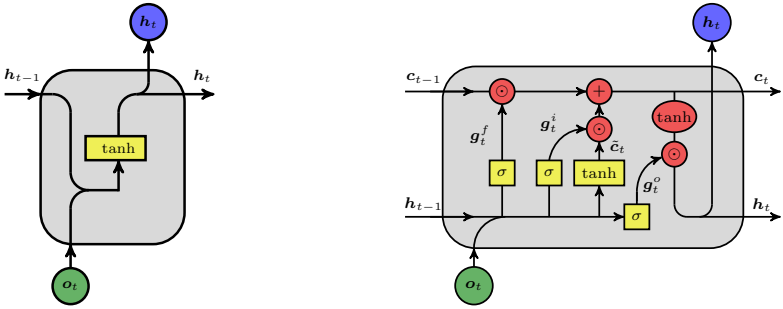
$$\boldsymbol{w}_{\mathcal{F}_{hh}}^{\star}, \boldsymbol{w}_{\mathcal{F}_{ho}}^{\star} = \underset{\boldsymbol{w}_{\mathcal{F}_{hh}}, \boldsymbol{w}_{\mathcal{F}_{ho}}}{\arg\min} ||\mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}\left(\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}\left(\boldsymbol{o}_t, \boldsymbol{h}_{t-1}\right)\right) - \boldsymbol{o}_{t+1}||_2^2. \tag{2.5}$$

RNNs are commonly trained with Backpropagation Through Time (BPTT) (P. J. Werbos, 1988). In many practical applications, RNNs suffer from the vanishing (or exploding) gradient problem and have failed to capture long-term dependencies Y. Bengio et al., 1994; Hochreiter, 1991, 1998. Today, the RNNs owe their renaissance to gating architectures that cope effectively with the abovementioned problem. Gating architectures have been successfully applied in sequence modeling Schmidhuber et al., 2005, speech recognition Graves, Mohamed, et al., 2013, hand-writing recognition Graves, Fernández, et al., 2008 and language translation Y. Wu et al., 2016.

The output mapping is given by a linear transformation, i.e.

$$\widetilde{\boldsymbol{o}}_{t+1} = \boldsymbol{W}_{h,o}\boldsymbol{h}_t + \boldsymbol{b}_o, \tag{2.6}$$

where $\boldsymbol{W}_{h,o} \in \mathbb{R}^{d_o \times d_h}$, $\boldsymbol{b}_o \in \mathbb{R}^{d_o}$. As a consequence, the set of trainable weights of the hidden-to-output mapping is just one matrix $\boldsymbol{w}_{\mathcal{F}_{ho}} = \boldsymbol{W}_{h,o} \in \mathbb{R}^{d_o \times d_h}$ and one vector $\boldsymbol{b}_o$.



(a) Elman RNN Cell (b) LSTM Cell

FIGURE 2.2: The information flow in an (a) Elman cell and (b) a Long Short-Term Memory (LSTM) cell. The LSTM cell employs gating mechanisms to alleviate the vanishing gradients problem of Elman RNNs. The gating mechanisms allow forgetting and storing information in the hidden state processing. Ellipses and circles denote entry-wise operations, while rectangles denote layer operations.

2.1.4.1 *Elman RNN*

The Elman RNN possesses a hidden state $\boldsymbol{h}$. The hidden-to-hidden mapping $\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}$ is defined by

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{h,o}\boldsymbol{o}_t + \boldsymbol{W}_{h,h}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h\right), \tag{2.7}$$

where $\boldsymbol{b}_h \in \mathbb{R}^{d_h}$, $\boldsymbol{W}_{h,o} \in \mathbb{R}^{d_h \times d_o}$, and $\boldsymbol{W}_{h,h} \in \mathbb{R}^{d_h \times d_h}$. The dimension of the hidden state d_h (number of hidden units) controls the capacity of the cell to encode history information. The set of trainable parameters of the recurrent mapping $\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}$ is given by

$$\boldsymbol{w}_{\mathcal{F}_{hh}} = \{\boldsymbol{b}_h, \boldsymbol{W}_{h,o}, \boldsymbol{W}_{h,h}.\} \tag{2.8}$$

An illustration of the information flow in an Elman RNN is given in Figure 2.2(b).

2.1.4.2 *Long Short-Term Memory Unit (LSTM)*

In Elman RNNs (Elman, 1990), the vanishing or exploding gradients problem stems from the fact that the gradient is multiplied repeatedly during backpropagation through time (P. J. Werbos, 1988) with a recurrent

weight matrix. Consequently, when the spectral radius of the weight matrix is positive (negative), the gradients are prone to explode (shrink). The LSTM (Hochreiter and Schmidhuber, 1997) was introduced in order to alleviate the vanishing gradient problem of Elman RNNs by leveraging gating mechanisms that allow information to be forgotten. The LSTM possesses two hidden states, a cell state $\boldsymbol{c}$ and an internal memory state $\boldsymbol{h}$. The equations that implicitly define the hidden-to-hidden (recurrent mapping $\mathcal{F}_{hh}$) mapping

$$\boldsymbol{h}_t, \boldsymbol{c}_t = \mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}\left(\boldsymbol{o}_t, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}\right) \tag{2.9}$$

of the LSTM are given by

$$\begin{aligned} \boldsymbol{g}_t^f &= \mathrm{act}_f\left(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{o}_t] + \boldsymbol{b}_f\right) & \boldsymbol{g}_t^i &= \mathrm{act}_i\left(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{o}_t] + \boldsymbol{b}_i\right) \\ \tilde{\boldsymbol{c}}_t &= \tanh\left(\boldsymbol{W}_c[\boldsymbol{h}_{t-1}, \boldsymbol{o}_t] + \boldsymbol{b}_c\right) & \boldsymbol{c}_t &= \boldsymbol{g}_t^f \odot \boldsymbol{c}_{t-1} + \boldsymbol{g}_t^i \odot \tilde{\boldsymbol{c}}_t \\ \boldsymbol{g}_t^o &= \mathrm{act}_h\left(\boldsymbol{W}_h[\boldsymbol{h}_{t-1}, \boldsymbol{o}_t] + \boldsymbol{b}_h\right) & \boldsymbol{h}_t &= \boldsymbol{g}_t^o \odot \tanh(\boldsymbol{c}_t), \end{aligned} \tag{2.10}$$

where $\boldsymbol{g}_t^f, \boldsymbol{g}_t^i, \boldsymbol{g}_t^o \in \mathbb{R}^{d_h}$, are the gate vector signals (forget, input and output gates), $\boldsymbol{o}_t \in \mathbb{R}^{d_o}$ is the observable input at time t, $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is the hidden state, $\boldsymbol{c}_t \in \mathbb{R}^{d_h}$ is the cell state, while $\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_c, \boldsymbol{W}_h \in \mathbb{R}^{d_h \times (d_h + d_o)}$, are weight matrices and $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c, \boldsymbol{b}_h \in \mathbb{R}^{d_h}$ biases. The symbol $\odot$ denotes the element-wise product. The activation functions act_f, act_i and act_h are sigmoids. For a more detailed explanation of the LSTM cell architecture 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 set of trainable parameters of the recurrent mapping $\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}$ is given by

$$\boldsymbol{w}_{\mathcal{F}_{hh}} = \{\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c, \boldsymbol{b}_h, \boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_c, \boldsymbol{W}_h\} \tag{2.11}$$

The forget gate bias is frequently initialized to one according to Bowman et al., 2015 to accelerate training. An illustration of the information flow in an LSTM cell is given in Figure 2.2(b).

2.1.5 *Mixture Density Networks*

Mixture density networks (MDNs) (Bishop, 1994) are powerful neural networks that can model non-Gaussian, multi-modal data distributions. The outputs of MDNs are parameters of a mixture density model (mixture of probability density functions). The most generic choice of the mixture component distribution is the Gaussian distribution. Gaussian MDNs are

widely deployed in machine learning applications to model structured dynamic environments, i. e. (video) games. The effectiveness of MDNs, however, in modeling physical systems remains unexplored.

In physical systems, the state may be bounded. In this case, the choice of a Gaussian MDN is problematic due to its unbounded support. To make matters worse, most applications of Gaussian MDNs when modeling random vectors do not consider the interdependence between the vector variables, i. e. the covariance matrix of the Gaussian mixture components is diagonal, in an attempt to reduce their computational complexity. Arguably in the applications where they were successful, modeling this interdependence was not imperative. In contrast, in physical systems, the variables of a state might be very strongly dependent on each other. In order to cope with these problems, the following approach is considered: We assume that the input to the MDN is a reduced-order latent state $\boldsymbol{z}$. We further assume that at the output of the MDN we want to model the distribution of a high-dimensional state $\boldsymbol{x}$. Firstly, an auxiliary vector variable is considered $\boldsymbol{v}$ along with its distribution $p(\boldsymbol{v}|\boldsymbol{z})$. $\boldsymbol{v} \in \mathbb{R}^{d_x}$ has the same dimensionality d_x as the high-dimensional state (input/output of the autoencoder). The distribution is modeled as a mixture of K multivariate normal distributions

$$p(\boldsymbol{v}|\boldsymbol{z}) = \sum_{k=1}^{K} \pi^k(\boldsymbol{z})\, \mathcal{N}\Big(\boldsymbol{\mu}_v^k(\boldsymbol{z}), \Sigma_v^k(\boldsymbol{z}) \Big), \tag{2.12}$$

The multivariate normal distribution is parametrized in terms of a mean vector $\boldsymbol{\mu}_v^k$, a positive definitive covariance matrix Σ_v^k, and the mixing coefficients π^k which are functions of $\boldsymbol{z}$. The covariance matrix is parametrised by a lower-triangular matrix L_v^k with positive-valued diagonal entries, such that $\Sigma_v^k = L_v^k L_v^{k\,T} \in \mathbb{R}^{d_x \times d_x}$ (This triangular matrix can be recovered by Cholesky factorization of the positive definite Σ_v^k). The functional forms of $\pi^k(\boldsymbol{z}) \in \mathbb{R}$, $\boldsymbol{\mu}_v(\boldsymbol{z}) \in \mathbb{R}^{d_x}$, and the $n(n+1)/2$ entries of L_v^k are neural networks, their values are given by the outputs of the decoder for all mixture components $k \in \{1, \ldots, K\}$, i. e. $\boldsymbol{w}_\mathcal{D} = \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}(\boldsymbol{z}) = \{\pi^k, \boldsymbol{\mu}_v^k, L_v^k\}_{1,\ldots,K}$. The positivity of the diagonal elements of L_v^k is ensured by a softplus activation function

$$f(x) = \ln(1 + \exp(x)) \tag{2.13}$$

in the respective outputs of the decoder. The mixing coefficients satisfy $0 \leq \pi^k < 1$ and $\sum_{k=1}^{K} \pi^k = 1$. To ensure these conditions, the respective outputs of the decoder are passed through a softmax activation

$$\text{softmax}(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_i e^{\boldsymbol{x}_i}}. \tag{2.14}$$

The rest (non-diagonal elements and mean vector) of the decoder outputs have linear activations, so there is no restriction in their sign. In total, the decoder output is composed of $K(n-1)n/2 + Kn$ single-valued outputs with linear activation for the non-diagonal elements of L_v^k and the mean vectors $\boldsymbol{\mu}_v^k$, and Kn positive outputs with "softplus" activation for the diagonal of L_v^k, and K outputs with softmax activation for the mixing coefficients.

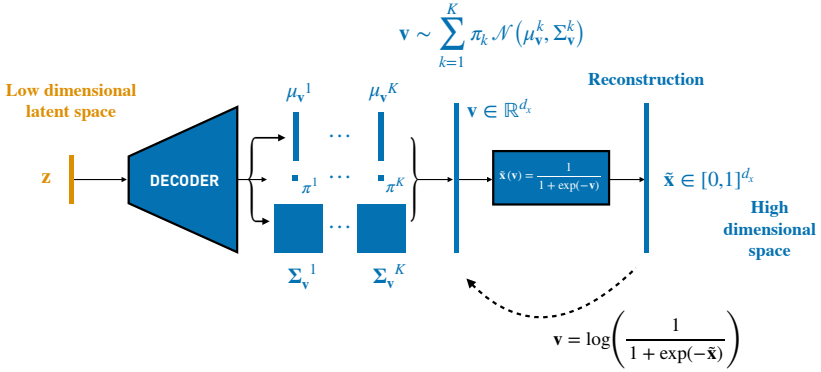


FIGURE 2.3: A mixture density network modeling the probability density $p(\tilde{\boldsymbol{x}}|\boldsymbol{z})$, with bounded $\tilde{\boldsymbol{x}}$. The decoder maps the latent state $\boldsymbol{z}$ to the parameters of a mixture model on the latent vector $\boldsymbol{v} \in \mathbb{R}^{d_x}$, which are the mixing coefficients $\pi^k \in \mathbb{R}$, mean vectors $\boldsymbol{\mu}_v^k \in \mathbb{R}^{d_x}$, and a lower-triangular matrix $L_v^k \in \mathbb{R}^{d_x \times d_x}$ with positive-valued diagonal entries. From the latter, the covariance matrix is derived from $\Sigma_v^k = L_v^k L_v^{k\,T}$ which is positive definite by construction. The mixture models the probability distribution of the latent state $p(\boldsymbol{v}|\boldsymbol{z})$. The targets, however, used to train the network in a supervised way are defined on the reconstruction $\tilde{\boldsymbol{x}}$. The targets are scaled to $\tilde{\boldsymbol{x}} \in [0,1]^{d_x}$, and then transformed to targets for $\boldsymbol{v}$ using the inverse of the softplus activation. The MDN autoencoder is trained to maximize the likelihood $p(\boldsymbol{v}|\boldsymbol{z})$ of the transformed data $\boldsymbol{v}$.

MDNs can be employed in stochastic systems, e. g. in order to model molecular systems. We assume that the high-dimensional state of the molecular system is described by $\boldsymbol{s}_t \in \mathbb{R}^{d_s}$. The decoder $\mathcal{D}^{\boldsymbol{w}_\mathcal{D}}$ is modeled with an MDN. The MDN approximates the probability distribution of the state $\tilde{\boldsymbol{s}}_t \sim p(\cdot; \boldsymbol{w}_{\text{MDN}})$, where $\boldsymbol{w}_{\text{MDN}} = \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}(\boldsymbol{z}_t)$ is the output of the decoder that

parametrizes the distribution. The optimal parameters of the MDN autoencoder are identified by maximizing the log-likelihood of the reconstruction,

$$\boldsymbol{w}_{\mathcal{E}}^{\star}, \boldsymbol{w}_{\mathcal{D}}^{\star} = \underset{\boldsymbol{w}_{\mathcal{E}}, \boldsymbol{w}_{\mathcal{D}}}{\arg\max} \log\, p\big(\boldsymbol{s}_t; \boldsymbol{w}_{\text{MDN}}\big),$$
$$\text{where} \quad \boldsymbol{w}_{\text{MDN}} = \mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}(\boldsymbol{z}_t) = \mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}\big(\mathcal{E}^{\boldsymbol{w}_{\mathcal{E}}}(\boldsymbol{s}_t)\big).$$

2.2 DYNAMICAL SYSTEMS

2.2.1 *The Kuramoto-Sivashinsky Equation*

The Kuramoto-Sivashinsky (KS) equation is a nonlinear partial differential equation (PDE) of fourth order, first derived by Kuramoto (Kuramoto, 1978; Kuramoto and Tsuzuki, 1976) as a turbulence model of the phase gradient of a slowly varying amplitude in a reaction-diffusion type medium with negative viscosity coefficient. Later, Sivashinsky (Sivashinsky, 1977) studied the spontaneous instabilities of the plane front of a laminar flame ending up with the KS equation, while in Sivashinsky and Michelson, 1980 the KS equation is found to describe the surface behavior of viscous liquid in a vertical flow.

In this work, we restrict ourselves to the one-dimensional (1D) KS equation with boundary and initial conditions given by

$$\frac{\partial u}{\partial t} = -\nu \frac{\partial^4 u}{\partial x^4} - \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x}, \tag{2.15}$$

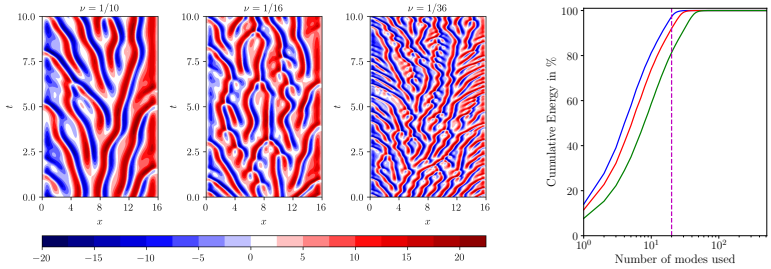
where $u(x,t)$ is the modeled quantity of interest depending on a spatial variable $x \in [0, L]$ and time $t \in [0, \infty)$. L denotes the dimensionless boundary size. The negative viscosity is modeled by the parameter $\nu > 0$. In order to spatially discretize Equation 2.15 we select a grid size Δx 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 $i \in \{0, \ldots, D-1\}$. Discretization using a second order differences scheme yields

$$\frac{du_i}{dt} = -\nu \frac{u_{i-2} - 4u_{i-1} + 6u_i - 4u_{i+1} + u_{i+2}}{\Delta x^4} - \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} - \frac{u_{i+1}^2 - u_{i-1}^2}{4\Delta x}. \tag{2.16}$$

The Kuramoto-Sivashinsky equation exhibits different levels of chaos depending on the bifurcation parameter $\widetilde{L} = L/2\pi\sqrt{\nu}$ (Kevrekidis, Nicolaenko, et al., 1990). Higher values of $\widetilde{L}$ (or smaller values of ν) lead to more

chaotic systems (Wan and Sapsis, 2017). For large values of L, the attractor dimension scales linearly with L (Manneville, 1984).

A plot of the KS dynamics with Dirichlet and second-type boundary conditions is given in Figure 2.4(b).



(a) (b)

FIGURE 2.4: (a) Contour plots of the solution $u(x,t)$ of the Kuramoto-Sivashinsky equation for different values of ν in steady-state. Chaoticity rises with smaller values of ν. (b) Cumulative energy as a function of the number of the PCA modes for different values of ν.
$1/\nu = 10$ ——; $1/\nu = 16$ ——; $1/\nu = 36$ ——; 20 modes - - -

2.2.2 *The Lorenz 96 Model*

In E. Lorenz, 1995 a model of the large-scale behavior of the mid-latitude atmosphere is introduced. This model describes the time evolution of the components x_j for $j \in \{0, 1, \ldots, J-1\}$ of a spatially discretized (over a single latitude circle) atmospheric variable. The atmospheric variable is denoted with $\boldsymbol{x} = [x_0, \ldots, x_{J-1}]$. In the following we refer to this model as the Lorenz 96. The Lorenz 96 is employed in Basnarkov et al., 2012; Wan and Sapsis, 2017 as a toy problem to benchmark methods for weather prediction.

Lorenz 96 is described by a system of coupled ordinary differential equations (ODEs) given by

$$\frac{dx_j}{dt} = (x_{j+1} - x_{j-2})x_{j-1} - x_j + F, \tag{2.17}$$

for $j \in \{0, 1, \ldots, J-1\}$, where by definition we assume periodic boundary conditions $x_{-1} = x_{J-1}$, $x_{-2} = x_{J-2}$. The right-hand side of Equation 2.17

consists of a non-liner adjective term $(x_{j+1} - x_{j-2})x_{j-1} - x_j$, a linear advection (dissipative) term $-x_j$ and a positive external forcing term F. The discrete energy of the system remains constant throughout time and the Lorenz 96 states x_j remain bounded. By increasing the external forcing parameter F the behavior that the system exhibits changes from periodic $F < 1$ to weakly chaotic ($F = 4$) to end up in fully turbulent regimes ($F = 16$). These regimes can be observed in Figure 2.5.

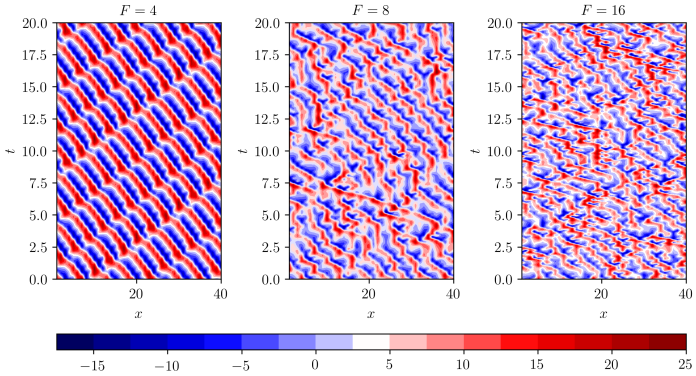


FIGURE 2.5: Lorenz 96 contour plots for different forcing regimes F. Chaoticity rises with bigger values of F.

3

COUPLING AN LSTM-RNN WITH A MEAN STOCHASTIC MODEL

3.1 RELATED WORK

In the context of modeling and prediction of chaotic dynamical systems, early application of neural networks can be traced to the work of (Lapedes et al., 1987), where they demonstrated the efficiency of feedforward artificial neural networks, in this case, multilayer perceptrons (MLPs), to model deterministic chaos. As an alternative to MLPs, wavelet networks were proposed in L. Cao et al., 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. In Lusch et al., 2018, MLPs are employed to identify an embedding space with linear dynamics that is then amenable to theoretical analysis. RNNs are practical and efficient data-driven approximators of chaotic dynamical systems due to their (1) universal approximation ability (Schäfer et al., 2006; Siegelmann et al., 1995) and (2) ability to capture temporal dependencies and implicitly identify the required embedding for forecasting. Early applications of RNNs to chaotic dynamics were limited by the vanishing gradients problem, and focused on low-order systems (Gers, Eck, et al., 2002). Similarly, other machine learning techniques based on Multi Layer Perceptrons (MLP) (Rico-Martinez et al., 1992), Echo State Networks (ESNs) (Chatzis et al., 2011; Jaeger et al., 2004), Reservoir Computing (RC) (Lukoševičius and Jaeger, 2009), or radial basis functions (Broomhead et al., 1988; K. B. Kim et al., 2000) have been successful, albeit only for low-order dynamical systems. In recent years (Bianchi et al., 2017) RNN architectures have been benchmarked for short-term load forecasting of demand and consumption of resources in a supply network, while in Laptev et al., 2017 they are utilized for extreme event detection in low-dimensional time series. RC and LSTM networks are applied in the long-term forecasting of partially observable chaotic chimera states in Neofotistos et al., 2019. Instead of an utterly model-free approach, ground-truth measurements of currently observed states are helping to improve the long-term forecasting capability.

Historically, the pioneering technique of analog forecasting proposed in E. N. Lorenz, 1969 inspired widespread research in non-parametric prediction approaches. Two dynamical system states are called analogues if they resemble one another on the basis of a specific criterion. This class of methods uses a training set of historical observations of the system. The system evolution is predicted using the evolution of the closest analogue from the training set corrected by an error term. This approach has led to promising results in practice (Xavier et al., 2007) but the selection of the resemblance criterion to pick the optimal analogue is far from straightforward. Moreover, the geometrical association between the current state and the training set is not exploited. More recently (Zhao et al., 2016), analog forecasting is performed using a weighted combination of data pointss based on a localized kernel that quantifies the similarity of the new point and the weighted combination. This technique exploits the local geometry instead of selecting a single optimal analogue. Similar kernel-based methods, (Chiavazzo et al., 2014) use diffusion maps to parametrize a low-dimensional manifold capturing the slower timescales globally. Moreover, non-trivial interpolation schemes are investigated to encode the system dynamics in this reduced-order space and map them to the full space (lifting). Although the geometrical structure of the data is taken into account, the solution of an eigensystem with a size proportional to the training data is required, rendering the approach computationally expensive. In addition, the inherent uncertainty due to sparse observations in some areas of the attractor introduces prediction errors that cannot be modeled in a deterministic context. In Wan and Sapsis, 2017 a method based on Gaussian process regression (GPR) (Rasmussen, 2003; C. K. Williams et al., 2006) was proposed for prediction and uncertainty quantification in the reduced-order space. The technique is based on a training set that sparsely samples the attractor. Stochastic predictions exploit the geometrical relationship between the current state and the training set, assuming a Gaussian prior over the modeled latent variables. A key advantage of GPR is that uncertainty bounds can be analytically derived from the hyperparameters of the framework. Moreover, in Wan and Sapsis, 2017 a Mean Stochastic Model (MeSM) is used for under-sampled regions of the attractor to ensure accurate modeling of the steady-state in the long-term regime. However, the resulting inference and training have a quadratic cost in terms of the number of data samples.

RNNs, offer a powerful method for addressing the challenges of previous approaches in data-driven prediction of complex, chaotic dynamics whose

application was limited to low-order problems. 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 they may be expensive to discretize and evaluate, let alone control, in real-time. The application of RNNs to chaotic dynamics can be challenging due to (i) partial observability (reduced-order information of the state), and (ii) under-resolved dynamics (insufficient training data near the attractor boundaries).

This chapter addresses these issues by proposing LSTM-RNN based methods that exploit information of the recent history of the reduced-order state to predict the high-dimensional dynamics. Time series data are used to train the model while no knowledge of the underlying system equations is required. Inspired by Taken's theorem (Takens, 1981) an embedding space is constructed using time-delayed versions of the reduced-order variable. The proposed method tries to identify an approximate forecasting rule globally for the reduced-order space. In contrast to GPR (Wan and Sapsis, 2017), the method has a deterministic output while its training cost scales linearly with the number of training samples, and it exhibits an $\mathcal{O}(1)$ inference computational cost. Moreover, following Wan and Sapsis, 2017, the LSTM is combined with a MeSM, to cope with attractor regions that are not captured in the training set. In attractor regions, under-represented in the training set, the MeSM is used to guarantee convergence to the invariant measure and avoid exponential growth of the prediction error. The effectiveness of the proposed hybrid method in accurate short-term prediction and capturing the long-term behavior is demonstrated in the Lorenz 96 and the Kuramoto-Sivashinsky equation. Finally, the method is also tested on predictions of a prototypical climate model.

The structure of the chapter is as follows: In Section 3.2 we explain how the LSTM can be employed for modeling and prediction of a dynamical system, and a blended LSTM-MeSM technique is introduced. In Section 3.3 three other state-of-the-art methods, GPR, MeSM and the hybrid GPR-MeSM scheme, are presented, and two comparison metrics are defined. The proposed LSTM technique and its LSTM-MeSM extension are benchmarked against GPR and GPR-MeSM in three complex, chaotic systems in Section 3.4. In Section 3.6 we discuss the computational complexity of training and inference in LSTM. Finally, Section 3.7 offers a summary of the results.

This chapter is based on the paper "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks" (P. R. Vlachas, Byeon, et al., 2018). The computational resources were provided by a grant from the Swiss National Supercomputing Centre (CSCS) under project s929.

3.2 METHODS

This chapter considers the reduced-order problem where the state of a dynamical system is projected in the reduced-order space. The system is considered to be autonomous, while $\dot{\boldsymbol{z}}_t = \frac{d\boldsymbol{z}_t}{dt}$ is the system state derivative at timestep t.

3.2.1 *Training the LSTM Recurrent Neural Network*

Following Gers, Eck, et al., 2002, The LSTM is trained using time series data from the system in the reduced-order space $D = \{\boldsymbol{z}_{1:T}, \dot{\boldsymbol{z}}_{1:T}\}$ to predict the reduced state derivative $\dot{\boldsymbol{z}}_t$ from a short history of the reduced-order state $\{\boldsymbol{z}_t, \boldsymbol{z}_{t-1}, \ldots, \boldsymbol{z}_{t-\kappa_2+1}\}$ consisting of κ_2 past temporally consecutive states. We approximated the derivative from the original time series using first order forward differences. The loss that has to be minimized is defined as The short-term history for the states before $\boldsymbol{z}_d$ is not available. That is why in total, we have $T - \kappa_2 + 1$ training samples from a time series with T samples. During training the weights of the LSTM are optimized according to $\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} \mathcal{L}(D, \boldsymbol{w})$. The parameter κ_2 is denoted as truncation layer, and time dependencies longer than κ_2 are not explicitly captured in the loss function.

Training of this model is performed using Backpropagation through time, truncated at layer κ_2 and mini-batch optimization with the Adam method (Kingma et al., 2014) with an adaptive learning rate (initial learning rate $\eta = 0.0001$). The LSTM weights are initialized using the method of Xavier (Glorot et al., 2010). Training is stopped when convergence of the training error is detected or the maximum of 1000 epochs is reached. During training, the loss of the model is evaluated on a separate validation dataset to avoid overfitting. The training procedure is explained in detail in Appendix A.1.

An important issue is how to select the hidden state dimension d_h and how to initialize the LSTM states $\boldsymbol{h}_{t-\kappa_2}, \boldsymbol{c}_{t-\kappa_2}$ at the truncation layer κ_2. A small d_h reduces the expressive capabilities of the LSTM and deteriorates inference performance. On the other hand, a large d_h is more sensitive to overfitting, and the computational cost of training rises. For this reason, d_h has to be tuned depending on the observed training behavior. We performed a grid search and selected the optimal d_h for each application. For the truncation layer κ_2, there are two alternatives, namely "stateless" and "stateful" LSTM. In "stateless" LSTM the LSTM states at layer κ_2 are initialized to zero. Consequently, the LSTM can only capture dependencies up to κ_2 previous timesteps. In the second variant, the "stateful" LSTM, the state is iteratively propagated for multiple timesteps in the future. The systems considered exhibit chaotic behavior, and the dependencies are inherently short-term, as the states in two timesteps that differ significantly can be considered statistically independent. For this reason, the short temporal dependencies can be captured without propagating the hidden state for a long horizon. As a consequence, we consider only the "stateless" variant. We also applied "stateful" LSTM without any significant improvement, so we omit the results for brevity. The trained LSTM model can be used to iterative predict the system dynamics as illustrated in Figure 3.1. This is a solely data-driven approach, and no explicit information regarding the form of the underlying equations is required.

The LSTM model is programmed in PYTHON (Van Rossum et al., 1995) in the TENSORFLOW (Abadi et al., 2016) library, and mapped to a single Nvidia Tesla P100 graphics processing unit (GPU).

3.2.2 *Mean Stochastic Model and Hybrid LSTM-MeSM*

The MeSM is a powerful data-driven method used to quantify uncertainty and perform forecasts in turbulent systems with high intrinsic attractor dimensionality (A. J. Majda and Harlim, 2012; Wan and Sapsis, 2017). It is parameterized a priori to capture global statistical information of the attractor by design, while its computational complexity is very low compared to LSTM or GPR. The concept behind MeSM is to model each component z^i of the state $\boldsymbol{z} = [z^1, \dots, z^{d_z}]^T$ independently with an Ornstein-Uhlenbeck (OU) process that captures the energy spectrum and the damping timescales of the statistical equilibrium. The process takes the following form

$$dz^i = c_i z^i dt + \xi_i dW_i, \tag{3.1}$$

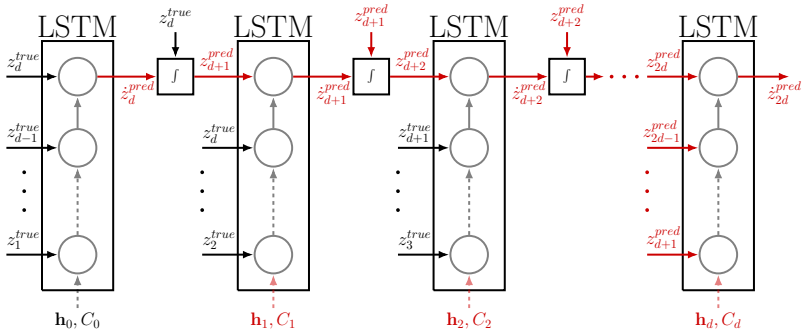


FIGURE 3.1: Iterative prediction using the trained LSTM model. A short-term history of the system, i. e. $\boldsymbol{z}_1^{true}, \ldots, \boldsymbol{z}_d^{true}$, is assumed to be known. Initial LSTM states are $\boldsymbol{h}_0, \boldsymbol{c}_0$. The trained LSTM is used predict the derivative $\dot{\boldsymbol{z}}_d^{pred} = \mathcal{F}^{\boldsymbol{w}}(\boldsymbol{z}_{\kappa_2:1}^{true}, \boldsymbol{h}_0, \boldsymbol{c}_0)$. The state prediction $\boldsymbol{z}_{\kappa_2+1}^{pred}$ is obtained by integrating this derivative. This value is used for the next prediction in an iterative fashion. After κ_2 timesteps, only predicted values are fed in the input. The short-term history is illustrated with black arrows, while predicted values with red. In stateless LSTM, $\boldsymbol{h}$ and $\boldsymbol{c}$ are initialized to zero before every prediction.

where c_i, ξ_i are parameters fitted to the centered training data and W_i is a Wiener process. In the statistical steady state the mean, energy and damping timescale of the process are given by

$$\mu_i = \mathbb{E}[z^i] = 0, \quad E_i = \mathbb{E}[z^i (z^i)^*] = -\frac{\xi^2}{2c_i}, \quad T_i = -\frac{1}{c_i}, \tag{3.2}$$

where $(z^i)^*$ denotes the complex conjugate of z^i. In order to fit the model parameters c_i, ξ_i we directly estimate the variance $\mathbb{E}[z^i (z^i)^*]$ from the time series training data and the decorrelation time using

$$T_i = \frac{1}{\mathbb{E}[z^i (z^i)^*]} \int_0^\infty \mathbb{E}\left[z^i(t)\left(z^i(t+\tau)\right)^*\right] d\tau. \tag{3.3}$$

After computing these two quantities, we replace in Equation 3.2 and solve with respect to c_i and ξ_i. Since the MeSM is modeled a priori to mimic the global statistical behavior of the attractor, forecasts made with MeSM can never escape. This is not the case with LSTM and GPR, as prediction errors accumulate and iterative forecasts escape the attractor due to the chaotic dynamics, although short-term predictions are accurate. This problem has

been addressed with respect to GPR in Wan and Sapsis, 2017. In order to cope effectively with this problem, we introduce a hybrid LSTM-MeSM technique that prevents forecasts from diverging from the attractor.

The state dependent decision rule for forecasting in LSTM-MeSM is given by

$$\dot{z}_t = \begin{cases} (\dot{z}_t)_{LSTM}, & \text{if } p^{train}(z_t) = \prod p_i^{train}(z_t^i) > \delta \\ (\dot{z}_t)_{MeSM}, & \text{otherwise} \end{cases} \tag{3.4}$$

where $p^{train}(z_t)$ is an approximation of the probability density function of the training dataset and $\delta \approx 0.01$ a constant threshold tuned based on $p^{train}(z_t)$. We approximate $p^{train}(z_t)$ using a mixture of Gaussian kernels. This hybrid architecture exploits the advantages of LSTM and MeSM. In case there is a high probability that the state z_i lies close to the training dataset (interpolation), the LSTM having memorized the local dynamics is used to perform inference. This ensures accurate LSTM short-term predictions. On the other hand, close to the boundaries, the attractor is only sparsely sampled $p^{train}(z_i) < \delta$, and errors from LSTM predictions would lead to divergence. In this case, MeSM guarantees that forecasting trajectories remain close to the attractor, and that we converge to the statistical invariant measure in the long term. The MeSM and the GPR are programmed in MATLAB (Higham et al., 2016).

3.3 BENCHMARK AND PERFORMANCE MEASURES

The performance of the proposed LSTM based prediction mechanism is benchmarked against the following state-of-the-art methods:

- Mean Stochastic Model (MeSM)
- Gaussian Process Regression (GPR)
- Hybrid Model (GPR-MeSM)

In order to guarantee that the prediction performance is independent of the initial condition selected for all applications and all performance measures considered, we report the average value of each measure for many different initial conditions sampled independently and uniformly from the attractor. The ground truth trajectory is obtained by integrating the discretized reference equation from each initial condition and projecting the states to the reduced-order space. The reference equation and the projection method are application-dependent.

From each initial condition, we generate an empirical Gaussian ensemble of dimension N_{en} around the initial condition with a small variance σ_{en}. This noise represents the uncertainty in the knowledge of the initial system state. We forecast the evolution of the ensemble by iteratively predicting the derivatives and integrating (deterministically for each ensemble member for the LSTM, stochastically for GPR), and we keep track of the mean. We select an ensemble size $N_{en} = 50$, which is the usual choice in environmental science, e. g. weather prediction and short-term climate prediction (A. Majda et al., 2005).

The ground truth trajectory at each time instant $\boldsymbol{z}$ is then compared with the predicted ensemble mean $\widetilde{\boldsymbol{z}}$. As a comparison measure we use the root mean squared error (RMSE) defined as

$$\mathrm{RMSE}(\boldsymbol{z}_k) = \sqrt{1/V \sum_{i=1}^{V} \left(z_k^i - \widetilde{z}_k^i\right)^2}, \tag{3.5}$$

where index k denotes the k^{th} component of the reduced-order state $\boldsymbol{z}$, i is the initial condition, and V is the total number of initial conditions. The RMSE is computed at each time instant for each component k of the reduced-order state, resulting in error curves that describe the evolution of error with time. Moreover, we use the standard deviation σ of the attractor samples in each dimension as a relative comparison measure. Assuming that the attractor consists of samples $\{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_N\}$, with $\boldsymbol{z}_j \in \mathbb{R}^{d_z}$, the attractor standard deviation in dimension $i \in \{1, \ldots, d_z\}$ is defined as $\mathrm{act}_i = \sqrt{\mathbb{E}[(z^i - \overline{z}^i)^2])}$, where $\overline{z}^i$ is the mean of the samples in this dimension. If the prediction error is bigger than this standard deviation, then a trivial mean predictor performs better.

Moreover, we use the mean Anomaly Correlation Coefficient (ACC) (Allgaier et al., 2012) over V initial conditions to quantify the pattern correlation of the predicted trajectories with the ground-truth. The ACC is defined as

$$\mathrm{ACC} = \frac{1}{V} \sum_{i=1}^{V} \frac{\sum_{k=1}^{d_z} w_k \left(z_k^i - \overline{z}_k\right)\left(\widetilde{z}_k^i - \overline{z}_k\right)}{\sqrt{\sum_{k=1}^{d_z} w_k \left(z_k^i - \overline{z}_k\right)^2 \sum_{k=1}^{d_z} w_k \left(\widetilde{z}_k^i - \overline{z}_k\right)^2}}, \tag{3.6}$$

where k refers to the mode number, i refers to the initial condition, w_k are mode weights selected according to the energies of the modes after dimensionality reduction and $\overline{z}_k$ is the time average of the respective mode, considered as reference. This score ranges from -1.0 to 1.0. If the forecast

is perfect, the score equals 1.0. The ACC coefficient is a widely used forecasting accuracy score in the meteorological community (Basnarkov et al., 2012).

3.4 RESULTS

In this section, the effectiveness of the proposed method is demonstrated with respect to three chaotic dynamical systems, exhibiting different levels of chaos, from weakly chaotic to fully turbulent, i. e. the Lorenz 96 model, the Kuramoto-Sivashinsky equation, and a prototypical barotropic climate model.

3.4.1 *The Lorenz 96 Model*

In this section, we consider the Lorenz 96 model, introduced in Section 2.2.2. The dimensionality of the state is set to $J = 40$.

Following A. Majda et al., 2005; Wan and Sapsis, 2017 we apply a shifting and scaling to standardize the Lorenz 96 states X_j. The discrete or Dirichlet energy is given by $E = \frac{1}{2}\sum_{j=1}^{J} X_j^2$. In order for the scaled Lorenz 96 states to have zero mean and unit energy we transform them using

$$\tilde{X}_j = \frac{X_j - \overline{X}}{\sqrt{E_p}}, \quad d\tilde{t} = \sqrt{E_p}dt, \quad E_p = \frac{1}{2T}\sum_{j=0}^{J-1}\int_{T_0}^{T_0+T}(X_j - \overline{X})^2 dt, \tag{3.7}$$

where E_p is the average energy fluctuation. In this way the scaled energy is $\tilde{E} = \frac{1}{2}\sum_{j=0}^{J-1}\tilde{X}_j^2 = 1$ and the scaled variables have zero mean $\overline{\tilde{X}} = \frac{1}{J}\sum_{j=0}^{J-1}\tilde{X}_j = 0$, with $\overline{X}$ the mean state. The scaled Lorenz 96 states $\tilde{X}_j$ obey the following differential equation

$$\frac{d\tilde{X}_j}{d\tilde{t}} = \frac{F - \overline{X}}{E_p} + \frac{(\tilde{X}_{j+1} - \tilde{X}_{j-2})\overline{X} - \tilde{X}_j}{\sqrt{E_p}} + (\tilde{X}_{j+1} - \tilde{X}_{j-2})\tilde{X}_{j-1} \tag{3.8}$$

3.4.1.1 *Dimensionality Reduction: Discrete Fourier Transform*

Firstly, the Discrete Fourier Transform (DFT) is applied to the energy standardized Lorenz 96 states $\tilde{X}_j$. The Fourier coefficients $\hat{X}_k \in \mathbb{C}$ and the inverse DFT to recover the Lorenz 96 states are given by

$$\hat{X}_k = \frac{1}{J}\sum_{j=0}^{J-1} \tilde{X}_j e^{-2\pi i k j/J}, \quad \tilde{X}_j = \sum_{k=0}^{J-1} \hat{X}_k e^{2\pi i k j/J}. \tag{3.9}$$

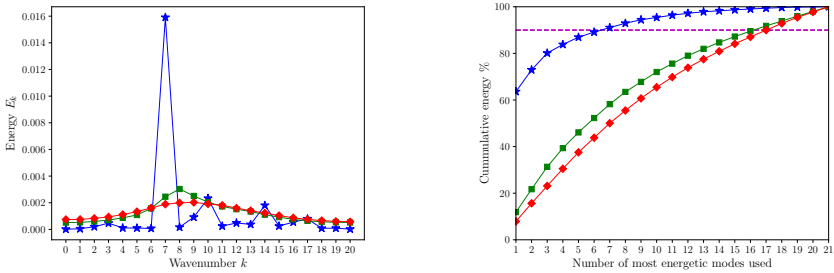


FIGURE 3.2: Energy spectrum E_k and cumulative energy with respect to the number of most energetic modes used for different forcing regimes of Lorenz 96 model. As the forcing increases, more chaoticity is introduced to the system.
$F = 4$ ★; $F = 8$ ■; $F = 16$ ◆; 90% of the total energy - - -

After applying the DFT to the Lorenz 96 states, we end up with a symmetric energy spectrum that can be uniquely characterized by $J/2+1$ (J is considered to be an even number) coefficients $\hat{X}_k$ for $k \in K = \{0, 1, \cdots, J/2\}$. In our case $J = 40$, thus we end up with $|K| = 21$ complex coefficients $\hat{X}_k \in \mathbb{C}$. These coefficients are referred to as the Fourier modes or simply modes. The Fourier energy of each mode is defined as $E_k = \sigma(\hat{X}_k)^2 = \mathbb{E}[(\hat{X}_k(\tilde{t}) - \overline{\hat{X}}_k)(\hat{X}_k(\tilde{t}) - \overline{\hat{X}}_k)^*]$.

The energy spectrum of the Lorenz 96 model is plotted in Figure 3.2 for different values of the forcing term F. The most energetic Fourier modes in the Lorenz 96 model for different forcing regimes $F \in \{4, 6, 8, 16\}$ are given in Table 3.1. These modes are used in order to construct the reduced-order phase space. We take into account only the $d_r = 6$ modes corresponding to the highest energies and the rest of the modes are truncated. For the different forcing regimes $F = 4, 8, 16$, the six most energetic modes

Forcing	Wavenumbers k	Forcing	Wavenumbers k
$F = 4$	7,10,14,9,17,16	$F = 8$	8,9,7,10,11,6
$F = 6$	8,7,9,10,11,6	$F = 16$	8,9,10,7,11,6

TABLE 3.1: Most energetic Fourier modes used in the reduced-order phase space

correspond to approximately 89%, 52% and 43.8% of the total energy, respectively. The space where the reduced variables live is referred to as the reduced-order phase space, and the most energetic modes are notated as $\hat{X}_k^r$ for $k \in \{1, \dots, d_r\}$. As shown in Crommelin et al., 2004 the most energetic modes are not necessarily the ones that capture better the dynamics of the model. Including more modes or designing a criterion to identify the most important modes in the reduced-order space may boost prediction accuracy. However, here we are not interested in an optimal reduced space representation but rather in the effectiveness of a prediction model given this space. The truncated modes are ignored for now. Nevertheless, their effect can be modeled stochastically as in Wan and Sapsis, 2017.

Since each Fourier mode $\hat{X}_k^r$ is a complex number, it consists of a real part and an imaginary part. By stacking these real and imaginary parts of the d_r truncated modes we end up with the $2\,d_r$ dimensional reduced model state

$$z \equiv [\mathrm{Re}(\hat{X}_1^r), \dots, \mathrm{Re}(\hat{X}_{d_r}^r), \mathrm{Im}(\hat{X}_1^r), \dots, \mathrm{Im}(\hat{X}_{d_r}^r)]^T \tag{3.10}$$

Assuming that X_j^t for $j \in \{0, 1, \dots, J-1\}$ are the Lorenz 96 states at time instant t, the mapping $X_j^t, \forall j \to z$ is unique and the reduced model state of the Lorenz 96 has a specific vector value.

3.4.1.2 *Training and Prediction in the Lorenz 96 Model*

The reduced Lorenz 96 model states z_t are considered as the true reference states. The LSTM is trained to forecast the derivative of the reduced-order state $\dot{z}_t$ as elaborated in Section 3.2. We use a "stateless LSTM" with $d_h = 20$ hidden units and the backpropagation truncation horizon set to $\kappa_2 = 10$.

In order to obtain training data for the LSTM, we integrate the Lorenz 96 model state in Equation 2.17 starting from an initial condition using a Runge-Kutta 4th order method with a timestep $dt = 0.01$ up to $T = 51$. In this way a time series X_j^t, $t \in \{0, 1, \cdots\}$ is constructed. We obtain the

reduced-order state time series z_t, $t \in \{0, 1, \cdots\}$, using the DFT mapping $X_t^j \forall j \rightarrow z_t$. From this time series, we discard the first 10^4 initial timesteps as initial transients, ending up with a time series with $N_{train} = 50000$ samples. A similar but independent process is repeated for the validation set.

3.4.1.3 *Results*

The trained LSTM models are used for prediction based on the iterative procedure explained in Section 3.2. In this section, we demonstrate the forecasting capabilities of LSTM and compare it with GPs. 100 different initial conditions uniformly sampled from the attractor are simulated. For each initial condition, an ensemble with size $N_{en} = 50$ is considered by perturbing it with a normal noise with variance $\sigma_{en} = 0.0001$.

In Figures 3.3(a) to 3.3(c) we report the mean RMSE prediction error of the most energetic mode $\hat{X}_1^r \in \mathbb{C}$, scaled with $\sqrt{E_p}$ for the forcing regimes $F \in \{6, 8, 16\}$ for the first $N = 10$ timesteps ($T = 0.1$). In the RMSE the complex norm $||v||_2 = vv^*$ is taken into account. The 10% of the standard deviation of the attractor is also plotted for reference ($10\%\sigma$). As F increases, the system becomes more chaotic and difficult to predict. As a consequence, the number of prediction steps that remain under the $10\%\sigma$ threshold are decreased. The LSTM models extend this predictability horizon for all forcing regimes compared to GPR and MeSM. However, when LSTM is combined with MeSM the short-term prediction performance is compromised. Nevertheless, hybrid LSTM-MeSM models outperform GPR methods in short-term prediction accuracy.

In Figures 3.3(d) to 3.3(f) the RMSE error for $T = 2$ is plotted. The standard deviation from the attractor σ is plotted for reference. We can observe that both GPR and LSTM diverge, while MeSM and blended schemes remain close to the attractor in the long term as expected.

In Figures 3.3(g) to 3.3(i), the mean ACC over 1000 initial conditions is given. The predictability threshold of 0.6 is also plotted. After crossing this critical threshold, the methods do not predict better than a trivial mean predictor. For $F = 4$ GPR methods show inferior performance compared to LSTM approaches as analyzed previously in the RMSE comparison. However, for $F = 8$ LSTM models do not predict better than the mean after $T \approx 0.35$, while GPR shows better performance. In turn, when blended with MeSM, the compromise in the performance for GPR-MeSM is much bigger compared to LSTM-MeSM. The LSTM-MeSM scheme shows slightly

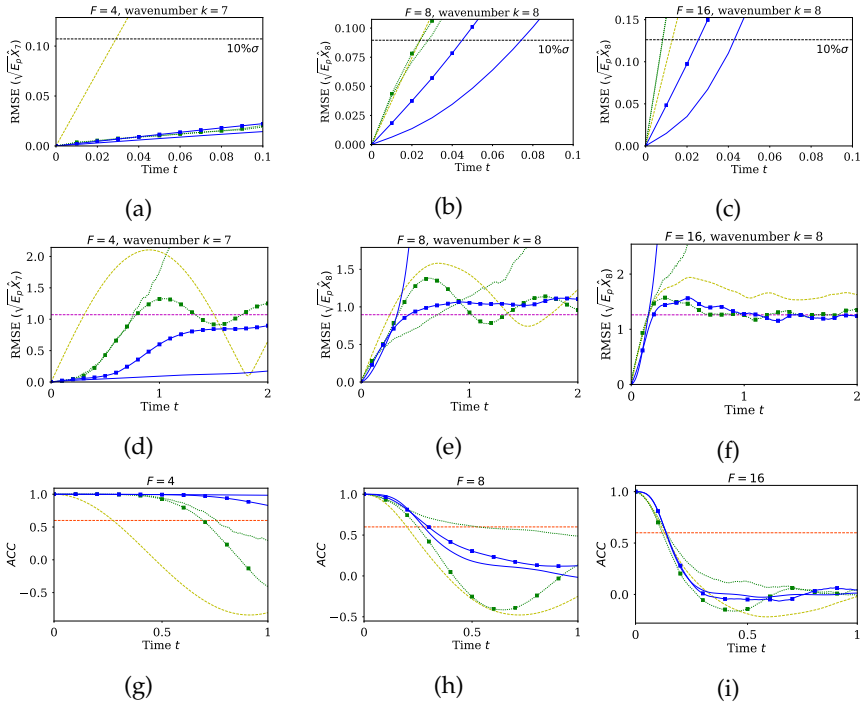


FIGURE 3.3: (a), (b), (c) Short-term RMSE evolution of the most energetic mode for forcing regimes $F = 4, 8, 16$ respectively of the Lorenz 96 model. (d), (e), (f) Long-term RMSE evolution. (g), (h), (i) Evolution of the ACC coefficient. (In all plots average over 1000 initial conditions is reported).
$10\%\,\sigma_{attractor}$ - - -; $\sigma_{attractor}$ - - -; $ACC = 0.6$ threshold - - -; MeSM - - -; GPR ····; GPR-MeSM ··■··; LSTM ——; LSTM-MeSM —■—

superior performance than GPR-MeSM during the entire relevant time period ($ACC > 0.6$). For the fully turbulent regime $F = 16$, LSTM shows comparable performance with both GPR and MeSM, and all methods converge as chaoticity rises since the intrinsic dimensionality of the system attractor increases and the system become inherently unpredictable.

In Figure 3.4, the evolution of the mean RMSE over 1000 initial conditions of the wavenumbers $k = 8, 9, 10, 11$ of the Lorenz 96 with forcing $F = 8$ is plotted. In contrast to GPR, the RMSE error of LSTM is much lower in the moderate and low energy wavenumbers $k = 9, 10, 11$ compared to the most

energetic mode $k = 8$. This difference among modes is not observed in GPR. This can be attributed to the highly nonlinear energy transfer mechanisms between these low-energy modes as opposed to the Gaussian and locally linear energy transfers of the most energetic mode.

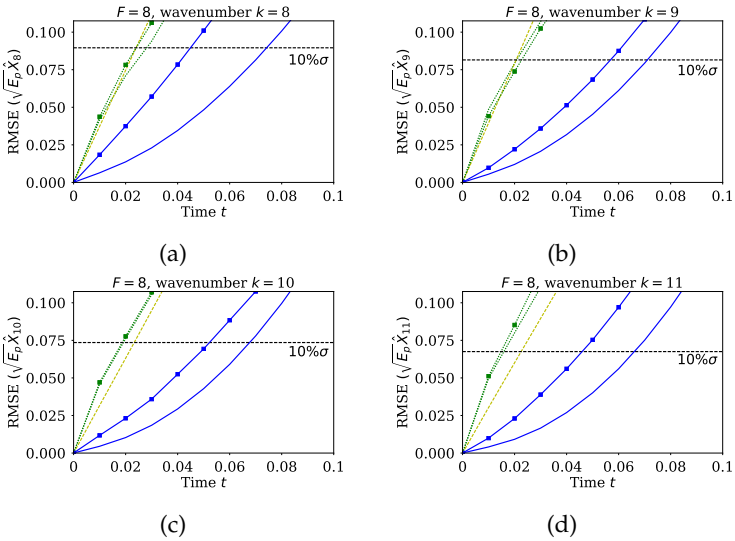
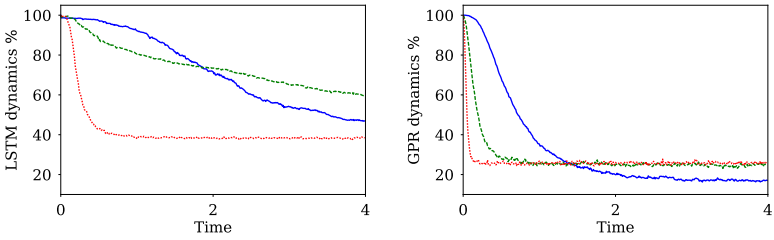


FIGURE 3.4: RMSE prediction error evolution of four energetic modes for the Lorenz 96 model with forcing $F = 8$. (a) Most energetic mode $k = 8$. (b) Low energy mode $k = 9$. (c) Low energy mode $k = 10$. (d) Low energy mode $k = 11$. (In all plots average over 1000 initial conditions reported)
$10\%\,\sigma_{attractor}$ - - -; MeSM- - -; GPR·····; GPR-MeSM··■·; LSTM——; LSTM-MeSM—■—

As illustrated before, the hybrid LSTM-MeSM architecture effectively combines the accurate short-term prediction performance of LSTM with the long-term stability of MeSM. The ratio of ensemble members modeled by LSTM in the hybrid scheme is plotted with respect to time in Figure 3.5(a). Starting from the initial ensemble of size 50, as the LSTM forecast might deviate from the attractor, the MeSM is used to forecast in the hybrid scheme. As a consequence, the ratio of ensemble members modeled by LSTM decreases with time. In parallel with the GPR results presented in Wan and Sapsis, 2017 and plotted in Figure 3.5(b), the slope of this ratio curve increases with F up to time $t \approx 1.5$. However, the LSTM ratio decreases slower compared to GPR.



(a) (b)

FIGURE 3.5: (a) Ratio of the ensemble members evaluated using the LSTM model over time for different Lorenz 96 forcing regimes in the hybrid LSTM-MeSM method and (b) the same for GPR in the hybrid GPR-MeSM method. (average over 500 initial conditions).
$F = 4$ ——; $F = 8$- - -; $F = 16$ ·····

3.4.2 *Kuramoto-Sivashinsky Equation*

In this Section, we consider the KS equations given in Section 2.2.1. We impose Dirichlet and second-type boundary conditions to guarantee ergodicity (Blonigan and Q. Wang, 2014). The boundary and initial conditions are given by

$$\begin{aligned} u(0,t) = u(L,t) = \left.\frac{\partial u}{\partial x}\right|_{x=0} = \left.\frac{\partial u}{\partial x}\right|_{x=L} = 0, \\ u(x,0) = u_0(x). \end{aligned} \tag{3.11}$$

In the discretized equations, Equation 2.16, we impose $u_0 = u_{D-1} = 0$ and add ghost nodes $u_{-1} = u_1$, $u_D = u_{D-2}$ to account for the Dirichlet and second-order boundary conditions. In our analysis, the number of nodes is $D = 513$.

The spatial variable bound is held constant to $L = 16$ and the chaoticity level is controlled through the negative viscosity ν, where a smaller value leads to a system with a higher level of chaos (see Figure Figure 2.4(a)). We consider two values, namely $\nu = 1/10$ and $\nu = 1/16$ to benchmark the prediction skills of the proposed method. The discretized equation (Equation 2.16) is integrated with a time interval $dt = 0.02$ up to $T = 11000$. The data pointss up to $T = 1000$ are discarded as initial transients. Half of the remaining data ($N = 250000$ samples) are used for training and the other half for validation.

3.4.2.1 *Dimensionality Reduction: Singular Value Decomposition*

The dimensionality of the problem is reduced using Singular Value Decomposition (SVD). By subtracting the temporal mean $\overline{\boldsymbol{U}}$ and stacking the data, we end up with the data matrix $\boldsymbol{U} \in \mathbb{R}^{N\times 513}$, where N is the number of data samples ($N = 500000$ in our case). Performing SVD on $\boldsymbol{U}$ leads to

$$\boldsymbol{U} = \boldsymbol{M}\boldsymbol{\Sigma}\boldsymbol{V}^T, \quad \boldsymbol{M} \in \mathbb{R}^{N\times N}, \quad \boldsymbol{\Sigma} \in \mathbb{R}^{N\times 513}, \quad \boldsymbol{V} \in \mathbb{R}^{513\times 513}, \tag{3.12}$$

with Σ diagonal, with descending diagonal elements. The right singular vectors corresponding to the d_r largest singular values are the first columns of $\boldsymbol{V} = [\mathbf{V_r}, \mathbf{V_{-r}}]$. Stacking these singular vectors yields $\mathbf{V_r} \in \mathbb{R}^{513\times d_r}$. Assuming that $\boldsymbol{U}_t \in \mathbb{R}^{513}$ is a vector of the discretized values of $u(x,t)$ in time t, in order to get a reduced-order representation $\mathbf{c} \equiv [c_1, \ldots, c_{d_r}]^T$ corresponding to the d_r components with the highest energies (singular values) we multiply

$$\mathbf{c} = \mathbf{V_r}^T\boldsymbol{U}, \quad \mathbf{c} \in \mathbb{R}^{d_r}. \tag{3.13}$$

The percentage of cumulative energy w. r. t. to the number of PCA modes considered is plotted in Figure 2.4(b). Here, we pick $d_r = 20$ (out of 513) most energetic modes, as they explain approximately 90% of the total energy.

The temporal average of the state of the Kuramoto-Sivashinsky equation and the cumulative energy are plotted in Figure 3.6. As ν declines, chaoticity in the system rises, and higher oscillations of the mean towards the Dirichlet boundary conditions are observed in Figure 3.6, while the number of modes needed to capture most of the energy is higher.

3.4.2.2 *Results*

We train "stateless" LSTM models with $d_h = 100$ and $\kappa_2 = 50$. For testing, starting from 1000 initial conditions uniformly sampled from the attractor, we generate a Gaussian ensemble of dimension $N_{en} = 50$ centered around the initial condition in the original space with standard deviation of $\sigma = 0.1$. This ensemble is propagated using the LSTM prediction models and GPR, MeSM, and GPR-MeSM models trained as in Wan and Sapsis, 2017. The root mean squared error between the predicted ensemble mean and the ground-truth is plotted in Figures 3.8(a) and 3.8(b) for different values of the parameter ν. All methods reach the invariant measure much faster for $1/\nu = 16$ compared to the less chaotic regime $1/\nu = 10$ (note the different integration times $T = 4$ for $1/\nu = 10$, while $T = 1.5$ for $1/\nu = 16$).

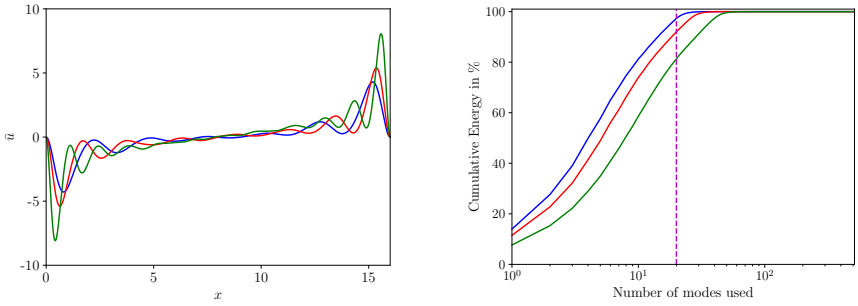


FIGURE 3.6: Temporal average $\overline{\boldsymbol{U}}$ and cumulative mode (PCA) energy for different values of ν in the Kuramoto-Sivashinsky equation.
$1/\nu = 10$ ——; $1/\nu = 16$ ——; $1/\nu = 36$ ——; 20 modes - - -

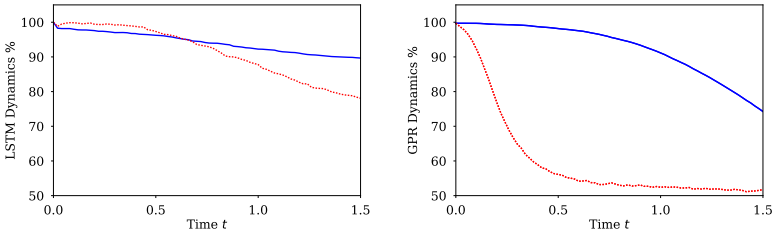
In both chaotic regimes $1/\nu = 10$ and $1/\nu = 16$, the reduced-order LSTM outperforms all other methods in the short-term before escaping the attractor. However, in the long-term, LSTM does not stabilize and will eventually diverge faster than GPR (see Figure 3.8(b)). Blending LSTM with MeSM alleviates the problem, and both accurate short-term predictions and long-term stability are attained. Moreover, the hybrid LSTM-MeSM has better forecasting capabilities compared to GPR.

The need for blending LSTM with MeSM in the KS equation is less imperative as the system is less chaotic than the Lorenz 96, and LSTM methods diverge much slower while they sufficiently capture the complex nonlinear dynamics. As the intrinsic dimensionality of the attractor rises, LSTM diverges faster.

The mean ACC (Equation 3.6) is plotted with respect to time in Figures 3.8(c) and 3.8(d) for $\nu = 10$ and 16 respectively. The evolution of the ACC justifies the aforementioned analysis. The mean ACC of the trajectory predicted with LSTM remains above the predictability threshold of 0.6 for the highest time duration compared to other methods. This predictability horizon is approximately 2.5 for $\nu = 1/10$ and 0.6 for $\nu = 1/16$, since the chaoticity of the system rises and accurate predictions become more challenging.

For the hybrid LSTM-MeSM, the ratio of the ensemble members that are modeled with LSTM is plotted with respect to time in Figure 3.7(a). The quotient drops slower for $1/\nu = 10$ in the long run as the intrinsic dimensionality of the attractor is smaller, and trajectories diverge slower.

However, in the beginning, the LSTM ratio is higher for $1/\nu = 16$ as the MeSM drives initial conditions close to the boundary faster towards the attractor due to the higher damping coefficients compared to the case $1/\nu = 10$. This explains the initial knick in the graph for $1/\nu = 16$. The slow damping coefficients for $1/\nu = 10$ do not allow the MeSM to drive the trajectories back to the attractor at a faster pace than the diffusion caused by the LSTM forecasting. Compared with GPR plotted in Figure 3.7(b), the ratio drops slower.



(a) (b)

FIGURE 3.7: (a) Ratio of LSTM-MeSM ensemble members modeled by the LSTM dynamics for the Kuramoto-Sivashinsky ($T = 1.5$). (b) The same for GPR in the hybrid GPR-MeSM. (Mean over 1000 initial conditions) $1/\nu = 10$ ——; $1/\nu = 16$ ·····

3.4.3 *A Barotropic Climate Model*

In this section, we examine a standard barotropic climate model (Selten, 1995) originating from a realistic winter circulation. The model equations are given by

$$\frac{\partial \zeta}{\partial t} = -\mathcal{J}(\psi, \zeta + f + h) + k_1 \zeta + k_2 \delta^3 \zeta + \zeta^*, \tag{3.14}$$

where ψ is the stream function, $\zeta = \delta\psi$ the relative vorticity, f the Coriolis parameter, ζ^* a constant vorticity forcing, while k_1 and k_2 are the Ekman damping and the scale-selective damping coefficient. $\mathcal{J}$ is the Jacobi operator given by

$$\mathcal{J}(a, b) = \left(\frac{\partial a}{\partial \lambda} \frac{\partial B}{\partial \widetilde{\mu}} - \frac{\partial a}{\partial \widetilde{\mu}} \frac{\partial B}{\partial \lambda} \right), \tag{3.15}$$

where $\widetilde{\mu}$ and λ denote the sine of the geographical latitude and longitude respectively. The equation of the barotropic model, i. e. Equation 3.14, is

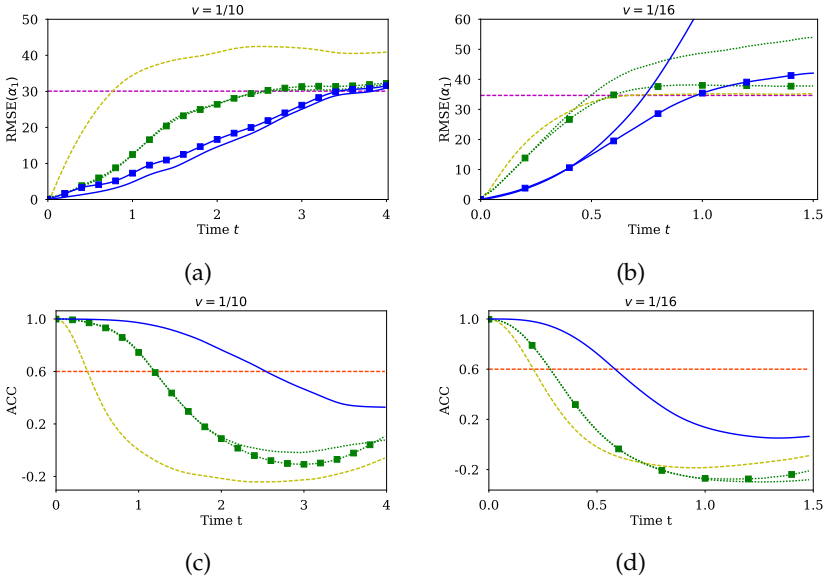
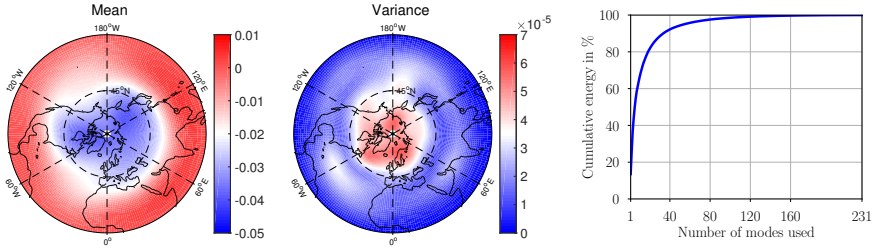


FIGURE 3.8: (a), (b) RMSE evolution of the most energetic mode of the KS equation with $1/\nu = 10$ and $1/\nu = 16$. (c), (d) ACC evolution of the most energetic mode of the KS equation with $1/\nu = 10$ and $1/\nu = 16$. (In all plots, average value over 1000 initial conditions is reported) $\sigma_{attractor}$; $ACC = 0.6$ threshold; MeSM; GPR; GPR-MeSM; LSTM; LSTM-MeSM

non-dimensionalized using the radius of the earth as unit length and the inverse of the earth angular velocity as time unit. The non-dimensional orography h is related to the real Northern Hemisphere orography $h^{'}$ by $h = 2sin(\phi_0)A_0h^{'}/H$, where ϕ_0 is a fixed amplitude of $45°N$, A_0 is a factor expressing the surface wind strength blowing across the orography, and H a scale height (Selten, 1995). The stream-function ψ is expanded into a spherical harmonics series and truncated at wavenumber 21, while modes with an even total wavenumber are excluded, avoiding currents across the equator and ending up with a hemispheric model with 231 degrees of freedom.

The training data are obtained by integrating Equation 3.14 for 10^5 days after an initial spin-up period of 1000 days, using a fourth-order Adams-Bashforth integration scheme with a 45-min timestep in accordance with Wan and Sapsis, 2017, with $k_1 = 15$ days, while k_2 is selected such that

wavenumber 21 is damped at a timescale of 3 days. In this way, we end up with a time series ζ_t with 10^4 samples. The spherical surface is discretized into a $D = 64 \times 32$ mesh with equally spaced latitude and longitude. From the gathered data, 90% is used for training and 10% for validation. The mean and variance of the statistical steady state are shown in Figures 3.9(a) and 3.9(b).



(a) (b) (c)

FIGURE 3.9: (a) Mean of the Barotropic model at statistical steady state. (b) Variance of the Barotropic model at statistical steady state. (c) Percentage of energy explained with respect to the modeled modes.

The dimensionality of the barotropic climate model truncated at wavenumber 21 is 231. In order to reduce it, we identify Empirical Orthogonal Functions (EOFs) ϕ_i, $i \in \{1, \ldots, 231\}$ that form an orthogonal basis of the reduced-order space. The details of the method are described in Appendix A.2. EOF analysis has been used to identify individual realistic climatic modes such as the Arctic Oscillation (AO), the Pacific/North America (PNA), and the Tropical/Northern Hemisphere (TNH) pattern known as teleconnections (Mo et al., 1986; Thompson et al., 2000). Accurate prediction of these modes is of high practical importance as they feature realistic climate patterns. After projecting the state of the barotropic model to the EOFs, we take into account only the d_r coefficients corresponding to the most energetic EOFs that form the reduced-order state $\boldsymbol{y}^*$. Here, we set the dimensionality of the reduced space to $d_r = 30$, as ϕ_{30} contains only 3.65% of the energy of ϕ_1, while the 30 most energetic modes contain approximately 82% of the total energy, as depicted in Figure 3.9(c).

3.4.3.1 *Training and Prediction*

The reduced-order state that we want to predict using the LSTM are the 30 components of $\boldsymbol{y}$. A "stateless" LSTM with $d_h = 140$ hidden units is

considered, while the truncated backpropagation horizon is set to $\kappa_2 = 10$. The prototypical system is less chaotic than the KS equation and the Lorenz 96, which enables us to use more hidden units. The reason is that as chaoticity is decreased, trajectories sampled from the attractor as training and validation dataset become more interconnected, and the task is inherently easier and less prone to overfitting. In the extreme case of a periodic system, the information would be identical. 500 points are randomly and uniformly picked from the attractor as initial conditions for testing. A Gaussian ensemble with small variance ($\sigma_{en} = 0.001$) along each dimension is formed and marched using the reduced-order GPR, MeSM, Mixed GPR-MeSM, and LSTM methods.

3.4.3.2 *Results*

The RMSE error of the four most energetic reduced-order space variables $\boldsymbol{y}_i$ for $i \in \{1, \ldots, 4\}$ is plotted in Figure 3.10. The LSTM takes $400 - 500$ hours to reach the attractor, while GPR based methods generally take $300 - 400$ hours. In contrast, the MeSM reaches the attractor already after 1 hour. This implies that the LSTM can better capture the nonlinear dynamics than GPR. The barotropic model is much less chaotic than the Lorenz 96 model with $F = 16$, where all methods show comparable prediction performance. Blended LSTM models with MeSM are omitted here, as LSTM models only reach the attractor standard deviation towards the end of the simulated time, and MeSM-LSTM shows identical performance.

3.5 SENSITIVITY TO NOISE IN THE DATA

In this section, we evaluate the robustness of the proposed approach to noise. For this purpose, the training data are perturbed with different noise levels. We add Gaussian noise sampled from $N(0, \sigma_{noise})$ where the noise standard deviation is proportional to the attractor standard deviation $\sigma_{attractor}$ of each system, i. e. $\sigma_{noise} = k\,\sigma_{attractor}$. We note that $\sigma_{attractor}$ is computed from the training data as the standard deviation of the samples of the reduced-order state of the system. Different noise levels k are considered.

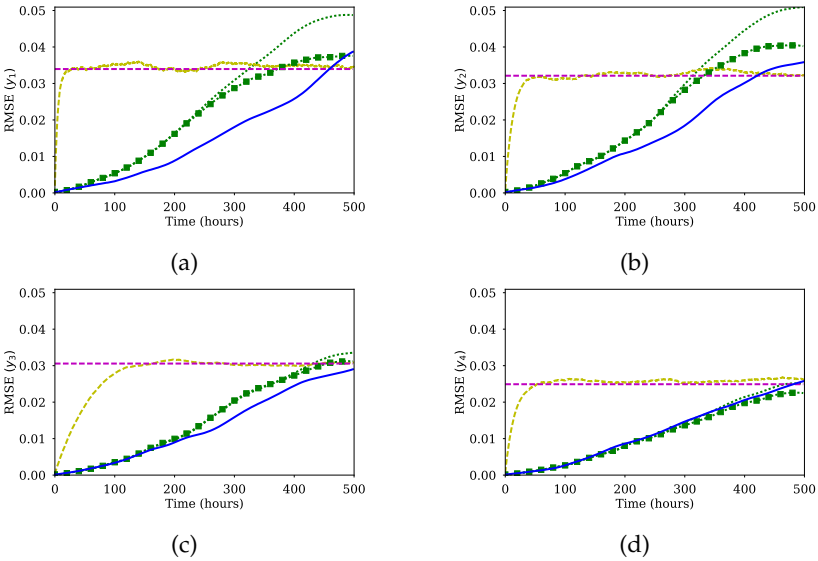


FIGURE 3.10: RMSE evolution of the four most energetic EOFs for the Barotropic climate model, average over 500 initial conditions reported. (a) Most energetic EOF. (b) Second most energetic EOF. (c) Third most energetic EOF. (d) Fourth most energetic EOF.
$\sigma_{attractor}$ - - -; MeSM - - -; GPR ·····; GPR-MeSM ··■·; LSTM ——

3.5.1 *Lorenz 96 Model*

In the following, we analyze the influence of noise in the training data for the Lorenz 96 model. In parallel with the main body of the paper, we plot the RMSE error evolution of the most energetic mode (first row of Figure 3.11) for short-term till $T = 0.1$, the same for time $T = 2$ (second row of Figure 3.11) and the ACC (third row of Figure 3.11). The columns of Figure 3.11 correspond to different chaotic regimes of the Lorenz 96 model. For the forcing $F = 4$ and noise levels $k \in \{0.01, 0.2\}$, noise does not affect the prediction performance of the LSTM. This can be attributed to the fact that the attractor dimensionality is really low in this case, and the amount of data is enough to capture the dynamics despite the noisy training data. However, for $F = 8$ and $F = 16$, adding noise leads to slight deterioration of the short-term prediction accuracy for the noise level $k = 0.01$, as illustrated by the last two figures in the first row of Figure 3.11. As a consequence, the

method can be considered robust against noise. Increasing the noise level to $k = 0.2$ corresponding to a noise standard deviation equal to 20% of the attractor standard deviation leads to deterioration in short-term prediction performance. The deterioration in the short-term prediction performance can be seen in the short-term RMSE error evolution of the fourth most energetic modes for the forcing regime $F = 8$ plotted in Figure 3.12.

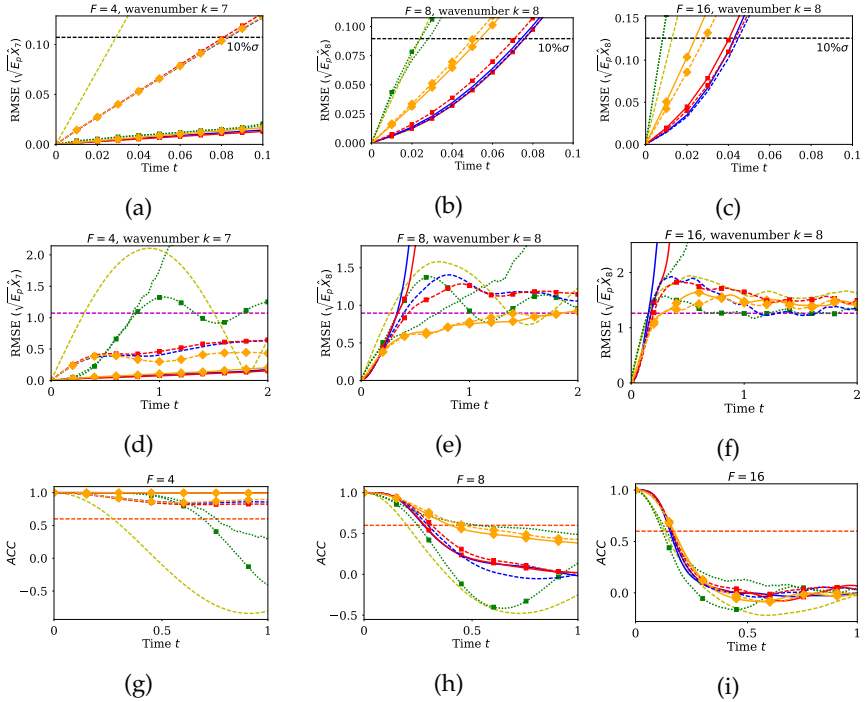


FIGURE 3.11: (a), (b), (c) Short-term RMSE evolution of the most energetic mode for forcing regimes $F = 4, 8, 16$ respectively of the Lorenz 96 model. (d), (e), (f) Long-term RMSE evolution. (g), (h), (i) Evolution of the ACC coefficient. (In all plots average over 1000 initial conditions is reported).
$10\%\,\sigma_{attractor}$ - - -; $\sigma_{attractor}$ - - -; $ACC = 0.6$ threshold - - -; MeSM - - -; GPR ····; GPR-MeSM ··■·; LSTM $k = 0‰$ ——; LSTM-MeSM $k = 0‰$ - - -; LSTM $k = 10‰$ —■—; LSTM-MeSM $k = 10‰$ - ■ -; LSTM $k = 200‰$ —◆—; LSTM-MeSM $k = 200‰$ - ◆ -

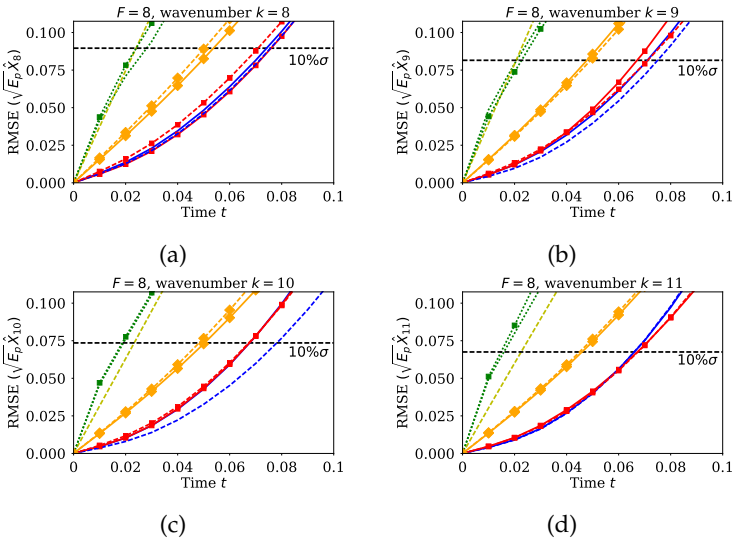


FIGURE 3.12: RMSE prediction error evolution of four energetic modes for the Lorenz 96 model with forcing $F = 8$. (a) Most energetic mode $k = 8$. (b) Low energy mode $k = 9$. (c) Low energy mode $k = 10$. (d) Low energy mode $k = 11$. (In all plots average over 1000 initial conditions reported)
$10\%\,\sigma_{attractor}$ - - -; MeSM- - -; GPR·····; GPR-MeSM·· ■ ·; LSTM $k = 0‰$ ——; LSTM-MeSM $k = 0‰$ - - -; LSTM $k = 10‰$ —■—; LSTM-MeSM $k = 10‰$ - ■ -; LSTM $k = 200‰$ —◆—; LSTM-MeSM $k = 200‰$ - ◆ -

3.5.2 *Kuramoto-Sivashinsky Equation*

In Figure 3.13 we plot the RMSE error evolution for the most energetic mode and the ACC of the Kuramoto-Sivashinsky equation for two different chaotic regimes $1/\nu \in \{10, 16\}$. Three different noise levels $k \in \{0.001, 0.01, 0.2\}$ are considered. For the low chaotic regime $1/\nu = 10$, predictability performance is robust against noise, as the error evolution changes slightly with $k \in \{0.001, 0.01\}$. The predictability performance deteriorates significantly only when the training data are polluted with noise with a standard deviation bigger than 20% of the attractor standard deviation. On the contrary, adding noise to the training data in the input improves the predictability performance of LSTM for the chaotic regime $\nu = 1/16$. This can be attributed to the fact that in this chaotic regime,

correlation patterns are much less prominent, and the LSTM is more prone to overfit. As a consequence, adding noise to the input forces the neural network to learn only robust patterns in the data that can be generalized. Short-term prediction performance deteriorates slightly, but in the long term, the LSTM is more robust against the accumulation of errors. This behavior has to be further investigated in future work.

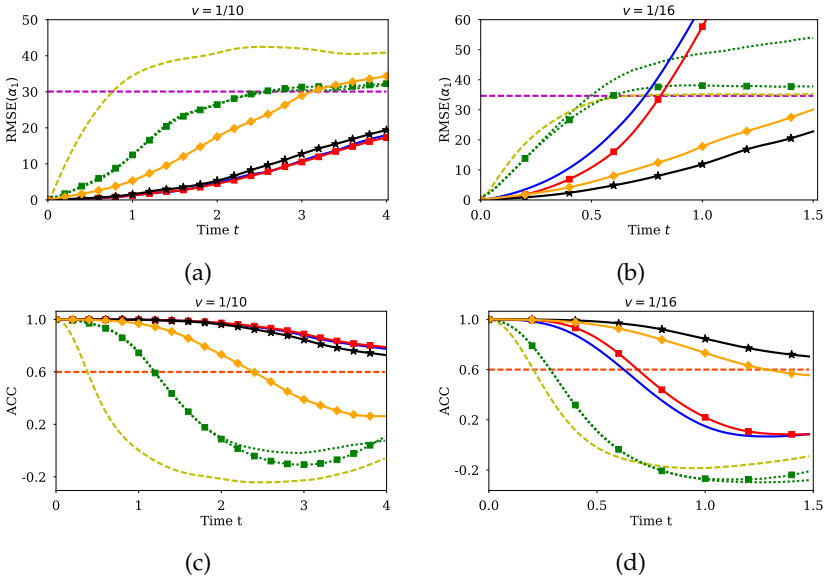


FIGURE 3.13: Training data of LSTM are perturbed with standard deviation $\sigma_{noise} = k\,\sigma_{attractor}$. Three different noise levels $k \in \{0.001, 0.01, 0.2\}$ are considered. (a), (b) RMSE evolution of the most energetic mode of the KS equation with $1/\nu = 10$ and $1/\nu = 16$. (c), (d) ACC evolution of the most energetic mode of the KS equation with $1/\nu = 10$ and $1/\nu = 16$. (In all plots, average value over 1000 initial conditions is reported)
$\sigma_{attractor}$ - - -; $ACC = 0.6$ threshold - - -; MeSM - - -; GPR ·····; GPR-MeSM ··■··; LSTM $k = 0‰$ ——; LSTM $k = 1‰$ —■—; LSTM $k = 10‰$ —★—; LSTM $k = 200‰$ —◆—

3.5.3 *Barotropic Model*

In Figure 3.14 we plot the RMSE error evolution for the four most energetic EOFs of the Barotropic model. Three different noise levels $k \in \{0.001, 0.01, 0.2\}$ are considered. Only for the highest noise level is the prediction performance deteriorated. For low noise levels, the prediction performance can be increased ($k = 0.001$), as the noise may regularize the Backpropagation procedure during training with stochastic methods. Adding noise to the input of neural networks can be used as a practical heuristic to increase their accuracy and can also be seen as a form of dropout in the input layer of the LSTM. The results indicate that the prediction performance of the LSTM is robust for the noise levels $k \in \{0.001, 0.01\}$.

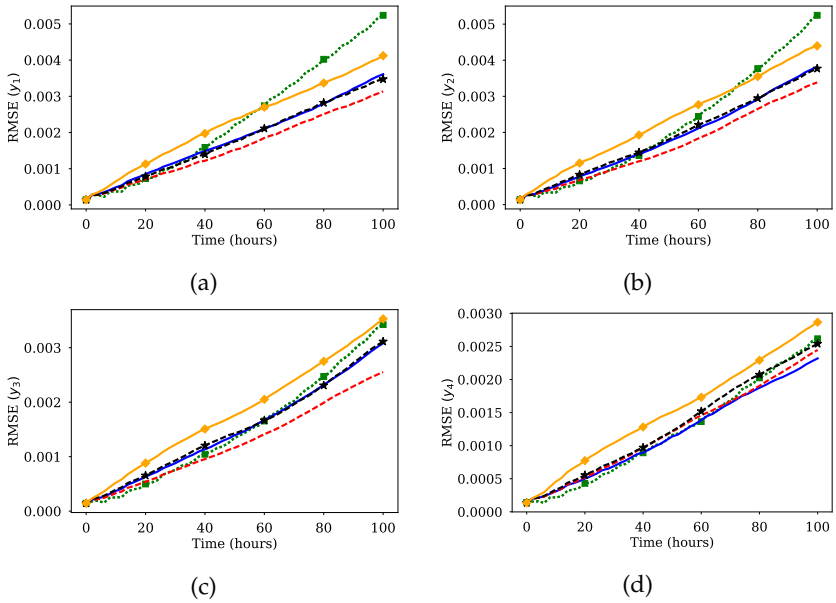


FIGURE 3.14: RMSE evolution of the four most energetic EOFs for the Barotropic climate model, average over 500 initial conditions reported. Training data are perturbed with Gaussian noise with standard deviation $\sigma_{noise} = k\,\sigma_{attractor}$. LSTM results for different noise levels k are plotted. (a) Most energetic EOF. (b) Second most energetic EOF. (c) Third most energetic EOF. (d) Fourth most energetic EOF.
GPR ····; GPR-MeSM ··■··; LSTM $k = 0‰$ ——; LSTM $k = 1‰$ - - -; LSTM $k = 10‰$ -·★·-; LSTM $k = 200‰$ —◆—;

3.6 COMPUTATIONAL COST OF PREDICTION

The computational cost of making a single prediction can be quantified by the number of operations (multiplications and additions) needed. In GPR based approaches, the computational cost is of order $O(N^2)$, where N is the number of samples used in training. For GPR methods illustrated in the previous section $N \approx 2500$. The GPR models the global dynamics by uniformly sampling the attractor and "carries" this training dataset at each time instant to identify the geometric relation between the input and the training dataset (modeled with a covariance matrix metric) and make (exact or approximate) probabilistic inference on the output.

In contrast, LSTM adjusts its parameters to reproduce the local dynamics. As a consequence, the computational complexity of inference does not depend on the number of samples used for training. The inference complexity is roughly $O(d_z \cdot \kappa_2 \cdot d_h + \kappa_2 \cdot d_h^2)$, where d_z is the dimension of each input (reduced-order state), κ_2 is the number of inputs (timesteps) and d_h is the number of hidden units. This complexity is significantly smaller than GPR, which can be translated to faster prediction. However, it is logical that the LSTM is more prone to diverge from the attractor, as there is no guarantee that the infrequent training samples near the attractor limits were memorized. This remark explains the faster divergence of LSTM in the more turbulent regimes considered in Section 3.4.

3.7 DISCUSSION

In this chapter, we proposed a data-driven method based on long short-term memory networks for modeling and prediction in the reduced space of chaotic dynamical systems. The LSTM uses the short-term history of the reduced-order variable to predict the state derivative and uses it for one-step prediction. The network is trained on time series data, and it requires no prior knowledge of the underlying governing equations. Long-term predictions are made by iteratively predicting one step forward using the trained network.

The features of the proposed technique are showcased through comparisons with GPR and MeSM on benchmarked cases. Three applications are considered, the Lorenz 96 model, the Kuramoto-Sivashinsky equation, and a barotropic climate model. The chaoticity of these systems ranges from weakly chaotic to fully turbulent, ensuring a complete simulation study.

Comparison measures include the RMSE and ACC between the predicted trajectories and trajectories of the real dynamics.

The proposed approach performs better in short-term predictions in all cases, as the LSTM is more efficient in capturing the local dynamics and complex interactions between the modes. However, the prediction error accumulates as we iteratively perform predictions and, similar to GPR, does not converge to the invariant measure. Furthermore, in the cases of increased chaoticity, the LSTM diverges faster than GPR. This may be attributed to the absence of certain attractor regions in the training data, insufficient training, and the exponentially increasing prediction error during propagation. To mitigate this effect, LSTM is combined with MeSM, following ideas presented in Wan and Sapsis, 2017, in order to guarantee convergence of the error to the invariant measure. Blending LSTM or GPR with MeSM deteriorates the short-term prediction performance, but the steady-state statistical behavior is captured. The hybrid LSTM-MeSM exhibits a slightly superior performance than GPR-MeSM in all systems considered in this study.

In the Kuramoto-Sivashinsky equation, LSTM can better capture local dynamics than Lorenz 96 due to the lower intrinsic attractor dimensionality. LSTM is more accurate than GPR in the short-term, but especially in the chaotic regime, $1/\nu = 16$ forecasts of LSTM fly away from the attractor faster. LSTM-MeSM counters this effect, and long-term forecasts converge to the invariant measure at the expense of a compromise in the short-term forecasting accuracy. The higher short-term forecasting accuracy of LSTM can be attributed to the fact that it is a nonlinear approximator and can also capture correlations between modes in the reduced space. In contrast, GPR is a locally linear approximator modeling each mode independently, assuming Gaussian correlations between modes in the input. LSTM and GPR show comparable forecasting accuracy in the barotropic model, as the intrinsic dimensionality is significantly smaller than Kuramoto-Sivashinsky and Lorenz 96, and both methods can effectively capture the dynamics.

Possible future directions for research include modeling the low-energy modes and interpolation errors using a stochastic component in the LSTM to improve forecasting accuracy. Another possible research direction is to model the attractor in the reduced space using a mixture of LSTM models, one model for each region. The LSTM proposed in this chapter models the attractor globally. However, different attractor regions may exhibit different dynamic behaviors, which cannot be modeled using only one network. Moreover, these local models can be combined with a closure scheme

compensating for truncation and modeling errors. This local modeling approach may further improve prediction performance.

4

TRAINING ALGORITHMS AND SCALABILITY OF RECURRENT NEURAL NETWORKS FOR FORECASTING DYNAMICAL SYSTEMS

4.1 RELATED WORK

As demonstrated in Chapter 3, RNNs can be employed successfully for forecasting chaotic dynamical systems at their reduced-order space. Their training, however, is difficult due to the problem of vanishing gradients (Hochreiter, 1998). Gated architectures like LSTMs alleviate the problem. These architectures are trained with BPTT, which can be slow in practice due to the iterative procedure of updating the neural network's weights. A faster alternative that aims to surpass this problem is the RC paradigm.

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 et al., 2019; Grigoryeva et al., 2018). In Pathak, Lu, 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 are fed at the network's input. The model takes advantage of the local interactions in the state-space to decouple the forecasting of each mode group and improve the scalability.

Despite the rich literature on both methods, namely RNNs trained with BPTT and RC, there are limited comparative studies of the two frameworks. This chapter examines these two prominent machine learning algorithms on challenging physical problems. We note that our RC implementation also uses a recurrent neural network, but it does not train the internal network parameters according to the RC paradigm. We consider the cases of fully observed systems and 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. We also examine the modeling capabilities of the two approaches for reproducing correct Lyapunov exponent and frequency spectra. Moreover, we include some more recent RNN architectures, like Unitary (Arjovsky et al., 2016; Jing et al., 2017) and Gated Recurrent Units (GRUs) (Cho et al., 2014; Chung, Gulcehre, et al., 2014) that have shown superior performance over LSTMs for a wide variety of language, speech signal, and polyphonic music modeling tasks. We demonstrate that RNNs have the potential to overcome scalability problems and be applied to high-dimensional spatio-temporal dynamics.

We are interested in the 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 RC. Finally, we discuss the advantages implementation aspects, such as Random Access Memory (RAM) requirements and training time, and limitations of each model. We remark that the comparison in terms of time and RAM memory consumption does not aim to quantify advantages/drawbacks among models but instead provides information for the end-users of the software.

The structure of the chapter is as follows. Section 4.2 provides an introduction to the tasks and an outline of the architectures and training methods used in this chapter. Section 4.3 introduces the measures used to compare the efficiency of the models. In Section 4.4 the networks are compared in forecasting reduced-order dynamics in the Lorenz 96 model. In Section 4.5, a parallel architecture leveraging local interactions in the state-space is introduced and utilized to forecast the dynamics of the Lorenz 96 model (E. Lorenz, 1995) and the Kuramoto-Sivashinsky equation (Kuramoto, 1978). In Section 4.6 the GRU and RC networks are utilized to reproduce the Lyapunov spectrum of the Kuramoto-Sivashinsky equation. The chapter concludes with Section 4.7.

This chapter is based on the paper "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatio-temporal dynamics" (P. R. Vlachas, Pathak, et al., 2020). The computational resources were provided by a grant from the Swiss National Supercomputing Centre (CSCS) under project s929.

4.2 METHODS

We consider RNNs for time series forecasting. The models are trained on time series of an observable $\boldsymbol{o} \in \mathbb{R}^{d_o}$ sampled at a fixed rate $1/\Delta t$, $\{\boldsymbol{o}_1, \ldots, \boldsymbol{o}_T\}$, where we eliminate Δt from the notation for simplicity. The models possess an internal high-dimensional hidden state denoted by $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ that enables the encoding of temporal dependencies on past state history. Given the current observable $\boldsymbol{o}_t$, the output of each model is a forecast $\widetilde{\boldsymbol{o}}_{t+1}$ for the observable at the next time instant $\boldsymbol{o}_{t+1}$. This forecast is a function of the hidden state. The general functional form of the RNN models, given in Equation 2.2 and repeated here, is given by

$$\boldsymbol{h}_t = \mathcal{F}_{hh}(\boldsymbol{o}_t, \boldsymbol{h}_{t-1}), \quad \widetilde{\boldsymbol{o}}_{t+1} = \mathcal{F}_{ho}(\boldsymbol{h}_t), \tag{4.1}$$

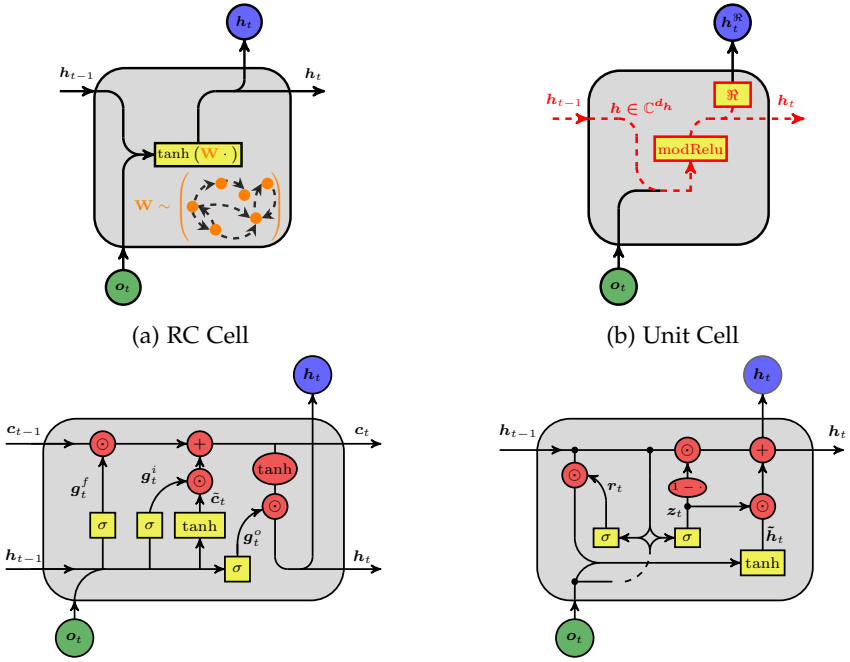
where $\mathcal{F}_{hh}$ is the hidden-to-hidden mapping and $\mathcal{F}_{ho}$ is the hidden-to-output mapping. All recurrent models analyzed in this chapter share this common architecture. They differ in the realizations of $\mathcal{F}_{hh}$ and $\mathcal{F}_{ho}$ and in the way the parameters or weights of these functions are learned from data, i.e., trained, to forecast the dynamics.

In the following, we describe the RNN cells considered in this chapter. The LSTM was introduced in Section 2.1.4.2. All RNN models are implemented in PYTHON (Van Rossum et al., 1995) in the PYTORCH (Paszke et al., 2019) library, mapped to a single Nvidia Tesla P100 GPU, and executed on the XC50 compute nodes of the Piz Daint supercomputer at the Swiss national supercomputing centre (CSCS).

4.2.1 *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, Gulcehre, et al., 2014). In the GRU cell, the functional form of the mapping $\boldsymbol{h}_t = \mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}(\boldsymbol{o}_t, \boldsymbol{h}_{t-1})$ is given by

$$\begin{aligned}
\boldsymbol{u}_t &= \mathrm{act}_g\left(\boldsymbol{W}_u[\boldsymbol{h}_{t-1}, \boldsymbol{o}_t] + \boldsymbol{b}_u\right) \\
\boldsymbol{r}_t &= \mathrm{act}_g\left(\boldsymbol{W}_r[\boldsymbol{h}_{t-1}, \boldsymbol{o}_t] + \boldsymbol{b}_r\right) \\
\widetilde{\boldsymbol{h}}_t &= \tanh\left(\boldsymbol{W}_h\left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{o}_t\right] + \boldsymbol{b}_h\right) \\
\boldsymbol{h}_t &= (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{u}_t \odot \widetilde{\boldsymbol{h}}_t,
\end{aligned} \tag{4.2}$$



(a) RC Cell (b) Unit Cell

(c) LSTM Cell (d) GRU Cell

FIGURE 4.1: The information flow for a Reservoir Computing (RC) cell, a complex Unitary cell (Unit), a Long Short-Term Memory (LSTM) cell and a Gated Recurrent Unit (GRU) cell. The cells were conceptualized to tackle the vanishing gradients problem of Elman RNNs. The cell used in RC is the standard architecture of the Elman RNN. However, the weights of the recurrent connections are randomly picked to satisfy the echo state property and create a large reservoir of rich dynamics. Only the output weights are trained (e. g. , with ridge regression). The Unitary RNN utilizes a complex unitary matrix to ensure that the gradients are not vanishing. LSTM and GRU cells employ gating mechanisms that allow forgetting and storing information in processing the hidden state. Ellipses and circles denote entry-wise operations, while rectangles denote layer operations. The information flow of the complex hidden state in the Unitary RNN is illustrated with dashed red color, while the untrained randomly picked weights of the RC with orange.

where $\boldsymbol{o}_t \in \mathbb{R}^{d_o}$ is the observable state provided at the input of the RNN at time t, $\boldsymbol{u}_t \in \mathbb{R}^{d_h}$ is the update gate vector, $\boldsymbol{r}_t \in \mathbb{R}^{d_h}$ is the reset gate vector, $\widetilde{\boldsymbol{h}}_t \in \mathbb{R}^{d_h}$, $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is the internal hidden memory state, $\boldsymbol{W}_u$, $\boldsymbol{W}_r$, $\boldsymbol{W}_h$ $\in \mathbb{R}^{d_h \times (d_h + d_o)}$ are weight matrices and $\boldsymbol{b}_u, \boldsymbol{b}_r, \boldsymbol{b}_h \in \mathbb{R}^{d_h}$ biases. The gating activation act_g is a sigmoid. The output $\widetilde{\boldsymbol{o}}_{t+1}$ is given by the linear layer:

$$\widetilde{\boldsymbol{o}}_{t+1} = \boldsymbol{W}_{h,o}\, \boldsymbol{h}_t, \tag{4.3}$$

where $\boldsymbol{W}_{h,o} \in \mathbb{R}^{d_o \times d_h}$. An illustration of the information flow in a GRU cell is given in Figure 4.1(d).

4.2.2 *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

$$\begin{aligned} \boldsymbol{h}_t &= \text{modReLU}\left(\boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{W}_o \boldsymbol{o}_t\right) \\ \widetilde{\boldsymbol{o}}_{t+1} &= \boldsymbol{W}_o\ \text{Re}\left(\boldsymbol{h}_t\right), \end{aligned} \tag{4.4}$$

where $\boldsymbol{W}_h \in \mathbb{C}^{d_h \times d_h}$ is the complex unitary recurrent weight matrix, $\boldsymbol{W}_o \in \mathbb{C}^{d_h \times d_o}$ is the complex input weight matrix, $\boldsymbol{h}_t \in \mathbb{C}^{d_h}$ is the complex state vector, $\text{Re}(\cdot)$ denotes the real part of a complex number, $\boldsymbol{W}_o \in \mathbb{R}^{d_h \times d_h}$ is the real output matrix, and the modified ReLU nonlinearity modReLU is given by

$$\Big(\text{modReLU}(\boldsymbol{z})\Big)_i = \frac{z_i}{|z_i|} \odot \text{ReLU}(|z_i| + b_i), \tag{4.5}$$

where $|z_i|$ is the norm of the complex number z_i. The complex unitary matrix $\boldsymbol{W}_h$ is parametrized as a product of a diagonal matrix and multiple rotational matrices. The reparametrization used here is the one proposed in Jing et al., 2017. The complex input weight matrix $\boldsymbol{W}_o \in \mathbb{C}^{d_h \times d_o}$ is initialized with $\boldsymbol{W}_o^{re} + j\, \boldsymbol{W}_o^{im}$, with real matrices $\boldsymbol{W}_o^{re}$, $\boldsymbol{W}_o^{im} \in \mathbb{R}^{d_h \times d_o}$ 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 Figure 4.1(b).

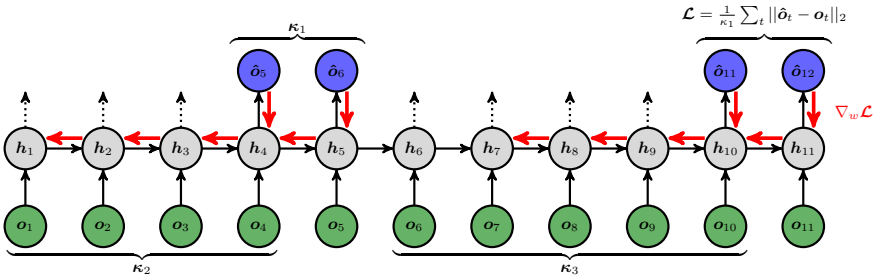


FIGURE 4.2: Illustration of an unfolded RNN. Time series data $\boldsymbol{o}$ are provided at the input of the RNN. The RNN is forecasting the evolution of the observable at its outputs $\tilde{\boldsymbol{o}}$. The average difference (mean square error) between κ_1 iterative predictions (outputs) of the RNN $\tilde{\boldsymbol{o}}$ and the targets $\boldsymbol{o}$ from the time series data is computed every κ_3 steps. The gradient of this quantity, illustrated with red arrows, is backpropagated through time for κ_2 previous temporal timesteps, computing the gradients of the network parameters that are shared at each time layer. The output of intermediate steps illustrated with dashed lines is ignored. Stateless models initialize the hidden state before training at a specific fragment of the sequence of size κ_2 with zero (in this case $\boldsymbol{h}_6 \hat{=} 0$) and cannot capture dependencies longer than κ_2. In this way, consecutive training batches (sequence fragments) do not have to be temporally adjacent. In stateful models, the hidden state is never set to zero, and in order to train at a specific fragment of the sequence, the initial hidden state has to be computed from the previously processed fragment. In order to eliminate the overlap between fragments, we teacher-force the network with ground-truth data for $\kappa_3 \geq \kappa_2$ timesteps. Here, we pick $\kappa_3 = \kappa_2 + \kappa_1 - 1$ as illustrated in the figure.

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 handwritten numbers task, demonstrating superior performance to LSTM either in terms of final testing accuracy or wall-clock training speed.

4.2.3 *Backpropagation Through Time*

Backpropagation dates back to the works of Dreyfus, 1962; Linnainmaa, 1976; Rumelhart et al., 1986, while its extension to RNNs, Backpropagation through time (BPTT) was presented in P. J. 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 backward 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 BBTT 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. At the same time (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 this chapter, we utilize BBTT to train the LSTM (Section 2.1.4.2), GRU (Section 4.2.1) and Unitary (Section 4.2.2) RNNs. There are three critical parameters of this training method that can be tuned. The first hyperparameter κ_1 is the number of forward-pass timesteps performed to accumulate the error for backpropagation. The second parameter is the number of previous timesteps for the backpropagation of the gradient κ_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 κ_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 κ_2 is hard-coded to zero, i.e., $\boldsymbol{h}_{-\kappa_2} = 0$. "Stateless" models cannot learn dependencies that expand in a time horizon larger than κ_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 $\boldsymbol{h}_{-\kappa_2} \neq 0$ between temporally consecutive batches. Here, we consider only "stateful" networks.

Training "stateful" networks is challenging because the hidden state $\boldsymbol{h}_{-\kappa_2}$ has to be available from a previous batch, and the network has to be trained to learn temporal dependencies that may span many timesteps in the past. In order to avoid overlap between two subsequent data fragments and compute $\boldsymbol{h}_{-\kappa_2}$ for the next batch update, the network is teacher-forced for κ_3 timesteps between two consecutive weight updates. That implies

providing ground-truth values at the input and performing forward-passing without any backpropagation. This parameter influences the training speed, determining how often the weights are updated. We pick $\kappa_3 = \kappa_2 + \kappa_1 - 1$ as illustrated in Figure 4.2, and optimize κ_1 as a hyperparameter.

The weights of the networks are initialized using the method of Xavier proposed in Glorot et al., 2010. We utilize a stochastic optimization method with an adaptive learning rate called Adam (Kingma et al., 2014) to update the weights and biases. We add Zoneout (Krueger et al., 2017) regularization in the recurrent weights and variational dropout (Gal et al., 2016) regularization at the output weights (with the same keep probability) to both GRU and LSTM networks to alleviate overfitting. Furthermore, following P. R. Vlachas, Byeon, et al., 2018 we add Gaussian noise sampled from $\mathcal{N}(0, \kappa_n \sigma)$ to the training data, where σ is the standard deviation of the data. The noise level κ_n is tuned. Moreover, we also vary the number of RNN layers by stacking residual layers (He et al., 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 timesteps, 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 $\mathcal{I} = \{1, \ldots, T\}$, 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 $\mathcal{I}$, training it with BBTT. Next, we remove all the intermediate indexes we trained on from $\mathcal{I}$. We repeat this process until $\mathcal{I} = \emptyset$, 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 the performance of the methods used in the analysis of this chapter.

As an additional overfitting 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 $N_{patience} = 20$ consecutive epochs, the training round is over. We train the network for $N_{rounds} = 10$ 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. β_1 and β_2 in the original

paper (Kingma et al., 2014), did not lead to important differences in the results. For this reason and due to our limited computational budget, we use the default values proposed in the paper Kingma et al., 2014 ($\beta_1 = 0.9$ and $\beta_2 = 0.999$).

4.2.4 *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 and Jaeger, 2009). RC relies on randomly selecting the recurrent weights such that the hidden state captures the history of the evolution of the observable $\boldsymbol{o}_t$ 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 that it captures temporal features of the observed state history. This technique has been proposed in the context of Echo-State-Networks (ESNs) (Jaeger et al., 2004) and Liquid State Machines with spiking neurons (LSM) (Maass et al., 2002).

We consider reservoir computing networks with $\mathcal{F}_{hh}$ given by the functional form

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}_{h,o}\boldsymbol{o}_t + \boldsymbol{W}_{h,h}\boldsymbol{h}_{t-1}\right), \tag{4.6}$$

where $\boldsymbol{W}_{h,o} \in \mathbb{R}^{d_h \times d_o}$, and $\boldsymbol{W}_{h,h} \in \mathbb{R}^{d_h \times d_h}$. Other choices of RC architectures are possible (Antonik et al., 2017; Haynes et al., 2015; Larger, Soriano, et al., 2012; Larger, Baylón-Fuentes, et al., 2017). Following Jaeger et al., 2004, the entries of $\boldsymbol{W}_{h,o}$ are uniformly sampled from $[-\omega, \omega]$, where ω is a hyperparameter. The reservoir matrix $\boldsymbol{W}_{h,h}$ 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 $\boldsymbol{h}_t$ conditioned on the input (observations $\boldsymbol{o}_t$) to be negative so that, for large t, the reservoir state $\boldsymbol{h}_t$ does not depend on initial conditions. For this purpose, $\boldsymbol{W}_{h,h}$ is set to a large low-degree matrix, scaled appropriately to possess a spectral radius (absolute value of the largest eigenvalue) ϱ 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

1 Because of the nonlinearity of the tanh function, $\varrho < 1$ is not necessarily required for the echo state property to hold true.

in Jiang et al., 2019. Following Pathak, Hunt, et al., 2018 the output coupling $\mathcal{F}_{ho}$ is set to

$$\widetilde{\boldsymbol{o}}_{t+1} = \boldsymbol{W}_{o,h}\hat{\boldsymbol{h}}_t, \tag{4.7}$$

where the augmented hidden state $\hat{\boldsymbol{h}}_t$ is a d_h dimensional vector such that the i^{th} component of $\hat{\boldsymbol{h}}_t$ is $\widetilde{h}_t^i = h_t^i$ for half of the reservoir nodes and $\widetilde{h}_t^i = (h_t^i)^2$ 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 the forecasting efficiency of RCs in the context of dynamical systems (Pathak, Hunt, et al., 2018). The matrix $\boldsymbol{W}_{o,h} \in \mathbb{R}^{d_o \times d_h}$ is trained with regularized least-squares regression with Tikhonov regularization to alleviate overfitting (Tikhonov et al., 1977; Yan et al., 2009) following the same recipe as in Pathak, Hunt, et al., 2018. The Tikhonov regularization $\widetilde{\eta}$ 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 short-term performance and stabilizing the RC in long-term forecasting. For this reason, we add noise sampled from $\mathcal{N}(0, \kappa_n \sigma)$ to the training data, where σ is the standard deviation of the data and the noise level κ_n a tuned hyperparameter.

4.3 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 in Appendix B.4. 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 centre (CSCS). In the following we quantify the prediction accuracy of the methods in terms of the Normalized Root Mean Squared Error (NRMSE), given by

$$\text{NRMSE}(\widetilde{\boldsymbol{o}}) = \sqrt{\left\langle \frac{(\widetilde{\boldsymbol{o}} - \boldsymbol{o})^2}{\boldsymbol{\sigma}^2} \right\rangle}, \tag{4.8}$$

where $\widetilde{\boldsymbol{o}} \in \mathbb{R}^{d_o}$ is the forecast at a single timestep, $\boldsymbol{o} \in \mathbb{R}^{d_o}$ is the target value, and $\boldsymbol{\sigma} \in \mathbb{R}^{d_o}$ is the standard deviation in time of each state component. In Equation 4.8, the notation $\langle \cdot \rangle$ 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.

Perhaps the most basic characterization of chaotic motion is through the concept of Lyapunov exponents (Ott, 2002) (LE): 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 LE Λ depends on the orientation $(\delta u(t)/|\delta u(t)|)$ of the vector displacement $\delta u(t)$. In the $t \to \infty$ limit, the number of possible values of Λ is typically equal to the state-space dimensionality. We denote these values $\Lambda_1 \geq \Lambda_2 \geq \Lambda_3 \geq \dots$ and collectively call them the Lyapunov exponent spectrum (LS) of the particular chaotic system. The LS will be evaluated in Section 4.6.

However, we note that a special role is played by Λ_1, and only Λ_1, the largest LE. We refer to the largest LE 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 Λ_1 with probability one. This is because in order for any of the other exponents $(\Lambda_2, \Lambda_3, \dots)$ 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 Λ_1, and $T^{\Lambda_1} = \Lambda_1^{-1}$, the "Lyapunov time", provides a characteristic timescale 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 Λ_1 as

$$\mathrm{VPT} = \frac{1}{\Lambda_1} \arg\max_{t_f} \{ t_f \,|\, \mathrm{NRMSE}(\boldsymbol{o}_t) < \varepsilon, \forall t \leq t_f \} \tag{4.9}$$

which is the largest time t_f the model forecasts the dynamics with a NRMSE error smaller than ε normalized with respect to Λ_1. In the following, we set $\varepsilon = 0.5$.

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 $i \in \{1, \dots, d_o\}$ elements $\boldsymbol{o}_t^i$ of the state (since the state $\boldsymbol{o}_t$ is a vector). The power spectrum

of the evolution of o_t^i is given by $\text{PSD}(f) = 20\log_{10}\left(2\,|\hat{o}(f)|\right)$ dB, where $\hat{o}(f) = \text{DFT}(o_t^i)$ is the complex Fourier spectrum of the state evolution.

4.4 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 timescale of this propagation is given by the MLE of the system as elaborated in Section 4.3. 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 $\boldsymbol{x}_t$, but we only have access to the less informative observable $\boldsymbol{o}_t$. The phase space state, i.e., the embedding state, is given by $\boldsymbol{z}_t = [\boldsymbol{o}_t, \boldsymbol{o}_{t-\tau}, \ldots, \boldsymbol{o}_{t-(d-1)\tau}]$, where the time-lag τ 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 et al., 1997) and that there exists a phase space forecasting rule $\boldsymbol{z}_{t+1} = \mathcal{F}^{\boldsymbol{z}}(\boldsymbol{z}_t)$, and thus an observable forecasting rule $\widetilde{\boldsymbol{o}}_{t+1} = \mathcal{F}^{\boldsymbol{o}}(\boldsymbol{o}_t, \boldsymbol{o}_{t-\tau}, \ldots, \boldsymbol{o}_{t-(d-1)\tau})$.

The recurrent architectures presented in Section 4.2 fit this framework, as the embedding state information can be captured in the high-dimensional hidden state $\boldsymbol{h}_t$ of the networks by processing the observable time series $\boldsymbol{o}_t$, without having to tune the embedding parameters τ 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 forecast the dynamics efficiently.

4.4.1 *Dimensionality Reduction on the Lorenz 96 Model*

Here, we consider the Lorenz 96 model introduced in Section 2.2.2. We consider a grid-size $J = 40$ and two different forcing regimes, $F = 8$ and $F = 10$. We solve Equation 2.17 starting from a random initial condition with a Fourth Order Runge-Kutta scheme and a timestep of $\delta t = 0.01$. We run the solver up to $T = 2000$ after ensuring that transient effects are discarded (eliminating the data from the first 1000 time units). The first half 10^5 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 Appendix B.3. 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 MLE, i.e., $T^{\Lambda_1} = 1/\Lambda_1$. The LS of the Lorenz 96 model is calculated using a standard technique based on QR decomposition (Abarbanel, 2012). This leads to $\Lambda_1 \approx 1.68$ for $F = 8$ and $\Lambda_1 \approx 2.27$ for $F = 10$.

4.4.2 *Results on the Lorenz 96 Model*

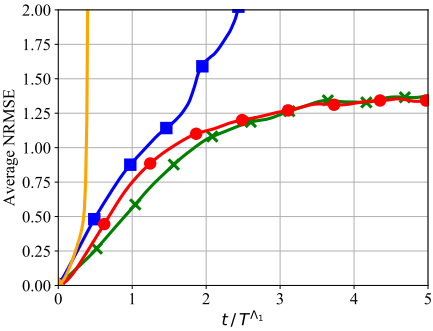
The evolution of the NRMSE of the model with the largest VPT of each architecture for $F \in \{8, 10\}$ is plotted in Figure 4.3 for two values of the dimension of the observable $d_o \in \{35, 40\}$, where $d_o = 40$ corresponds to full state information. Note that the observable is given by first transforming 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

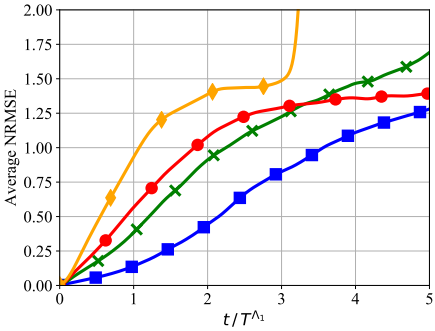
$F \in \{8, 10\}$. This divergence (inability to reproduce the long-term climate of the dynamics) stems from the iterative propagation of the forecasting error. The issue has also been demonstrated in RNNs as we saw in Chapter 3 (P. R. Vlachas, Byeon, et al., 2018), and previous studies in RC (Lu et al., 2018; Pathak, Wikner, 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 Ref. (Lu et al., 2018), assuming that the network has a full LS near the attractor, if any of the LEs 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 LE corresponds to infinitesimal perturbations transverse to the attractor will cause divergence.

Here, 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 Figure 4.3(a)-Figure 4.3(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 $\varepsilon = 1$, 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 $\varepsilon = 0.5$) is limited to one Lyapunov time when applied to reduced-order data and up to two Lyapunov times in the full state.

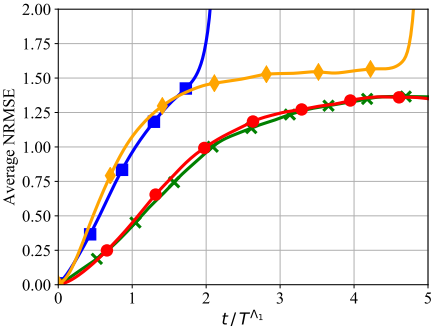
In order to analyze the sensitivity of the VPT to the hyperparameter selection, we present violin plots in Figure 4.4, showing a smoothed kernel density estimate of the VPT values of all tested hyperparameter sets for



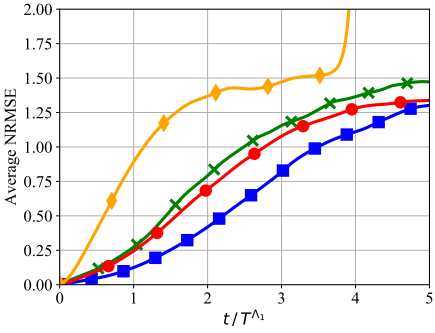
(a) Reduced-order observable ($d_o = 35$), $F = 8$



(b) Full state ($d_o = 40$), $F = 8$



(c) Reduced-order observable ($d_o = 35$), $F = 10$



(d) Full state ($d_o = 40$), $F = 10$

FIGURE 4.3: The evolution of the NRMSE error (average over 100 initial conditions) of the model with the highest VPT for each architecture in the Lorenz 96 with $F \in \{8, 10\}$ and $d_o \in \{35, 40\}$. Reservoir computing networks show remarkable predictive capabilities when the full state is observed, surpassing all other models (plots (b) and (d)). Predictions of Unitary networks diverge from the attractor in all scenarios, while iterative forecasts of RC suffer from instabilities when only partial information of a reduced-order observable is available. In contrast, GRU and LSTM show stable behavior and superior performance in the reduced-order scenario (plots (a) and (c)).
RC ; GRU ; LSTM ; Unit ;

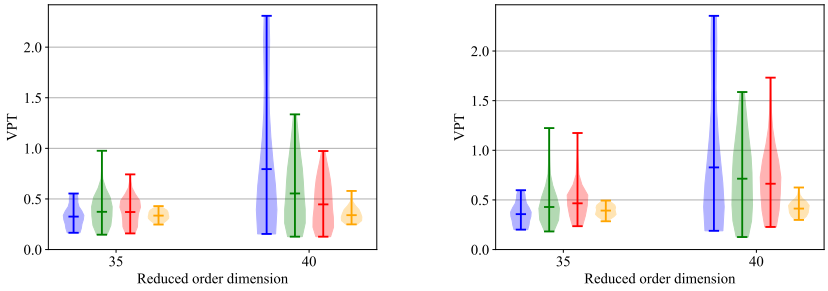
Model \ Scenario	$F = 8$				$F = 10$			
	$d_o = 35$		$d_o = 40$		$d_o = 35$		$d_o = 40$	
	MAX	AVG	MAX	AVG	MAX	AVG	MAX	AVG
Unit	0.43	0.34	0.58	0.34	0.49	0.39	0.63	0.41
LSTM	0.74	**0.37**	0.97	0.45	**1.17**	**0.47**	1.73	0.66
GRU	**0.98**	**0.37**	1.34	0.55	**1.22**	**0.43**	1.59	0.71
RC	0.55	0.32	**2.31**	**0.79**	0.60	0.36	**2.35**	**0.83**

TABLE 4.1: Maximum and average Valid Prediction Time (VPT) over all hyperparameter sets averaged over 100 initial conditions sampled from the testing data for each model.

$d_o = 35$ and $d_o = 45$ and $F = 8$ and $F = 10$. The horizontal markers denote the maximum, average, and minimum values. Quantitative results for both $F \in \{8, 10\}$ are provided on Table 4.1.

In the full state scenario ($d_o = 40$) and forcing regime $F = 8$, RC shows a remarkable performance with a maximum VPT ≈ 2.31, while GRU exhibits a max VPT of ≈ 1.34. The LSTM has a max VPT of ≈ 0.97, while Unitary RNNs show the lowest forecasting ability with a max VPT of ≈ 0.58. From the violin plots in Figure 4.4 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 ≈ 1.73, which is higher than that of GRU, which is ≈ 1.59. Still, the VPT of RC in the full state scenario is ≈ 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 ≈ 0.98 compared to LSTM showing a max VPT ≈ 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 ≈ 1.22 and ≈ 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≈ 0.55 for $F = 8$ and ≈ 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



(a) Forcing regime $F = 8$ (b) Forcing regime $F = 10$

FIGURE 4.4: 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 ; GRU ; LSTM ; Unit ;

to the expressiveness of Unitary networks but rather to the difficulty in identifying the right hyperparameters (Greff et al., 2016). In Figure 4.4 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 the reduced-order state is a more difficult task than 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 here, forecasts of Unitary RNN networks fail to remain close to the attractor and diverge. For this reason, we omit the results on these networks. This divergence effect that appears in both Unitary and RC networks is quantified in Appendix B.5.

We quantify the long-term behavior in terms of the power spectrum of the predicted dynamics and its difference with the true spectrum of the testing data. In Figure 4.5, 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 (Figure 4.5(b), Figure 4.5(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

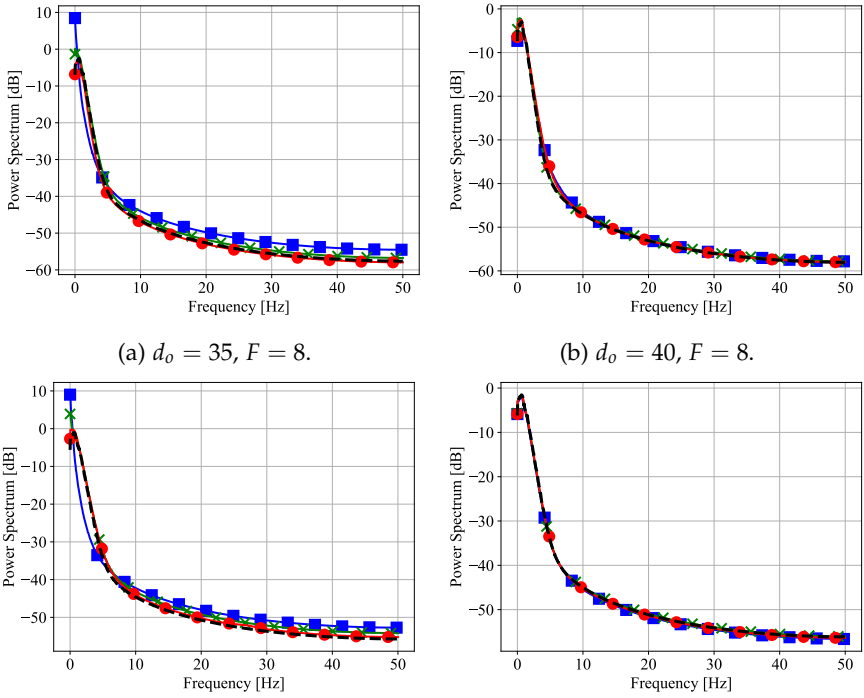
powerful predictive tool in the full-order state scenario, as RC models both capture the long-term statistics and have the highest VPT among all other models considered in this analysis. However, in the case of a reduced-order observable, the RC cannot match the statistics. In contrast, GRU and LSTM networks achieve superior forecasting performance while matching the long-term statistics, even in this challenging setting of a chaotic system with reduced-order information.

In Appendix B.7, we provide an analysis of the robustness of the results presented here with respect to the hyperparameters of BPTT.

An important aspect of machine learning models is their scalability to high-dimensional systems and their training time and memory utilization requirements. Large memory requirements and 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 Figure 4.6(a) and Figure 4.6(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 for an input dimensions of $d_o = 35$ (reduced-order) and $d_o = 40$ (full dimension) respectively. Figure 4.6(b) and Figure 4.6(e), show the corresponding Pareto fronts of the VPT with respect to the training time. In the case of the full state-space ($d_o = 40$), the RC can achieve superior VPT with smaller memory usage and vastly shorter training time than the other methods. However, in the case of reduced-order information ($d_o = 35$), the BPTT algorithms (GRU and LSTM) are superior to the RC even when the latter is provided with one order of magnitude more memory.

Because the RNN models are learning the recurrent connections, they can 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 GRU/LSTM trained with backpropagation (see Figure 4.6(a)).

At the same time, letting memory limitations aside, training of RC models requires the solution of a linear system of equations $\mathbf{H}\boldsymbol{W}_{out}^T = \boldsymbol{Y}$, with $\mathbf{H} \in \mathbb{R}^{d_N \times d_h}$, $\boldsymbol{W}_{out}^T \in \mathbb{R}^{d_h \times d_o}$ and $\boldsymbol{Y} \in \mathbb{R}^{d_N \times d_o}$ (see Appendix B.1). 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



(a) $d_o = 35$, $F = 8$.

(b) $d_o = 40$, $F = 8$.

(c) $d_o = 35$, $F = 10$.

(d) $d_o = 40$, $F = 10$.

FIGURE 4.5: 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 model 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 ;

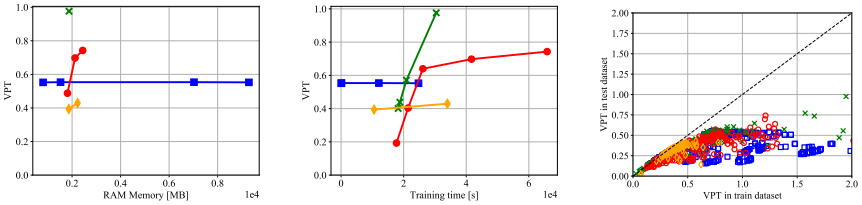
on diagonalization without any significant influence on the training time. In contrast, the training time of an RNN is tough to estimate a priori, as the convergence of the training method depends on initialization and various other hyperparameters and is not necessarily dependent on the size. That is why we observe a more significant variation of the training time of RNN models. Similar results are obtained for $F = 10$, the interested reader is referred to Appendix B.6.

In the following, we evaluate to which extent the trained models overfit 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 into the generalization error of the models. The results are shown in Figure 4.6(c), and Figure 4.6(f) for $d_o = 35$ and $d_o = 40$. Ideally, a model architecture that effectively guards against overfitting exhibits a low generalization error and should be represented by a plot point 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 ($\widetilde{\eta}$). However, this requires rerunning the training for every other combination of hyperparameters. For the four tested values $\widetilde{\eta} \in \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ of the Tikhonov regularization parameter the RC models tend to exhibit higher generalization error compared to the RNNs trained with BBTT. We also tested more values $\widetilde{\eta} \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}\}$, 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 Figure 4.6(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 Figure 4.8 and Figure 4.7 for $F = 8$ and $d_o \in \{35, 40\}$. Unitary networks suffer from the propagation of forecasting error, and eventually, their forecasts



(a) VPT w. r. t. RAM memory for $d_o = 35$.

(b) VPT w. r. t. the total training time for $d_o = 35$.

(c) VPT in test data w. r. t. VPT in the train data for $d_o = 35$.

(d) VPT w. r. t. RAM memory for $d_o = 40$.

(e) VPT w. r. t. the total training time for $d_o = 40$.

(f) VPT in test data w. r. t. VPT in the train data for $d_o = 40$.

FIGURE 4.6: Forecasting results on the dynamics of an observable consisting of the SVD modes of the Lorenz 96 model with $F = 8$ 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 ■(or □) ; GRU ✚(or ✕) ; LSTM ●(or ○) ; Unit ◆(or ◇) ; Ideal ▪ ▪ ▪;

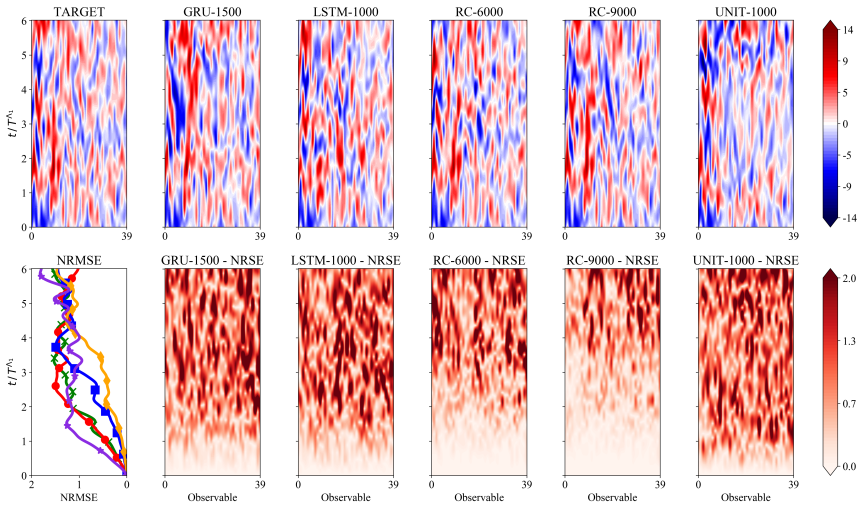


FIGURE 4.7: Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz 96 model with $F = 10$ in the testing dataset with GRU, LSTM, RC, and a Unitary network along with the true (target) evolution and the associated NRSE contours for the full state observable $d_o = 40$. The evolution of the component average normalized RMSE (NRMSE) is plotted to facilitate comparison.
GRU ; LSTM ; RC-6000 ; RC-9000 ; Unit ;

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. RC models achieve superior forecasting accuracy in the full-state setting 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 frequently diverge from the attractor. An analysis of the robustness of the predictive performance and generalization ability of the networks as a function of the noise in the data is provided in Appendix B.2.

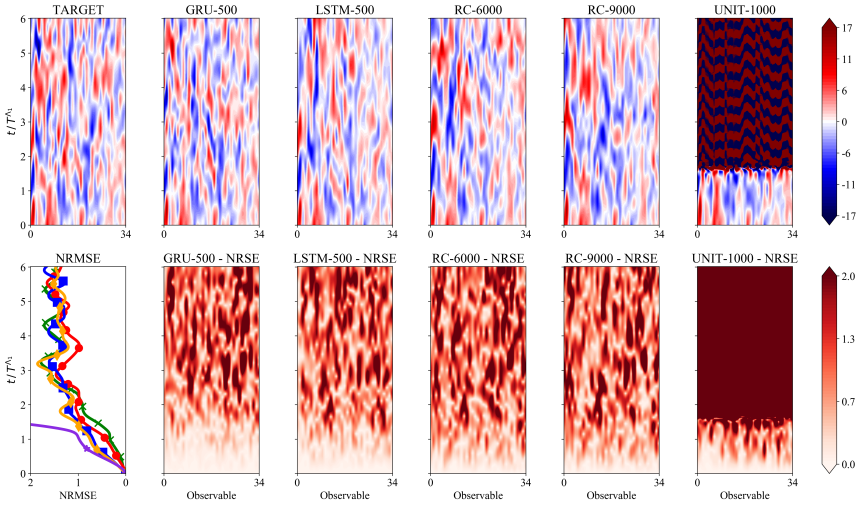


FIGURE 4.8: Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz 96 model with $F = 10$ in the testing dataset with GRU, LSTM, RC, and a Unitary network along with the true (target) evolution and the associated NRSE contours for the reduced-order observable $d_o = 35$. The evolution of the component average normalized RMSE (NRMSE) is plotted to facilitate comparison.
GRU ✱; LSTM ●; RC-6000 ■; RC-9000 ◆; Unit ▲;

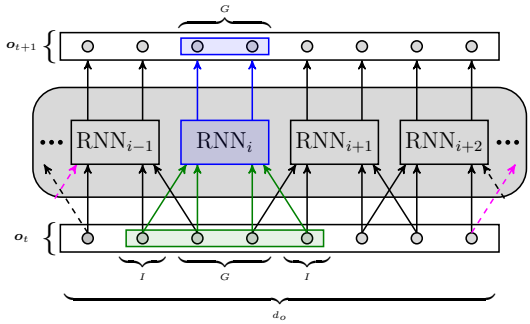


FIGURE 4.9: Illustration of the parallel architecture for a group size of $G = 2$ and an interaction length of $I = 1$. The network consists of multiple RNNs with different parameters. Each RNN is trained to forecast the evolution of G elements of the observable. Additional information of I elements from each neighboring network (left and right) are provided as additional input to capture local correlations.

4.5 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 4.2 do not scale efficiently with the input dimension, i.e., as the dimensionality of the observable $\boldsymbol{o}_t \in \mathbb{R}^{d_o}$ increases. Two limiting factors are the required time and RAM memory to train the model. As d_o increases, the size d_h 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 to RNNs. More limiting factors arise by taking the process of identification of optimal model hyperparameters into account since loading, storing, and processing a vast 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 Figure 4.9 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 Parlitz et al., 2000; Pathak, Hunt, et al., 2018. The idea dates back to local delay coordinates (Parlitz et al., 2000). The model shares ideas from convolutional RNN architectures (Sainath et al., 2015; Shi, Z. Chen, et al., 2015) designed to capture local features that are translationally invariant in image and video processing tasks. In this section, we extend this parallelization scheme to RNNs and compare the efficiency of parallel RNNs and RCs in forecasting the state dynamics of the Lorenz 96 model and Kuramoto-Sivashinsky equation discretized in a fine grid.

4.5.1 *Parallel Architecture*

Assume that the observable is $\boldsymbol{o}_t \in \mathbb{R}^{d_o}$ and each element of the observable is denoted by $\boldsymbol{o}_t^i \in \mathbb{R}, \forall i \in \{1, \ldots, d_o\}$. In the case of local interactions, the evolution of each element is affected by its spatially neighboring grid points. The elements $\boldsymbol{o}^i$ are split into N_g groups, each of which consisting of G spatially neighboring elements such that $d_o = GN_g$. 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 $\boldsymbol{o}^i$. Each of the N_g RNNs receives G inputs $\boldsymbol{o}^i$ 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 Figure 4.9.

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 Message Passing Interface (MPI) (L. Dalcín et al., 2008; L. D. Dalcín

et al., 2011; D. W. Walker et al., 1996) communication protocol is utilized for communicating the elements of the interaction for long-term forecasting.

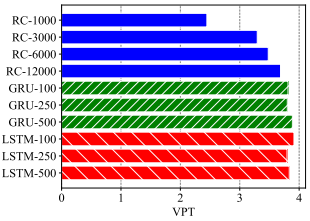
4.5.2 *Results on the Lorenz 96 Model*

In this section, we employ the parallel architecture to forecast the state dynamics of the Lorenz 96 model explained in Section 4.4.1 with a state dimension of $d_o = 40$. Note that in contrast to Section 4.4.2, 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.

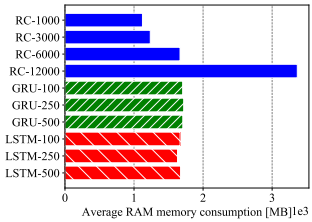
Instead of a single RNN model forecasting the $d_o = 40$ dimensional global state (composed of the values of the state in the 40 grid nodes), we consider $N_g = 20$ separate RNN models, each forecasting the evolution of a $G = 2$ dimensional local state (composed of the values of the state in 2 grid nodes). In order to forecast the evolution of the local state, we take into account its interaction with $I = 4$ 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 an $2I + G = 10$ 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 Appendix B.4. 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 Figure 4.10(a), we plot the VPT time of the RC and the BPTT networks. We find that RNNs trained by BPTT achieve comparable predictions with RC, albeit using a much smaller number of hidden nodes (between 100 and 500 for BPTT vs. 6000 to 12000 for RC). Finally, 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 Figure 4.10(c). At the same time, using 12000 nodes for RC implies high RAM requirements, more than 3 GB per rank, as depicted in Figure 4.10(b).

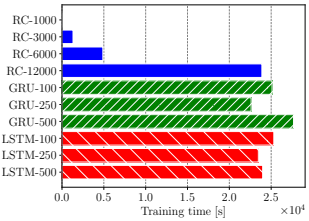
As elaborated in Section 4.4.2 and depicted in Figure 4.3(a), the VPT reached by large nonparallelized models that are forecasting the 40 SVD modes of the system is approximately 1.4. We also verified that the non-parallelized models of Section 4.4.1 when forecasting the 40 dimensional



(a) Valid prediction time in the test dataset



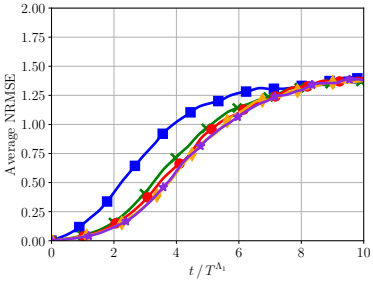
(b) Average RAM memory requirement



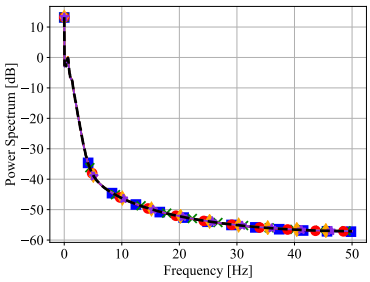
(c) Training time

FIGURE 4.10: (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 to more than 12000 nodes was not feasible due to memory requirements.

state containing local interactions instead of the 40 modes of SVD, reach the same predictive performance. Consequently, as expected, the VPT remains the same whether we are forecasting the state or the SVD modes as the system is deterministic. By exploiting the local interactions and employing the parallel networks, the VPT is increased from ≈ 1.4 to ≈ 3.9 as shown in Figure 4.10(a). The NRMSE error of the best performing hyperparameters is given in Figure 4.11(a). All models can reproduce the climate as the reconstructed power spectrum plotted in Figure 4.11(b) matches the true one. An example of an iterative prediction with LSTM, GRU, and RC models starting from an initial condition in the test dataset is provided in Figure 4.12.



(a) NRMSE error evolution



(b) Power spectrum

FIGURE 4.11: (a) The evolution of the NRMSE error (averaged over 100 initial conditions) of different parallel models in the Lorenz 96 with state dimension $d_o = 40$. (b) The reconstructed power spectrum. All models accurately capture the power spectrum. RCs with $d_h \in \{6000, 12000\}$ nodes are needed to match the predictive performance of an LSTM with 100 nodes.
RC-1000 ; RC-6000 ; RC-12000 ; GRU-500 ; LSTM-100 ; Groundtruth ;

4.5.3 *The Kuramoto-Sivashinsky Equation*

Here, we consider the KS model given in Section 2.2.1. We consider periodic boundary conditions $u(0,t) = u(L,t)$. In the following, we select $\nu = 1, L = 200$, $\delta t = 0.25$ and a grid of $d_o = 512$ nodes. We consider the discretized KS equation (Equation 2.16) with periodic boundary conditions and solve it using the fourth-order method for stiff PDEs introduced in Kassam et al., 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 into 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^{Λ_1} of the system (see Section 4.3) 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 $T^{\Lambda_1} \approx 0.094$.

4.5.4 *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.,

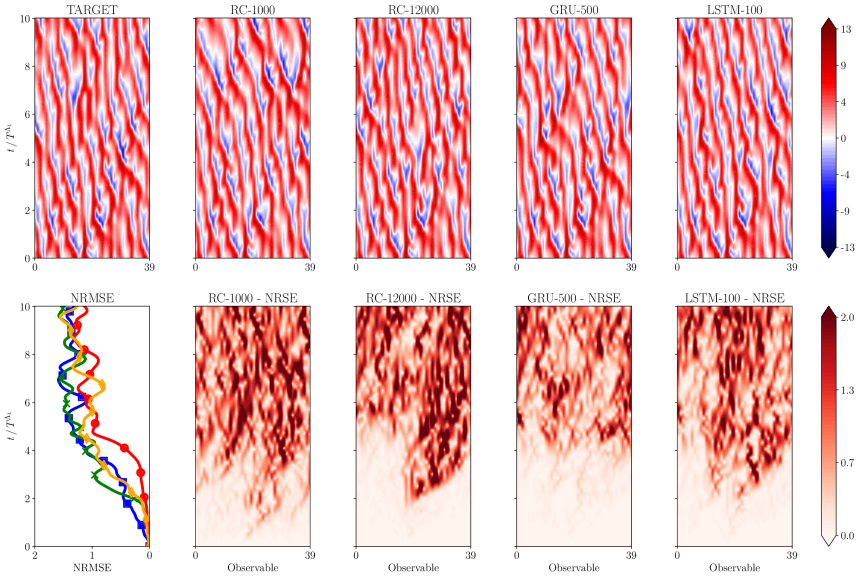


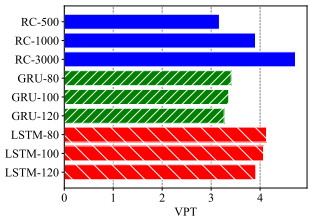
FIGURE 4.12: Contour plots of a spatio-temporal forecast in the testing dataset with parallel GRU, LSTM, and RC networks along with the ground-truth (target) evolution and the associated NRSE contours in the Lorenz 96 model with the full state as an observable $d_o = 40$. The evolution of the component average normalized RMSE (NRMSE) is plotted to facilitate comparison.
RC-1000 ; RC-12000 ; GRU-500 ; LSTM-100 ;

$d_o = 512$. 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 $d_h \in \{500, 1000, 3000\}$. For GRU and LSTM networks we vary $d_h \in \{100, 250, 500\}$. The rest of the hyperparameters are given in Appendix B.4. Results on Unitary networks are omitted, as predictions of Unitary networks diverge after a few timesteps in the iterative forecasting procedure.

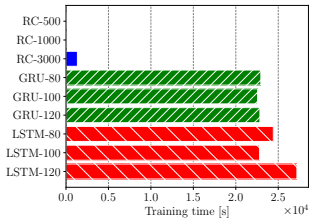
The results are summed up in the bar plots in Figure 4.13. In Figure 4.13(a), we plot the VPT time of the models. LSTM models reach VPTs of ≈ 4, while GRU show an inferior predictive performance with VPTs of ≈ 3.5. An RC with $d_h = 500$ reaches a VPT of ≈ 3.2, and an RC with 1000 modes reaches the VPT of LSTM models with a VPT of ≈ 3.9. Increasing the reservoir capacity of the RC to $d_h = 3000$ leads to a model exhibiting a VPT of ≈ 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 the 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 Figure 4.13, 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 Figure 4.13(b).

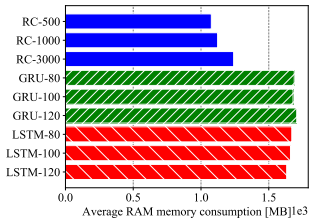
The evolution of the NRMSE is given in Figure 4.14(a). The predictive performance of a small LSTM network with 80 hidden units matches that of a large RC with 1000 hidden units. In Figure 4.14(b), the power spectrum of the predicted state dynamics of each model is plotted along with the true spectrum of the equations. The three models successfully captured 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 Figure 4.15.



(a) Valid prediction time in the test dataset

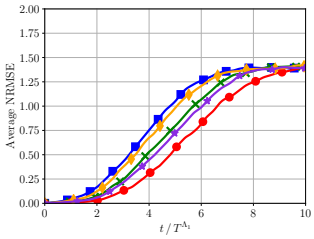


(b) Training time

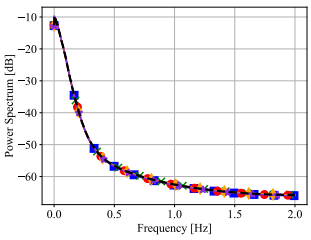


(c) Average RAM memory requirement

FIGURE 4.13: (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 $d_o = 512$.



(a) NRMSE error evolution



(b) Power spectrum

FIGURE 4.14: (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 ; RC-1000 ; RC-3000 ; GRU-80 ; LSTM-80 ; Groundtruth ;

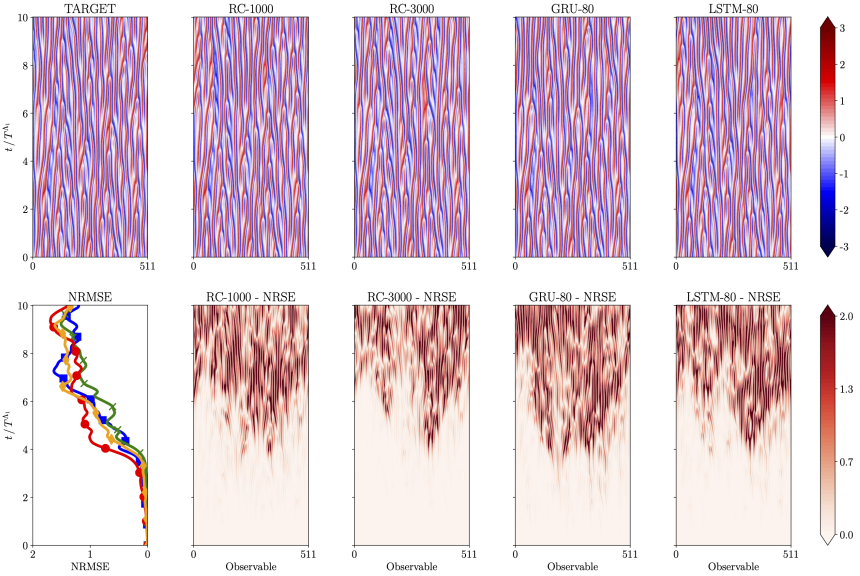


FIGURE 4.15: Contour plots of a spatio-temporal forecast starting from an initial condition in the testing dataset with parallel GRU, LSTM, and RC networks along with the ground-truth (target) evolution and the associated NRSE contours in the Kuramoto-Sivashinsky equation with the full state as an observable $d_o = 512$. The component average normalized RMSE (NRMSE) evolution is plotted to facilitate comparison.
RC-1000 ; RC-3000 ; GRU-80 ; LSTM-80 ; Groundtruth ;

4.6 CALCULATION OF LYAPUNOV SPECTRUM

The recurrent models utilized in this study can be used as surrogate models to calculate the LS 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 formula (Kaplan et al., 1979). Early efforts to solve the challenging problem of data-driven LE identification led to local approaches (Sano et al., 1985; Wolf et al., 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 et al., 2013) and calculate the LS 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 LS of the hidden state of this surrogate model. The RC method demonstrates excellent agreement for all positive LEs and many of the negative exponents for the KS equation with $L = 60$ (Pathak, Hunt, et al., 2018), alleviating the problem of spurious LEs of delay coordinate embeddings (Dechert et al., 1996). We build on top of this work and demonstrate that a GRU trained with BPTT can reconstruct the LS accurately with lower error for all positive LEs at the cost of higher training times.

The LS of the KS equation is computed by solving the KS equations in the Fourier space with a fourth-order time-stepping method (Kassam et al., 2005) and utilizing a QR decomposition approach as in Pathak, Hunt, et al., 2018. The LS 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 over an LSTM cell due to the fact that the latter has two coupled hidden states, rendering the computation of the LS mathematically more involved and computationally more expensive. The trained RNN model with GRU cell is used as a surrogate to compute

the full LS of the Kuramoto-Sivashinsky equation. Recall that the RNN dynamics are given by

$$\begin{aligned} \boldsymbol{h}_t &= \mathcal{F}_{hh}(\boldsymbol{o}_t, \boldsymbol{h}_{t-1}) \\ \boldsymbol{o}_{t+1} &= \mathcal{F}_{ho}(\boldsymbol{h}_t), \end{aligned} \tag{4.10}$$

where $\mathcal{F}_{hh}$ is the hidden-to-hidden and $\mathcal{F}_{ho}$ is the hidden-to-output mapping, $\boldsymbol{o} \in \mathbb{R}^{d_o}$ is an observable of the state, and $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is the hidden state of the RNN. All RNN models considered here share this common architecture. They only differ in the forms of $\mathcal{F}_{ho}$ and $\mathcal{F}_{hh}$. More importantly, the output mapping is linear, i.e.,

$$\boldsymbol{o}_{t+1} = \mathcal{F}_{ho}(\boldsymbol{h}_t) = \boldsymbol{W}_{h,o}\, \boldsymbol{h}_t. \tag{4.11}$$

The LEs are calculated based on the Jacobian $J = \frac{dh_t}{dh_{t-1}}$ of the hidden state dynamics along the trajectory. In the following we compute the Jacobian using Equation 4.10. By writing down the equations for two consecutive timesteps, we get

$$\text{Timestep } t-1 : h_{t-1} = \mathcal{F}_{hh}(o_{t-1}, h_{t-2}) \tag{4.12}$$

$$o_t = \mathcal{F}_{ho}(h_{t-1}) = W_o h_{t-1} \tag{4.13}$$

$$\text{Timestep } t : h_t = \mathcal{F}_{hh}(o_t, h_{t-1}). \tag{4.14}$$

The partial Jacobians needed to compute the total Jacobian are:

$$\frac{\partial \mathcal{F}_{hh}}{\partial o} = J_o^{hh} \in \mathbb{R}^{d_h \times d_o} \tag{4.15}$$

$$\frac{\partial \mathcal{F}_{hh}}{\partial h} = J_h^{hh} \in \mathbb{R}^{d_h \times d_h} \tag{4.16}$$

$$\frac{\partial \mathcal{F}_{ho}}{\partial h} = J_h^{oh} \in \mathbb{R}^{d_o \times d_h}. \tag{4.17}$$

In total we can write:

$$\frac{dh_t}{dh_{t-1}} = \frac{d\mathcal{F}_{hh}(o_t, h_{t-1})}{dh_{t-1}} = \frac{\partial \mathcal{F}_{hh}(o_t, h_{t-1})}{\partial o_t} \frac{\partial o_t}{\partial h_{t-1}} + \frac{\partial \mathcal{F}_{hh}(o_t, h_{t-1})}{\partial h_{t-1}} \implies \tag{4.18}$$

$$\frac{dh_t}{dh_{t-1}} = \frac{\partial \mathcal{F}_{hh}(o_t, h_{t-1})}{\partial o_t} \frac{\partial \mathcal{F}_{ho}(h_{t-1})}{\partial h_{t-1}} + \frac{\partial \mathcal{F}_{hh}(o_t, h_{t-1})}{\partial h_{t-1}} \implies \tag{4.19}$$

$$\frac{dh_t}{dh_{t-1}} = \underbrace{J_o^{hh}\Big|_{(o_t, h_{t-1})}}_{\text{evaluated at t}} \cdot \underbrace{J_h^{oh}\Big|_{h_{t-1}}}_{\text{evaluated at t-1}} + \underbrace{J_h^{hh}\Big|_{(o_t, h_{t-1})}}_{\text{evaluated at t}} \tag{4.20}$$

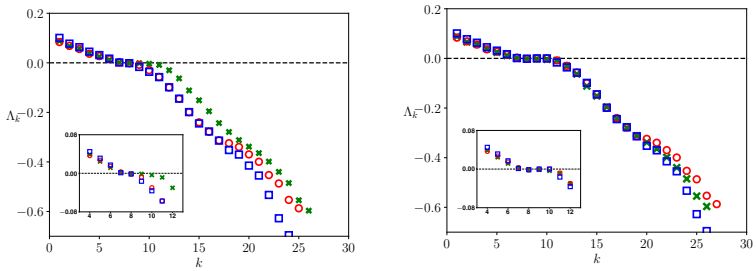
A product of this Jacobian along the orbit δ is developed and iteratively orthonormalized every T_n steps using the Gram-Schmidt method to avoid numerical divergence and keep the columns of the matrix R independent. We check the convergence criterion by tracking the estimated LE values every T_c timesteps. The input provided to the algorithm is a short time series of length T_{warm} to initialize the RNN and warm-up the hidden state $\widetilde{\boldsymbol{o}}_{1:T_w+1}$ (where the tilde denotes experimental or simulation data), the length of this warm-up time series T_{warm}, 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 ε used as an additional termination criterion. The function ColumnSum$(\cdot)$ computes the sum of each column of a matrix, i.e., 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 LEs of the RNN is given in Algorithm 1.

The identified maximum LE is $\Lambda_1 \approx 0.08844$. Here, 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 timesteps was needed to achieve convergence of the spectrum. The largest LE 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 ≈ 15, which is the largest LE such that $\sum_i \Lambda_i > 0$.

The first 26 LEs computed the GRU, RC as well as using the true equations of the Kuramoto-Sivashinsky are plotted in Figure 4.16. We observe a good match between the positive LEs by both GRU and RC surrogates. The positive LEs are characteristic of chaotic behavior. However, the zero LEs Λ_7 and Λ_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 does not alleviate the problem. In Figure 4.16(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 Figure 4.17. After 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.

Algorithm 1 Algorithm to calculate LS from trained RNN model

procedure LE_RNN($\widetilde{\boldsymbol{o}}_{1:T_w+1}, T_w, N, T, T_n, \varepsilon$)
  Initialize $\boldsymbol{h}_0 \leftarrow 0$.
  for $t = 1 : T_w$ **do** ▷ Warming-up the hidden state
    $\boldsymbol{h}_t \leftarrow \mathcal{F}_{hh}(\widetilde{\boldsymbol{o}}_t, \boldsymbol{h}_{t-1})$
  end for
  $\boldsymbol{h}_0 \leftarrow \boldsymbol{h}_{T_w}$, $\boldsymbol{o}_1 \leftarrow \widetilde{\boldsymbol{o}}_{T_w+1}$
  Pick random orthonormal matrix $\delta \in \mathbb{R}^{d_h \times N_{LE}}$.
  $\widetilde{T} \leftarrow T/T_n$
  Initialize $\widetilde{R} \leftarrow \mathbf{0} \in \mathbb{R}^{N \times \widetilde{T}}$.
  $l_{prev}, l \leftarrow \mathbf{0} \in \mathbb{R}^N$ ▷ Initializing the N LE to zero.
  $J_0 \leftarrow \nabla_h \mathcal{F}_{ho}(\boldsymbol{h}_0)$.
  for $t = 1 : T$ **do** ▷ Evolve the RNN dynamics
    $\boldsymbol{h}_t \leftarrow \mathcal{F}_{hh}(\boldsymbol{o}_t, \boldsymbol{h}_{t-1})$
    $\boldsymbol{o}_{t+1} \leftarrow \mathcal{F}_{ho}(\boldsymbol{h}_t)$
    $J_1 \leftarrow \nabla_h \mathcal{F}_{hh}(\boldsymbol{o}_{t+1}, \boldsymbol{h}_t)$. ▷ Calculating the partial Jacobians
    $J_2 \leftarrow \nabla_o \mathcal{F}_{hh}(\boldsymbol{o}_{t+1}, \boldsymbol{h}_t)$.
    $J \leftarrow J_1 + J_2 \cdot J_0$. ▷ Calculating the total Jacobian
    $\delta \leftarrow J \cdot \delta$ ▷ Evolving the deviation vectors δ
    if $\mod(t, T_{norm}) = 0$ **then** ▷ Re-orthonormalizing
      $Q, R \leftarrow QR(\delta)$
      $\delta \leftarrow Q[:, :N]$
      $\widetilde{R}[:, t/T_{norm}] \leftarrow \log(\mathrm{diag}(R[:N, :N]))$
      if $\mod(t, T_c) = 0$ **then** ▷ Checking convergence criterion
        $l \leftarrow \mathrm{Real}(\mathrm{ColumnSum}(\widetilde{R}))/(t * \delta t)$
        $l \leftarrow \mathrm{sort}(l)$
        $d \leftarrow |l - l_{prev}|_2$
        if $d < \varepsilon$ **then**
          break
        end if
      end if
    end if
    $J_0 \leftarrow \nabla_h \mathcal{F}_{ho}(\boldsymbol{h}_t)$.
  end for
  return l ▷ Return estimated Lyapunov exponents
end procedure

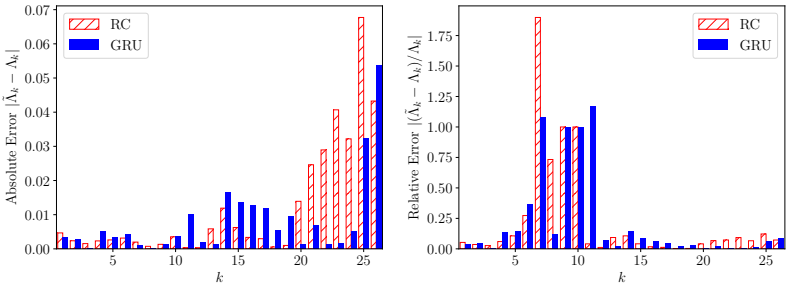


(a) Lyapunov spectrum.

(b) Lyapunov spectrum augmented with Λ_7, Λ_8 set to zero.

FIGURE 4.16: (a) Estimated Lyapunov exponents Λ_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, and the blue rectangles with GRU. In (b), we augment the computed spectrums with the two zero Lyapunov exponents Λ_7, Λ_8. Inset plots zoom in the zero-crossing regions.
True ✖; RC ○; GRU □;

In conclusion, GRU in par with RC networks can be used to replicate the chaotic behavior of a reference system and calculate the LS accurately.



(a) Absolute error (b) Relative error

FIGURE 4.17: (a) Absolute and (b) Relative error of the LS of the KS equation with $L = 60$. The LS identified using the GRU shows a better agreement with the spectrum identified by the Kuramoto-Sivashinsky equations.

4.7 DISCUSSION

In this chapter, 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 scale to systems with high-dimensional state-space, and how to guard against overfitting. Finally, we highlight the advantages and limitations of these methods and elucidate their applicability to forecasting spatio-temporal dynamics.

We considered three different types of RNN cells that alleviate the well-known vanishing and exploding gradient problem in backpropagation through time training (BPTT), namely LSTM, GRU, and Unitary cells. We benchmarked these networks against reservoir computing 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 model 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 utilize validation-based early stopping and regularization techniques (e. g. , Zoneout, Dropout) that guard against overfitting that 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 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 concerning 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 model 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 timesteps in both systems. Parallel GRU, LSTM, and RC networks reproduced the long-term attractor climate and 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.

Last but not least, we evaluated and compared the efficiency of GRU and RC networks in capturing the LS of the KS equation. The positive LEs are captured accurately by both RC and GRU. Both networks cannot reproduce two zero LEs Λ_7 and Λ_8. When these two are discarded from the spectrum, GRU and RC networks show comparable accuracy in terms of relative and absolute error of the LS.

5

SCHEDULED AUTOREGRESSIVE BACKPROPAGATION THROUGH TIME FOR LONG-TERM FORECASTING

As we demonstrate in Chapter 4, and indicated also in other recent works (Geneva et al., 2020; Pathak, Hunt, et al., 2018; Pathak, Lu, et al., 2017; P. R. Vlachas, Byeon, et al., 2018; P. R. Vlachas, Pathak, et al., 2020; Wan, P. Vlachas, et al., 2018), in the deterministic setting, RNNs are powerful approximators for modeling and forecasting of high-dimensional chaotic spatio-temporal dynamics. Moreover, RNNs extended with Convolutional Neural Networks (CNNs), either in the form of Convolutional RNN (ConvRNN) architectures (Shi, Z. Chen, et al., 2015) or coupled Convolutional Autoencoder RNN architectures (CNN-RNN) (Wiewel et al., 2019) can model high-dimensional spatio-temporal data, i. e. fluid flows or image data, where models based on equations are expensive or not available. The importance of long-term prediction of fluid flows is paramount for various practical cases from prediction of extreme events (Blonigan, Farazmand, et al., 2019), traffic management (Y. Li et al., 2017), surrogate modeling (Wiewel et al., 2019), to typhoon alert systems, and climate and precipitation forecasting (Kumar et al., 2020; Rasp et al., 2020; Shi, Z. Chen, et al., 2015; Shi, Z. Gao, et al., 2017). A recent literature survey on long-term spatio-temporal forecasting is given in Shi and Yeung, 2018. In this chapter, we develop a training procedure for RNNs to improve their predictive accuracy in the long term. The procedure can be applied in both ConvRNNs, and CNN-RNNs.

5.1 RELATED WORK

The workhorse of most deep learning algorithms is Backpropagation (BP) (LeCun, 1985; Parker, 1985; Rumelhart et al., 1986; P. Werbos, 1974). Backpropagation amounts to three steps: forward-pass, backward-pass, and the update step. In the forward-pass, the network's output is computed given the input data. Given the network's output, the loss is calculated based on the deviation between the output of the network and its target, encoding the goal. In the backward pass, the gradient of the loss with respect to the parameters of the network is calculated by iteratively applying the chain rule starting

from the network's output to the input. The gradient is finally used in the update step to alter the network's weights towards minimizing the loss based on an optimization algorithm (e. g. gradient descent). BP identifies each neuron's contribution to the network's overall performance (credit assignment problem) and then updates its value towards achieving the goal encoded in the minimized loss function.

In problems handling sequential or temporal data, RNNs are employed to spare computational resources considering the sequential aspect of the tasks. Backpropagation through time (BPTT) (Elman, 1990; P. J. Werbos, 1988, 1990) is the extension of BP to RNNs and temporal tasks. The use of BPTT has been widespread from Reinforcement Learning (Bakker, 2002), natural language and signal processing (Oord, Dieleman, et al., 2016), image processing (Gregor, Danihelka, Graves, et al., 2015), speech recognition (Ahmad et al., 2004), to forecasting (P. R. Vlachas, Byeon, et al., 2018), and its employment has been decisive to solve many forms of complex temporal credit assignment problems (Gers, Schraudolph, et al., 2002; Lillicrap et al., 2019). Video prediction has been a central research focus of the computer vision community recently (Castrejon et al., 2019; Fragkiadaki et al., 2015; Mathieu et al., 2015; Oh et al., 2015; Srivastava et al., 2015; J. Walker et al., 2014), where BPTT is utilized usually in a probabilistic context, e. g. variational RNNs (Castrejon et al., 2019; Chung, Kastner, et al., 2015).

In most applications, RNNs are trained with BPTT in the so-called "teacher-forcing" mode (Sutskever, 2013) (BPTT-TF), where sequences from the training dataset are provided in the input of the RNN, and the network is trained to minimize the one step ahead forecasting error. In our work, we argue that the gradient of the weights computed during training with BPTT-TF is biased towards one step ahead predictions. To make matters worse, the training loss is defined over the probability distribution of the training data. However, in the autoregressive testing phase, the probability distribution of the testing data might be very different, as the network was never trained on its own predictions. This discrepancy is, in general, known as exposure bias (Schmidt, 2019).

In the context of natural language processing (NLP), where probabilistic RNNs are employed, alternatives to BPTT-TF have been proposed to handle the exposure bias problem (Bowman et al., 2015; Norouzi et al., 2016; Ranzato et al., 2015; Sangiorgio et al., 2020; Sutskever, 2013). These works aim to remedy this discrepancy by replacing, masking or perturbing ground-truth context in "teacher-forcing" mode by samples of the distribution at the output of the network. Other notable works in the natural language

processing (NLP) literature propose alternatives to alleviate the pitfalls of BPTT-TF at the expense of limited temporal memory by deep autoregressive models (Gregor, Danihelka, A. Mnih, et al., 2014; Oord, Dieleman, et al., 2016; Oord, Kalchbrenner, et al., 2016; Vaswani et al., 2017) without hidden memory states, discussed in Miller et al., 2018. Closer to the method presented in this chapter, a scheduled sampling approach is considered in S. Bengio et al., 2015. However, the gradient is not backpropagated through the predicted outputs.

In this chapter, we propose an auxiliary loss that considers the iterative forecasting (autoregressive) error, debiasing the gradient by altering the computational graph of BPTT. A scheduled approach is considered, where the standard one step ahead prediction loss is minimized at early training epochs, while the loss is gradually switching to the autoregressive version at later stages. The procedure is termed Scheduled Autoregressive BPTT (BPTT-SA), while the loss is referred to as autoregressive loss.

We illustrate the merits of BPTT-SA in deterministic spatio-temporal prediction with RNNs and benchmark against standard BPTT and the schedule sampling approach of S. Bengio et al., 2015. Note that BPTT-TF is minimizing the one step ahead prediction error and is therefore biased towards short-term forecasting. In contrast, the autoregressive loss is biased towards the long-term. The two objectives can be contradicting, and even for a linear prediction model, training a new model might be a satisfactory strategy (J.-L. Lin et al., 1994). We demonstrate that the proposed BPTT-SA manages to conciliate the two objectives of short-term accuracy and long-term accuracy more effectively compared to the scheduled sampling approach of S. Bengio et al., 2015 in the viscous flow past a cylinder in a channel, at no extra training cost, alleviating the iterative propagation of the error, and improving long-term prediction. We find that the merits of BPTT-SA are negligible in low-dimensional time series prediction, where we considered forecasting of the Mackey-Glass equation and the Darwin sea level temperature time series dataset (see Appendix C.2).

This chapter is based on the paper "Scheduled Autoregressive Backpropagation Through Time for Robust Long-Term Spatiotemporal Forecasting" (P. R. Vlachas and Koumoutsakos, in preparation).

5.2 METHODS

Here, we assume that the RNN is processing an input stream $\{\boldsymbol{o}_0, \dots, \boldsymbol{o}_T\}$. The input stream is composed of the state $\boldsymbol{o} \in \mathbb{R}^{d_z}$. The recurrent mapping of the RNN, as introduced in Equation 2.2 and repeated here for clarity (with a slightly different notation for t), is given by $\boldsymbol{h}_{t+1} = \mathcal{F}_{hh}(\boldsymbol{o}_t, \boldsymbol{h}_t)$. Under the condition of linear activation functions (no nonlinearity in the RNN cell), or considering a linearization around a fixpoint, we may decompose the hidden-to-hidden mapping into two contributions, one from the previous hidden state $\mathcal{H}_h$, and one from the current state $\mathcal{H}_o$, i.e.

$$\begin{aligned} &\boldsymbol{h}_{t+1} = \mathcal{F}_{hh}(\boldsymbol{o}_t, \boldsymbol{h}_t) \implies \\ &\boldsymbol{h}_{t+1} \approx \mathcal{H}_h(\boldsymbol{h}_t) + \mathcal{H}_o(\boldsymbol{o}_t) \end{aligned} \tag{5.1}$$

where in order to simplify notation, the dependency of the mappings on the weights is omitted. The proposed method does not depend on this decomposition assumption, and its merits, illustrated in multiple applications later in this chapter, are valid independent of this analysis. This decomposition, and the following analysis, is serving us for the purpose of providing intuition about the merits of the proposed method.

In most scenarios, when the RNNs are trained for prediction of some state, the output is a prediction of the state at the next timestep $\boldsymbol{o}_{t+1}$. The output at the next timestep is computed based on the hidden-to-output mapping $\boldsymbol{o}_{t+1} = \mathcal{F}_{ho}(\boldsymbol{h}_{t+1})$ (see Equation 2.2). Both GRUs (Chung, Gulcehre, et al., 2014) and LSTMs (Hochreiter and Schmidhuber, 1997) can be framed under this unifying lens. In the following, to simplify notation, the mappings are considered as operators applied to other vectors or operators with the symbolism $\circ$, i.e.

$$\boldsymbol{h}_{t+1} = \mathcal{H}_h \circ \boldsymbol{h}_t + \mathcal{H}_o \circ \boldsymbol{o}_t, \tag{5.2}$$

and

$$\boldsymbol{o}_{t+1} = \mathcal{F}_{ho} \circ \boldsymbol{h}_{t+1}. \tag{5.3}$$

5.2.1 *Truncated Backpropagation Through Time*

Provided data from a stream $\{\widetilde{\boldsymbol{o}}_0, \dots, \widetilde{\boldsymbol{o}}_T\}$, RNNs are usually trained with truncated BPTT to learn to forecast the next value of the stream given some short history of size κ_2 (truncation length). The process was explained in previous chapters but is repeated here for completeness. The RNN is unfolded

over κ_2 steps. The data from a history of length κ_2, i.e. $\{\widetilde{\boldsymbol{o}}_{t-\kappa_2+1}, \ldots, \widetilde{\boldsymbol{o}}_t\}$ are provided to the inputs of the unfolded RNN. This is usually called "teacher-forcing" of the RNN, as ground-truth values are provided in the input. A loss is formulated based on the output of the RNN at step κ_2 and the target. For prediction purposes, the target is the stream value at the next timestep $\widetilde{\boldsymbol{o}}_{t+1}$. The loss can be written as $\mathcal{L}\big(\boldsymbol{o}_{t+1}, \widetilde{\boldsymbol{o}}_{t+1}\big)$. The loss measures the difference between the RNN prediction $\boldsymbol{o}_{t+1}$ and the target value $\widetilde{\boldsymbol{o}}_{t+1}$. A possible selection of the loss can be the mean squared error (MSE) loss, i.e. $\mathcal{L}\big(\boldsymbol{o}_{t+1}, \widetilde{\boldsymbol{o}}_{t+1}\big) = ||\boldsymbol{o}_{t+1} - \widetilde{\boldsymbol{o}}_{t+1}||_2^2$. The loss is a function of the input stream $\{\widetilde{\boldsymbol{o}}_{t-\kappa_2+1}, \ldots, \widetilde{\boldsymbol{o}}_t, \widetilde{\boldsymbol{o}}_{t+1}\}$, the initialization of the hidden state $\widetilde{\boldsymbol{h}}_{t-\kappa_2+1}$, and the weights of the RNN (omitted here for brevity). In order to see this dependency, we evaluate the output of the unfolded RNN, i.e $\boldsymbol{o}_{t+1} = \mathcal{F}_{ho}\big(\boldsymbol{h}_{t+1}\big) = \mathcal{F}_{ho} \circ \boldsymbol{h}_{t+1}$. By reformulating the expression for the output $\boldsymbol{o}_{t+1}$ iteratively applying Equation 5.2 for κ_2 times (unrolling the RNN functional form, see Equation C.1 in Appendix C.1.1), we get

$$\boldsymbol{o}_{t+1} = \mathcal{F}_{ho} \circ \boldsymbol{h}_{t+1} = \mathcal{F}_{ho} \circ (\mathcal{H}_h)_2^{\kappa} \circ \boldsymbol{h}_{t-\kappa_2+1} + \mathcal{F}_{ho} \circ \sum_{k=0}^{\kappa_2-1} (\mathcal{H}_h)^k \circ \mathcal{H}_o \circ \boldsymbol{o}_{t-k}. \tag{5.4}$$

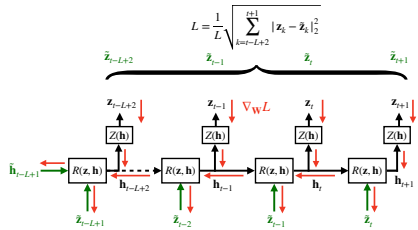
In classical truncated BPTT, Equation 5.4 is evaluated by "teacher-forcing" the RNN with input stream data $\{\widetilde{\boldsymbol{o}}_{t-\kappa_2+1}, \ldots, \widetilde{\boldsymbol{o}}_t\}$, and an initialization of the hidden state $\widetilde{\boldsymbol{h}}_{t-\kappa_2+1}$ at the point where the gradient flow is truncated. The hidden state at the gradient truncation step can be set from the previous batch ("stateful" RNN) or set to zero in "stateless" RNNs. Here, the "stateful" RNN case is considered. Thus, the functional form of the output $\boldsymbol{o}_{t+1}$ computed by unrolling the RNN and "teacher-forcing" is given by:

$$\begin{aligned} \boldsymbol{o}_{t+1} &= \mathcal{F}_{ho} \circ (\mathcal{H}_h)_2^{\kappa} \circ \widetilde{\boldsymbol{h}}_{t-\kappa_2+1} + \mathcal{F}_{ho} \circ \sum_{k=0}^{\kappa_2-1} (\mathcal{H}_h)^k \circ \mathcal{H}_o \circ \widetilde{\boldsymbol{o}}_{t-k} \\ &= \mathrm{TF}\big(\underbrace{\widetilde{\boldsymbol{h}}_{t-\kappa_2+1}}_{\text{initial hidden state}}, \underbrace{\widetilde{\boldsymbol{o}}_{t-\kappa_2+1}, \ldots, \widetilde{\boldsymbol{o}}_t}_{\text{input stream}} \big). \end{aligned} \tag{5.5}$$

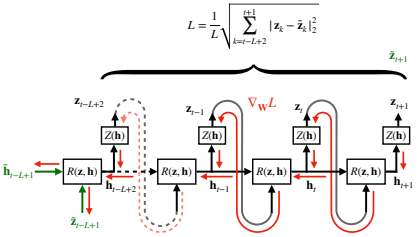
A schematic view of the computational graph of BPTT-TF and the backward flow of the gradient is illustrated in Figure 5.1.

5.2.2 *Autoregressive Backpropagation Through Time*

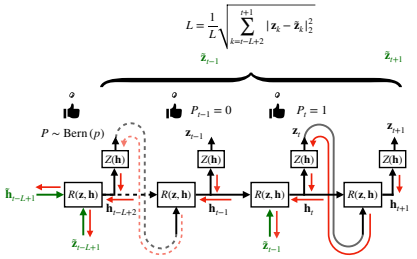
Here, we introduce a different variant of BPTT, i.e. the autoregressive scenario. In Autoregressive BPTT, the network is not teacher-forced. The



(a) BBTT-TF



(b) BPTT-A



(c) BPTT-SA

FIGURE 5.1: (a) Graph of the forward-pass and backward gradient flow of the BPTT with Teacher Forcing (BPTT-TF). Data input and target values are illustrated with green color, the forward-pass of the network with black colors, while the backward gradient pass with red. (b) In Autoregressive BPTT (BPTT-A) the output is propagated through the network, and only the data input at the first timestep is used. (c) In Scheduled Autoregressive BPTT (BPTT-SA) the behavior depends on the autoregressive probability p parametrizing a Bernoulli distribution.

output of the network $\boldsymbol{o}_t = \mathcal{F}_{h0}\boldsymbol{h}_t$ at the previous timestep is used, instead of the data value $\tilde{\boldsymbol{o}}_t$. The functional form of the output $\boldsymbol{o}_{t+1}$ computed by

unrolling the RNN and iteratively feeding the previous output is given by (elaborate derivation in Equation C.2 in Appendix C.1.2):

$$\boldsymbol{o}_{t+1} = \mathcal{F}_{ho} \circ \boldsymbol{h}_{t+1} = \mathcal{F}_{ho} \circ \left(\mathcal{H}_h + \mathcal{H}_o \circ \mathcal{F}_{ho}\right)^{\kappa_2 - 1} \circ \left(\mathcal{H}_h \circ \boldsymbol{h}_{t-\kappa_2+1} + \mathcal{H}_o \circ \boldsymbol{o}_{t-\kappa_2+1}\right) \tag{5.6}$$

In contrast to Equation 5.4, the output in Equation 5.6 is a function of the recursive operator $(\mathcal{H}_h + \mathcal{H}_o \circ \mathcal{F}_{ho})_2^{\kappa}$ that includes the operator $\mathcal{F}_{ho}$. In Equation 5.4, only the $\mathcal{H}_h$ operator is recursive. We argue that the inclusion of the hidden-to-output mapping in the recursive operator during training is vital to regularize the weights and avoid instabilities in long-term prediction, especially in long-term spatio-temporal forecasting, where the operator $\mathcal{F}_{ho}$ can be a large CNN.

In Autoregressive BPTT, the data used to compute the output $\boldsymbol{o}_{t+1}$ is the initial hidden state $\widetilde{\boldsymbol{h}}_{t-\kappa_2+1}$ and the initial input $\widetilde{\boldsymbol{o}}_{t-\kappa_2+1}$, i.e.

$$\begin{aligned} \boldsymbol{o}_{t+1} &= \mathcal{F}_{ho} \circ \left(\mathcal{H}_h + \mathcal{H}_o \circ \mathcal{F}_{ho}\right)^{\kappa_2 - 1} \circ \left(\mathcal{H}_h \circ \widetilde{\boldsymbol{h}}_{t-\kappa_2+1} + \mathcal{H}_o \circ \widetilde{\boldsymbol{o}}_{t-\kappa_2+1}\right) \\ &= \text{IF}\,\big(\underbrace{\widetilde{\boldsymbol{h}}_{t-\kappa_2+1}}_{\text{initial state}}, \underbrace{\widetilde{\boldsymbol{o}}_{t-\kappa_2+1}}_{\text{initial input}}\big). \end{aligned} \tag{5.7}$$

A schematic view of Autoregressive BPTT (BPTT-A) is given in Figure 5.1(b). Here, we consider a scheduled approach, BPTT-SA, inspired by S. Bengio et al., 2015.

In Scheduled Autoregressive BPTT (BPTT-SA) the behavior depends on the iterative forecasting probability p parametrizing a Bernoulli distribution. At each timestep, we decide on BPTT-TF or BPTT-A based on a sample from this distribution (coin-flip). The validation loss is computed for $p = 1$ (equivalent to BPTT-A). This ensures that the validation loss according to which we pick the optimal model is the autoregressive loss. During training, p follows an inverse sigmoid schedule for the training loss. At initial training epochs, p is close to zero, and the model is trained with the standard BPTT loss (equivalent to BPTT-TF). As training progresses, p is gradually increased till it reaches $p \to 1$ towards the final training epochs, leading to BPTT-A. The schedule is depicted in Figure 5.2 for a training procedure of 2000 epochs in total.

5.3 RESULTS

In the following, we compare the proposed BPTT-SA with standard BPTT-TF and the method proposed in S. Bengio et al., 2015 which is equivalent

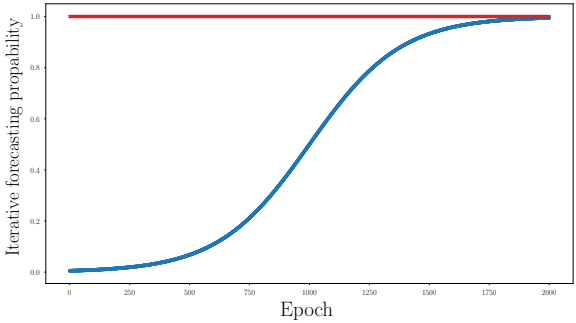


FIGURE 5.2: Schedule of iterative forecasting probability p for training (—●—) and validation loss (—×—).

to BPTT-SA without backpropagating the gradient. The latter is denoted BPTT-SS (scheduled sampling). All models are programmed in PYTHON (Van Rossum et al., 1995) in the PYTORCH (Paszke et al., 2019) library, and mapped to a single Nvidia Tesla P100 GPU. In all experiments, the models are trained with the Adam (Kingma et al., 2014) optimizer and validation-based early stopping.

5.3.1 *The Mackey-Glass Equation*

We evaluate the effectiveness of BPTT-SA in the Mackey-Glass (MG) chaotic time series, which is frequently used as a challenging benchmark problem for prediction methods due to its chaotic nature (Gers, Eck, et al., 2002; Voelker et al., 2019). The time series is generated by the delay differential equation

$$\frac{dx}{dt} = \frac{\alpha\, x(t-\tau)}{1 + x^n(t-\tau)} - \beta x(t). \tag{5.8}$$

We consider the parameter setting $\alpha = 0.2$, $\beta = 0.1$, $c = 10$, and $\tau = 17$.

The maximum Lyapunov exponent, calculated with the method in P. R. Vlachas, Pathak, et al., 2020 is $\Lambda_1 \approx 8.9 \cdot 10^{-3}$, leading to a Lyapunov time of $T^{\Lambda_1} = 112$ time units. We integrate Equation 5.8 with a fourth order Runge-Kutta scheme with $\delta t = 0.1$ up to $T = 2 \cdot 10^5$. The data are subsampled after integration to $\Delta t = 1.0$. 32 sequences of 1120 timesteps each (10 Lyapunov

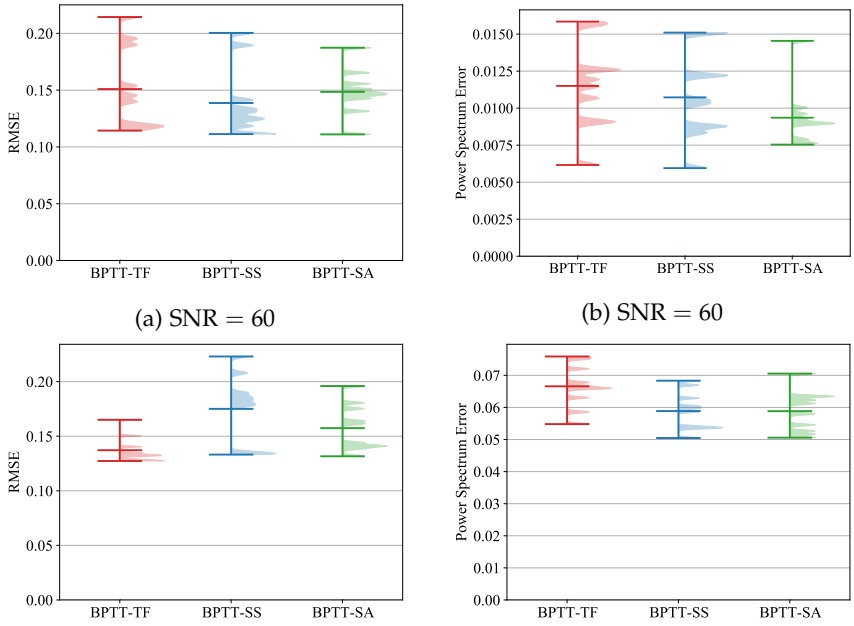
times) are generated for training. A validation set of the same size is considered for validation. The remaining data are considered for testing. In order to test the proposed algorithm in the autoregressive setting, 100 initial conditions are randomly sampled from the test data, and the networks are asked to forecast the next 896 steps, which amounts to approximately 8 Lyapunov times (after an initial warm-up period of 20 timesteps). As comparison metrics, we consider the Root Mean Squared Error (RMSE) and the error on the power spectrum (frequency content). Moreover, we consider two different noise levels on the data, a signal-to-noise ratio of SNR $= 60$, and a case of SNR $= 10$. The network hyperparameters are reported in Appendix C.3.

The results are illustrated in Figure 5.3. In the case of low-level noise (SNR $= 60$), all methods show approximately the same performance in terms of the RMSE. BPTT-SA shows slightly better performance on average and smaller variations between the different seeds (increased robustness) on the power spectrum error. However, the differences are minor. In the SNR $= 10$ noise level, all three methods exhibit similar errors.

5.3.2 *Viscous Flow Past a Cylinder in a Channel*

In the following, we consider long-term forecasting of the dynamics of the incompressible Navier-Stokes flow past a cylinder in a channel in Reynolds number Re $= 200$ in two dimensions. The reason for selecting this flow is that long-term prediction is possible and that the flow is characterized by reduced-order effective dynamics. The latter implies that snapshots of the flow can be mapped to a reduced-order latent space, representing the manifold of the effective dynamics.

We evaluate the effectiveness of BPTT-SA in two types of recurrent neural networks, Convolutional RNNs (ConvRNN) and Convolutional Autoencoder RNNs (CNN-RNNs). The latter first identify a latent reduced-order representation encoding the intrinsic dimensionality of the fluid flow and learn the temporal dynamics on the latent space. In contrast, ConvRNNs are replacing the operations on the RNN cell with convolutions while keeping the gating mechanisms (Kumar et al., 2020; Shi, Z. Chen, et al., 2015) and do not require low-dimensional intrinsic dynamics. For more information about the data generation, and hyperparameter tuning refer to Appendix C.4.1 and Appendix C.4.2 respectively.



(a) SNR = 60

(b) SNR = 60

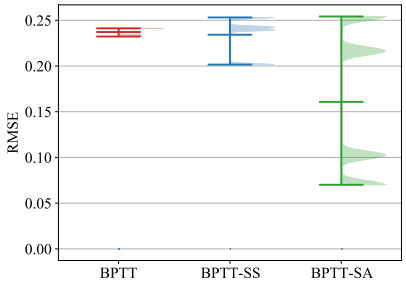
(c) SNR = 10

(d) SNR = 10

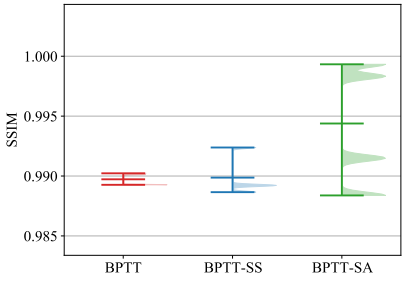
FIGURE 5.3: Evaluation of the performance of the training methods in forecasting the long-term dynamics of the Mackey-Glass time series. A prediction horizon of 896 timesteps is considered, and results are averaged over 100 initial conditions in the test data.

Two comparison metrics are considered, the RMSE error (the smaller, the better) and the structural similarity index measure (SSIM) (Z. Wang et al., 2004) (the higher, the better). The performance of the CNN-RNN models is illustrated in Figures 5.4(a) and 5.4(b). Four random seeds are considered to evaluate the robustness of the training algorithms. BPTT-SA leads on average to a drastic reduction of the RMSE, and increase in the SSIM. The same holds for ConvRNNs models as depicted in Figures 5.4(c) and 5.4(d). CNN-RNNs exhibit lower errors in both metrics compared to ConvRNNs as they take into account the reduced-order nature of the effective dynamics, and predict on a low-dimensional latent space.

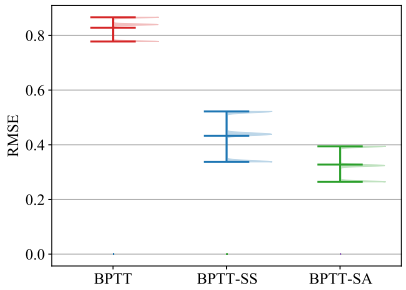
In Figure 5.5, we plot the evolution of the RMSE and the SSIM errors in time. We observe that BPTT-SA alleviates the error propagation and leads to more accurate long-term predictions in both metrics.



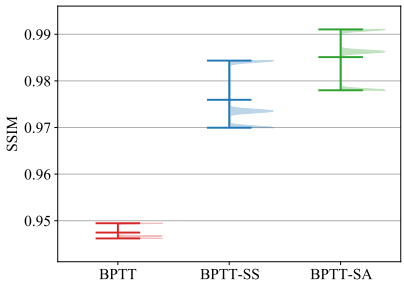
(a) CNN-RNN



(b) CNN-RNN

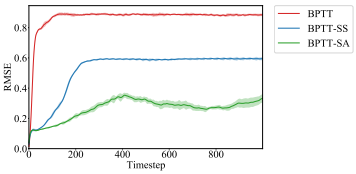


(c) Conv-RNN

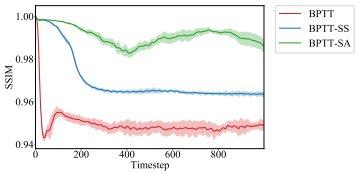


(d) Conv-RNN

FIGURE 5.4: Evaluation of the performance of Scheduled Autoregressive BPTT (BPTT-SA) in long-term prediction on the viscous flow past a cylinder in a channel for two model types, CNN-RNNs and ConvRNNs. Better prediction capability of a model is demonstrated by lower RMSE and higher SSIM scores.



(a) RMSE in ConvRNNs



(b) SSIM in ConvRNNs

FIGURE 5.5: The evolution of the SSIM and RMSE errors in time in long-term autoregressive prediction in the test data on the viscous flow past a cylinder in a channel for ConvRNNs. Better prediction capability of a model is demonstrated by lower RMSE and higher SSIM scores.

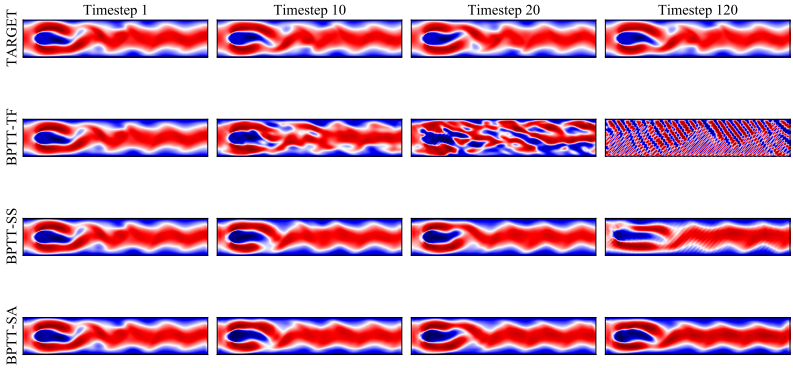


FIGURE 5.6: Prediction samples in different timesteps in the autoregressive testing mode of ConvRNN networks trained with different methods in the viscous flow past a cylinder in a channel dataset.

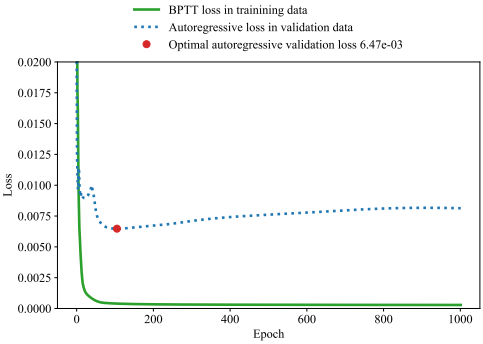
In Figure 5.6 we plot samples from the autoregressive testing phase for the ConvRNN models. We observe that models trained with BPTT and BPTT-SS lead to unphysical predictions after some timesteps. At lead time $T = 120$, only BPTT-SA captures the flow characteristics.

In the following, we analyze the behavior of the errors during training. The evolution of the training and validation error in the CNN-RNN training on the in viscous flow past a cylinder in a channel dataset is given in Figure 5.7. In BPTT, the training error decreases, but it does not capture the long-term autoregressive prediction error. For this reason, the autoregressive validation error is increasing, as the model is overfitting in the one-step-ahead prediction error. In BPTT-SS, the training error encodes the autoregressive loss due to the scheduling approach. However, as the method is not backpropagating the gradients, training is challenging as the iterative forecasting probability p is increased and the training error is not reduced. For this reason, the validation error also remains high. In contrast, in the model trained with BPTT-SA, the autoregressive validation error decreases, demonstrating that the training loss and the gradient successfully encode the objective. A similar behavior is observed in Figure 5.8 for the training of ConvRNN models.

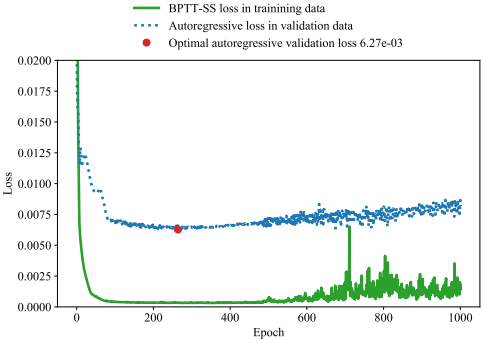
5.4 DISCUSSION

In this chapter, we proposed a scheduled autoregressive variant of BPTT (BPPT-SA) to alleviate the exposure bias problem, i. e. the iterative error propagation in iterative forecasting of RNN architectures. The method is benchmarked against standard BPTT and a schedule sampling approach in low-dimensional time series problems and the viscous flow past a cylinder in a channel. We demonstrate that BPTT-SA can successfully reduce the error in long-term high-dimensional spatio-temporal prediction in ConvRNNs and CNN-RNNs at the viscous flow past a cylinder, at no extra training cost.

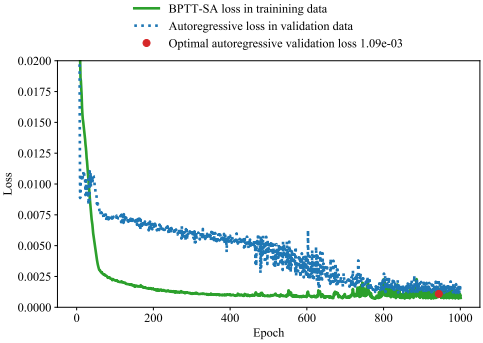
The BPTT-SA method can be applied to any recurrent architecture, without any additional training time or memory cost. BPTT-SA can be very helpful in applications where building models for long-term forecasting is desirable, and retraining different models for different lead times is computationally prohibitive. BPTT-SA can be used to improve long-term prediction of data-driven surrogate models, or ROMs of dynamical systems, Computational Fluid Dynamic (CFD), or Finite Element (FEM) codes. The method can also be used for surrogate modeling of the environment dynamics in model-based Reinforcement Learning. Moreover, the method can be utilized to fine-tune any recurrent architecture to achieve state-of-the-art results in long-term prediction in any relevant application, e. g. (Su et al., 2020). In low-dimensional time series, however, BPTT-SA does not improve long-term forecasting ability, according to the results presented in this chapter. An exciting avenue for future research is the application of BPTT-SA to climate (Rasp et al., 2020) and fluid flow (Wiewel et al., 2019) datasets.



(a) CNN-RNN trained with BPTT

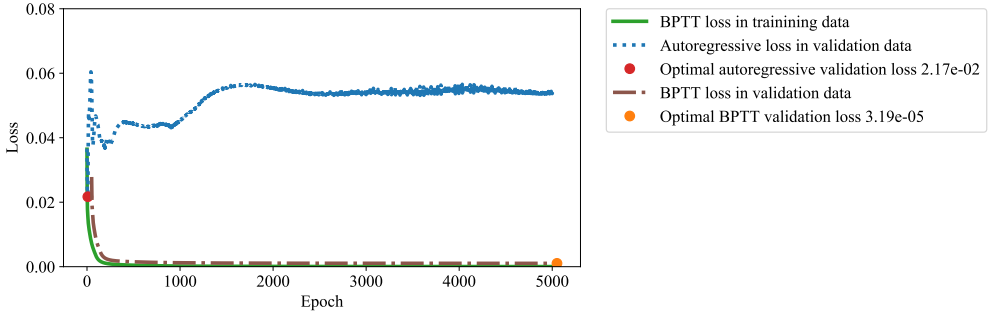


(b) CNN-RNN trained with BPTT-SS

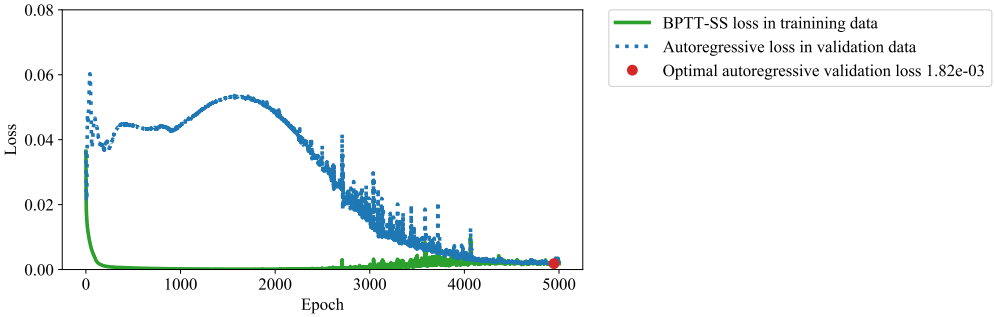


(c) CNN-RNN trained with BPTT-SA

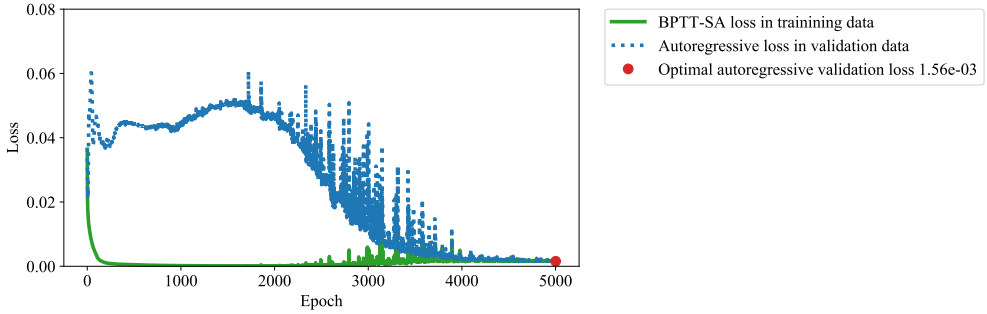
FIGURE 5.7: Evolution of the training and validation losses in CNN-RNN training in the viscous flow past a cylinder in a channel dataset.



(a) ConvRNN trained with BPTT



(b) ConvRNN trained with BPTT-SS



(c) ConvRNN trained with BPTT-SA

FIGURE 5.8: Evolution of the training and validation losses in ConvRNN training in the viscous flow past a cylinder in a channel dataset.

6

MULTISCALE SIMULATIONS OF COMPLEX SYSTEMS BY LEARNING THEIR EFFECTIVE DYNAMICS

6.1 RELATED WORK

In recent years, a number of multiscale methodologies have been proposed to tackle the prohibitive computational cost of resolving all scales in multiscale problems. Such methodologies are the Equation-Free Framework (EFF) (Bar-Sinai et al., 2019; Kevrekidis, Gear, and Hummer, 2004; Kevrekidis, Gear, Hyman, et al., 2003; Laing et al., 2010), the Heterogeneous Multiscale Method (HMM) (Weinan, Engquist, et al., 2003, 2007; Weinan, X. Li, et al., 2004), and the FLow AVeraged integatoR (FLAVOR) (M. Tao et al., 2010).

These algorithms try to exploit the multiscale character of the dynamics (Car et al., 1985; Erban et al., 2006; Kevrekidis, Gear, and Hummer, 2004; Kevrekidis, Gear, Hyman, et al., 2003; Kevrekidis and Samaey, 2009; Weinan, Engquist, et al., 2003), by distinguishing fine (expensive to simulate) and coarse scales (affordable). The EFF relies on few fine-scale simulations that are used to acquire, through "restricting", information about the evolution of the coarse-grained quantities of interest. The fine-scale dynamics are obtained by judiciously "lifting" the coarse scales to return to the fine-scale description of the system and repeat. In order to propagate the coarse-grained (latent) dynamics, EFF employs timesteppers (latent propagators). When the EFF reproduces trajectories of the original system, the identified low-order dynamics represent the intrinsic system dynamics, also called effective dynamics, inertial manifold (Linot et al., 2020; J. C. Robinson, 1994) or reaction coordinates in molecular kinetics. The goal of multiscale methods is to minimize the computational cost while maintaining the predictive accuracy of the propagated dynamics. Their success and scalability to high-dimensional systems have been hindered by the challenges of identifying accurate "lifting" operators to recover the high-dimensional micro-scale system dynamics from the reduced-order information of the coarse-grained state and the identification of accurate coarse-grained (latent) state propagators.

Close to our work presented in this chapter, in the context of the EFF, the authors in S. Lee et al., 2020 identify a PDE on a coarse representation by diffusion maps, Gaussian processes, or neural networks and utilize forward integration in the coarse representation. Such previous works, however, fail to employ one or more of the following mechanisms: consider the coarse-scale dynamics (Geneva et al., 2020; Milano et al., 2002), account their non-Markovian (Bhatia et al., 2021; S. Lee et al., 2020) or nonlinear nature (Lusch et al., 2018), exploit a probabilistic generative mapping (Geneva et al., 2020; Gonzalez et al., 2018; Hasegawa et al., 2020; Maulik et al., 2021) from the coarse to the fine-scale, learn simultaneously the latent space and its dynamics in an end-to-end fashion and not sequentially (Bhatia et al., 2021; Geneva et al., 2020; Gonzalez et al., 2018; Hasegawa et al., 2020; S. Lee et al., 2020; Lusch et al., 2018; Maulik et al., 2021), alternate between micro and macro dynamics (Geneva et al., 2020; Gonzalez et al., 2018; Hasegawa et al., 2020; S. Lee et al., 2020; Lusch et al., 2018; Maulik et al., 2021), and scale to high-dimensional systems (Gonzalez et al., 2018; S. Lee et al., 2020; Maulik et al., 2021).

We present a framework that resolves critical issues of previously proposed multiscale methodologies through ML algorithms that (i) deploy RNNs with gating mechanisms to evolve the coarse-grained dynamics and (ii) employ advanced (convolutional, or probabilistic) AEs to transfer in a systematic, data-driven manner, the information between coarse and fine-scale descriptions. The framework learns the effective reduced-order dynamics (LED) and is orders of magnitude faster than the micro-scale solvers while maintaining predictive accuracy.

Augmenting multiscale frameworks (including EFF, HMM, FLAVOR) with state-of-the-art ML algorithms allows for evolving the coarse-scale dynamics by taking into account their time history and by providing consistent lifting (decoding) and restriction (encoding) operators to transfer information between fine and coarse scales. We demonstrate that the proposed framework allows for simulations of complex multiscale systems that reduce the computational cost by orders of magnitude to capture spatio-temporal scales that would be impossible to resolve with existing computing resources.

This chapter is based on the paper "Multiscale Simulations of Complex Systems by Learning their Effective Dynamics" (P. R. Vlachas, Arampatzis, et al., 2022). The computational resources were provided by a grant from the Swiss National Supercomputing Centre (CSCS) under project s929.

6.2 METHODS

We propose a framework for learning the effective dynamics of complex systems that allows for accurate prediction of the system evolution at a significantly reduced computational cost.

In the following, the high-dimensional state of a dynamical system is given by $\boldsymbol{s}_t \in \mathbb{R}^{d_s}$, and the discrete time dynamics are given by

$$\boldsymbol{s}_{t+\Delta t} = \boldsymbol{F}(\boldsymbol{s}_t),$$

where Δt is the sampling period and $\boldsymbol{F}$ may be nonlinear, deterministic or stochastic. We assume that the state of the system at time t can be described by a vector $\boldsymbol{z}_t \in \mathcal{Z}$, where $\mathcal{Z} \subset \mathbb{R}^{d_z}$ is a low dimension manifold with $d_z \ll d_s$. In order to identify this manifold, an encoder $\mathcal{E}^{\boldsymbol{w}_\mathcal{E}} : \mathbb{R}^{d_s} \to \mathbb{R}^{d_z}$ is utilized, where $\boldsymbol{w}_\mathcal{E}$ are trainable parameters, transforming the high-dimensional state $\boldsymbol{s}_t$ to $\boldsymbol{z}_t = \mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{s}_t)$. In turn, a decoder maps back this latent representation to the high-dimensional state, i. e. $\widetilde{\boldsymbol{s}}_t = \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}(\boldsymbol{z}_t)$.

For deterministic systems, the optimal parameters $\{\boldsymbol{w}^\star_\mathcal{E}, \boldsymbol{w}^\star_\mathcal{D}\}$ are identified by minimizing the mean squared reconstruction error (MSE),

$$\boldsymbol{w}^\star_\mathcal{E}, \boldsymbol{w}^\star_\mathcal{D} = \underset{\boldsymbol{w}_\mathcal{E}, \boldsymbol{w}_\mathcal{D}}{\arg\min} \left\langle (\boldsymbol{s}_t - \widetilde{\boldsymbol{s}}_t)^2 \right\rangle = \underset{\boldsymbol{w}_\mathcal{E}, \boldsymbol{w}_\mathcal{D}}{\arg\min} \left\langle \left(\boldsymbol{s}_t - \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}\left(\mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{s}_t)\right)\right)^2 \right\rangle,$$

where $\langle \cdot \rangle$ denotes the mean. Convolutional neural network (LeCun et al., 2015) autoencoders (CNN-AE) that take advantage of the spatial structure of the data are embedded into LED.

For stochastic systems, $\mathcal{D}^{\boldsymbol{w}_\mathcal{D}}$ is modeled with a Mixture Density Network (MDN) decoder (Bishop, 1994). Further details on the implementation of the MDN decoder are provided in Chapter 2, in Section 2.1.5, along with other components embedded in LED like AEs in Section 2.1.1, Variational AEs in Section 2.1.2, and CNNs in Section 2.1.3.

As a nonlinear propagator in the low-order manifold (coarse scale), an RNN is employed, capturing non-Markovian, memory effects by keeping an internal memory state. The RNN is learning a forecasting rule

$$\boldsymbol{h}_t = \mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}\left(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t}\right), \quad \widetilde{\boldsymbol{z}}_{t+\Delta t} = \mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}\left(\boldsymbol{h}_t\right),$$

where $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is an internal hidden memory state, $\widetilde{\boldsymbol{z}}_{t+\Delta t}$ is a latent state prediction, $\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}$ and $\mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}$ are the hidden-to-hidden, and the hidden-to-output mappings, and $\boldsymbol{w}_{\mathcal{F}_{hh}}$, $\boldsymbol{w}_{\mathcal{F}_{ho}}$ are the trainable parameters of the RNN.

One possible implementation of $\mathcal{F}_{hh}^{\boldsymbol{w}_{\mathcal{F}_{hh}}}$ and $\mathcal{F}_{ho}^{\boldsymbol{w}_{\mathcal{F}_{ho}}}$ is the LSTM (Hochreiter and Schmidhuber, 1997), presented in Section 2.1.4.2.

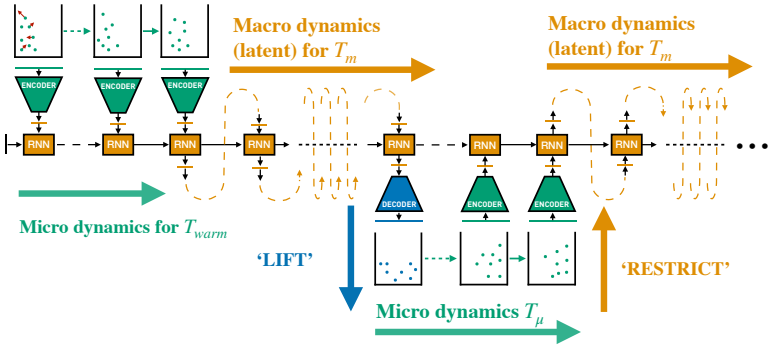


FIGURE 6.1: Multiscale-LED: Starting from an initial condition, use the equations/first principles to evolve the high-dimensional dynamics for a short period T_{warm}. During this warm-up period, the state $\boldsymbol{s}_t$ is passed through the encoder network. The outputs of the autoencoder are iteratively provided as inputs to the RNN, to warm-up its hidden state. Next, iteratively, (1) starting from the last latent state $\boldsymbol{z}_t$ the RNN propagates the latent dynamics for $T_m \gg T_{warm}$, (2) lift the latent dynamics at $t = T_{warm} + T_m$ back to the high-dimensional state, (3) starting from this high-dimensional state as an initial condition, use the equations/first principles to evolve the dynamics for $T_\mu \ll T_m$.

The role of the RNN is twofold. First, it is updating its hidden memory state $\boldsymbol{h}_t$, given the current state provided at the input $\boldsymbol{z}_t$ and the hidden memory state at the previous timestep $\boldsymbol{h}_{t-\Delta t}$, tracking the history of the low-order state to model non-Markovian dynamics. Second, given the updated hidden state $\boldsymbol{h}_t$ the RNN forecasts the latent state at the next timestep(s) $\widetilde{\boldsymbol{z}}_{t+\Delta t}$. The RNN is trained to minimize the forecasting loss $||\widetilde{\boldsymbol{z}}_{t+\Delta t} - \boldsymbol{z}_{t+\Delta t}||_2^2$ by BPTT (P. J. Werbos, 1988).

The LSTM and the AE, jointly referred to as LED, are trained on data from simulations of the fully resolved (or micro-scale) dynamical system. The two networks can either be trained sequentially or together. In the first case, the AE is pre-trained to minimize the reconstruction loss, and then the LSTM is trained to minimize the prediction loss on the latent space (AE-LSTM). In the second case, they are seen as one network trying to minimize the sum of reconstruction and prediction losses (AE-LSTM-end2end). For large, high-dimensional systems, the later approach of end-to-end training

is computationally expensive. After training, LED is employed to forecast the dynamics on unseen data by propagating the low-order latent state with the RNN and avoiding the computationally expensive simulation of high-dimensional dynamics. We refer to this mode of propagation, iteratively propagating only the latent/macro-dynamics, as Latent-LED. As non-Markovian models are not self-starting, we note that an initial small warm-up period is required, feeding the LED with data from the micro dynamics.

The LED framework allows for data-driven information transfer between coarse and fine scales through the AE. Moreover, it propagates the latent space dynamics without the need to upscale back to the high-dimensional state-space at every timestep. As is the case for any approximate iterative integrator (here is the RNN), the initial model errors will propagate. In order to mitigate potential instabilities, inspired by the Equation-Free (Kevrekidis, Gear, Hyman, et al., 2003), we propose the multiscale forecasting scheme in Figure 6.1, alternating between micro dynamics for T_μ and macro-dynamics for T_m. In this way, the approximation error can be reduced at the cost of the computational complexity associated with evolving the high-dimensional dynamics. We refer to this mode of propagation as Multiscale-LED, and the ratio $\rho = T_m/T_\mu$ as multiscale ratio. In Multiscale-LED, the interface with the high-dimensional state-space is enabled only at the timesteps and scales of interest. This is in contrast to Hernández et al., 2018; Sultan et al., 2018, and is easily adaptable to the needs of particular applications, thus augmenting the arsenal of models developed for multiscale problems.

Training of LED models is performed with the Adam stochastic optimization method (Kingma et al., 2014), and validation-based early stopping is employed to avoid overfitting. All LED models are implemented in PYTHON (Van Rossum et al., 1995) in the PYTORCH (Paszke et al., 2019) library, mapped to a single Nvidia Tesla P100 GPU, and executed on the XC50 compute nodes of the Piz Daint supercomputer at the Swiss national supercomputing centre (CSCS). LED supports data-parallelism by employing the HOROVOD (Sergeev et al., 2018) library to parallelize the data of a batch among multiple GPUs.

6.3 COMPARISON MEASURES

In this section, we elaborate on the metrics used to quantify the effectiveness of the proposed approach to capture the dynamics and the state statistics of the systems under study. The Mean Normalized Absolute Difference (MNAD) is used to quantify the prediction performance of a method in a deterministic system. This metric was selected to facilitate comparison of LED with Equation-Free variants (S. Lee et al., 2020).

6.3.1 *Mean Normalised Absolute Difference*

Assume that a model is used to predict a spatio-temporal field $\boldsymbol{o}(x,t)$, at discrete state x_i and time t_j locations. Predicted values from a model (neural network, etc.) are denoted with $\widetilde{\boldsymbol{o}}$, while the ground-truth (simulation of the equations with a solver based on first principles) with $\boldsymbol{o}$. The normalized absolute difference (NAD) between the model output and the ground-truth is defined as

$$\mathrm{NAD}(t_j) = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{|\boldsymbol{o}(x_i,t_j)-\hat{\boldsymbol{o}}(x_i,t_j)|}{\max_{i,j}(\boldsymbol{o}(x_i,t_j))-\min_{i,j}(\boldsymbol{o}(x_i,t_j))}, \tag{6.1}$$

where N_x is the dimensionality of the discretized state x. The NAD depends on the time t_j. The mean NAD (MNAD) is given by the mean over time of the NAD score, i.e.

$$\mathrm{MNAD} = \frac{1}{N_T}\sum_{j=1}^{N_T}\mathrm{NAD}(t_j), \tag{6.2}$$

where N_T is the number of timesteps considered. The MNAD is used in the FitzHugh-Nagumo model, and the Kuramoto-Sivashinsky equation, to quantify the prediction accuracy of LED and benchmark against other methods (e. g. other propagators on the latent space) or against other Equation-Free variants.

6.3.2 *Pearson Correlation Coefficient*

Assume as before, the spatio-temporal field $\boldsymbol{o}(x,t)$, at discrete state x_i and time t_j locations. This can be vectorized in $\boldsymbol{o}_{vec} = \mathrm{vec}(\boldsymbol{o}(x,t)) \in \mathbb{R}^{N_x\cdot N_t\times 1}$. The same applies to the vectorized prediction $\widetilde{\boldsymbol{o}}_{vec} = \mathrm{vec}(\widetilde{\boldsymbol{o}}(x,t)) \in \mathbb{R}^{N_x\cdot N_t\times 1}$.

We can compute the Pearson correlation coefficient, or simply correlation, as

$$\text{Correlation} = \frac{\text{COV}\left(\boldsymbol{o}vec, \widetilde{\boldsymbol{o}}_{vec}\right)}{\sigma(\boldsymbol{o}_{vec})\,\sigma(\widetilde{\boldsymbol{o}}_{vec})}, \tag{6.3}$$

where COV is the covariance, and σ is the standard deviation.

The correlation is used as a prediction performance metric in the Kuramoto-Sivashinsky equation.

6.4 RESULTS

We demonstrate the application of LED in several benchmark problems and compare its performance with existing state-of-the-art algorithms.

6.4.1 *FitzHugh-Nagumo Model*

LED is employed to capture the dynamics of the FitzHugh-Nagumo model (FHN) (FitzHugh, 1961; Nagumo et al., 1962). The FHN model describes the evolution of an activator $u(x,t) = \rho^{ac}(x,t)$ and an inhibitor density $v(x,t) = \rho^{in}(x,t)$ on the domain $x \in [0, L]$:

$$\begin{aligned} \frac{\partial u}{\partial t} &= D^u \frac{\partial^2 u}{\partial x^2} + u - u^3 - v, \\ \frac{\partial v}{\partial t} &= D^v \frac{\partial^2 v}{\partial x^2} + \varepsilon(u - \alpha_1 v - \alpha_0). \end{aligned} \tag{6.4}$$

The system evolves periodically under two timescales, with the activator/inhibitor density acting as the "fast"/"slow" variable, respectively. The bifurcation parameter $\varepsilon = 0.006$ controls the difference in the timescales. We choose $D^u = 1$, $D^v = 4$, $L = 20$, $\alpha_0 = -0.03$ and $\alpha_1 = 2$.

Equation 6.4 is discretized with $N = 101$ grid points and solved using the Lattice Boltzmann (LB) method (Karlin et al., 2006), with timestep $\delta t = 0.005$. To facilitate comparison with S. Lee et al., 2020, we employ the LB method to gather data starting from 6 different initial conditions to obtain the mesoscopic solution considered here as the fine-grained solution. The data is sub-sampled, keeping every 200^{th} data points, i. e. the coarse timestep is $\Delta t = 1$. Three time series with 451 points are considered for training, two time series with 451 points for validation, and 10^4 data points from a different initial condition for testing. For the identification of the latent

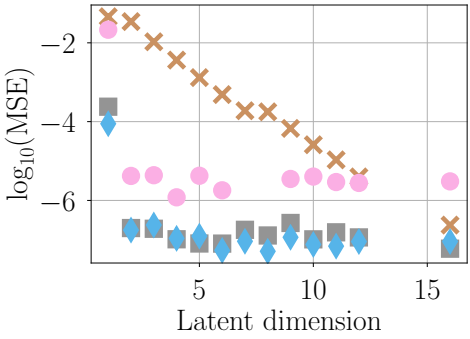


FIGURE 6.2: Comparison of the reconstruction MSE in the test data in the FHN between PCA (×), Diffusion Maps (●), Autoencoder (■), and Convolutional Autoencoder (◆).

space, we compare PCA, diffusion maps (DiffMaps), feedforward AE, and CNN-AE, in terms of the mean squared error (MSE) of the reconstruction in the test data, plotted in Figure 6.2. The MSE is plateauing after $d_z = 2$, and the AE and CNN-AE exhibit at least an order of magnitude lower MSR compared to PCA and DiffMaps. For this reason, we employ an AE with $d_z = 2$ for the LED. The hyperparameters of the networks (reported in Table D.1) are tuned based on the MSE on the validation data. The architecture of the CNN is reported in Table D.3, and depicted in Figure 6.3. The inhibitor and activator density are considered two channels of the CNN.

In Figure 6.4, we compare various propagators in the forecasting of the macro (latent dynamics), starting from 32 different initial conditions in the test data, up to a horizon of $T_f = 8000$. We benchmark an AE-LSTM trained end-to-end (AE-LSTM-end2end), an AE-LSTM where the AE is pre-trained (AE-LSTM), a fully connected feedforward neural network, i. e. multi-layered perceptron (AE-MLP), Reservoir Computing (AE-RC) (Pathak, Hunt, et al., 2018; P. R. Vlachas, Pathak, et al., 2020), and the sparse identification of nonlinear dynamics (SINDy) (Brunton, Proctor, et al., 2016) algorithm (AE-SINDy). As a comparison metric, we consider the mean normalized absolute difference (MNAD), averaged over the 32 initial conditions. The MNAD is computed on the inhibitor and the activator density, as the difference between the result of the LB simulation $v(x, t)$, considered

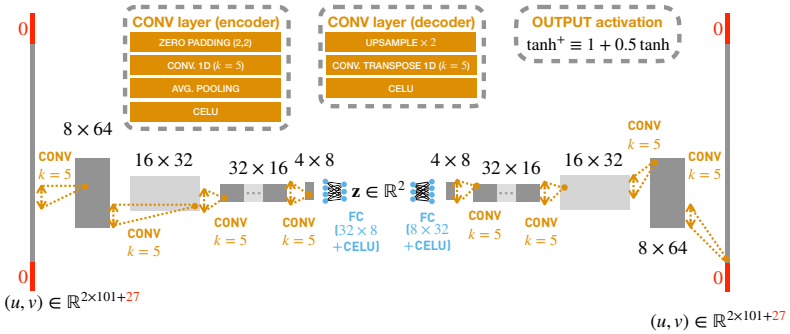
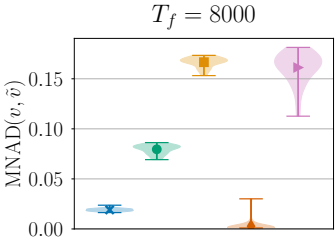


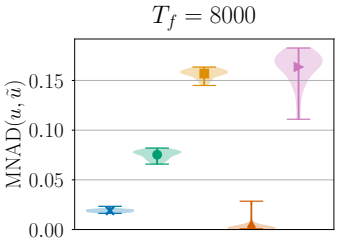
FIGURE 6.3: The architecture of the CNN employed in the FHN equation. First, the input is padded to the closest power of two. Then, four layers of consecutive application of 1D convolutions, average pooling, CELU activations functions, and dropout are used. Then an MLP is utilized to project to the low-order latent space. The output activation of the MLP is also $1 + 0.5 \tanh(\cdot)$.

as ground-truth, and the model forecasts $\widetilde{v}$. The warm-up period for all propagators is set to $T_{warm} = 60$. The hyperparameters of the networks (reported in Tables D.2, D.4 and D.5, along with the training times) are tuned based on the MNAD on the validation data. The LSTM-end2end and the RC show the lowest test error, while the variance of the RC is higher. In the following, we consider an LSTM-end2end propagator for the LED.

LED is benchmarked against EFF variants (S. Lee et al., 2020) in the FHN equation in Figure 6.5. As a metric for the accuracy, the MNAD is considered consistent with S. Lee et al., 2020 to facilitate comparison. The EFF variants (S. Lee et al., 2020) are based on the identification of PDEs on the coarse level (CSPDE). LED is compared with CSPDEs in forecasting the dynamics of the FHN equation starting from an initial condition from the test data up to the final time $T_f = 451$. CSPDE variants utilize Gaussian processes (GP) or neural networks, features of the fine-scale dynamics obtained through diffusion maps (F1 to F3), and forward integration to propagate the coarse representation in time. LED outperforms CSPDE variants by an order of magnitude. In Figure 6.7, the latent space of LED is plotted against the attractor of the data embedded in the latent space. Even for long time horizons (here $T_f = 8000$), the LED forecasts stay on the periodic attractor.

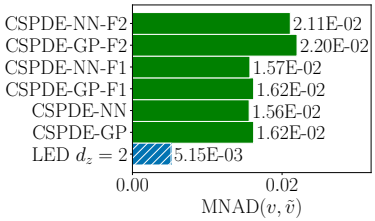


(a) MNAD on the inhibitor density

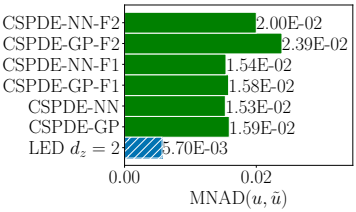


(b) MNAD on the activator density

FIGURE 6.4: Comparison of macrodynamics propagators (AE-LSTM-end2end; AE-LSTM; AE-MLP; AE-RC; AE-SINDy) in iterative latent forecasting. The density and mean MNAD (averaged over 32 initial conditions) between the predicted and ground-truth evolution of the inhibitor density is plotted.

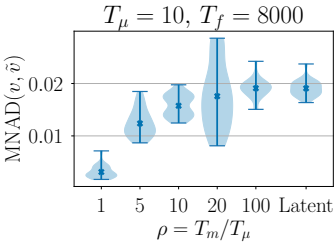


(a) MNAD on the inhibitor density

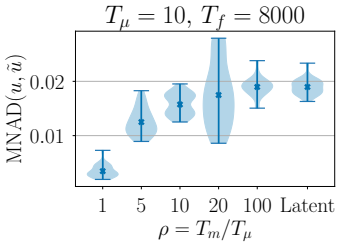


(b) MNAD on the activator density

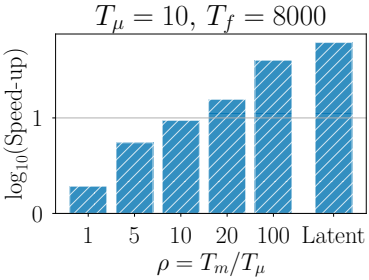
FIGURE 6.5: Comparison of Latent-LED with $d_z = 2$ with Equation-Free variants from S. Lee et al., 2020 in forecasting the dynamics of FHN starting from one initial condition from the testing dataset.



(a) The inhibitor MNAD of Multiscale-LED (AE-LSTM-end2end, $d_z = 2$) plotted as a function of the multiscale ratio $\rho = T_m/T_\mu$.



(b) The activator MNAD of Multiscale-LED (AE-LSTM-end2end, $d_z = 2$) plotted as a function of the multiscale ratio $\rho = T_m/T_\mu$.



(c) The speed-up of Multiscale-LED compared to the LB solver plotted w. r. t. ρ. As T_m is increased (increase ρ), the speed-up is increased, albeit at the cost of an increasing MNAD.

FIGURE 6.6: Results of LED applied on the FHN model.

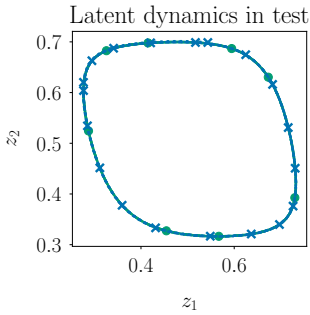


FIGURE 6.7: The LED latent state (▪▪▪✕▪▪▪) compared against the attractor of the data embedded in the latent space (▬●▬).

Latent-LED propagates the low-order dynamics and up-scales back to the inhibitor density, forecasting its evolution accurately while being 60 times faster than the LB solver. This speed-up can be decisive in accelerating simulations and achieving much larger time horizons.

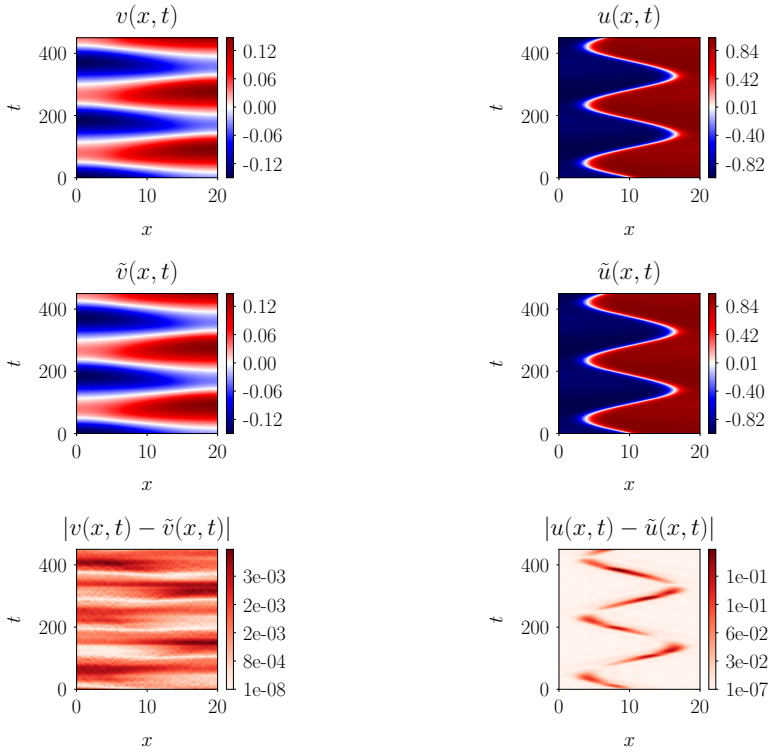
In Multiscale-LED, the approximation error of LED decreases at the cost of reduced speed-up. This interplay can be seen in Figure 6.6.

Latent-LED ($T_\mu = 0$), and Multiscale-LED, alternating between macro-dynamics for $T_m = 10$ and high-dimensional dynamics for T_μ, are employed to approximate the evolution and compare it against the LB solver in forecasting up to $T_f = 8000$ starting from 32 initial conditions as before. The warm-up period is set to $T_{warm} = 60$. For $T_m = T_\mu = 10$ ($\rho = 1$), the MNAD is reduced from approximately 0.019, to approximately 0.003 compared to Latent-LED. The speed-up, however, is reduced from 60 to 2. By varying $T_m \in \{50, 100, 200, 1000\}$, Multiscale-LED achieves a trade-off between speed-up and MNAD.

A prediction of the Latent-LED in the inhibitor and the activator density is compared against the ground-truth in Figure 6.8.

6.4.2 *The Kuramoto-Sivashinsky Equation*

Here, we apply LED on the KS equation (Equation 2.15) on the domain $\Omega = [0, L]$ with periodic boundary conditions $u(0, t) = u(L, t)$ and $\nu = 1$. In the case of high dissipation and small spatial extent L (domain size),



(a) Inhibitor density.

(b) Activator density.

FIGURE 6.8: A trajectory starting from a testing initial condition (top), along with the Latent-LED prediction (middle), and absolute difference (bottom).

the long-term dynamics of KS can be represented on a low-dimensional inertial manifold (Linot et al., 2020; J. C. Robinson, 1994), that attracts all neighboring states at an exponential rate after a transient period. LED is employed to learn the low-order manifold of the effective dynamics in KS. The special case $L = 22$ considered here is studied extensively in Cvitanović et al., 2010 and exhibits a structurally stable chaotic attractor, i. e. an inertial manifold where the long-term dynamics lie. Equation 2.15 is discretized to Equation 2.16, considering a grid of size 64 points, and solved using the fourth-order method for stiff PDEs introduced in Kassam et al., 2005 with a timestep of $\delta t = 2.5 \cdot 10^{-3}$ starting from a random initial condition. The data are subsampled to $\Delta t = 0.25$ (coarse timestep of the LED). $15 \cdot 10^3$ samples are used for training, and another $15 \cdot 10^3$ for validation. For testing purposes, the process is repeated with a different random seed, generating another $15 \cdot 10^3$ samples.

For the identification of a reasonable latent space dimension, we compare PCA, AEs, and CNNs in terms of the reconstruction MSE in the test data as a function of d_z, plotted in Figure 6.10(a). MSE is plateauing after $d_z = 8$, indicating arguably the dimensionality of the attractor in agreement with previous studies (Cvitanović et al., 2010; Linot et al., 2020), and that the CNN is superior to the AE, while orders of magnitude better than PCA. For this reason, we employ a CNN with $d_z = 8$ for the autoencoding part of the LED. The hyperparameters of the networks are tuned based on the MSE on the validation data, reported in Tables D.6 and D.7, with the network training times. The CNN architecture is illustrated in Figure 6.9 and described in detail in Table D.8.

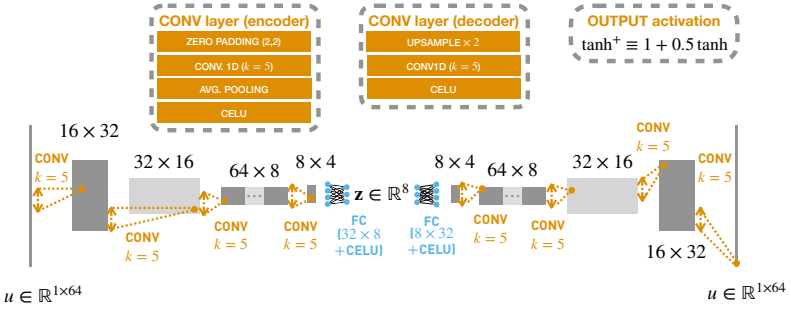
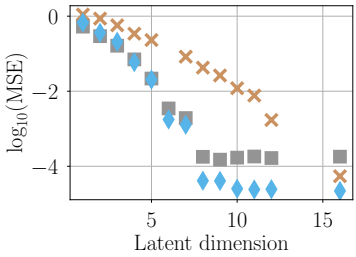
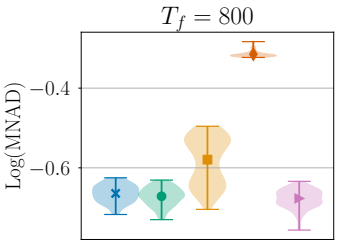


FIGURE 6.9: The architecture of the CNN employed in KS.



(a) Comparison of the reconstruction mean squared error (MSE) in the test data in the FHN dynamics between PCA (×), Autoencoder (■), and Convolutional Autoencoder (◆) as a function of the latent dimension.

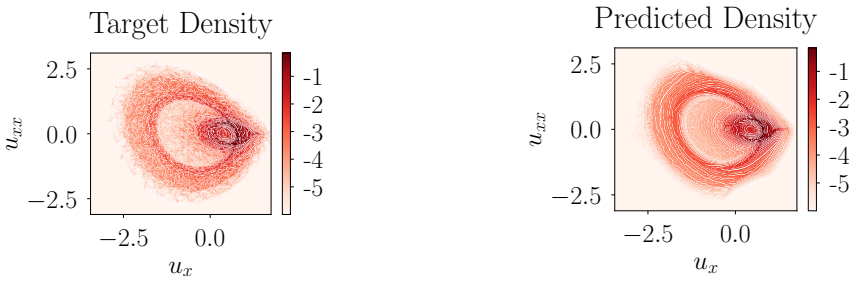


(b) Comparison of different macrodynamics propagators (✱ AE-LSTM-end2end; ⬮ AE-LSTM; ▮ AE-MLP; ◆ AE-RC; ▶ AE-SINDy) in iterative latent forecasting.

FIGURE 6.10

In Figure 6.10(b), we compare various propagators in predicting the macro dynamics of LED, starting from 100 test initial conditions, up to $T_f = 800$ (3200 timesteps). We employ a CNN-LSTM trained end-to-end (CNN-LSTM-end2end), a CNN-LSTM where the CNN is pre-trained (CNN-LSTM), a multi-layered perceptron (CNN-MLP), Reservoir Computing (CNN-RC) (Pathak, Hunt, et al., 2018; P. R. Vlachas, Pathak, et al., 2020), and the SINDy algorithm (CNN-SINDy) (Brunton, Proctor, et al., 2016). As a comparison metric, we consider the MNAD, averaged over the 100 initial conditions. The warm-up period for all propagators is set to $T_{warm} = 60$. The hyperparameters (reported on Tables D.9 to D.11 along with the training times) are tuned based on the MNAD on the validation data. While the MLP and RC propagators exhibit large errors, the LSTM, LSTM-end2end, and SINDy show comparable accuracy. In the following, we consider an LSTM propagator for the LED.

Due to the chaoticity of the KS equation, iterative forecasting with LED is challenging, as initial errors propagate exponentially. In order to assess whether the iterative forecasting with LED leads to reasonable, physical predictions, we plot the density of values in the $u_x - u_{xx}$ space in Figure 6.11(a). The data come from a single long trajectory of size $T_f = 8000$ (32000 timesteps). We observe that LED, Figure 6.11(b), is able to qualitatively reproduce the density of the simulation.



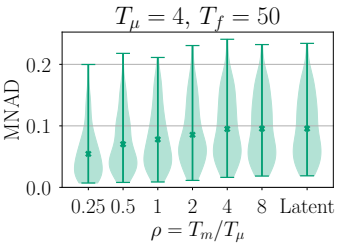
(a) Reference (b) Prediction

FIGURE 6.11: The density of values in the $u_x - u_{xx}$ space computed from a single long trajectory of size $T_f = 8000$ (32000 timesteps) that matches the prediction closely, illustrating that the LED is able to replicate characteristics of the dynamical system and remain at the attractor, even though propagating coarse dynamics.

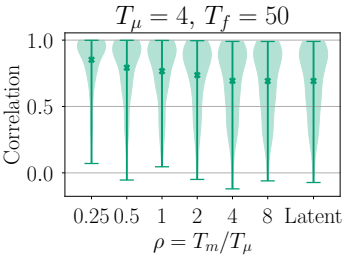
In Figures 6.12(a) and 6.12(b) we plot the MNAD and correlation between forecasts of LED and the reference with respect to the multiscale ratio ρ. In Figure 6.12(c) the speed-up of LED is plotted against ρ. Latent-LED is able to reproduce the long-term "climate dynamics" (P. R. Vlachas, Pathak, et al., 2020), and remain at the attractor, while being more than two orders of magnitude faster compared to the micro solver. As ρ is increased, the error is reduced (correlation increased), at the cost of reduced speed-up.

Finally, in Figure 6.13, we compare the performance of Latent-LED (CNN-LSTM) with previous studies (Pathak, Hunt, et al., 2018; P. R. Vlachas, Pathak, et al., 2020), that forecast directly on the high-dimensional space. Specifically, the Latent-LED matches the performance of an LSTM (no dimensionality reduction) but shows inferior short-term forecasting ability compared to an RC (no dimensionality reduction) forecasting on the high-dimensional space. This is expected as the RC, and the LSTM have full information about the state. In turn, when the RC is employed on the latent space of LED as a macro-dynamics propagator, the error grows significantly, and the performance is inferior to the CNN-LSTM case.

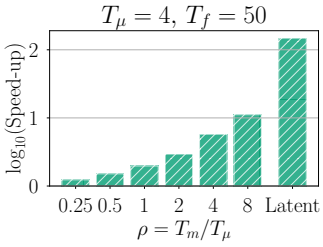
A KS trajectory is plotted in Figure 6.14, along with the latent space evolution of Latent-LED and the predicted trajectory. We observe that the long-term climate is reproduced, although the LED is propagating an 8-dimensional latent state.



(a) The MNAD of Multiscale-LED is plotted as a function of the multiscale ratio $\rho = T_m/T_\mu$.



(b) Correlation between Multiscale-LED predictions and the reference data plotted as a function of the multiscale ratio $\rho = T_m/T_\mu$.



(c) The speed-up of Multiscale-LED w. r. t. ρ. Evolution of the latent state of LED ($T_\mu = 0$) is up to two orders of magnitude cheaper than the micro-scale dynamics.

FIGURE 6.12: Results of Multiscale-LED applied on the KS equation.

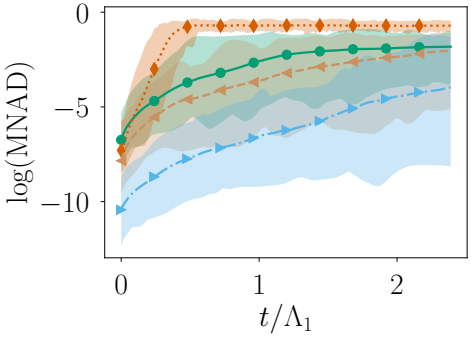


FIGURE 6.13: Comparison of CNN-LSTM (LED) (—●—), CNN-RC (···♦···), LSTM (—◀··), and RC (—▶··) in short-term forecasting of the KS dynamics, time is normalized with the Lyapunov time $T^{\Lambda_1} = 1/\Lambda_1 = 20.83$.

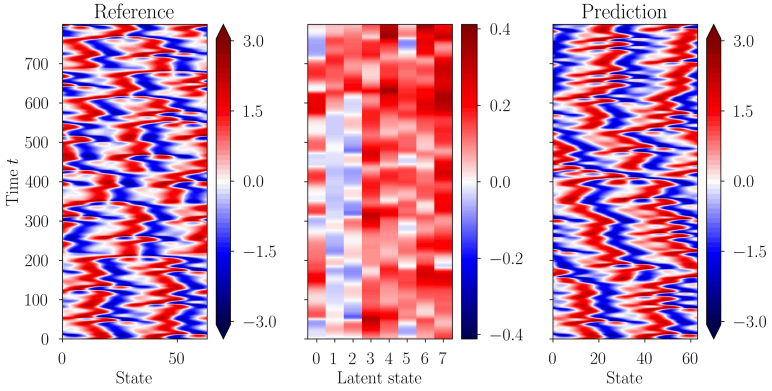


FIGURE 6.14: Contour plot of the KS dynamics starting from an initial condition from the test data (left). The evolution of the $d_z = 8$ dimensional latent state of Latent-LED (middle). The predicted field by Latent-LED iteratively propagating the dynamics on a $d_z = 8$ dimensional latent space, after a warm-up period $T_{warm} = 60$ ($T_\mu = 0$) (right).

6.4.3 *Viscous Flow Past a Cylinder*

The flow past a cylinder is a widely studied problem in fluids (Zdravkovich, 1997), that exhibits a rich range of dynamical phenomena like the transition from laminar to turbulent flow in high Reynolds numbers and is used as a benchmark for reduced-order modeling (ROM) approaches. The flow past a cylinder in the two-dimensional space is simulated by solving the incompressible Navier-Stokes equations with Brinkman penalization to enforce the no-slip boundary conditions on the surface of the cylinder Bost et al., 2010; Rossinelli et al., 2015, i. e.

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} &= -\frac{\nabla p}{\widetilde{\rho}} + \nu \Delta \boldsymbol{u} + \lambda \chi^{(s)}(\boldsymbol{u}^{(s)} - \boldsymbol{u}) \,, \\ \nabla \cdot \boldsymbol{u} &= 0 \,, \end{aligned} \tag{6.5}$$

where $\boldsymbol{u} = [u_x, u_y]^T \in \mathbb{R}^2$ is the velocity, $p \in \mathbb{R}$ is the pressure field, $\widetilde{\rho}$ is the density, ν is the kinematic viscocity, and λ is the penalization coefficient. The velocity-field $\boldsymbol{u}^{(s)} \in \mathbb{R}^2$ describes the translation of the cylinder. The numerical method of the flow solver is finite differences, with the incompressibility enforced through pressure projection. The solver is implemented in C++. The computational domain is $\Omega = [0, 1] \times [0, 0.5]$, the cylinder is positioned at $(0.2, 0.5) \in \Omega$, with diameter $D_{cyl} = 0.075$. The cylinder is described by the characteristic function $\chi^{(s)}$, that is $\chi^{(s)} = 1$ inside the cylinder $\Omega^{(s)}$ and $\chi^{(s)} = 0$ outside $\Omega \setminus \Omega^{(s)}$. The domain is discretized using 1024×512 grid-points, and the timestep δt is adapted to ensure that the Courant–Friedrichs–Lewy (CFL) number is fixed at 0.5.

Equation 6.5 is solved for the velocity $\boldsymbol{u} \in \mathbb{R}^2$ and pressure field $p \in \mathbb{R}$ using the pressure projection method. First, we perform advection and diffusion of the flow field in the whole domain

$$\boldsymbol{u}^* = \boldsymbol{u}^t + \delta t \left(\nu \Delta \boldsymbol{u}^t - (\boldsymbol{u}^t \cdot \nabla)\boldsymbol{u}^t \right) . \tag{6.6}$$

The continuity equation requires the field to be divergence-free. This condition is imposed with the pressure projection

$$\boldsymbol{u}^{**} = \boldsymbol{u}^* - \delta t \frac{\nabla p^{t+1}}{\widetilde{\rho}} . \tag{6.7}$$

The pressure field used here is obtained by solving the Poisson equation emerging from the divergence of Equation 6.7, i.e.

$$\Delta p^{t+1} = \frac{\widetilde{\rho}}{\delta t} \nabla \cdot \boldsymbol{u}^* . \tag{6.8}$$

Note that adding Equation 6.7 and Equation 6.6 yields the original Equation 6.5 without the penalization term for Euler timestepping. The timestep is completed by applying the penalization force using $\delta t \lambda = 1$,

$$\boldsymbol{u}^{t+1} = \boldsymbol{u}^{**} + \chi^{(s),t+1}(\boldsymbol{u}^{(s),t+1} - \boldsymbol{u}^{**}) \,. \tag{6.9}$$

We remark that the penalization force acts as a Lagrange multiplier enforcing the translation motion of the cylinder on the fluid. The temporally discrete equations described above are solved on a grid with spacing Δx using second-order central finite differences for diffusion terms and a third-order upwind scheme for advection terms.

For the simulated impulsively started cylinder the Reynolds number for a cylinder with diameter D_{cyl} moving with velocity v_{cyl} in a fluid with kinematic viscosity ν is defined as

$$\mathrm{Re} = \frac{D_{cyl} v_{cyl}}{\nu}. \tag{6.10}$$

In the present simulations the cylinder moves with constant velocity $v_{cyl} = 0.15$ in $-x$-direction. The computational domain is chosen to be $\Omega = [0,1] \times [0,0.5]$ and moves with the center of mass of the sphere with diameter $D_{cyl} = 0.075$, that is fixed at $(0.2, 0.5) \in \Omega$. We consider the application of LED to two Reynolds numbers $\mathrm{Re} = 100$ and $\mathrm{Re} = 1000$, by setting $\nu = 0.0001125$ and $\nu = 0.00001125$ respectively. For both cases, the domain is discretized using 1024×512 grid-points, and the timestep δt is adapted to ensure that the CFL-number is fixed at 0.5.

The Strouhal number St describes the periodic vortex shedding at the wake of the cylinder. It is defined as

$$\mathrm{St} = \frac{D_{cyl} f_{vs}}{\nu}. \tag{6.11}$$

where f_{vs} is the frequency of vortex shedding. In our case, $\mathrm{St} = 0.175$ for $\mathrm{Re} = 100$, and $\mathrm{St} = 0.225$ for $\mathrm{Re} = 1000$.

The state of the simulation is described by the velocity $\boldsymbol{u} \in \mathbb{R}^2$ and the pressure $p \in \mathbb{R}$ at each grid point. The drag coefficient (C_d) around the cylinder for the viscosity μ and pressure p is calculated as

$$\mathbf{F}_\mu = \oiint \mu(\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^\intercal) \cdot \mathbf{n} \, dS, \tag{6.12}$$

$$\mathbf{F}_p = \oiint -p\mathbf{n} \, dS, \tag{6.13}$$

$$C_{d,\mu} = \frac{2 \cdot \mathbf{F}_\mu \cdot \boldsymbol{u}_\infty}{\varrho \cdot \|\boldsymbol{u}_\infty\|^3 \cdot D_{cyl}}, \tag{6.14}$$

$$C_{d,p} = \frac{2 \cdot \mathbf{F}_p \cdot \boldsymbol{u}_\infty}{\varrho \cdot \|\boldsymbol{u}_\infty\|^3 \cdot D_{cyl}}, \tag{6.15}$$

$$C_d = C_{d,\mu} + C_{d,p}, \tag{6.16}$$

where $\boldsymbol{u}_\infty = (1, 0)^\intercal$ is the free-stream velocity and $\mathbf{n}$ is the outward normal of the cylinder perimeter.

The state of the LED at every timestep is composed of four fields, the two components of the velocity field u_x, and u_y, the scalar pressure p at each grid-point, and the vorticity field ω computed a-posteriori from the velocity field, i. e. $\mathbf{s}_t = \{u_x, u_y, p, \omega\} \in \mathbb{R}^{4\times512\times1024}$. The simulation state $\mathbf{s}_t$ is saved at a coarse time resolution $\Delta t = 0.2$ for a total of 1000 coarse timesteps. There are 512 grid points along the length of the channel and 1024 gird points along the width of the channel. The flow is simulated in a cluster with 12 CPU Cores, up to $T = 200$, after discarding the initial transient. 250 timesteps distanced $\Delta t = 0.2$ in time (total time $T = 50$) are used for training, 250 for validation, and the rest for testing purposes. The vortex sheeding period is $T \approx 2.86$ for $\mathrm{Re} = 100$, and $T \approx 2.22$ for $\mathrm{Re} = 1000$.

For the autoencoding part, LED employs CNNs that take advantage of the spatial correlations. The architecture of the CNN is illustrated in Figure 6.15 and explained in detail in Table D.12. The dimension of the latent space is tuned based on the performance on the validation dataset to $d_z = 4$ for $\mathrm{Re} = 100$ and $d_z = 10$ for $\mathrm{Re} = 1000$.

The LSTM propagator of LED is benchmarked against SINDy and RC in predicting the dynamics, starting from 10 initial conditions randomly sampled from the test data for a prediction horizon of $T = 20$ (100 timesteps). The hyperparameters (reported on Tables D.13 to D.15 along with the training times) are tuned based on the MNAD on the validation data. The logarithm of the MNAD is given in Figure 6.18(a) for $\mathrm{Re} = 100$ and Figure 6.19(a) for $\mathrm{Re} = 1000$. For the $\mathrm{Re} = 100$ case, the LSTM exhibits

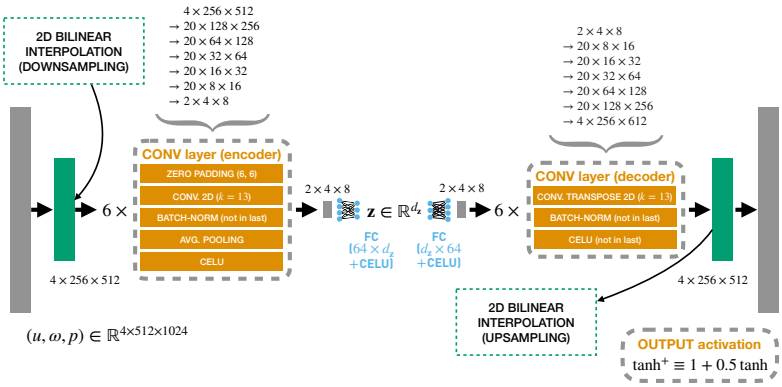


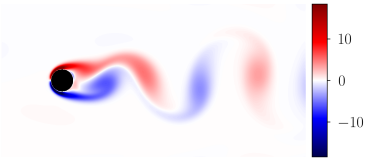
FIGURE 6.15: The architecture of the CNN employed in the flow past a cylinder example.

lower MNAD and lower variance than RC and SINDy. For the challenging $\mathrm{Re} = 1000$ scenario, LSTM and RC exhibit lower MNAD than SINDy, with the LSTM being more robust (lower variance).

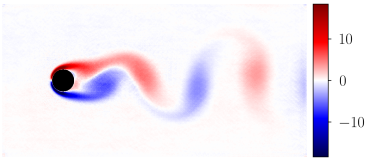
A prediction of the vorticity ω by Latent-LED at lead time $T = 4$ is given in Figure 6.16 for $\mathrm{Re} = 100$, and Figure 6.17 for $\mathrm{Re} = 1000$. LED captures the flow for both $\mathrm{Re} \in \{100, 1000\}$. The error concentrates mainly around the cylinder, rendering the accurate prediction of the drag coefficient challenging. In Figures 6.16(d) and 6.17(d), the latent space of Latent-LED is compared with the transformation of the data to the latent space. The predictions stay close to the attractor even for a very large horizon ($T = 20$). The Strouhal number St (defined in Equation 6.11) describes the periodic vortex shedding at the wake of the cylinder. By estimating the dominant frequency of the latent state using a Fourier analysis, we find that LED reproduces exactly the St of the system dynamics for both $\mathrm{Re} \in \{100, 1000\}$ cases.

In the $\mathrm{Re} = 100$ case, Latent-LED recovers a periodic nonlinear mode in the latent space and can forecast the dynamics accurately, as illustrated in Figure 6.16(d). In this case, approaches based on the Galerkin method or dynamic mode decomposition (DMD), construct ROM with six to eight degrees of freedom (Taira et al., 2020) that capture the most energetic spatio-temporal modes. In contrast, the latent space of LED in the $\mathrm{Re} = 100$ case has a dimensionality of $d_z = 4$. In the challenging $\mathrm{Re} = 1000$ scenario, LED with $d_z = 10$ can capture accurately the characteristic vortex street and

long-term dynamics. We note that, to the best of our knowledge, ROMs for flows past a cylinder have been so far limited to laminar periodic flows in the order of $\mathrm{Re} = 100$ while this study advances the state-of-the-art by one order of magnitude.

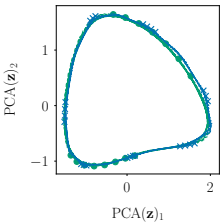


(a) The vorticity field ω at lead time $T = 4$ for $\mathrm{Re} = 100$.



(b) The vorticity field $\widetilde{\omega}$ predicted by Latent-LED with $d_z = 4$ at final time $T = 4$ for $\mathrm{Re} = 100$.

(c) The MNAD for $\mathrm{Re} = 100$.

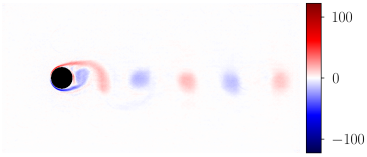


(d) The LED latent state dynamics (▪▪▪✕▪▪▪) compared against the attractor of the data embedded in the latent space (▬●▬).

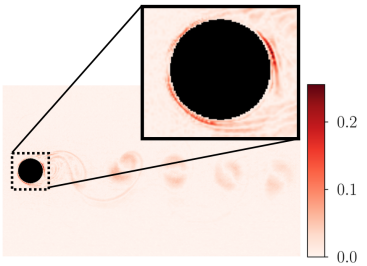
FIGURE 6.16: Results of Latent-LED applied on the viscous flow past a cylinder at $\mathrm{Re} = 100$.

Starting from 4 initial conditions randomly sampled from the test data, six LED variants (Latent-LED, Multiscale-LED with $T_\mu = 0.4$, $T_m \in \{0.4, 0.8, 1.2, 2, 4\}$ for $\mathrm{Re} = 100$, and Latent-LED, Multiscale-LED with $T_\mu = 1.6$, $T_m \in \{0.8, 1.6, 3.2, 6.4, 12.8\}$ for $\mathrm{Re} = 1000$) are tested on predicting the dynamics of the flow up to $T_f = 20$, after $T_{warm} = 2$. The MNAD is plotted in Figure 6.18(b) for $\mathrm{Re} = 100$, and Figure 6.19(b) for $\mathrm{Re} = 1000$. The speed-up is plotted in Figure 6.18(d) for $\mathrm{Re} = 100$, and Figure 6.19(d) for $\mathrm{Re} = 1000$. The Latent-LED is two orders of magnitude faster than the flow solver, while exhibiting MNAD errors of 0.02 and 0.04 for $\mathrm{Re} = 100$, and $\mathrm{Re} = 1000$ respectively. By alternating between macro and micro, the error is reduced, at the cost of decreased speed-up.

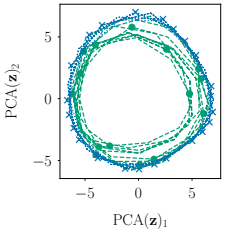
(a) The vorticity field ω at lead time $T = 4$ for $\mathrm{Re} = 1000$.



(b) The vorticity field $\widetilde{\omega}$ predicted by Latent-LED with $d_z = 10$ at final time $T = 4$ for $\mathrm{Re} = 1000$.

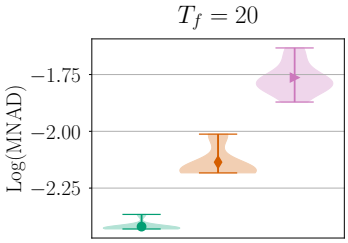


(c) The MNAD for $\mathrm{Re} = 1000$.

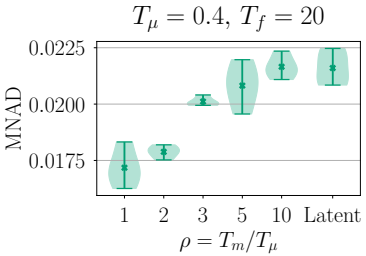


(d) The LED latent state dynamics (▪▪▪✕▪▪▪) compared against the attractor of the data embedded in the latent space (▬●▬).

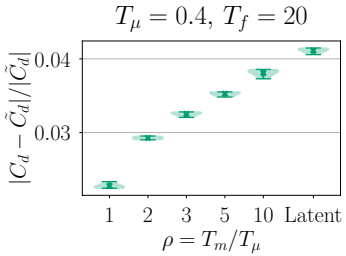
FIGURE 6.17: Results of Latent-LED applied on the viscous flow past a cylinder at $\mathrm{Re} = 1000$.



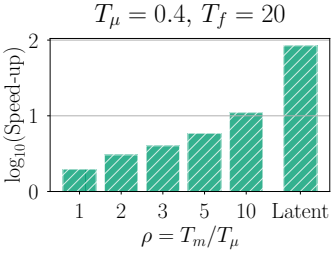
(a) Comparison of different macrodynamics propagators (AE-LSTM; AE-RC; AE-SINDy) for Re = 100.



(b) The MNAD error on the drag between predictions by Multiscale-LED with $d_z = 4$ and the reference data as a function of the multiscale ratio ρ for Re = 100.

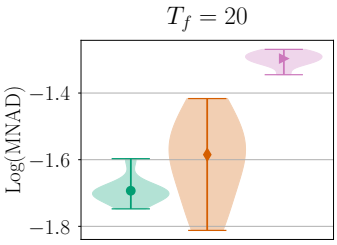


(c) The relative error on the drag between predictions by Multiscale-LED with $d_z = 4$ and the reference data as a function of the multiscale ratio ρ for Re = 100.

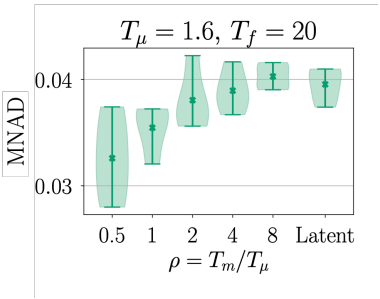


(d) The speed-up of Multiscale-LED compared to the flow solver w. r. t. ρ for Re = 100.

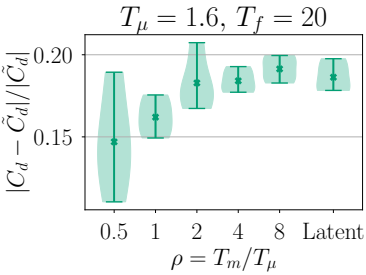
FIGURE 6.18: Results on the viscous flow past a cylinder at Re = 100.



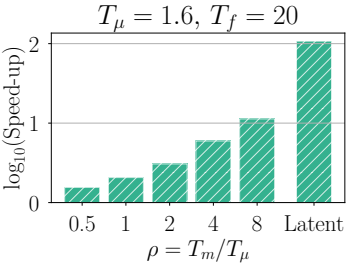
(a) Comparison of different macrodynamics propagators (AE-LSTM; AE-RC; AE-SINDy) for Re = 1000.



(b) The MNAD error on the drag between predictions by Multiscale-LED with $d_z = 10$ and the reference data as a function of the multiscale ratio ρ for Re = 1000.



(c) The relative error on the drag between predictions by Multiscale-LED with $d_z = 10$ and the reference data as a function of the multiscale ratio ρ for Re = 1000.



(d) The speed-up of Multiscale-LED compared to the flow solver w. r. t. ρ for Re = 1000.

FIGURE 6.19: Results on the viscous flow past a cylinder at Re = 1000.

In Figures 6.18(c) and 6.19(c), the relative error on the drag coefficient C_d (defined in Equation 6.12) is plotted as a function of the multiscale ratio ρ. Latent-LED exhibits a relative error of 0.04 that is reduced to approximately 0.02 for $\rho = 1$. For $\mathrm{Re} = 1000$, as we observe in Figure 6.17(c), the prediction error of LED concentrates around the cylinder, which leads to an inaccurate computation of the drag. Even though Multiscale-LED reduces this error, it remains on the order of 0.15.

6.5 DISCUSSION

We have presented a novel framework for learning the effective dynamics (LED) and accelerating the simulations of multiscale (stochastic or deterministic) complex dynamical systems. Our work relies on augmenting the Equation-Free formalism with state-of-the-art ML methods.

The LED framework is tested on several benchmark problems. In systems where high-dimensional state dynamics are computationally expensive, LED accelerates the simulation by propagating on the latent space and upscaling to the high-dimensional states with the probabilistic, generative mixture density, or deterministic convolutional decoder. This comes at the cost of training the networks, which is performed once offline. The trained model can forecast the dynamics starting from any arbitrary initial condition.

The efficiency of LED was evaluated in forecasting the FitzHugh-Nagumo model dynamics, achieving an order of magnitude lower approximation error compared to other Equation-Free approaches while being two orders of magnitude faster than the Lattice Boltzmann solver. We demonstrated that the proposed framework identifies the effective dynamics of the Kuramoto-Sivashinsky equation with $L = 22$, capturing the long-term behavior ("climate dynamics"), achieving a speed-up of $S \approx 100$. Furthermore, LED captures the long-term dynamics of a flow past a cylinder in $\mathrm{Re} = 100$ and $\mathrm{Re} = 1000$ accurately, while being two orders of magnitude faster than a flow solver. We note that the present method is readily applicable to all problems where Equation-Free, HMM, and FLAVOR methodologies have been applied.

In summary, LED identifies and propagates the effective dynamics of dynamical systems with multiple spatio-temporal scales providing significant computational savings. Moreover, LED provides a systematic way of trading between speed-up and accuracy for a multiscale system by switch-

ing between the propagation of the latent dynamics and evolution of the original equations, iteratively correcting the statistical error at the cost of reduced speed-up.

The LED does not presently contain any mechanism to decide when to upscale the latent space dynamics. We do not expect LED to generalize to dynamical regions drastically different from those represented in the training data. We can alleviate this issue by considering the uncertainty of LED in the latent space predictions and by employing Bayesian networks that can detect samples unseen during training. The present methodology can be deployed both in problems described by first principles as well as for problems where only data are available for either the macro or micro-scale descriptions of the system.

In summary, LED creates unique algorithmic alloys between data-driven and first-principles models and opens new horizons for the accurate and efficient prediction of complex multiscale systems.

7

ACCELERATING MOLECULAR SIMULATIONS BY LEARNING THEIR EFFECTIVE DYNAMICS

7.1 RELATED WORK

In the context of Molecular Dynamics (MD), ML methods (Bishop, 2006; Michie, 1968), exploiting the expressive power of deep networks and their scalability to large datasets, have been used to alleviate the computational burden associated with the simulation of proteins, leading to profound scientific discoveries (Butler et al., 2018; Noé, Tkatchenko, et al., 2020; Schütt, Chmiela, et al., 2020).

The pioneering work of Behler et al., 2007 utilized neural networks to learn an approximate potential energy surface of density functional theory (DFT) in bulk silicon from quantum mechanical calculations, performing MD simulations with this approximate potential and accelerating the DFT simulations. The field of data-driven learning of potential energy surfaces and force fields is rapidly attracting attention with important recent extensions and applications (Bartók et al., 2017; Cheng et al., 2019; Chmiela et al., 2018; Faber et al., 2017; Hansen et al., 2013; Imbalzano et al., 2018; Rowe et al., 2020; Rupp et al., 2012; Schütt, Sauceda, et al., 2018). ML is employed to identify CG models for MD in Durumeric et al., 2019; J. Wang et al., 2019; L. Zhang et al., 2018. Boltzmann generators, proposed in Noé, Olsson, et al., 2019, sample from the equilibrium distribution of a molecular system directly surpassing the need to perform MD.

Early ML methods for the identification of CVs utilized manifold learning techniques, i.e. diffusion maps (Boninsegna et al., 2015; Coifman, Kevrekidis, et al., 2008; Preto et al., 2014; Rohrdanz et al., 2011; W. Zheng et al., 2013), while others were based on the variational approach (Nuske et al., 2014) leading to the time-lagged independent analysis (TICA) (Pérez-Hernández et al., 2016). TICA is based on the Koopman operator theory, suggesting the existence of a latent transformation to an infinite-dimensional space that linearizes the dynamics on average. As a consequence, slow CVs are modeled as linear combinations of feature functions of the state of the protein (atom coordinates or internal structural coordinates). Coarse-

graining of the molecular dynamics is achieved by discretizing the state-space and employing indicator vector functions as features (Buchete et al., 2008; Noé, Doose, et al., 2011; Nuske et al., 2014; Ribeiro et al., 2018). Consequently, the feature state dynamics reduce to the propagation law of a Markov State Model (MSM). MSMs have been extended to "Core Set MSMs" in Schütte et al., 2011 employing Markovian milestoning (Vanden-Eijnden et al., 2009) on metastable core sets.

More recently, the need for expert knowledge to construct the latent feature functions has been alleviated by learning the latent space using neural networks (Mardt et al., 2018; Wehmeyer et al., 2018). The dynamics on the latent space are assumed to be linear and Markovian. For example, VAMPnets (W. Chen et al., 2019; Mardt et al., 2018) learn nonlinear features of the molecular state with autoencoder (AE) networks. However, they are not generative and cannot recover the detailed configuration of the protein (decoding part). Moreover, the method requires the construction of an MSM to sample the latent dynamics and approximate the timescales of the dynamics. Time-lagged AEs have been utilized to identify a reaction coordinate embedding and propagate the dynamics in Ref. (Wehmeyer et al., 2018), but they are not generative, as the learned mappings are deterministic, while the effective dynamics are assumed to be Markovian. AEs have been coupled with dynamic importance sampling in Bhatia et al., 2021 to accelerate multiscale simulations and investigate the interactions of RAS proteins with a plasma membrane.

Extensions to generative approaches include Refs. (Hernández et al., 2018; Sidky et al., 2020; H. Wu et al., 2018). In Ref. (H. Wu et al., 2018), a deep generative MSM is utilized to capture the long-timescale dynamics and sample realistic alanine dipeptide configurations. Even though Mixture Density Networks (MDNs) are employed in Ref. (Sidky et al., 2020) to propagate the dynamics in the latent space, memory effects are not taken into account. The proposed method is based on the autocorrelation loss, which suffers from the dependency on the batch-size (Hernández et al., 2018). In Ribeiro et al., 2018; Y. Wang et al., 2019, the Reweighted autoencoded variational Bayes for enhanced sampling (RAVE) method is proposed that alternates between iterations of MD and a Variational AE (VAEs) model. RAVE encodes each timestep independently without considering the temporal aspect of the latent dynamics. RAVE requires the transition to the high-dimensional configuration space to progress the simulation in time, which can be computationally expensive. In recent work, RAVE has been ex-

panded to incorporate a "Variational Mixture of Posteriors" (Tomczak et al., 2018) prior in the VAEs (D. Wang et al., 2021) enhancing its performance.

State-of-the-art methods in modeling molecular systems imply memoryless (Markovian) latent space dynamics by selecting an appropriate time-lag in the master equations (Buchete et al., 2008; Noé, Doose, et al., 2011). The time-lag is usually estimated heuristically, balancing the requirements to be large enough so that the Markovian assumption holds, and at the same time small enough to ensure that the method samples the configuration space efficiently. We remark that in cases where a protein interacts with a solvent, only the configuration of the protein is taken into account and not the solvent. This renders the Markovian assumption in the latent dynamics somewhat unrealistic. Furthermore, in multiple cases in practice, the friction extracted from molecular dynamics simulations exhibits significant memory with a decay time that is in the nanosecond range and thus, of the same order as the folding and unfolding time (Ayaz et al., 2021). This issue is addressed here by employing probabilistic LSTM-RNNs (Hochreiter and Schmidhuber, 1997) in the LED framework introduced in Chapter 6 that capture memory effects of the latent dynamics. An LSTM has been used in Tsai et al., 2020 in the form of a language model to learn non-Markovian protein dynamics. The model they propose, however, requires discretization of the state-space, is not generative, and does not identify a low-order latent representation of the dynamics.

Here we propose a novel data-driven generative framework that relies on learning the effective dynamics of molecular systems. As described in Chapter 6, LED is founded on the Equation-Free Framework (Kevrekidis, Gear, Hyman, et al., 2003). In this chapter, we show how the LED enriches the EFF by employing ML methodologies to evolve the latent space dynamics with the Mixture Density Network - Long Short-Term Memory RNN (MDN-LSTM) and the two-way mapping between coarse and fine-scales with Mixture Density Network Autoencoders (MDN-AEs) (Bishop, 1994). These enrichments are essential in extending the applicability of EFF to non-Markovian dynamics with probabilistic transitions on the latent space with strong nonlinearities. We demonstrate the effectiveness of the LED framework in simulations of the Müeller-Brown potential (MBP), the Trp Cage miniprotein, and the alanine dipeptide in water. LED can accurately capture the statistics and reproduce the free energy landscape from data. Moreover, LED uncovers low-energy metastable states in the free energy projected to the latent space and recovers the transition time scales between them. We find that in simulations of the alanine dipeptide and the Trp Cage

miniprotein, LED is three orders of magnitude faster than the classical MD solver. As a data-driven generative method, LED can sample novel unseen configurations interpolating the training data and accelerating state-space exploration.

This chapter is based on the paper "Accelerated Simulations of Molecular Systems through Learning of Effective Dynamics" (P. R. Vlachas, Zavadlav, et al., 2021). The computational resources were provided by a grant from the Swiss National Supercomputing Centre (CSCS) under project s930.

7.2 METHODS

Here, we extend the LED framework introduced in Chapter 6 for molecular systems. The LED framework is founded on the Equation-Free Framework (EFF) (Kevrekidis, Gear, Hyman, et al., 2003). It addresses the critical bottlenecks of EFF, namely, the coarse to fine mapping and the evolution of the latent space using an MDN-AE and an MDN-LSTM, respectively. An illustration of the LED framework is given in Figure 7.1.

In the following, the state of a molecule at time t is described by a high-dimensional vector $\boldsymbol{s}_t \in \Omega \subseteq \mathbb{R}^{d_s}$, where $d_s \in \mathbb{N}$ denotes its dimension. The state vector can include the atom positions or their rotation/translation-invariant features obtained using, for example, the Kabsch transform (Kabsch, 1976). A trajectory of this system is obtained by an MD integrator and the state of the molecule after a timestep Δt is described by the probability distribution function (PDF):

$$p(\boldsymbol{s}_{t+\Delta t} | \boldsymbol{s}_t). \tag{7.1}$$

The transition distribution in Equation 7.1 depends on the choice of Δt.

7.2.1 *Mixture Density Network Autoencoder*

Here, the MDN-AE identifies the latent (coarse) representation and upscales it probabilistically to the high-dimensional state-space. MDNs (Bishop, 2006) are neural architectures that can represent arbitrary conditional distributions. The MDN output is a parametrization of the distribution of a multivariate random variable conditioned on the network's input.

The latent state is computed by $\boldsymbol{z}_t = \mathcal{E}(\boldsymbol{s}_t; \boldsymbol{w}_\mathcal{E})$, where $\mathcal{E}$ is the encoder (a deep neural network) with trainable parameters $\boldsymbol{w}_\mathcal{E}$ and $\boldsymbol{z}_t \in \mathbb{R}^{d_z}$ with

$d_z \ll d_s$. Since z_t is a coarse approximation, many states can be mapped to the same z_t. As a consequence, a deterministic mapping $z_t \to s_t$ like the one used in Mardt et al., 2018; Wehmeyer et al., 2018 does not provide the full distribution $p(s_t|z_t)$. Here, an MDN is employed to model the upscaling conditional PDF $p(s_t|z_t)$ described by the parameters $w_{s|z}$. These parameters are the outputs of the decoder with weights $w_{\mathcal{D}}$ and are a function of the latent representation z_t, i.e.

$$w_{s|z}(z_t) = \mathcal{D}(z_t; w_{\mathcal{D}}). \tag{7.2}$$

The state of the molecule can then be sampled from $p(s_t|z_t) := p(s_t; w_{s|z})$.

Arguably, including the rotation/translation-invariant features of the molecule under study in the state s_t, encourages the MDN to sample physically meaningful molecular configurations. The state s_t is composed of states representing bond lengths $s_t^b \in \mathbf{R}^{d_s^b}$, and angles $s_t^a \in \mathbf{R}^{d_s^a}$. Initially, the MD data of the bonds are scaled to $[0, 1]$. An auxiliary variable vector $v_t \in \mathbf{R}^{d_s^b}$ is defined to model the distribution of bonds. In particular, $p(v_t|z_t)$ is modeled as a Gaussian mixture model with K_s mixture kernels as

$$p(v_t|z_t) = \sum_{k=1}^{K_s} \pi_v^k(z_t)\, \mathcal{N}\Big(\mu_v^k(z_t), \sigma_v^k(z_t) \Big), \tag{7.3}$$

and the mapping $s_t^b = \ln(1 + \exp(v_t))$ is used to recover the distribution of the scaled bond lengths at the output. The functional form of the mixing coefficients $\pi_v^k(z_t)$, the means $\mu_v^k(z_t)$, and the variances $\sigma_v^k(z_t)$ is a deep neural network (decoder $\mathcal{D}$). The distribution of the dihedral angles is modeled with the circular normal (von Mises) distribution, i.e.

$$p(s_t^a|z_t) = \sum_{k=1}^{K_s} \pi_{s^a}^k(z_t)\, \frac{\exp\Big(\nu_{s^a}^k(z_t)\, \cos\Big(s_t^a - \mu_{s^a}^k(z_t) \Big) \Big)}{2\pi I_0\big(\nu_{s^a}^k(z_t)\big)}, \tag{7.4}$$

where $I_0(\nu_{s^a}^k)$ is the modified Bessel function of order 0. Here, again the functional form of $\pi_{s^a}^k(z_t)$, $\mu_{s^a}^k(z_t)$ and $\nu_{s^a}^k(z_t)$ is a deep neural network (decoder $\mathcal{D}$).

In total, the outputs of the decoder $\mathcal{D}$ that parametrize $p(s_t|z_t)$ are

$$w_{s|z} = \{\pi_v^k, \mu_v^k, \sigma_v^k, \pi_{s^a}^k, \mu_{s^a}^k, \nu_{s^a}^k\}_{k \in \{1,\dots,K_s\}}, \tag{7.5}$$

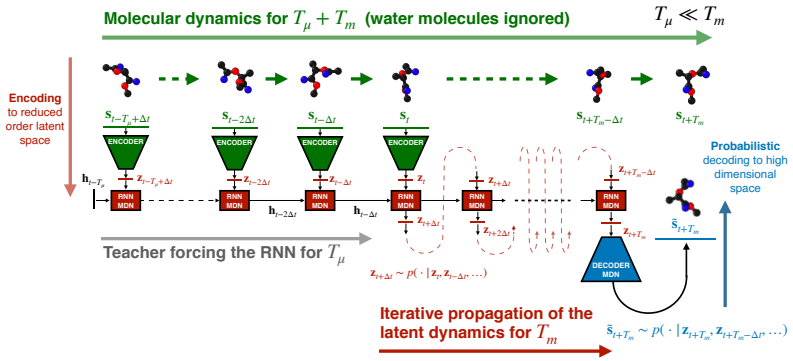


FIGURE 7.1: High-dimensional (fine-scale) dynamics $\mathbf{s}_t$ are simulated for a short period (T_μ). During this warm-up period, the state $\mathbf{s}_t$ is passed through the encoder network. The outputs of the encoder $\mathbf{z}_t$ provide the time series input to the LSTM, allowing for the update of its hidden state $\mathbf{h}_t$, thus capturing non-Markovian effects. The output of the LSTM is a parametrization of the probabilistic non-Markovian latent dynamics $p(\mathbf{z}_t|\mathbf{h}_t)$. Starting from the last latent state $\mathbf{z}_t$, the LSTM iteratively samples $p(\mathbf{z}_t|\mathbf{h}_t)$ and propagates the low-order latent dynamics up to a total horizon of T_m time units, with $T_m > T_\mu$. The LED decoder may be utilized at any desired time scale to map the latent state $\mathbf{z}_t$ back to a high-dimensional representation $\mathbf{s}_t \sim p(\cdot|\mathbf{z}_t, \mathbf{z}_{t-\Delta t}, \ldots)$. Propagation in the low-order space unraveled by LED is orders of magnitude cheaper than evolving the high-dimensional system based on first principles (molecular dynamics/density functional theory, etc.).

which are all functions of the latent state z_t, which is the decoder input. The MDN-AE is trained to predict the mixing coefficients maximizing the data likelihood

$$\begin{aligned} \boldsymbol{w}_{\mathcal{E}}, \boldsymbol{w}_{\mathcal{D}} &= \arg\max_{\boldsymbol{w}_{\mathcal{E}}, \boldsymbol{w}_{\mathcal{D}}} p(\boldsymbol{s}_t|\boldsymbol{z}_t) \\ &= \arg\max_{\boldsymbol{w}_{\mathcal{E}}, \boldsymbol{w}_{\mathcal{D}}} p\big(\boldsymbol{s}_t; \boldsymbol{w}_{\boldsymbol{s}|\boldsymbol{z}}\big), \end{aligned} \tag{7.6}$$

where $\boldsymbol{w}_{\boldsymbol{s}|\boldsymbol{z}} = \mathcal{D}\big(\mathcal{E}(\boldsymbol{s}_t; \boldsymbol{w}_{\mathcal{E}}); \boldsymbol{w}_{\mathcal{D}}\big)$ is the output of the MDN-AE and $\boldsymbol{s}_t$ are the MD data.

7.2.2 *Long Short-Term Memory Recurrent Neural Network*

The latent dynamics may be characterized by non-Markovian effects, i.e.

$$p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{z}_t, \boldsymbol{z}_{t-\Delta t}, \dots),$$

due to the neglected degrees of freedom (solvent) or the selection of a relatively small time-lag Δt.

Here the LSTM cell architecture introduced in Section 2.1.4.2 is utilized to evolve the nonlinear and non-Markovian latent dynamics. Here, we rewrite the propagation in the LSTM cell given by:

$$\boldsymbol{h}_t, \boldsymbol{c}_t = \mathcal{F}_{hh}\big(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t}, \boldsymbol{c}_{t-\Delta t}; \boldsymbol{w}_{\mathcal{F}_{hh}}\big), \tag{7.7}$$

where the hidden-to-hidden recurrent mapping $\mathcal{F}_{hh}$ takes the form given in Equation 2.10. The dimension of the hidden state $d_{\boldsymbol{h}}$ (number of hidden units) in the LSTM controls the capacity of the cell to encode history information. The set of trainable parameters of the LSTM are given in Equation 2.11.

7.2.3 *Mixture Density LSTM Network*

The LSTM captures the history of the latent state and the non-Markovian latent transition dynamics are expressed as:

$$p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{z}_t, \boldsymbol{z}_{t-\Delta t}, \dots) = p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{h}_t), \tag{7.8}$$

where $\boldsymbol{h}_t$ given in Equation 7.7. A second MDN is used to model the conditional distribution $p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{h}_t)$ of the latent transition dynamics. This MDN is conditioned on the hidden state of the LSTM $\boldsymbol{h}_t$ and implicitly conditioned on the history, i.e., $p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{z}_t, \boldsymbol{z}_{t-\Delta t}, \dots) := p(\boldsymbol{z}_{t+\Delta t}; \boldsymbol{w}_{z|\boldsymbol{h}})$, so it can capture non-Markovian dynamics. The distribution $p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{h}_t)$ is modeled as a Gaussian mixture with K_z mixture kernels

$$p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{h}_t) = \sum_{k=1}^{K_z} \pi_z^k(\boldsymbol{h}_t)\, \mathcal{N}\Big(\boldsymbol{\mu}_z^k(\boldsymbol{h}_t), \boldsymbol{\sigma}_z^k(\boldsymbol{h}_t)\Big), \tag{7.9}$$

with parameters $\boldsymbol{w}_{z|\boldsymbol{h}}$ given by

$$\boldsymbol{w}_{z|\boldsymbol{h}}(\boldsymbol{h}_t) = \{\pi_z^k(\boldsymbol{h}_t), \boldsymbol{\mu}_z^k(\boldsymbol{h}_t), \boldsymbol{\sigma}_z^k(\boldsymbol{h}_t)\}, \tag{7.10}$$

that are a function of $\boldsymbol{h}_t$. These parameters are the outputs of the neural network $\mathcal{Z}(\boldsymbol{h}_t; \boldsymbol{w}_{\mathcal{Z}})$, with trainable weights $\boldsymbol{w}_{\mathcal{Z}}$, and are a function of the hidden state, i.e.

$$\begin{aligned} p(\boldsymbol{z}_{t+\Delta t}|\boldsymbol{h}_t) &:= p(\boldsymbol{z}_{t+\Delta t}; \boldsymbol{w}_{\boldsymbol{z}|\boldsymbol{h}}), \\ \boldsymbol{w}_{\boldsymbol{z}|\boldsymbol{h}}(\boldsymbol{h}_t) &= \mathcal{Z}(\boldsymbol{h}_t; \boldsymbol{w}_{\mathcal{Z}}). \end{aligned} \tag{7.11}$$

The weights of the LSTM $\boldsymbol{w}_{\mathcal{F}_{hh}}$ and the latent MDN $\boldsymbol{w}_{\mathcal{Z}}$ are trained to output the parameters $\boldsymbol{w}_{\boldsymbol{z}|\boldsymbol{h}}$ that maximize the likelihood of the latent evolution

$$\boldsymbol{w}_{\mathcal{F}_{hh}}, \boldsymbol{w}_{\mathcal{Z}} = \underset{\boldsymbol{w}_{\mathcal{F}_{hh}}, \boldsymbol{w}_{\mathcal{Z}}}{\arg\max}\, p\left(\boldsymbol{z}_{t+\Delta t}; \boldsymbol{w}_{\boldsymbol{z}|\boldsymbol{h}}\right), \tag{7.12}$$

where $\boldsymbol{w}_{\boldsymbol{z}|\boldsymbol{h}}$ is defined in Equation 7.11, and $\boldsymbol{h}_t$ appearing in Equation 7.11 is defined in Equation 7.7. During the training phase, the MD trajectory data $\boldsymbol{s}_t$ are provided at the input of the trained MDN-AE $\boldsymbol{z}_t = \mathcal{E}(\boldsymbol{s}_t; \boldsymbol{w}_{\mathcal{E}})$. The encoder outputs the latent dynamics $\boldsymbol{z}_t$ that are used to update the hidden state of the LSTM and optimize its weights according to Equation 7.12. In contrast to the linear operator utilized in MSMs, the recurrent functional form in Equation 7.7 can be nonlinear and incorporate memory effects via the hidden state of the LSTM.

7.2.4 *LED for Molecular Systems*

The LED framework can be employed to accelerate MD simulations and enable more efficient exploration of the state-space and uncovering of novel protein configurations. The networks in LED are trained on trajectories from MD simulations in two phases. First, the MDN-AE provides a reduced-order representation, maximizing the data likelihood. The MDN-AE is trained with Backpropagation (Rumelhart et al., 1986) using the adaptive stochastic optimization method Adam (Kingma et al., 2014). Adding a pre-training phase fitting the kernels $\boldsymbol{\mu}^k, \boldsymbol{\sigma}^k$ of the MDN-AE to the data, and fixing them during MDN-AE training led to better results. Next, the MDN-LSTM is trained to forecast the latent space dynamics (the MDN-AE weights are considered fixed) to maximize the latent data likelihood. MDN-LSTM is trained with BPTT (P. J. Werbos, 1988) with Adam optimizer.

The LED propagates the computationally inexpensive dynamics on its latent space. Starting from an initial state from a test dataset (unseen during training), a short time history T_μ of the state evolution is utilized to warm-up the hidden state of the LED. The MDN-LSTM is used to propagate the latent dynamics for a time horizon $T_m \gg T_\mu$. High-dimensional state

configurations can be recovered at any time instant by using the probabilistic decoder part of MDN-AE. We find that the LED framework can accelerate MD simulations by three orders of magnitude.

The LED is implemented in PYTHON (Van Rossum et al., 1995) in the PYTORCH (Paszke et al., 2019) library, mapped to a single Nvidia Tesla P100 GPU, and executed on the XC50 compute nodes of the Piz Daint supercomputer at the Swiss national supercomputing centre (CSCS).

7.3 RESULTS

The LED framework is tested in three systems, single-particle Langevin dynamics using the two-dimensional MBP, the Trp Cage miniprotein, and the alanine dipeptide, widely adopted as benchmarks for molecular dynamics modeling (Mardt et al., 2018; Müller et al., 1979; Nuske et al., 2014; Sidky et al., 2020; Wehmeyer et al., 2018).

7.3.1 *Müller-Brown Potential*

The Langevin dynamics of a particle in the MBP are characterized by the stochastic differential equation

$$m\ddot{\boldsymbol{x}}(t) = -\nabla V\big(\boldsymbol{x}(t)\big) - \gamma\dot{\boldsymbol{x}}(t) + \sqrt{2\kappa_B\mathcal{T}}R(t), \tag{7.13}$$

where $\boldsymbol{x} \in \mathbb{R}^2$ is the position, $\dot{\boldsymbol{x}}$ is the velocity, $\ddot{\boldsymbol{x}}$ is the acceleration, $V(\boldsymbol{x})$ is the MBP (defined in Appendix E.1), κ_B is the Boltzmann's constant, $\mathcal{T}$ is the temperature, γ is the damping coefficient, and $R(t)$ a delta-correlated stationary Gaussian process with zero-mean. The nature of the dynamics is affected by the damping coefficient γ. Low damping coefficients lead to an inertial regime. High damping factors lead to a diffusive regime (Brownian motion) with less prominent memory effects. Here, a damping $\gamma = 1$ is considered, along with $\kappa_B\mathcal{T} = 15$.

The equations are integrated with the Velocity Verlet algorithm with timestep $\delta t = 10^{-2}$, starting from 96 initial conditions randomly sampled uniformly from $\boldsymbol{x} \in [-1.5, 1.2] \times [-0.2, 2]$ till $T = 10^4$, after truncating an initial transient period of $\tilde{T} = 10^3$. The data are sub-sampled, keeping every 50^{th} data points to create the training and testing datasets for LED. The coarse timestep of LED is $\Delta t = 0.5$. We use 32 initial conditions for training, 32 for validation and all 96 for testing. LED is trained with a 1D reduced-order latent representation $\boldsymbol{z}_t \in \mathbb{R}$. The reader is referred to Appendix E.1

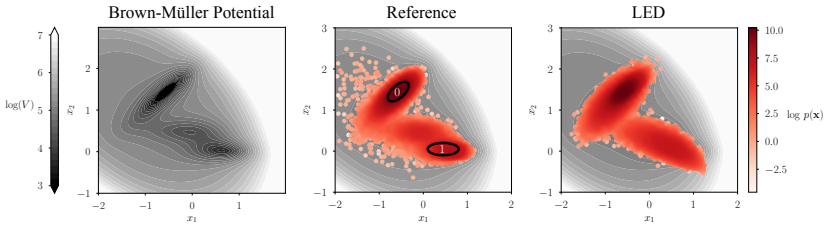


FIGURE 7.2: From left to right: the Müller-Brown potential, a scatter plot of the joint state distribution computed from reference data (with annotation of two long-lived metastable states), and the same scatter plot obtained by LED sampled trajectories.

for further information regarding the MBP parameterization of Müller et al., 1979 and in Appendix E.1.2 hyperparameters of LED.

The MBP is shown in Figure 7.2, along with a density scatter plot of the joint distribution of the MBP states computed from the testing data and LED. The potential has three minima. The potential value around the middle one, however, is more than one order of magnitude higher than the other two and lies very close to the one on the right (see Figure 7.2 left and middle). The joint distribution reveals two dominant long-lived metastable states that correspond to the two dominant low-energy regions. The local minimum in the middle is not clearly distinguishable. The LED learns to transition probabilistically between the metastable states, mimicking the system's dynamics and reproducing the state statistics. We note, however, that the LED cannot distinguish the region of the local minimum in the middle as a separate metastable state (Figure 7.2, right). As LED relies on reducing the dimensionality of the state, projecting to a very low-order space, it captures only the large-scale (coarse) characteristics. This dimensionality reduction is essential for the acceleration that LED provides for large molecular systems. In turn, as a consequence of this dimensionality reduction, a "weak" metastable state region (local minimum, but high potential compared to other nearby minima) that lies close to a dominant one of much lower potential value can be missed (or be absorbed to the nearby regions).

The free energy projected on the latent space, i.e., $F = -\kappa_B \mathcal{T} \log\, p(z_t)$ is plotted in Figure 7.3. The free energy profile of the trajectories sampled from LED matches closely the one from the reference data with an RMSE between the two free energy profiles of $\approx 0.74 \kappa_B \mathcal{T}$. LED reveals two minima in the free energy profile. As noted before, the third local minima of the

BMP is absorbed by the dominant one close to it. Even though LED cannot distinguish the third local minima, the projected free energy profile is reproduced. Utilizing the LED decoder, the latent states in these regions are mapped to their image in the two-dimensional state representation $s_t \in \mathbb{R}^2$ (here corresponding to $x_t \in \mathbb{R}^2$) in Figure 7.3. LED is mapping the low-energetic regions in the free energy profile to the long-lived metastable states in the two-dimensional space of the MBP.

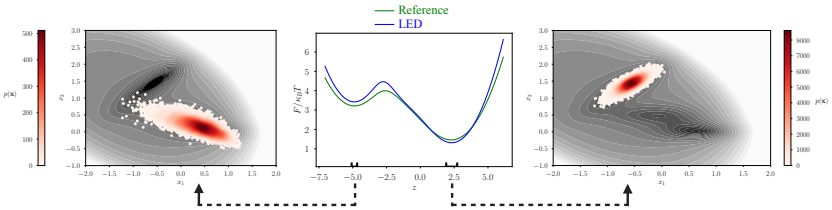


FIGURE 7.3: Middle: free energy profile projected on the latent space learned by the LED encoder, i.e., $F = -\kappa_B \mathcal{T} \ln p(z_t)$. The free energy profile computed by LED (propagation of the latent dynamics with LED) matches closely the one from the reference data. Quantitatively, the RMSE is $0.74\kappa_B\mathcal{T}$. LED recovers two low-energy regions that are mapped to the two long-lived metastable states (left and right) in the two-dimensional state-space $s_t \in \mathbb{R}^2$.

The marginal distributions of the MBP states from trajectories sampled from the LED is compared with the ground-truth (test data) in Figure 7.4.

Next, we evaluate the LED framework in reproducing the transition times between the long-lived states. In LED, metastable states can be defined either on the reduced-order latent space $z_t \in \mathbb{R}$ or the state-space $s_t \in \mathbb{R}^2$ (as the decoder can map any latent state to a state-space). In the following, two metastable states are defined as ellipses on the state-space (see Appendix E.1.1) and depicted in Figure 7.2. The time scales will vary depending on the definition of the metastable states in the phase space. The distribution of transition times computed from LED trajectories is compared with the transition time distribution from the test data in Figure 7.5. LED captures the transition time distributions quantitatively, and the mean values are close to each other. In Appendix E.1.3, we also report the transition times obtained with metastable states definition on the latent space. This approach has the benefit that it does not require prior knowledge about the metastable states in the state-space. In conclusion, LED captures the joint state distribution on the MBP and matches the timescales of the system.

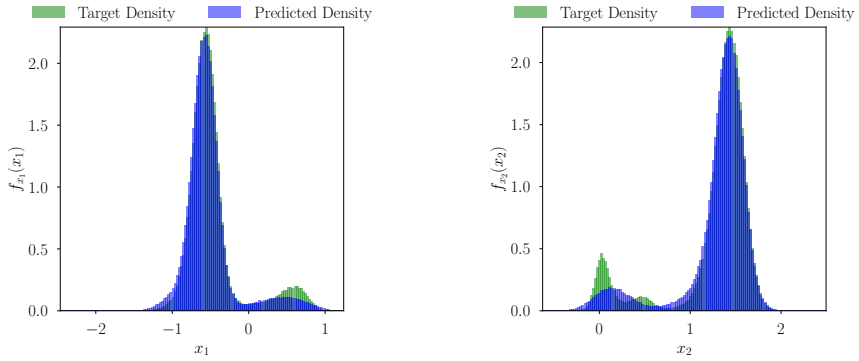


FIGURE 7.4: Comparison of the marginal distributions of the MBP states x_1 and x_2 between the test data and trajectories of LED. The LED is propagating the dynamics on a 1D reduced-order latent state, i.e., $d_z = 1$.

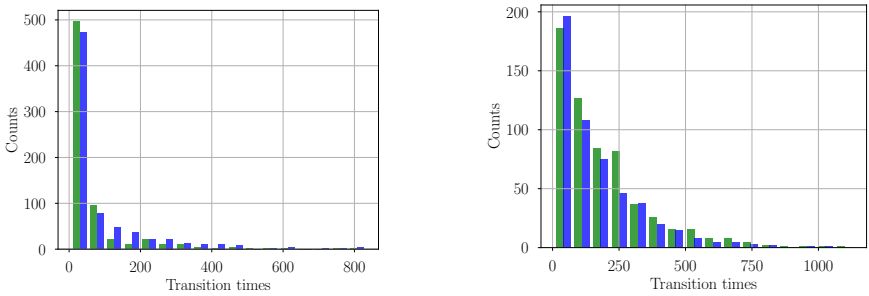


FIGURE 7.5: The distribution of the transition times learned by LED (blue), computed from sampled trajectories, matches the original fine-scale transition times of the MBP dynamics (green). Left: Histogram of $T_{0\to1}$. Mean $T_{0\to1}$ of MD trajectories is 61, mean $T_{0\to1} = 91$ for LED. Right: Histogram of $T_{1\to0}$. Mean $T_{1\to0}$ of MD trajectories is 188, mean $T_{1\to0} = 164$ for LED. LED has learned to propagate the effective dynamics (a 1D latent state z) and capture the non-Markovian effects.

7.3.2 *Trp Cage*

The Trp-cage is considered a prototypical miniprotein for the study of protein folding (Sidky et al., 2020). The protein is simulated with MD (Guzman et al., 2019) with a timestep $\delta t = 1$fs, up to a total time of $T = 100$ns. The

data is sub-sampled at $\Delta t = 0.1$ps, creating a trajectory with $N = 10^6$ samples. The data is divided into 248 sequences of 4000 samples ($T = 400$ps each). The first 96 sequences are used for training (corresponding to 38.4ns), the subsequent 96 sequences for validation, while all the data is used for testing.

The protein positions are transformed into rototranslational invariant features (internal coordinates), composed of bonds, angles, and dihedral angles, leading to a state with dimension $d_s = 456$. LED is trained with a latent space $\boldsymbol{z}_t \in \mathbb{R}^2$, i.e., $d_z = 2$. LED is tested by starting from the initial condition in each of the 248 test sequences, iteratively propagating the latent space to forecast $T = 400$ps. For more information on the hyperparameters of LED, refer to Appendix E.2.1.

The projection of MD trajectory data to LED latent space is illustrated in Figure 7.6 left, in the form of the free energy, i.e., $F = -\kappa_B \mathcal{T} \log p(\boldsymbol{z}_t)$, with $\boldsymbol{z}_t = (\mathbf{z}_1, \mathbf{z}_2)^T \in \mathbb{R}^2$. The free energy on the latent space computed from trajectories sampled from LED is given in Figure 7.6 on the right. LED successfully captures the three metastable states of the Trp Cage miniprotein, while being three orders of magnitude faster than the MD solver. Quantitatively, the two profiles agree up to an error margin of approximately $22.5\kappa_B\mathcal{T}$. In Appendix E.2.2, we provide additional results on the agreement of the marginal state distributions, and realistic samples of the protein configuration sampled from LED in Figure E.2.

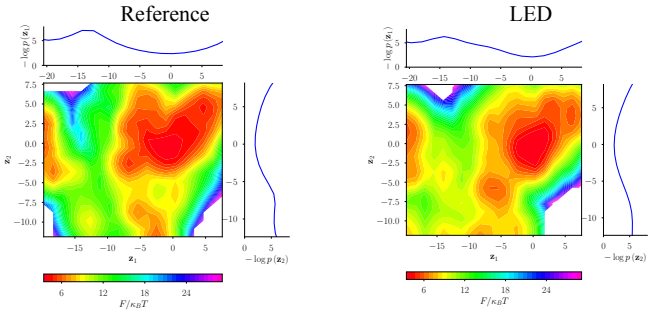


FIGURE 7.6: Free energy projection on the latent space $F = -\kappa_B \mathcal{T} \log p(\boldsymbol{z}_t)$, with $\boldsymbol{z}_t \in \mathbb{R}^2$. Left: MD data projected to the LED latent space. Right: the free energy of trajectories sampled from LED. LED is capturing the free energy profile.

7.3.3 *Alanine Dipeptide*

The alanine dipeptide is often used as the testing ground for enhanced sampling methods (McCarty et al., 2017). LED is evaluated in learning and propagating the dynamics of alanine dipeptide in water. The molecule is simulated with MD (Guzman et al., 2019), and the same data acquisition procedure with Trp-cage in Section 7.3.2 is used ($\delta t = 1$fs, $N = 10^6$, $\Delta t = 0.1$ps, 96 trajectories for training, 96 for validation). LED is tested by starting from the initial condition in each of the total 248 test trajectories, iteratively propagating the latent dynamics to forecast $T = 400$ps. This testing is repeated 5 times with a different random seed, producing data equivalent to $T = 496$ns in total.

The dipeptide positions are transformed into rototranslational invariant features (internal coordinates), composed of bonds, angles, and dihedral angles, leading to a state with dimension $d_s = 24$. In order to demonstrate that LED can uncover the dynamics in a drastically reduced-order latent space, the dimension of the latter is set to one $d_z = 1$, i. e. $z_t \in \mathbb{R}$. For more information on the hyperparameters of LED, refer to Appendix E.3.2.

The metastable states of the dynamics are represented in terms of the energetically favored regions in the state-space of two backbone dihedral angles, ϕ and ψ, i.e., the Ramachandran space (Ramachandran, 1963) plotted in Figure 7.7. Specifically, previous works consider five low-energy clusters, i.e., $\{C5, P_{II}, \alpha_R, \alpha_L, C_7^{ax}\}$. The trained LED is qualitatively reproducing the density in the Ramachandran plot in Figure 7.7, identifying the three dominant low-energy metastable states $\{C5, P_{II}, \alpha_R\}$. LED, however, fails to capture the state density on the less frequently observed states in the training data $\{\alpha_L, C_7^{ax}\}$. The marginal distributions of the trajectories generated by LED match the ground-truth ones (MD data) closely, as depicted in Figure S4.

Even though LED is propagating a 1D latent state, once trained, it can reproduce the statistics while being three orders of magnitude faster than the MD solver.

The free energy is projected to the latent space, i.e., $F = -\kappa_B \mathcal{T} \ln(p(z_t))$, and plotted in Figure 7.8. The free energy projection computed from MD trajectories (train and test data) is compared with the one computed from trajectories sampled from the LED. We estimate the mean free energy profile and the associated standard error of the mean (SEM) using 96 splits of the data. The free energy profile of LED agrees with the reference (test data)

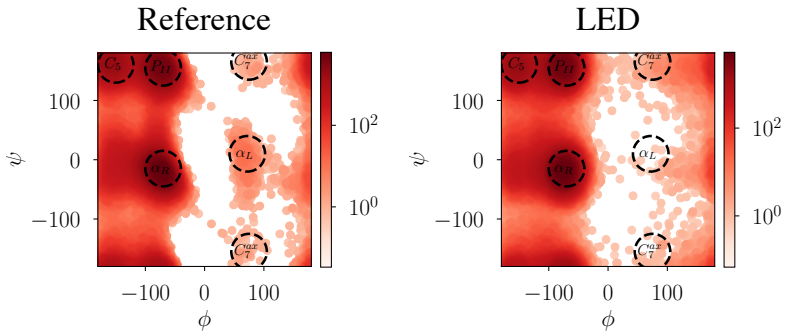


FIGURE 7.7: Ramachandran plot of the alanine dipeptide, i.e., space spanned by two backbone dihedral angles (ϕ, ψ). Scatter plots are colored based on the joint density of (ϕ, ψ). Left: test data. Right: LED trajectories. We observe five energetically favorable metastable states denoted with $\{C5, P_{II}, \alpha_R, \alpha_L, C_7^{ax}\}$. LED captures the three dominant metastable states $\{C5, P_{II}, \alpha_R, \}$. The states $\{\alpha_L, C_7^{ax}\}$ are rarely observed in the training data.

up to an RMSE of $0.25\kappa_B\mathcal{T}$. The profile estimated from the training data exhibits a slightly higher error $1.22\kappa_B\mathcal{T}$, and higher SEM. Note that LED unravels three dominant minima in the latent space. These low-energy regions correspond to metastable states of the dynamics.

The Ramachandran space (ϕ, ψ) is frequently used to describe the long-term behavior and metastable states of the system (Trendelkamp-Schroer et al., 2016; Wehmeyer et al., 2018). The latent encoding of the LED is evaluated based on the mapping between the latent space and the Ramachandran space. Utilizing the MDN decoder, the LED can map the latent state z to the respective rototranslational invariant features (bonds and angles) and regions in the Ramachandran plot. As illustrated in Figure 7.8, the LED is mapping the three low-energy regions in the latent space to the three dominant metastable states in the Ramachandran plot $\{C_5, P_{II}, \alpha_R\}$.

Next, we evaluate LED's effectiveness in unraveling novel protein configurations (state-space) absent from the training data. For this purpose, we create four different small datasets composed of trajectories of the protein, each one not including one of the metastable states $\{C5, P_{II}, \alpha_R, C_7^{ax}\}$. This is done by removing any state closer than 40 degrees to the metastable states' centers. In this way, we guarantee that LED has not seen any state close to the metastable state missing from the data. Note that in this case, the

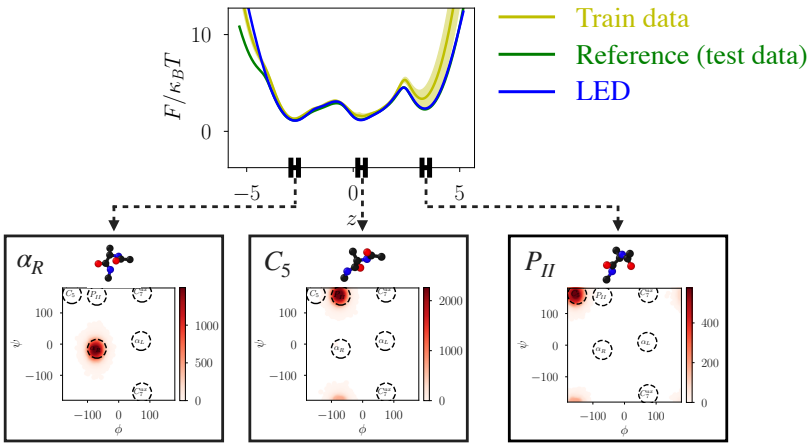


FIGURE 7.8: Plot of the free energy profile projected on the latent state learned by the LED, i.e., $F = -\kappa_B \mathcal{T} \ln p(\boldsymbol{z}_t)$. The latent free energy profile of MD trajectories is compared with the latent free energy profile of trajectories sampled from the LED. The two profiles agree up to an RMSE of $2\kappa_B \mathcal{T}$. Utilizing the LED decoder, the low-energy regions in the latent space (close to the minima) can be mapped to the corresponding protein configurations and metastable states in the Ramachandran plot. The LED uncovers the three dominant metastable states $\{C_5, P_{II}, \alpha_R\}$ in the free energy surface (minima). The LED captures the free energy profile and the dominant metastable states while being computationally three orders of magnitude cheaper than MD.

LED is not trained on a single large MD trajectory but on small trajectories that are not temporally adjacent. We end up with four datasets, each one consisting of approximately 800 trajectories of length $T = 50$ps (500 steps of 0.1 ps). Each dataset covers approximately 40ns protein simulation time. These datasets are created to evaluate the effectiveness of LED in generating truly novel configurations for faster exploration of the state-space. We do not care in this case for accurate reproduction of the statistics due to the minimal data used for training. In Figure 7.9, we plot the Ramachandran plots of the training data along with the ones obtained by analyzing the trajectories of the trained LED models in each of the four cases. We observe that the LED can unravel the metastable states P_{II}, C_5, C_7^{ax}, and α_R, even though they were not part of the training data. However, by removing states

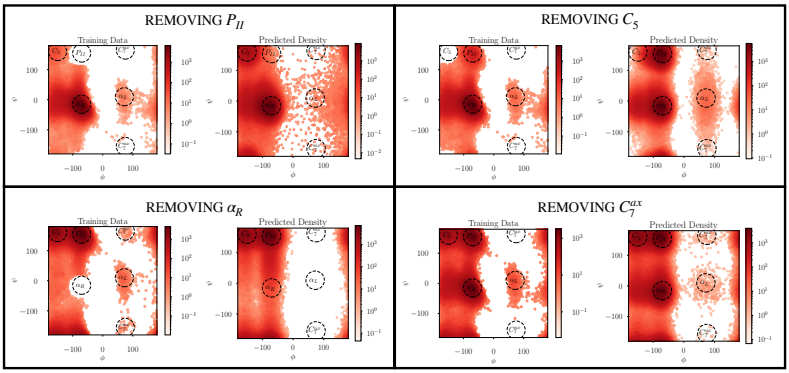


FIGURE 7.9: LED is trained in four scenarios hiding data that lie closer than 40 degrees to one of the metastable states $\{P_{II}, C_5, \alpha_R, C_7^{ax}\}$ each time. LED can successfully generate novel probable configurations close to the metastable states $\{P_{II}, C_5, \alpha_R, C_7^{ax}\}$. Due to the limited training data, however, capturing the state density in the Ramachandran plot is challenging.

that lie close to the metastable state α_R, the LED cannot capture the α_R and C_7^{ax} metastable states. This is because the LED is trained on only a small subset of the training dataset and the transitions to these metastable states are rare.

The dynamics learned by LED are evaluated according to the mean first-passage times (MFPTs) between the dominant metastable states. The MFPT is the average time scale to reach a final metastable state, starting from any initial state. The MFPTs are computed a posteriori from trajectories sampled from the LED and the MD test trajectories, using the PyEMMA software (Scherer et al., 2015). The metastable states considered here are given in the Supporting Information.

As a reference for the MFPTs, we consider an MSM fitted to the MD data (test dataset). The reference MFPTs agree with previous literature (Chekmarev et al., 2004; Jang et al., 2006; Trendelkamp-Schroer et al., 2016; H. Wang et al., 2014). The time-lag of the MSM is set to $\Delta t_{MSM} = 10$ps to ensure the necessary Markovianity of the dynamics. This time-lag is two orders of magnitude larger than the timestep of LED. Fitting an MSM with a time-lag of $\Delta t_{MSM} = 1$ps on the MD data results in very high errors ($\approx 85\%$ on average) in the computation of MFPTs. This emphasizes the

need for non-Markovian models that can reproduce the system's dynamics and statistics independent of the selection of the time-lag.

The MFPTs of trajectories sampled from LED are estimated with an MSM with a time-lag $\Delta t_{MSM} = 10$ps. We consider 1232 trajectories sampled from LED, split them into 32 groups, and report the mean MFPT and the associated standard error of the mean (SEM). Note that the LED is operating on a timestep $\Delta t = 0.1$ps. The MFPTs are identified with a low average relative error of 8.41%. The results on the MFPTs are summarized in Table 7.1. LED captures very well the transitions that are dominant in the data e. g. $T_{P_{II}\to C_5}$, or $T_{\alpha_R\to C_5}$. In contrast, LED exhibits higher MFPT errors in transitions that are less dominant in the training data.

LED identifies the dominant MFPTs successfully by utilizing a very small amount of training data (38.4ns for training and 38.4ns validation) and propagating the latent dynamics on a reduced-order space ($d_z = 1$). LED trajectories are three orders of magnitude cheaper to obtain compared to MD data. As multiple trajectories (here 1232) can be sampled from LED at a fraction of the computational cost of MD, the SEM in the estimation of the MFPTs is small.

At the same time, MSM fitting is a relatively fast procedure once the clustering based on the metastable states is obtained. In contrast, a careless selection of the time-lag in the MSM that fails to render the dynamics Markovian (e. g. $\Delta t = 1$ps) leads to a surrogate model that fails to capture the system time scales. This emphasizes the need to model non-Markovian effects with LED in case of limited data sampled at a high frequency (small timesteps Δt). A more informative selection of the time-lag may alleviate this problem, rendering the dynamics Markovian as in the reference MSM. Still, the consequent sub-sampling of the data can lead to omissions of effects whose time scales are smaller than the time-lag. Consequently, the heuristic selection of the time-lag is rendering the modeling process error-prone. In Appendix E.3.4 we provide additional results on the MFPTs estimated based on metastable state definition in the latent space of LED (without prior knowledge).

If the metastable states are known a priori, or a latent state is available (e. g. from a trained Autoencoder or any other method to unravel collective variables), the MFPTs are represented in, and can be computed directly from the training data, (Table 7.1, 8 splits of the trajectories from the training dataset considered). The SEM, however, is large due to the limited amount of data. We note that LED is not expected to reproduce accurately transitions that are not present in the training data.

MFPT [ns]	**MSM** – 10ps on **MD** data **Reference**	**MSM** – 1ps on **MD** data		**MSM** – 10ps on **MD train** data		**MSM** – 10ps on **LED** – 0.1ps data	
		MFPT	Error (%)	MFPT±SEM	Error (%)	MFPT±SEM	Error (%)
$T_{C_5 \to P_{II}}$	0.112	0.017	84	0.123 ± 0.013	9	0.094±**0.002**	16
$T_{C_5 \to \alpha_R}$	0.096	0.014	86	0.107 ± 0.012	11	0.093±**0.003**	4
$T_{P_{II} \to C_5}$	0.238	0.038	84	0.242 ± 0.021	2	0.218±**0.005**	8
$T_{P_{II} \to \alpha_R}$	0.098	0.014	86	0.109 ± 0.012	11	0.093±**0.003**	5
$T_{\alpha_R \to C_5}$	0.247	0.038	85	0.251 ± 0.021	1	0.232±**0.005**	6
$T_{\alpha_R \to P_{II}}$	0.124	0.018	86	0.134 ± 0.014	8	0.110±**0.002**	11
Average Relative Error			85.01%		6.99%		**8.41%**

TABLE 7.1: Mean first-passage times (MFPTs) between the metastable states of alanine dipeptide in water in [ns]. MFPTs are estimated by fitting MSMs with different time-lags (10ps and 1ps) on trajectories generated by MD, or the LED framework. The average relative error is given for reference.

LED utilizes 76ns of MD data for training, generated with the MD solver in approximately 10 days. The total training time of LED is approximately 20 hours. The trained model can generate 12µs of MD data per day, while with MD we can generate approximately 7.8ns per day. In order to acquire the $T = 496$ns total data used in this case for statistics with MD we would need approximately 64 days. By training the LED (20 hours) on data acquired from MD (10 days) and then sampling multiple trajectories (4 hours) in parallel, we can acquire the same amount of data in 11 days. A realistic speed-up estimation, taking into account data acquisition and training, is thus 3. For a larger protein, this speed-up is expected to be higher.

7.4 DISCUSSION

In this chapter, we extended LED for molecular systems. We presented a data-driven framework to learn and propagate the effective dynamics of molecular systems resulting in dramatically accelerated MD simulations. The LED maximizes the data likelihood for a continuous reduced-order latent representation. The nonlinear dynamics are propagated in the latent space, and the memory effects are captured through the hidden state of the LSTM. Moreover, the method is generative, and the decoder part of the MDN-AE can be employed to sample high-dimensional configurations on any desired time scales. This is in contrast to previous state-of-the-art methods based on the Markovian assumption on the latent state or the minimization of the autocorrelation or the variational loss on the data.

These approaches take into account the error on the long-term equilibrium statistics explicitly to capture the system time scales but suffer from a dependency on the batch-size (Hernández et al., 2018).

The encoder of LED is analogous to the coarse-graining model design, while the decoder is implicitly learning a backmapping to atomistic configurations. The LED automates the dimensionality reduction often associated with the empirical a priori selection of Collective Variables in molecular simulations (Maragliano et al., 2006; Wehmeyer et al., 2018). At the same time, the MDN-LSTM propagates the dynamics on the latent space in a form that is comparable to nonlinear, non-Markovian metadynamics (McCarty et al., 2017).

The effectiveness of LED is demonstrated in simulations of three systems. In Langevin dynamics using BMP, LED recovers the free energy landscape in the latent space, identifies two low-energetic states corresponding to the long-lived metastable states of the potential, and captures the transition times between the metastable states. In the Trp Cage miniprotein, LED captures the free energy projection on the latent space and unravels three metastable states. Finally, for the system of alanine dipeptide in water, LED captures the configuration statistics of the system accurately while being three orders of magnitude faster than MD solvers. Moreover, it identifies three low-energetic regions in the free energy profile projected to the 1D latent state that corresponds to the three dominant metastable states $\{\alpha_R, C_5, P_{II}\}$. LED also captures the dominant mean first-passage times in contrast to the MSM operating on the same time scale, owing to the non-Markovian latent propagation in the latent state with the MDN-LSTM. Furthermore, we showcase how our framework is capable of unraveling novel protein configurations interpolating on the training data.

We note that the speed-up achieved by LED depends on the MD solver used, the dimensionality, and the complexity of the protein under study. While extrapolating estimates to systems not yet tested requires caution, our evidence suggests that the computationally efficient propagation in the latent space of the LED will always provide dramatic accelerations over molecular simulations. Further acceleration can be accomplished by coupling LED and MD solvers in different domain parts for faster exploration of the state-space.

We believe that LED paves the way for faster exploration of the conformational space of molecular systems. More research efforts are needed to target the application of LED to larger proteins and investigate its capabilities in uncovering the metastable states in the free energy profile.

8

CONCLUSION AND OUTLOOK

8.1 CONCLUSIONS

This thesis has focused on developing and applying supervised deep learning methods to spatio-temporal chaotic dynamics and complex multiscale systems, from fluid flows to molecules.

Coupling an LSTM-RNN with a Mean Stochastic Model

Previous approaches on forecasting chaotic dynamics with neural networks (Broomhead et al., 1988; L. Cao et al., 1995; Jaeger et al., 2004; Lapedes et al., 1987; Rico-Martinez et al., 1992) have been limited to low-order systems or assume partial knowledge of the underlying dynamics and are not based solely on data (Pathak, Wikner, et al., 2018). We developed a data-driven method based on LSTM networks to model and forecast high-dimensional dynamical systems. The LSTM predicts the reduced space dynamics (identified by a dimensionality reduction method) of chaotic dynamical systems. The method is applied to the KS equation, the Lorenz 96 system, and a barotropic climate model. The proposed method is benchmarked against GPR, demonstrating higher predictive accuracy and better scaling performance, as it can be trained efficiently with large amounts of data. We argue that the higher short-term predictive accuracy of the LSTM is attributed to its ability to capture nonlinear correlations between modes in the reduced-order space. GPR assumes Gaussian correlations between the modes, and each mode is modeled independently.

Moreover, a challenging problem in forecasting chaotic systems is that densely sampling the attractor can be computationally challenging or impossible (e. g. climate). Data near the attractor boundary can be scarce. The predictions of the LSTM in undersampled regions of the attractor can be wildly inaccurate and lead to high errors (unphysical/unrealistic predictions) in iterative forecasting. We couple the LSTM with a mean stochastic model (MeSM) to tackle this issue. The LSTM learns the local dynamics accurately, while the MeSM captures the global attractor statistics. Blending

LSTM or GPR with MeSM leads to a slight deterioration in the short-term prediction performance, but the steady-state statistical behavior is captured. The hybrid LSTM-MeSM exhibits superior performance than GPR-MeSM in all systems considered in this study.

As the intrinsic attractor dimensionality in the KS is lower than Lorenz 96 in the dynamical regime considered in this work, the LSTM can better capture the local dynamics. Even though the LSTM accurately captures the local dynamics, especially in the chaotic regime $1/\nu = 16$, forecasts of LSTM fly away from the attractor. This effect is alleviated by the coupled LSTM-MeSM method, which makes sure that the error converges to the invariant measure. We show that LSTM and GPR show comparable forecasting accuracy in the barotropic model, and both methods can effectively capture the dynamics.

The proposed framework is entirely data-driven and can be employed in very high-dimensional, fully turbulent dynamical systems, where densely sampling the state space is difficult, extending the applicability of RNNs to challenging forecasting problems.

Training Algorithms and Scalability of RNNs for Dynamical Systems

Although RNNs are powerful algorithms for modeling dynamical systems, fitting these networks to data is difficult due to the vanishing gradients problem (Hochreiter, 1998). In order to tackle this issue, multiple alterations in the architecture of the RNN cell have been proposed like the long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) cell, the Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung, Gulcehre, et al., 2014), and the Unitary recurrent unit (Arjovsky et al., 2016; Jing et al., 2017). These networks are trained with BPTT. A different perspective is adopted in the Reservoir Computing paradigm, where the architecture of the network cell is kept simple, but the training algorithm is changed to least squares regression on the output weights only. The internal weights of the network are fixed, creating a reservoir of dynamics in an attempt to surpass the vanishing gradients problem by design. Previous works on gated architectures trained with BPTT and RC (apart from some very recent works in RC (Pathak, Hunt, et al., 2018; Pathak, Lu, et al., 2017)) have been limited in low-order systems. Moreover, the architectures have not been analyzed adequately and benchmarked extensively in the context of spatio-temporal chaos. We presented a comparative study based on the

efficiency of the networks in capturing temporal patterns and scalability to high-dimensional spatio-temporal dynamics. We distinguish two scenarios, one where the complete state of the system is observed and one where the data manifest only a part of the system state (reduced-order information). The methods are benchmarked in the Lorenz 96 system and the Kuramoto-Sivashinsky equation. We demonstrate that gated architectures exhibit higher forecasting accuracy in the challenging reduced-order scenario, while RC is more prone to divergence of the iterative forecasting error. Moreover, we find that the measures against overfitting in the gated architectures (validation-based early stopping, Zoneout (Krueger et al., 2017), Dropout) are more effective than those used in RC (Tikhonov regularization), as RC models tend to overfit easier. This finding emphasizes the need for more effective mechanisms to guard against overfitting in RC. In the case of full-order information of the state, RC exhibits excellent forecasting accuracy outperforming all other models while reproducing the frequency spectrum of the underlying dynamics.

The scalability of RNNs and RC to very high-dimensional problems is hindered because the required hidden state to capture the dynamics can become large, rendering the training process prohibitively slow or computationally expensive. We tackle this issue by designing a parallelization scheme that exploits the local interactions in the state of a dynamical system. Parallelized RNNs reproduced the long-term attractor climate and the power spectrum of the state in the high-dimensional Lorenz 96 system and the Kuramoto-Sivashinsky equation. Further, we demonstrated that a trained gated architecture can be employed to identify the positive part of the Lyapunov Spectrum of a high-dimensional spatio-temporal chaotic dynamical system accurately in a data-driven manner.

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

Scheduled Autoregressive Backpropagation Through Time

Autoregressive forecasting with RNN architectures suffers from the iterative propagation of the prediction error. An expensive computational alternative is to train a different model for each lead time Rasp et al., 2020, instead

of employing RNNs. This, however, can be prohibitive in practice. Backpropagation through time (BPTT) with teacher-forcing employed to train recurrent architectures in most previous works (Geneva et al., 2020; Kumar et al., 2020; Su et al., 2020; Wiewel et al., 2019) is biased towards a one step ahead prediction loss. We propose a scheduled autoregressive variant of BPTT (BPPT-SA) to alleviate this problem by considering the autoregressive error during training. We benchmark the proposed method against BPTT and a schedule sampling approach in low-dimensional time series problems and the viscous flow past a cylinder in a channel. BPTT-SA shows superior long-term predictive performance without increasing the training cost in ConvRNN and CNN-RNN architectures. The proposed method applies to any recurrent model and can be of great practical help in applications where long-term forecasting is desirable, but retraining one model for each lead time is computationally prohibitive.

Multiscale Simulations of Complex Systems by LED

The scientific importance of multiscale simulations led to the development of multiple potent multiscale algorithms. The scalability of these methods to high-dimensional, complex dynamics has been hindered by the difficulties in the design of accurate data-driven "lifting" operators to recover the full state description from the low-order one, and timestepping routines to capture the nonlinear (coarse-grained) latent dynamics. We augmented the Equation-Free Framework (EFF) with ML models to learn the effective dynamics (LED) and accelerate the simulations of multiscale complex dynamical systems, alleviating the pittfalls of previously proposed approaches and extending the applicability of multiscale algorithms to high-dimensional nonlinear problems.

The efficiency of LED was evaluated in propagating the FitzHugh-Nagumo model dynamics, achieving an order of magnitude lower approximation error compared to state-of-the-art Equation-Free approaches while being two orders of magnitude faster than the micro-scale (Lattice Boltzmann) solver. We demonstrated that the proposed framework identifies the effective dynamics of the Kuramoto-Sivashinsky equation with $L = 22$, capturing the long-term statistics of the attractor and achieving a speed-up of $S \approx 100$. Furthermore, LED accurately captures the long-term dynamics of a flow past a cylinder in $\mathrm{Re} = 100$ and $\mathrm{Re} = 1000$, while being two orders of magnitude faster than the utilized flow solver.

LED identifies and propagates the effective dynamics of dynamical systems with multiple spatio-temporal scales providing significant computational savings. Moreover, the proposed method provides a systematic way of trading between speed-up and accuracy for a multiscale system by switching between the propagation of the latent dynamics and evolution of the original equations, iteratively correcting the statistical error at the cost of reduced speed-up. The proposed methodology can be deployed both in problems described by first principles and for problems where only data are available for the system's macro or micro-scale descriptions.

In summary, LED creates unique algorithmic alloys between data-driven and first principles models and opens new horizons for the accurate and efficient prediction of complex multiscale systems.

Accelerating Molecular Simulations by LED

In this chapter, we couple Mixture Density Networks (MDNs) with LSTMs to model the probabilistic latent space dynamics of molecular systems. In this way, we extended LED for molecular dynamics to alleviate the limitations of previous data-driven Markovian approaches. LED learns and propagates the effective dynamics of molecular systems resulting in dramatically accelerated MD simulations. The proposed method is generative, and the decoder can be employed to sample high-dimensional molecular configurations on any desired time scale. The efficiency and efficacy of LED in capturing the statistics, kinetics, and timescales are demonstrated on three systems, particles moving on the BMP with Langevin dynamics, the dynamics of alanine dipeptide, and the Trp Cage miniprotein. In the first case, LED reproduces the free energy landscape in the latent space, identifies two low-energetic states corresponding to the long-lived metastable states of the potential, and captures the transition times between the metastable states. In the Trp Cage miniprotein, LED reproduces the free energy projection on the latent space and uncovers three metastable states. In alanine dipeptide in water, LED captures the configuration statistics of the system accurately while being three orders of magnitude faster than MD solvers. Moreover, LED identifies three low-energetic regions in the free energy profile projected to the 1D latent state. These regions correspond to the three dominant metastable states $\{\alpha_R, C_5, P_{II}\}$. LED can capture the dominant mean first-passage times in contrast to the MSM operating on the same time scale. This is due to the ability of LED to learn non-Markovian dynamics by the MDN-LSTM on the latent space. Last but not least, we demonstrate that

LED can unravel novel protein configurations interpolating on the training data.

Although extrapolating to dynamical regions undersampled in the training data with LED requires further attention, we demonstrate that the computational savings due to LED's efficient latent space propagation dramatically accelerates molecular simulations. For this reason, we argue that LED paves the way for faster exploration of the conformational space of molecular systems.

8.2 OUTLOOK

RNNs for Chaotic Dynamics

In Chapter 4 we show that both the RNNs and RC cannot capture the zero LEs in the LS of the Kuramoto-Sivashinsky equation. Further investigation is needed on the underlying reasons why the RNNs and RC cannot capture the zero LEs. Another exciting direction could include studying the memory capacity of the networks. This could offer more insight into which architecture and training method is more appropriate for tasks with long-term dependencies. Moreover, coupling the two training approaches may further improve 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 the LSTM weights can be optimized by backpropagation. 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.

Further directions could be initializing 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 backpropagation is also challenging. Another interesting topic is to analyze the influence of the amount of training data and system size on the predictive efficiency of the methods, under the lens of the recently discovered "double descent" curve (Belkin et al., 2019),

supporting that over-parametrized networks (increasing model capacity beyond the point of interpolation) results in improved generalization.

One topic not covered in this work is the invertibility of the models when forecasting the full state dynamics. Non-invertible models like the RNNs may suffer from spurious dynamics that are not part of the training data and the underlying governing equations (Frouzakis et al., 1997; Gicquel et al., 1998). Invertible RNNs may constitute a promising alternative to improve accurate short-term prediction and capture the long-term dynamics.

Another possible research direction is to model the attractor dynamics in the reduced space of a dynamical system using a mixture of LSTM models, one model for each region. The LSTMs proposed in this work model the attractor dynamics globally. However, different attractor regions may exhibit different dynamic behaviors, which cannot be modeled using only one network. These regimes can be uncovered by unsupervised clustering, and a different local model can be trained for each region. Moreover, these local models can be combined with a closure scheme compensating for truncation and modeling errors. This local modeling approach may improve prediction performance significantly.

LED for Experimental Processes and Climate

In this thesis, we demonstrated the application of LED to complex multiscale systems from fluid flows to molecules. Future research efforts need to be concentrated on the extension of the LED methodology to experimental settings and real-world data. We envision how experimental snapshots may inform the latent space dynamics through autoencoders. However, the challenge remains on how to reinitialize the experiments from the decoded microscale description. As a consequence, more research efforts are needed to resolve this. Real-world applications often entail noisy data. The robustness of LED to noise needs to be further investigated. Moreover, LED can be employed to build accurate data-driven climate models by fitting to publicly available data (Rasp et al., 2020).

The methods developed in this thesis can be employed as digital twins of physical objects or processes. Both RNNs and the LED can be trained to mimic the dynamics in real-time of physical systems based on the digital twin paradigm (El Saddik, 2018; Schluse et al., 2018; F. Tao et al., 2019).

Physical systems are often characterized by physical properties, like energy conservation, mass conservation, physical symmetries, etc. Enforcing

these properties by constraining the architecture of the network can improve predictive performance and lead to more physical predictions. Physical properties can also be enforced in a soft way by introducing an auxiliary loss. This method has shown promising results in the climate sciences (Beucler et al., 2021). The design and implementation of methods to enforce analytic constraints representing the system's physical properties is a promising research direction.

Scaling LED for 3D High-dimensional Dynamics

The performance of LED for high-dimensional fluid flows can be improved by introducing geometric priors in the architecture of the AE. Wavelets (Chui et al., 1992) are employed widely to design computational tools for analysis, modeling, and understanding of fluid flows (Farge et al., 1996; Hejazialhosseini et al., 2010). Wavelets have been embedded in neural networks (Alexandridis et al., 2013; Q. Li et al., 2020), showing promising results in hydrology (Partal et al., 2009). Incorporating wavelets in the Autoencoding part of LED can improve predictive ability and scalability to higher-dimensional fluid flows.

In this work, LED has been demonstrated on a high-dimensional 2D flow, i.e., the Navier-Stokes flow past a cylinder. Recent hardware advances alleviate the GPU memory requirement burden for scaling LED to 3D flows. We consider this a promising research direction. Training the RNN of the LED requires a significant amount of computational resources and training time. The utilization of recently proposed alternatives, e. g. Transformer networks Bertasius et al., 2021; Vaswani et al., 2017, might alleviate this burden.

Adaptive and Parametric LED

The LED does not presently contain any mechanism to decide when to upscale the latent space dynamics. In a real-time scenario, if the dynamics of the system change due to an external forcing, which may be known or unknown, the LED cannot detect or adapt to this change. Consequently, the predictions of LED will eventually diverge from the actual physical system. We can alleviate this issue by considering the uncertainty of LED in the latent space predictions and by employing an ensemble of networks (Lakshminarayanan et al., 2016) that provide uncertainty estimates of the predic-

tions and detect samples unseen during training. The training procedure can be adapted to take into account retraining every time a new dynamic regime is detected online. Moreover, we do not expect LED to generalize to dynamical regions drastically different from those represented in the training data. If the dynamics of a system depend on a few parameters, and these are known for each simulation, we can inform the latent space of the LED with these parameters and generalize to unseen dynamical regimes.

Another exciting direction is the extension of LED to fluid flow prediction of varying geometries and shapes. The LED can be trained on large datasets to predict the fluid flow around any object. The object geometry can be provided as an additional input to the network. In this way, the LED will generalize and predict flows around shapes unseen during training.

LED for Realistic Molecular Systems

In the context of molecular simulations, LED can be coupled with an MD solver in different parts of the domain for faster exploration of the state-space. Currently, the scalability of the LED to large proteins is challenging due to the fact that the probability distribution of the protein configuration is difficult to model. The protein can be described with its coordinates (x, y, and z coordinates of the molecules), its internal coordinates (e.g., angles, bond lengths, etc.), or as a graph. Modeling the protein with its coordinates poses significant challenges and was proven problematic in practice because the protein structure is not embedded in the data, and training a probabilistic decoder to sample realistic configurations is hard. This thesis investigated the second approach, representing the protein with its internal coordinates. This encouraged the MDN decoder to sample realistic configurations. However, training the MDN decoder is challenging, as the description is multi-modal. On the one hand, angles are circular variables that can be modeled with a Von-Mishes distribution. On the other hand, bond lengths are positive real variables modeled with a log-normal distribution (or a normal distribution along with an invertible non-linearity mapping to positive values, as employed in this work). Designing and training a multi-modal probability density model remains an under-explored area of research.

A third option is to model the protein as a graph (Winter et al., 2021). This direction is encouraging, primarily due to the development of Graph Neural Networks (GNNs) (Scarselli et al., 2008), and Graph Mixture Den-

sity Networks (Errica et al., 2021). GNNs have the potential to alleviate the problems mentioned above (Z. Li et al., 2021). LED can be extended with autoencoding GNNs to identify the latent space and a probabilistic model to sample a graph from the latent (coarse-grained) description. This approach is promising to scale LED to high-dimensional realistic protein configuration.

Moreover, further acceleration can be accomplished by coupling LED and MD solver in different domain parts for faster exploration of the state-space. More research efforts are needed to target the application of LED to larger proteins and investigate its capabilities in uncovering the metastable states in the free energy profile.

LED for Surrogate Environment Modeling in Reinforcement Learning

Much of the recent success of ML has been achieved by Reinforcement Learning (Andrychowicz et al., 2020; Jumper et al., 2021; V. Mnih et al., 2015; Silver et al., 2016) methods that interact actively with an environment (e.g., a physical system, a dynamical system, or a game). These successes required extraordinary advancements in both algorithms, high-performance computing architectures, and hardware to produce the vast amounts of data required to train the RL agents. RL methods are employed to tackle real world problems from health-care (Chakraborty et al., 2014), traffic management (Mannion et al., 2016), electric power grids (Z. Zhang et al., 2019), and finance (Deng et al., 2016). In many practical applications (driving cars, fluid flows, physical processes), the computational burden associated with simulating the environment in RL is prohibitive. In fluid dynamics, recent works employ advanced algorithms (Novati and Koumoutsakos, 2019) that manage the replay memory (by remembering and forgetting experiences) to tackle computationally expensive RL problems (Bae et al., 2021; Mandralis et al., 2021; Novati, de Laroussilhe, et al., 2021), by improving data efficiency. An overview can be found in (Brunton, Noack, et al., 2020). In Novati, de Laroussilhe, et al., 2021, RL is employed to identify neural closure models for isotropic turbulence, generalizing across grid sizes and Reynolds numbers. The LED can be employed for surrogate modeling in model-based RL, mimicking the environment dynamics to alleviate the computational cost further and improve data efficiency.

A

COUPLING AN LSTM-RNN AND A MEAN STOCHASTIC MODEL

A.1 METHODS

A.1.1 *Training and Inference*

In this section, the LSTM training procedure is explained in detail. We assume that time series data stemming from a dynamical system is available in the form $D = \{z_{t:N}, \dot{z}_{t:N}\}$, where $z_t \in \mathbb{R}^{d_z}$ is the state at timestep t and $\dot{z}_t$ is the derivative. The available time series data are divided into two separate sets, the training dataset and the validation dataset, i. e. z_t^{train}, $\dot{z}_t^{train}$, $t \in \{1, \cdots, N_{train}\}$, and z_t^{val}, $\dot{z}_t^{val}$, $t \in \{1, \cdots, N_{val}\}$. N_{train} and N_{val} are the number of training and validation samples, respectively. We set the ratio to $N_{train}/N = 0.8$. This data is stacked as

$$\underbrace{\boldsymbol{i}_t^{train} = \begin{pmatrix} \boldsymbol{z}_{t+\kappa_2-1}^{train} \\ \boldsymbol{z}_{t+\kappa_2-2}^{train} \\ \vdots \\ \boldsymbol{z}_{t}^{train} \end{pmatrix}}_{\text{Input stack}}, \quad \underbrace{\boldsymbol{o}_t^{train} = \dot{\boldsymbol{z}}_{t+\kappa_2-1}^{train}}_{\text{Output stack}}, \tag{A.1}$$

for $t \in \{1, 2, \ldots, N_{train} - \kappa_2 + 1\}$, in order to form the training (and validation) input and output of the LSTM. These training samples are used to optimize the parameters of the LSTM (weights and biases) in order to learn the mapping $\boldsymbol{i}_t \rightarrow \boldsymbol{o}_t$. The loss function of each sample is

$$\mathcal{L}_{sample}(\boldsymbol{i}_t^{train}, \boldsymbol{o}_t^{train}, w) = ||\mathcal{F}^w(\underbrace{\boldsymbol{z}_{t:t-\kappa_2+1}^{train}}_{\boldsymbol{i}_t^{train}}) - \boldsymbol{o}_t^{train}||^2, \tag{A.2}$$

while the total Loss is defined as

$$\mathcal{L}(D, w) = \frac{1}{S} \sum_{b=1}^{S} \mathcal{L}(\boldsymbol{i}_b^{train}, \boldsymbol{o}_b^{train}, w), \tag{A.3}$$

where $S = N_{train} - \kappa_2 + 1$ is the total number of samples. These samples can be further stacked together as batches of size B, with the loss of the batch defined as the mean loss of the samples belonging to the batch. Using only one sample for the loss gradient estimation may lead to noisy gradient estimates and slow convergence. Mini-batch optimization tackles this issue.

At the beginning of the training, the weights are randomly initialized to w^0 using Xavier initialization. We also tried other initialization methods like drawing initial weights from random normal distributions or initializing them to constant values, but they often led to saturation of the activation functions, especially for architectures with higher backpropagation horizon κ_2. The training proceeds by optimizing the network weights iteratively for each batch. In order to perform this optimization step, a gradient descent optimizer can be used

$$w^{i+1} = w^i - \eta \nabla_w \mathcal{L}(\boldsymbol{i}_t^{train}, \boldsymbol{o}_t^{train}, w^i), \tag{A.4}$$

where η is the step-size parameter, w^i are the weights before optimizing the batch i and w^{i+1} are the updated weights. Plain gradient descent optimization suffers from slow convergence in practice and convergence to local sub-optimal solutions. This approach is especially not well-suited for high-dimensional problems in deep learning, where the number of parameters (weights) to be optimized lie in a high-dimensional manifold with many local optima. Sparse gradients stemming from the mini-batch-optimization also lead to slow convergence as previously computed gradients are ignored. Recent advances in stochastic optimization led to the invention of adaptive schemes that efficiently cope with the problems mentioned above.

We used the Adam stochastic optimization method. Adam exploits previously computed gradients using moments. The weights are initialized to w^0 and the moment vectors to m_1^0 and m_2^0. At each step the updates in the Adam optimizer are

$$\begin{aligned}
g &= \nabla_w \mathcal{L}(\boldsymbol{i}_t^{train}, \boldsymbol{o}_t^{train}, w^i) \\
m_1^{i+1} &= \beta_1 m_1^i + (1-\beta_1)\, g \\
m_2^{i+1} &= \beta_2 m_2^i + (1-\beta_2)\, g^2 \\
\hat{m}_1 &= m_1^{i+1} / (1-\beta_1^i) \\
\hat{m}_2 &= m_2^{i+1} / (1-\beta_2^i) \\
w_{i+1} &= w_i - \eta\, \hat{m}_1 / (\sqrt{\hat{m}_2} + \varepsilon),
\end{aligned} \tag{A.5}$$

where $\beta_1, \beta_2, \varepsilon$, and η are hyperparameters, g^2 is the point-wise square of the gradient and β_1^i is the parameter β_1 in the i^{th} power, where i is the iteration number. After updating the weights using the Adam optimizer, described in Equation A.5, for every batch, a "training epoch" is completed. Many such epochs are performed until the total training loss reaches a plateau. After each epoch, the loss is also evaluated in the validation data set in order to avoid overfitting. The validation loss is used as a proxy of the generalization error. The training is stopped when the validation error is not decreasing for 30 consecutive epochs, or the maximum of 1000 epochs is reached. In our work we used $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 1e-8$. We found that our results were robust towards the selection of these hyperparameters. A higher initial learning rate $\eta = 0.001$ was used to speed up convergence. The models are then refined with $\eta = 0.0001$.

A.1.2 *Weighting the Loss Function*

In the training procedure described above the loss function for each sample is given by

$$\mathcal{L}_{sample}(\boldsymbol{i}_t, \boldsymbol{o}_t, w) = ||\mathcal{F}^w(\underbrace{\boldsymbol{z}_{t:t-\kappa_2+1}}_{\boldsymbol{i}_t}) - \boldsymbol{o}_t||^2. \tag{A.6}$$

However, the neural network output $\mathcal{F}^w$ is a multi-dimensional vector and represents a prediction of the derivative of the reduced-order state of a dynamical system. In a dynamical system, specific reduced-order states are more important than others as they may explain a more significant portion of the total energy. This importance can be introduced in the loss function by assigning different weights in different neural network outputs. The loss of each sample takes then the following form

$$\mathcal{L}_{sample}(\boldsymbol{i}_t^j, \boldsymbol{o}_t^j, w) = \frac{1}{d_o}\sum_{j=1}^{d_o} w_j \Big(\mathcal{F}^w(\underbrace{\boldsymbol{z}_{t:t-\kappa_2+1}^j}_{\boldsymbol{i}_t^j}) - \boldsymbol{o}_t^j\Big)^2, \tag{A.7}$$

where d_o is the output dimension and weights w_j are selected according to the significance of each output component, e. g. energy of each component in the physical system.

A.1.3 *LSTM Architecture*

An RNN unfolded κ_2 temporal timesteps in the past is illustrated in Figure A.1. The following discussion on "stateless" and "stateful" RNNs generalizes to LSTMs, with the only difference that the hidden state consists of $\boldsymbol{h}_t, \boldsymbol{c}_t$ instead of solely $\boldsymbol{h}_t$ and the functions coupling the hidden states with the input as well as the output with the hidden states are more complicated.

In "stateless" RNNs the hidden states at the truncation layer κ_2, $\boldsymbol{h}_{t-\kappa_2}$ are initialized always with 0. As a consequence, $\boldsymbol{o} = \mathcal{F}^w(\boldsymbol{i}_{t:t-\kappa_2+1})$ and only the short-term history is used to perform a prediction. The only difference when using LSTM cells is that the function $\mathcal{F}^w$ has a more complex structure and additionally $\boldsymbol{h}_{t-\kappa_2}, \boldsymbol{c}_{t-\kappa_2} = 0$.

In contrast, in "stateful" RNNs the states $\boldsymbol{h}_{t-\kappa_2} \neq 0$. In this case, these states can be initialized by "teacher-forcing" the RNN using data from a long history in the past. For example, assuming $\boldsymbol{i}_{t-\kappa_2:t-2d+1}$ is known, we can set $\boldsymbol{h}_{t-2d} = 0$, and compute $\boldsymbol{h}_{t-\kappa_2}$ using the given history $\boldsymbol{i}_{t-\kappa_2:t-2d+1}$ ignoring the outputs. This value can then be used to predict

$$\boldsymbol{o}_t = \mathcal{F}^w(\boldsymbol{i}_{t:t-\kappa_2+1}, \boldsymbol{h}_{t-\kappa_2}) \tag{A.8}$$

as in Figure A.1. This approach has two disadvantages.

- In order to forecast starting from various initial conditions, even with "teacher-forcing", some initialization of the hidden states is imperative. This initialization introduces additional error, which is not the case for the "stateless" RNN.
- In the "stateful" RNN a longer history needs to be known in order to initialize the hidden states with "teacher-forcing." Even though more data needs to be available, we did not observe any prediction accuracy improvement in the cases considered. This follows from the chaotic nature of the systems, as information longer than some timesteps in the past are irrelevant for the prediction.

A.2 BAROTROPIC MODEL

In this section, we describe the method used to reduce the dimensionality of the Barotropic climate model. First, the original problem dimension of 231 is reduced using a generalized version of the classical multi-dimensional

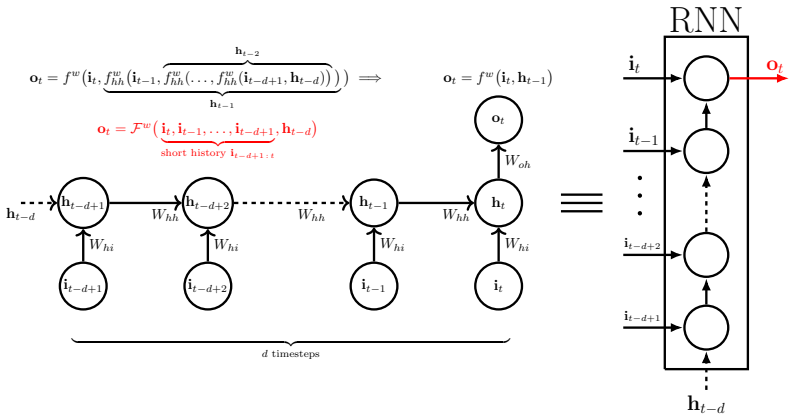


FIGURE A.1: An RNN unfolded κ_2 timesteps. In mathematical terms, unfolding is equivalent with iteratively applying f^w_{hh} to $\mathbf{h}_{t-\kappa_2}$ and finally feeding the result to the output function f^w. The output of the RNN is thus a function of the κ_2 previous inputs $\boldsymbol{i}_{t:t-\kappa_2+1}$ and the initialization of the hidden states $\mathbf{h}_{t-\kappa_2}$. This function is denoted with $\mathcal{F}^w$. For the RNN the hidden state mapping has the simple form $f^w_{hh}(\mathbf{i}_t, \mathbf{h}_{t-1}) = \mathrm{act}_h(W_{hi}\mathbf{i}_t + W_{hh}\mathbf{h}_{t-1})$, while the output mapping is $f^w(\mathbf{i}_t, \mathbf{h}_{t-1}) = \sigma_o(W_{oh}\mathbf{h}_t) = \sigma_o\big(W_{oh} f^w_{hh}(\mathbf{i}_t, \mathbf{h}_{t-1})\big)$. The same argumentation holds for LSTM, though the form of f^w_{hh}, f^w and $\mathcal{F}^w$ are more complicated.

scaling method. Then, we construct Empirical Orthogonal Functions (EOFs) that form an orthogonal basis of the reduced-order space. The dynamics are projected to the EOFs.

The classical multi-dimensional scaling procedure tries to identify an embedding with a lower dimensionality such that the pairwise inner products of the dataset are preserved. Assuming that the dataset consists of points $\boldsymbol{\zeta}_i$, $i \in \{1, \ldots, N\}$, whose reduced-order representation is denoted with $\boldsymbol{y}_i$, the procedure is equivalent with the solution of the following optimization problem

$$\underset{\boldsymbol{y}_1, \cdots, \boldsymbol{y}_N}{\text{minimize}} \sum_{i<j} \big(\langle \boldsymbol{\zeta}_i, \boldsymbol{\zeta}_j \rangle_{\boldsymbol{\zeta}} - \langle \boldsymbol{y}_i, \boldsymbol{y}_j \rangle_{\boldsymbol{y}}\big)^2, \tag{A.9}$$

where $\langle \cdot, \cdot \rangle_{\boldsymbol{\zeta}}$, and $\langle \cdot, \cdot \rangle_{\boldsymbol{y}}$ denote some well defined inner product of the original space $\boldsymbol{\zeta}$ and the embedding space $\boldsymbol{y}$ respectively. The problem given in (A.9) minimizes the total squared error between pairwise products. In case both products are scalar products, the solution of (A.9) is equivalent with PCA. Assuming only $\langle \cdot, \cdot \rangle_{\boldsymbol{y}}$ is the scalar product, problem (A.9) also

accepts an analytic solution. Let $W_{ij} = \langle \boldsymbol{\zeta}_i, \boldsymbol{\zeta}_j \rangle_\zeta$ be the coefficients of the Gram matrix, $|k_1| \geq |k_2| \geq \cdots \geq |k_N|$ its eigenvalues sorted in descending absolute value and $\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_N$ the respective eigenvectors. The optimal d_z-dimensional embedding for a point $\boldsymbol{\zeta}_n$ is given by

$$\boldsymbol{y}_n = \begin{pmatrix} k_1^{1/2} \boldsymbol{U}_1^n \\ k_2^{1/2} \boldsymbol{U}_2^n \\ \vdots \\ k_{d_z}^{1/2} \boldsymbol{U}_{\kappa_2}^n \end{pmatrix}, \tag{A.10}$$

where $\boldsymbol{U}_m^n$ denotes the n^{th} component of the m^{th} eigenvector. The optimality of Equation A.10 can be proven by the Eckart-Young-Mirsky theorem, as problem Equation A.9 is equivalent with finding the best d_z rank approximation in the Frobenius norm. In our problem, the standard kinetic energy product is used to preserve the nonlinear symmetries of the system dynamics:

$$\langle \boldsymbol{\zeta}_i, \boldsymbol{\zeta}_j \rangle_\zeta = \int_{\mathcal{S}} \nabla \psi_i \cdot \nabla \psi_j d\,\mathcal{S} = -\int_{\mathcal{S}} \zeta_i \psi_j d\,\mathcal{S} = -\int_{\mathcal{S}} \zeta_j \psi_i d\,\mathcal{S}, \tag{A.11}$$

where the last identities are derived using partial integration and the fact that $\boldsymbol{\zeta} = \Delta \boldsymbol{y}$.

The solution in Equation A.10 is only optimal w. r. t. the N training data points used to construct the Gram matrix. In order to calculate the embedding for a new point, it is convenient to compute the EOFs, which form an orthogonal basis of the reduced-order space $\boldsymbol{y}$. The EOFs are given by

$$\boldsymbol{\phi}_m = \sum_{n=1}^{N} k_m^{-1/2} \boldsymbol{U}_m^n \boldsymbol{\zeta}_n, \tag{A.12}$$

where m runs from 1 to d_z. The EOFs are sorted in descending order according to their energy level. The first four EOFs are plotted in Figure A.2. EOF analysis has been used to identify individual realistic climatic modes such as the Arctic Oscillation (AO) known as teleconnections. The first EOF is characterized by a center of action over the Arctic surrounded by a zonal symmetric structure in mid-latitudes. This pattern resembles the Arctic Oscillation/Northern Hemisphere Annular Mode (AO/NAM) and explains approximately 13.5% of the total energy. The second, third, and fourth EOFs are quantitatively very similar to the East Atlantic/West Russia, the Pacific/North America (PNA), and the Tropical/Northern Hemisphere (TNH)

patterns end account for 11.4%, 10.4% and 7.1% of the total energy respectively. Since these EOFs feature realistic climate teleconnections, performing accurate predictions is of high practical importance.

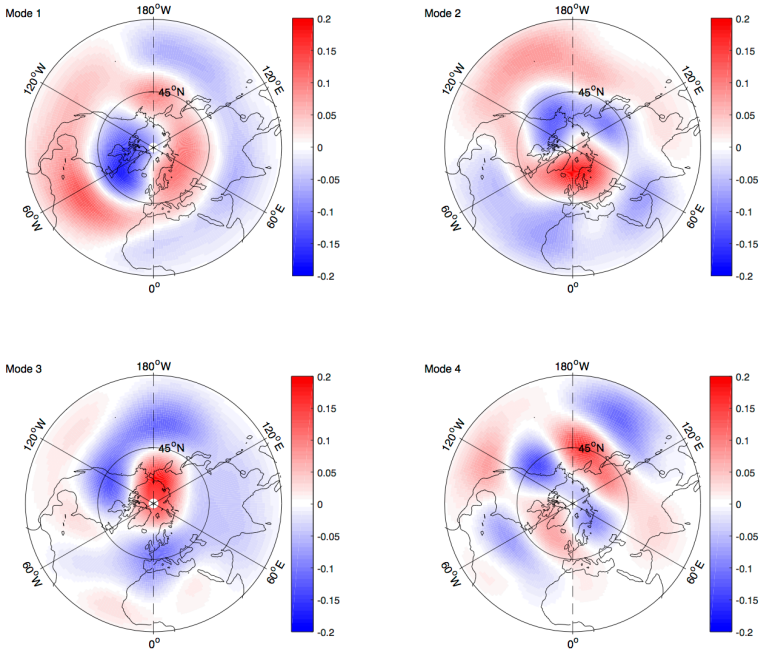


FIGURE A.2: The four most energetic empirical orthogonal functions of the barotropic model

As a consequence of the orthogonality of the EOFs w. r. t. the kinetic energy product, the reduced representation $\boldsymbol{y}^*$ of a new state $\boldsymbol{\zeta}^*$ can be recovered from

$$\boldsymbol{y}^* = \begin{pmatrix} \langle \boldsymbol{\zeta}^*, \boldsymbol{\phi}_1 \rangle_\zeta \\ \langle \boldsymbol{\zeta}^*, \boldsymbol{\phi}_2 \rangle_\zeta \\ \vdots \\ \langle \boldsymbol{\zeta}^*, \boldsymbol{\phi}_{d_z} \rangle_\zeta \end{pmatrix}. \tag{A.13}$$

Note that only the d_z coefficients that correspond to the most energetic EOFs from the reduced-order state $\boldsymbol{y}^*$ are considered. In essence, the EOFs

act as an orthogonal basis of the reduced-order space, and the state obtained from classical multi-dimensional scaling ζ^* is projected to this basis.

B

RECURRENT NEURAL NETWORKS

B.1 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 $\boldsymbol{h} \in \mathbb{R}^{d_h}$, by teacher forcing the RC network with ground-truth data from the system for d_N timesteps and stacking the evolution of the hidden state in a single matrix, we end up with matrix $\mathbf{H} \in \mathbb{R}^{d_N \times d_h}$. Moreover, by stacking the target values, which are the input data shifted by one timestep, we end up in the target matrix $\boldsymbol{y} \in \mathbb{R}^{d_N \times d_o}$, where d_o is the dimension of the observable we are predicting. In order to identify the output weights $W_{out} \in \mathbb{R}^{d_o \times d_h}$, we need to solve the linear system of $d_N \cdot d_o$ equations

$$\mathbf{H} W_{out}^T = \boldsymbol{y}. \tag{B.1}$$

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

$$W_{out} = \underbrace{\boldsymbol{y}^T \mathbf{H}}_{\overline{\boldsymbol{y}}} \left(\underbrace{\mathbf{H}^T \mathbf{H}}_{\overline{\mathbf{H}}} + \widetilde{\eta} \mathbf{I} \right)^{-1} \tag{B.2}$$

where $\widetilde{\eta}$ is the Tikhonov regularization parameter and $\mathbf{I}$ the unit matrix. In our case d_N is of the order of 10^5 and $d_N >> d_h$. To reduce the memory requirements of the training method, we compute the matrices $\overline{\mathbf{H}} = \mathbf{H}^T \mathbf{H} \in \mathbb{R}^{d_h \times d_h}$ and $\overline{\boldsymbol{y}} = \boldsymbol{y}^T \mathbf{H} \in \mathbb{R}^{d_o \times d_h}$ in a time-batched schedule.

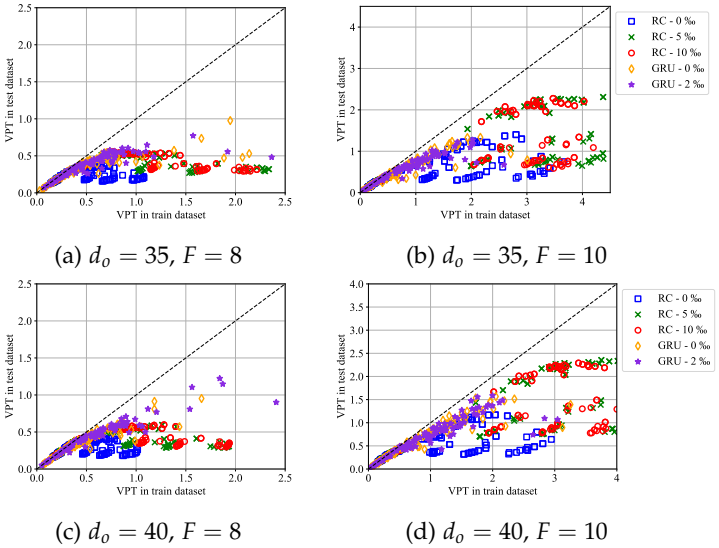
Specifically, we initialize $\overline{\boldsymbol{y}} = \mathbf{0}$ and $\overline{\mathbf{H}} = \mathbf{0}$. Then every d_n timesteps with $d_n << d_N$, we compute the batch matrix $\overline{\mathbf{H}_b} = \mathbf{H}_b^T \mathbf{H}_b \in \mathbb{R}^{d_h \times d_h}$, where $\mathbf{H}_b \in \mathbb{R}^{d_n \times d_h}$ is formed by the stacking the hidden state only for the last d_n timesteps. In the same way, we compute $\overline{\boldsymbol{y}_b} = \boldsymbol{y}_b^T \mathbf{H}_b \in \mathbb{R}^{d_o \times d_h}$, where $\boldsymbol{y}_b \in \mathbb{R}^{d_n \times d_o}$ is formed by the stacking of the target data for the last d_n timesteps. After every batch computation we update our beliefs with $\overline{\mathbf{H}} \leftarrow \overline{\mathbf{H}} + \overline{\mathbf{H}_b}$ and $\overline{\boldsymbol{y}} \leftarrow \overline{\boldsymbol{y}} + \overline{\boldsymbol{y}_b}$.

In addition, we also experimented with two alternative solvers for the linear system Equation B.1 in the Lorenz 96. We tried a dedicated regularized

least-squares routine utilizing an iterative procedure (scipy.sparse.linalg.lsqr) 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 the Lorenz 96 model, 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 equation.

B.2 REGULARIZING TRAINING WITH NOISE

Here, we investigate the effect of noise to the training data. In Figure B.1, 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 datasets. Moreover, the effect of divergent predictions by iterative forecasts is alleviated significantly. In contrast, adding noise does not seem to have a significant impact on the performance of GRU models.



(a) $d_o = 35$, $F = 8$ (b) $d_o = 35$, $F = 10$

(c) $d_o = 40$, $F = 8$ (d) $d_o = 40$, $F = 10$

FIGURE B.1: VPT in the testing data plotted against VPT in the training data for RC and GRU models trained with added noise of different levels in the data. The noise only slightly varies the forecasting efficiency in GRU networks. In contrast, the effectiveness of RC in forecasting the full-order system is increased as depicted in plots (b) and (d).

B.3 DIMENSIONALITY REDUCTION WITH SINGULAR VALUE DECOMPOSITION

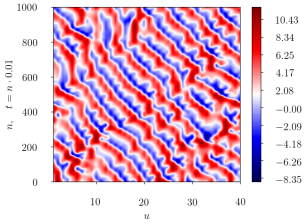
Singular Value Decomposition (SVD) can be utilized to perform dimensionality reduction in a dataset by identifying the modes that capture the highest variance in the data and then performing a projection on these modes. Assuming that a data matrix is given by stacking the time-evolution of a state $\boldsymbol{U} \in \mathbb{D}$ as $U = [\boldsymbol{U}_1, \boldsymbol{U}_2, \ldots, \boldsymbol{U}_N]$, where the index N is the number of data samples. By subtracting the temporal mean $\overline{\boldsymbol{U}}$ and stacking the data, we end up with the data matrix $\boldsymbol{U} \in \mathbb{R}^{T \times D}$. Performing SVD on $\boldsymbol{U}$ leads to

$$\boldsymbol{U} = \boldsymbol{M}\boldsymbol{\Sigma}\boldsymbol{V}^T, \quad \boldsymbol{M} \in \mathbb{R}^{N \times N}, \quad \boldsymbol{\Sigma} \in \mathbb{R}^{N \times D}, \quad \boldsymbol{V} \in \mathbb{R}^{D \times D}, \tag{B.3}$$

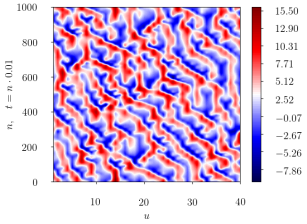
with Σ diagonal, with descending diagonal elements. The columns of matrix V are considered the modes of the SVD, while the square D singular values of $\boldsymbol{\Sigma}$ correspond to the data variance explained by these modes. This variance is also referred to as energy. In order to calculate the 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 $d_r < D$. Then we identify the eigenvectors corresponding to the most high-energetic eigenmodes. These are given by the first columns $\mathbf{V_r}$ of $\boldsymbol{V}$, i.e., $\boldsymbol{V} = [\mathbf{V_r}, \mathbf{V_{-r}}]$. We discard the low-energetic modes $\mathbf{V_{-r}}$. The dimension of the truncated eigenvector matrix is $\mathbf{V_r} \in \mathbb{R}^{D \times d_r}$. In order to reduce the dimensionality of the dataset, each vector $\boldsymbol{U} \in \mathbb{D}$ is projected to $\boldsymbol{U}_r \in d_r$ by

$$\mathbf{c} = \mathbf{V_r}^T \boldsymbol{U}, \quad \mathbf{c} \in \mathbb{R}^{d_r}. \tag{B.4}$$

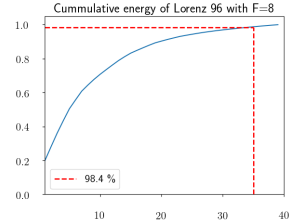
In the Lorenz 96 model, we construct a reduced-order observable with $d_o = 35$ modes of the system. The cumulative energy distribution, along with a contour plot of the state and the mode evolution, is illustrated in Figure B.2.



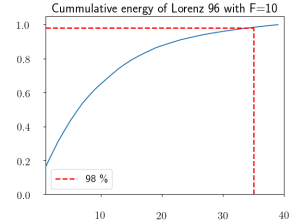
(a) State evolution $F = 8$



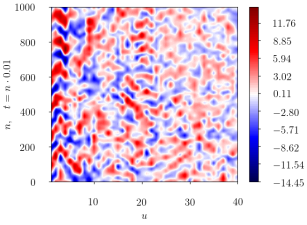
(b) State evolution $F = 10$



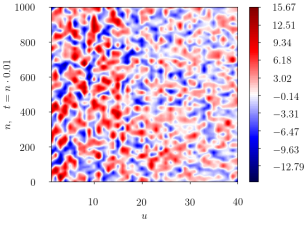
(c) Energy spectrum $F = 8$



(d) Energy spectrum $F = 10$



(e) SVD mode evolution $F = 8$



(f) SVD mode evolution $F = 10$

FIGURE B.2: Energy spectrum of Lorenz 96

Hyperparameter	Explanation	Values
D_r	reservoir size	{6000, 9000, 12000, 18000}
N	training data samples	10^5
	Solver	Pseudoinverse/LSQR/Gradient descent
d	degree of $W_{h,h}$	{3, 8}
ϱ	radius of $W_{h,h}$	{0.4, 0.8, 0.9, 0.99}
ω	input scaling	{0.1, 0.5, 1.0, 1.5, 2.0}
$\widetilde{\eta}$	regularization	$\{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$
d_o	observed state dimension	{35, 40}
n_w	warm-up steps (testing)	2000
κ_n	noise level in data	{0, 0.5%, 1%}

TABLE B.1: Hyperparameters of **RC** for Lorenz 96

Hyperparameter	Explanation	Values
d_h	hidden state size	{1, 2, 3} layers of {500, 1000, 1500}
N	training data samples	10^5
B	batch-size	32
κ_1	BPTT forward timesteps	{1, 8}
κ_2	BPTT truncated backprop. length	{8, 16}
κ_3	BPTT skip gradient parameter	$= \kappa_2 + \kappa_1 - 1$
$\widetilde{\eta}$	initial learning rate	10^{-3}
p	zoneout probability	{0.99, 0.995 1.0}
d_o	observed state dimension	{35, 40}
n_w	warm-up steps (testing)	2000
κ_n	noise level in data	{0, 0.2%}

TABLE B.2: Hyperparameters of **GRU/LSTM** for Lorenz 96

B.4 HYPERPARAMETERS

For the Lorenz 96 model space with $d_o \in \{35, 40\}$ (in the PCA mode), we used the hyperparameters reported on Table B.1 for RC and Table B.2 for GRU/LSTM models. For the parallel architectures in the state-space of Lorenz 96 the hyperparameters are reported on Table B.4 and Table B.5 for the parallel RC and GRU/LSTM models respectively. For the parallel architectures in the state-space of the Kuramoto-Sivashinsky architecture the hyperparameters are reported on Table B.6 and Table B.7 for the parallel RC and GRU/LSTM models respectively. We note here that in all RNN methods, the optimizer used to update the network can also be optimized. To alleviate the computational burden, we stick to Adam.

Hyperparameter	Explanation	Values
d_h	hidden state size	$\{1, 2, 3\}$ layers of $\{500, 1000, 1500\}$
N	training data samples	10^5
B	batch-size	32
κ_1	BPTT forward timesteps	$\{1, 8\}$
κ_2	BPTT truncated backprop. length	$\{8, 16\}$
κ_3	BPTT skip gradient parameter	$= \kappa_2 + \kappa_1 - 1$
$\widetilde{\eta}$	initial learning rate	10^{-3}
p	zoneout probability	1.0
d_o	observed state dimension	$\{35, 40\}$
n_w	warm-up steps (testing)	2000
κ_n	noise level in data	$\{0, 0.2\%\}$

TABLE B.3: Hyperparameters of **Unitary Evolution** networks for Lorenz 96

Hyperparameter	Explanation	Values
D_r	reservoir size	$\{1000, 3000, 6000, 12000\}$
N_g	number of groups	20
G	group size	2
I	interaction length	4
N	training data samples	10^5
	Solver	Pseudoinverse/LSQR/Gradient descent
d	degree of $W_{h,h}$	10
ϱ	radius of $W_{h,h}$	0.6
ω	input scaling	0.5
$\widetilde{\eta}$	regularization	10^{-6}
d_o	observed state dimension	40
n_w	warm-up steps (testing)	2000

TABLE B.4: Hyperparameters of **Parallel RC** for Lorenz 96

Hyperparameter	Explanation	Values
d_h	hidden state size	{100, 250, 500}
N_g	number of groups	20
G	group size	2
I	interaction length	4
N	training data samples	10^5
B	batch-size	32
κ_1	BPTT forward timesteps	4
κ_2	BPTT truncated backprop. length	4
κ_3	BPTT skip gradient parameter	4
$\widetilde{\eta}$	initial learning rate	10^{-3}
p	zoneout probability	{0.998, 1.0}
d_o	observed state dimension	40
n_w	warm-up steps (testing)	2000

TABLE B.5: Hyperparameters of **Parallel GRU/LSTM** for Lorenz 96

Hyperparameter	Explanation	Values
D_r	reservoir size	{500, 1000, 3000, 6000, 12000}
N_g	number of groups	64
G	group size	8
I	interaction length	8
N	training data samples	10^5
	Solver	Pseudoinverse
d	degree of $W_{h,h}$	10
ϱ	radius of $W_{h,h}$	0.6
ω	input scaling	1.0
$\widetilde{\eta}$	regularization	10^{-5}
d_o	observed state dimension	512
n_w	warm-up steps (testing)	2000

TABLE B.6: Hyperparameters of **Parallel RC** for Kuramoto-Sivashinsky

Hyperparameter	Explanation	Values
d_h	hidden state size	{80, 100, 120}
N_g	number of groups	64
G	group size	8
I	interaction length	8
N	training data samples	10^5
B	batch-size	32
κ_1	BPTT forward timesteps	4
κ_2	BPTT truncated backprop. length	4
κ_3	BPTT skip gradient parameter	4
$\widetilde{\eta}$	initial learning rate	10^{-3}
p	zoneout probability	{0.998, 1.0}
d_o	observed state dimension	512
n_w	warm-up steps (testing)	2000

TABLE B.7: Hyperparameters of **Parallel GRU/LSTM** for Kuramoto-Sivashinsky

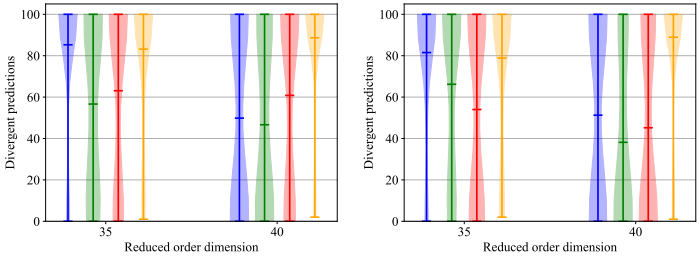
Hyperparameter	Explanation	Values
d_h	hidden state size	{100, 200, 400}
N_g	number of groups	64
G	group size	8
I	interaction length	8
N	training data samples	10^5
B	batch-size	32
κ_1	BPTT forward timesteps	4
κ_2	BPTT truncated backprop. length	4
κ_3	BPTT skip gradient parameter	4
$\widetilde{\eta}$	initial learning rate	10^{-2}
p	zoneout probability	1.0
d_o	observed state dimension	512
n_w	warm-up steps (testing)	2000

TABLE B.8: Hyperparameters of **Parallel Unitary Evolution** networks for Kuramoto-Sivashinsky

B.5 DIVERGENCE OF UNITARY AND RC RNNS IN LORENZ 96

In this section, we try to quantify the divergence effect due to the accumulation of the forecasting error in the iterative prediction. In Figure B.3 we present violin plots with fitted kernel density 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 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 in the violin plot. This divergence effect seems more prominent in RC and Unitary networks in the reduced-order scenario, as indicated by the skinny 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 is cumbersome when the system state is not fully observed compared to LSTM and GRU networks.

One example of this divergence effect in the reduced-order scenario in an initial condition from the test dataset is illustrated in Figure B.4. After approximately two Lyapunov times, the RC and the Unitary networks diverge in the reduced-order state predictions. A second example for the full state information scenario is given in Figure B.5. Unitary networks suffer from the 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. 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. RC models achieve superior forecasting accuracy in the full-state setting 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.



(a) $F = 8$ (b) $F = 10$

FIGURE B.3: Violin plots with kernel density estimates of the number of divergent predictions over the 100 initial conditions from the test data, over all hyperparameter sets for RC, GRU, LSTM, and Unitary networks, for reduced-order state $d_o = 35$ and full-order state $d_o = 40$ in two forcing regimes (a) $F = 8$ and (b) $F = 10$ in the Lorenz 96 model. Most hyperparameter sets of Unitary networks lead to models that diverge in iterative forecasting in both reduced-order and full-order scenarios for both $F \in \{8, 10\}$. Although the divergence effect is a non-issue in RC in the full state scenario $d_o = 40$, indicated by the wide part in the lower end of the density plot, the effect is more prominent in the reduced-order scenario compared to GRU and LSTM. Identifying hyperparameters for LSTM and GRU networks that show stable iterative forecasting behavior in the reduced-order space is significantly easier than RC and Unitary networks, as indicated by the wide/thin lines in the lower part of the density plots of the first/latter.
RC ; GRU ; LSTM ; Unit ;

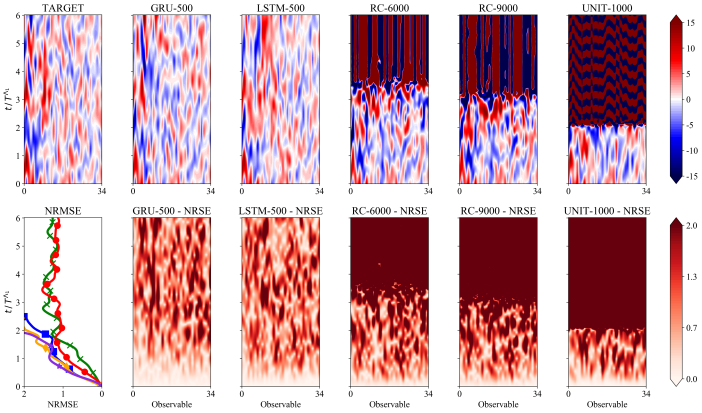


FIGURE B.4: Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz 96 model with $F = 8$ in the testing dataset with GRU, LSTM, RC, and a Unitary network along with the groud-truth (target) evolution and the associated NRSE contours for the reduced-order observable $d_o = 35$. The component average normalized RMSE (NRMSE) evolution is plotted to facilitate comparison.
GRU ✱; LSTM ●; RC-6000 ■; RC-9000 ◆; Unit ★;

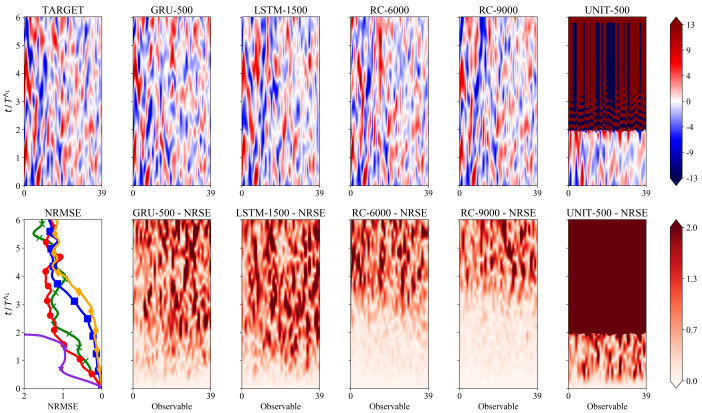
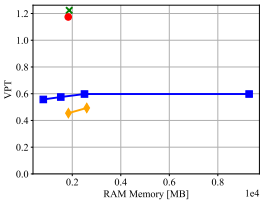


FIGURE B.5: Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz 96 model with $F = 8$ in the testing dataset with GRU, LSTM, RC, and a Unitary network along with the ground-truth (target) evolution and the associated NRSE contours for the full state $d_o = 40$. The evolution of the component average normalized RMSE (NRMSE) is plotted to facilitate comparison.
GRU ✱; LSTM ●; RC-6000 ■; RC-9000 ◆; Unit ★;

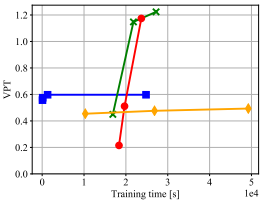
B.6 RESULTS ON LORENZ 96 FOR $F = 10$

In Figure B.6, 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 $F = 8$.

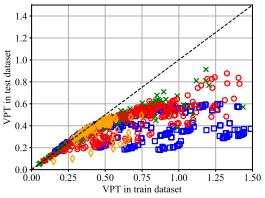
An example of a single forecast of the models starting from an initial condition in the test dataset is given in Figure B.7 for the reduced-order scenario, i. e. $d_o = 35$, and Figure B.8 for the full state scenario, i. e. $d_o = 40$. Forecasting the observable with $d_o = 35$ is more challenging compared to the full state scenario ($d_o = 40$) due to both (1) sensitivity to initial condition and (2) incomplete state information that requires the capturing of temporal dependencies. For this reason, the iterative prediction error increases slower in $d_o = 40$. Even in the challenging scenario of $d_o = 35$, LSTM and GRU networks demonstrate a stable behavior in iterative prediction and reproduce the long-term statistics of the attractor. In the reduced-order setting $d_o = 35$, accurate short-term predictions can be achieved with very large RC networks ($d_h = 9000$) at the cost of high memory requirements. However, even in this case, RC models may diverge from the attractor and do not reproduce the attractor climate.



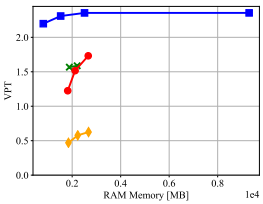
(a) VPT w. r. t. RAM memory for $d_o = 35$.



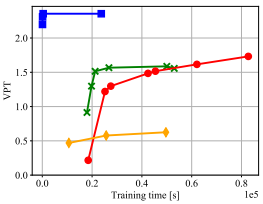
(b) VPT w. r. t. the training time for $d_o = 35$.



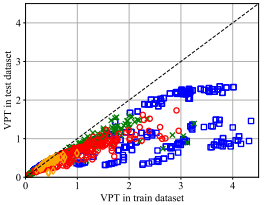
(c) VPT in test data w. r. t. VPT in the training data for $d_o = 35$.



(d) VPT w. r. t. RAM memory for $d_o = 40$.



(e) VPT w. r. t. the training time for $d_o = 40$.



(f) VPT in test data w. r. t. VPT in the training data for $d_o = 40$.

FIGURE B.6: Forecasting results on the dynamics of an observable consisting of the SVD modes of the Lorenz 96 model with $F = 10$ 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. training time for dimension $d_o \in \{35, 40\}$. (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 ;

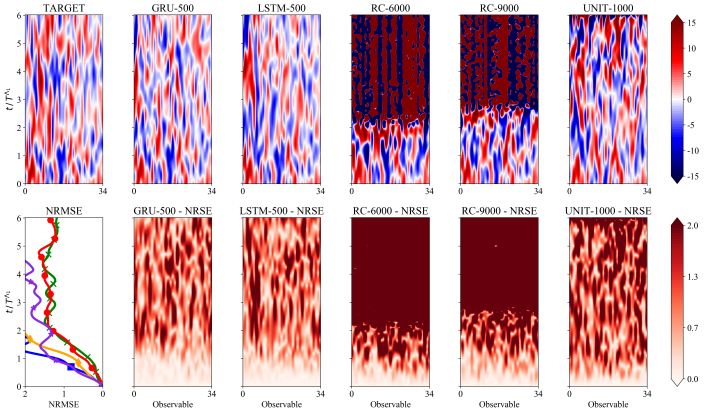


FIGURE B.7: Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz 96 model with $F = 10$ in the testing dataset with GRU, LSTM, RC, and a Unitary network along with the true (target) evolution and the associated NRSE contours for the reduced-order observable $d_o = 35$. The evolution of the component average normalized RMSE (NRMSE) is plotted to facilitate comparison.
GRU ✱; LSTM ●; RC-6000 ■; RC-9000 ◆; Unit ★;

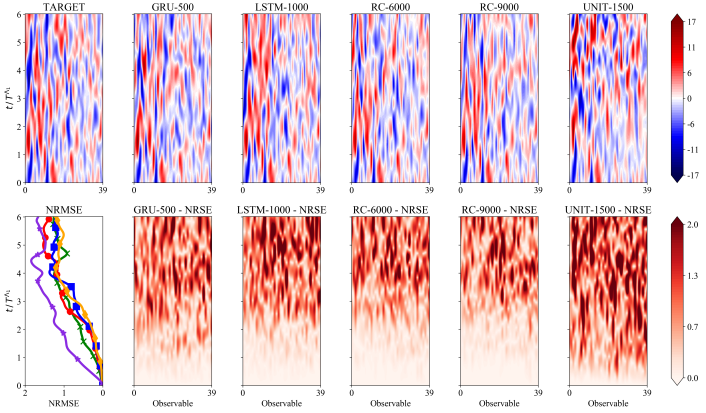


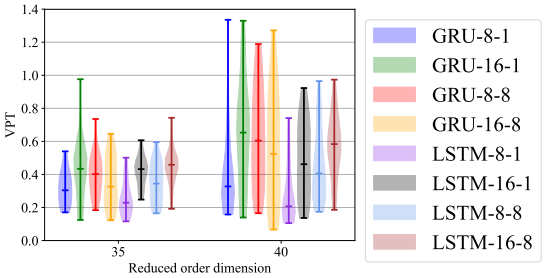
FIGURE B.8: Contour plots of a spatio-temporal forecast on the SVD modes of the Lorenz 96 model with $F = 10$ in the testing dataset with GRU, LSTM, RC, and a Unitary network along with the true (target) evolution and the associated NRSE contours for the full state $d_o = 40$. The evolution of the component average normalized RMSE (NRMSE) is plotted to facilitate comparison.
GRU ✱; LSTM ●; RC-6000 ■; RC-9000 ◆; Unit ★;

B.7 TEMPORAL DEPENDENCIES AND BACKPROPAGATION

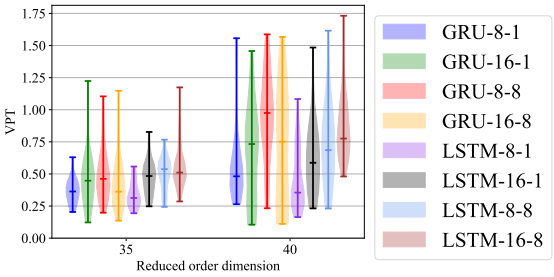
In order to train the GRU and LSTM models with BPTT, we need to tune the parameters κ_1 and κ_2. The first one denotes the truncated backpropagation length (also referred to as sequence length), and the second the number of future timesteps used to compute the loss and backpropagate the gradient during training at each batch. In the hyperparameter study, 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 B.4.

In Figure B.9 we present a violin plot that illustrates the forecasting efficiency of LSTM and GRU models trained with the listed κ_1 and κ_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 model 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 κ_1 and a large prediction length κ_2 are pivotal to achieving a high forecasting efficiency. This implies temporal correlations in the data that other models with smaller horizons cannot easily capture. In contrast, in the full-order scenario, models with smaller κ_1 perform reasonably well, demonstrating that the need to capture temporal correlations in the data to forecast the evolution is less prominent since the full information of the state of the system is available.



(a) $F = 8$



(b) $F = 10$

FIGURE B.9: Violin plots of the VPT in the testing data for stateful LSTM and GRU models trained with different truncated Backpropagation through time parameters κ_1 and κ_2 in the (reduced) SVD mode observable of the Lorenz 96 model. The legend of each plot reports the models and 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 backpropagation parameter κ_1 (also referred to as sequence length) is vital to capture the temporal dependencies in the data and achieve high forecasting efficiency. In contrast, a model with a small backpropagation horizon suffices in the full-state scenario ($d_0 = 40$).

C

SCHEDULED AUTOREGRESSIVE BACKPROPAGATION THROUGH TIME FOR LONG-TERM FORECASTING

C.1 SCHEDULED AUTOREGRESSIVE BACKPROPAGATION THROUGH TIME

C.1.1 *Equation 5.4*

$$
\begin{aligned}
\boldsymbol{z}_{t+1} = \mathcal{F}_{ho} \circ \boldsymbol{h}_{t+1} &\overset{(5.2)}{=} \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \boldsymbol{h}_t + \mathcal{F}_{ho} \circ \mathcal{H}_o \circ \boldsymbol{z}_t \\
&\overset{(5.2)}{=} \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \left(\mathcal{H}_h \circ \boldsymbol{h}_{t-1} + \mathcal{H}_o \circ \boldsymbol{z}_{t-1}\right) + \mathcal{F}_{ho} \circ \mathcal{H}_o \circ \boldsymbol{z}_t \\
&= \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \mathcal{H}_h \circ \boldsymbol{h}_{t-1} + \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \mathcal{H}_o \circ \boldsymbol{z}_{t-1} + \mathcal{F}_{ho} \circ \mathcal{H}_o \circ \boldsymbol{z}_t \\
&\overset{(5.2)}{=} \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \mathcal{H}_h \circ \left(\mathcal{H}_h \circ \boldsymbol{h}_{t-2} + \mathcal{H}_o \circ \boldsymbol{z}_{t-2}\right) + \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \mathcal{H}_o \circ \boldsymbol{z}_{t-1} + \mathcal{F}_{ho} \circ \mathcal{H}_o \circ \boldsymbol{z}_t \\
&= \mathcal{F}_{ho} \circ (\mathcal{H}_h)^3 \circ \boldsymbol{h}_{t-2} + \mathcal{F}_{ho} \circ (\mathcal{H}_h)^2 \circ \mathcal{H}_o \circ \boldsymbol{z}_{t-2} + \mathcal{F}_{ho} \circ \mathcal{H}_h \circ \mathcal{H}_o \circ \boldsymbol{z}_{t-1} + \mathcal{F}_{ho} \circ \mathcal{H}_o \circ \boldsymbol{z}_t \\
&\vdots \\
&\overset{(5.2)}{=} \mathcal{F}_{ho} \circ (\mathcal{H}_h)_2^{\kappa} \circ \boldsymbol{h}_{t-\kappa_2+1} + \mathcal{F}_{ho} \circ \sum_{k=0}^{\kappa_2-1} (\mathcal{H}_h)^k \circ \mathcal{H}_o \circ \boldsymbol{z}_{t-k}.
\end{aligned}
\tag{C.1}
$$

C.1.2 *Equation 5.6*

$$
\begin{aligned}
\boldsymbol{z}_{t+1} = \mathcal{F}_{ho} \circ \boldsymbol{h}_{t+1} &\overset{(5.2)}{=} \mathcal{F}_{ho} \circ \left(\mathcal{H}_h \circ \boldsymbol{h}_t + \mathcal{H}_o \circ \boldsymbol{z}_t\right) \\
&\overset{(5.3)}{=} \mathcal{F}_{ho} \circ \left(\mathcal{H}_h \circ \boldsymbol{h}_t + \mathcal{H}_o \circ \mathcal{F}_{ho} \circ \boldsymbol{h}_t\right) \\
&= \mathcal{F}_{ho} \circ \left(\mathcal{H}_h + \mathcal{H}_o \circ \mathcal{F}_{ho}\right) \circ \boldsymbol{h}_t \\
&\vdots \\
&\overset{(5.2),(5.3)}{=} \mathcal{F}_{ho} \circ \left(\mathcal{H}_h + \mathcal{H}_o \circ \mathcal{F}_{ho}\right)^{\kappa_2 - 1} \circ \boldsymbol{h}_{t-\kappa_2+2} \\
&= \mathcal{F}_{ho} \circ \left(\mathcal{H}_h + \mathcal{H}_o \circ \mathcal{F}_{ho}\right)^{\kappa_2 - 1} \circ \left(\mathcal{H}_h \circ \boldsymbol{h}_{t-\kappa_2+1} + \mathcal{H}_o \circ \boldsymbol{z}_{t-\kappa_2+1}\right)
\end{aligned}
\tag{C.2}
$$

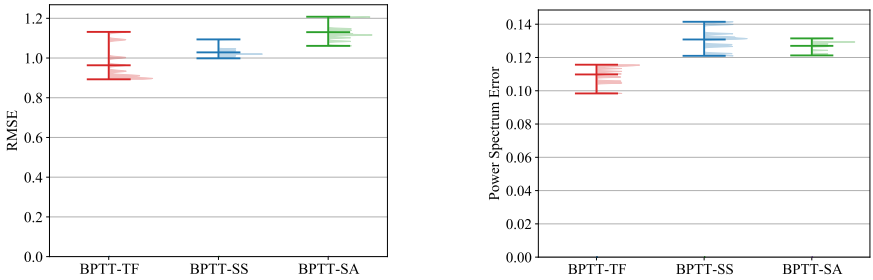


FIGURE C.1: Results on the Darwin sea level pressure dataset.

C.2 DARWIN SEA LEVEL TEMPERATURES

We evaluate the long-term predictability of the methods in the open-source dataset of the monthly values of the Darwin Sea Level Pressure series *Monthly values of the Darwin Sea Level Pressure series, 1882-1998* n.d., frequently used for benchmarking of time series prediction methods. The dataset consists of 1400 samples. The first 600 samples are used for training and the next 400 for validation. The long-term forecasting accuracy of the methods is evaluated on 32 initial conditions randomly sampled from the test data. The RNNs forecast up to a prediction horizon of 100 timesteps after an initial warm-up period of 50 timesteps. The results are illustrated in Figure C.1. We observe that both variants BPTT-SA and BPTT-SS do not improve the RMSE or the power spectrum error.

The hyperparameters for the networks employed in the Darwin dataset are given in Table C.1.

C.3 MACKEY-GLASS EQUATION

The hyperparameters for the networks employed in the Mackey-Glass equation are given in Table C.2.

Hyperparameter	Values
Optimizer	Adam
Batch size	32
Initial learning rate	0.0001
Max Epochs	2000
Random Seed	$\{1, \ldots, 10\}$
BPTT sequence length κ_2	100
Prediction horizon	100
Number of testing initial conditions	32
Number of LSTM layers	1
Size of LSTM layers	100
Activation of LSTM Cell	tanh
Scaling	$[0, 1]$

TABLE C.1: Hyperparameter tuning in Darwin dataset

Hyperparameter	Values
Optimizer	Adam
SNR	$\{10, 60\}$
Batch size	32
Initial learning rate	0.0005
Max Epochs	2000
Random Seed	$\{1, \ldots, 10\}$
BPTT sequence length κ_2	40
Prediction horizon	896
Number of testing initial conditions	100
Number of LSTM layers	1
Size of LSTM layers	100
Activation of LSTM Cell	tanh
Scaling	$[0, 1]$

TABLE C.2: Hyperparameter tuning in Mackey-Glass equation

C.4 VISCOUS FLOW PAST A CYLINDER IN A CHANNEL

C.4.1 *Data Generation*

The flow is simulated with a Finite Element (FEM) solver Alnæs et al., 2015 with a timestep of 0.001. The velocity $u \in \mathbb{R}^2$ and pressure $p \in \mathbb{R}$ values were extracted in a uniform grid of 160×32, and data are subsampled to $\Delta t = 0.01$. For plotting purposes, we plot the Frobenius norm of the velocity $u = \sqrt{u_x^2 + u_y^2}$. For more information on the geometry and simulation, details refer to Petter Langtangen et al., 2017. The total simulation time is $T = 24$. The first 200 timesteps are used for training, the next 200 for validation, and the next 2000 for testing. The long-term iterative prediction performance of the methods is evaluated on prediction of 1000 timesteps starting from 10 initial conditions randomly sampled from the test data.

C.4.2 *Hyperparameters*

The hyperparameters for the networks employed in the viscous flow past a cylinder in a channel dataset are given in Table C.5 for ConvRNNs and in Table C.4 for CNN-RNNs. The autoencoder of CNN-RNNs is composed of consecutive Convolutional layers, Average pooling, CELU activation, and Batch-Norm layers. The exact architecture is given in Table C.3. The autoencoder is reducing the dimensionality on a $z \in \mathbb{R}^5$ latent space. An LSTM with 24 units is predicting on this latent space.

Layer	Encoder	Decoder
1	ZeroPad2d(padding=(5, 5, 5, 5), value=0.0)	Upsample(scale_factor=2.0, mode=bilinear)
2	Conv2d(3, 5, kernel_size=(11, 11), stride=(1, 1))	ConvTranspose2d(1, 2, kernel_size=(3, 3), stride=(1, 1), padding=[1, 1])
3	AvgPool2d(kernel_size=2, stride=2, padding=0)	CELU(alpha=1.0)
4	CELU(alpha=1.0)	BatchNorm2d(2, eps=1e-05, momentum=0.1, affine=False)
5	BatchNorm2d(5, eps=1e-05, momentum=0.1, affine=False)	Upsample(scale_factor=2.0, mode=bilinear)
6	ZeroPad2d(padding=(4, 4, 4, 4), value=0.0)	ConvTranspose2d(2, 20, kernel_size=(3, 3), stride=(1, 1), padding=[1, 1])
7	Conv2d(5, 10, kernel_size=(9, 9), stride=(1, 1))	CELU(alpha=1.0)
8	AvgPool2d(kernel_size=2, stride=2, padding=0)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=False)
9	CELU(alpha=1.0)	Upsample(scale_factor=2.0, mode=bilinear)
10	BatchNorm2d(5, eps=1e-05, momentum=0.1, affine=False)	ConvTranspose2d(20, 10, kernel_size=(7, 7), stride=(1, 1), padding=[3, 3])
11	ZeroPad2d(padding=(3, 3, 3, 3), value=0.0)	CELU(alpha=1.0)
12	Conv2d(10, 20, kernel_size=(7, 7), stride=(1, 1))	BatchNorm2d(10, eps=1e-05, momentum=0.1, affine=False)
13	AvgPool2d(kernel_size=2, stride=2, padding=0)	Upsample(scale_factor=2.0, mode=bilinear)
14	Conv2d(5, 10, kernel_size=(9, 9), stride=(1, 1))	ConvTranspose2d(10, 5, kernel_size=(9, 9), stride=(1, 1), padding=[4, 4])
15	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=False)	CELU(alpha=1.0)
16	ZeroPad2d(padding=(1, 1, 1, 1), value=0.0)	BatchNorm2d(5, eps=1e-05, momentum=0.1, affine=False)
17	Conv2d(20, 2, kernel_size=(3, 3), stride=(1, 1))	Upsample(scale_factor=2.0, mode=bilinear)
18	AvgPool2d(kernel_size=2, stride=2, padding=0)	ConvTranspose2d(5, 3, kernel_size=(11, 11), stride=(1, 1), padding=[5, 5])
19	CELU(alpha=1.0)	0.5 + 0.5 Tanh()
20	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=False)	
21	ZeroPad2d(padding=(1, 1, 1, 1), value=0.0)	
22	Conv2d(20, 2, kernel_size=(3, 3), stride=(1, 1))	
23	AvgPool2d(kernel_size=2, stride=2, padding=0)	
24	CELU(alpha=1.0)	
Latent	$z \in \mathbb{R}^5$	
Scaling	$[0, 1]$	

TABLE C.3: Architecture of CNN of CNN-RNNs in viscous flow past a cylinder in a channel dataset

Hyperparameter	Values
Optimizer	Adam
Batch size	32
Initial learning rate	0.001
Max Epochs	1000
Random Seed	$\{1, 2, 3, 4\}$
BPTT sequence length κ_2	20
Prediction horizon	1000
Number of testing initial conditions	10
RNN Cell	LSTM
Number of RNN layers	1
Size of RNN layers	24
Scaling	$[0, 1]$

TABLE C.4: Hyperparameters of CNN-RNNs in Viscous Flow Past a Cylinder in a Channel

Hyperparameter	Values
Optimizer	Adam
Batch size	16
Initial learning rate	0.0001
Max Epochs	5000
Random Seed	$\{1, 2, 3\}$
BPTT sequence length κ_2	50
Prediction horizon	1000
Number of testing initial conditions	10
RNN Cell	LSTM
Number of RNN layers	1
Size of RNN layers	8
Kernel size	5
Scaling	$[0, 1]$

TABLE C.5: Hyperparameters of ConvRNNs in in Viscous Flow Past a Cylinder in a Channel

D

LEARNING EFFECTIVE DYNAMICS

D.1 FITZHUGH-NAGUMO MODEL

Input and output are scaled to $[0, 1]$ and an output activation function of the form $1 + 0.5 \tanh(\cdot)$ is used to ensure that the data at the output lie at this range. The hyper-parameter tuning of the autoencoder of LED and training times are reported in Table D.1. PCA and Diffusion maps have very short fitting (training) times of approximately one minute. The layers of the CNN autoencoder employed in the FHN and its training times are given in Table D.3.

The hyperparameters for the LSTM and its training times are given in Table D.2. For the MLP, a three-layered network with CELU activations is employed. Training time for the MLP is 100 minutes. The hyperparameters and training times for the RC are given in Table D.4. The hyperparameters and training times for SINDy are given in Table D.5.

In all cases, the parameters of the best performing model on the validation data is denoted with red color.

D.2 THE KURAMOTO-SIVASHINSKY EQUATION

The hyper-parameter tunings and training times for the AE and CNN are given in Table D.6 and Table D.7 respectively . The architecture of the CNN autoencoder employed in KS is given in Table D.8 along with the training times and depicted in Figure 6.9. PCA fitting time is approximately one minute. The hyperparameters and training times of the LSTM-RNN of LED are given in Table D.9. The hyperparameters and training times of the RC are given in Table D.10. The hyperparameters and training times of SINDy are given in Table D.11.

Hyper-parameter tuning	Values
Number of AE layers	$\{3\}$
Size of AE layers	$\{100\}$
Activation of AE layers	$\mathrm{celu}(\cdot)$
Latent dimension	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64\}$
Input/Output data scaling	$[0, 1]$
Output activation	$1 + 0.5 \tanh(\cdot)$
Weight decay rate	$\{0.0, 0.0001\}$
Batch size	32
Initial learning rate	0.001

Training times [minutes]

Min	Mean	Max
130	180	235

TABLE D.1: Autoencoder hyperparameters and training times for FHN

D.3 VISCOUS FLOW PAST A CYLINDER

LED employs a Convolutional neural network (CNN) to identify a low-dimensional latent space $\mathbf{z} \in \mathbb{R}^4$ in the $\mathrm{Re} = 100$ scenario, and $\mathbf{z} \in \mathbb{R}^{10}$ in the $\mathrm{Re} = 1000$ scenario. The CNN architecture is depicted in Figure 6.15, and the layers are given in Table D.12. We experimented with various activation functions, addition of batch-normalization layers, addition of transpose convolutional layers in the decoding part, different kernel sizes, and optimizers. The data are scaled to $[0, 1]$. The output activation function of the CNN autoencoder is set to $0.5 + 0.5 \tanh(\cdot)$, whose image range matches the data range.

The hyper-parameter tuning and training times for the LSTM-RNN of LED are given in Table D.13. The hyperparameters and training times for the RC are given in Table D.14. The hyperparameters and training times for SINDy are given in Table D.15.

The hyperparameters and training times for the RC are given in Table D.14. The hyperparameters and training times for SINDy are given in Table D.15.

Hyper-parameter	Values
end2end training	True / False
Number of AE layers	$\{3\}$
Size of AE layers	$\{100\}$
Activation of AE layers	$\mathrm{celu}(\cdot)$
Latent dimension	2
Input/Output data scaling	$[0, 1]$
Output activation	$1 + 0.5 \tanh(\cdot)$
Weight decay rate	0.0
Batch size	32
Initial learning rate	0.001
BPTT Sequence length	$\{20, 40, 60\}$
Output forecasting loss	True/False
RNN cell type	lstm
Number of RNN layers	1
Size of RNN layers	$\{16, 32, 64\}$
Activation of RNN Cell	$\tanh(\cdot)$
Output activation of RNN Cell	$1 + 0.5 \tanh(\cdot)$

Training times [minutes]	Min	Mean	Max
end2end training	2.2	2.5	2.8
only the RNN (sequential)	0.9	1.2	1.6

TABLE D.2: LED-RNN hyperparameters and training times for FHN

Layer	ENCODER
(0)	ConstantPad1d(padding=(13, 14), value=0.0)
(1)	ConstantPad1d(padding=(2, 2), value=0.0)
(2)	Conv1d(2, 8, kernel_size=(5,), stride=(1,))
(3)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(4)	CELU(alpha=1.0)
(5)	ConstantPad1d(padding=(2, 2), value=0.0)
(6)	Conv1d(8, 16, kernel_size=(5,), stride=(1,))
(7)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(8)	CELU(alpha=1.0)
(9)	ConstantPad1d(padding=(2, 2), value=0.0)
(10)	Conv1d(16, 32, kernel_size=(5,), stride=(1,))
(11)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(12)	CELU(alpha=1.0)
(13)	ConstantPad1d(padding=(2, 2), value=0.0)
(14)	Conv1d(32, 4, kernel_size=(5,), stride=(1,))
(15)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(16)	Flatten(start_dim=-2, end_dim=-1)
(17)	Linear(in_features=32, out_features=$d_{\mathbf{z}}$, bias=True)
(18)	CELU(alpha=1.0)
	$\mathbf{z} \in \mathbb{R}^{d_{\mathbf{z}}}$

Layer	DECODER
(1)	Linear(in_features=$d_{\mathbf{z}}$, out_features=32, bias=True)
(2)	CELU(alpha=1.0)
(3)	Upsample(scale_factor=2.0, mode=linear)
(4)	ConvTranspose1d(4, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(5)	CELU(alpha=1.0)
(6)	Upsample(scale_factor=2.0, mode=linear)
(7)	ConvTranspose1d(32, 16, kernel_size=(5,), stride=(1,), padding=(2,))
(8)	CELU(alpha=1.0)
(9)	Upsample(scale_factor=2.0, mode=linear)
(10)	ConvTranspose1d(16, 8, kernel_size=(5,), stride=(1,), padding=(2,))
(11)	CELU(alpha=1.0)
(12)	Upsample(scale_factor=2.0, mode=linear)
(13)	ConvTranspose1d(8, 2, kernel_size=(5,), stride=(1,), padding=(2,))
(14)	1 + 0.5 Tanh()
(15)	Unpad()

Latent dimension $d_{\mathbf{z}}$	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64\}$

Training times [minutes]

Min	Mean	Max
215	370	530

TABLE D.3: CNN Autoencoder and training times for FHN

Hyper-parameter tuning	Values
Solver	Pseudoinverse
Size	1000
Degree	10
Radius	0.99
Input scaling σ	$\{0.5, 1, 2\}$
Dynamics length	100
Regularization $\widetilde{\eta}$	$\{0.0, 0.001, 0.0001, 0.00001\}$
Noise level per mill	$\{10, 20, 30, 40, 100\}$

Training times [minutes]

Min	Mean	Max
0.15	0.18	0.19

TABLE D.4: Reservoir Computer hyperparameters and training times (in CNN-RC) for FHN

Hyper-parameter tuning	Values
Degree	$\{1, 2, 3\}$
Threshold	$\{0.001, 0.0001, 0.00001\}$
Library	Polynomials

Training times [minutes]

Min	Mean	Max
0.14	0.23	0.32

TABLE D.5: SINDy hyperparameters and training times (in CNN-SINDy) for FHN

Hyper-parameter tuning	Values
Number of AE layers	$\{3\}$
Size of AE layers	$\{100\}$
Activation of AE layers	$\mathrm{celu}(\cdot)$
Latent dimension	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64\}$
Input/Output data scaling	$[0, 1]$
Output activation	$1 + 0.5 \tanh(\cdot)$
Weight decay rate	$\{0.0, 0.0001\}$
Batch size	32
Initial learning rate	0.001

Training times [minutes]

Min	Mean	Max
160	192	311

TABLE D.6: Autoencoder hyperparameters for KS

Hyper-parameter	Values
Convolutional	True
Kernels	Encoder: $5-5-5-5$, Decoder: $5-5-5-5$
Channels	$1-16-32-64-8-\mathbf{d}_z-8-64-32-16-1$
Batch normalization	True / False
Transpose convolution	True / False
Pooling	Average
Activation	$\mathrm{celu}(\cdot)$
Latent dimension	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64\}$
Input/Output data scaling	$[0, 1]$
Output activation	$1 + 0.5 \tanh(\cdot)$
Weight decay rate	0.0
Batch size	32
Initial learning rate	0.001

Training times [minutes]

Min	Mean	Max
236	311	476

TABLE D.7: CNN hyperparameters for KS

Layer	ENCODER
(1)	ConstantPad1d(padding=(2, 2), value=0.0)
(2)	Conv1d(1, 16, kernel_size=(5,), stride=(1,))
(3)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(4)	CELU(alpha=1.0)
(5)	ConstantPad1d(padding=(2, 2), value=0.0)
(6)	Conv1d(16, 32, kernel_size=(5,), stride=(1,))
(7)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(8)	CELU(alpha=1.0)
(9)	ConstantPad1d(padding=(2, 2), value=0.0)
(10)	Conv1d(32, 64, kernel_size=(5,), stride=(1,))
(11)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(12)	CELU(alpha=1.0)
(13)	ConstantPad1d(padding=(2, 2), value=0.0)
(14)	Conv1d(64, 8, kernel_size=(5,), stride=(1,))
(15)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(16)	CELU(alpha=1.0)
(17)	Flatten(start_dim =-2, end_dim = -1)
(18)	Linear(in_features =32, out_features =8, bias=True)
(19)	CELU(alpha=1.0)
	$\mathbf{z} \in \mathbb{R}^8$
Layer	**DECODER**
(1)	Linear(in_features=8, out_features=32, bias=True)
(2)	CELU(alpha=1.0)
(3)	Upsample(scale_factor=2.0, mode=linear)
(4)	Conv1d(8, 64, kernel_size=(5,), stride=(1,), padding=(2,))
(5)	CELU(alpha=1.0)
(6)	Upsample(scale_factor=2.0, mode=linear)
(7)	Conv1d(64, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(8)	CELU(alpha=1.0)
(9)	Upsample(scale_factor=2.0, mode=linear)
(10)	Conv1d(32, 16, kernel_size=(5,), stride=(1,), padding=(2,))
(11)	CELU(alpha=1.0)
(12)	Upsample(scale_factor=2.0, mode=linear)
(13)	Conv1d(16, 1, kernel_size=(5,), stride=(1,), padding=(2,))
(14)	1 + 0.5 Tanh()

TABLE D.8: CNN Autoencoder for KS

Hyper-parameter	Values
end2end training	False / True
Convolutional AE (CNN)	True
Kernels	Encoder: $5-5-5-5$, Decoder: $5-5-5-5$
Channels	$1-16-32-64-8-\mathbf{d}_z-8-64-32-16-1$
Batch normalization	False
Transpose convolution	False
Pooling	Average
Activation	$\text{celu}(\cdot)$
Latent dimension	8
Input/Output data scaling	$[0,1]$
Output activation	$1+0.5\tanh(\cdot)$
Weight decay rate	0.0
Batch size	32
Initial learning rate	0.001
BPTT Sequence length	$\{25, 50, 100\}$
Output forecasting loss	True/False
RNN cell type	lstm
Number of RNN layers	1
Size of RNN layers	$\{64, 128, 256, 512\}$
Activation of RNN Cell	$\tanh(\cdot)$
Output activation of RNN Cell	$1+0.5\tanh(\cdot)$

Training times [minutes]	Min	Mean	Max
end2end training	476	978	1140
only the RNN (sequential)	960	1100	1140

TABLE D.9: LED (LSTM-RNN) hyperparameters and training times for KS

Hyper-parameter tuning	Values
Solver	Pseudoinverse
Size	1000
Degree	10
Radius	0.99
Input scaling σ	$\{0.5, 1, 2\}$
Dynamics length	100
Regularization $\widetilde{\eta}$	$\{0.0, 0.001, 0.0001, 0.00001\}$
Noise level per mill	$\{10, 20, 30, 40, 100\}$

Training times [minutes]

Min	Mean	Max
0.25	0.35	0.38

TABLE D.10: Reservoir Computer hyperparameters and training times (in CNN-RC) for KS

Hyper-parameter tuning	Values
Library	Polynomials
Degree	$\{1, 2, 3\}$
Threshold	$\{0.001, 0.0001, 0.00001\}$

Training times [minutes]

Min	Mean	Max
0.13	0.62	1.59

TABLE D.11: SINDy hyperparameters and training times (in CNN-SINDy) for KS

Layer	ENCODER
(0)	interpolationLayer()
(1)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(2)	Conv2d(4, 20, kernel_size=(13, 13), stride=(1, 1))
(3)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(4)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(5)	CELU(alpha=1.0)
(6)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(7)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(8)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(9)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(10)	CELU(alpha=1.0)
(11)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(12)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(13)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(14)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(15)	CELU(alpha=1.0)
(16)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(17)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(18)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(19)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(20)	CELU(alpha=1.0)
(21)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(22)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(23)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(24)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(25)	CELU(alpha=1.0)
(26)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(27)	Conv2d(20, 2, kernel_size=(13, 13), stride=(1, 1))
(28)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(29)	CELU(alpha=1.0)
(30)	Flatten(start_dim=-3, end_dim=-1)
(31)	Linear(in_features=64, out_features=d_z, bias=True)
(32)	CELU(alpha=1.0)
	$\mathbf{z} \in \mathbb{R}^{d_z}$
Layer	**DECODER**
(0)	Linear(in_features=d_z, out_features=64, bias=True)
(1)	CELU(alpha=1.0)
(2)	ViewModule()
(3)	ConvTranspose2d(2, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(4)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(5)	CELU(alpha=1.0)
(6)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(7)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(8)	CELU(alpha=1.0)
(9)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(10)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(11)	CELU(alpha=1.0)
(12)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(13)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(14)	CELU(alpha=1.0)
(15)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(16)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(17)	CELU(alpha=1.0)
(18)	ConvTranspose2d(20, 4, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(19)	interpolationLayer()
(20)	1 + 0.5 Tanh()
Latent dimension d_z	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16\}$

Training times [minutes]

Min	Mean	Max
1080	1081	1083

TABLE D.12: CNN Autoencoder of LED for the flow past a cylinder at Re $\in \{100, 1000\}$

Hyperparameter	Values
Optimizer	Adabelief
Batch size	32
Initial learning rate	0.001
Max Epochs	1000
BPTT sequence length κ_2	$\{10, 25\}$
Warm-up steps	10
Prediction horizon	1000
RNN Cell	LSTM
Number of RNN layers	1
Size of RNN layers	$\{32, 64\}$
Scaling	$[0, 1]$

Training times [minutes]

Min	Mean	Max
722	723	724

TABLE D.13: LED (LSTM-RNN) hyperparameters and training times for the flow past a cylinder example

Hyper-parameter tuning	Values for Re = 100	Values for Re = 1000
Solver	Pseudoinverse	Pseudoinverse
Size	200	200
Degree	10	10
Radius	0.99	0.99
Input scaling σ	$\{0.5, 1, 2\}$	$\{0.5, 1, 2\}$
Dynamics length	100	100
Regularization $\widetilde{\eta}$	$\{0.0, 0.001, 0.0001, 0.00001\}$	$\{0.0, 0.001, 0.0001, 0.00001\}$
Noise level per mill	$\{10\}$	$\{10\}$

Training times [minutes]

Min	Mean	Max
1.2	1.4	1.92

TABLE D.14: Reservoir Computer hyperparameters and training times (in CNN-RC) for flow past a cylinder example

Hyper-parameter tuning	Values for Re = 100	Values for Re = 1000
Library	Polynomials	Polynomials
Degree	{1, 2, 3}	{1, 2, 3}
Threshold	{0.001, 0.0001, 0.00001}	{0.001, 0.0001, 0.00001}

Training times [minutes]

Min	Mean	Max
1.14	1.55	2.05

TABLE D.15: SINDy hyperparameters and training times (in CNN-SINDy) for flow past a cylinder example

E

LED FOR MOLECULAR SYSTEMS

E.1 MÜLLER-BROWN POTENTIAL

The MBP has the form

$$\begin{aligned} V(\boldsymbol{x}) = \sum_{k=1}^{4} A_k \exp\big(&\alpha_k(x_1 - \hat{X}_{1,k}) + \\ &b_k(x_1 - \hat{X}_{1,k})(x_2 - \hat{X}_{2,k}) + \\ &c_k(x_2 - \hat{X}_{2,k})\big), \end{aligned} \tag{E.1}$$

where $\boldsymbol{x} = [x_1, x_2]^T$ is the position. The parametrization

$$\begin{aligned} \alpha &= [-1, -1, -6.5, 0.7]^T, \\ b &= [0, 0, 11, 0.6]^T, \\ c &= [-10, -10, -6.5, 0.7]^T, \\ A &= [-200, -100, -170, 15]^T, \\ \hat{X} &= \begin{bmatrix} 1 & 0 & -0.5 & -1 \\ 0 & 0.5 & 1.5 & 1 \end{bmatrix}, \end{aligned} \tag{E.2}$$

is followed according to Ref. Müller et al., 1979.

E.1.1 *Definition of Metastable States*

The metastable states of the MB potential are defined as ellipses in the $\boldsymbol{x} \in \mathbb{R}^2$ space. The centers and axes given in Table E.1.

E.1.2 *LED Hyperparameters*

In order to prepare the dataset for training, validation, and testing of the LED in the MBP, 96 initial conditions are sampled from $\boldsymbol{x} \in [-1.5, 1.2] \times [-0.2, 2]$. The dynamics are solved with the Velocity Verlet algorithm, with

State	Center (x_1, x_2)	Axes $\alpha, \beta)$	θ
0	$(-0.57, 1.45)$	$(0.15, 0.3)$	$\pi/4$
1	$(0.45, 0.05)$	$(0.35, 0.15)$	0

TABLE E.1: Metastable states in the MBP modeled as ellipses $x_1^2/\alpha^2 + x_2^2/\beta^2 \leq 1$. The ellipses are rotated by θ.

timestep $\delta t = 10^{-2}$ up to $T = 5000$, after an initial transient period of $\tilde{T} = 10^3$ discarded from the data. The data are sub-sampled to $\Delta t = 0.5$, keeping every 50^{th} data points. In this way, 96 trajectories of $N = 10^4$ samples, each corresponding to $T = 5000$ time units are created. LED is trained on 32 of these trajectories. 32 trajectories are used for validation, while all 96 trajectories are used for testing.

The number and size of hidden layers are the same for the encoder $\mathcal{E}$, the decoder $\mathcal{D}$, and the latent MDN $\mathcal{Z}$. In the first phase, the MDN-AE is trained, tuning its hyperparameters based on a grid search reported in Table E.2. The autoencoder with the smallest error on the state statistics on the validation dataset is picked. Next, the MDN-LSTM is trained, tuning its hyperparameters based on a grid search reported in Table E.3. The LED model with the smallest error on the state statistics on the validation dataset is picked. Both networks are trained with validation-based early stopping. The LED is tested on the total 96 initial conditions. For more information of the training technicalities the interested reader is referred to Ref. P. R. Vlachas, Arampatzis, et al., 2022.

E.1.3 *Timescales in the LED Latent Space*

The latent space learned by LED can be utilized to identify low-energy metastable states without the need for prior knowledge. The definition of the metastable states in the rotationally and translationally invariant space constitutes such prior knowledge. Minima in the free energy projection on the LED latent space constitute probable metastable states.

The trajectories sampled with LED are clustered based on these latent metastable clusters depicted in Figure 4 (main text). An MSM is fitted on the clustered trajectories. The time-lag of the MSM is set to 100 time units to ensure Markovianity. The timescales computed by MSM are $\overline{T}_{0\to1} = 49$ and $\overline{T}_{1\to0} = 321$. LED is overestimating $\overline{T}_{1\to0}$ and underestimating $\overline{T}_{0\to1}$.

Hyperparameter	Values
Batch size	32
Initial learning rate	0.001
Weight decay rate	$\{0, 10^{-5}\}$
Number of AE layers	$\{2, 3\}$
Size of AE layers	$\{10, 20, 40\}$
Activation of AE layers	selu, tanh
Latent dimension	$\{1\}$
Input/Output data scaling	$[0, 1]$
MDN-AE kernels	$\{2, 3\}$
MDN-AE hidden units	50
MDN-AE multivariate	1
MDN-AE covariance scaling factor	$\{0.4, 0.6, 0.8\}$

TABLE E.2: Hyperparameter tuning of AE for MBP

Hyperparameter	Values
Batch size	32
Initial learning rate	10^{-3}
BPTT sequence length	$\{200, 400\}$
Number of LSTM layers	1
Size of LSTM layers	$\{10, 20, 40\}$
Activation of LSTM Cell	tanh
MDN-LSTM kernels	$\{4, 5, 6\}$
MDN-LSTM hidden units	$\{10, 20\}$
MDN-LSTM multivariate	0
MDN-LSTM covariance scaling factor	$\{0.1, 0.2, 0.3, 0.4\}$

TABLE E.3: Hyperparameter tuning of LSTM for MBP

Hyperparameter	Values
Number of AE layers	3
Size of AE layers	40
Activation of AE layers	tanh
Latent dimension	1
MDN-AE kernels	3
MDN-AE hidden units	50
MDN-AE multivariate	1
MDN-AE covariance scaling factor	0.6
Weight decay rate	0.0
BPTT sequence length	400
Number of LSTM layers	1
Size of LSTM layers	20
Activation of LSTM Cell	tanh
MDN-LSTM kernels	4
MDN-LSTM hidden units	20
MDN-LSTM multivariate	0
MDN-LSTM covariance scaling factor	0.4

TABLE E.4: Hyperparameters of LED model with lowest validation error on MBP

The order of the timescales, however, is captured. In contrast, an MSM with a time-lag of $\Delta t = 0.5$, which is the timestep of the LED, fails to capture the order of the timescales due to the violated Markovianity assumption ($\overline{T}_{0\to1} = 3$ and $\overline{T}_{0\to1} = 21$).

E.2 TRP CAGE

E.2.1 *LED Hyperparameters*

In the LED architecture, the number and size of hidden layers are the same for the encoder $\mathcal{E}$, the decoder $\mathcal{D}$, and the latent MDN $\mathcal{Z}$. The MDN-AE is trained, tuning its hyperparameters based on the grid search reported in Table E.5. The latent space of the MDN-AE is $z \in \mathbb{R}^2$, i.e., $d_z = 2$. The MDN-AE model with the lowest error on the state statistics in the validation dataset is picked. Then, the MDN-AE is coupled with the MDN-LSTM as LED. The MDN-LSTM is trained to minimize the latent data likelihood. The hyperparameters of the MDN-LSTM are tuned according to the grid search reported in Table E.6. The LED model with the lowest error on the state statistics in the validation dataset is selected. Its hyperparameters are reported in Table E.7. The LED is tested in 248 initial conditions randomly sampled from the testing data. Starting from these initial conditions, we utilize the iterative propagation in the latent space of the LED to forecast $T = 400$ps.

Hyperparameter	Values
Batch size	32
Initial learning rate	10^{-3}
Weight decay rate	$\{0, 10^{-4}, 10^{-5}, 10^{-6}\}$
Number of AE layers	$\{4, 6\}$
Size of AE layers	$\{100, 200, 500\}$
Activation of AE layers	selu, tanh
Latent dimension	2
Input/Output data scaling	$[0, 1]$
MDN-AE kernels	$\{3, 4, 5\}$
MDN-AE hidden units	$\{20, 50\}$
MDN-AE covariance scaling factor	0.8

TABLE E.5: Hyperparameter tuning of AE for Trp Cage

Hyperparameter	Values
Batch size	32
Initial learning rate	10^{-3}
BPTT sequence length	$\{200, 400\}$
Number of LSTM layers	1
Size of LSTM layers	$\{10, 20, 40\}$
Activation of LSTM Cell	tanh
MDN-LSTM kernels	$\{4, 8, 12, 24\}$
MDN-LSTM hidden units	$\{10, 20, 40, 80\}$
MDN-LSTM multivariate	$\{0, 1\}$
MDN-LSTM covariance scaling factor	$\{0.1, 0.2, 0.3, 0.4\}$

TABLE E.6: Hyperparameter tuning of LSTM for Trp Cage

E.2.2 *Marginal State Distributions*

The marginal distributions of the trajectories generated by LED match the ground-truth ones (MD data) closely, as depicted in Figure E.1. In Figure E.2, a sample from MD data of the TRP cage is compared with a close sample (in terms of the latent space) of LED. The RMSD is 2.784$\mathring{A}$.

Hyperparameter	Values
Number of AE layers	6
Size of AE layers	500
Activation of AE layers	tanh
Latent dimension	2
MDN-AE kernels	5
MDN-AE hidden units	50
MDN-AE multivariate	0
MDN-AE covariance scaling factor	0.8
Weight decay rate	0
BPTT sequence length	400
Number of LSTM layers	1
Size of LSTM layers	40
Activation of LSTM Cell	tanh
MDN-LSTM kernels	4
MDN-LSTM hidden units	20
MDN-LSTM multivariate	0
MDN-LSTM covariance scaling factor	0.2

TABLE E.7: Hyperparameters of LED model with lowest validation error on Trp Cage

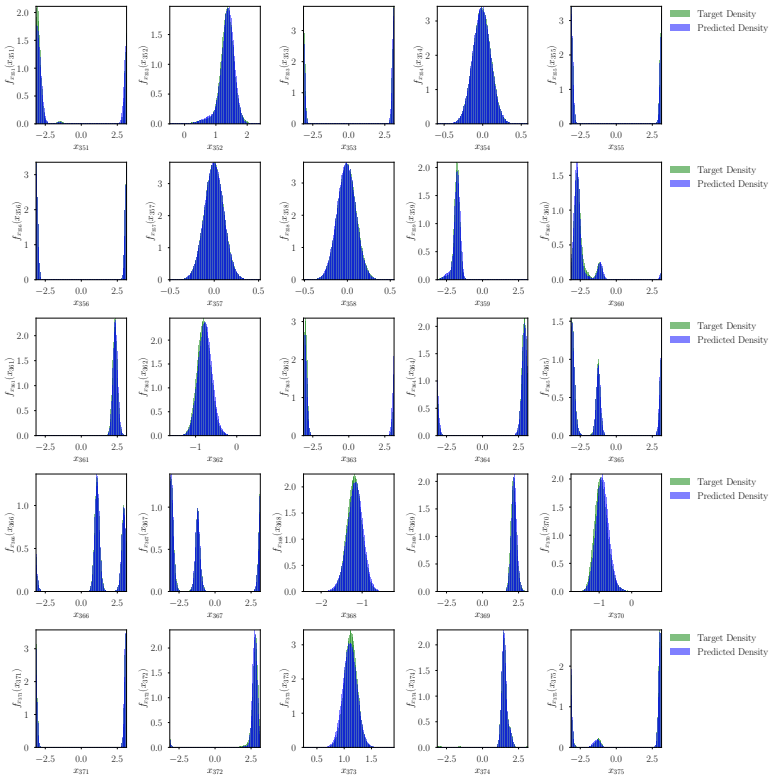


FIGURE E.1: Plot of the marginal state distributions $s_{351} - s_{375}$ in the Trp Cage miniprotein. Comparison of the state distributions estimated from the MD data (test dataset) and from trajectories sampled from LED.

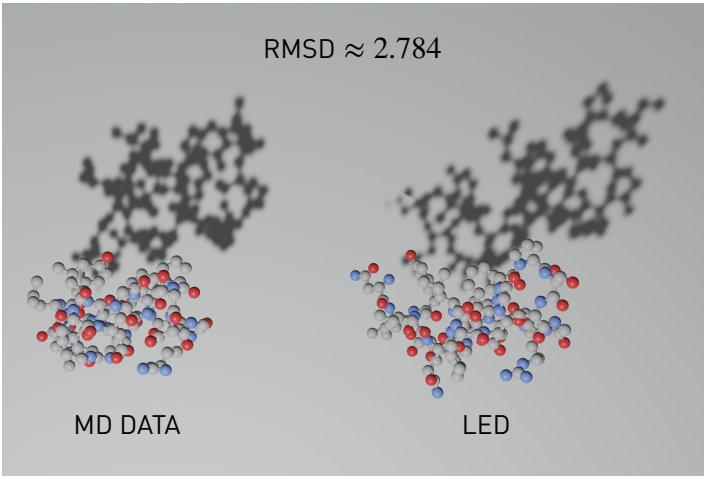


FIGURE E.2: Trp Cage protein configurations found in the MD data compared to a sample of LED that is in close proximity in the latent space. The RMSD error between the two configurations is 2.784$\mathring{A}$.

E.3 ALANINE DIPEPTIDE

A molecule of alanine dipeptide in water is simulated with MD Guzman et al., 2019. The peptide is modeled with the AMBER03 force field Duan et al., 2003, while the water is modeled with TIP3P/Fs Schmitt et al., 1999. The Velocity Verlet algorithm is employed for the integration. The simulation domain is a cubic box (edge length 2.7 nm) with periodic boundary conditions and minimum image convention. The temperature is maintained at 298 K with a local Langevin thermostat Grest et al., 1986, with the value of the friction constant equal to 1.0/ps. The cutoff distance for the nonbonded interactions is $r_c = 0.9$ nm. The reaction field method Neumann, 1985 is used for the electrostatic interaction beyond the cutoff, with the dielectric permittivity of inner and outer regions equal to 1 and 80, respectively.

A timestep of $\delta t = 1$fs is considered, and the dynamics are integrated up to a total time of $T = 100$ns, creating a dataset with a total of 10^8 data samples. The data are subsampled, keeping every 100^{th} data points, creating a trajectory with $N = 10^6$ samples. The coarse timestep of LED is thus $\Delta t = 0.1$ps. The protein positions are transformed into rototranslational invariant features (internal coordinates), composed of bonds, angles, and dihedral angles. The data are split to 248 trajectories of 4000 samples (each trajectory corresponds to $T = 400$ps of MD data), discarding the remaining data. The first 96 trajectories (corresponding to a total of 38.4ns of MD data) are used for training and the subsequent 96 trajectories for validation. All 248 initial conditions are used for testing.

E.3.1 *Metastable State Definition*

The protein is considered to lie in each of the five metastable states $\{C5, P_{II}, \alpha_R, \alpha_L, C_7^{ax}\}$ if the distance in the Ramachandran plot between the protein state and the metastable state center is smaller than 10 degrees. The metastable state centers are defined in Table E.8.

E.3.2 *LED Hyperparameters*

Regarding the LED architecture, the number and size of hidden layers are the same for the encoder $\mathcal{E}$, the decoder $\mathcal{D}$, and the latent MDN $\mathcal{Z}$. The MDN-AE is trained, tuning its hyperparameters based on the grid search reported in Table E.9. The latent space of the MDN-AE is $z \in \mathbb{R}^2$, i.e.,

Metastable state	Center (ϕ, ψ)
P_{II}	$(-75, 150)$
$C5$	$(-155, 155)$
α_R	$(-75, -20)$
α_L	$(67, 5)$
C_7^{ax}	$(70, 160)$

TABLE E.8: Centers of the metastable states in the Ramachandran plot.

$d_z = 1$. The MDN-AE model with the lowest validation error on the state statistics is picked. Then, the MDN-AE is coupled with the MDN-LSTM in LED. The MDN-LSTM is trained to minimize the latent data likelihood. The hyperparameters of the MDN-LSTM are tuned according to the grid search reported in Table E.10. The LED model with the lowest error on the state statistics in the validation dataset is selected. Its hyperparameters are reported in Table E.11. The LED is tested in 248 initial conditions randomly sampled from the testing data. Starting from these initial conditions, we utilize the iterative propagation in the latent space of the LED to forecast $T = 400$ps.

E.3.3 *Marginal State Distributions*

The marginal distributions of the trajectories generated by LED match the ground-truth ones (MD data) closely, as depicted in Figure E.3.

In Figure E.4, a configuration randomly sampled from MD data is given for each metastable state. The closest configuration sampled from LED is compared with the MD data sample in terms of the Root Mean Square Deviation (RMSD) score. The LED samples realistic configurations with low RMSD errors for all metastable states. The mean and standard deviation of the RMSD scores of the 10 closest neighbors sampled from LED are $\mu \pm \sigma = 0.148 \pm 0.021 \mathring{A}$ for the C_5 MD sample configuration (Figure E.4 top left). This score for the rest of the metastable states is $0.340 \pm 0.463 \mathring{A}$ for P_{II}, $0.101 \pm 0.019 \mathring{A}$ for α_R, $0.885 \pm 0.162 \mathring{A}$ for α_L, and $0.383 \pm 0.125 \mathring{A}$ for C_7^{ax}. The LED samples similar configurations with low RMSD scores for the most frequently observed metastable states $\{C_5, P_{II}, \alpha_R\}$. The average

Hyperparameter	Values
Batch size	32
Initial learning rate	10^{-3}
Weight decay rate	$\{0, 10^{-5}\}$
Number of AE layers	$\{4, 6\}$
Size of AE layers	$\{50, 100\}$
Activation of AE layers	selu, tanh
Latent dimension	2
Input/Output data scaling	$[0, 1]$
MDN-AE kernels	5
MDN-AE hidden units	$\{20, 50\}$
MDN-AE multivariate	0
MDN-AE covariance scaling factor	0.8

TABLE E.9: Hyperparameter tuning of AE for alanine dipeptide

Hyperparameter	Values
Batch size	32
Initial learning rate	10^{-3}
BPTT sequence length	$\{200, 400\}$
Number of LSTM layers	1
Size of LSTM layers	$\{10, 20, 40\}$
Activation of LSTM Cell	tanh
MDN-LSTM kernels	$\{4, 5, 6\}$
MDN-LSTM hidden units	$\{10, 20\}$
MDN-LSTM multivariate	$\{0, 1\}$
MDN-LSTM covariance scaling factor	$\{0.1, 0.2, 0.3, 0.4\}$

TABLE E.10: Hyperparameter tuning of LSTM for alanine dipeptide

Hyperparameter	Values
Number of AE layers	4
Size of AE layers	50
Activation of AE layers	tanh
Latent dimension	2
MDN-AE kernels	5
MDN-AE hidden units	50
MDN-AE multivariate	0
MDN-AE covariance scaling factor	0.8
Weight decay rate	0
BPTT sequence length	400
Number of LSTM layers	1
Size of LSTM layers	20
Activation of LSTM Cell	tanh
MDN-LSTM kernels	5
MDN-LSTM hidden units	20
MDN-LSTM multivariate	0
MDN-LSTM covariance scaling factor	0.4

TABLE E.11: Hyperparameters of LED model with lowest validation error on alanine dipeptide

RMSD error is slightly higher and fluctuates more for the less frequently observed $\{\alpha_R, C_7^{ax}\}$.

E.3.4 *Metastable States on the Latent Space and Mean First Passage Times*

The metastable states can be defined on the latent space of LED by projecting the free energy on the latent space and identifying the local minima. This alleviates the need for expert knowledge (definition of the metastable states). The MFPTs between the metastable states on the latent space of the LED are compared with the MFPTs between the corresponding metastable states on the Ramachadran space in Table E.12. Note that the results depend on how the latent metastable states are defined. However, in order to capture the order of the timescales without the need of prior expert knowledge, a rough approximation (small region around the minima in the latent space) is adequate. The LED is able to capture the order of the timescales, alleviating the need for expert knowledge on the definition of the metastable states.

		Metastable states on **Ramachandran Space**		Metastable states on **LED Latent Space**	
MFPT [ns]	MSM − 10ps on MD data Reference	MSM − 10ps on LED − 0.1ps data MFPT	Error (%)	MSM − 10ps on LED − 0.1ps data MFPT	Error (%)
$T_{C_5 \to P_{II}}$	0.105	0.103	**2**	0.143	36
$T_{C_5 \to \alpha_R}$	0.104	0.082	**21**	0.124	19
$T_{P_{II} \to C_5}$	0.226	0.242	**7**	0.356	57
$T_{P_{II} \to \alpha_R}$	0.105	0.083	**21**	0.123	18
$T_{\alpha_R \to C_5}$	0.236	0.258	**9**	0.361	53
$T_{\alpha_R \to P_{II}}$	0.116	0.119	**2**	0.148	27
Average Relative Error		10.51%			35.21%

TABLE E.12: Mean first-passage times (MFPT) between the metastable states of alanine dipeptide in water in [ns]. MFPTs are estimated by fitting MSMs with a time-lag of 10ps on MD trajectories. In LED, the metastable states are considered as regions around the local minima of the free energy projection on the latent space. The average relative error is given for reference.

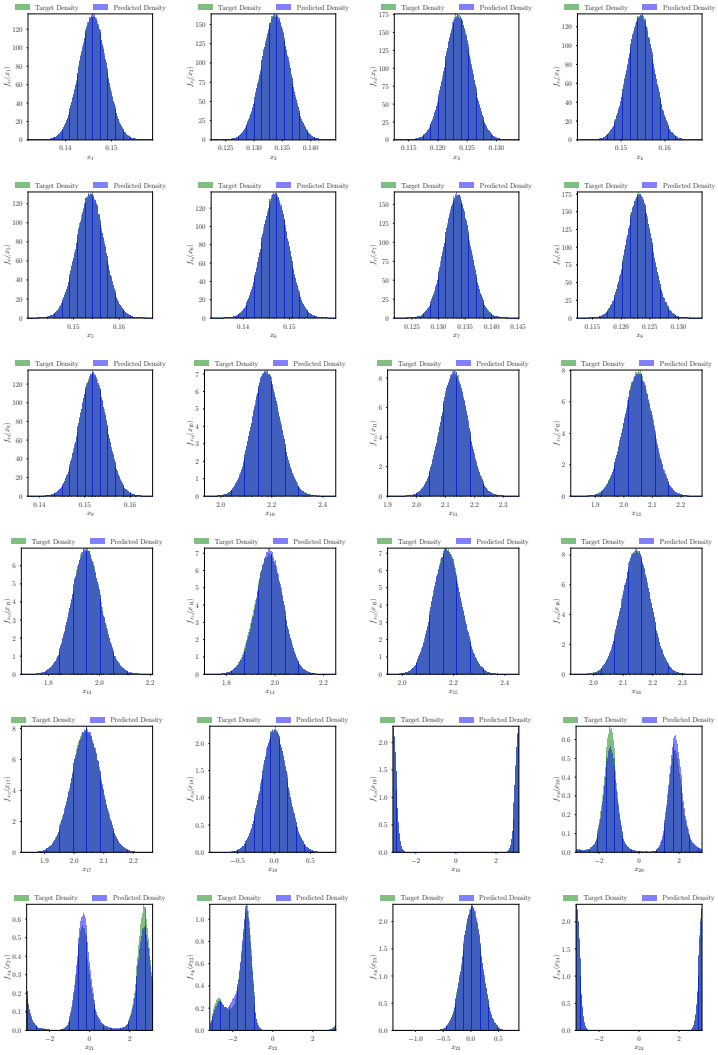


FIGURE E.3: A plot of the marginal state distributions. Comparison of the state distributions estimated from the MD data (test dataset) and from trajectories sampled from LED.

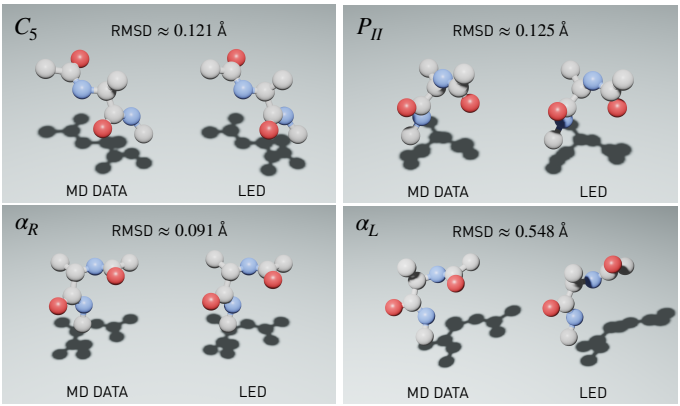


FIGURE E.4: For each metastable state, a random alanine dipeptide configuration sampled from MD data is compared against the closest configuration sampled from the LED with $d_{\mathbf{z}} = 1$. The Root Mean Square Deviation (RMSD) in $\mathring{A}$ between the two is plotted for reference.

BIBLIOGRAPHY

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng (2016). "Tensorflow: a system for large-scale machine learning". In: *12th usenix symposium on operating systems design and implementation (osdi 16)*, 265.

Abarbanel, H. (2012). *Analysis of observed chaotic data*. Springer Science & Business Media.

Abdollahzade, M., A. Miranian, H. Hassani, and H. Iranmanesh (2015). "A new hybrid enhanced local linear neuro-fuzzy model based on the optimized singular spectrum analysis and its application for nonlinear and chaotic time series forecasting". In: *Information sciences* 295, 107.

Ahmad, A. M., S. Ismail, and D. Samaon (2004). "Recurrent neural network with backpropagation through time for speech recognition". In: *Ieee international symposium on communications and information technology, 2004. iscit 2004.* Vol. 1. Ieee, 98.

Alexandridis, A. K. and A. D. Zapranis (2013). "Wavelet neural networks: a practical guide". In: *Neural networks* 42, 1.

Alipanahi, B., A. Delong, M. T. Weirauch, and B. J. Frey (2015). "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning". In: *Nat. biotechnol.* 33.8, 831.

Allgaier, N. A., K. D. Harris, and C. M. Danforth (2012). "Empirical correction of a toy climate model". In: *Phys. rev. e* 85.2, 026201.

Alnæs, M., J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells (2015). "The fenics project version 1.5". In: *Archive of numerical software* 3.100.

Andrychowicz, O. M., B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. (2020). "Learning dexterous in-hand manipulation". In: *The internatl. j. (wash.) of robotics research* 39.1, 3.

Antonik, P., M. Haelterman, and S. Massar (2017). "Brain-inspired photonic signal processor for generating periodic patterns and emulating chaotic systems". In: *Phys. rev. applied* 7 (5), 054014.

Arbabi, H. and I. Mezic (2017). "Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator". In: *Siam j. appl. dyn. syst.* 16.4, 2096.

Arjovsky, M., A. Shah, and Y. Bengio (2016). "Unitary evolution recurrent neural networks". In: *Proceedings of the 33rd international conference on international conference on machine learning - volume 48*. Icml'16. New York, NY, USA: JMLR.org, 1120.

Ayaz, C., L. Tepper, F. N. Brünig, J. Kappler, J. O. Daldrop, and R. R. Netz (2021). "Non-markovian modeling of protein folding". In: *Proc. natl. acad. sci. u.s.a.* 118.31.

Ayton, G. S., W. G. Noid, and G. A. Voth (2007). "Multiscale modeling of biomolecular systems: in serial and in parallel". In: *Curr. opin. struct. biol.* 17, 192.

Bae, H. J. and P. Koumoutsakos (2021). "Scientific multi-agent reinforcement learning for wall-models of turbulent flows". In: *Arxiv preprint arxiv:2106.11144*.

Bakker, B. (2002). "Reinforcement learning with long short-term memory". In: *Adv. neural inf. process. syst.* 1475.

Baldi, P., P. Sadowski, and D. Whiteson (2014). "Searching for exotic particles in high-energy physics with deep learning". In: *Nat. commun.* 5.1, 1.

Balsera, M. A., W. Wriggers, Y. Oono, and K. Schulten (1996). "Principal component analysis and long time protein dynamics". In: *J. phys. chem.* 100.7, 2567.

Bar-Sinai, Y., S. Hoyer, J. Hickey, and M. P. Brenner (2019). "Learning data-driven discretizations for partial differential equations". In: *Proc. natl. acad. sci. u.s.a.* 116.31, 15344.

Bartók, A. P., S. De, C. Poelking, N. Bernstein, J. R. Kermode, G. Csányi, and M. Ceriotti (2017). "Machine learning unifies the modeling of materials and molecules". In: *Sci. adv.* 3.12, e1701816.

Basnarkov, L. and L. Kocarev (2012). "Forecast improvement in lorenz 96 system". In: *Nonlinear processes geophys.* 19.5, 569.

Behler, J. and M. Parrinello (2007). "Generalized neural-network representation of high-dimensional potential-energy surfaces". In: *Phys. rev. lett.* 98, 146401.

Belkin, M., D. Hsu, S. Ma, and S. Mandal (2019). "Reconciling modern machine-learning practice and the classical bias–variance trade-off". In: *Proc. natl. acad. sci. u.s.a.* 116.32, 15849.

Bellomo, N. and C. Dogbe (2011). "On the modeling of traffic and crowds: a survey of models, speculations, and perspectives". In: *Siam rev.* 53.3, 409.

Bengio, S., O. Vinyals, N. Jaitly, and N. Shazeer (2015). "Scheduled sampling for sequence prediction with recurrent neural networks". In: *Adv. neural inf. process. syst.* 28, 1171.

Bengio, Y., P. Simard, and P. Frasconi (1994). "Learning long-term dependencies with gradient descent is difficult". In: *Trans. neur. netw.* 5.2, 157.

Bertasius, G., H. Wang, and L. Torresani (2021). "Is space-time attention all you need for video understanding?" In: *Arxiv preprint arxiv:2102.05095*.

Beucler, T., M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine (2021). "Enforcing analytic constraints in neural networks emulating physical systems". In: *Phys. rev. lett.* 126.9, 098302.

Bhatia, H., T. S. Carpenter, H. I. Ingólfsson, G. Dharuman, P. Karande, S. Liu, T. Oppelstrup, C. Neale, F. C. Lightstone, B. Van Essen, et al. (2021). "Machine-learning-based dynamic-importance sampling for adaptive multiscale simulations". In: *Nat. mach. intell.* 3.5, 401.

Bianchi, F. M., E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen (2017). "An overview and comparative analysis of recurrent neural networks for short term load forecasting". In: *Corr* abs/1705.04378. arXiv: 1705.04378.

Bishop, C. M. (1994). "Mixture density networks". In: *Technical report ncrg/97/004, neural computing research group, aston university*.

Bishop, C. M. (2006). "Pattern recognit." In: *Machine learning* 128.9.

Bittracher, A., R. Banisch, and C. Schütte (2018). "Data-driven computation of molecular reaction coordinates". In: *J. chem. phys.* 149, 154103.

Bittracher, A., S. Klus, B. Hamzi, P. Koltai, and C. Schütte (2021). "Dimensionality reduction of complex metastable systems via kernel embeddings of transition manifolds". In: *J nonlinear sci* 31.1, 1.

Bittracher, A., P. Koltai, S. Klus, R. Banisch, M. Dellnitz, and C. Schütte (2018). "Transition manifolds of complex metastable systems". In: *J. nonlinear sci.* 28.2, 471.

Blonigan, P. J., M. Farazmand, and T. P. Sapsis (2019). "Are extreme dissipation events predictable in turbulent fluid flows?" In: *Phys. rev. fluids* 4.4, 044606.

Blonigan, P. J. and Q. Wang (2014). "Least squares shadowing sensitivity analysis of a modified kuramoto–sivashinsky equation". In: *Chaos, solitons & fractals* 64, 16.

Bonati, L., V. Rizzi, and M. Parrinello (2020). "Data-driven collective variables for enhanced sampling". In: *J. phys. chem. lett.* 11, 2998.

Bongard, J. and H. Lipson (2007). "Automated reverse engineering of nonlinear dynamical systems". In: *Proc. natl. acad. sci. u.s.a.* 104.24, 9943.

Boninsegna, L., G. Gobbo, F. Noé, and C. Clementi (2015). "Investigating molecular kinetics by variationally optimized diffusion maps". In: *J. chem. theory comput.* 11.12, 5947.

Bost, C., G.-H. Cottet, and E. Maitre (2010). "Convergence analysis of a penalization method for the three-dimensional motion of a rigid body in an incompressible viscous fluid". In: *Siam j. numer. anal.* 48.4, 1313.

Bowman, S. R., L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio (2015). "Generating sentences from a continuous space". In: *Arxiv preprint arxiv:1511.06349*.

Bradley, E. and H. Kantz (2015). "Nonlinear time-series analysis revisited". In: *Chaos: an interdisciplinary j nonlinear sci* 25.9, 097610.

Brooks, B. and M. Karplus (1983). "Harmonic dynamics of proteins: normal modes and fluctuations in bovine pancreatic trypsin inhibitor". In: *Pnas* 80.21, 6571.

Broomhead, D. S. and D. Lowe (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Tech. rep. Royal Signals and Radar Establishment Malvern (United Kingdom).

Brunton, S. L., B. R. Noack, and P. Koumoutsakos (2019). "Machine learning for fluid mechanics". In: *Annu. rev. fluid mech.* 52, 477.

Brunton, S. L., B. R. Noack, and P. Koumoutsakos (2020). "Machine learning for fluid mechanics". In: *Annu. rev. fluid mech.* 52, 477.

Brunton, S. L., J. L. Proctor, and J. N. Kutz (2016). "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proc. natl. acad. sci. u.s.a.* 113.15, 3932.

Buchete, N.-V. and G. Hummer (2008). "Coarse master equations for peptide folding dynamics". In: *J. phys. chem. b* 112, 6057.

Butler, K. T., D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh (2018). "Machine learning for molecular and mater. sci." In: *Nature* 559, 547.

Cao, L., Y. Hong, H. Fang, and G. He (1995). "Predicting chaotic time series with wavelet networks". In: *Physica d* 85.1, 225.

Car, R. and M. Parrinello (1985). "Unified approach for molecular dynamics and density-functional theory". In: *Phys. rev. lett.* 55.22, 2471.

Castrejon, L., N. Ballas, and A. Courville (2019). "Improved conditional vrnns for video prediction". In: *Proceedings of the ieee international conference on computer vision*, 7608.

Chakraborty, B. and S. A. Murphy (2014). "Dynamic treatment regimes". In: *Annual review of statistics and its application* 1, 447.

Champion, K. P., S. L. Brunton, and J. N. Kutz (2019). "Discovery of nonlinear multiscale systems: sampling strategies and embeddings". In: *Siam j. appl. dyn. syst.* 18.1, 312.

Chatzis, S. P. and Y. Demiris (2011). "Echo state gaussian process". In: *Ieee transactions on neural networks* 22.9, 1435.

Chekmarev, D. S., T. Ishida, and R. M. Levy (2004). "Long-time conformational transitions of alanine dipeptide in aqueous solution: continuous and discrete-state kinetic models". In: *J. phys. chem. b* 108, 19487.

Chen, H., O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke (2018). "The rise of deep learning in drug discovery". In: *Drug discovery today* 23.6, 1241.

Chen, W., H. Sidky, and A. L. Ferguson (2019). "Nonlinear discovery of slow molecular modes using state-free reversible vampnets". In: *J. chem. phys.* 150, 214114.

Cheng, B., E. A. Engel, J. Behler, C. Dellago, and M. Ceriotti (2019). "Ab initio thermodynamics of liquid and solid water". In: *Pnas* 116, 1110.

Chiavazzo, E., C. W. Gear, C. J. Dsilva, N. Rabin, and I. G. Kevrekidis (2014). "Reduced models in chemical kinetics via nonlinear data-mining". In: *Processes* 2.1, 112.

Chmiela, S., H. E. Sauceda, K.-R. Müller, and A. Tkatchenko (2018). "Towards exact molecular dynamics simulations with machine-learned force fields". In: *Nat. commun.* 9, 1.

Cho, K., B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio (2014). "Learning phrase representations using rnn encoder–decoder for statistical machine translation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)*. Doha, Qatar: Association for Computational Linguistics, 1724.

Chui, C. K., J. M. Lemm, and S. Sedigh (1992). *An introduction to wavelets*. Vol. 1. Academic press.

Chung, J., C. Gulcehre, K. Cho, and Y. Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling". English (US). In: *Nips 2014 workshop on deep learning, december 2014*.

Chung, J., K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio (2015). "A recurrent latent variable model for sequential data". In: *Adv. neural inf. process. syst.* 28, 2980.

Coifman, R. R., I. G. Kevrekidis, S. Lafon, M. Maggioni, and B. Nadler (2008). "Diffusion maps, reduction coordinates, and low dimensional representation of stochastic systems". In: *Multiscale model. simul.* 7.2, 842.

Coifman, R. R. and S. Lafon (2006). "Diffusion maps". In: *Appl. comput. harmon. anal.* 21.1, 5.

Comeau, D., Z. Zhao, D. Giannakis, and A. J. Majda (2017). "Data-driven prediction strategies for low-frequency patterns of north pacific climate variability". In: *Climate dyn.* 48.5-6, 1855.

Council, N. R. (2012). *A national strategy for advancing climate modeling*. The National Academies Press.

Cousins, W. and T. P. Sapsis (2014). "Quantification and prediction of extreme events in a one-dimensional nonlinear dispersive wave model". In: *Physica d* 280, 48.

Cousins, W. and T. P. Sapsis (2016). "Reduced-order precursors of rare events in unidirectional nonlinear water waves". In: *J. fluid mech.* 790, 368.

Crommelin, D. and A. Majda (2004). "Strategies for model reduction: comparing different optimal bases". In: *J. atmos. sci.* 61.17, 2206.

Cunningham, J. P. and Z. Ghahramani (2015). "Linear dimensionality reduction: survey, insights, and generalizations". In: *The journal of machine learning research* 16.1, 2859.

Cvitanović, P., R. L. Davidchack, and E. Siminos (2010). "On the state space geometry of the kuramoto–sivashinsky flow in a periodic domain". In: *Siam j. appl. dyn. syst.* 9.1, 1.

Dalcín, L., R. Paz, M. Storti, and J. D'Elía (2008). "Mpi for python: performance improvements and mpi-2 extensions". In: *J. parallel distrib. comput.* 68.5, 655.

Dalcín, L. D., R. R. Paz, P. A. Kler, and A. Cosimo (2011). "Parallel distributed computing using python". In: *Advances in water resour.* 34.9, 1124.

Davies, A., P. Veličković, L. Buesing, S. Blackwell, D. Zheng, N. Tomašev, R. Tanburn, P. Battaglia, C. Blundell, A. Juhász, et al. (2021). "Advancing mathematics by guiding human intuition with ai". In: *Nature* 600.7887, 70.

Dechert, W. D. and R. Gençay (1996). "The topological invariance of lyapunov exponents in embedded dynamics". In: *Physica d nonlinear phenomena* 90, 40.

Dellago, C., P. G. Bolhuis, and D. Chandler (1998). "Efficient transition path sampling: application to lennard-jones cluster rearrangements". In: *J. chem. phys.* 108, 9236.

Deng, Y., F. Bao, Y. Kong, Z. Ren, and Q. Dai (2016). "Deep direct reinforcement learning for financial signal representation and trading". In: *Ieee trans neural netw learn syst* 28.3, 653.

Dennis, J. M. and H. M. Tufo (2008). "Scaling climate simulation applications on the ibm blue gene/l system". In: *Ibm j. res. dev.* 52.1.2, 117.

Dreyfus, S. (1962). "The numerical solution of variational problems". In: *J. math. anal* 5.1, 30.

Duan, Y., C. Wu, S. Chowdhury, M. C. Lee, G. Xiong, W. Zhang, R. Yang, P. Cieplak, R. Luo, T. Lee, et al. (2003). "A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations". In: *J. comput. chem.* 24.16, 1999.

Duriez, T., S. L. Brunton, and B. R. Noack (2017). *Machine learning control-taming nonlinear dyn. and turbulence*. Springer.

Durumeric, A. E. and G. A. Voth (2019). "Adversarial-residual-coarse-graining: applying machine learning theory to systematic molecular coarse-graining". In: *J. chem. phys.* 151.12, 124110.

Einicke, G. A. and L. B. White (1999). "Robust extended kalman filtering". In: *Ieee transactions on signal process.* 47.9, 2596.

El Saddik, A. (2018). "Digital twins: the convergence of multimedia technologies". In: *Ieee multimedia* 25.2, 87.

Elman, J. L. (1990). "Finding structure in time". In: *Cognitive science* 14.2, 179.

Erban, R., I. G. Kevrekidis, D. Adalsteinsson, and T. C. Elston (2006). "Gene regulatory networks: a coarse-grained, equation-free approach to multiscale computation". In: *The j. chem. phys.* 124.8, 084106.

Errica, F., D. Bacciu, and A. Micheli (2021). "Graph mixture density networks". In: *International conference on machine learning*. PMLR, 3025.

Faber, F. A., L. Hutchison, B. Huang, J. Gilmer, S. S. Schoenholz, G. E. Dahl, O. Vinyals, S. Kearnes, P. F. Riley, and O. A. Von Lilienfeld (2017). "Prediction errors of molecular machine learning models lower than hybrid dft error". In: *J. chem. theory comput.* 13.11, 5255.

Faller, W. E. and S. J. Schreck (1997). "Unsteady fluid mechanics applications of neural networks". In: *Journal of aircraft* 34.1, 48.

Farazmand, M. and T. P. Sapsis (2016). "Dynamical indicators for the prediction of bursting phenomena in high-dimensional systems". In: *Phys. rev. e* 94.3, 032212.

Farge, M., N. Kevlahan, V. Perrier, and E. Goirand (1996). "Wavelets and turbulence". In: *Proceedings of the ieee* 84.4, 639.

FitzHugh, R. (1961). "Impulses and physiological states in theoretical models of nerve membrane". In: *Biophys. j.* 1.6, 445.

Forrester, A. I., N. W. Bressloff, and A. J. Keane (2006). "Optimization using surrogate models and partially converged computational fluid dyn. simulations". In: *Proc. math. phys. eng. sci.* 462.2071, 2177.

Fragkiadaki, K., P. Agrawal, S. Levine, and J. Malik (2015). "Learning visual predictive models of physics for playing billiards". In: *Arxiv preprint arxiv:1511.07404*.

Frouzakis, C. E., L. Gardini, I. G. Kevrekidis, G. Millerioux, and C. Mira (1997). "On some properties of invariant sets of two-dimensional noninvertible maps". In: *Internatl. j. (wash.) of bifurcation and chaos* 7.06, 1167.

Gal, Y. and Z. Ghahramani (2016). "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in neural information processing systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 1019.

Geneva, N. and N. Zabaras (2020). "Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks". In: *J. comput. phys.* 403, 109056.

Gers, F. A., D. Eck, and J. Schmidhuber (2002). "Applying lstm to time series predictable through time-window approaches". In: *Neural nets wirn vietri-01*. Springer, 193.

Gers, F. A., N. N. Schraudolph, and J. Schmidhuber (2002). "Learning precise timing with lstm recurrent networks". In: *J. mach. learn. res.* 3.Aug, 115.

Gicquel, N., J. Anderson, and I. Kevrekidis (1998). "Noninvertibility and resonance in discrete-time neural networks for time-series processing". In: *Phys. lett. a* 238.1, 8.

Gilmour, D., M. Rembold, and M. Leptin (2017). "From morphogen to morphogenesis and back". In: *Nature* 541.7637, 311.

Glorot, X. and Y. Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artif. intell. and statistics*, 249.

Gonon, L. and J.-P. Ortega (2019). "Reservoir computing universality with stochastic inputs". In: *Ieee trans neural netw learn syst*.

Gonzalez, F. J. and M. Balajewicz (2018). "Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems". In: *Arxiv preprint arxiv:1808.01346*.

Goodfellow, I., Y. Bengio, A. Courville, and Y. Bengio (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Graves, A., S. Fernández, M. Liwicki, H. Bunke, and J. Schmidhuber (2008). "Unconstrained online handwriting recognition with recurrent neural

networks". In: *Advances in neural information processing systems 20, nips 2008*.

Graves, A., A.-r. Mohamed, and G. Hinton (2013). "Speech recognition with deep recurrent neural networks". In: *2013 ieee international conference on acoustics, speech and signal process.* Ieee, 6645.

Greff, K., R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber (2016). "Lstm: a search space odyssey". In: *Ieee trans neural netw learn syst* 28.10, 2222.

Gregor, K., I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra (2015). "Draw: a recurrent neural network for image generation". In: *Arxiv preprint arxiv:1502.04623*.

Gregor, K., I. Danihelka, A. Mnih, C. Blundell, and D. Wierstra (2014). "Deep autoregressive networks". In: *International conference on machine learning*. Pmlr, 1242.

Grest, G. S. and K. Kremer (1986). "Molecular dynamics simulation for polymers in the presence of a heat bath". In: *Phys. rev. a* 33, 3628.

Grigoryeva, L. and J.-P. Ortega (2018). "Echo state networks are universal". In: *Neural networks* 108, 495.

Gu, J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. (2018). "Recent advances in convolutional neural networks". In: *Pattern recognit.* 77, 354.

Guzman, H. V., N. Tretyakov, H. Kobayashi, A. C. Fogarty, K. Kreis, J. Krajniak, C. Junghans, K. Kremer, and T. Stuehn (2019). "Espresso++ 2.0: advanced methods for multiscale mol. simul." In: *Comput. phys. commun.* 238, 66.

Han, J., A. Jentzen, and E. Weinan (2018). "Solving high-dimensional partial differential equations using deep learning". In: *Proc. natl. acad. sci. u.s.a.* 115.34, 8505.

Hansen, K., G. Montavon, F. Biegler, S. Fazli, M. Rupp, M. Scheffler, O. A. Von Lilienfeld, A. Tkatchenko, and K.-R. Muller (2013). "Assessment and validation of machine learning methods for predicting molecular atomization energies". In: *J. chem. theory comput.* 9.8, 3404.

Hasegawa, K., K. Fukami, T. Murata, and K. Fukagata (2020). "Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes". In: *Theor. comput. fluid dyn.* 34.4, 367.

Haynes, N. D., M. C. Soriano, D. P. Rosin, I. Fischer, and D. J. Gauthier (2015). "Reservoir computing with a single time-delay autonomous boolean node". In: *Phys. rev. e* 91 (2), 020801.

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep residual learning for image recognition". In: *Cvpr*, 770.

Hejazialhosseini, B., D. Rossinelli, M. Bergdorf, and P. Koumoutsakos (2010). "High order finite volume methods on wavelet-adapted grids with local time-stepping on multicore architectures for the simulation of shock-bubble interactions". In: *J. comput. phys.* 229.22, 8364.

Hernández, C. X., H. K. Wayment-Steele, M. M. Sultan, B. E. Husic, and V. S. Pande (2018). "Variational encoding of complex dynamics". In: *Phys. rev. e* 97.6, 062412.

Hess, B., S. León, N. Van Der Vegt, and K. Kremer (2006). "Long time atomistic polymer trajectories from coarse grained simulations: bisphenol-a polycarbonate". In: *Soft matter* 2.5, 409.

Higham, D. J. and N. J. Higham (2016). *Matlab guide*. SIAM.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory". In: *Neural comput.* 9, 1735.

Hochreiter, S. (1991). "Untersuchungen zu dynamischen neuronalen netzen". In: *Diploma, technische universität münchen* 91.1.

Hochreiter, S. (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *Internatl. j. (wash.) of uncertainty, fuzziness and knowledge-based systems* 6, 107.

Huber, T., A. E. Torda, and W. F. V. Gunsteren (1994). "Local elevation: a method for improving the searching properties of molecular dynamics simulation". In: *J. comput. aided mol. des.* 8, 695.

Ichiye, T. and M. Karplus (1991). "Collective motions in proteins: a covariance analysis of atomic fluctuations in molecular dynamics and normal mode simulations". In: *Proteins* 11.3, 205.

Imbalzano, G., A. Anelli, D. Giofré, S. Klees, J. Behler, and M. Ceriotti (2018). "Automatic selection of atomic fingerprints and reference configurations for machine-learning potentials". In: *J. chem. phys.* 148.24, 241730.

Inizan, T. J., F. Célerse, O. Adjoua, D. El Ahdab, L.-H. Jolly, C. Liu, P. Ren, M. Montes, N. Lagarde, L. Lagardère, et al. (2021). "High-resolution mining of the sars-cov-2 main protease conformational space: supercomputer-driven unsupervised adaptive sampling". In: *Chem. sci.* 12.13, 4889.

Jaeger, H. and H. Haas (2004). "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication". In: *Science* 304.5667, 78. eprint: http://science.sciencemag.org/content/304/5667/78.full.pdf.

Jang, H. and T. B. Woolf (2006). "Multiple pathways in conformational transitions of the alanine dipeptide: an application of dynamic importance sampling". In: *J. comput. chem.* 27, 1136.

Jiang, J. and Y.-C. Lai (2019). "Model-free prediction of spatiotemporal dynamical systems with recurrent neural networks: role of network spectral radius". In: *Phys. rev. research* 1 (3), 033056.

Jing, L., Y. Shen, T. Dubcek, J. Peurifoy, S. A. Skirlo, Y. LeCun, M. Tegmark, and M. Soljacic (2017). "Tunable efficient unitary neural networks (EUNN) and their application to rnns". In: *Icml*. Vol. 70. Proceedings of Machine Learning Research. Pmlr, 1733.

Julier, S. J. and J. K. Uhlmann (1997). "New extension of the kalman filter to nonlinear systems". In: *Signal process., sensor fusion, and target recognition vi*. Vol. 3068. International Society for Optics and Photonics, 182.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. (2021). "Highly accurate protein structure prediction with alphafold". In: *Nature* 596.7873, 583.

Kabsch, W. (1976). "A solution for the best rotation to relate two sets of vectors". In: *Acta crystallogr. a* 32, 922.

Kantz, H. and T. Schreiber (1997). *Nonlinear time series analysis*. New York, NY, USA: Cambridge University Press.

Kaplan, J. L. and J. A. Yorke (1979). "Chaotic behavior of multidimensional difference equations". In: *Functional differential equations and approximation of fixed points*. Ed. by H.-O. Peitgen and H.-O. Walther. Berlin, Heidelberg: Springer Berlin Heidelberg, 204.

Karlin, I. V., S. Ansumali, C. E. Frouzakis, and S. S. Chikatamarla (2006). "Elements of the lattice boltzmann method i: linear advection equation". In: *Commun. comput. phys* 1.4, 616.

Karplus, M. and J. A. McCammon (2002). "Molecular dynamics simulations of biomolecules". In: *Nat. struct. mol. biol.* 9, 646.

Kassam, A.-K. and L. N. Trefethen (2005). "Fourth-order time-stepping for stiff pdes". In: *Siam j. sci. comput.* 26.4, 1214.

Kerschen, G., J.-c. Golinval, A. F. Vakakis, and L. A. Bergman (2005). "The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: an overview". In: *Nonlinear dyn.* 41.1, 147.

Kevrekidis, I. G., C. W. Gear, and G. Hummer (2004). "Equation-free: the computer-aided analysis of complex multiscale systems". In: *Aiche j.* 50.7, 1346.

Kevrekidis, I. G., C. W. Gear, J. M. Hyman, P. G. Kevrekidid, O. Runborg, C. Theodoropoulos, et al. (2003). "Equation-free, coarse-grained multiscale computation: enabling mocroscopic simulators to perform system-level analysis". In: *Commun math sci* 1.4, 715.

Kevrekidis, I. G., B. Nicolaenko, and J. C. Scovel (1990). "Back in the saddle again: a computer assisted study of the kuramoto–sivashinsky equation". In: *Siam j. appl. math.* 50.3, 760.

Kevrekidis, I. G. and G. Samaey (2009). "Equation-free multiscale computation: algorithms and applications". In: *Annu. rev. phys. chem.* 60, 321.

Kim, K. B., J. B. Park, Y. H. Choi, and G. Chen (2000). "Control of chaotic dynamical systems using radial basis function network approximators". In: *Information sciences* 130.1-4, 165.

Kim, S.-J., T. J. Crowley, D. J. Erickson, B. Govindasamy, P. B. Duffy, and B. Y. Lee (2008). "High-resolution climate simulation of the last glacial maximum". In: *Climate dyn.* 31.1, 1.

Kingma, D. P. and J. Ba (2014). "Adam: a method for stochastic optimization". In: *Arxiv preprint arxiv:1412.6980*.

Kramer, M. A. (1991). "Nonlinear principal component analysis using autoassociative neural networks". In: *Aiche j.* 37.2, 233.

Krischer, K., R. Rico-Martínez, I. Kevrekidis, H. Rotermund, G. Ertl, and J. Hudson (1993). "Model identification of a spatiotemporally varying catalytic reaction". In: *Aiche j.* 39.1, 89.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2017). "Imagenet classification with deep convolutional neural networks". In: *Commun. acm*. Nips'12 60.6, 84.

Krueger, D., T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. C. Courville, and C. J. Pal (2017). "Zoneout: regularizing rnns by randomly preserving hidden activations". In: *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings*.

Kumar, A., T. Islam, Y. Sekimoto, C. Mattmann, and B. Wilson (2020). "Convcast: an embedded convolutional lstm based architecture for precipitation nowcasting using satellite data". In: *Plos one* 15.3, e0230114.

Kuramoto, Y. (1978). "Diffusion-induced chaos in reaction systems". In: *Progress of theoretical physics supplement* 64, 346.

Kuramoto, Y. and T. Tsuzuki (1976). "Persistent propagation of concentration waves in dissipative media far from thermal equilibrium". In: *Progress of theoretical physics* 55.2, 356.

Kurth, T., S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, et al. (2018). "Exascale deep learning for climate analytics". In: *Sc18: international conference for high performance computing, networking, storage and analysis*. Ieee, 649.

Kutz, J. N., S. L. Brunton, B. W. Brunton, and J. L. Proctor (2016). *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM.

Kutz, J. N., X. Fu, and S. L. Brunton (2016). "Multiresolution dynamic mode decomposition". In: *Siam j. appl. dyn. syst.* 15.2, 713.

Laing, C. R., T. Frewen, and I. G. Kevrekidis (2010). "Reduced models for binocular rivalry". In: *J. comput. neurosci.* 28.3, 459.

Laio, A. and M. Parrinello (2002). "Escaping free-energy minima". In: *Pnas* 99, 12562.

Lakshminarayanan, B., A. Pritzel, and C. Blundell (2016). "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Arxiv preprint arxiv:1612.01474*.

Lange, H., S. L. Brunton, and J. N. Kutz (2021). "From fourier to koopman: spectral methods for long-term time series prediction." In: *J. mach. learn. res.* 22.41, 1.

Lapedes, A. and R. Farber (1987). *Nonlinear signal process. using neural networks: prediction and system modelling*. Tech. rep.

Laptev, N., J. Yosinski, L. E. Li, and S. Smyl (2017). "Time-series extreme event forecasting with neural networks at uber". In.

Larger, L., M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutierrez, L. Pesquera, C. R. Mirasso, and I. Fischer (2012). "Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing". In: *Opt. express* 20.3, 3241.

Larger, L., A. Baylón-Fuentes, R. Martinenghi, V. S. Udaltsov, Y. K. Chembo, and M. Jacquot (2017). "High-speed photonic reservoir computing using a time-delay-based architecture: million words per second classification". In: *Phys. rev. x* 7 (1), 011015.

LeCun, Y. (1985). "Une procedure d'apprentissage ponr reseau a seuil asymetrique". In: *Proceedings of cognitiva 85*, 599.

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep learning". In: *Nature* 521.7553, 436.

Lee, E. H., J. Hsin, M. Sotomayor, G. Comellas, and K. Schulten (2009). "Discovery through the computational microscope". In: *Structure* 17.10, 1295.

Lee, J. A. and M. Verleysen (2007). *Nonlinear dimensionality reduction*. Springer Science & Business Media.

Lee, S., M. Kooshkbaghi, K. Spiliotis, C. I. Siettos, and I. G. Kevrekidis (2020). "Coarse-scale pdes from fine-scale observations via machine learning". In: *Chaos: an interdisciplinary j nonlinear sci* 30.1, 013141.

Lee, Y. and A. J. Majda (2016). "State estimation and prediction using clustered particle filters". In: *Proc. natl. acad. sci. u.s.a.* 113.51, 14609.

Li, Q., L. Shen, S. Guo, and Z. Lai (2020). "Wavelet integrated cnns for noise-robust image classification". In: *Proceedings of the ieee/cvf conference on computer vision and pattern recognit.* 7245.

Li, Y., R. Yu, C. Shahabi, and Y. Liu (2017). "Diffusion convolutional recurrent neural network: data-driven traffic forecasting". In: *Arxiv preprint arxiv:1707.01926*.

Li, Z., K. Meidani, P. Yadav, and A. B. Farimani (2021). "Graph neural networks accelerated molecular dynamics". In: *Arxiv preprint arxiv:2112.03383*.

Lillicrap, T. P. and A. Santoro (2019). "Backpropagation through time and the brain". In: *Curr. opin. neurobiol.* 55, 82.

Lin, J.-L. and C. W. Granger (1994). "Forecasting from non-linear models in pract." In: *Journal of forecasting* 13.1, 1.

Lin, Y. and L. Peng (2011). "Combined model based on emd-svm for short-term wind power prediction". In: *Proceedings of the csee* 31.31, 102.

Linnainmaa, S. (1976). "Taylor expansion of the accumulated rounding error". In: *Bit numerical mathematics* 16.2, 146.

Linot, A. J. and M. D. Graham (2020). "Deep learning to discover and predict dynamics on an inertial manifold". In: *Phys. rev. e* 101.6, 062209.

Lorenz, E. (1995). "Predictability: a problem partly solved". In: *Seminar on predictability, 4-8 september 1995*. Vol. 1. Ecmwf. Shinfield Park, Reading: Ecmwf, 1.

Lorenz, E. N. (1969). "Atmospheric predictability as revealed by naturally occurring analogues". In: *J. atmos. sci.s* 26.4, 636.

Lu, Z., B. R. Hunt, and E. Ott (2018). "Attractor reconstruction by machine learning". In: *Chaos: an interdisciplinary j nonlinear sci* 28.6, 061104.

Lukoševičius, M. (2012). "A practical guide to applying echo state networks". In: *Neural networks: tricks of the trade*.

Lukoševičius, M. and H. Jaeger (2009). "Reservoir computing approaches to recurrent neural network training". In: *Computer science review* 3.3, 127.

Lusch, B., J. N. Kutz, and S. L. Brunton (2018). "Deep learning for universal linear embeddings of nonlinear dyn." In: *Nat. commun.* 9.1, 1.

Maass, W., T. Natschläger, and H. Markram (2002). "Real-time computing without stable states: a new framework for neural comput. based on perturbations". In: *Neural comput.* 14.11, 2531.

Mahadevan, A. (2016). "The impact of submesoscale physics on primary productivity of plankton". In: *Annual review of marine science* 8, 161.

Majda, A., R. V. Abramov, and M. J. Grote (2005). *Information theory and stochastics for multiscale nonlinear systems*. Vol. 25. American Mathematical Soc.

Majda, A. J. and J. Harlim (2012). *Filtering complex turbulent systems*. Cambridge University Press.

Majda, A. J. and Y. Lee (2014). "Conceptual dynamical models for turbulence". In: *Proc. natl. acad. sci. u.s.a.* 111.18, 6548.

Makarenko, A. V. (2018). "Deep convolutional neural networks for chaos identification in signal process." In: *2018 26th european signal process. conference (eusipco)*. Ieee, 1467.

Mandralis, I., P. Weber, G. Novati, and P. Koumoutsakos (2021). "Learning swimming escape patterns for larval fish under energy constraints". In: *Phys. rev. fluids* 6.9, 093101.

Manneville, P. (1984). "Macroscopic modelling of turbulent flows, proceedings of a workshop held at inria, sophia-antipolis, france, 1984". In: *Lect. notes phys.* 230, 319.

Mannion, P., J. Duggan, and E. Howley (2016). "An experimental review of reinforcement learning algorithms for adaptive traffic signal control". In: *Autonomic road transport support systems*, 47.

Maragliano, L., A. Fischer, E. Vanden-Eijnden, and G. Ciccotti (2006). "String method in collective variables: minimum free energy paths and isocommittor surfaces". In: *J. chem. phys.* 125, 024106.

Mardt, A., L. Pasquali, H. Wu, and F. Noé (2018). "Vampnets for deep learning of molecular kinetics". In: *Nat. commun.* 9.1, 1.

Marques, C., J. Ferreira, A. Rocha, J. Castanheira, P. Melo-Gonçalves, N. Vaz, and J. Dias (2006). "Singular spectrum analysis and forecasting of hydrological time series". In: *Phys. chem. earth., parts a/b/c* 31.18, 1172.

Mathieu, M., C. Couprie, and Y. LeCun (2015). "Deep multi-scale video prediction beyond mean square error". In: *Arxiv preprint arxiv:1511.05440*.

Maulik, R., B. Lusch, and P. Balaprakash (2021). "Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders". In: *Phys. fluids* 33.3, 037106.

Maus, A. and J. C. Sprott (2013). "Evaluating lyapunov exponent spectra with neural networks". In: *Chaos, solitons and fractals* 51, 13.

McCarty, J. and M. Parrinello (2017). "A variational conformational dynamics approach to the selection of collective variables in metadynamics". In: *J. chem. phys.* 147.20, 204109.

Michie, D. (1968). ""memo" functions and machine learning". In: *Nature* 218.5136, 19.

Milano, M. and P. Koumoutsakos (2002). "Neural network modeling for near wall turbulent flow". In: *J. comput. phys.* 182.1, 1.

Miller, J. and M. Hardt (2018). "When recurrent models don't need to be recurrent". In: *Arxiv preprint arxiv:1805.10369* 4.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, 529.

Mo, K. C. and R. E. Livezey (1986). "Tropical-extratropical geopotential height teleconnections during the northern hemisphere winter". In: *Mon. weather rev.* 114.12, 2488.

Monthly values of the Darwin Sea Level Pressure series, 1882-1998 (n.d.). http://research.jisao.washington.edu/data_sets/darwin.

Müller, K. and L. D. Brown (1979). "Location of saddle points and minimum energy paths by a constrained simplex optimization procedure". In: *Theor. chim. acta* 53.1, 75.

Nagumo, J., S. Arimoto, and S. Yoshizawa (1962). "An active pulse transmission line simulating nerve axon". In: *Proc. ire* 50.10, 2061.

Neofotistos, G., M. Mattheakis, G. D. Barmparis, J. Hizanidis, G. P. Tsironis, and E. Kaxiras (2019). "Machine learning with observers predicts complex spatiotemporal behavior". In: *Front. phys.* 7, 24.

Neumann, M. (1985). "The dielectric constant of water. computer simulations with the mcy potential". In: *J. chem. phys.* 82, 5663.

Noé, F., A. Tkatchenko, K.-R. Müller, and C. Clementi (2020). "Machine learning for mol. simul." In: *Annu. rev. phys. chem.* 71, 361.

Noé, F., S. Doose, I. Daidone, M. Löllmann, M. Sauer, J. D. Chodera, and J. C. Smith (2011). "Dynamical fingerprints for probing individual relaxation processes in biomolecular dynamics with simulations and kinetic experiments". In: *Pnas* 108.12, 4822.

Noé, F., S. Olsson, J. Köhler, and H. Wu (2019). "Boltzmann generators: sampling equilibrium states of many-body systems with deep learning". In: *Science* 365.6457, eaaw1147.

Noid, W. G. (2013). "Perspective: coarse-grained models for biomolecular systems". In: *J. chem. phys.* 139, 09b201_1.

Norouzi, M., S. Bengio, N. Jaitly, M. Schuster, Y. Wu, D. Schuurmans, et al. (2016). "Reward augmented maximum likelihood for neural structured prediction". In: *Advances in neural information processing systems*, 1723.

Novati, G., H. L. de Laroussilhe, and P. Koumoutsakos (2021). "Automating turbulence modelling by multi-agent reinforcement learning". In: *Nat. mach. intell.* 3.1, 87.

Novati, G. and P. Koumoutsakos (2019). "Remember and forget for experience replay". In: *International conference on machine learning*. PMLR, 4851.

Novati, G., L. Mahadevan, and P. Koumoutsakos (2019). "Controlled gliding and perching through deep-reinforcement-learning". In: *Phys. rev. fluids* 4.9, 093902.

Nuske, F., B. G. Keller, G. Pérez-Hernández, A. S. Mey, and F. Noé (2014). "Variational approach to molecular kinetics". In: *J. chem. theory comput.* 10.4, 1739.

Oh, J., X. Guo, H. Lee, R. L. Lewis, and S. Singh (2015). "Action-conditional video prediction using deep networks in atari games". In: *Adv. neural inf. process. syst.* 28, 2863.

Oord, A. v. d., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu (2016). "Wavenet: a generative model for raw audio". In: *Arxiv preprint arxiv:1609.03499*.

Oord, A. v. d., N. Kalchbrenner, and K. Kavukcuoglu (2016). "Pixel recurrent neural networks". In: *Arxiv preprint arxiv:1601.06759*.

Ott, E. (2002). *Chaos in dynamical systems*. 2nd ed. Cambridge University Press.

Parker, D. B. (1985). "Learnins logic." In: *Technical report*.

Parlitz, U. and C. Merkwirth (2000). "Prediction of spatiotemporal time series based on reconstructed local states". In: *Phys. rev. lett.* 84 (9), 1890.

Partal, T. and H. K. Cigizoglu (2009). "Prediction of daily precipitation using wavelet—neural networks". In: *Hydrol. sci. j.* 54.2, 234.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th international conference on international conference on machine learning - volume 28*. Icml'13. Atlanta, GA, USA: JMLR.org, Iii-1310.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019). "Pytorch: an imperative style, high-performance deep learning library". In: *Adv. neural inf. process. syst.* 8026.

Pathak, J., B. Hunt, M. Girvan, Z. Lu, and E. Ott (2018). "Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach". In: *Phys. rev. lett.* 120.2 (2), 024102.

Pathak, J., Z. Lu, B. R. Hunt, M. Girvan, and E. Ott (2017). "Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data". In: *Chaos: an interdisciplinary j nonlinear sci* 27.12, 121102.

Pathak, J., A. Wikner, R. Fussell, S. Chandra, B. R. Hunt, M. Girvan, and E. Ott (2018). "Hybrid forecasting of chaotic processes: using machine learning in conjunction with a knowledge-based model". In: *Chaos: an interdisciplinary j nonlinear sci* 28.4. Exported from https://app.dimensions.ai on 2019/02/13, 041101.

Pennington, J., R. Socher, and C. Manning (2014). "Glove: global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)*, 1532.

Pérez-Hernández, G. and F. Noé (2016). "Hierarchical time-lagged independent component analysis: computing slow modes and reaction coordinates for large molecular systems". In: *J. chem. theory comput.* 12, 6118.

Peters, B. and B. L. Trout (2006). "Obtaining reaction coordinates by likelihood maximization". In: *J. chem. phys.* 125, 054108.

Petter Langtangen, H. and A. Logg (2017). *Solving pdes in python: the fenics tutorial i*.

Pezeshkian, W., M. König, T. A. Wassenaar, and S. J. Marrink (2020). "Backmapping triangulated surfaces to coarse-grained membrane models". In: *Nat. commun.* 11, 1.

Praprotnik, M. and D. Janežič (2005). "Molecular dynamics integration meets standard theory of molecular vibrations". In: *J. chem. inf. model.* 45, 1571.

Praprotnik, M., L. D. Site, and K. Kremer (2008). "Multiscale simulation of soft matter: from scale bridging to adaptive resolution". In: *Annu. rev. phys. chem.* 59, 545.

Preto, J. and C. Clementi (2014). "Fast recovery of free energy landscapes via diffusion-map-directed molecular dynamics". In: *Phys. chem. chem. phys.* 16.36, 19181.

Quade, M., M. Abel, K. Shafi, R. K. Niven, and B. R. Noack (2016). "Prediction of dynamical systems by symbolic regression". In: *Phys. rev. e* 94.1, 012214.

Rackovsky, S. and H. A. Scheraga (2020). "The structure of protein dynamic space". In: *Proc. natl. acad. sci. u.s.a.* 117.33, 19938.

Raissi, M., P. Perdikaris, and G. E. Karniadakis (2019). "Physics-informed neural networks: a deep learning framework for solving forward and

inverse prob. involving nonlinear partial differential equations". In: *J. comput. phys.* 378, 686.

Ramachandran, G. N. (1963). "Stereochemistry of polypeptide chain configurations". In: *J. mol. biol.* 7, 95.

Ranzato, M., S. Chopra, M. Auli, and W. Zaremba (2015). "Sequence level training with recurrent neural networks". In: *Arxiv preprint arxiv:1511.06732*.

Rasmussen, C. E. (2003). "Gaussian processes in machine learning". In: *Summer school on machine learning*. Springer, 63.

Rasp, S., P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey (2020). "Weatherbench: a benchmark dataset for data-driven weather forecasting". In: *Arxiv preprint arxiv:2002.00469*.

Rasthofer, U., F. Wermelinger, P. Hadijdoukas, and P. Koumoutsakos (2017). "Large scale simulation of cloud cavitation collapse". In: *Procedia comput. sci.* 108, 1763.

Ribeiro, J. M. L., P. Bravo, Y. Wang, and P. Tiwary (2018). "Reweighted autoencoded variational Bayes for enhanced sampling (rave)". In: *J. chem. phys.* 149, 072301.

Rico-Martinez, R., K. Krischer, I. Kevrekidis, M. Kube, and J. Hudson (1992). "Discrete-vs. continuous-time nonlinear signal process. of cu electrodissolution data". In: *Chem. eng. commun.* 118.1, 25.

Robinson, J. C. (1994). "Inertial manifolds for the kuramoto-sivashinsky equation". In: *Phys. lett. a* 184.2, 190.

Robinson, P. A., C. J. Rennie, D. L. Rowe, S. C. O'Connor, and E. Gordon (2005). "Multiscale brain modelling". In: *Philos. trans. r. soc. lond., b, biol. sci.* 360.1457, 1043.

Rohrdanz, M. A., W. Zheng, M. Maggioni, and C. Clementi (2011). "Determination of reaction coordinates via locally scaled diffusion map". In: *J. chem. phys.* 134.12, 124116.

Rossinelli, D., B. Hejazialhosseini, W. van Rees, M. Gazzola, M. Bergdorf, and P. Koumoutsakos (2015). "Mrag-i2d: multi-resolution adapted grids for remeshed vortex methods on multicore architectures". In: *J. comput. phys.* 288, 1.

Rowe, P., V. L. Deringer, P. Gasparotto, G. Csányi, and A. Michaelides (2020). "An accurate and transferable machine learning potential for carbon". In: *The j. chem. phys.* 153.3, 034702.

Rowley, C. W. (2005). "Model reduction for fluids, using balanced proper orthogonal decomposition". In: *Internatl. j. (wash.) of bifurcation and chaos* 15.03, 997.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, 533.

Rupp, M., A. Tkatchenko, K.-R. Müller, and O. A. V. Lilienfeld (2012). "Fast and accurate modeling of molecular atomization energies with machine learning". In: *Phys. rev. lett.* 108, 058301.

Sainath, T. N., O. Vinyals, A. Senior, and H. Sak (2015). "Convolutional, long short-term memory, fully connected deep neural networks". In: *2015 ieee international conference on acoustics, speech and signal process. (icassp)*. Ieee, 4580.

Sangiorgio, M. and F. Dercole (2020). "Robustness of lstm neural networks for multi-step forecasting of chaotic time series". In: *Chaos, solitons & fractals* 139, 110045.

Sano, M. and Y. Sawada (1985). "Measurement of lyapunov spectrum from a chaotic time series". In: *Phys. rev. lett.* 55, 1082.

Sapsis, T. P. and A. J. Majda (2013). "Statistically accurate low-order models for uncertainty quantification in turbulent dynamical systems". In: *Proc. natl. acad. sci. u.s.a.* 110.34, 13705.

Sauer, T., J. A. Yorke, and M. Casdagli (1991). "Embedology". In: *J. stat. phys.* 65.3, 579.

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2008). "The graph neural network model". In: *Ieee transactions on neural networks* 20.1, 61.

Schaeffer, H. (2017). "Learning partial differential equations via data discovery and sparse optimization". In: *Proc. math. phys. eng. sci.* 473.2197, 20160446.

Schäfer, A. M. and H. G. Zimmermann (2006). "Recurrent neural networks are universal approximators". In: *Proceedings of the 16th international conference on artificial neural networks - volume part i*. Icann'06. Berlin, Heidelberg: Springer-Verlag, 632.

Scherer, M. K., B. Trendelkamp-Schroer, F. Paul, G. Pérez-Hernández, M. Hoffmann, N. Plattner, C. Wehmeyer, J.-H. Prinz, and F. Noé (2015). "Pyemma 2: a software package for estimation, validation, and analysis of markov models". In: *J. chem. theory comput.* 11.11, 5525.

Schluse, M., M. Priggemeyer, L. Atorf, and J. Rossmann (2018). "Experimentable digital twins—streamlining simulation-based systems engineering for industry 4.0". In: *Ieee transactions on industrial informatics* 14.4, 1722.

Schmidhuber, J., D. Wierstra, and F. J. Gomez (2005). "Evolino: hybrid neuroevolution/optimal linear search for sequence prediction". In: *Proceedings of the 19th international joint conferenceon artif. intell. (ijcai)*.

Schmidt, F. (2019). "Generalization in generation: a closer look at exposure bias". In: *Arxiv preprint arxiv:1910.00292*.

Schmitt, U. W. and G. A. Voth (1999). "The computer simulation of proton transport in water". In: *J. chem. phys.* 111, 9361.

Schütt, K. T., H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko, and K.-R. Müller (2018). "Schnet–a deep learning architecture for molecules and materials". In: *J. chem. phys.* 148, 241722.

Schütt, K. T., S. Chmiela, O. A. von Lilienfeld, A. Tkatchenko, K. Tsuda, and K.-R. Müller, eds. (2020). *Machine learning meets quantum physics*. Springer, Cham.

Schütte, C., F. Noé, J. Lu, M. Sarich, and E. Vanden-Eijnden (2011). "Markov state models based on milestoning". In: *J. chem. phys.* 134.20, 204105.

Selten, F. M. (1995). "An efficient description of the dynamics of barotropic flow". In: *J. atmos. sci.* 52.7, 915.

Sergeev, A. and M. Del Balso (2018). "Horovod: fast and easy distributed deep learning in tensorflow". In: *Arxiv preprint arxiv:1802.05799*.

Shaw, D., R. Dror, J. Salmon, J. Grossman, K. Mackenzie, J. Bank, C. Young, M. Deneroff, B. Batson, K. Bowers, et al. (2009). "Proceedings of the conference on high performance computing networking, storage and analysis". In: Association for Computing Machinery Portland, Oregon.

Shi, X., Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo (2015). "Convolutional lstm network: a machine learning approach for precipitation nowcasting". In: *Adv. neural inf. process. syst.* 28, 802.

Shi, X., Z. Gao, L. Lausen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo (2017). "Deep learning for precipitation nowcasting: a benchmark and a new model". In: *Adv. neural inf. process. syst.* 5617.

Shi, X. and D.-Y. Yeung (2018). "Machine learning for spatiotemporal sequence forecasting: a survey". In: *Arxiv preprint arxiv:1808.06865*.

Shirts, M. and V. S. Pande (2000). "Screen savers of the world unite!" In: *Science* 290.5498, 1903.

Shlens, J. (2014). "A tutorial on principal component analysis". In: *Arxiv preprint arxiv:1404.1100*.

Sidky, H., W. Chen, and A. L. Ferguson (2020). "Molecular latent space simulators". In: *Chem. sci.* 11, 9459.

Siegelmann, H. and E. Sontag (1995). "On the computational power of neural nets". In: *J. comput. syst. sci.* 50.1, 132.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). "Mastering the game of go with deep neural networks and tree search". In: *Nature* 529.7587, 484.

Sivashinsky, G. I. (1977). "Nonlinear analysis of hydrodynamic instability in laminar flames–i. derivation of basic equations". In: *Acta astronaut.* 4.11, 1177.

Sivashinsky, G. I. and D. Michelson (1980). "On irregular wavy flow of a liquid film down a vertical plane". In: *Progress of theoretical physics* 63.6, 2112.

Skjaerven, L., A. Martinez, and N. Reuter (2011). "Principal component and normal mode analysis of proteins; a quantitative comparison using the groel subunit". In: *Proteins* 79.1, 232.

Springel, V., S. D. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, et al. (2005). "Simulations of the formation, evolution and clustering of galaxies and quasars". In: *Nature* 435.7042, 629.

Srivastava, N., E. Mansimov, and R. Salakhudinov (2015). "Unsupervised learning of video representations using lstms". In: *International conference on machine learning*, 843.

Stamati, H., C. Clementi, and L. E. Kavraki (2010). "Application of nonlinear dimensionality reduction to characterize the conformational landscape of small peptides". In: *Proteins* 78.2, 223.

Stieffenhofer, M., M. Wand, and T. Bereau (2020). "Adversarial reverse mapping of equilibrated condensed-phase molecular structures". In: *Mach. learn.: sci. technol.* 1.4, 045014.

Su, J., W. Byeon, F. Huang, J. Kautz, and A. Anandkumar (2020). "Convolutional tensor-train lstm for spatio-temporal learning". In: *Arxiv preprint arxiv:2002.09131*.

Sultan, M. M., H. K. Wayment-Steele, and V. S. Pande (2018). "Transferable neural networks for enhanced sampling of protein dynamics". In: *J. chem. theory comput.* 14.4, 1887.

Sutskever, I. (2013). *Training recurrent neural networks*. University of Toronto Toronto, Canada.

Taira, K., M. S. Hemati, S. L. Brunton, Y. Sun, K. Duraisamy, S. Bagheri, S. T. Dawson, and C.-A. Yeh (2020). "Modal analysis of fluid flows: applications and outlook". In: *Aiaa journal* 58.3, 998.

Takens, F. (1981). "Detecting strange attractors in turbulence". In: *Dynamical systems and turbulence, warwick 1980*. Ed. by D. Rand and L.-S. Young. Berlin, Heidelberg: Springer Berlin Heidelberg, 366.

Tao, F. and Q. Qi (2019). *Make more digital twins*.

Tao, M., H. Owhadi, and J. E. Marsden (2010). "Nonintrusive and structure preserving multiscale integration of stiff odes, sdes, and hamiltonian systems with hidden slow dynamics via flow averaging". In: *Multiscale modeling & simulation* 8.4, 1269.

Thompson, D. W. and J. M. Wallace (2000). "Annular modes in the extratropical circulation. part i: month-to-month variability". In: *J. climate* 13.5, 1000.

Tikhonov, A. N. and V. Y. Arsenin (1977). *Solutions of ill-posed problems*. W.H. Winston.

Tomczak, J. and M. Welling (2018). "Vae with a vampprior". In: *Aistats*. Pmlr, 1214.

Trendelkamp-Schroer, B. and F. Noé (2016). "Efficient estimation of rare-event kinetics". In: *Phys. rev. x* 6.1, 011009.

Tsai, S.-T., E.-J. Kuo, and P. Tiwary (2020). "Learning molecular dynamics with simple language model built upon long short-term memory neural network". In: *Nat. commun.* 11.1, 1.

Tu, J. H. (2013). "Dynamic mode decomposition: theory and applications". PhD thesis. Princeton University.

Van Erp, T. S., D. Moroni, and P. G. Bolhuis (2003). "A novel path sampling method for the calculation of rate constants". In: *The j. chem. phys.* 118.17, 7762.

Van Rossum, G. and F. L. Drake Jr (1995). *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands.

Vanden-Eijnden, E. and M. Venturoli (2009). "Markovian milestoning with voronoi tessellations". In: *J. chem. phys.* 130.19, 194101.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). "Attention is all you need". In: *Adv. neural inf. process. syst.* 5998.

Vlachas, P. R., G. Arampatzis, C. Uhler, and P. Koumoutsakos (2022). "Multiscale simulations of complex systems by learning their effective dynamics". In: *Nat. mach. intell.* 4.4, 359.

Vlachas, P. R., W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos (2018). "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks". In: *Proc. math. phys. eng. sci.* 474.2213, 20170844.

Vlachas, P. R. and P. Koumoutsakos (in preparation). "Scheduled autoregressive backpropagation through time for robust long-term spatiotemporal forecasting". In: *Tbd*.

Vlachas, P. R., J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos (2020). "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics". In: *Neural networks* 126, 191.

Vlachas, P. R., J. Zavadlav, M. Praprotnik, and P. Koumoutsakos (2021). "Accelerated simulations of molecular systems through learning of effective dynamics". In: *J. chem. theory comput.*

Voelker, A., I. Kajić, and C. Eliasmith (2019). "Legendre memory units: continuous-time representation in recurrent neural networks". In: *Advances in neural information processing systems*, 15570.

Voudouris, C. (1998). "Guided local search—an illustrative example in function optimisation". In: *Bt technol. j.* 16, 46.

Wagner, J. W., J. F. Dama, A. E. Durumeric, and G. A. Voth (2016). "On the representability problem and the physical meaning of coarse-grained models". In: *The j. chem. phys.* 145.4, 044108.

Walker, D. W. and J. J. Dongarra (1996). "Mpi: a standard message passing interface". In: *Supercomputer* 12, 56.

Walker, J., A. Gupta, and M. Hebert (2014). "Patch to the future: unsupervised visual prediction". In: *Proceedings of the ieee conference on computer vision and pattern recognit.* 3302.

Wan, Z. Y. and T. P. Sapsis (2017). "Reduced-space gaussian process regression for data-driven probabilistic forecast of chaotic dynamical systems". In: *Physica d* 345, 40.

Wan, Z. Y. and T. P. Sapsis (2018). "Machine learning the kinematics of spherical particles in fluid flows". In: *J. fluid mech.* 857.

Wan, Z. Y., P. Vlachas, P. Koumoutsakos, and T. Sapsis (2018). "Data-assisted reduced-order modeling of extreme events in complex dynamical systems". In: *Plos one* 13.5, e0197704.

Wang, D. and P. Tiwary (2021). "State predictive information bottleneck". In: *J. chem. phys.* 154.13, 134111.

Wang, H., C. Schütte, G. Ciccotti, and L. D. Site (2014). "Exploring the conformational dynamics of alanine dipeptide in solution subjected to an external electric field: a nonequilibrium molecular dynamics simulation". In: *J. chem. theory comput.* 10, 1376.

Wang, J., S. Olsson, C. Wehmeyer, A. Pérez, N. E. Charron, G. De Fabritiis, F. Noé, and C. Clementi (2019). "Machine learning of coarse-grained molecular dynamics force fields". In: *Acs cent. sci.* 5.5, 755.

Wang, Y., J. M. L. Ribeiro, and P. Tiwary (2019). "Past–future information bottleneck for sampling molecular reaction coordinate simultaneously with thermodynamics and kinetics". In: *Nat. commun.* 10.1, 1.

Wang, Z., A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli (2004). "Image quality assessment: from error visibility to structural similarity". In: *Ieee transactions on image processing* 13.4, 600.

Warrach-Sagi, K., T. Schwitalla, V. Wulfmeyer, and H.-S. Bauer (2013). "Evaluation of a climate simulation in europe based on the wrf–noah model system: precipitation in germany". In: *Climate dyn.* 41.3-4, 755.

Wehmeyer, C. and F. Noé (2018). "Time-lagged autoencoders: deep learning of slow collective variables for molecular kinetics". In: *The j. chem. phys.* 148.24, 241703.

Weinan, E., B. Engquist, et al. (2003). "The heterognous multiscale methods". In: *Commun math sci* 1.1, 87.

Weinan, E., B. Engquist, X. Li, W. Ren, and E. Vanden-Eijnden (2007). "Heterogeneous multiscale methods: a review". In: *Comm. comput. phys.* 2.3, 367.

Weinan, E., X. Li, and E. Vanden-Eijnden (2004). "Some recent progress in multiscale modeling". In: *Multiscale modelling and simulation*. Ed. by S. Attinger and P. Koumoutsakos. Berlin, Heidelberg: Springer Berlin Heidelberg, 3.

Werbos, P. (1974). "Beyond regression:" new tools for prediction and analysis in the behav. sci.s". In: *Ph. d. dissertation, harvard university*.

Werbos, P. J. (1988). "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural networks* 1.4, 339.

Werbos, P. J. (1990). "Backpropagation through time: what it does and how to do it". In: *Proceedings of the ieee* 78.10, 1550.

Werder, T., J. H. Walther, and P. Koumoutsakos (2005). "Hybrid atomistic–continuum method for the simulation of dense fluid flows". In: *J. comput. phys.* 205, 373.

Wiewel, S., M. Becher, and N. Thuerey (2019). "Latent space physics: towards learning the temporal evolution of fluid flow". In: *Comput. graphics forum*. Vol. 38. 2. Wiley Online Library, 71.

Williams, C. K. and C. E. Rasmussen (2006). *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA.

Williams, M. O., I. G. Kevrekidis, and C. W. Rowley (2015). "A data–driven approximation of the koopman operator: extending dynamic mode decomposition". In: *J nonlinear sci* 25.6, 1307.

Winter, R., F. Noé, and D.-A. Clevert (2021). "Auto-encoding molecular conformations". In: *Arxiv preprint arxiv:2101.01618*.

Wolf, A., J. B. Swift, H. Swinney, and J. A. Vastano (1985). "Determining lyapunov exponents from a time series". In: *Physica d* 16, 285.

Wu, H., A. Mardt, L. Pasquali, and F. Noé (2018). "Deep generative markov state models". In: *Neurips*, 3975.

Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. (2016). "Google's neural machine translation system: bridging the gap between human and machine translation". In: *Arxiv preprint arxiv:1609.08144*.

Xavier, P. K. and B. N. Goswami (2007). "An analog method for real-time forecasting of summer monsoon subseasonal variability". In: *Mon. weather rev.* 135.12, 4149.

Yan, X. and X. G. Su (2009). *Linear regression analysis*. World Scientific.

Zavadlav, J., G. Arampatzis, and P. Koumoutsakos (2019). "Bayesian selection for coarse-grained models of liquid water". In: *Sci. rep.* 9, 1.

Zdravkovich, M. (1997). "Flow around circular cylinders; vol. i fundamentals". In: *J. fluid mech.* 350.1, 377.

Zhang, L., J. Han, H. Wang, R. Car, and W. E (2018). "Deepcg: constructing coarse-grained models via deep neural networks". In: *J. chem. phys.* 149.3, 034101.

Zhang, Z., D. Zhang, and R. C. Qiu (2019). "Deep reinforcement learning for power system applications: an overview". In: *Csee journal of power and energy syst.* 6.1, 213.

Zhao, Z. and D. Giannakis (2016). "Analog forecasting with dynamics-adapted kernels". In: *Nonlinearity* 29.9, 2888.

Zheng, W., M. A. Rohrdanz, and C. Clementi (2013). "Rapid exploration of configuration space with diffusion-map-directed molecular dynamics". In: *J. phys. chem. b* 117.42, 12769.

PUBLICATIONS

Vlachas, P. R., W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos (2018). "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks". In: *Proc. math. phys. eng. sci.* 474.2213, 20170844.

Wan, Z. Y., P. Vlachas, P. Koumoutsakos, and T. Sapsis (2018). "Data-assisted reduced-order modeling of extreme events in complex dynamical systems". In: *Plos one* 13.5, e0197704.

Vlachas, P. R., J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos (2020). "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics". In: *Neural networks* 126, 191.

Vlachas, P. R., J. Zavadlav, M. Praprotnik, and P. Koumoutsakos (2021). "Accelerated simulations of molecular systems through learning of effective dynamics". In: *J. chem. theory comput.*

Vlachas, P. R., G. Arampatzis, C. Uhler, and P. Koumoutsakos (2022). "Multiscale simulations of complex systems by learning their effective dynamics". In: *Nat. mach. intell.* 4.4, 359.

Vlachas, P. R. and P. Koumoutsakos (in preparation). "Scheduled autoregressive backpropagation through time for robust long-term spatiotemporal forecasting". In: *Tbd*.

CURRICULUM VITAE

PERSONAL DATA

Name	Pantelis R. Vlachas
Date of Birth	28.10.1993
Place of Birth	Ioannina, Greece
Citizen of	Greece

EDUCATION

October, 2014 – August, 2016	M.Sc. in Electrical Engineering & Information Technology Technical University of Munich Munich, Germany
October, 2011 – July, 2014	B.Sc. in Electrical Engineering & Information Technology Technical University of Munich Munich, Germany

ACADEMIC EXPERIENCE

May, 2021 – ongoing	Associate Researcher in Applied Mathematics Harvard John A. Paulson School of Engineering and Applied Sciences, USA