Reduced Order Modelling and Model-Free Prediction of Chaotic Systems Using Deep Learning

Master of Science Thesis
Aerospace Engineering

Rohan Kaushik

TU Delft

Reduced Order Modelling and Model-Free Prediction of Chaotic Systems Using Deep Learning

Master thesis submitted to Delft University of Technology in partial fulfilment of the requirements for the degree of

Master of Science

by

Rohan Kaushik

Student Number : 5280036

To be defended in public on 23rd August 2023

Supervisor:	Dr. Anh Khoa Doan
Institution:	Delft University of Technology
Faculty:	Faculty of Aerospace Engineering, Delft
Project Duration:	January 2022 - August 2023
Graduation Committee:	Dr. Marios Kotsonis
	Dr. Carmine Varriale

Cover: Time evolution of the Kuramoto-Sivashinsky system, a well-known chaotic model, from a uniform initial condition.

TU Delft

Acknowledgement

Mixed feelings take hold whilst writing these acknowledgements. This thesis project marks the culmination of a somewhat turbulent chapter of my life, one filled with both high highs and rather low lows. However, it does provide a sense of closure and climax to all the effort put into my higher education, and the years spent at Delft have been some of the most enlightening (personally and academically) of my life. I will cherish them for years to come, both the tumult and the repose.

First and foremost, I would like to take this opportunity to thank my parents for their unwavering support, emotional and financial, without whom this journey would not have been possible. My sister Aditi, for her love and un-ending stream of memes and Russo content. My grandmothers, Dadi and Nani, for their faith and words of encouragement, and everyone for their constant support. Ironic that it took living apart to realize just how important and strong familial support is.

Towards my supervisor, Prof. Anh Khoa Doan, I can only ever express gratitude, for his endless patience and assistance. Despite my meandering, he has always been eager to help, and his focus has helped me keep sight of my goals as well. His pragmatic and non-judgemental approach has been instrumental in completing this project.

Finally, I would like to thank the amazing people I am fortunate to call friends. To NH44, Prateek, Noor, Shreya and Divvay, thank you for all the support and love and dinners and distractions. Funny how life can bond together strangers in ways that make the last few years seem unimaginable without. To the og leegh boys, thank you for the open-arms-welcome and for making the covid times bearable. And of course, Sarthak and Himanshu, without whom neither Bombay nor Delft would have been what they were.

Writing this makes me feel both blessed and humbled, realizing the sheer number of people on whose support I've stood, knowingly and unknowingly.

Rohan Kaushik
Delft, August 2023

Abstract

Most physical systems of interest are chaotic in nature, that is, they exhibit strong sensitivity to their initial conditions and evolve “seemingly random" patterns. Quick and reasonably accurate solutions for these systems are essential to various fields such as the design of effective control mechanisms, and early-stage design, amongst others. However, their chaotic nature also leads to them being computationally expensive to model using traditional numerical techniques. Reduced Order Modelling (ROM) is an umbrella term that describes any method wherein a system state is projected into a *simplified* state space in order to extract meaningful information about its dominant components, and/or make the system state’s propagation-in-time easier. Traditional ROMs whilst highly effective in capturing statistical information of a dynamical system, tend to neglect the smaller-energy components responsible for short-term dynamics. With the rapid advances seen in data-driven approaches over the past decade, namely in machine learning (specifically deep learning), these methods seem to offer attractive alternatives to traditional ROM techniques.

Part I of this report sketches out the problem in more detail, and presents the research objective of this project along with the necessary research questions guiding the investigation. Part II details the requisite theoretical background, with Chapter 3 detailing the fundamentals of chaotic systems and the quantification of chaos, and Chapter 4 detailing the fundamentals of modern machine learning. Chapter 5 deals with ROMs, both traditional (Proper Orthogonal Decomposition with Galerkin Projection) and the different existing ML based approaches, in broad strokes. This chapter also summarises the advantages and drawbacks of the discussed approaches and identifies the gaps in existing literature.

It is found that there is no consensus on which ML models to use and no real extensive comparative studies either. Further, there is no unified approach to training the different RNN models, and only one surveyed study utilises an auto-regressive optimization strategy. This is a major pitfall, since the RNNs are only trained for single-step prediction but tested on multi-step prediction - a clear contradiction in the training and testing objectives. Additionally, no studies have been performed on true Runge-Kutta methods-inspired layering architectures, which show initial promise in increasing prediction horizons at little additional compute cost. As such, an extensive comparative study is performed, pitting long-short term memory networks, gated recurrent unit networks and echo state networks against each other. Further, the effect of RK-inspired layering is tested against a standard single-layer network. Moreover, the auto-encoders themselves are augmented to try and produce easier-to-model latent spaces. To this end, the effect of additional contractive loss incorporation is investigated, as is the incorporation of a self-attention layer in the CNN-based models. Lastly, it is noted that all these methods are trained and operated on single-parameter-set regimes, whereas for general-purpose prediction a model with the ability to operate on multiple regimes would be required. For this purpose, the first steps are taken and multi-regime autoencoders also tested.

Part III describes the methodology adopted, and provides an overview of the general workflow of programming and testing. It also details the four chaotic systems selected for investigations, namely the Lorenz ‘63 system, the Charney-DeVore system, the Kuramoto-Sivashinsky system and the Kolmogorov flow system. Note the progressively increasing complexity of the systems, chosen such to be able to identify trends across systems.

For the experiments with the auto-encoders, the results are clear. Contractive loss minimizes the magnitude of the latent space vectors without significantly affecting the reconstruction error or providing the kind of improved latent spaces as were hoped. The incorporation of attention in the convolution auto-encoders - in the Kolmogorov flow system - does not have as significant an impact on the reconstruction errors as do the kernel sizes. This is attributed to the small physical dimensions of the system tested, wherein the receptive fields of the kernels are able to cover the entire input, thus negating the need for the kind of global information that attention is used to provide. Additionally,

the auto-encoders being non-linear processes, are found to be much better at capturing statistical information than straightforward PCA (as expected), and the results for the respective systems are discussed in Chapter 9. Lastly, multi-regime auto-encoders are found to be quite feasible, with vividly separated yet coherently shared latent spaces and minor performance penalties.

Experiments with the RNNs show that additional auto-regressive training campaigns are crucial in improving the prediction horizons of the back-propagation trained RNNs, and should be adopted as standard practice, whereas they have little-to-no effects on the predictive performance of the ESN based models. Further, the RK-inspired layering is found to be only marginally effective at improving prediction performance, and eventually adding more layers only leads to diminishing results. Lastly, it is found that echo-state networks outperform or (at their worst) rival the back-propagation trained RNNs even with these additional tweaks. Add to this their relative ease of training and it becomes clear the ESNs represent a far more accessible and accurate prediction method.

The report ends with a section offering concluding remarks, as well as recommendations for future research. The study performed herein and the additional training procedures tested are by no means exhaustive in nature, but should serve as a clarifying and standardising launch-board for future research.

Contents

Acknowledgement i

Abstract ii

Contents iv

List of Figures viii

List of Tables xii

Nomenclature xiv

I Introduction 1

1 Introduction 2

2 Research Questions 4
2.1 Identified Research Gaps . . . 4
2.2 Research Objective and Questions . . . 4

II Theoretical Background 6

3 Fundamentals of Chaotic Systems 7
3.1 Chaotic Attractors, a Brief Overview . . . 7
3.2 Quantifying Chaos . . . 8
3.2.1 Lyapunov Exponents . . . 9
3.2.2 Fractal Dimensions . . . 9

4 Fundamentals of Machine Learning 11
4.1 Fundamentals of Machine Learning . . . 11
4.1.1 The Artificial Neuron . . . 11
4.1.2 The Feed-Forward Artificial Neural Network . . . 11
4.1.3 The Activation Function . . . 12
4.1.4 Universal Approximation Theorem . . . 12
4.1.5 Stochastic Gradient Descent . . . 13
4.1.6 Regularization . . . 13
4.2 Generative Adversarial Networks and Autoencoders . . . 14
4.3 Networks for Series to Series Transformations . . . 15
4.3.1 Simple RNN . . . 16
4.3.2 Long Short-Term Memory Networks . . . 16
4.3.3 Gated Recurrent Units . . . 17
4.3.4 Echo State Networks . . . 17
4.4 Summary . . . 19

5 Reduced Order Modelling 20
5.1 Classical POD with Galerkin Projection . . . 20
5.1.1 Proper Orthogonal Decomposition . . . 20
5.1.2 Galerkin Projection . . . 21
5.1.3 Limitations . . . 22
5.2 Deep Learning Approaches . . . 22
5.2.1 Single-Model Prediction . . . 22
5.2.2 Multi-Model Prediction . . . 27

5.2.3 Some Other Considerations . . . 29
5.3 Summary . . . 29

III Methodology 30

6 Chaotic Systems Selected for Investigation 31
6.1 The Lorenz '63 System . . . 31
6.2 The Charney-DeVore System . . . 33
6.3 Kuramoto-Sivashinsky System . . . 35
6.4 Kolmogorov Flow . . . 36
6.5 Summary . . . 38

7 Traditional Numerical Solvers 39
7.1 Runge-Kutta Solvers . . . 39
7.2 Exponential Time Differencing with RK4 . . . 40
7.3 Pseudo-Spectral Method . . . 42
7.4 Lyapunov Spectrum Computation . . . 42
7.5 Summary . . . 43

8 General Workflow 44
8.1 Simulation and Data Generation . . . 44
8.1.1 The Lorenz '63 System . . . 44
8.1.2 The Charney-DeVore System . . . 46
8.1.3 The Kuramoto-Sivashinsky System . . . 46
8.1.4 The Kolmogorov Flow System . . . 48
8.2 Auto-encoders . . . 50
8.2.1 Layering and Network Architecture . . . 50
8.2.2 Error Metrics . . . 52
8.2.3 Loss Function . . . 52
8.2.4 Training . . . 53
8.2.5 Hyper-Parameter Selection . . . 54
8.2.6 Latent Space Principal Directions . . . 55
8.3 Recurrent Neural Networks . . . 55
8.3.1 Error Metrics . . . 55
8.3.2 Training . . . 56
8.3.3 Layering and Layer Sizes . . . 56
8.3.4 Hyper-Parameter Selection . . . 57
8.4 Combined AE-RNN . . . 59
8.4.1 Training . . . 59
8.4.2 Long-term Prediction Statistics . . . 60
8.5 Software and Libraries . . . 61

IV Results and Discussion 62

9 Dimensionality Reduction Using Auto-encoders 63
9.1 The Lorenz '63 System . . . 63
9.1.1 Contractive Loss . . . 63
9.1.2 AE and POD Comparison . . . 64
9.1.3 Multi-regime AE . . . 66
9.2 The Charney-DeVore System . . . 67
9.2.1 Contractive Loss . . . 67
9.2.2 AE and POD . . . 68
9.2.3 Multi-regime AE . . . 69
9.3 The Kuramoto-Sivashinsky System . . . 71
9.3.1 Contractive Loss . . . 71
9.3.2 AE and POD . . . 72

9.3.3 Multi-regime AE . . . 74
9.4 Kolmogorov Flow . . . 76
9.4.1 Contractive Loss . . . 76
9.4.2 AE and POD . . . 76
9.4.3 Multi-regime AE . . . 78
9.5 Summary . . . 80

10 Time Series Predictions using RNNs **82**
10.1 The Lorenz '63 System . . . 82
10.1.1 Comparison of teacher-forced and AR-trained networks . . . 83
10.1.2 Increasing layer sizes . . . 84
10.1.3 Comparison of single-layer networks . . . 84
10.1.4 Effect of RK-inspired layering in the GRUs . . . 86
10.2 The Charney-DeVore System . . . 87
10.2.1 Comparison of teacher-forced and AR-trained networks . . . 87
10.2.2 Increasing layer sizes . . . 88
10.2.3 Comparison of single-layer networks . . . 89
10.2.4 Effect of RK-inspired layering in the GRUs . . . 90
10.3 The Kuramoto-Sivashinsky System . . . 92
10.3.1 Comparison of teacher-forced and AR-trained networks . . . 92
10.3.2 Increasing layer sizes . . . 93
10.3.3 Comparison of single-layer networks . . . 94
10.3.4 Effect of RK-inspired layering in the GRUs . . . 96
10.4 Kolmogorov Flow . . . 97
10.4.1 Comparison of teacher-forced and AR-trained networks . . . 97
10.4.2 Increasing layer sizes . . . 98
10.4.3 Comparison of single-layer networks . . . 99
10.4.4 Effect of RK-inspired layering in the GRUs . . . 100
10.5 Summary . . . 101

V Conclusions and Recommendations **102**

11 Conclusion **103**

12 Recommendations **105**

References **107**

VI Appendices **111**

A General Workflow **112**
A.1 Simulation and Data Generation . . . 112
A.1.1 The Lorenz '63 System . . . 112
A.1.2 The Charney-DeVore System . . . 112
A.1.3 The Kuramoto-Sivashinsky System . . . 113
A.1.4 The Kolmogorov Flow System . . . 114
A.2 Auto-encoders . . . 115
A.2.1 Layering and Network Architecture . . . 115

B Dimensionality Reduction Using Auto-encoders **118**
B.1 The Lorenz '63 System . . . 118
B.1.1 Bayesian Optimization . . . 118
B.2 The Charney-DeVore System . . . 119
B.2.1 Bayesian Optimization . . . 119
B.3 The Kuramoto-Sivashinsky System . . . 119
B.3.1 Bayesian Optimization . . . 119
B.4 The Kolmogorov Flow System . . . 120

B.4.1 Bayesian Optimization . . . 120
B.4.2 AE and POD . . . 120

C Time Series Predictions using RNNs **121**
C.1 The Lorenz '63 System . . . 121
C.1.1 Bayesian Optimization . . . 121
C.1.2 Auto-regressively Trained BPTT RNNs . . . 122
C.1.3 Long-term Evolution of Sample Trajectories . . . 123
C.2 The Charney-DeVore System . . . 124
C.2.1 Bayesian Optimization . . . 124
C.2.2 Auto-regressively Trained BPTT RNNs . . . 124
C.2.3 Long-term Evolution of Sample Trajectories . . . 125
C.3 The Kuramoto-Sivashinsky System . . . 127
C.3.1 Bayesian Optimization . . . 127
C.3.2 Auto-regressively Trained BPTT RNNs . . . 127
C.3.3 Long-term Evolution of Sample Trajectories . . . 128
C.4 The Kolmogorov Flow System . . . 129
C.4.1 Bayesian Optimization . . . 129
C.4.2 Auto-regressively Trained BPTT RNNs . . . 129
C.4.3 Long-term Evolution of Sample Trajectories . . . 130
C.4.4 Typical Evolution of u, v and ω for a Sample Trajectory - AE-GRU Model . . . 131
C.4.5 Typical Evolution of u, v and ω for a Sample Trajectory - AE-LSTM Model . . . 134
C.4.6 Typical Evolution of u, v and ω for a Sample Trajectory - AE-ESN Model . . . 137
C.4.7 Typical Evolution of u, v and ω for a Sample Trajectory - AE-SimpleRNN Model 140
C.4.8 Typical Evolution of u, v and ω for a Sample Trajectory - POD-Galerkin Method 143
C.5 Fraction of Non-trainable parameters in an ESN . . . 146

List of Figures

3.1 The well-known Lorenz attractor. . . . 8

4.1 Flowchart of an artificial neuron. . . . 11
4.2 A feed-forward ANN. . . . 12
4.3 Typical loss curves obtained during training [32]. . . . 14
4.4 An autoencoder setup that is tasked with finding a compressed representation of the original data [6]. . . . 15
4.5 Typical graph of an RNN [32]. . . . 16
4.6 Unrolled graph of an RNN [32]. . . . 16
4.7 An LSTM cell and its inner connections [75]. . . . 17
4.8 A GRU cell and its inner connections [3]. . . . 17

5.1 2000 time-unit trajectories of the CDV system, projected to the first two POD modes [73]. 22
5.2 Time evolution of the normalized RMSE for three different prediction methods - ESN (red), LSTM (cyan) and feed-forward ANN (blue). [13] 23
5.3 Time evolution of the normalized RMSE of the Lorenz 96 system, for different prediction methods. [73] 25
5.4 Parallelized scheme of moderately sized reservoirs operating on their own local regions. [57] 26
5.5 Absolute error in the velocity fields between the true and predicted data at the 199^{th} (left column) and 399^{th} (right column) time-steps, having been fed only 10 time-steps as the input history. [78] 28

6.1 Typical Lorenz cell. . . . 32
6.2 Time evolution of the solution (integrated using the RK4 method with $\Delta t = 0.01$) in the phase space. . . . 32
6.3 Time evolution of the individual components (integrated using the RK4 method with $\Delta t = 0.01$). . . . 32
6.4 A plane tangent to a sphere [69]. . . . 34
6.5 150,000 points sampled from the CDV attractor, projected to the x_1-x_4 plane. The different flow regimes occupy distinct spaces. . . . 34
6.6 Time evolution of all the state variables (integrated using the RK4 method with $\Delta t = 0.1$), with the blocked flow regime clearly highlighted. . . . 34
6.7 Time evolution of KS equation, with $L = 35$ (integrated using the ETDRK4 method with 100 gridpoints and $\Delta t = 0.1$). . . . 36
6.8 Iso-contours of u_x, $Re = 30$ [19]. . . . 36
6.9 Iso-contours of u_y, $Re = 30$ [19]. . . . 36
6.10 Iso-contours of the vorticity, $Re = 30$ [19]. . . . 36
6.11 Time evolution of the dissipation and the absolute value of the mode $(1, 0)$, $Re = 40$ [75]. 38

8.1 Phase space of the Lorenz system with different parameter sets. . . . 45
8.2 CDV solution trajectories, plotted in the $x_1 - x_4$ plane. . . . 46
8.3 Time evolution of the individual components in a CDV system, for different parameter sets. 47
8.4 Time evolution of the KS system with different parameter sets. . . . 48
8.5 Iso-contours of u_x, u_y and vorticity ω ($Re = 30$) - snapshots a, b, c and time means d, e, f. 49
8.6 Iso-contours of u_x, u_y and vorticity ω ($Re = 40$) - snapshots a, b, c and time means d, e, f. 49
8.7 Energy spectrum of the Kolmogorov flow system. . . . 50
8.8 Typical flow-charts of the auto-encoder configurations. . . . 50
8.9 Examples of the convolution and transposed convolution operations. . . . 51

8.10 Multi-scale Convolutional Auto-encoder (w_i^e and w_i^d are the weights for the outputs of the parallel encoders and decoders, respectively). . . . 52
8.11 Typical auto-encoder training curves. . . . 53
8.12 Typical RNN teacher-forced training curves. . . . 56
8.13 Teacher-forced RNN training flow-chart, where $\mathbf{i}^{t_n}$ and $\mathbf{o}^{t_n}$ represent the input and output (respectively) vectors, $\mathbf{s}^{t_n}$ the RNN state being received by the cell, all at time instance t_n. The mauve cells represent the RNN cells and their internal structure, while the red squiggly arrows mark the direction of gradient-flow. . . . 56
8.14 n^r values computed for the various reported studies. A is [13], B is [75], C is [73], D is [57], E is [74], F is [19], G is [55]. . . . 57
8.15 Flow-charts depicting the RK-inspired layering architectures. Note that the weights are shared amongst the RK-cells. . . . 58
8.16 Auto-regressive training flow-chart of the combined AE-RNN. $\mathcal{E}(\cdot)$ and $\mathcal{D}(\cdot)$ represent the application of the encoder and decoder networks (respectively), and the remaining symbols are defined as in Figure 8.13 . . . 60

9.1 Various computed metrics for the auto-encoders with and without contractive loss. . . . 64
9.2 Normalized and original values of the mean Jacobian norm, for the auto-encoders with and without contractive loss. . . . 64
9.3 Reconstruction error (NRMSE). . . . 65
9.4 Total variance captured by the reconstructed data, w.r.t. the original data. . . . 65
9.5 Decoded latent space principal directions, and POD principal directions . . . 66
9.6 Different metrics representing the comparisons between the auto-encoder and the POD method. . . . 66
9.7 Multi-regime auto-encoders. . . . 67
9.8 Various computed metrics for the auto-encoders with and without contractive loss. . . . 68
9.9 Normalized and original values of the mean Jacobian norm, for the auto-encoders with and without contractive loss. . . . 68
9.10 Reconstruction error (NRMSE). . . . 69
9.11 Total variance captured by the reconstructed data, w.r.t. the original data. . . . 69
9.12 Multi-regime auto-encoder. . . . 70
9.13 Decoded latent space principal directions, and POD principal directions . . . 70
9.14 Different metrics representing the comparisons between the auto-encoder and the POD method. . . . 71
9.15 Various computed metrics for the auto-encoders with and without contractive loss. . . . 72
9.16 Normalized and original values of the mean Jacobian norm, for the auto-encoders with and without contractive loss. . . . 72
9.17 Reconstruction error (NRMSE). . . . 73
9.18 Amount of total variance captured by the reconstructed data, w.r.t. the original data. . . . 73
9.19 Multi-regime auto-encoder. . . . 74
9.20 Decoded latent space principal directions, and POD principal directions . . . 75
9.21 Different metrics representing the comparisons between the auto-encoder and the POD method. . . . 76
9.22 Performance metrics for the reconstructed data for a fixed latent space size and varying kernel filter size (Kolmogorov flow system, $Re = 40$). . . . 77
9.23 Performance metrics for the reconstructed data for a fixed kernel size (multi-scale) and varying latent space size (Kolmogorov flow system, $Re = 40$). . . . 78
9.24 Multi-regime auto-encoder. . . . 79
9.25 Different metrics representing the comparisons between the auto-encoder and the POD method. . . . 79
9.26 Principal directions . . . 80

10.1 Prediction horizons for the teacher-forced and AR-trained networks (combined AE-RNN, AR Training Steps' refer to the number of output time-steps the network is auto-regressively trained on). . . . 83

10.2 Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN). . . . 84
10.3 Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 85
10.4 W_1 Wasserstein distances computed for the different x_j PDFs, with predictions taken over a time-interval of 50 t_L. . . . 86
10.5 Long-term evolution of sample trajectories for the Lorenz '63 system. . . . 86
10.6 Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 87
10.7 Prediction horizons for the teacher-forced and AR-trained networks (combined AE-RNN). 88
10.8 Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN). . . . 89
10.9 Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 89
10.10 W_1 Wasserstein distances computed for the different x_j PDFs, with predictions taken over a time-interval of 50 t_L. . . . 90
10.11 Long-term evolution of sample trajectories for the Charney-DeVore system. . . . 91
10.12 Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 92
10.13 Prediction horizons for the teacher-forced and AR-trained networks (combined AE-RNN). 93
10.14 Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN). . . . 94
10.15 Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 95
10.16 W_1 Wasserstein distances computed for the TKE and D PDFs, with predictions taken over a time-interval of 50 t_L. . . . 95
10.17 Long-term evolution of sample trajectories for the Kuramoto-Sivashinsky system. . . . 96
10.18 Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 97
10.19 Prediction horizons for the teacher-forced and AR-trained single layer GRU based models (combined AE-RNN, $n^r = 20$). . . . 98
10.20 Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN). . . . 98
10.21 Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 99
10.22 W_1 Wasserstein distances computed for the different x_j PDFs, with predictions taken over a time-interval of 50 t_L. . . . 100
10.23 Long-term evolution of sample trajectories for the Kolmogorov flow system. . . . 100
10.24 Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 101

A.1 $ACF(\tau)$ for the CDV system with $(x_1^*, x_4^*) = (0.99, -0.79299)$ . . . 113
A.2 Time evolution of the mean kinetic energy (KE) of the KS system with different parameter sets. . . . 114
A.3 Time evolution of the mean turbulent kinetic energy (TKE) of the KS system with different parameter sets. . . . 114
A.4 Time evolution of the mean dissipation rate (D) of the KS system with different parameter sets. . . . 114
A.5 $ACF(\tau)$ for the KS system with $(\nu_1, \nu_2, \nu_3) = (1, 2, 1)$a, $(1, 1, 2)$b . . . 114
A.6 Snippets of the time-evolution of the KE, TKE and D, for the Kolmogorov flow system ($Re = 30$). . . . 115
A.7 Snippets of the time-evolution of the KE, TKE and D, for the Kolmogorov flow system ($Re = 40$). . . . 115

B.1 Iteration-wise evolution of the MSE for the Bayesian optimization. . . . 118
B.2 Iteration-wise evolution of the MSE for the Bayesian optimization. . . . 119

B.3 Iteration-wise evolution of the MSE for the Bayesian optimization. 119
B.4 Search for f_{noise} and λ_{reg}, and the iteration-wise evolution of the MSE. 120

C.1 Iteration-wise evolution of the MSE for the Bayesian optimization. 121
C.2 Evolution of the prediction horizon distribution as auto-regressive training progresses. 122
C.3 Long-term evolution of sample trajectories for the Lorenz '63 system. 123
C.4 Iteration-wise evolution of the MSE for the Bayesian optimization. 124
C.5 Evolution of the prediction horizon distribution as auto-regressive training progresses. 124
C.6 Sample trajectory evolution for the AE-LSTM model. 125
C.7 Sample trajectory evolution for the AE-ESN model. 125
C.8 Sample trajectory evolution for the AE-SimpleRNN model. 126
C.9 Iteration-wise evolution of the MSE for the Bayesian optimization. 127
C.10 Evolution of the prediction horizon distribution as auto-regressive training progresses. 127
C.11 Long-term evolution of sample trajectories for the Kuramoto-Sivashinsky system. 128
C.12 Iteration-wise evolution of the MSE for the Bayesian optimization. 129
C.13 Evolution of the prediction horizon distribution as auto-regressive training progresses. 129
C.14 Long-term evolution of sample trajectories for the Kolmogorov flow system. 130
C.15 Evolution of u at different time-steps of a sample trajectory, for the AE-GRU model. . . . 131
C.16 Evolution of v at different time-steps of a sample trajectory, for the AE-GRU model. . . . 132
C.17 Evolution of ω at different time-steps of a sample trajectory, for the AE-GRU model. . . 133
C.18 Evolution of u at different time-steps of a sample trajectory, for the AE-LSTM model. . . 134
C.19 Evolution of v at different time-steps of a sample trajectory, for the AE-LSTM model. . . 135
C.20 Evolution of ω at different time-steps of a sample trajectory, for the AE-LSTM model. . . 136
C.21 Evolution of u at different time-steps of a sample trajectory, for the AE-ESN model. . . . 137
C.22 Evolution of v at different time-steps of a sample trajectory, for the AE-ESN model. . . . 138
C.23 Evolution of ω at different time-steps of a sample trajectory, for the AE-ESN model. . . . 139
C.24 Evolution of u at different time-steps of a sample trajectory, for the AE-SimpleRNN model. 140
C.25 Evolution of v at different time-steps of a sample trajectory, for the AE-SimpleRNN model. 141
C.26 Evolution of ω at different time-steps of a sample trajectory, for the AE-SimpleRNN model. 142
C.27 Evolution of u at different time-steps of a sample trajectory, for the POD-Galerkin method. 143
C.28 Evolution of v at different time-steps of a sample trajectory, for the POD-Galerkin method. 144
C.29 Evolution of ω at different time-steps of a sample trajectory, for the POD-Galerkin method. 145

List of Tables

7.1 A typical Butcher tableau for an m-stage explicit RK method. . . . 40
7.2 Butcher tableau for RK4 method. . . . 40

8.1 Hyper-parameters and their selection methods. . . . 54
8.2 Hyper-parameters, their imposed bounds and assumed priors. . . . 54
8.3 Chosen values of the multiplicative factor n^r. . . . 57
8.4 Hyper-parameters and their selection methods. . . . 58
8.5 ESN hyper-parameters, their imposed bounds and assumed priors. . . . 58
8.6 Hyper-parameters and their selection methods. . . . 59
8.7 Hyper-parameters, their imposed bounds and assumed priors. . . . 59
8.8 Quantities used to track attractor trajectories for different systems. . . . 61

9.1 Reconstruction error (NRMSE). . . . 65
9.2 Total variance captured by the reconstructed data, w.r.t. the original data. . . . 65
9.3 Reconstruction error (NRMSE). . . . 69
9.4 Total variance captured by the reconstructed data, w.r.t. the original data. . . . 69
9.5 Reconstruction error (NRMSE). . . . 73
9.6 Amount of total variance captured by the reconstructed data, w.r.t. the original data. . . . 73

10.1 Optimized ESN hyper-parameters. . . . 82
10.2 VPT [LT] at different stages, for the ESN ($n^r = 200$) and GRU ($n^r = 20$) based AE-RNN. 84
10.3 VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN. . . . 84
10.4 VPT for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 85
10.5 VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 87
10.6 Optimized ESN hyper-parameters. . . . 87
10.7 VPT [LT] at different stages, for the ESN ($n^r = 200$) and GRU ($n^r = 20$) based AE-RNN. 88
10.8 VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN. . . . 88
10.9 VPT for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 89
10.10 VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 92
10.11 Optimized ESN hyper-parameters. . . . 92
10.12 VPT [LT] at different stages, for the ESN ($n^r = 200$) and GRU ($n^r = 20$) based AE-RNN. 93
10.13 VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN. . . . 93
10.14 VPT for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 95
10.15 VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 97
10.16 Optimized ESN hyper-parameters. . . . 97
10.17 VPT [LT] at different stages, for the GRU ($n^r = 20$) based AE-RNN. . . . 98
10.18 VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN. . . . 99
10.19 VPT for the best performing RNN networks (highest layer size, combined AE-RNN). . . . 99
10.20 VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN). . . . 101

A.1 MLE, KY dimension and Lyapunov times for different parameter sets of the Lorenz system 112
A.2 MLE, KY dimension and Lyapunov times for different parameter sets of the CDV system 112
A.3 MLE, KY dimension and Lyapunov times for different parameter sets of the KS system . 113
A.4 MLE, KY dimension and Lyapunov times for different Reynolds numbers (Kolmogorov flow) . . . 115

A.5 Auto-encoder layer sizes for the Lorenz '63 system. 116
A.6 Auto-encoder layer sizes for the CDV system. 116
A.7 Auto-encoder layer sizes for the KS system. 116
A.8 Auto-encoder layer sizes for the Kolmogorov flow system, with the kernel filter size set to 7×7. Other filter sizes follow the same layer structures, with different values for the periodic padding and cropping layers. 117

B.1 Reconstruction error (NRMSE) for the auto-encoders with changing kernel filter sizes. . 120
B.2 Captured total variance for the auto-encoders with changing kernel filter sizes. 120
B.3 Reconstruction error (NRMSE) for the multi-scale auto-encoder with changing latent space dimensionality. 120
B.4 Captured total variance for the multi-scale auto-encoder with changing latent space dimensionality. 120

Nomenclature

Abbreviations

ROM	Reduced Order Model
AE	Auto-encoder
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory (network)
ESN	Echo State Network
GRU	Gated Recurrent Unit (network)
BPTT	Back-Propagation Through Time
ANN	Artificial Neural Network
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
POD	Proper Orthogonal Decomposition
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
MSE	Mean Squared Error
NMSE	Normalized Mean Squared Error
NRMSE	Normalized Root Mean Squared Error
NRSE	Normalized Root Squared Error
VPT	Valid Prediction Time
MLE	Maximal Lyapunov Exponent
LT	Lyapunov Time
RK	Runge-Kutta
CDV	Charney-DeVore
KS	Kuramoto-Sivashinsky
KY	Kaplan-Yorke
ML	Machine Learning
AR	Auto-regressive
TKE	Turbulent Kinetic Energy
KE	Kinetic Energy

Greek Symbols

η	ratio of the total variance of reconstructed data and actual data
Λ_j	j^{th} Lyapunov exponent
$\lambda_{\text{contractive}}$	Contractive penalty hyper-parameter
λ_{reg}	L_2 regularization hyper-parameter
σ_i	standard deviation of the i^{th} component
λ_i	The i^{th} eigenvalue
ν_i	i^{th} parameter of the KS system
ρ	Lorenz '63 system parameter
σ	Lorenz '63 system parameter
β	Lorenz '63 system parameter
ρ_{res}	Spectral radius of the reservoir of an ESN
ω_{in}	Input scaling factor of an ESN
α	Leaking coefficient of an ESN
$\boldsymbol{\theta}$	The trainable parameters of an ANN

Latin Symbols

D_{KY}	Kaplan-Yorke dimension
D_H	Hausdorff dimension
t_L	Lyapunov time
Re	Reynolds number
$\mathbf{a}$	A vector of concatenated states
$\boldsymbol{a}$	A vector quantity such as velocity
$\underline{\boldsymbol{A}}$	A 2D matrix
u	x-direction velocity component
v	y-direction velocity component
ω	z-direction component of vorticity
$\mathcal{E}$	The encoder network
$\mathcal{D}$	The decoder network
D	Mean Dissipation Rate
$\mathcal{F}$	Fourier transform
n^r	RNN layer multiplicative factor
n^{ls}	Dimensionality of the latent space
$\mathbf{z}$	Vector of concatenated latent space variables
f_{noise}	Noise controlling hyper-parameter

PART I
Introduction

1

Introduction

All natural dynamical processes that are of research (and practical) interest - stemming from fields as differing as climate systems, ocean circulation, fluid dynamics, nonlinear electrodynamics and optics, and even mechanical systems - are generally governed by partial differential equations, and possess dynamics occurring at multiple spatio-temporal scales [76, 31, 1, 74, 78]. The study of fluid dynamics is no exception, and every scale regime is important - with the smallest affecting the largest and vice-versa[1][70]. Adding to this strong multi-scale linkage, is the fact that these systems often possess high intrinsic dimensionality[2]. The combination of these factors leads to such systems repeatedly exhibiting *chaotic* behaviour, leading to their study through classical techniques being only so effective.

Traditional methods to study such systems have involved two approaches. The first, to identify the governing equations, and then study the structure and properties of these identified systems through experimentation and mathematical modelling. This kind of work was the only way to study chaotic systems before the introduction and wide-scale adoption of computers. Examples of these kinds of investigations include the works of Rayleigh, Prandtl and Kolmogorov (for turbulent flows) [70]. The second approach involves using the identified governing equations (and their discovered properties) to create appropriate discrete models, and then using computers to solve for approximate solutions of these discrete models [2]. This has become the standard approach to modelling chaotic systems over the past seven decades. The ability to compute chaotic solutions has ushered in an era of technological advances, from more efficient airplanes and wind turbines to better designed heart valves and construction materials. However, this method is not without its pitfalls. The linkage of multi-scale dynamics leads to a requirement of extremely fine spatial and temporal discretization for a stable solution, which can be prohibitively expensive. As such, cheaper to compute sub-optimal models of these equations are employed, all of which use approximations to avoid a full-scale simulation. Examples include Reynolds Averaged Navier-Stokes (RANS) models, Large Eddy Simulations (LES) and the Euler equations for compressible flows, among others. Further, their also exists a variety of discretization methods to choose from (such as Finite Volume Methods, Finite Element Methods, mesh-free Lagrangian methods, et cetera).

While high-fidelity approaches (such as Direct Numerical Simulations) can yield extremely accurate solutions, their aforementioned computational cost confines them to the research domain. In real-world applications, simpler yet faster models which can produce reasonable results within realistic time-frames, are more frequently utilized (such as RANS and LES). Even still, these less-accurate models can be quite computationally expensive, as is the case for large scale LES. To tackle this, model-reduction techniques have been proposed, which operate on the assumption that the full-scale system dynamics are a higher-dimensional representation of the actual core dynamics, taking place in a lower-dimensional latent space. Thus, these methods aim to construct an optimal reduced space, model the dynamics in this space and then reconstruct the full-scale system from the propagated reduced space variables. As can be inferred, errors are introduced at every step of the aforementioned approach, and these classical reduced order models have found limited (but not non-existent) success in modelling chaotic behaviour

[1] As is typified in the energy cascade and back-scatter observed in well-developed turbulent flows.
[2] In traditional discretized systems, the discretized system also ends up possessing an extremely high dimensional state vector.

[38, 71].

The need for fast and accurate models is even more pronounced in domains where each simulation is only a single step in a larger multi-query process, such as preliminary design, fluid-dynamic and/or structural-stress based shape optimization, model-predictive control and non-intrusive system monitoring [48, 79, 61, 81]. In such environments, even the aforementioned sub-optimal models can be too costly, and reduced order models (with all their inaccuracies) are generally settled for. In this context of fast yet not-very-accurate solutions, switching to the domain of imprecise computation seems only logical. Specifically, the domain of Machine Learning (ML) has seen rapid growth in the past decade, both in terms of the mathematical analysis, and the specialized hardware/software frameworks involved.

Indeed, there have been a spate of promising studies in recent years which have shown the potential of ML based models to serve as surrogates for various chaotic systems, at a fraction of the computation cost. Even then, naïvely applying traditional ML models to full-order dynamics can be costly, and other ML based models have been used to reduce the system's dimensionality. However, each such study utilises different ML models and/or different training methods, and there seem to be few-to-none comparative studies to judge them side-by-side. Also, due to the very nature of this rapidly evolving field, there exist models and techniques that have not yet been explored within the context of chaotic systems prediction - namely Runge-Kutta methods inspired skip connections. Lastly, the great advantage of machine learning based systems is that they can learn spatio-temporal co-dependencies directly from raw data and do not require pre-defined governing equations (as opposed to traditional numerical solvers). This can be especially useful in cases where it is too tedious to construct a governing equation, or those where only directly observed data is available.

The aim of this thesis project is twofold. First to perform effective dimensionality reduction by means of an auto-encoder, and try to augment them so as to produce easier-to-model latent spaces themselves. To this end, in addition to a standard auto-encoder, a contractive auto-encoder is tested, along with a novel idea for a multi-regime auto-encoder that has significant consequences for future research. Second, to investigate the different aspects of time-series modelling using RNNs. These include bridging the gap between RNN training and testing objectives with the exploration of the effect of auto-regressive training, as well as performing a comparative study between four popular time series modelling ML models - the long-short term memory networks, the gated recurrent unit network, the vanilla (simple) recurrent neural network and the echo state network. Also investigated is the effect of RK-inspired skip-connected layer architectures when compared to a standard single-layer architecture.

With these goals, this report is organised into five parts. Part I serves as an introduction, and Chapter 2 details the research objective and scope of this project. Part II details the requisite theoretical background, namely the fundamentals of chaos, machine learning and the traditional numerical techniques used for data generation, with Chapter 5 providing an overview of the literature on present state-of-the-art ML-based ROMs. Part III presents the experimental methodology followed and describes the general workflow of the project, along with the tested chaotic systems. Part IV presents and discusses the results of this project, with Chapter 9 focusing on discussing the developed auto-encoders and Chapter 10 the results of the time-series modelling of the combined AE-RNN models. Part V serves as the culmination of this project report, by offering concluding remarks and detailing the recommendations for future research directions.

2 Research Questions

This chapter identifies and presents the key research questions based on the goals of this project. It further lists out the sub-questions contained within each, and their motivations touched upon.

2.1. Identified Research Gaps

As can be gleaned from the various studies presented in Chapter 5, there is no consensus on which ML models to use. Each study employs and tests different ML models. Further, there is no unified approach to training the different models, and only one surveyed study utilises an auto-regressive optimization strategy (for the RNNs). This appears to be a major pitfall in the rest of the studies, since the RNNs are only trained for a single-step prediction but tested on multi-step prediction. This presents a clear contradiction in the training and testing objectives.

There is also no true test of RK residual methods. Even the study that tests RK methods, uses a single ANN to model the time-series as opposed to any RNNs. Further, the network is constructed such that the ANN essentially learns to predict only the time derivative. This seems less powerful than incorporating a RK-inspired skip architecture into the residual model itself. Lastly, no studies investigate the possibility of using these ML based ROMs to model multiple dynamical regimes of a system. This would be crucial if these methods are to be used as effective surrogates in the kinds of applications mentioned in Chapter 1.

To this end, a comparative study of a broad slice of machine learning based model-free prediction techniques for chaotic systems is performed. Since chaos is a property also shared by many systems simpler than fluid dynamical systems, it would be helpful to start by applying these methods to such systems. This would help build testing/training pipelines, and intuition, before moving on to a fluid dynamics case. Additionally, effective dimensionality reduction by the auto-encoders is also investigated, and a contractive-loss penalty is tested to try and produce 'smoother' latent spaces. Further, the first step in the multi-regime modelling paradigm is taken with the tested multi-regime auto-encoders. These goals lend themselves to the following research questions.

2.2. Research Objective and Questions

Based on the identified knowledge gap, the overarching research objective of this thesis is:

> to identify and quantify how accurately are different machine learning based ROMs able to forecast the evolution of chaotic systems, particularly a fluid dynamical system, and the best strategies for their training.

In this thesis, chaotic systems of different complexity are considered, starting from low-order systems governed by a set of ODEs moving on to those governed by PDEs. This progressive increase in complexity is to help build models and intuition along the way, before finally dealing with a chosen representative fluid dynamical system. To help achieve this objective, the main research questions are outlined below:

1. **Can the AE be augmented to provide better constructed latent spaces?**
   (a) *Specifically, does the incorporation of a contractive loss improve accuracy?* This penalizes the Jacobian of the latent variable w.r.t. the input variable, which penalizes changes in the latent variable w.r.t. small changes in the input variable. This can help produce a smoother latent variable distribution, thereby making its time-series modelling easier.
   (b) *For the CNN-based AE, how much does the incorporation of self-attention affect accuracy?* CNNs excel at capturing and preserving local interactions, but the same architecture leads them to neglect global information. Attention can help overcome this, by capturing global dependencies that the CNN neglects, and should ideally produce more informed lower-level representations.
   (c) *Can the model parameters be incorporated into the AE architecture in a meaningful way such that it produces a multi-regime AE?* Up till now the analysis focused on systems characterized by a single value of its dependent parameters. Investigating this sub-question can help extend the prediction power of the ML-ROM to span multiple regimes, thus making a more general-purpose ROM.

2. **Which series-to-series ML models are better suited to the combined AE-RNN ROM paradigm, according to a suitably defined prediction horizon (as the accuracy metric)?** This will entail the assessment of various architectures, specifically simple RNNs, LSTMs, GRUs and ESNs will be considered. This will further be investigated through the following sub-questions:
   (a) *How is the BPTT-trained RNN[1] based model's accuracy affected when they are trained auto-regressively vs. when they are trained using teacher-forcing?* Teacher-forcing essentially trains them for one-step-ahead prediction, whereas the auto-regressive training optimizes them for multi-step-ahead prediction. This tackles the misalignment of the training and testing objectives mentioned earlier.
   (b) *How does an additional BPTT-based auto-regressive training campaign affect an already (non-BPTT) trained ESN based model's accuracy?* ESNs are 'trained' using linear regression, which involves a single matrix inversion. However, this too is teacher-forcing the network, since the output to match for the regression problem is the next time-step's output. Thus the networks are again trained for one-step-ahead prediction, but tested on multi-step-ahead predictions. The output matrix obtained from the linear regression can be used as its initial value, whereas the auto-regressive training can be used as a way of fine-tuning it for the multi-step-ahead prediction task.
   (c) *How do these ML-based ROMs compare against a traditional POD-Galerkin approach?* Are there improvements in
      i. prediction horizons?
      ii. long-term statistical behaviour of predicted data?

Answering the above questions solidifies the training procedures for the different RNNs. Then the layer-connection architecture can be explored satisfactorily by considering the following question:

3. **How does layering affect prediction accuracy in BPTT-trained RNNs, specifically GRUs?**
   (a) *How is accuracy affected when using a single layer and multiple layers with RK-inspired skip connections?*

Answering the above research questions leads to achieving the stated research objective, and successfully concludes this thesis project.

[1] these being GRUs, LSTMs and simple RNNs

PART II
Theoretical Background

3

Fundamentals of Chaotic Systems

Dynamical systems are systems that evolve over time. They can be found in every modern field of study, from history to physics. By creating an appropriate mathematical model, these systems can be studied in great detail. As mentioned in Chapter 1, all such systems of interest are found to be characteristically chaotic, necessitating an additional field of study to deal with them - namely *chaos theory*. This section is by no means a treatise on the subject, but rather aims to familiarize the reader with some key concepts and chaos-quantification measures that are useful for the present project.

3.1. Chaotic Attractors, a Brief Overview

In mathematical modelling, an attractor refers to a set of states or values which a dynamical system tends to orbit over time. It represents the system's long-term behaviour and provides insights into its stable patterns or recurrent behaviour. Attractors are a fundamental concept used to understand and analyze dynamical systems across various disciplines.

In chaos theory, chaotic attractors are fascinating and intricate patterns that emerge in dynamical systems characterized by sensitive dependence on initial conditions. These attractors are central to understanding chaotic behaviour and are often associated with complex and unpredictable dynamics. Chaotic attractors exhibit a few defining characteristics. First and foremost, they are nonperiodic, and do not repeat in a regular, predictable manner. Instead, chaotic attractors display a *seemingly* random and ever-changing nature. This property arises from the system's extreme sensitivity to initial conditions, often referred to as the 'butterfly effect'. Even the tiniest perturbation in the initial state can lead to drastically different outcomes over time, causing trajectories to diverge and explore the system's phase space in intricate patterns. This was first observed and reported on by Edward Lorenz [49] (see Figure 3.1).

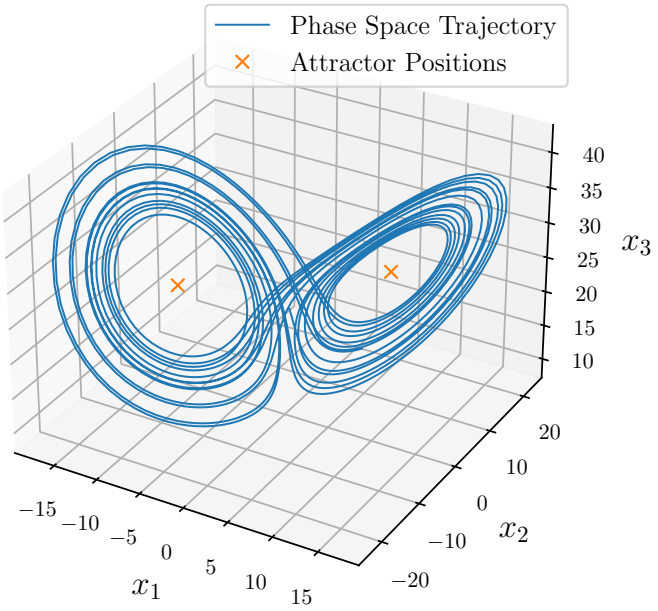


Figure 3.1: The well-known Lorenz attractor.

The exploration of chaotic attractors is not limited to mathematical curiosity; it has practical implications across numerous fields. In physics, chaotic behaviour can be observed in celestial mechanics, fluid dynamics, and even in simple systems like double pendulums. Chaotic attractors have been instrumental in understanding weather patterns, as even small changes in initial conditions can have a profound impact on long-term weather forecasts.

3.2. Quantifying Chaos

Owing to their non-periodic and sensitive nature, chaotic systems require certain additional techniques in order to glean meaningful information about them. Quantifying chaos involves employing various methods and measures to analyze the complexity, unpredictability, and sensitivity to initial conditions exhibited by these systems. These quantitative approaches provide insights into the nature of chaotic dynamics and help distinguish them from regular or random behaviour. Some of the key methods used to quantify chaos are listed below [5, 56]:

1. **Lyapunov Exponents**: Lyapunov exponents characterize the rate of exponential divergence or convergence of nearby trajectories in phase space. Positive Lyapunov exponents indicate sensitive dependence on initial conditions and chaotic behaviour, while zero or negative exponents suggest stability or convergence. The largest Lyapunov exponent is often used as an indicator of chaos.
2. **Fractal Dimensions**: Chaotic attractors possess intricate and self-similar structures. Fractal dimensions measure the complexity and space-filling nature of these attractors. Methods such as box-counting and correlation dimension estimation are used to estimate the dimensionality of a chaotic attractor, revealing the system's self-similarity and complexity.
3. **Power Spectra and Fourier Analysis**: Chaotic signals often exhibit a broad range of frequencies due to their complex and non-periodic nature. Power spectra and Fourier analysis are used to examine the frequency content of chaotic systems. These methods help identify the presence of different frequency components and provide insights into the system's dynamics.
4. **Recurrence Plots and Recurrence Quantification Analysis (RQA)**: Recurrence plots visualize recurrent behaviour in chaotic systems. They represent the times at which states revisit a specific region in phase space. RQA measures, such as recurrence rate, determinism, and entropy, quantify the presence of patterns, predictability, and complexity of recurrent behaviour in chaotic systems.
5. **Entropy Measures**: Entropy measures capture the complexity and information content of a system's dynamics. The Kolmogorov-Sinai entropy (also known as metric entropy) characterizes

the rate at which information about initial conditions is lost due to sensitivity and divergence in chaotic systems. Permutation entropy and other entropy-based measures provide additional insights into the complexity and randomness of a system's behaviour.

6. **Poincaré Maps and Return Time Statistics**: Poincaré maps are two-dimensional slices of a higher-dimensional chaotic attractor. They capture the intersections of the trajectory with a specific plane or surface in phase space. Analyzing the return time statistics of these intersections provides information about the periodic or chaotic nature of the system and the distribution of time intervals between successive returns.
7. **Bifurcation Diagrams**: Bifurcation diagrams illustrate the different attractors and their stability as control parameters are varied in a system. They help identify critical points where qualitative changes occur, leading to the emergence of new attractors or transitions from stable to chaotic behaviour.

Of interest to this project are the first two measures, which are discussed in further detail below.

3.2.1. Lyapunov Exponents

In the context of chaos theory, Lyapunov exponents are a fundamental concept used to quantify the degree of chaos and the sensitivity to initial conditions exhibited by dynamical systems. They provide valuable insights into the exponential divergence or convergence of nearby trajectories in a system's phase space. These exponents measure the average rate at which nearby trajectories in the system's phase space separate or come together. Positive Lyapunov exponents indicate chaotic behaviour, while zero or negative exponents indicate stability or convergence [35, 64].

To understand Lyapunov exponents, consider a system described by a set of differential equations or iterative equations that govern its evolution over time. The phase space of the system represents all possible states or configurations of the system. Trajectories in phase space represent the evolution of the system starting from different initial conditions. The Lyapunov exponents quantify the divergence or convergence of nearby trajectories in the system's phase space. They are defined as the average exponential growth rates of the distances between nearby trajectories over time:

$$|\mathbf{z}_1(t) - \mathbf{z}_2(t)| \propto |\mathbf{z}_1(t_0) - \mathbf{z}_2(t_0)| e^{\Lambda(t-t_0)} \tag{3.1}$$

For a system with n-dimensional phase space, there are n possible orthogonal ways of choosing $|\mathbf{z}_1(t_0) - \mathbf{z}_2(t_0)|$ and hence n Lyapunov exponents, each associated with a particular direction in the phase space. The largest Lyapunov exponent, also known as the maximal Lyapunov exponent (MLE), denoted as Λ_{MLE}, is of particular interest. It characterizes the dominant exponential growth rate of nearby trajectories and serves as an indicator of the system's sensitivity to initial conditions. If Λ_{MLE} is positive, the system exhibits sensitive dependence on initial conditions, and even tiny differences in the initial conditions can lead to significantly divergent trajectories over time.

Lyapunov exponents can be computed numerically using iterative methods such as the Gram-Schmidt process or by analyzing the evolution of tangent vectors in the system's phase space. In practice, the computation of Lyapunov exponents requires tracking the trajectories of a set of nearby initial conditions and calculating the exponential growth rates of their separations.

Lyapunov Time

Lyapunov time is a handy measure to non-dimensionalize the time-axis when comparing results across chaotic systems. Noting the presence of the MLE in the exponent, and that $e^{\Lambda(t-t_0)} = e^{\frac{t-t_0}{1/\Lambda}}$, it is defined as $t_L = 1/\Lambda_{MLE}$. This is an effective normalisation measure, and the measure of a 'prediction horizon' is also defined with its help.

3.2.2. Fractal Dimensions

While a chaotic system can exhibit its dynamics in an n-dimensional phase space, the structure of its attractors can usually be described accurately within a much smaller dimensional space. The **Lyapunov dimension** (also known as the **Kaplan-Yorke dimension**) provides an estimate of the Hausdorff dimension of the chaotic attractor (see [23] for an exact definition). A higher Lyapunov dimension

implies a more complex and intricate attractor. It is derived from the Lyapunov exponents, which as mentioned above quantify the exponential divergence or convergence of nearby trajectories in a chaotic system.

The Kaplan-Yorke conjecture states that the sum of the positive Lyapunov exponents converges to a finite value, and the Lyapunov dimension is equal to the Hausdorff dimension. This conjecture allows us to estimate the fractal dimension of chaotic attractors using the Lyapunov exponents, providing a practical method for characterizing the complexity of chaotic systems:

$$\dim_{Lyap} S = j + \frac{\sum_{i=1}^{j} \Lambda_i}{|\Lambda_{j+1}|} \tag{3.2}$$

where S is the set of all possible chaotic trajectories, Λ_i is the i^{th} Lyapunov exponent[1], and j is the last index where $\sum_{i=1}^{j} \Lambda_i \geq 0$. Further details and extensive examples can be found in [30]. The Kaplan-Yorke conjecture is particularly significant because it relates the quantitative measure of chaos, captured by the Lyapunov exponents, to the qualitative measure of complexity, captured by the fractal dimension. It establishes a connection between the dynamical behaviour and the topological properties of chaotic systems.

While the Kaplan-Yorke conjecture has been validated for many chaotic systems, it is important to note that it is not universally applicable. However, in the commonly encountered chaotic systems, the conjecture holds true and provides valuable insights into the fractal dimensions of chaotic attractors. This can be interpreted as the minimum number of effective dimensions needed to represent the system, which is helpful in deciding the dimensions of the latent space of the auto-encoders.

[1] after having been arranged in a descending order

4

Fundamentals of Machine Learning

In order to develop machine learning models for dynamical systems, it is important to understand the concepts underlying modern day ML. This chapter details the mathematical concepts and training strategies that modern ML, and especially deep learning, rely on. It also outlines certain kinds of model architectures with specific purposes such as model order reduction and series-to-series transformations which will be useful in this project.

4.1. Fundamentals of Machine Learning

4.1.1. The Artificial Neuron

The artificial neuron is a crude approximation of the functioning of a biological neuron, that nonetheless serves as the bedrock for most modern machine learning methods [32]. A single artificial neuron takes a vector $\boldsymbol{x}$ as input, performs a dot product on it with its weight vector $\boldsymbol{w}$ and adds a bias term to it. Finally, it passes this computation through an activation function, which is generally nonlinear in nature. Diagrammatically it is represented in Figure 4.1.

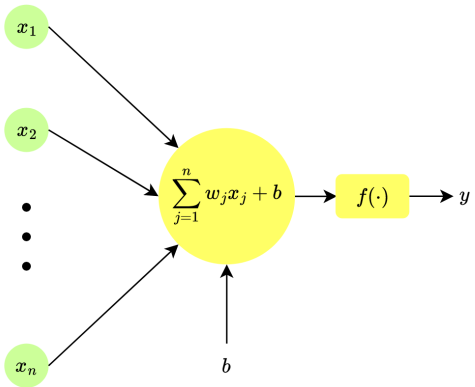


Figure 4.1: Flowchart of an artificial neuron.

The output y can be formally written as:

$$y = f(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{4.1}$$

4.1.2. The Feed-Forward Artificial Neural Network

Stacking artificial neurons together yields a densely connected *layer*, whose output *vector* $\boldsymbol{y}$ can then be written as $\boldsymbol{y} = f(\underline{\boldsymbol{W}}^T \boldsymbol{x} + \boldsymbol{b})$, where:

- $\underline{\boldsymbol{W}}$ is the weight matrix of the layer, and each column is the weight vector of one of the neurons in the layer,

- $\boldsymbol{b}$ is the vector formed from the concatenation of all the biases of the layer's neurons and
- f is applied elementwise on its argument.

This layer of neurons forms the building block of all feed-forward Artificial Neural Networks (ANNs). Just as the neurons were stacked to create a layer, the layers can be stacked (with the output of one layer being fed as input to the next) to form a densely connected feed-forward ANN (as shown in Figure 4.2).

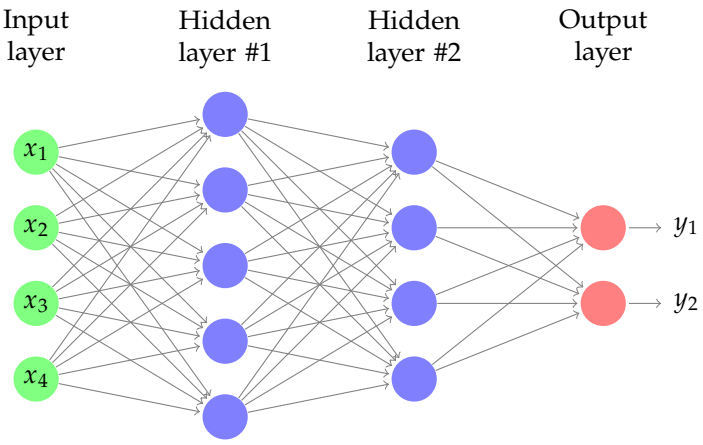


Figure 4.2: A feed-forward ANN.

4.1.3. The Activation Function

The choice of the activation function f is important as it directly impacts the output of an ANN and the computational resources used (for both training and inference). The activation function is also the only place where a nonlinearity is injected into the network. Different kinds of activation functions exist, and some popular choices are listed below [32]:

- Linear - $f(x) = x$
- Sigmoid - $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent (tanh) - $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$
- Rectified Linear Unit (ReLU) - $f(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases}$
- Leaky ReLU - $f(x) = \begin{cases} -\alpha x \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases}$ (α is a small positive number)
- Exponential Linear Unit (ELU) - $f(x) = \begin{cases} -1 + e^x \text{ if } x < 0 \\ x \text{ if } x \geq 0 \end{cases}$

The sigmoid is the classically popular activation function that was eventually phased out in favour of the hyperbolic tangent. These two functions, however, have a 'quashing' effect wherein substantially positive/negative inputs tend to get 'saturated' around the limits of their (respective) outputs. While this might be seen as a desirable property in the output layers as it naturally bounds the output, it is not very useful for the hidden layers [32]. In the hidden layers often a near-linear activation function - like the ReLU (or one of its variants) - is used so that the gradient does not get 'zeroed out' for large layer inputs.

4.1.4. Universal Approximation Theorem

The universal approximation theorem [32, 40] states that any multi-layer feed-forward ANN with a linear output layer and at least one hidden layer with a quashing function is capable of representing a Borel measurable function from one finite dimensional space to another to any desired degree of (non-zero) accuracy, provided sufficient hidden units are available. A discussion on Borel sets and the

Borel measure is beyond the scope of this study, but suffice it to say that the discrete representations of the dynamical systems described in Chapter 6 fall into this category.

While this theorem was initially stated for quashing functions, similar approximation theorems have also been proved for other classes of activation functions, such as the ReLU [47]. All of these approximation theorems point to the fact that the dynamics of a chaotic system are indeed 'learnable' by a sufficiently large ANN.

4.1.5. Stochastic Gradient Descent

There are several heuristic training algorithms for optimizing the weights of an ANN w.r.t. a defined cost function - such as Newton's method, genetic algorithms, conjugate gradient method, et cetera - however, of interest to this study is only the Stochastic Gradient Descent (SGD) method [32]. It is arguably the most widely used heuristic training algorithm and a wildly popular choice for optimizing the weights and biases of an ANN.

To build intuition, consider the function $y = f(x)$. Through simple calculus, $f(x + \epsilon) \approx f(x) + \epsilon \nabla_x f$, for small values of ϵ. Then, one way of minimizing y is to find ϵ such that $f(x + \epsilon) < f(x)$. This can be ensured if the term $\epsilon \nabla_x f$ is negative, which can be achieved if ϵ is of the form $\epsilon = -\eta \nabla_x f$. This gives $f(x + \epsilon) = f(x) - \eta |\nabla_x f|^2$. Thus, this outlines an indirect method to minimize y by affecting tiny changes to x.

In the context of ANNs, y from the previous discussion is the cost function $J(\boldsymbol{\theta})$ (which is to be minimized), and x is $\boldsymbol{\theta}$ (the weights and biases of the ANN). The entire procedure then boils down to two steps:

- selecting a suitable value for η (known as the learning rate) and
- selecting an appropriate way of computing/approximating $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

The original gradient descent algorithm approximates the gradient of the cost function over the entire set of training samples using the empirical probability over the training data as follows:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \mathbb{E}_{\boldsymbol{x}, y \sim \hat{p}_{\text{data}}(\boldsymbol{x}, y)} \left[\nabla_{\boldsymbol{\theta}} L\left(f(\boldsymbol{x}; \boldsymbol{\theta}), y\right) \right] \\ &= \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(f(\boldsymbol{x}^i; \boldsymbol{\theta}), y^i\right) \end{aligned} \tag{4.2}$$

where:

- $J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}, y \sim \hat{p}_{\text{data}}(\boldsymbol{x}, y)} \left[L\left(f(\boldsymbol{x}; \boldsymbol{\theta}), y\right) \right]$ and $L\left(f(\boldsymbol{x}^i; \boldsymbol{\theta}), y^i\right)$ is an appropriately defined loss function per sample $\boldsymbol{x}^i$,
- $\hat{p}_{\text{data}}(\boldsymbol{x}, y)$ is the probability distribution of the training data, and m is the total number of samples available in the training data.

As can be seen, the cost of a single update grows linearly with the sample size m. Nowadays it is not uncommon to have hundreds of thousands of training samples, which makes gradient descent in this form prohibitively expensive. A simple tweak that results in faster convergence and lesser overall computational effort arises from the observation that the LHS in Equation 4.2 is simply an expectation of the gradient. This expectation can be computed over a smaller number m' of randomly drawn samples, an update performed, and this process repeated until the entire training data set is exhausted. These smaller baskets of samples are called 'mini-batches', and this procedure is known as the Mini-batch Stochastic Gradient Descent or simply the Stochastic Gradient Descent.

This still leaves the task of computing the gradient $\nabla_{\boldsymbol{\theta}} L\left(f(\boldsymbol{x}; \boldsymbol{\theta}), y\right)$ for a particular sample. This is done using automatic differentiation and back-propagation, the details of which can be found in [32].

The optimal choice of η and m' is still an area of research and often stems from the researcher's own experience with training ML models. They are canonically referred to as *hyper-parameters*.

4.1.6. Regularization

A key concern with trained ANNs is how well they generalize to data unseen during training. To that end, strategies exist to reduce the test error (possibly at the expense of training error), and these strategies are collectively known as regularization [32]. Listed below are some of the commonly used regularization techniques:

- **Penalty based regularization**
  The common theme here is to add an additional penalty to the cost function as:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y\sim\hat{p}_{\text{data}}(\boldsymbol{x},y)}\left[L\left(f(\boldsymbol{x};\boldsymbol{\theta}),y\right)\right] + \lambda_{penalty}\Omega\left(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y},\boldsymbol{\theta}\right)$$

  where $\lambda_{penalty}$ is a hyper-parameter that determines how much the penalty $\Omega\left(f(\boldsymbol{x};\boldsymbol{\theta}),\boldsymbol{y},\boldsymbol{\theta}\right)$ affects the cost function. This is a compelling idea, of letting the learning strategy itself optimize the new cost function and regularize the ANN. It is also easy to program since it does not require any changes to the network or the learning strategy other than the augmented cost function. Common choices for the function $\Omega\left(\hat{\boldsymbol{y}},\boldsymbol{y},\boldsymbol{\theta}\right)$ include the L^1 and L^2 norms of $\boldsymbol{\theta}$ [32].
- **Noise robustness**
  Noise can be added to the inputs, hidden units, outputs, and weights during training. The various analyses are provided in [32], and the underlying current through all these augmentations is that noise-injection is a stochastic implementation of Bayesian inference over the quantities to which it is added. This generally results in a network that has been galvanized against variations in the noise-injected quantities and a more robust ANN overall.
- **Dropout**
  Dropout is a particular method of noise-injection in the hidden units that merits its own separate mention. It is a practical (stochastic) method of training an ensemble of networks without actually having to train multiple networks. At every weight update step, some non-output hidden units are randomly selected and set to zero. An in-depth discussion can be found in [32].
- **Early stopping**
  Typically learning curves during the training of an ANN look like the one shown in Figure 4.3. The training loss keeps on reducing with every epoch. However, the validation loss drops to a point and eventually starts increasing. This signals *over-fitting* - where the ANN starts to model the training data better and better but, in the process, loses out on its ability to generalize on unseen data. To combat this, a general strategy is presented. A copy of the parameters $\boldsymbol{\theta}$ is stored every time the validation loss improves. In parallel, a count of the number of epochs for which the validation has not improved is also kept. If this count exceeds a pre-defined value (called *patience*), the training is halted and the parameters restored to the stored ones (corresponding to the best achieved validation loss). It can be seen as a computationally efficient way of selecting a hyper-parameter - the number of training epochs.

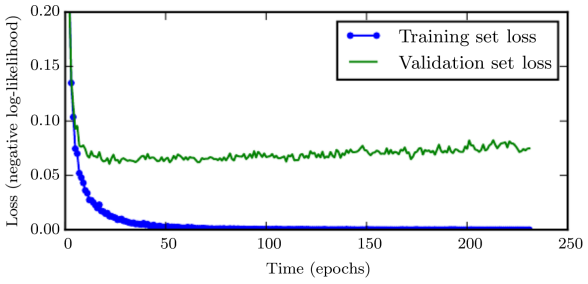


Figure 4.3: Typical loss curves obtained during training [32].

4.2. Generative Adversarial Networks and Autoencoders

Generative Adversarial Networks (GANs) are a kind of ML algorithm/framework that involves the simultaneous training of two networks pitted against each other in an adversarial manner (hence the name). The goal here is to develop a generative model that serves as a mapping from a latent space (that the data is hypothesized to reside in) to the data itself, where the model successfully approximates the probability $p_{data}(\boldsymbol{x})$. The first network is called the *Generator*, and its task is to learn a mapping from the latent space to the space of the data. In contrast, the second network is called a *Discriminator*, and its task is to predict whether a given input was generated by the generator or belongs to the actual data. The combined cost function can then be written as:

$$J(\boldsymbol{\theta}_G,\boldsymbol{\theta}_D) = \mathbb{E}_{\boldsymbol{x}\sim\hat{p}_{\text{data}}(\boldsymbol{x})}\left[L\left(\mathcal{D}\left(\boldsymbol{x}\right)\right)\right] + \mathbb{E}_{\boldsymbol{z}\sim p_{\text{latent space}}(\boldsymbol{z})}\left[L\left(1-\mathcal{D}\left(\mathcal{G}(\boldsymbol{z})\right)\right)\right]$$

where L is an appropriate loss function, $\mathcal{G}(z)$ is the output of the generator, $\mathcal{D}(x)$ is the output of the discriminator (which is the probability that its input is *fake*, i.e. produced by the generator). Further details can be found in [32, 33], but this is sufficient to motivate a discussion on Autoencoders.

Autoencoders are a ML strategy that seeks to find a meaningful alternative representation of a given data set and be able to decode that representation such that the reconstruction error is minimized [6]. Like GANs, they too consist of two ANNs placed adversarially against each other and trained simultaneously. The *Encoder* network takes as input a sample drawn from the data distribution $p_{\text{data}}(x)$. It produces an encoded representation (said to reside in the latent space). The decoder, on the other hand, takes a sample drawn from the latent space' distribution $p_{\text{latent space}}(z)$ and aims to produce data points distributed according to the original data distribution $p_{\text{data}}(x)$. In practice, this boils down to minimizing the following cost function:

$$J(\theta_{\text{encoder}}, \theta_{\text{decoder}}) = \mathbb{E}_{x \sim \hat{p}_{\text{data}}(x)} \left[L(\mathcal{D}(\mathcal{E}(x)), x) \right]$$

where L is an appropriate reconstruction error function (generally the mean squared error function), $\mathcal{E}(x)$ is the encoded representation produced by the encoder and $\mathcal{D}(z)$ is the reconstructed vector produced by the decoder. All the regularization techniques mentioned in subsection 4.1.6 can be applied here, and an in-depth discussion is provided in [6]. Note that if the latent space is defined to have a lower dimensionality than the original data, then the autoencoder can potentially find a meaningful *compressed* representation of the original data Figure 4.4.

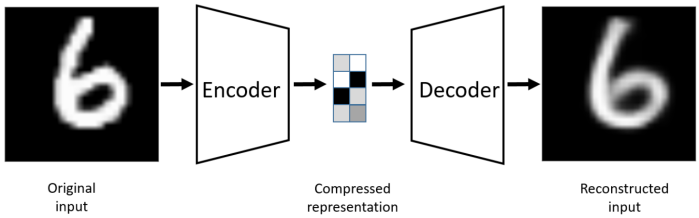


Figure 4.4: An autoencoder setup that is tasked with finding a compressed representation of the original data [6].

An interesting kind of autoencoders worth mentioning here are *contractive autoencoders*, wherein an additional penalty is added to the cost function. This penalty is the Frobenius norm of the Jacobian of $\mathcal{E}(x)$ w.r.t x [6]. This penalty seeks to minimize the variation in the encoded representation w.r.t. small changes in the encoder input. This should, in theory, result in a smooth latent space representation if the data being encoded is smooth - which is the case in all the systems mentioned in Chapter 6.

4.3. Networks for Series to Series Transformations

So far, the models mentioned act on individual samples, assuming no correlation between those samples. This section describes specialized ANNs that act on series of sequential data, actively considering the fact that the samples are drawn form a coherent sequence. This family of ANNs is called *Recurrent Neural Networks* (RNN), and the critical feature shared amongst all its members is the presence of a hidden cell state that is passed along and captures some information about the data seen by the ANN so far. Figure 4.5 is a diagrammatic representation of an RNN, and the arrow originating and ending at the hidden unit represents the passed-along hidden state (i.e. the *recurrence*). Such networks can be trained using the same SGD strategy described in subsection 4.1.5, with the gradient computed using a tweaked back-propagation algorithm called Back-Propagation Through Time (BPTT). This unrolls the network in time (as shown in Figure 4.6) and then computes the gradients using regular back-propagation [32].

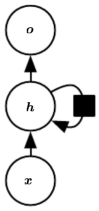


Figure 4.5: Typical graph of an RNN [32].

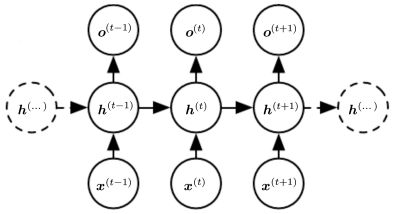


Figure 4.6: Unrolled graph of an RNN [32].

4.3.1. Simple RNN

The simple RNN is one of the least complicated RNNs. It consists of a single hidden cell state that is updated at every time step and is also used to produce the output. Its update equations are provided below:

$$\boldsymbol{a}^{(t)} = \underline{\boldsymbol{W}}_{hh}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{hx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_h \tag{4.3}$$

$$\boldsymbol{h}^{(t)} = \tanh\left(a^{(t)}\right) \tag{4.4}$$

$$\boldsymbol{o}^{(t)} = \underline{\boldsymbol{W}}_{oh}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o \tag{4.5}$$

This simple setup performs extremely well at learning time series representations, however it suffers from the well-known vanishing/exploding gradient problem [32] which can make its training challenging.

4.3.2. Long Short-Term Memory Networks

As mentioned in the previous section, simple RNNs suffer from an unstable gradient, either vanishing or exploding, thus making the learning of long-term dependencies tough. Long Short-Term Memory Networks (LSTMs) are a cell architecture to combat the vanishing gradient problem, by introducing *memory cells*, and were proposed by Hochreiter and Schmidhuber in [39]. These cells are self-recurrent neurons, with *input*, *forget* and *output* gates which determine how much of the input's information to use, how much of the previous state's information to forget and how much to pass on (respectively). This is one of the earliest examples of a gated RNN architecture where instead of all information from the inputs and the hidden states being propagated forward, some is purposefully 'forgotten' [75, 32]. Its update equations are provided below, and the cell's architecture is shown in Figure 4.7.

$$\text{(input gate) } \boldsymbol{i}^{(t)} = \sigma\left(\underline{\boldsymbol{W}}_{ih}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{ix}\boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right) \tag{4.6}$$

$$\text{(forget gate) } \boldsymbol{f}^{(t)} = \sigma\left(\underline{\boldsymbol{W}}_{fh}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{fx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) \tag{4.7}$$

$$\text{(output gate) } \boldsymbol{o}^{(t)} = \sigma\left(\underline{\boldsymbol{W}}_{oh}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{ox}\boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right) \tag{4.8}$$

$$\text{(candidate cell memory) } \tilde{\boldsymbol{C}}^{(t)} = \tanh\left(\underline{\boldsymbol{W}}_{ch}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{cx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right) \tag{4.9}$$

$$\text{(updated cell memory) } \boldsymbol{C}^{(t)} = \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{C}}^{(t)} + \boldsymbol{f}^{(t)} \circ \boldsymbol{C}^{(t-1)} \tag{4.10}$$

$$\text{(updated output) } \boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh\left(\boldsymbol{C}^{(t)}\right) \tag{4.11}$$

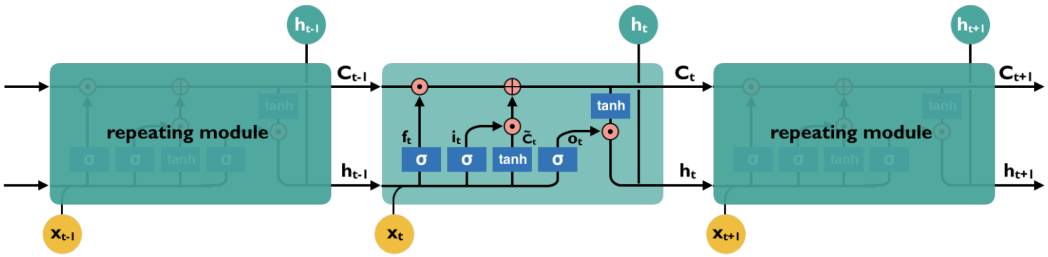


Figure 4.7: An LSTM cell and its inner connections [75].

4.3.3. Gated Recurrent Units

Gated Recurrent Units (GRUs) were first proposed by Cho et al. in [14] as an improvement over a simple RNN cell for modelling long-term dependencies. This cell architecture also uses gates - the *update* and *reset* gates - and, just like the LSTM, manages to avoid the vanishing gradient problem. Its update equations[1] are given below, and the cell's architecture is shown in Figure 4.8.

$$\text{(update gate) } \boldsymbol{z}^{(t)} = \sigma\left(\underline{\boldsymbol{W}}_{zh}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{zx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_z\right) \tag{4.12}$$

$$\text{(reset gate) } \boldsymbol{r}^{(t)} = \sigma\left(\underline{\boldsymbol{W}}_{rh}\boldsymbol{h}^{(t-1)} + \underline{\boldsymbol{W}}_{rx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right) \tag{4.13}$$

$$\text{(candidate activation) } \tilde{\boldsymbol{h}}^{(t)} = \tanh\left(r^{(t)} \circ \left(\underline{\boldsymbol{W}}_{hh}\boldsymbol{h}^{(t-1)}\right) + \underline{\boldsymbol{W}}_{hx}\boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right) \tag{4.14}$$

$$\text{(updated activation) } \boldsymbol{h}^{(t)} = \boldsymbol{z}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)} + \left(1 - \boldsymbol{z}^{(t)}\right) \circ \boldsymbol{h}^{(t-1)} \tag{4.15}$$

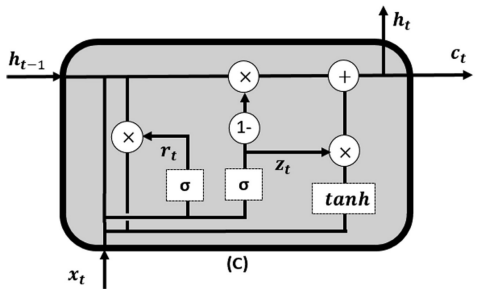


Figure 4.8: A GRU cell and its inner connections [3].

This cell architecture, while safe from the vanishing gradient problem, is still prone to the exploding gradient problem, much like the LSTM. It has been reported in [15] (and numerous studies since) that GRUs perform comparably to LSTMs. As can be gleaned from the fewer update equations and a missing gate, they are also more computationally efficient than an LSTM cell.

4.3.4. Echo State Networks

The RNNs described thus far are all trained using BPTT, and prone to unstable gradients. Additionally, the training itself is quite computationally expensive making it time-consuming and resource-heavy. An easier alternative was proposed by Jaeger in [41], called Echo State Networks (ESNs). These class of models are a subset of what has now been come to known as Reservoir Computing (RC). The key insight here is that for a sufficiently large and adequately initialized 'reservoir' (the recurrent matrix in a simple RNN), not all the weights need to be trained. Only optimizing the weights of the output layer is

[1] These are the so-called 'gate after' equations, where the reset gate is applied after the matrix multiplication $\underline{\boldsymbol{W}}_{hh}\boldsymbol{h}^{(t-1)}$. This formulation has lately become the standard, having lent itself to a more efficient GPU implementation as compared to the original (where the reset gate was applied to $\boldsymbol{h}^{(t-1)}$ before the multiplication with $\underline{\boldsymbol{W}}_{hh}$).

sufficient to learn an effective series-to-series representation. The update equations are given below:

$$\text{(candidate activation) } \tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\underline{\boldsymbol{W}}_{in}\boldsymbol{x}^{(t)} + \underline{\boldsymbol{W}}_{res}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h\right) \tag{4.16}$$

$$\text{(updated activation) } \boldsymbol{h}^{(t)} = (1-\alpha)\boldsymbol{h}^{(t-1)} + \alpha\tilde{\boldsymbol{h}}^{(t)} \tag{4.17}$$

$$\text{(updated output) } \hat{\boldsymbol{y}} = \underline{\hat{\boldsymbol{W}}}_{out}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o \tag{4.18}$$

where $\underline{\boldsymbol{W}}_{in}$ is the input matrix, $\underline{\boldsymbol{W}}_{res}$ is the 'reservoir', $\underline{\hat{\boldsymbol{W}}}_{out}$ is the output matrix and α is the leaking rate ($\alpha \in (0, 1]$). Note that $\underline{\hat{\boldsymbol{W}}}_{out}\boldsymbol{h}^{(t)} + \boldsymbol{b}_o = \underline{\boldsymbol{W}}_{out}\left[1; \boldsymbol{h}^{(t)}\right]$, where the $\underline{\boldsymbol{W}}_{out}$ is simply $\underline{\hat{\boldsymbol{W}}}_{out}$ concatenated column-wise with $\boldsymbol{b}_o$.

With these equations, the ESN setup task is then:

- setup $\underline{\boldsymbol{W}}_{in}$ and $\underline{\boldsymbol{W}}_{res}$ suitably (these weighs are fixed and not updated during training),
- train the weights of $\underline{\boldsymbol{W}}_{out}$ such that the cost function $J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}\left[L\left(\hat{\boldsymbol{y}}, \boldsymbol{y}\right)\right]$ (L being some pre-defined loss function and $\hat{\boldsymbol{y}}$ the output of the ESN for the input $\boldsymbol{x}$) is minimized.

Taking the loss L to be the mean square error function, the weights of the matrix $\underline{\boldsymbol{W}}_{res}$ can be computed using least squares regression, as opposed to using SGD. This has the benefit of greatly speeding up the training of an ESN when compared to other RNNs that rely on BPTT for weight optimization. This can be done as follows:

$$\underline{\boldsymbol{W}}_{out} = \underline{\boldsymbol{Y}}\,\underline{\hat{\boldsymbol{H}}}^T\left(\underline{\hat{\boldsymbol{H}}}\,\underline{\hat{\boldsymbol{H}}}^T\right)^{-1} \tag{4.19}$$

where:

- $\underline{\boldsymbol{Y}} \in \mathbb{R}^{n_y \times n_T}$ is the column-wise concatenation of all $\boldsymbol{y}$ (the true output) and
- $\underline{\hat{\boldsymbol{H}}} \in \mathbb{R}^{(n_h+1)\times n_T}$ is the column-wise concatenation of all $\left[1; \boldsymbol{h}^{(t)}\right]$ (the concatenated activation).

The inverse term in Equation 4.19 may not exist if $\underline{\hat{\boldsymbol{H}}}\,\underline{\hat{\boldsymbol{H}}}^T$ is singular or poorly conditioned. To combat this, a small additive matrix is added to it as follows:

$$\underline{\boldsymbol{W}}_{out} = \underline{\boldsymbol{Y}}\,\underline{\hat{\boldsymbol{H}}}^T\left(\underline{\hat{\boldsymbol{H}}}\,\underline{\hat{\boldsymbol{H}}}^T + \beta\underline{\boldsymbol{I}}\right)^{-1} \tag{4.20}$$

While this addition is primarily done to aid in the numerical stability when taking the inverse, it is well-known that this term, when viewed through the lens of the cost function, is equivalent to adding an additional L^2-norm penalty:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_{data}(\boldsymbol{x})}\left[L\left(\hat{\boldsymbol{y}}, \boldsymbol{y}\right)\right] + \frac{\beta}{2}\|\underline{\boldsymbol{W}}_{out}\|^2 \tag{4.21}$$

What is now left is to see how to set up the matrices $\underline{\boldsymbol{W}}_{in}$ and $\underline{\boldsymbol{W}}_{res}$, and how to choose α. A detailed discussion on both parameter tuning and the respective influences of $\underline{\boldsymbol{W}}_{in}$ and $\underline{\boldsymbol{W}}_{res}$ on the output can be found in [50]. Only a few salient points are touched upon below:

- **The Input Matrix, $\underline{\boldsymbol{W}}_{in}$**
  Elements of $\underline{\boldsymbol{W}}_{in}$ are taken from a uniform distribution in the interval $[-a, a]$, where a is known as the input scaling. It serves to scale and shift the input before it is fed into the reservoir. Canonically $\underline{\boldsymbol{W}}_{in}$ has been taken to be a dense matrix, but in recent years it has been noticed that making $\underline{\boldsymbol{W}}_{in}$ sparse has no discernible effect on the outputs. Thus, the only important parameter here is the input scaling a, since it determines how non-linear the reservoir responses are (by virtue of residing within the argument of $\tanh(\cdot)$ in Equation 4.12). For linear tasks, a should be small, thus allowing the $\tanh(\cdot)$ to operate in its linear region. A similar argument can be made for nonlinear tasks, although a large input scaling could saturate the $\tanh(\cdot)$ leading to worse results.

- **The Reservoir, $\underline{\boldsymbol{W}}_{res}$**
  As described in [50], the reservoir really serves two purposes. Firstly, much like a kernel method it serves as a higher dimensional expansion of the input, where the dynamics of the system are hopefully easier to predict. Second, it also serves as the memory of the ESN, telling it which values and how much of them to pass on. Thus, this is a crucial aspect of ESNs, and a poorly initialized reservoir will lead to poorly positioned outputs. Three things determine the reservoir, its size, its sparsity and its spectral radius.
  - The size of the reservoir n_h is specified in [50] to be at least $n_x \times \hat{n}_T$ where $\hat{n}_T$ is the number of time-steps the reservoir is expected to remember. However, it is also noted in [50] that the non-linear dynamics typically can lend themselves to reservoirs with acceptable sizes smaller than $n_x \times \hat{n}_T$. Anecdotal evidence points to $10^2 n_x$ being a good initial guess.
  - It is recommended to make the reservoir highly sparse by setting most of the weights to zero [41, 50]. This has the added benefit of sped-up computations. Lukoševičius notes that the degree of sparsity has little effect on the final outputs and is not that important a parameter.
  - The spectral radius $\rho(\underline{\boldsymbol{W}}_{res})$ is perhaps the most important parameter of the reservoir. It is the maximum of the absolute value of the eigenvalues of $\underline{\boldsymbol{W}}_{res}$. It is crucial to the functioning of an ESN that the reservoir satisfy the so-called 'echo state property', which (loosely speaking) states that for a long enough input history of $\boldsymbol{x}$, the reservoir state $\boldsymbol{h}$ should not depend on the initial conditions. Large values of ρ can lead to reservoirs that violate this property and in general $\rho < 1$ enforces the echo state property. However, these are not axioms, and it is possible to have reservoirs with $\rho < 1$ that violate the echo state property and those with $\rho > 1$ that satisfy it.
- The choice of α is harder to make, and must be set through trial and error. Lukoševičius provides a rough comparison between α and Δt in an Euler time-stepping scheme in [50], but even they note that there is no analytical way to compute it.

4.4. Summary

An overview of the basics of machine learning is provided in this chapter, along with a general purpose gradient based optimization method and generalization-ability enhancing devices (regularization). Further, a machine learning based strategy to produce reduced-order representations is outlined in the section on auto-encoders, and various state-of-the-art network models for handling time-series data are also outlined in their respective sections.

5

Reduced Order Modelling

This chapter provides a foundational overview of reduced order modelling (ROM) techniques, offering insights into the fundamental concepts and principles that underpin this field. Covered here are the basics of traditional ROM (based on proper orthogonal decomposition and Galerkin projection) along with a review of existing literature on machine learning based ROM techniques.

5.1. Classical POD with Galerkin Projection

5.1.1. Proper Orthogonal Decomposition

Proper Orthogonal Decomposition (POD) is closely related to the concept of Principal Component Analysis (PCA) from statistics. It can in fact, be seen as a simple application of PCA to dynamical data. It was first applied to the analysis of turbulence by Lumley in [52, 51] and has since been applied to a host of other non-linear chaotic systems [37, 8, 9, 62]. The principle driving POD is to find a reduced basis of the phase space, such that the linear projection onto this basis has the greatest contribution to the 'energy' of the system. The term 'energy' here is defined as the square of the L^2 norm of the phase vector. The problem can then also be viewed as simply finding a reduced basis with the lowest reconstruction mean squared error (MSE).

Let the basis Φ be composed of n_p unit vectors $\{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_{n_p}\}$, where $n_p \leq n_u$ (n_u being the dimension of the phase space of the original system, and $\boldsymbol{\phi}_i \in \mathbb{R}^{n_u}$). A projection of $\boldsymbol{u}$ onto the basis Φ can be written as:

$$\boldsymbol{u}^{proj} = \underline{\boldsymbol{P}}\boldsymbol{a} \tag{5.1}$$

$$\boldsymbol{a} = \underline{\boldsymbol{P}}^T\boldsymbol{u} \tag{5.2}$$

where $\underline{\boldsymbol{P}} \in \mathbb{R}^{n_u \times n_p}$ is a matrix whose i^{th} column is the vector $\boldsymbol{\phi}_i$, $\boldsymbol{a} \in \mathbb{R}^{n_p}$ is a vector of the coefficients (that each of the basis vectors $\boldsymbol{\phi}_i$ are to be multiplied by) and $\boldsymbol{u}^{proj}$ is the linear projection of $\boldsymbol{u}$ onto the basis Φ. Formally, finding the basis Φ can then be written out as an optimization task:

$$\underline{\boldsymbol{P}} = \underset{\underline{\boldsymbol{P}}}{\operatorname{argmin}}\, \mathbb{E}_{\boldsymbol{u}\sim p(\boldsymbol{u})}\left[\|\boldsymbol{u} - \underline{\boldsymbol{P}}\underline{\boldsymbol{P}}^T\boldsymbol{u}\|_2\right] \tag{5.3}$$

This has been shown to be equivalent to solving the following eigenvalue problem [51]:

$$\underline{\boldsymbol{C}}\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \tag{5.4}$$

where $\underline{\boldsymbol{C}} \in \mathbb{R}^{n_u \times n_u}$ is the covariance matrix of the snapshots matrix $\underline{\boldsymbol{S}}_u \in \mathbb{R}^{n_u \times n_t}$, $\boldsymbol{v}_i$ its eigenvectors and λ_i its eigenvalues. The snapshots matrix $\underline{\boldsymbol{S}}_u$ is a matrix composed of the column-wise concatenation of n_t 'snapshots' of the state vector $\boldsymbol{u}$ taken at different time instances, while $\underline{\boldsymbol{C}}$ is defined as:

$$\underline{\boldsymbol{C}} = \left(\frac{1}{n_t - 1}\right)\underline{\boldsymbol{S}}_u\,\underline{\boldsymbol{S}}_u^T \tag{5.5}$$

Clearly, $\underline{\boldsymbol{C}}$ is real and symmetric, and hence its eigenvalues are real and its eigenvectors orthogonal to each other. Thus, its eigenvectors can be used to form an orthonormal basis, and $\underline{\boldsymbol{C}}$ can be decomposed as follows:

$$\underline{\boldsymbol{C}} = \underline{\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T} = \underline{\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}} \tag{5.6}$$

where,

$$\underline{\boldsymbol{\Lambda}} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_{n_u} \end{bmatrix} \in \mathbb{R}^{n_u \times n_u}$$

$$\underline{\boldsymbol{V}} = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots & \boldsymbol{v}_{n_u} \end{bmatrix} \in \mathbb{R}^{n_u \times n_u}$$

It is easy to see that $\text{tr}\left(\underline{\boldsymbol{C}}\right) = \text{tr}\left(\underline{\boldsymbol{\Lambda}}\right) = \sum_i \lambda_i$. If the 'energy' of the system, as mentioned earlier, is defined as the square of the L^2 norm of the state vector, then $\sum_{j=1}^{n_t} |\boldsymbol{u}^{(j)}|_2^2 = (n_t - 1)\text{tr}\left(\underline{\boldsymbol{C}}\right) = (n_t - 1)\text{tr}\left(\underline{\boldsymbol{\Lambda}}\right) = (n_t - 1)\sum_i \lambda_i$, and the projection along $\boldsymbol{v}_i$ can be seen as contributing a fraction $\lambda_i / \sum_i \lambda_i$ to the total energy.

Hence, the reduced basis Φ is composed of the eigenvectors corresponding to the n_p largest eigenvalues of the matrix $\underline{\boldsymbol{C}}$.

It is worth noting that it is general practice to subtract the mean vector $\bar{\boldsymbol{u}} = \frac{1}{n_t}\sum_{j=1}^{n_t} \boldsymbol{u}^{(j)}$ from the state vectors $\boldsymbol{u}$ before computing the snapshot matrix[1], in essence only modelling the variations around a time-mean of the state vector. In this context, it also becomes easy to see where the 'energy' analogy comes from, since in a fluid dynamical system where $\boldsymbol{u}$ is the concatenation of the velocities at different grid points, $\frac{1}{2}\left(\frac{1}{n_t}\sum_{j=1}^{n_t} |\boldsymbol{u}^{(j)} - \bar{\boldsymbol{u}}|_2^2\right) = TKE$ (the average turbulent kinetic energy).

5.1.2. Galerkin Projection

Galerkin projection methods are a family of methods based around the principle of finding a reduced set of basis functions that can be used to represent a dataset, and then modelling the coefficients attached with these basis functions instead of computing the original field itself. Within the context of POD, this would imply, having computed a suitable basis $\boldsymbol{\phi}$, computing the time-evolution of the coefficient vector $\boldsymbol{a}$ (from Equation 5.2) directly, as opposed to computing the time-evolution of the entire vector $\boldsymbol{u}$ [8, 9, 62, 37]. The state vector $\boldsymbol{u}$ at any time instance can then be computed from the coefficient vector and the basis vectors. As done earlier, consider a system composed of linear and non-linear parts:

$$\begin{aligned} \dot{\boldsymbol{u}} &= \underline{\boldsymbol{L}}\boldsymbol{u}(t) + \boldsymbol{N}(\boldsymbol{u}(t)) \\ \boldsymbol{u}(t=0) &= \boldsymbol{u}_0 \end{aligned}$$

Replacing $\boldsymbol{u}$ with $\underline{\boldsymbol{P}}\boldsymbol{a}$ yields:

$$\begin{aligned} \underline{\boldsymbol{P}}\dot{\boldsymbol{a}} &= \underline{\boldsymbol{L}\boldsymbol{P}}\boldsymbol{a}(t) + \boldsymbol{N}(\underline{\boldsymbol{P}}\boldsymbol{a}(t)) \\ \implies \underline{\boldsymbol{P}^T\boldsymbol{P}}\dot{\boldsymbol{a}} &= \left(\underline{\boldsymbol{P}^T\boldsymbol{L}\boldsymbol{P}}\right)\boldsymbol{a}(t) + \underline{\boldsymbol{P}^T}\boldsymbol{N}(\underline{\boldsymbol{P}}\boldsymbol{a}(t)) \\ \implies \dot{\boldsymbol{a}} &= \left(\underline{\boldsymbol{P}^T\boldsymbol{L}\boldsymbol{P}}\right)\boldsymbol{a}(t) + \underline{\boldsymbol{P}^T}\boldsymbol{N}(\underline{\boldsymbol{P}}\boldsymbol{a}(t)) \end{aligned} \tag{5.7}$$

since (considering all the columns of $\underline{\boldsymbol{P}}$ are a subset of the set of orthonormal eigenvectors of the covariance matrix $\underline{\boldsymbol{C}}$),

$$\underline{\boldsymbol{P}^T}\underline{\boldsymbol{P}} = \begin{bmatrix} \boldsymbol{v}_1^T\boldsymbol{v}_1 & \boldsymbol{v}_1^T\boldsymbol{v}_2 & \cdots & \boldsymbol{v}_1^T\boldsymbol{v}_{n_u} \\ \boldsymbol{v}_2^T\boldsymbol{v}_1 & \boldsymbol{v}_2^T\boldsymbol{v}_2 & \cdots & \boldsymbol{v}_2^T\boldsymbol{v}_{n_u} \\ \vdots & & \ddots & \vdots \\ \boldsymbol{v}_{n_u}^T\boldsymbol{v}_1 & \boldsymbol{v}_{n_u}^T\boldsymbol{v}_2 & \cdots & \boldsymbol{v}_{n_u}^T\boldsymbol{v}_{n_u} \end{bmatrix} = \underline{\boldsymbol{I}}_{n_u}$$

[1] Easy to see where the name 'covariance matrix' comes from now.

All that remains is to compute the new operators, and then the time-evolution of the coefficients-vector $\boldsymbol{a}$ can be proceeded with as any other system. Since the reduced basis is generally much lower in dimensionality than the original state vector, this approach has the benefit of cutting down on a significant amount of computational effort.

5.1.3. Limitations

The POD-Galerkin approach is known to suffer from a number of problems [73]. Its key pitfall is that it only models the *statistically* important modes, neglecting the rest. This completely neglects any dynamical properties of the system, in which even statistically insignificant modes may play key roles (such as conduits for the exchange of energy between more active modes [65]). Further, the non-linear operator $\boldsymbol{N}(\underline{\boldsymbol{P}}\boldsymbol{a}(t))$ is assumed to depend only on the coefficients $\boldsymbol{a}$. This is known as the 'flat Galerkin' approximation [73], and is known to introduce additional errors in the evaluation of the time-derivative of $\boldsymbol{a}$ which also degrades performance. Take for example the POD-Galerkin approach applied to the Charney-DeVore system, as presented in [73]. Even though the three and four mode reduced system capture 97.6% and 98.9% (respectively) of the total variance, the Galerkin model of these POD projections end up producing a single-point attractor. This is in contrast to the strange attractor that has far richer dynamics, that failed to get modelled in this linear approach (Figure 5.1).

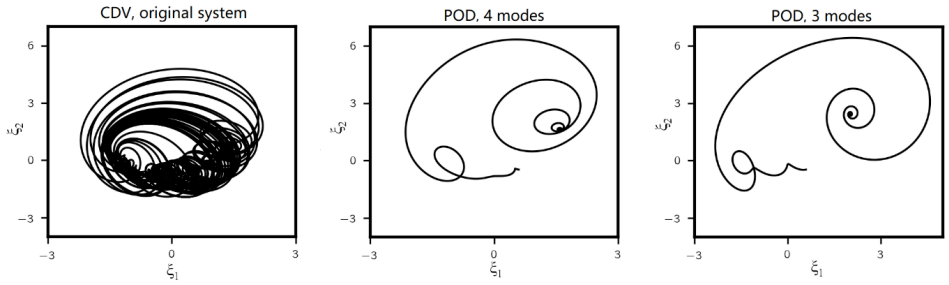


Figure 5.1: 2000 time-unit trajectories of the CDV system, projected to the first two POD modes [73].

5.2. Deep Learning Approaches

The explosion of deep learning, along with its associated specialized hardware and frameworks, has made it an attractive option to try and model physical systems. As such, in recent years, it has been applied to various dynamical systems, in an ever-increasing variety of ways. It thus becomes essential to separate these approaches out into specific categories, so as to be able to identify the common threads among these differing approaches. Typical deep learning based ROMs can broadly be classified into two categories:

- using a single ANN model (like a RNN or another architecture adapted to time-series prediction) and applying it to either the full order model or a POD-ROM,
- using two separate ANN models - one for the construction of a reduced order model and the other for the propagation of the reduced state in time.

A parallel classification can also be inferred based on whether the training (and/or network architecture itself) uses information from the specific model of the physical system being predicted - *model-free* methods and *model-supplemented* methods[2]. Model-supplemented methods include the likes of Physics Informed Neural Networks [11]. This study, however, focuses only on model-free methods.

5.2.1. Single-Model Prediction

Single-model prediction methods, while united in their usage of a single model, differ widely in the choice of ANN models used, the architecture and workflow of the network and even the training strategy. This section provides an overview of such methods.

[2]Note that the Galerkin projection method is a model-supplemented method since it uses the structure of the original system's time-evolution equation to create the time-evolution equation for the vector of coefficients ($\boldsymbol{a}$).

Simple Feed-forward ANNs

A single memory-less deep feed-forward ANN has been a popular choice in this field, owing to its relative simplicity. It has been employed in several studies, often alongside other models in an attempt to predict the evolution of the system. Chattopadhyay, Hassanzadeh, and Subramanian in [13] used a single deep ANN to model a multiscale three-tier extension of the Lorenz 96 system - a climate model with multiple time-scales mimicking the dynamics found in an actual full-scale atmospheric model. The ANN is trained using the SGD algorithm to map an input $\boldsymbol{u}^{(t)}$ to $\Delta\boldsymbol{u}^{(t)} = \boldsymbol{u}^{(t+1)} - \boldsymbol{u}^{(t)}$. In the predictive phase, the output $\boldsymbol{u}^{(t+1)}$ is computed using the Adams-Bashforth method as $\boldsymbol{u}^{(t+1)} = \boldsymbol{u}^{(t)} + \frac{1}{2}\left(3\Delta\boldsymbol{u}^{(t)} - \Delta\boldsymbol{u}^{(t-1)}\right)$. This convoluted process is undertaken because a simple feed-forward ANN is observed to better predict $\Delta\boldsymbol{u}^{(t)}$ as opposed to $\boldsymbol{u}^{(t+1)}$ directly. The normalized RMSE is plotted w.r.t. the time (non-dimensionalized with the Lyapunov time) in Figure 5.2, and as can be clearly observed that despite all the precautions taken, the ANN performs worse than the RNNs in terms of accurate short-term predictions.

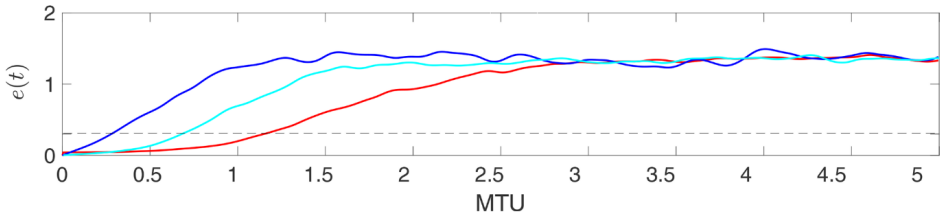


Figure 5.2: Time evolution of the normalized RMSE for three different prediction methods - ESN (red), LSTM (cyan) and feed-forward ANN (blue). [13]

This is attributed to the lack of any system memory possessed by the feed-forward ANN, and is a clear win for RNN models with a passed along hidden state.

In [83], Zhuang et al. model the temperature distribution (and variation) of a heat-sink using a similar ANN-based time-modeller as in the previous study. However, here they employ POD to effect model-order-reduction first. Further of interest is the fact that in addition to a simple ANN, they also test the effect of Runge-Kutta inspired networks, namely the Euler and the RK4 methods. However, these methods are tested by simply letting the ANN learn the time-derivative in the RK setting instead of employing an actual residual connection. The study finds the RK networks to perform marginally better for coarser time-stepped data, whilst at a lower time-step neither architecture could conclusively be said to possess superior performance.

CNNs

Lee and You use an interesting approach in [46], wherein they train a set of CNNs to operate on four consecutive time instances of velocity and pressure fields in order to predict their values at the next (i.e. the fifth) time instance. Further, they also make use of the concept of a GAN, wherein they train a discriminator in tandem, whose job it is to predict whether a given snapshot came from the original data or was generated by the set of CNNs. They train this network on the canonical problem of flow past a circular cylinder, using the incompressible Navier-Stokes equations. The CNNs all operate on different resolutions of the domain, the aim here being that different networks then specialize and learn features from different spatial scales. Further, information from the larger scales is also passed onto the CNNs operating on finer grid resolutions. The chaining together of four consecutive time instances, to be used as a combined input, can be seen as providing the network with some dynamical information about the system just prior to the update step. Having trained on multiple Reynolds number flow cases, they find that the networks have slower error growth rates for lower Reynolds numbers. This implies that the networks learn to model large scale vortical structures (such as those active in vortex shedding) more accurately than the smaller scale ones. This is also apparent in the networks' inability to model smaller scale structures (in the wake) accurately in larger Renolds number flows. It is also important to note here that this approach is not entirely model-free, since they do use physics-inspired penalties in the loss function (mass and momentum conservation). They have trained an equivalent network without these penalties, and found that without the imposition of these penalties un-physical vortical structures tend to develop after a few time steps. This approach is also quite complex in terms of the

programming involved, owing to the complex structure of the network and how the different CNNs interact with each other, and takes a long time to train.

LSTMs, GRUs and ESNs

LSTMs, GRUs and ESNs are mainstays of time-series modelling, and have been used extensively to predict the evolution of chaotic systems. Briefly summarised below are some important studies employing these in the service of the prediction of chaotic systems.

In [13], Chattopadhyay, Hassanzadeh, and Subramanian train ANNs, LSTMs and ESNs on a multi-scale three-tier extension of the Lorenz 96 system. The RNNs are single-layered, and for the ESN they perform a study on the effect of ρ and reservoir dimension on the prediction horizon. The main resultant plot is presented in Figure 5.2, which shows that the ESN performs the best, with the LSTM trailing behind it and the ANN performing the worst (as mentioned previously). They make an interesting observation regarding the dimension of the ESN's reservoir, that there does seem to exist a point of diminishing returns where increasing the reservoir size leads to huge increases in computation time but only a marginal increase in the prediction horizon.

Wan et al. present another approach in [75] wherein they first construct a reduced order model using POD, and a new set of update equations for the reduced space using a flat Galerkin projection. The relevant equations are given below:

$$\boldsymbol{u} = \underline{\boldsymbol{Y}}\boldsymbol{\xi} + \underline{\boldsymbol{Z}}\boldsymbol{\eta} + \boldsymbol{b} \tag{5.8}$$

$$\frac{\mathrm{d}\boldsymbol{\xi}}{\mathrm{d}t} = \boldsymbol{F}_{\xi}(\boldsymbol{\xi}) + \boldsymbol{G}(\boldsymbol{\xi}, \boldsymbol{\eta}) \tag{5.9}$$

where $\boldsymbol{u}$ is the original state vector, the columns of $\underline{\boldsymbol{Y}}$ form an orthonormal basis for an n_m dimensional subspace Y of $\mathbb{R}^{n_u}$ and the columns of $\underline{\boldsymbol{Z}}$ form an orthonormal basis for $Z \in \mathbb{R}^{n_u - n_m}$ - the orthogonal complement of Y in $\mathbb{R}^{n_u}$. $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ are the projection vectors associated with $\underline{\boldsymbol{Y}}$ and $\underline{\boldsymbol{Z}}$, $\boldsymbol{F}_{\xi}(\boldsymbol{\xi})$ is the linear projection of the update equation for $\boldsymbol{u}$ onto the subspace Y (the same as described in Equation 5.7). $\boldsymbol{G}(\boldsymbol{\xi}, \boldsymbol{\eta})$ is the non-linear interaction between $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$ that is neglected in a flat/linear Galerkin projection, and is often the cause of the loss in dynamical characteristics in such an approach. Wan et al. model $\boldsymbol{G}(\boldsymbol{\xi}, \boldsymbol{\eta})$ as a function of $\boldsymbol{\xi}$ (at the present and previous time instances) only, cutting out the need to model $\boldsymbol{\eta}$. They use an LSTM to model $\hat{\boldsymbol{G}}(\boldsymbol{\xi}) \approx \boldsymbol{G}(\boldsymbol{\xi}, \boldsymbol{\eta})$, add it to $\boldsymbol{F}_{\xi}(\boldsymbol{\xi})$ and integrate the resulting time-derivative to obtain $\boldsymbol{\xi}$ at the next time step[3]. They also detail a multi-staged multi-level training strategy for the LSTM, wherein the first stage is teacher-forced and the second stage is trained auto-regressively using progressively increasing output time-steps. This is one of the only auto-regressive training strategies described in any chaotic-modelling literature, in addition to being quite straight-forward to implement. These modelling methods are applied to the CDV system reduced to 5 dimensions and a Kolmogorov flow reduced to just 3 wave-number modes. The obtained results showed that the LSTM aided models consistently outperforming the unaided flat Galerkin projections. However, it is important to note that the method described so far is not model-free, since the underlying system equation is used to construct $\boldsymbol{F}_{\xi}(\boldsymbol{\xi})$. The authors also describe a model-free method, which can be arrived at by simply allowing the network to approximate the entire RHS of Equation 5.9. This method performs worse than both the data-assisted and the flat Galerkin method for the CDV case (where the reduced space actually captures a lot of statistical variation), but outperforms the flat Galerkin approach in the case of the Kolmogorov flow (where the reduced space does not capture as much of the system's dynamics).

Vlachas et al. combine the superior short-term predictions of an LSTM with more traditional data-based approaches such as Gaussian Process Regression (GPR) and Mean Stochastic Modelling (MSM) in [73]. Here too, like in [75], the target of the learning process is not the state $\boldsymbol{z}$ itself but rather $\dot{\boldsymbol{z}}$, which is then integrated to arrive at the updated $\boldsymbol{z}$. They present a mixed algorithm, where first the probability density function of the data is estimated using a mixture of Gaussian kernels. Then given a data-point, if it is determined to lie in a region of high probability the LSTM is used otherwise the MSM model is used. The reasoning behind this approach is that regions of low probability lie

[3] In spirit, this approach is similar to how the feed-forward ANNs are used to predict $\Delta\boldsymbol{u}$ in [13] instead of $\boldsymbol{u}$ itself ($\mathrm{d}\boldsymbol{\xi}/\mathrm{d}t$ can be viewed to be $\Delta\boldsymbol{\xi}/\Delta t$ with $1/\Delta t$ being a simple scaling factor).

far away from typical attractor orbits and the LSTM would have trouble modelling the dynamics in these regions, whereas an MSM model will at least produce outputs that will bear similar statistics to the actual data. A similar mixed model is also created for the GPR. These are applied to reduced order models of the Lorenz 96 system, the KS equation and a realistic barotropic climate model. An RMSE plot for the methods applied to the Lorenz 96 system is presented in Figure 5.3. The results are representative of all three cases, and show the LSTM to possess superior short-term performance but quickly diverging for large time periods (owing to the accumulation and propagation of errors during prediction and integrations). Other models perform reasonably well with the mixed LSTM-MSM having the best short-term prediction accuracy and closest long-term statistics.

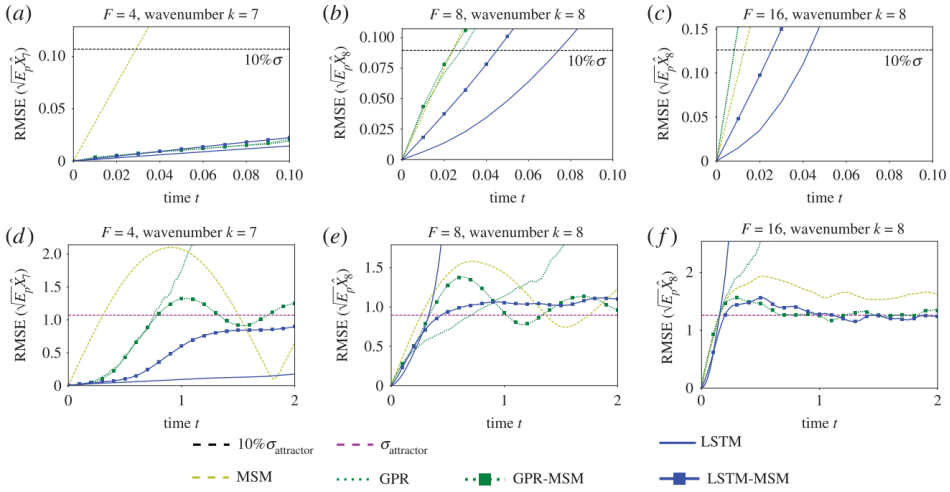


Figure 5.3: Time evolution of the normalized RMSE of the Lorenz 96 system, for different prediction methods. [73]

Pathak et al. use ESNs to model the Lorenz 63 system, the homogeneous and the in-homogeneous KS equation in [58]. They note the importance of accurately selecting the ESN's key parameters - the reservoir's spectral radius, the input scaling and the leaking coefficient. Their trained networks perform quite well on the data, and are able to successfully recreate the Lyapunov spectra of these systems as well. Further, they test the effect of adding noise to the input signal for the case of the KS system, and find that the trained ESN generalize to noisy input data reasonably well.

Pathak et al., in [57], build upon their earlier work in [58] to advance chaos prediction using reservoir computing. Two important improvements are presented here. The first is the introduction of a non-linearity in the ESN architecture, right before the final matrix multiplication. Instead of feeding the updated reservoir state $\boldsymbol{r}^{(t)}$ into the output linear map $\underline{\boldsymbol{W}}_{out}$, they pass in $\hat{\boldsymbol{r}}^{(t)}$ which is a non-linear function of $\boldsymbol{r}^{(t)}$. They've explored three different nonlinearities in the appendix, and the one that works best is to simply square each evenly-indexed element of $\boldsymbol{r}^{(t)}$. This breaks the symmetry of the reservoir, which is important since not all problems (and especially the in-homogeneous KS equation they test on) admit this symmetry. This approach produces ESNs with low errors over long good prediction horizons. Their second contribution stems from the observation that the cost of training and performing inference on very large reservoirs grows quite rapidly with the reservoir size. The reservoir size needs to grow with increase in the number of grid-points (either due to a finer resolution requirement or a larger domain size or both) in order to have reasonable predictive power. Thus, it would seem ESNs are not too viable for large domain sizes, from a computational efficiency point of view. Their solution is to split the entire spatial domain into smaller regions, each with its own moderately sized reservoir performing predictions for its local domain (as shown in Figure 5.4). This takes advantage of the local nature of interactions even in large dynamical systems. This approach is similar to the operating principle of a CNN. They investigate the effects of sharing the reservoirs and the output weights, as opposed to using different reservoirs and differently trained output weights. They report that the shared reservoirs/weights method performs as well as the different reservoirs/weights method on the homogeneous KS equation. However, when applied to the in-homogeneous case, the different reservoirs/weights method turns out to be superior (which seems intuitive enough as well). It

is worthwhile to note here that while this method is quite successful at modelling large systems at a reduced cost, it can only be applied in cases where local interactions play a strong role in determining the dynamics of a system.

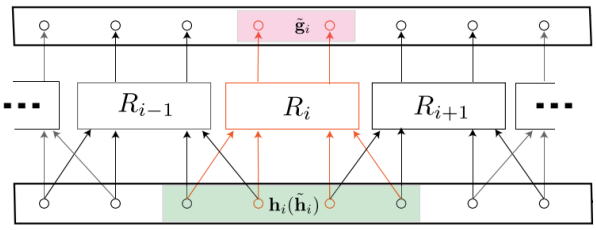


Figure 5.4: Parallelized scheme of moderately sized reservoirs operating on their own local regions. [57]

Vlachas et al. perform a comparative study of RC and RNN approaches to chaos prediction in [74]. They compare the predictive performance, RAM utilisation and training times of ESNs (trained with simple linear regression), LSTMs, GRUs and Unitary RNN (all trained with teacher-forced BPTT). Further, they also extend the BPTT-trained RNN architectures to multiple residual layers. In the first part of this study, the Lorenz 96 system is reduced to a set of observables using POD, and then the ESN and other RNN models applied to these reduced spaces. Here, they find that the ESNs struggle to perform accurate predictions, whereas the BPTT-trained RNN models perform relatively better. However, all the models, from an absolute standpoint, possess poor prediction horizons (none crossing even a single Lyapunov time interval). In the second part of this study, the idea of a large number of parallel (smaller) RNN/RC models being applied to data with strong local interactions from [57] is utilized. The test used here are the Lorenz 96 and the KS system. It is found when predicting the full order dynamics, the ESNs fare much better than the BPTT-trained RNNs with much lower training times and memory usage requirements. Even from an absolute value standpoint, all the networks have excellent predictive performances, with prediction horizons ranging anywhere from 3.5 to 4 Lyapunov time intervals. Lastly, to check whether the trained models are actual surrogates of the modelled systems, the power spectra and Lyapunov spectra are computed. Both of these match the original systems' respective spectra within reasonable error rates, with the Lyapunov spectra diverging (slightly) only for the negative values. It is indeed the positive Lyapunov coefficients that determine a system's chaotic properties, and hence this result can be interpreted to mean that the trained models are accurate surrogates of the original systems' dynamics.

5.2.2. Multi-Model Prediction

The common thread running through multi-model predictors is that they all use some form of an auto-encoder to reduce the dimensionality of some high fidelity data (in a non-linear manner), and then a 'reasoner' network (ESN, RNN, et cetera) to advance that reduced order representation in time. Some key studies illustrating the important principles of using combined-models of this nature are summarised in this section.

Doan, Polifke, and Magri present a deep learning based ROM approach to learning turbulent features in [19], which uses a Convolutional Auto-Encoder (CAE) to produce a reduced order representation of the Kolmogorov flow and an ESN to model the temporal propagation of this compressed representation. Their approach differs from the ROMs described so far in that after the initial separate training of the CAE and ESN, a final 'enmeshing' of the two networks is performed wherein the networks are trained jointly. This has the effect of accelerating the training, as compared to having the joint network train from scratch. While no comments are made on the prediction horizon of the trained network, owing to the chaotic nature of the system, they provide analyses of the statistics of the generated from the joint network when run auto-regressively. These statistics, such as the time-averaged vorticity and x and y direction velocity profiles are provided, and they bear good resemblance to the actual data. Further, an error analysis is presented which shows the normalized RMSE to be less than 6% for the various quantities. This implies that at least in a statistical sense (for the first-moment) the network succeeds in learning the underlying dynamics of modelled system. The approach presented here, of separately training the two networks and then a final enmeshing, is important for DL based ROMs since the combined training of two large models is indeed tedious and time-consuming.

Wu et al. in [78] present a similar convolutional auto-encoder + LSTM approach as [19], and apply it to the classical case of a flow past a circular cylinder. The key improvement here is that they also incorporate a self-attention[4] mechanism into the CAE itself, enabling the convolutional layers to incorporate global information as well. They compare their new model against a traditional convolution based auto-encoder and a simple MLP based auto-encoder, and find the incorporation of self-attention to be quite beneficial. Figure 5.5 shows a plot of the absolute errors at different time-steps for the compared ROMs. As can be observed, the self-attention based model has consistently low error profiles. This study proves that the addition of self-attention is a valuable component in a CNN based ROM, which can add essential global information that the CNN overlooks, and can be added at low computational cost. They also perform a study on the feasibility of transfer learning within this context, by transferring the weights learned for this problem to another network and training it on a separate problem of flow past a series of spoilers. They note that the training time of the new network is greatly reduced when using transfer learning, as opposed to starting the learning procedure from scratch. This points to the conclusion that their original model learns some universal features of the dynamics of the underlying flow itself, as opposed to it 'over-fitting' (in a sense) to the cylinder problem.

[4] Self-attention, and attention in general, is the key DL breakthrough behind recent successes such as transformers, Google's BERT model, and (the recently popularised) GPT models from OpenAI.

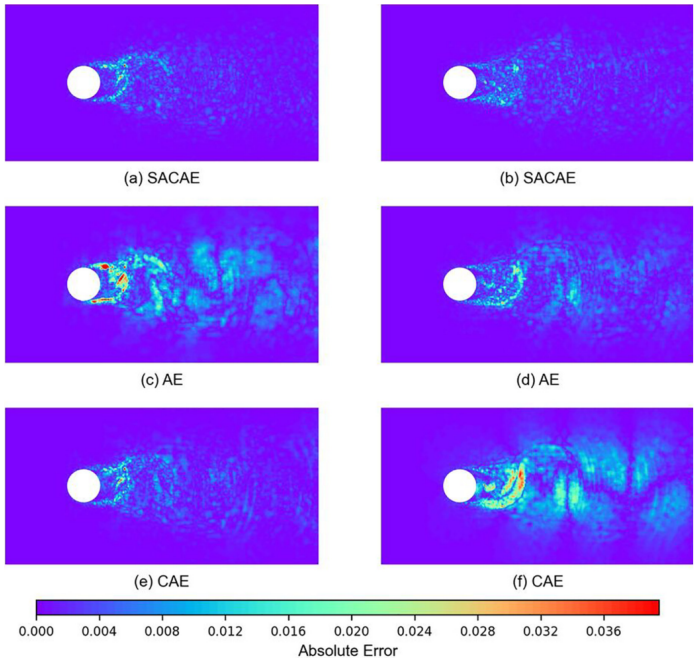


Figure 5.5: Absolute error in the velocity fields between the true and predicted data at the 199^{th} (left column) and 399^{th} (right column) time-steps, having been fed only 10 time-steps as the input history. [78]

Mohan et al. in [55] use a similar CAE and LSTM architecture as discussed above, to model 3D turbulence. Their key contribution here is to use a Convolutional-LSTM (C-LSTM), instead of a regular one. They note that most chaotic phenomena (turbulence included) are *spatio*-temporal phenomena, and posses strong local interactions. In the cases described previously, while the CAE is able to leverage these local-interactions in building a reduced order representation, its output is then flattened before being fed into the LSTM. This has the disadvantage of losing any spatial structure captured in the compressed representation, which as mentioned might be uniquely important in capturing dynamical phenomena. A C-LSTM is an extension of the regular LSTM cell architecture, where all the matrix multiplications involved in an LSTM cell (as described in subsection 4.3.2) are replaced with convolution operations. This has the effect of both preserving the spatial structure of the compressed representation and using the local interactions captured in it. An additional advantage is the parameter-sharing that the use of convolutional kernels in the LSTM cell bring with them, namely the reduction in the number of parameters to be learned. This makes the training of DL based ROMs of spatially large data-sets more tractable. They apply this model to two test cases - homogeneous isotropic turbulence in a box with and without additional forcing passive scalars. In both cases, they analyse the statistics of the predictions and compare them to the those of the original DNS data. Excellent agreement is found between the predicted case and the original data both in the energy spectra and the velocity gradient PDFs. Only a small variance is found for the high wave-number regions, corresponding to small scale phenomena, which was in fact predicted by the authors in an analysis of the design of the CAE. Overall, the authors present a promising approach using a natural extension of the LSTM cell into the spatial domain.

5.2.3. Some Other Considerations

This section describes additional techniques that can be used to construct better models for deep learning based ROMs.

With the exception of [74], all the studies mentioned so far use single RNN/ESN networks. The advantage of using stacked RNN layers has been known for quite some time now, and further using residual layers (as introduced in [36]) as opposed to simple stacked layers has the advantage of letting the networks learn a small change in contrast to a complete transformation. This smaller problem can be thought of as 'less challenging' to learn, and hence residual networks tend to possess higher accuracy and shorter training times. These residual networks bear a strong similarity to time-stepping schemes, which has led researchers to incorporate connection-maps inspired from classical numerical methods such as Runge-Kutta methods and other time-stepping schemes into deep learning architectures. Some representative studies of this concept can be found in [63, 59, 83], all of which use different types of ANNs to approximate the time-derivative and then incorporate this into a multi-step time stepping scheme. A common thread running through their results seems to be the accumulation of error which eventually leads to the predictions diverging from the actual system dynamics. It is also worth noting that these systems use the full network to model the time derivative, whereas the residual network concept is to use additional layers stacked on top of the RNN layer itself, before passing the combined output through a dense layer. However, such an approach has to date not been used in the context of multi-step methods.

Lastly, an important study was performed by Wu et al. in [77], wherein the authors trained a series of GANs to generate physically accurate solutions to a Rayleigh-Bénard convection system. The GAN architecture is quite complex, and is not detailed here because it is out of the scope of present work. Their key contribution is the enforcing of what they call 'statistics-informed' penalties. This involves introducing an additionally penalty in the cost function while training the networks, which is simply the Frobenius norm of the difference in the covariance matrices of the predicted data and that of the training data. It should be noted that this method is still model-free since no information about the model's parameters or structure is used. Simply the first moment of the training data is being used to introduce an additional penalty. This simple yet powerful concept has a major impact on their trained GANs, and the networks trained with this penalty not only have more physically accurate solutions but also converge faster when compared to networks trained without this penalty term. They also note that the imposition of this first moment penalty leads to the predicted solutions having more accurate higher moments. Of equal importance is the fact that this penalty is both cheap to compute and to back-propagate, hence adding huge improvements at negligible cost.

5.3. Summary

This chapter outlines the various state-of-the-art methods using machine learning to predict the time-evolution of chaotic systems. As can be observed, different models can be combined in various architectures (single-layered versus multi-layered versus residual networks). Further, there exists a great variance in training strategies as well as a general mismatch in the training and testing objectives of the RNNs. Clear from these, is the fact that there exist no standard comparisons of the different machine learning models, and no standard metric to compare against either (since the mentioned prediction horizons are all computed differently in different studies). Neither do any of the mentioned studies attempt to model multiple dynamical regimes, or test RK-methods in an actual residual setting.

PART III
Methodology

6

Chaotic Systems Selected for Investigation

This chapter presents the different chaotic systems that have been selected for investigation, and provides justification for the inclusion of each. Section 6.1 describes the Lorenz'63 system, Section 6.2 the Charney-DeVore truncated set of equations, Section 6.3 the Kuramoto-Sivashinsky equation and finally Section 6.4 presents the equations describing the Kolmogorov flow. Loosely speaking, these systems increase in their complexity (in terms of analysis and computation), with the first two being Ordinary Differential Equations (ODEs) and the final two being Partial Differential Equations (PDEs). Further the Lorenz '63 system has only three dimensions while the CDV system has 6 and displays two different types of behaviours in its natural evolution. The KS equation is one-dimensional whereas the Kolmogorov flow is two dimensional in space and hence presents other challenges in the construction of its ML models. These systems have been presented and investigated in the mentioned order so as to build up intuition and models along the way.

6.1. The Lorenz '63 System

The Lorenz '63 system of equations was proposed by Edward Lorenz in 1963 [49] as a simplified system exhibiting what he termed 'nonperiodic deterministic flow'. It is a simplified solution of a convective flow occurring in a layer of fluid of uniform depth with a constant temperature difference (ΔT) held between its two boundaries. The flow is effectively two dimensional since no variation is assumed to be present in the direction orthogonal to the depth and parallel to the boundaries (in the standard coordinate system, if the direction of depth is taken to be the z axis and the direction of flow the x axis, then this would mean no variation in the y direction). This has often been viewed as a simplified 'cell' (Figure 6.1) for the study of atmospheric behavior. The equations are a set of ordinary differential equations and are presented below:

$$\dot{x_1} = -\sigma x_1 + \sigma x_2 \tag{6.1}$$

$$\dot{x_2} = -x_1 x_3 + \rho x_1 - x_2 \tag{6.2}$$

$$\dot{x_3} = x_1 x_2 - \beta x_3 \tag{6.3}$$

Here $\sigma = \nu/\kappa$ (ν being the kinematic viscosity and κ the thermal conductivity) denotes the Prandtl number, ρ is the ratio between the Rayleigh number ($R_a = g\alpha H^3 \Delta T \nu^{-1}\kappa^{-1}$, g is the acceleration due to gravity and α the thermal expansion coefficient) and the critical Rayleigh number ($R_c = \pi^4 a^{-2}(1+a^2)^3$, $a = H/L$ is the ratio of the geometric lengths of the cell). The parameter b depends purely on the geometry of the cell and is given by $\beta = 4(1+a^2)^{-1}$. Further, the time-derivatives on the LHS are themselves with respect to a non-dimensional time $\tau = \pi^2 H^{-2}(1+a^2)\kappa t$.

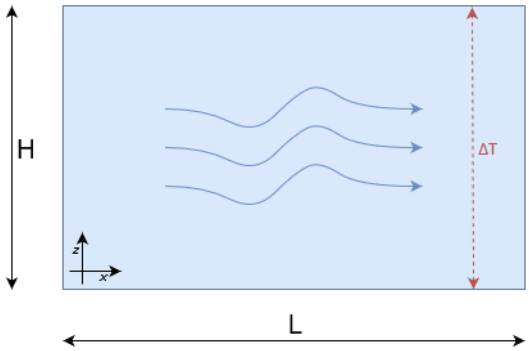


Figure 6.1: Typical Lorenz cell.

For certain values of the coefficients ρ, σ and β, the above set of equations displays unstable behaviour. The detailed analysis of the critical values for which this behaviour can be observed is provided in [49], and it states that for $\sigma > \beta + 1$ there exists a critical value of ρ and for sufficiently high values the steady convective flow described by these equations is unstable. Canonical values used to demonstrate this property are $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$. The time evolution of a solution trajectory in the phase space is plotted in Figure 6.2 and the time evolution of the individual components given in Figure 6.3. Note that in Figure 6.2 only the strange attractors are plotted. There is another attractor at $x_1 = 0, x_2 = 0, x_3 = 0$ however this is not a strange attractor.

This is a fairly simple set of equations displaying intermittency and chaotic behaviour. As such, it is an ideal starting point for testing out different modelling techniques.

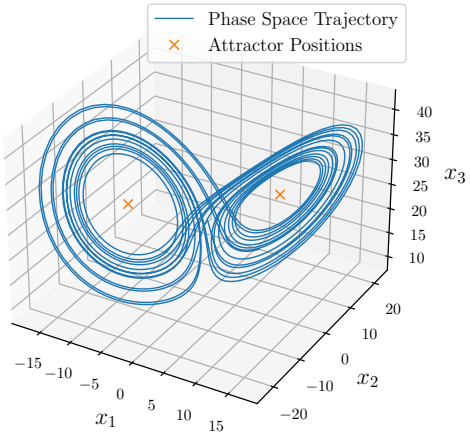


Figure 6.2: Time evolution of the solution (integrated using the RK4 method with $\Delta t = 0.01$) in the phase space.

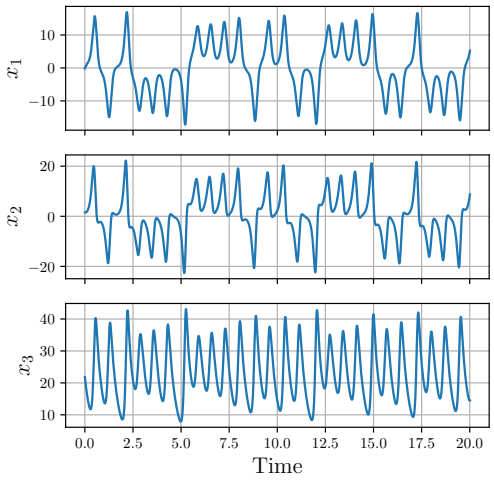


Figure 6.3: Time evolution of the individual components (integrated using the RK4 method with $\Delta t = 0.01$).

6.2. The Charney-DeVore System

The Charney-DeVore (henceforth referred to as CDV) system of equations was first proposed in 1979 by Jule Charney and John DeVore [12] as a reduced model to study the behaviour of planetary-scale zonal atmospheric motion in a barotropic environment. They present an equation for the conservation of 'potential vorticity' on a β-plane[1] and then project it onto the eigenfuncions of the Δ operator. The ensuing spectral expansion is truncated to just six terms, whose coefficients are time-varying and to be computed. However, these coefficients are complex in nature and hence after a suitable transformation the final set of ODEs (with variables in the real space) are arrived at.

The equations presented here are scaled versions [18] of the original set of ODEs proposed by Charney and DeVore and have lately become a standard set used in testing predictive models [75, 20]:

$$\dot{x}_1 = \tilde{\gamma}_1 x_3 - C(x_1 - x_1^*) \tag{6.4}$$
$$\dot{x}_2 = -(\alpha_1 x_1 - \beta_1)x_3 - Cx_2 - \delta_1 x_4 x_6 \tag{6.5}$$
$$\dot{x}_3 = (\alpha_1 x_1 - \beta_1)x_2 - \gamma_1 x_1 - Cx_3 + \delta_1 x_4 x_5 \tag{6.6}$$
$$\dot{x}_4 = \tilde{\gamma}_2 x_6 - C(x_4 - x_4^*) + \epsilon(x_2 x_6 - x_3 x_5) \tag{6.7}$$
$$\dot{x}_5 = -(\alpha_2 x_1 - \beta_2)x_6 - Cx_5 - \delta_2 x_4 x_3 \tag{6.8}$$
$$\dot{x}_6 = (\alpha_2 x_1 - \beta_2)x_5 - \gamma_2 x_4 - Cx_6 + \delta_2 x_4 x_2 \tag{6.9}$$

The various coefficients used above are computed as:

$$\alpha_m = \frac{8\sqrt{2}}{\pi}\frac{m^2}{4m^2-1}\frac{b^2+m^2-1}{b^2+m^2} \qquad \tilde{\gamma}_m = \gamma\frac{4m}{4m^2-1}\frac{\sqrt{2}b}{\pi}$$
$$\beta_m = \frac{\beta b^2}{b^2+m^2} \qquad \epsilon = \frac{16\sqrt{2}}{5\pi}$$
$$\delta_m = \frac{64\sqrt{2}}{15\pi}\frac{b^2-m^2+1}{b^2+m^2} \qquad \gamma_m = \gamma\frac{4m^3}{4m^2-1}\frac{\sqrt{2}b}{\pi(b^2+m^2)}$$

for $m = 1, 2$. This just leaves the six constants x_1^*, x_4^*, C, β, γ and b. An extensive study on the stability and various kinds of equilibrium points that develop depending on the values of these six constants is provided in [18]. For the purpose of testing predictive models, the values[2] $x_1^* = 0.95$, $x_4^* = -0.76095$, $C = 0.1$, $\beta = 1.25$, $\gamma = 0.2$ and $b = 0.5$ are suitable since they result in the model intermittently alternating between two distinct regimes of behavior, as shown in Figure 6.6. Further, the two flow regimes - *zonal* and *blocked* - can also be seen occupying distinct regions when viewed in the x_1-x_4 plane, as shown in Figure 6.5.

[1] A β-plane is an approximation of a patch of atmosphere, and is taken to be a plane tangent to the earth's surface (as shown in Figure 6.4). The plane is bounded in the x (u-axis in Figure 6.4) and y (v-axis in Figure 6.4) directions. On this plane the Coriolis coefficient f is approximated by a linear function of the latitude (ϕ), which is what characterises this tangent plane patch as a β-plane.

[2] This choice of the model parameters results in each non-dimensional time unit corresponding to roughly one day.

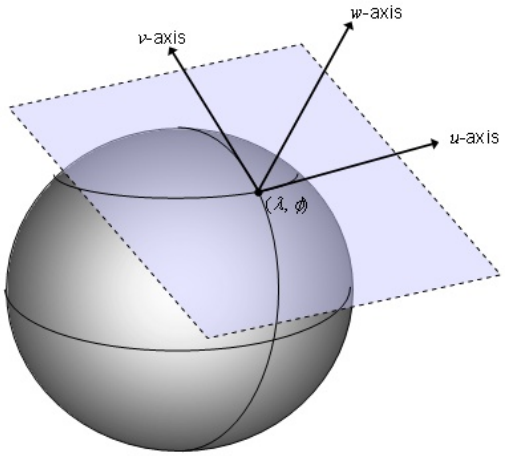


Figure 6.4: A plane tangent to a sphere [69].

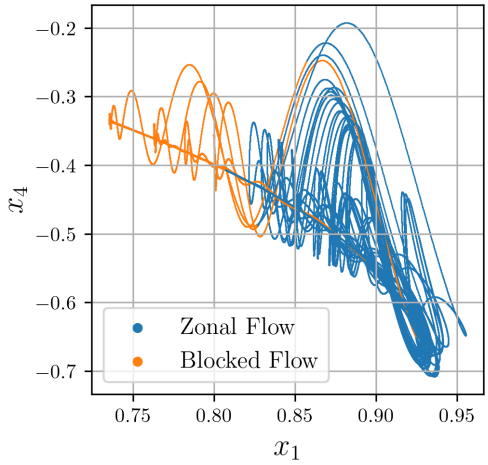


Figure 6.5: 150,000 points sampled from the CDV attractor, projected to the x_1-x_4 plane. The different flow regimes occupy distinct spaces.

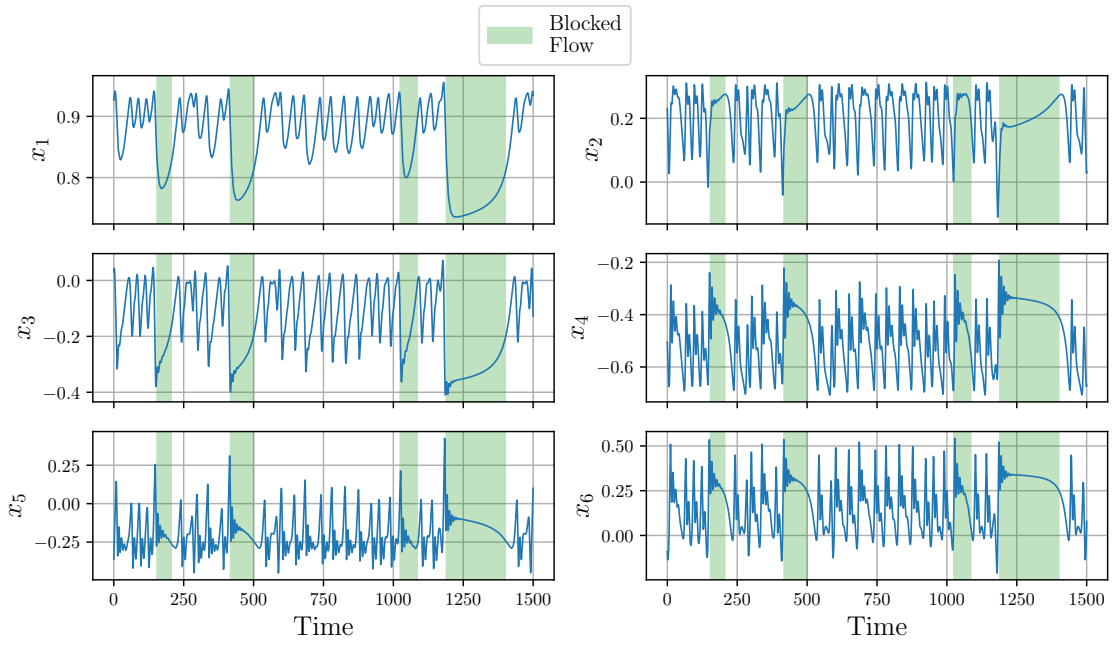


Figure 6.6: Time evolution of all the state variables (integrated using the RK4 method with $\Delta t = 0.1$), with the blocked flow regime clearly highlighted.

6.3. Kuramoto-Sivashinsky System

The Kuramoto-Sivashinsky (henceforth referred to as KS) equation is a non-linear PDE of the fourth order. It was arrived at independently by Yoshiki Kuramoto [45] and Gregory Sivashinsky [66] in the late seventies, in completely different research contexts. Kuramoto was researching the behaviour of point concentrations in a turbulent reaction-diffusion system (such as the Brusselator) while Sivashinsky was researching the time evolution of a laminar flame front. It is yet another example of how two different physical systems can be described by similar mathematics.

It describes a system that is well known to be prone to chaos, and the 1D KS equation is given below:

$$\frac{\partial u}{\partial t} = -\nu_1 u \frac{\partial u}{\partial x} - \nu_2 \frac{\partial^2 u}{\partial x^2} - \nu_3 \frac{\partial^4 u}{\partial x^4} + G(u, x, t) \tag{6.10}$$

where ν_1, ν_2, ν_3 are constants, all typically set equal to 1 and $G(u, x, t)$ is an additional forcing term which is canonically set to zero. The spatial domain $x \in [0, L]$ and the scalar field $u(x, t)$ is considered to be periodic within this domain. The domain size L plays an important role in determining the behavior of the system since the maximal Lyapunov exponent depends directly upon it [24].

Considering the solution to be periodic and taking the Fourier transform of the equation yields:

$$\frac{\mathrm{d}\hat{u}_k}{\mathrm{d}t} = -\nu_1 \mathcal{F}\left[u \frac{\partial u}{\partial x}\right](k) + \left(\nu_2 k^2 - \nu_3 k^4\right) \hat{u}_k + \mathcal{F}[G(u, x, t)](k) \quad k \in \frac{2\pi m}{L}, m \in \mathbb{Z} \tag{6.11}$$

where $\mathcal{F}$ denotes the Fourier transform, $i = \sqrt{-1}$ and k is the wave number. The non-linear advection term is typically transformed before taking the Fourier transform as follows:

$$u \frac{\partial u}{\partial x} = \frac{1}{2} \frac{\partial u^2}{\partial x}$$
$$\implies \mathcal{F}\left[u \frac{\partial u}{\partial x}\right](k) = \frac{ik}{2} \mathcal{F}\left[u^2\right](k)$$

This gives:

$$\frac{\mathrm{d}\hat{u}_k}{\mathrm{d}t} = -\frac{ik\nu_1}{2} \mathcal{F}\left[u^2\right](k) + \left(\nu_2 k^2 - \nu_3 k^4\right) \hat{u}_k + \mathcal{F}[G(u, x, t)](k), \quad k = \frac{2\pi m}{L}, m \in \mathbb{Z} \tag{6.12}$$

It is clear from Equation 6.12 that for positive values of ν_2 and ν_3 the second derivative term acts as an energy source whereas the fourth derivative term acts as an energy sink. The advection term appears to re-distribute energy between different wave numbers. All of these combined lead to the spatio-temporal chaotic behaviour that this equation is know for. Further, for the typical values $\nu_1 = \nu_2 = \nu_3 = 1$ and $G = 0$ it can be seen that the smaller wave numbers (corresponding to large wavelengths) have energy added to them whereas the higher wave numbers (corresponding to small wavelengths) have energy drained from them. Figure 6.7 shows the time evolution of a KS system with $\nu_1 = \nu_2 = \nu_3 = 1$, $G = 0$ and $L = 35$.

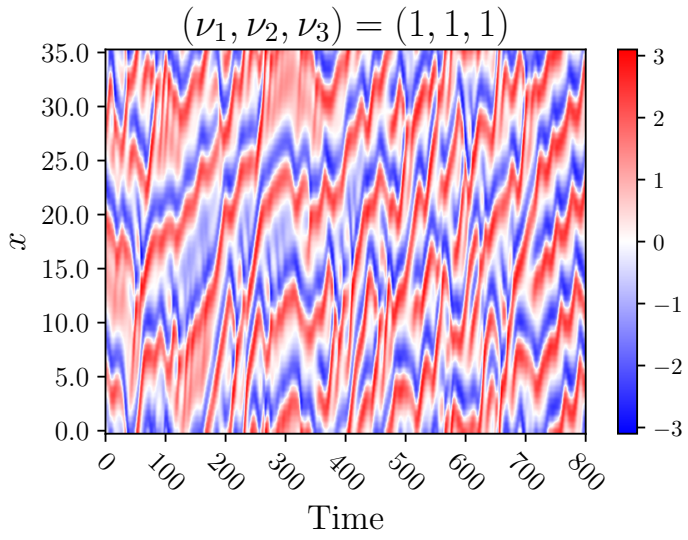


Figure 6.7: Time evolution of KS equation, with $L = 35$ (integrated using the ETDRK4 method with 100 gridpoints and $\Delta t = 0.1$).

6.4. Kolmogorov Flow

Consider the Navier-Stokes equations for incompressible flow in two dimensions:

$$\partial_t \boldsymbol{u} = -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nabla p + \frac{1}{Re}\Delta \boldsymbol{u} + \boldsymbol{f} \tag{6.13}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{6.14}$$

where $\boldsymbol{u} = \begin{pmatrix} u_x & u_y \end{pmatrix}^T$ is the velocity field of the fluid, over the domain $(x, y) \in \Omega = [0, 2\pi] \times [0, 2\pi]$, p is the pressure field over the same spatial domain and $\boldsymbol{f}$ is a volume forcing term. Re denotes the Reynolds number and the non-dimensional viscosity ν is related to it as $\nu = 1/Re$. Periodic boundary conditions are imposed on $\boldsymbol{u}$ and the forcing function is taken to be of the form $\boldsymbol{f} = \begin{pmatrix} \sin(k_f y) & 0 \end{pmatrix}^T$. The solution to this particular boundary value problem is known as the *2D Kolmogorov Flow* [75, 19, 60, 26]. This particular formulation was introduced by Andrey Kolmogorov in a seminar as an example of a simple linear instability problem.

This problem admits a laminar solution $\boldsymbol{u} = \begin{pmatrix} Re\, k_f^{-2} \sin(k_f y) & 0 \end{pmatrix}^T$ [19, 75]. The solution grows unstable and chaotic for large values of k_f and Re. Typical contours of u_x, u_y and the vorticity are shown in Figure 6.8, Figure 6.9 and Figure 6.10.

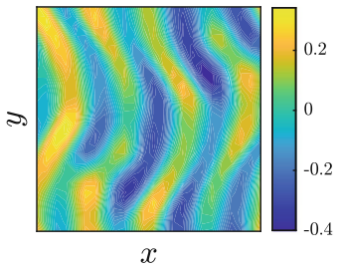


Figure 6.8: Iso-contours of u_x, $Re = 30$ [19].

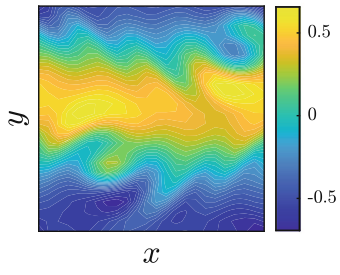


Figure 6.9: Iso-contours of u_y, $Re = 30$ [19].

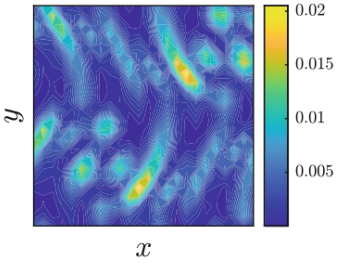


Figure 6.10: Iso-contours of the vorticity, $Re = 30$ [19].

The three statistical quantities of interest are the kinetic energy (E), the dissipation (D) and the

energy input (I):

$$E(\boldsymbol{u}) = \frac{1}{|\Omega|}\int_\Omega \frac{1}{2}|\boldsymbol{u}|^2 \mathrm{d}\Omega$$
$$D(\boldsymbol{u}) = \frac{Re^{-1}}{|\Omega|}\int_\Omega |\nabla \boldsymbol{u}|^2 \mathrm{d}\Omega$$
$$I(\boldsymbol{u}) = \frac{1}{|\Omega|}\int_\Omega \boldsymbol{u}\cdot\boldsymbol{f}\ \mathrm{d}\Omega$$

where $|\nabla \boldsymbol{u}|$ is the Frobenius norm of the Jacobian of $\boldsymbol{u}$ and $|\Omega| = (2\pi)^2$ is the area of the domain. It is worth noting that the statistical quantities mentioned above, in the context of Kolmogorov flow (i.e. incompressible flow with periodic boundary conditions), obey the relationship $\dot{E} = I - D$ and:

$$\begin{aligned} D(\boldsymbol{u}) &= \frac{Re^{-1}}{|\Omega|}\int_\Omega |\nabla \boldsymbol{u}|^2 \mathrm{d}\Omega \\ &= \frac{Re^{-1}}{|\Omega|}\int_\Omega \omega^2\ \mathrm{d}\Omega, \end{aligned} \tag{6.15}$$

with $\omega = (\nabla \times \boldsymbol{u})\cdot\hat{\boldsymbol{e}}_3$ being the only non-zero component of the vorticity.

Owing to the periodicity of $\boldsymbol{u}$, it can be easily analysed in the Fourier domain, and its Fourier expansion can be written as [75, 60, 26]:

$$\boldsymbol{u}(\boldsymbol{x},t) = \sum_{\boldsymbol{k}} \frac{a(\boldsymbol{k},t)}{|\boldsymbol{k}|}\begin{pmatrix} k_2 \\ -k_1 \end{pmatrix} e^{i\boldsymbol{k}\cdot\boldsymbol{x}} \tag{6.16}$$

where $\boldsymbol{x} = \begin{pmatrix} x & y \end{pmatrix}^T$ is the spatial co-ordinate, $\boldsymbol{k} = \begin{pmatrix} k_1 & k_2 \end{pmatrix}^T$ is the wave number vector corresponding to the Fourier space and $a(\boldsymbol{k},t)$ is the Fourier coefficient ($k_1 \in \mathbb{Z}$, $k_2 \in \mathbb{Z}$ and $a \in \mathbb{C}$). Additionally, since $\boldsymbol{u}$ is real valued, $a(\boldsymbol{k},t) = -\overline{a(-\boldsymbol{k},t)}$. This particular formulation of the Fourier expansion (with a unified Fourier coefficient) is possible owing to the divergence-free nature of $\boldsymbol{u}$[3].

To find a suitable time-evolution equation for $\boldsymbol{a}(\boldsymbol{k},t)$ requires the removal of the dependence on the pressure term in Equation 6.13. This can be done by using the Leray projection operator $\mathcal{P}$. This is a pseudo-differential operator that (loosely speaking) takes the orthogonal projection of its argument onto a subspace (this subspace is not arbitrary and is itself used to define $\mathcal{P}$). In the present case of the Kolmogorov flow, $\mathcal{P}$ is taken to operate on vectors from the $\mathcal{L}^2$ space onto the subspace of divergence-free vectors [16]. Applying $\mathcal{P}$ to Equation 6.13 yields:

$$\begin{aligned} \mathcal{P}\left(\partial_t \boldsymbol{u}\right) &= -\mathcal{P}\left(\left(\boldsymbol{u}\cdot\nabla\right)\boldsymbol{u} - \nabla p + \frac{1}{Re}\Delta\boldsymbol{u} + \boldsymbol{f}\right) \\ \implies \partial_t \mathcal{P}\left(\boldsymbol{u}\right) &= -\mathcal{P}\left(\left(\boldsymbol{u}\cdot\nabla\right)\boldsymbol{u}\right) - \mathcal{P}\left(\nabla p\right) + \frac{1}{Re}\mathcal{P}\left(\Delta\boldsymbol{u}\right) + \mathcal{P}\left(\boldsymbol{f}\right) \end{aligned} \tag{6.17}$$

Since $\boldsymbol{u}$ is divergence-free (Equation 6.14), $\Delta\boldsymbol{u}$ is divergence-free ($\nabla\cdot(\Delta\boldsymbol{u}) = \Delta(\nabla\cdot\boldsymbol{u}) = 0$) and it can be shown that ∇p is orthogonal to any divergence-free vector, Equation 6.17 can be simplified as follows:

$$\partial_t \boldsymbol{u} = -\mathcal{P}\left(\left(\boldsymbol{u}\cdot\nabla\right)\boldsymbol{u} - \boldsymbol{f}\right) + \frac{1}{Re}\Delta\boldsymbol{u} \tag{6.18}$$

Taking a Fourier transform at this stage results in Equation 6.19 [75, 26].

$$\dot{a}(\boldsymbol{k},t) = \sum_{\boldsymbol{p}+\boldsymbol{q}=\boldsymbol{k}} i\frac{(p_1q_2 - p_2q_1)(k_1q_1 + k_2q_2)}{|\boldsymbol{p}||\boldsymbol{q}||\boldsymbol{k}|}a(\boldsymbol{p},t)a(\boldsymbol{q},t) - \frac{1}{Re}|\boldsymbol{k}|^2 a(\boldsymbol{k},t) - \frac{1}{2}i(\delta_{\boldsymbol{k},\boldsymbol{k}_f} + \delta_{\boldsymbol{k},-\boldsymbol{k}_f}) \tag{6.19}$$

[3]This can be derived by simply assuming separate Fourier expansions for u_x and u_y individually and then enforcing the divergence-free property.

It is easy to see from the above time-evolution equation that the Fourier mode with wave number $\boldsymbol{k}$ is impacted (in a non-linear manner) by any pair of modes with wave numbers $\boldsymbol{p}$ and $\boldsymbol{q}$ that satisfy the relationship $\boldsymbol{p} + \boldsymbol{q} = \boldsymbol{k}$. Such a group is called a *triad*. Additionally, note that even modes that do not form triads with $\boldsymbol{k}$ can affect it indirectly, by affecting the modes that do.

This type of flow is also prone to sharp intermittent bursts in kinetic energy and dissipation, and in [26] it is reported that the triad formed by the modes $(0, k_f)$, $(1, 0)$ and $(1, k_f)$ is the most influential to this phenomena. The time evolution of the dissipation and the absolute value of the mode corresponding to $(1, 0)$ is shown in Figure 6.11.

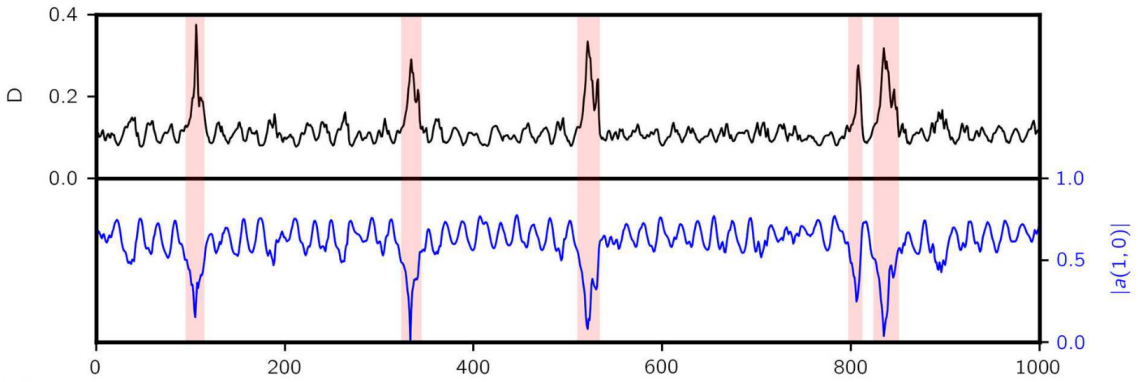


Figure 6.11: Time evolution of the dissipation and the absolute value of the mode $(1, 0)$, $Re = 40$ [75].

6.5. Summary

This chapter introduces four different chaotic systems, with increasing orders of complexity. This order and range is ideal in its choice of systems, since it allows the gradual build-up of complexity in both analysis methods and developed models. The final chosen system is a 2D PDE system, the Kolmogorov flow system, a fluid-dynamical system, and can be considered the final 'verification' system that all the developed models and techniques will be applied to.

7

Traditional Numerical Solvers

Data is an important aspect of any machine learning pipeline. As the saying goes, 'garbage in, garbage out'. Thus, to have proper data for the ML models to train on, and adequate control over its parameters, it is important to be able to accurately solve the selected systems and generate high quality data. The systems selected for investigation and their details are presented in Chapter 6. This chapter presents the various numerical methods used to achieve this.

7.1. Runge-Kutta Solvers

Runge-Kutta (henceforth referred to as RK) methods are a class of implicit and explicit iterative numerical solvers, used for the time-discretization of a set of ODEs [10]. They were introduced in the early twentieth century by Carl Runge and expanded upon by Wilhelm Kutta and others, and have been popular ever since - owing to their extremely simple iterative nature, high accuracy and ease of programming. Assuming an initial value problem as follows:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}\left(\boldsymbol{x}(t), t\right) \\ \boldsymbol{x}(t=0) &= \boldsymbol{x}_0 \end{aligned} \tag{7.1}$$

the general m-stage explicit RK method can be written down as:

$$\begin{aligned} \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n + \Delta t \sum_{i=1}^{m} b_i \boldsymbol{k}_i \\ (\boldsymbol{x}_n &= \boldsymbol{x}(t_n), \quad t_n = t_0 + n\Delta t) \end{aligned} \tag{7.2}$$

where:

$$\begin{aligned} \boldsymbol{k}_1 &= \boldsymbol{f}(\boldsymbol{x}_n, t_n) \\ \boldsymbol{k}_2 &= \boldsymbol{f}(\boldsymbol{x}_n + (a_{2,1}\boldsymbol{k}_1)\Delta t, t_n + c_2\Delta t) \\ &\vdots \\ \boldsymbol{k}_m &= \boldsymbol{f}(\boldsymbol{x}_n + (a_{m,1}\boldsymbol{k}_1 + a_{m,2}\boldsymbol{k}_2 + \cdots + a_{m,m-1}\boldsymbol{k}_{m-1})\Delta t, t_n + c_m\Delta t) \end{aligned}$$

The coefficients $a_{i,j}$, c_i and b_i ($i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, i-1\}$) determine a particular RK method, and are often visualized in the form of *Butcher tableau* (as in Table 7.1).

0					
c_2	$a_{2,1}$				
c_3	$a_{3,1}$	$a_{3,2}$			
$\vdots$	$\vdots$		$\ddots$		
c_m	$a_{m,1}$	$a_{m,2}$	$\cdots$	$a_{m,m-1}$	
	b_1	b_2	$\cdots$	b_{m-1}	b_m

Table 7.1: A typical Butcher tableau for an m-stage explicit RK method.

Further conditions are imposed upon the coefficients to ensure desirable properties. The method is consistent (i.e. $\lim_{\Delta t \to 0} \Delta t \phi(s, \Delta t) = 0$, where $\phi(s, \Delta t)$ is the local truncation error) if and only if $\sum_{i=1}^{m} b_i = 1$. To ensure the method is of order p, i.e. the local truncation error is of the order $O(\Delta t^{p+1})$, additional conditions can be derived from the Taylor series expansions of the various stages.

Of all these methods, of interest to this study is the widely popular RK4 method. It's Butcher tableau is given in Table 7.2. It is a 4^{th} order method with 4 stages, and the local truncation error is of the order $O(\Delta t^5)$.

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

Table 7.2: Butcher tableau for RK4 method.

7.2. Exponential Time Differencing with RK4

The exponential time differencing with RK4 (henceforth referred to as ETDRK4) scheme was put forth by Cox and Matthews [17] and is a time-stepping scheme designed expressly to deal with stiff ODEs. There is no formal definition of a stiff system of ODEs is, but commonly accepted descriptions include:

- a system where the time step is restricted not by the required accuracy but by the stability of the numerical method,
- a system of ODEs that describe a physical problem covering a wide range of time-scales, in effect requiring careful tuning of the time step for regular explicit time-stepping methods.

Often the spectral and pseudo-spectral methods that aim to analyse a PDE with spatially periodic boundary conditions in the Fourier domain result in a stiff system of ODEs [17], thus requiring the application of either explicit methods with extremely small time steps or implicit methods. Both these approaches end up being very computationally expensive. The method described here completely side-steps this issue by separating out the stiff linear part and the explicitly time-differenceable non-linear part of the equation. Consider a system of stiff ODEs, having decomposed it into its linear portion $\underline{\boldsymbol{L}}\boldsymbol{x}(t)$ ($\underline{\boldsymbol{L}}$ is a matrix) and non-linear portion $\boldsymbol{N}(\boldsymbol{x}(t))$:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \underline{\boldsymbol{L}}\boldsymbol{x}(t) + \boldsymbol{N}(\boldsymbol{x}(t)) \\ \boldsymbol{x}(t=0) &= \boldsymbol{x}_0 \end{aligned} \tag{7.3}$$

Integrating Equation 7.3 exactly yields:

$$\boldsymbol{x}(t) = e^{\underline{L}t}\boldsymbol{x}_0 + \int_0^t e^{\underline{L}(t-\tau)}\boldsymbol{N}(\boldsymbol{x}(\tau))\mathrm{d}\tau \tag{7.4}$$

Note that the first term here is easily computable and requires no special handling. The second term, i.e. the integral, is what needs to be approximated. Cox and Matthews approximate the integral using the RK4 time-stepping scheme, and the resulting method turns out to be extremely stable and extremely fast. However, their formulation is only applicable to cases with a diagonal $\underline{\boldsymbol{L}}$ matrix and is also prone to instability owing to a few singularities. These issues are dealt with by Kassam and Trefethen in [43] by

explicitly diagonalizing the $\underline{\boldsymbol{L}}$ matrix and using the residue theorem from complex analysis to side-step computing the singularity directly. This formulation is given below, but first Equation 7.4 needs to be reformulated into another form.

Changing the integration limits in Equation 7.4 to $[t_n, t_{n+1}]$ ($t_n = t_0 + n\Delta t$) results in:

$$\boldsymbol{x}_{n+1} = e^{\underline{\boldsymbol{L}}\Delta t}\boldsymbol{x}_n + \int_0^{\Delta t} e^{-\underline{\boldsymbol{L}}(\Delta t-\tau)}\boldsymbol{N}(\boldsymbol{x}(t_n+\tau))\mathrm{d}\tau \tag{7.5}$$

Now the final formulation from [43] is as follows:

$$\boldsymbol{x}_{n+1} = \underline{f_1}(\Delta t\underline{\boldsymbol{L}})\boldsymbol{x}_n + \Delta t\underline{\boldsymbol{\alpha}}(\Delta t\underline{\boldsymbol{L}})\boldsymbol{N}(\boldsymbol{x}_n) + \Delta t\underline{\boldsymbol{\beta}}(\Delta t\underline{\boldsymbol{L}})\left[\boldsymbol{N}(\boldsymbol{a}_n) + \boldsymbol{N}(\boldsymbol{b}_n)\right] + \Delta t\underline{\boldsymbol{\gamma}}(\Delta t\underline{\boldsymbol{L}})\boldsymbol{N}(\boldsymbol{c}_n) \tag{7.6}$$

where:

$$\begin{aligned}
\boldsymbol{a}_n &= \underline{f_1}\left(\frac{\Delta t\underline{\boldsymbol{L}}}{2}\right)\boldsymbol{x}_n + \frac{\Delta t}{2}\underline{f_2}\left(\frac{\Delta t\underline{\boldsymbol{L}}}{2}\right)\boldsymbol{N}(\boldsymbol{x}_n)\\
\boldsymbol{b}_n &= \underline{f_1}\left(\frac{\Delta t\underline{\boldsymbol{L}}}{2}\right)\boldsymbol{x}_n + \frac{\Delta t}{2}\underline{f_2}\left(\frac{\Delta t\underline{\boldsymbol{L}}}{2}\right)\boldsymbol{N}(\boldsymbol{a}_n)\\
\boldsymbol{c}_n &= \underline{f_1}\left(\frac{\Delta t\underline{\boldsymbol{L}}}{2}\right)\boldsymbol{a}_n + \frac{\Delta t}{2}\underline{f_2}\left(\frac{\Delta t\underline{\boldsymbol{L}}}{2}\right)\left[2\boldsymbol{N}(\boldsymbol{b}_n) - \boldsymbol{N}(\boldsymbol{x}_n)\right]
\end{aligned} \tag{7.7}$$

$$\begin{aligned}
\underline{\boldsymbol{\alpha}}(\Delta t\underline{\boldsymbol{L}}) &= \underline{f_2}(\Delta t\underline{\boldsymbol{L}}) - 3\underline{f_3}(\Delta t\underline{\boldsymbol{L}}) + 4\underline{f_4}(\Delta t\underline{\boldsymbol{L}})\\
\underline{\boldsymbol{\beta}}(\Delta t\underline{\boldsymbol{L}}) &= 2\underline{f_3}(\Delta t\underline{\boldsymbol{L}}) - 4\underline{f_4}(\Delta t\underline{\boldsymbol{L}})\\
\underline{\boldsymbol{\gamma}}(\Delta t\underline{\boldsymbol{L}}) &= -\underline{f_3}(\Delta t\underline{\boldsymbol{L}}) + 4\underline{f_4}(\Delta t\underline{\boldsymbol{L}})
\end{aligned} \tag{7.8}$$

$$\begin{aligned}
\underline{f_1}(\Delta t\underline{\boldsymbol{L}}) &= e^{\Delta t\underline{\boldsymbol{L}}}\\
\underline{f_2}(\Delta t\underline{\boldsymbol{L}}) &= (\Delta t\underline{\boldsymbol{L}})^{-1}\left(e^{\Delta t\underline{\boldsymbol{L}}} - \underline{\boldsymbol{I}}\right)\\
\underline{f_3}(\Delta t\underline{\boldsymbol{L}}) &= (\Delta t\underline{\boldsymbol{L}})^{-2}\left(e^{\Delta t\underline{\boldsymbol{L}}} - \Delta t\underline{\boldsymbol{L}} - \underline{\boldsymbol{I}}\right)\\
\underline{f_4}(\Delta t\underline{\boldsymbol{L}}) &= (\Delta t\underline{\boldsymbol{L}})^{-3}\left(e^{\Delta t\underline{\boldsymbol{L}}} - (\Delta t\underline{\boldsymbol{L}})^2/2 - \Delta t\underline{\boldsymbol{L}} - \underline{\boldsymbol{I}}\right)
\end{aligned} \tag{7.9}$$

The above equations are not easily arrived at and Cox and Matthews themselves used a symbolic manipulator to derive their original forms. The singularities present themselves in Equation 7.9 where $\underline{f_2}$, $\underline{f_3}$ and $\underline{f_4}$ are just higher dimensional analogues of $(e^z-1)/z$, $(e^z-z-1)/z^2$ and $(e^z-z^2/2-z-1)/z^3$ - all of which contain removable singularities. To see how these singularities can be removed, assume $\underline{\boldsymbol{L}}$ is diagonal (if it is not, it can be diagonalized and the ensuing analysis applied to those diagonal elements only), and l is one element from its diagonal. Then, making use of the residue theorem:

$$f_j(l\Delta t) = \frac{1}{2\pi i}\oint_\Gamma \frac{f_j(l\Delta t)}{z - l\Delta t}\mathrm{d}z, \quad j \in \{2,3,4\}$$

As done in [43], a circle in the complex plane centered at $l\Delta t$ is used as the path Γ, and the integral approximated using m equally spaced points on the circle:

$$f_j(l\Delta t) = \frac{1}{m}\sum_{k=1}^{m} f_j(l\Delta t + re^{2\pi i(k-1)/m})$$

where r is a suitable radius. Further, if l is real, this integral can be approximated by taking just the upper half of the circle.

7.3. Pseudo-Spectral Method

Spectral methods are a family of numerical methods for computing approximate solutions of various PDEs, by expanding the terms involved into a set of orthonormal basis functions (that span the entire domain of the problem at hand). Pseudo-spectral methods are a family of methods based on the spectral method, that compliment the spectral basis with additional basis functions, in order to simplify and speed-up the computation of non-linear terms [27]. Of interest to this study is the pseudo-spectral method based on the Fourier series [28]. As an illustrative example, consider Equation 6.12 from Section 6.3:

$$\frac{\mathrm{d}\hat{u}_k}{\mathrm{d}t} = -\frac{ikv_1}{2}\underbrace{\mathcal{F}\left[u^2\right](k)}_{\mathcal{A}} + \left(v_2k^2 - v_3k^4\right)\hat{u}_k + \mathcal{F}[G(u,x,t)](k), \quad k = \frac{2\pi m}{L}, m \in \mathbb{Z}$$

The exact expansion of the the term $\mathcal{A}$ on the RHS, which is non-linear in nature, would be:

$$\mathcal{F}\left[u^2\right](k) = \frac{1}{L^2}\sum_q\sum_p \hat{u}_{k_p}\hat{u}_{k_q}\langle\phi_k, \phi_{k_p}\phi_{k_q}\rangle, \quad k_p = \frac{2\pi p}{L}, k_q = \frac{2\pi q}{L},\ p,q \in \mathbb{Z}$$

where $\phi_k = e^{ikx}$ is the discrete Fourier basis function and $\langle a,b\rangle = \int_{x=0}^{L} a\bar{b}\,\mathrm{d}x$ is the inner product on this space. In the discrete case, with N grid-points, the above term is $O(N^2)$ in its computational cost. Instead of performing this quadratic calculation, one simply computes $u(x_j,t)$ using the inverse FFT, squares it and then performs the FFT on this new squared quantity. This replaces the quadratic multiplication with a $O(N\log N)$ operation (the FFT), and hence the computational cost drops to $O(N\log N)$. This approach, while attractive on the surface, does introduce an additional error - the *aliasing* error [27]. The effect of this error can be seen in this very example. Consider the squared quantity $u^2(x_j,t)$:

$$\begin{aligned} u^2(x_j,t) &= \left(\frac{1}{N}\sum_{m=-N/2}^{N/2}\hat{u}_k e^{ikx_j}\right)^2 \\ &= \frac{1}{N^2}\sum_{q=-N/2}^{N/2}\sum_{p=-N/2}^{N/2}\hat{u}_{k_p}\hat{u}_{k_q}e^{i(k_p+k_q)x_j}, \quad x_j = j\Delta x, k_p = \frac{2\pi p}{N\Delta x} \end{aligned}$$

Clearly, this summation involves quantities with wave numbers $k_{new} = k_p + k_q$ and for N such wave numbers $|k_{new}| > \pi/\Delta x$. When taking the discrete Fourier transform of this term, all such wave numbers would be 'aliased' and their contribution added to the mode corresponding to the wave number $k = k_{new} - \pi/\Delta x$. To tackle this, additional steps can be taken, such as:

- adding modes (with value zero) corresponding to these problematic wave numbers ($|k_{new}| = l\pi/\Delta x, l \in \{1, 1+2/N, 1+4/N, \cdots, 2\}$) before applying the inverse FFT operation,
- then truncating the modes to the original set of wave numbers after applying the final FFT operation.

This is the general idea of pseudo-spectral methods, to avoid computing convolutions in the basis functions' domain, by performing the non-linear operation in the original function's space. The transformation is then applied on this newly computed non-linear quantity. The method retains the high accuracy and convergence rates of spectral methods, while offering an improved computational cost.

7.4. Lyapunov Spectrum Computation

The Lyapunov spectrum is an attempt at a quantitative measure of chaos. In a dynamical system within an n-dimensional phase space, there exists a set of n orthogonal vectors, along whose i^{th} vector, two infinitesimally close trajectories evolve with an average separation rate of $e^{\Lambda_i t}$. The set of values Λ_i is known as the Lyapunov spectrum. The presence of even a single positive value in this set implies the presence of chaos, since it represents the presence of local instability in that particular direction [35, 64]. The largest value of Λ_i is called the *maximal Lyapunov exponent* (MLE) - Λ_{MLE}. Considering a system

with a positive MLE, two narrowly separated vectors (with the separation vector being in an arbitrary direction) will eventually grow apart at a rate $e^{\Lambda_{MLE}t}$ [1]. Hence the MLE is an excellent quantifier of the chaos inherent in a system, and is often used in the study of predictive models to normalize time when comparing/presenting results.

A method of estimating the Lyapunov spectrum of the KS system, with a relatively high accuracy and small computational time is provided in [57]. This can easily be adapted to any dynamical system, and this extended formulation is provided here.

Consider a dynamical system:

$$\dot{\boldsymbol{u}} = \boldsymbol{f}(\boldsymbol{u}, t)$$

with an initial condition $\boldsymbol{u}(t = 0) = \boldsymbol{u}_0$ and an n-dimensional phase space. Consider now a matrix $\underline{\boldsymbol{M}}$ made up of orthogonal columns each of whose Euclidean norm is Δu, where Δu is a small number[2]. These are the initial set of orthogonal separation vectors along which the system shall now be evolved. Effectively, there are now $n + 1$ trajectories that need to be evolved - starting from $\boldsymbol{u}_0$ (called $\boldsymbol{u}(t)$) and $\boldsymbol{u}_0 + \boldsymbol{M}_j$ (called $\boldsymbol{y}_j(t)$, where $\boldsymbol{M}_j$ is the j^{th} column of the matrix $\underline{\boldsymbol{M}}$). After ζ time steps, the matrix of separation vectors is recomputed as $\boldsymbol{M}_j = \boldsymbol{y}_j(t) - \boldsymbol{u}_j(t)$. A QR decomposition is performed on this re-computed $\underline{\boldsymbol{M}}$ which yields the matrices $\underline{\boldsymbol{Q}}$ and $\underline{\boldsymbol{R}}$. The absolute values of the diagonal elements of $\underline{\boldsymbol{R}}$ are stored, $\underline{\boldsymbol{M}}$ is reset to $\Delta u \underline{\boldsymbol{Q}}$ and the whole computation repeated a total of ξ times. Then the Lyapunov exponents are computed as the average of the exponents over each of those ζ time steps:

$$\Lambda_j = \frac{1}{\xi} \sum_{i=1}^{\xi} \frac{\log\left(R_{jj}(i)\right)}{\zeta \Delta t}$$

where $R_{jj}(i)$ is the stored j^{th} diagonal element of $\underline{\boldsymbol{R}}$ (of the i^{th} iteration).

7.5. Summary

This chapter provides an overview of the different solvers to be used. As can be seen, the PDEs require special treatment under the pseudo-spectral and ETDRK4 methods, since their simulation using regular Runge-Kutta methods would require extremely fine time-steps and large computational resources. Also covered is an approximate method of computing the Lyapunov spectrum of a given system, thereby quantifying its inherent chaos. This is important since as outlined Chapter 3, these quantities are essential to computing both the KY dimension and the Lyapunov time.

[1] The separation vector will typically have a component in the direction corresponding to Λ_{MLE}, and owing to the exponential growth rates eventually the growth in the MLE's direction will outpace all other growth directions.

[2] This value depends on the kinds of values $\boldsymbol{u}$ takes on during the system's evolution, but anywhere between 10^{-9} to 10^{-7} is generally considered small enough.

8

General Workflow

This chapter describes the general methodology adopted for the entire project, across the different chaotic systems. Starting with the simulation of the chosen chaotic systems, Section 8.1 describes in detail the solvers used to generate the training/testing datasets, and the parameters used to do so. Section 8.2 details the layering architectures of the auto-encoders and the selection of hyper-parameters, along with the utilized error metrics and other comparison methods. Section 8.3 deals with design of the RNNs - the layering architectures and sizes - along with their training and hyper-parameter selection.

8.1. Simulation and Data Generation

8.1.1. The Lorenz '63 System

The Lorenz '63 system equations as described in Section 6.1 are directly integrated using the RK4 method, with a time-step $\Delta t = 0.01$. The first 1000 time-units (i.e. the first 100000 time-steps) are discarded to remove the influence of the initial condition, and the next 42000 time-units stored as the useable dataset.

For the single parameter-set case, the canonical parameters $\sigma = 10, \rho = 28, \beta = 8/3$ are used (Figure 6.2), whilst for the multi-regime auto-encoder the parameter-set obtained from all possible combinations of $\sigma \in (10, 20)$, $\rho \in (35, 45)$ and $\beta \in (4/3, 8/3)$ is used. These parameter sets are chosen to ensure chaotic dynamics are observed, by ensuring $\sigma > \beta + 1$ and $\beta(\rho - 1) > 0$ (Section 6.1). The various phase spaces are shown in Figure 8.1. The differing dynamics caused by the changing parameters can also be clearly observed. The increase in σ appears to increase the size of the lobes, in addition to making the trajectory stay farther away from the attractor positions. This is to be expected since it governs the magnitude of $\dot{x}_1$. Aside from changing the attractor positions, the increase in ρ and β appear to correspond to increased larger lobes and more chaotic trajectories as well. The MLE, KY dimension and the Lyapunov time (as discussed and defined in subsection 3.2.1 and subsection 3.2.2) of the different cases are provided in subsection A.1.1. These different dynamics should be able to test the multi-regime auto-encoder's compression abilities well.

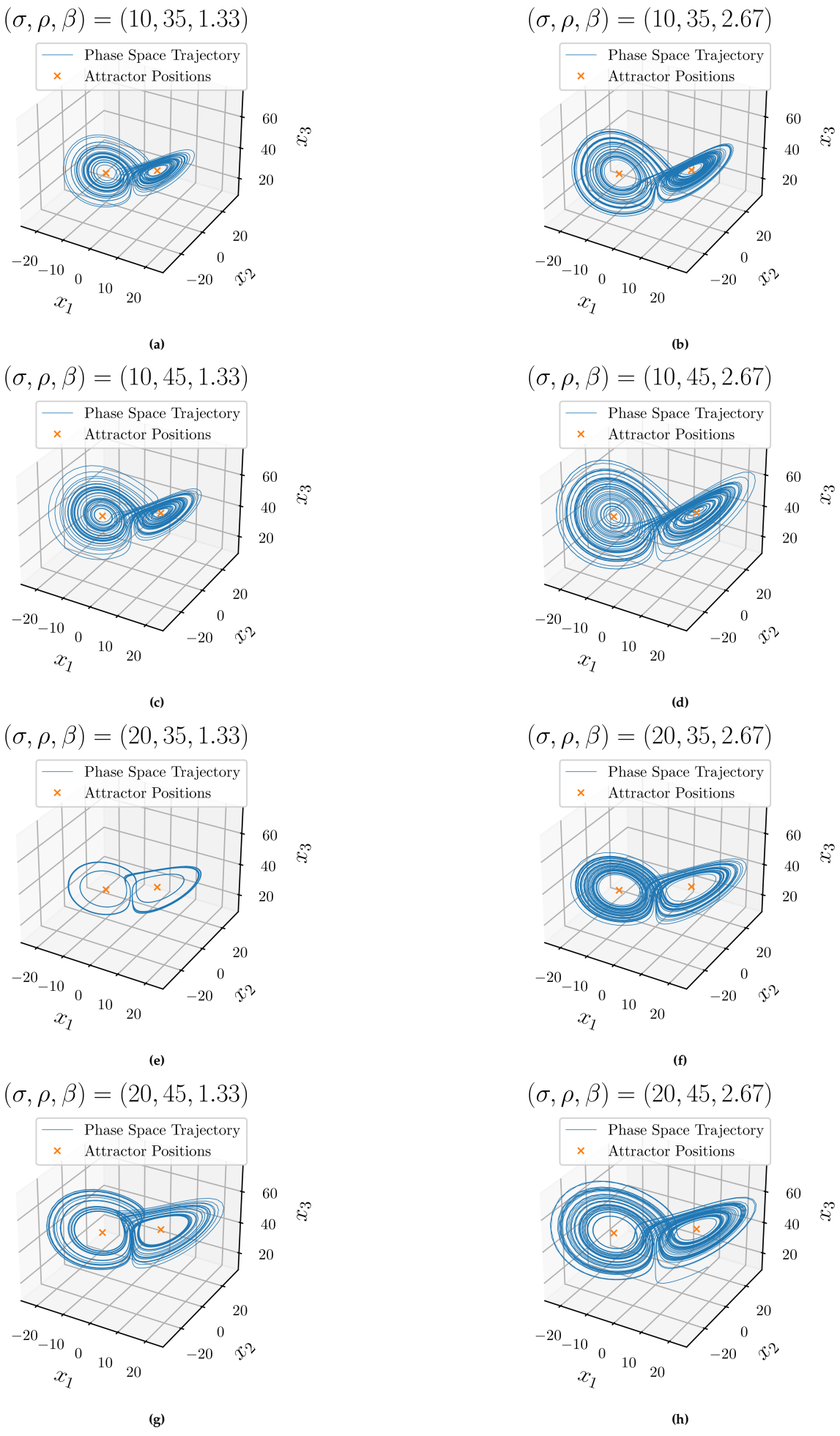


Figure 8.1: Phase space of the Lorenz system with different parameter sets.

8.1.2. The Charney-DeVore System

The CDV system equations (Section 6.2) are directly integrated using the RK4 method with a time-step of $\Delta t = 0.1$. The first 1000 time-units (i.e. the first 10000 time-steps) are discarded to remove the influence of the initial condition, and the next 50000 time-units are stored as the useable dataset.

The parameter set $(x_1^*, x_4^*, C, \beta, \gamma, b) = (0.95, -0.76095, 0.1, 1.25, 0.2, 0.5)$ is used for the single parameter-set case. It is harder to find parameter sets that produce chaotic trajectories for the CDV system, as noted in [18], and one method of doing so is to follow $r = x_1^*/x_4^* = -0.801$, keeping the other parameters constant. Even then, the trajectories need to be inspected lest they possess a non-chaotic attractor (Figure 8.3). Here, the first three cases result in constant trajectories, and only the $x_1^* = 0.95$ case is truly chaotic. The remaining two appear to be quasi-periodic rather than fully chaotic (as can be seen in the $x_1 - x_4$ plane Figure 8.2). Hence following this parameter search, for the multi parameter-set case, only a single additional set $(0.99, -0.79299, 0.1, 1.25, 0.2, 0.5)$ is added. The MLE, KY dimension and Lyapunov times are given in subsection A.1.2. A deeper analysis (subsection A.1.2) reveals that this parameter set produces not just a quasi-periodic but a fully periodic solution trajectory. This gives the auto-encoder two dynamical regimes to model.

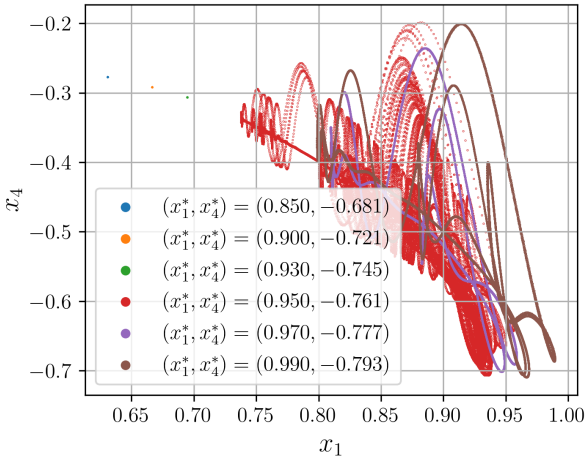


Figure 8.2: CDV solution trajectories, plotted in the $x_1 - x_4$ plane.

8.1.3. The Kuramoto-Sivashinsky System

The KS system equations (Section 6.3) are solved using the pseudo-spectral method with time-stepping using the ETDRK4 method (using a provided in-house solver), with a time-step of $\Delta t = 0.1$, on a periodic spatial domain of length $L = 35$ and 64 points to discretize the domain. The first 10000 time-units (i.e. the first 100000 time-steps) are discarded to remove the influence of the initial condition, and the next 50000 time-units are stored as the useable dataset.

The parameter set $(\nu_1, \nu_2, \nu_3) = (1, 1, 1)$ is used with $G(u, x, t) = 0$ for the single parameter-set case. For the multi parameter-set, $(\nu_1, \nu_2, \nu_3) = (1, 1, 1),\ (2, 1, 1),\ (1, 2, 1),\ (1, 1, 2)$ are used (Figure 8.4). Only the first two sets result in chaotic dynamics, with the rest resulting in periodic solutions (subsection A.1.3). This is even more visible when the mean kinetic energy, the mean dissipation rate and the mean turbulent kinetic energy are plotted (Figure A.2, Figure A.3, Figure A.4). In the first two cases, the increase in ν_1 can be interpreted to lead to more 'active dynamics', since this controls the rate at which energy is redistributed amongst the different modes (Equation 6.12). This can also be observed in the marginally smaller Lyapunov time of the second case as compared to the first (subsection A.1.3), implying nearby trajectories will tend to diverge just a little bit faster as compared to the first case. These four cases provide the auto-encoder with enough regime changes to model.

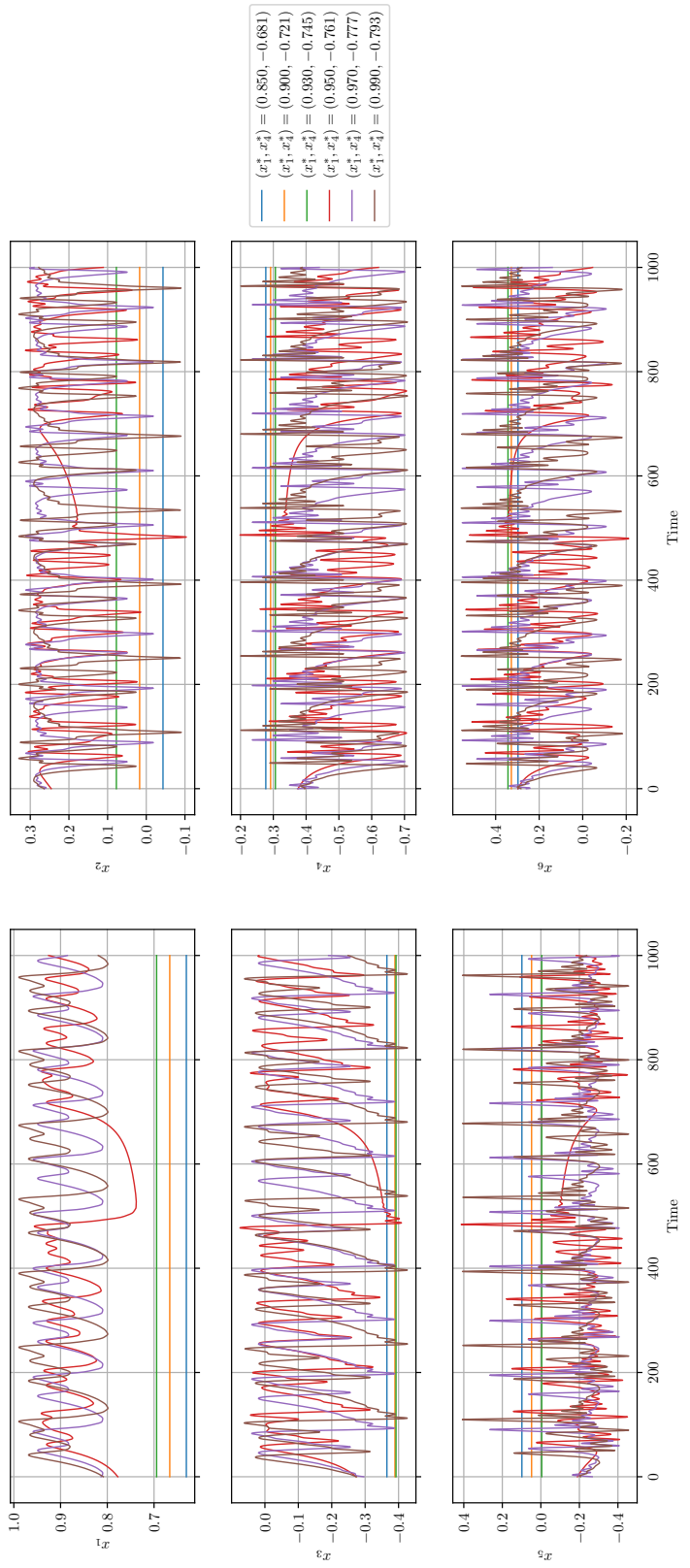


Figure 8.3: Time evolution of the individual components in a CDV system, for different parameter sets.

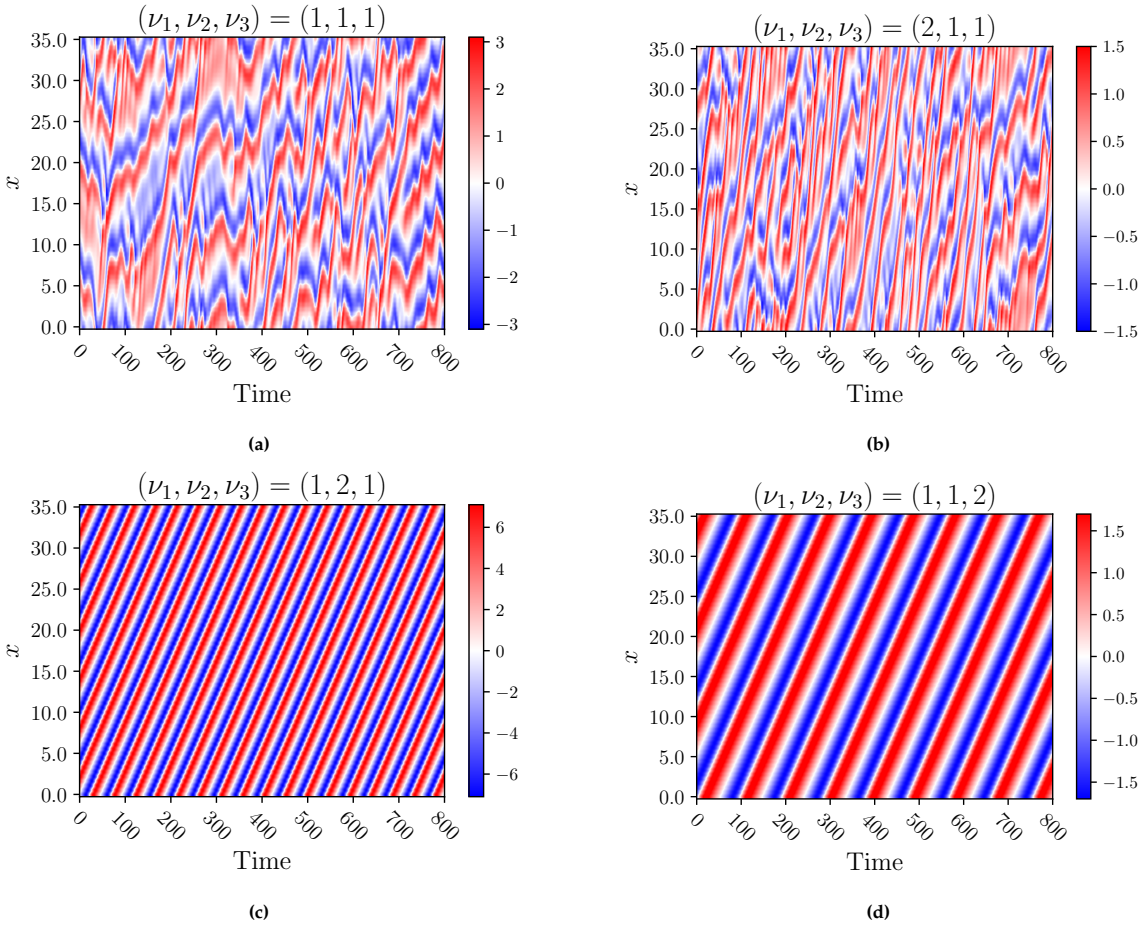


Figure 8.4: Time evolution of the KS system with different parameter sets.

8.1.4. The Kolmogorov Flow System

The 2D incompressible Navier-Stokes equations (Section 6.4) are solved on a $2\pi \times 2\pi$ doubly-periodic spatial domain using a provided in-house solver that employs the pseudo-spectral method after first applying the Leray projection operator. The equations are time-integrated in Fourier space using the RK4 method. The Fourier transform is taken using 32 wave-number pairs (a total of 65 modes) each in the x and y directions. The solution is computed on a 50×50 grid in the spatial domain using the inverse Fourier transform. The equations are time-stepped using $\Delta t = 0.01$, however owing to storage size constraints the solution is saved only every 0.25 time-units (i.e. 25 time-steps). The first 500 time-units are discarded to remove the influence of the initial condition, and Reynolds number $Re = 30, 40$ along with $k_f = 4$ (the forcing frequency) are used to produce chaotic solutions.

Snapshots and means of the x-direction and y-direction velocities (u_x, u_y) and the vorticity $\omega = (\nabla \times \boldsymbol{u}) \cdot \boldsymbol{e}_3 = \frac{\partial u_x}{\partial y} - \frac{\partial u_y}{\partial x}$ are plotted Figure 8.6 and Figure 8.5. The forcing function $\boldsymbol{f}$ can be observed in the mean u_x ($\overline{u}$) plot, where the alternating of the four sinusoids (corresponding to $k_f = 4$) are clearly visible (Figure 8.6d). Further, the fact that the energy spectrum (provided in Figure 8.7b) exactly replicates that in [25] justifies the choice of using 32 wave-number pairs to accurately model the system dynamics.

Further dynamics can be observed in the mean kinetic energy, mean turbulent kinetic energy and mean dissipation rate plots (Figure A.6, Figure A.7). Other chaos quantifiers such as the MLE, the KY dimension and the Lyapunov times are provided in subsection A.1.4.

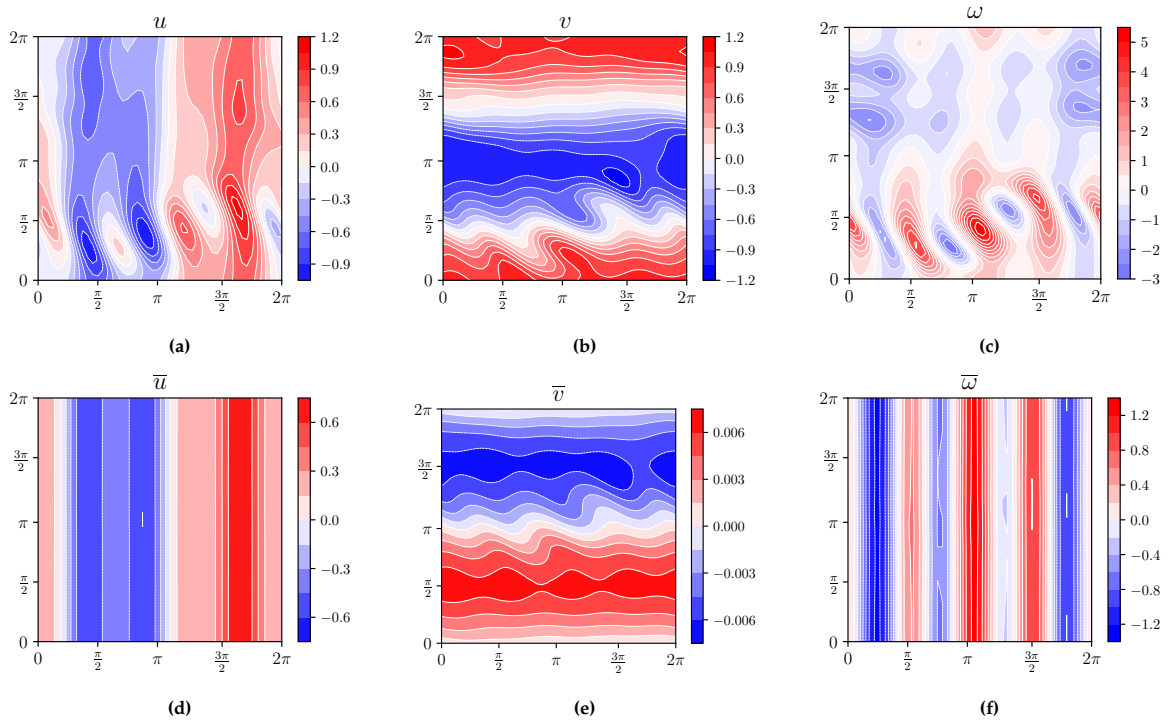


Figure 8.5: Iso-contours of u_x, u_y and vorticity ω ($Re = 30$) - snapshots a, b, c and time means d, e, f.

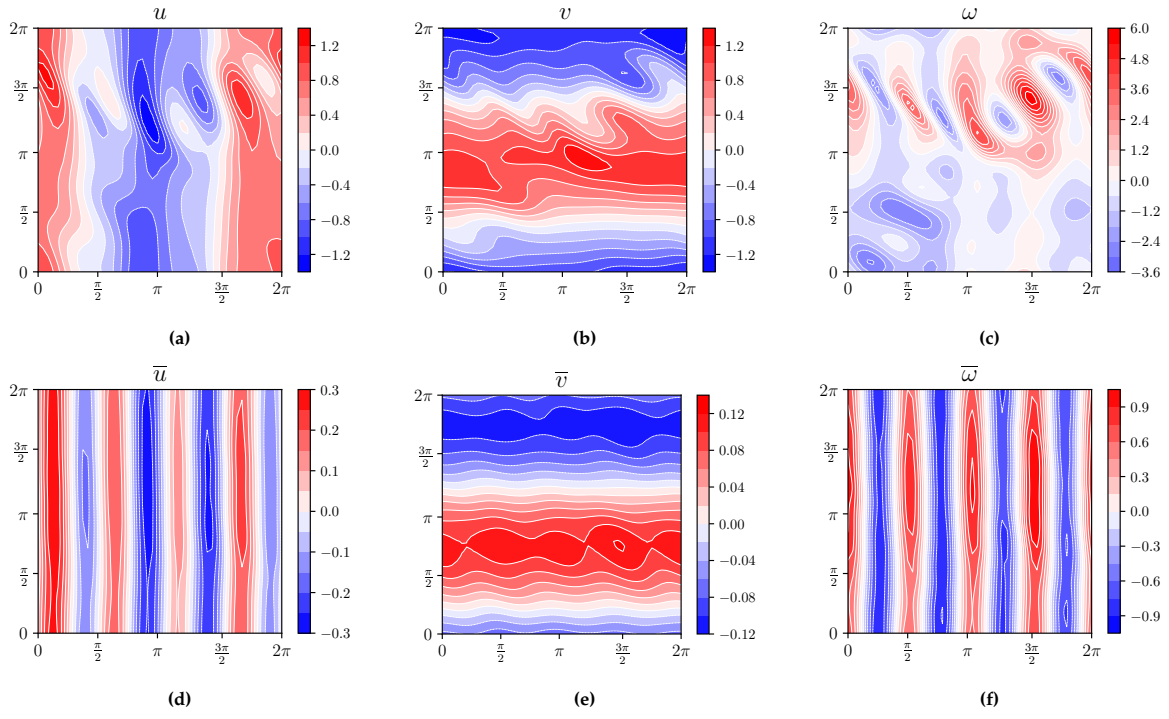
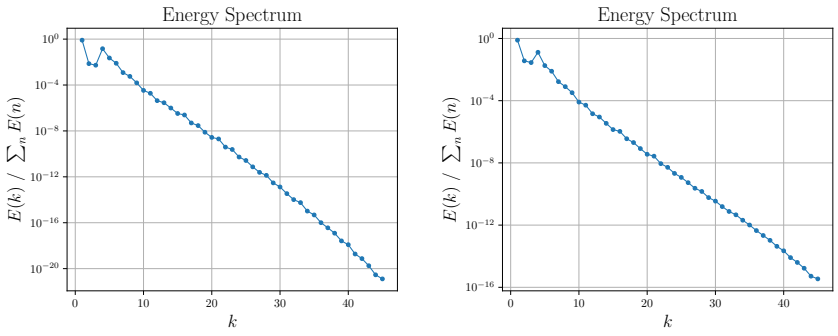


Figure 8.6: Iso-contours of u_x, u_y and vorticity ω ($Re = 40$) - snapshots a, b, c and time means d, e, f.



(a) $Re = 30$ **(b)** $Re = 40$

Figure 8.7: Energy spectrum of the Kolmogorov flow system.

8.2. Auto-encoders

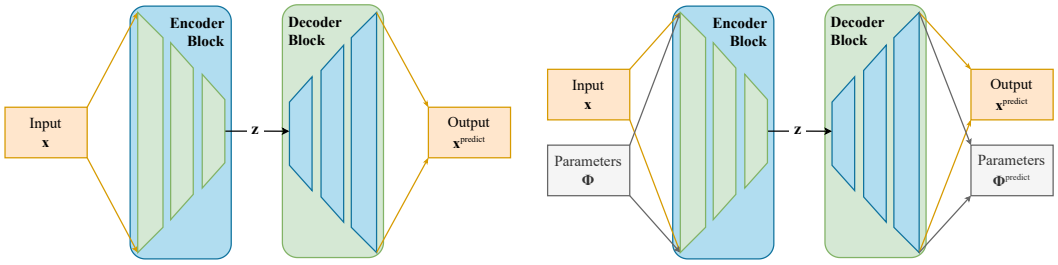
This section delves into the fundamental principles of auto-encoder training, including the choice of network architectures, loss functions, and optimization techniques. Further discussed are the strategies adopted to mitigate over-fitting and optimal selection of certain hyper-parameters.

8.2.1. Layering and Network Architecture

Fully Connected Auto-Encoder Network

For the first three systems - the Lorenz, CDV and KS systems - the input feature vector is one-dimensional. Thus a simple fully-connected feed-forward network architecture is used for both the encoder and decoder networks. The exponential linear unit (ELU) function is used as the activation function in the intermediate layers, and the hyperbolic tangent used as the final layer's activation function for both the encoder and decoder networks (chosen for its *quashing* property - subsection 4.1.4). The ELU function has been found to perform better than the ReLU function. This is owed to its tail in the $x < 0$ regime which helps add balance to the layer's output. This assists with the issues of non-centeredness and 'dying' neurons that plague ReLU units (much like the leaky ReLU [80]).

In the single regime case, the general architecture of the auto-encoder looks like Figure 8.8a. Here the system's state vector is fed into the network, and the reconstructed data is its output. The latent state vector $\mathbf{z}$ is the output of the encoder and the input of the decoder. Similarly, for the multi-regime auto-encoders, an initial test is performed using this architecture (as shown in subsection 9.1.3). However, noting its generally poor performance and failure to segregate different regimes in the latent space, a second architecture (Figure 8.8b) is tested. Here, the parameters specific to each regime are passed into encoder as an additional input, and generated by the decoder as an additional output. The reconstruction error on these parameter vectors is included in the cost function, thus forcing the network to learn to separate the different regimes. In practice, this is implemented by simply appending the parameter vector to the system state vector and then commencing the training as per usual.



(a) Only system state treated as input feature vector. **(b)** System state and parameter set treated as input feature vectors.

Figure 8.8: Typical flow-charts of the auto-encoder configurations.

The exact layer sizes and number of layers for each case are given in subsection A.2.1. The general strategy adopted in picking the layer sizes for the encoder network is to double the input's dimensionality

(to the nearest power of two), and then go on reducing the dimensions in successive layers. The decoder network is simply the encoder network's layer configuration in reverse.

Convolutional Auto-Encoder Network

For the Kolmogorov flow system, since the input data resides in a $\mathbb{R}^{N_x \times N_y \times 2}$ space[1] and possesses strong spatial relationships, a convolutional auto-encoder is employed to generate the latent space [19, 78, 55]. Convolution (Figure 8.9a) is a weight-sharing measure oft used in cases where the data displays spatial relationships such as images and videos [32]. However, since it is a local operation it tends to neglect global relationships. This is mitigated by adding self-attention skip-layers in the encoder and decoder networks, allowing them to detect large-scale patterns [78, 82]. The exact layer-configurations can be found in subsection A.2.1.

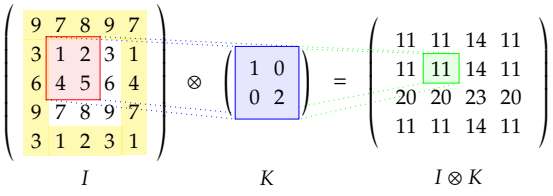
Convolutions are only employed in the encoder, the decoder utilitzes *de-convolutions*, also known as *transposed convolutions* [22]. These are the exact reverse operation of a convolution, and are an effective method of super-sampling a lower dimensional input. An example is shown in Figure 8.9b. Like the convolutions, these too are local operations, and thus self-attention skip-layers are incorporated in the decoder as well.

Further, instead of a single filter-size kernel, a multi-scale network is used to construct the auto-encoder [21]. The general auto-encoder architecture can be seen in Figure 8.10. Here, there exist parallel encoder and decoder networks, operating independently on their inputs. A weighted sum is performed on the individual outputs to produce the final output. The weights used to compute the outputs are themselves learnable parameters and are optimized during training, starting from initial values $1/n_p$ (n_p being the number of parallel networks). Lastly, keeping the periodic nature of the system in mind, periodic padding is employed to pad the inputs before performing the convolutions.

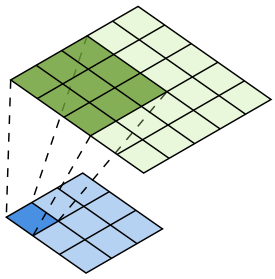
For the multi-regime case, the Reynolds number is treated as an additional data channel, effectively transforming the data into a $\mathbb{R}^{N_x \times N_y \times 3}$ space. This amounts to adding $N_x N_y$ 'parameter variables' to the original state vector and gives the final data channel (corresponding to the Reynolds number) too much sway over the loss function. Note that in all previous cases the respective parameter sets are appended to the state vector only once, thus resulting in only a single instance of each system parameter in the augmented input data. In order to mitigate the undue influence resulting from its multiplicity, the MSE error is modified as follows:

$$\mathrm{MSE}_{\text{augmented}}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) = \mathrm{MSE}\left(\mathbf{x}^{\text{true}}_{(1,2)}, \mathbf{x}^{\text{predict}}_{(1,2)}\right) + \left(\frac{1}{N_x N_y}\right) \mathrm{MSE}\left(\mathbf{x}^{\text{true}}_{(3)}, \mathbf{x}^{\text{predict}}_{(3)}\right), \tag{8.1}$$

where $\mathbf{x}_{(i,j,\dots)}$ represents the slice of $\mathbf{x}$ containing only the $i^{th}, j^{th}, \dots$ data channels. This removes the added data channel's undue influence on the loss function, and then training is proceeded with as usual.



(a) Convolution operation performed on a 2D matrix using a 2×2 kernel (with periodic padding, highlighted in yellow).



(b) Transposed convolution operation performed on a 2D matrix using a 3×3 kernel (note the 3×3 input matrix gets up-sampled to a 5×5 matrix).

Figure 8.9: Examples of the convolution and transposed convolution operations.

Choice of Latent Space Dimension

The choice of the latent space dimension is crucial for accurate data reconstruction. Here, it is motivated by an argument from differential topology. Assume the chaotic attractor represents a m-manifold $\mathcal{M}$ (m being the Lebesgue covering dimension), and has a Hausdorff dimension D_H. It is known that $m \leq D_H$

[1] N_x and N_y are the domain sizes in the x and y directions, respectively (both being equal to 50 in the present case).

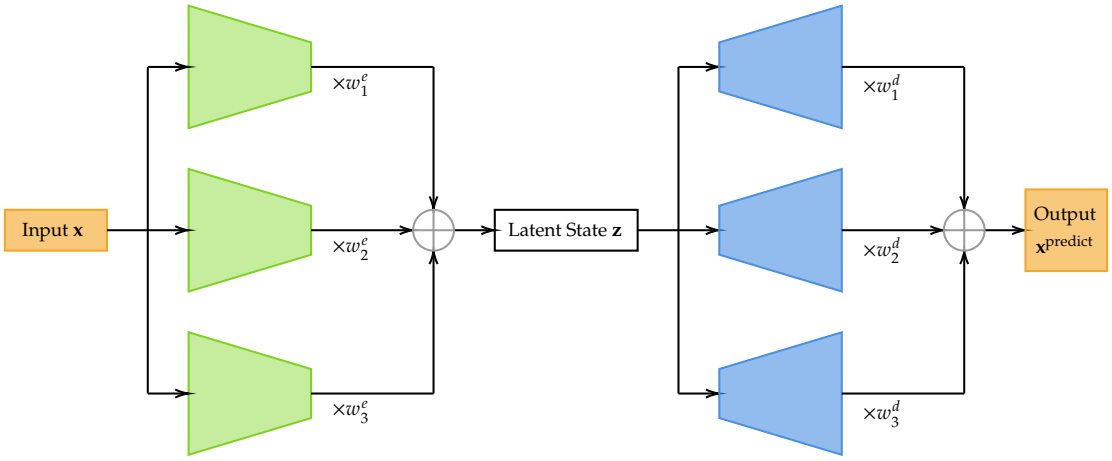


Figure 8.10: Multi-scale Convolutional Auto-encoder (w_i^e and w_i^d are the weights for the outputs of the parallel encoders and decoders, respectively).

[23, Theorem 6.3.10 on p.183], and that the mainfold $\mathcal{M}$ can be embedded in a Euclidean space $\mathbb{R}^n$ with $n \geq 2m$ (under suitable assumptions of smoothness, see Whitney's strong embedding theorem [67]). This implies that the chaotic attractor should be able to be accurately represented in a Euclidean space $\mathbb{R}^p$ with $p = \lceil 2D_H \rceil$[2]. The Hausdorff dimension D_H is estimated using the Kaplan-Yorke dimension D_{KY} (subsection 3.2.2), and thus $n^{ls} = \lceil 2D_{KY} \rceil$ is chosen to be the dimensionality of the latent space for the auto-encoders.

8.2.2. Error Metrics

Error metrics play a pivotal role in assessing the accuracy and efficacy of the model. This subsection focuses on the selection and application of error metrics, which quantitatively evaluate the reconstruction performance of auto-encoders. In addition to the standard MSE, another error metrics that has been monitored is the *Normalized* Mean Squared Error (NMSE). These are defined below:

$$\text{MSE}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) = \mathbb{E}_{\mathbf{x}}\left[\frac{|\mathbf{x}^{\text{true}} - \mathbf{x}^{\text{predict}}|^2}{n_x}\right] \tag{8.2}$$

$$\text{NMSE}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) = \mathbb{E}_{\mathbf{x}}\left[\frac{1}{n_x}\sum_i \left(\frac{x_i^{\text{true}} - x_i^{\text{predict}}}{\sigma_i}\right)^2\right], \tag{8.3}$$

where $\mathbf{x}$ represents the flattened feature vector, x_i is its i^{th} component, σ_i the standard deviation of x_i and n_x the length of vector $\mathbf{x}$. The NMSE is a good normalized error metric that equalizes the different components in magnitude. As noted in subsection 5.1.3, the smaller components may not be statistically significant but are important to the dynamics of a system and this ensures they do not get neglected. To present the results in Chapter 9, another error metric, the *Normalized Root* Mean Squared Error (NRMSE), has been utilized. This has the same benefit as the NMSE of being normalized and additionally is of the same dimensional units as the input data.

$$\text{NRMSE}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) = \mathbb{E}_{\mathbf{x}}\left[\sqrt{\frac{1}{n_x}\sum_i \left(\frac{x_i^{\text{true}} - x_i^{\text{predict}}}{\sigma_i}\right)^2}\right] \tag{8.4}$$

8.2.3. Loss Function

The choice of appropriate loss functions for training is of utmost significance, since they quantify the dissimilarity between the reconstructed output and the original input data. The standard loss function

[2] $D_H \geq m \implies \lceil 2D_H \rceil \geq 2m$

used is:

$$J(\boldsymbol{\theta}) = \text{MSE}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) + \lambda_{\text{reg}} L^2_{penalty}(\boldsymbol{\theta}), \tag{8.5}$$

where $L^2_{penalty}(\boldsymbol{\theta})$ is the L^2 norm of the network's weights and biases ($\boldsymbol{\theta}$), and λ_{reg} a hyper-parameter controlling the amount of added penalty.

Contractive Loss

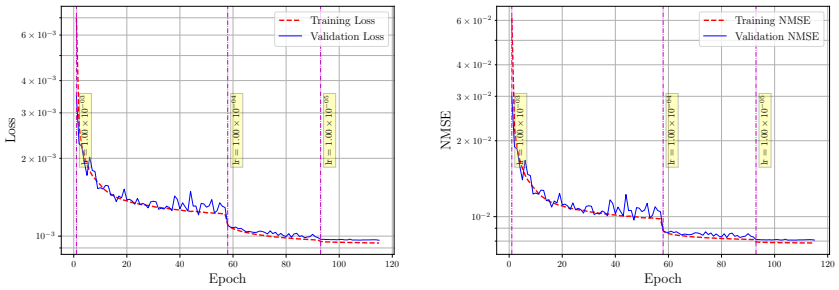
As mentioned in Section 4.2, a contractive loss is added to the loss function as follows:

$$J(\boldsymbol{\theta}) = \text{MSE}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) + \lambda_{\text{reg}} L^2_{penalty}(\boldsymbol{\theta}) + \lambda_{\text{contractive}} \|\nabla_{\mathbf{x}} \mathbf{z}\|^2, \tag{8.6}$$

where $\|\nabla_{\mathbf{x}} \mathbf{z}\|^2$ is the Frobenius norm of the encoded latent space vector ($\mathbf{z}$) w.r.t. the input feature vector ($\mathbf{x}$), and $\lambda_{\text{contractive}}$ the hyper-parameter controlling how much of it to add to the loss. The idea behind this addition is to have the network itself train to place similar inputs close to each other in the latent space, thereby 'smoothening out' the variations, hopefully leading to an easier-to-model time series for the RNNs.

8.2.4. Training

The Adam optimizer [44] is used to train the networks, in three successive rounds. Each round has a successively lower learning rate, starting from 10^{-3}, progressing to 10^{-4} and eventually 10^{-5}. The maximum number of epochs per round is limited to 200, and early stopping with patience (subsection 4.1.6) set to 10 epochs is utilized. This training strategy is found to perform quite well, and a typical training curve is provided in Figure 8.11. It can be observed how the loss decreases even further between successive rounds. The latter rounds essentially fine-tune the weights, as evidenced by the minimal changes in the loss function. The data is split into three sections - 80% is used for training, 10% for validation and the remaining 10% for testing.



(a) Loss training history **(b)** NMSE training history

Figure 8.11: Typical auto-encoder training curves.

Data Normalization before Training

The input data, as can be seen in Section 8.1, covers a vast range of scales, all of which are not suitable for use in ML training. Most activation functions work in a narrow range, generally around the interval $(-1, 1)$, and hence input data needs to be altered before training can commence. To this end, the input data in all cases is first mean-centered by subtracting the mean, and then *standardised* by dividing each entry x_i by $3\sigma_i$. This normalization ensures the data is both zero-centered and roughly within the interval $(-1, 1)$. Other normalization options exist, such as the min-max normalization which forcibly rescales the data to be in the range $[0, 1]$ or $[-1, 1]$ using the minimum and maximum values. This was considered unsuitable since it lends too much weight to the outliers in the data and has a tendency to skew the input distribution [32].

Noise Addition

Gaussian noise is added to the input data in order to speed up the training and make the network robust

to errors in the input. The mean of the noise distribution is always set to zero, and all that needs to be specified is its standard deviation, σ_{noise}. This set as:

$$\sigma_{\text{noise}} = f_{\text{noise}} \sigma_{\text{mean}}, \tag{8.7}$$

where $\sigma_{\text{mean}} = \left(\sum_i \sigma_i\right) / n_x$ is the mean of the standard deviations of the individual components of the input feature vector. Then f_{noise} becomes the hyper-parameter that needs specification, and its selection process is detailed in subsection 8.2.5.

8.2.5. Hyper-Parameter Selection

The success of any ML training campaign hinges upon the judicious selection of hyper-parameters, and this section describes the methodology employed in this project. The different hyper-parameters and their selection strategies are given in Table 8.1. The learning rate(s), maximum epochs per round, early stopping patience, and the total number of rounds are set based on prior experience and initial exploratory experiments.

Hyper-parameter	Selection method	Value
learning rate(s)	constant, pre-set	$\{10^{-3}, 10^{-4}, 10^{-5}\}$
λ_{reg}	Bayesian optimization	*problem dependent*
$\lambda_{\text{contractive}}$	Bayesian optimization	*problem dependent*
f_{noise}	Bayesian optimization	*problem dependent*
early stopping patience	constant, pre-set	10
maximum epochs (per round)	constant, pre-set	200
number of rounds	constant, pre-set	3
batch size	constant, pre-set	64

Table 8.1: Hyper-parameters and their selection methods.

Bayesian Optimization

Bayesian optimization (using Gaussian process regression [53, 68, 29]) utilizes a probabilistic surrogate model to model a blackbox function and efficiently explores the hyper-parameter space, minimizing said blackbox function. It is ideal for use in cases where each function evaluation is expensive, and hence perfectly suited for hyper-parameter tuning within the machine learning context. As such, it is used to optimize the values of the hyper-parameters λ_{reg}, λ_{noise} and f_{noise}.

The general strategy employed is to provide the optimizer with an initial set of input vectors, and the function evaluations at those initial points. The optimizer then finds new optimal points using the specified acquisition function. The *Expected Improvement* acquisition function is used which is said to balance exploration of the search space with straightforward optimization of the computed surrogate model (termed 'exploitation'). Table 8.2 provides the bounds imposed and the priors assumed on the different hyper-parameters.

The search is carried out in two stages. In the first stage, $\lambda_{\text{contractive}}$ is set to 0 and the search space is two-dimensional, made up of λ_{reg} and f_{noise}. Twenty initial points are sampled from a space filling curve to ensure that the search space is adequately represented. Auto-encoders are trained on these initial points and the testing data's reconstruction MSE is used as the blackbox function to minimize. The Bayesian optimization is allowed to run for an additional 10 iterations, which is found to be sufficient to find the optimal values of λ_{reg} and f_{noise}. The hyper-parameters are found for the single parameter-set case and then the same values are used for the multi-regime auto-encoders.

Hyper-parameter	Bounds	Prior
λ_{reg}	$[10^{-7}, 10^{-3}]$	log-uniform
$\lambda_{\text{contractive}}$	$[10^{-6}, 10^{-1}]$	log-uniform
f_{noise}	$[10^{-4}, 10^{-1}]$	log-uniform

Table 8.2: Hyper-parameters, their imposed bounds and assumed priors.

Once the optimal λ_{reg} and f_{noise} have been found, stage two is commenced, in which the Bayesian optimization strategy is used to find the optimal value of $\lambda_{\text{contractive}}$. Ten initial values are chosen for $\lambda_{\text{contractive}}$ and again the testing data's reconstruction MSE used as the objective to minimize. Further twenty iterations are run for the Bayesian optimization algorithm to find the optimal value.

8.2.6. Latent Space Principal Directions

Once the auto-encoders are trained, the identified latent space is analysed through its principal directions. These are computed by applying the POD method described in subsection 5.1.1 to the computed latent states:

$$\mathbf{p}_i^{ls} = \frac{\lambda_i^{ls}}{\sum_j \lambda_j^{ls}} \phi_i^{ls} + \overline{\mathbf{z}}, \tag{8.8}$$

where $\overline{\mathbf{z}}$ is the mean of the latent states (subtracted before performing the decomposition), ϕ_i^{ls} is the i^{th} principal direction unit-vector obtained from the decomposition and λ_j^{ls} are the eigenvalues associated with each principal direction. The term $\frac{\lambda_i^{ls}}{\sum_j \lambda_j^{ls}}$ is meant to weigh each direction with its contribution to the cumulative variance of the mean subtracted data (see subsection 5.1.1).

Likewise, the principal directions of the original data are computed as:

$$\mathbf{p}_i^{true} = \frac{\lambda_i^{true}}{\sum_j \lambda_j^{true}} \phi_i^{true} + \overline{\mathbf{x}}, \tag{8.9}$$

with $\overline{\mathbf{x}}$ being the subtracted mean of $\mathbf{x}$ and the remaining terms defined similarly as before. Then $\mathcal{D}\left(\mathbf{p}_i^{ls}\right)$, where $\mathcal{D}(\cdot)$ represents the application of the decoder network, is plotted and analysed alongside $\mathbf{p}_i^{true}$. Further, a correlation matrix $\underline{\boldsymbol{C}}$ defined as:

$$C_{p,q} = \frac{\mathcal{D}\left(\mathbf{p}_p^{ls}\right) \cdot \mathbf{p}_q^{true}}{\|\mathcal{D}\left(\mathbf{p}_p^{ls}\right)\| \, \|\mathbf{p}_q^{true}\|}, \tag{8.10}$$

is computed to better understand the relationships between the latent and physical spaces.

8.3. Recurrent Neural Networks

Once the auto-encoders are trained and the latent states computed, the task at hand switches to modelling the time-series they represent. This section details the RNN design choices - the layer sizes and their connectivity - along with their training and hyper-parameter optimization strategies.

8.3.1. Error Metrics

The error metrics used here are the same as those for the auto-encoders, the only difference is that now there is an additional dimension corresponding to the time-series that is added to the data being worked on.

Prediction Horizon

To quantify the effectiveness of a time-series modelling system, a separate metric called the *prediction horizon* is defined. This is defined as the first moment the NRMSE increases past a pre-set threshold value:

$$\text{Prediction Horizon} = \underset{t}{\text{argmin}} \left\{ t \;\middle|\; \text{NRMSE}\left(\mathbf{x}^{\text{true}}(t), \mathbf{x}^{\text{predict}}(t)\right) \geq \epsilon_t \right\}, \tag{8.11}$$

where ϵ_t is the error threshold and t is the time from the start of the prediction cycle. In this project, $\epsilon_t = 0.5$ has been used. The prediction horizon is normalized by the Lyapunov time of the system when reporting.

8.3.2. Training

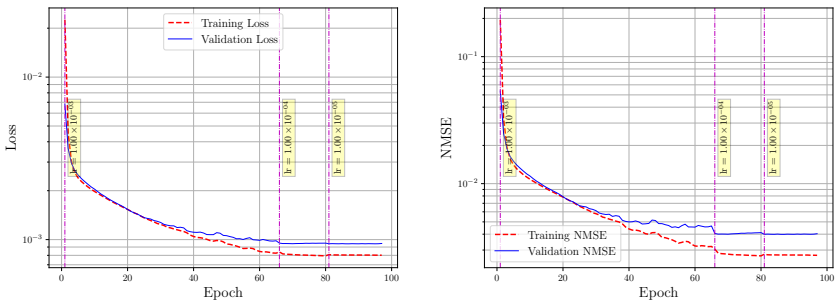
In this stage the RNNs are trained on the latent states computed by the encoder networks of the auto-encoders. These are then normalized using the same technique as subsection 8.2.4. The RNNs are then trained in a teacher-forced mode.

Teacher-forced training

Teacher-forced training refers to a specific technique in which, during the training phase, the RNN receives the true ground-truth data as inputs at each time step, rather than its own predictions from the previous step. This implies that they are essentially trained in a single-step prediction mode, as can be seen in Figure 8.13. Whilst imperfect, this method has the advantage of decoupling the inputs to a cell from previous outputs, thus enabling faster training times [32].

The ESNs are trained by simply setting up the required matrices and computing an inverse, as described in subsection 4.3.4. The remaining RNNs are trained using BPTT, employing the Adam optimizer and a multi-round set-up as done for the auto-encoders in subsection 8.2.4. A typical training curve is shown in Figure 8.12.

Note that for the ESNs, it is common practice to train an ensemble and use their averaged outputs as the prediction [41, 50]. The randomly initialized reservoir and input connections in an ESN introduce variability in the network's behaviour, leading to differences in its performance for different runs. The use of an ensemble addresses the issue of variability and instability in individual ESNs. By creating multiple ESN instances, each with different random initializations of the reservoir, the ensemble captures a broader range of behaviours. In this project, an ensemble of 10 ESNs is used for this purpose.



(a) Loss training history

(b) NMSE training history

Figure 8.12: Typical RNN teacher-forced training curves.

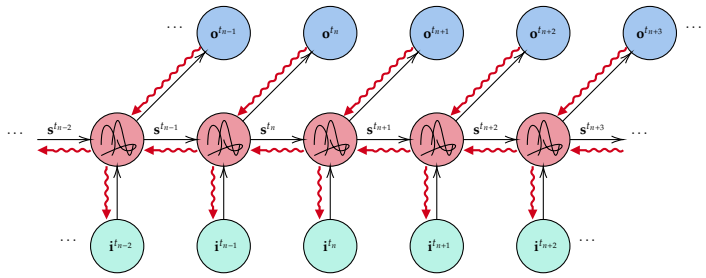


Figure 8.13: Teacher-forced RNN training flow-chart, where $\mathbf{i}^{t_n}$ and $\mathbf{o}^{t_n}$ represent the input and output (respectively) vectors, $\mathbf{s}^{t_n}$ the RNN state being received by the cell, all at time instance t_n. The mauve cells represent the RNN cells and their internal structure, while the red squiggly arrows mark the direction of gradient-flow.

8.3.3. Layering and Layer Sizes

Based on the reports surveyed, two trends are observed (Figure 8.14). First, the layer-sizes of the BPTT-trained RNNs (GRUs, LSTMs) are of the order of 10^1 times the dimensionality of the data being modelled. Second, the layer-sizes of the ESNs are of the order of 10^2 times the dimensionality of the data

being modelled. Based on these observations, a strategy for picking the layer sizes is decided as follows:

$$\text{RNN layer size} = n^r \times n^{ls}, \tag{8.12}$$

where n^{ls} is the number of latent states being modelled, and n^r the factor of multiplication. Based on this, the values of n^r chosen for this project are given in Table 8.3. Note that only the GRU and ESN are tested at multiple layer sizes, and all network types are compared at the highest n^r values against each other.

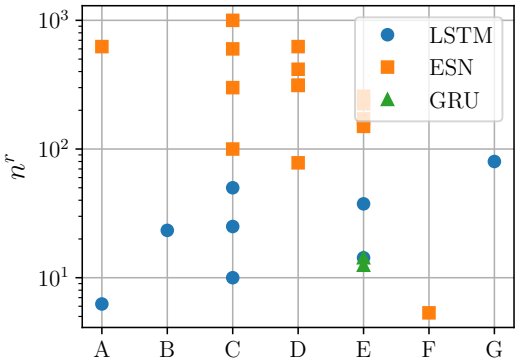


Figure 8.14: n^r values computed for the various reported studies. *A* is [13], *B* is [75], *C* is [73], *D* is [57], *E* is [74], *F* is [19], *G* is [55].

Network type	n^r values
ESN	$\{200, 500, 800\}$
GRU	$\{20, 50, 80\}$
LSTM	$\{80\}$
Simple RNN	$\{80\}$

Table 8.3: Chosen values of the multiplicative factor n^r.

RK-inspired Layering

Runge-Kutta methods are numerical techniques commonly used to solve ordinary differential equations (ODEs) with improved accuracy and stability. Drawing inspiration from these mathematical solvers, multi-layer RNNs are tested to adopt a similar hierarchical approach to model sequences of data. Note that in Section 7.1, each stage of the RK method essentially involves a weighted sum of $\Delta t\mathbf{f}(\cdots)$. In this spirit, the general layering architecture of the tested RK RNNs involves an initial RNN layer that expands the input into the RNN's hidden layer dimension ($n^r \times n^{ls}$) and subsequent applications of RK layers. Each RK layer maintains its owns hidden states, but shares the weights with all other RK layers. The RNN cell (representing the layer's weights and biases) they use is called the $RK\ Cell$. The weighted sums of the outputs of each of the layers results in the final output. These weights are taken directly from the different RK schemes. In essence, the $RK\ Cell$ learns to model the function $\Delta t\mathbf{f}(\cdots)$.

The effect of RK-inspired layering is tested on the GRU networks with the highest n^r values, and weights from the RK1 (Euler method), RK2 and RK4 schemes are tested. Figure 8.15 shows the flow-chart of the RK1 and RK2 inspired layering architectures. Note that the RK1-inspired network is simply a skip-layer, which is known to be beneficial for deep neural networks.

8.3.4. Hyper-Parameter Selection

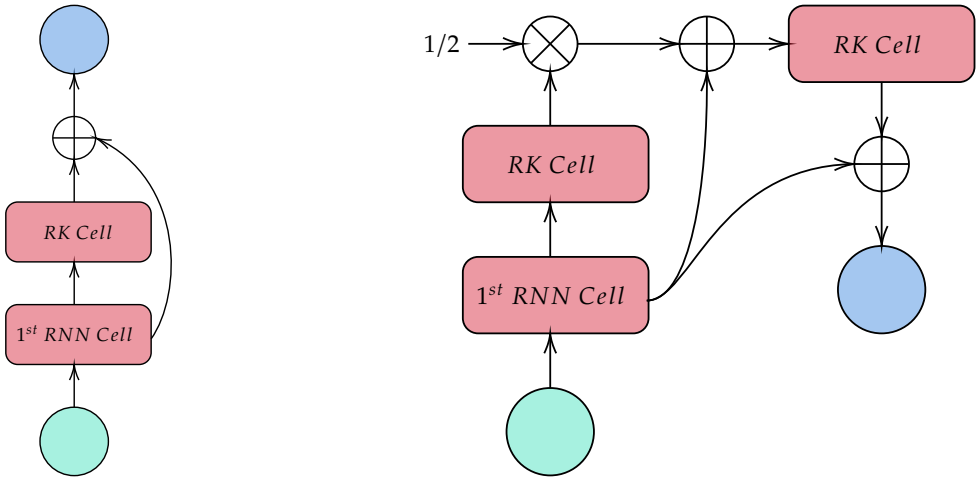
The ESNs and the BPTT-trained RNNs, having been built using different philosophies, possess different and separate sets of hyper-parameters. All of these are optimized using Bayesian optimization, as outlined below (similar to subsection 8.2.5). For this section, the objective-to-minimize for the optimizer is not the MSE of the reconstructed data, rather it is the negative of the 50^{th} percentile of the prediction horizons[3]. This effectively maximizes the prediction accuracy of the networks.

ESNs

ESNs depend on 6 hyper-parameters, listed in Table 8.4. The first five are optimized using Bayesian optimization, and their ranges given in Table 8.5. The degree of connectivity represents the average number neurons each reservoir neuron is connected to. Lukoševičius [50] reports this to not have much of an effect on the ESN's performance, and recommends setting it to a small constant value.

Fifty initial points are chosen, using the same space-filling curve to cover the defined domain. Twenty additional iterations are run for the optimizer to maximize the prediction horizons. The

[3]Computed using 50 initializations from the testing data-set.



(a) Flow-chart for RK1 inspired (simple skip layer) RNN network.

(b) Flow-chart for RK2 inspired RNN network.

Figure 8.15: Flow-charts depicting the RK-inspired layering architectures. Note that the weights are shared amongst the RK-cells.

hyper-parameters are optimized for the ESN with the smallest n^r value, and the found values are used for the higher n^r cases [50].

Hyper-parameter	Selection method	Value
input scaling, ω_{in}[4]	Bayesian optimization	*problem dependent*
reservoir spectral radius, ρ_{res}	Bayesian optimization	*problem dependent*
leaking rate, α	Bayesian optimization	*problem dependent*
noise, f_{noise}	Bayesian optimization	*problem dependent*
regularization, λ_{reg}[5]	Bayesian optimization	*problem dependent*
degree of connectivity	constant, pre-set	3

Table 8.4: Hyper-parameters and their selection methods.

Hyper-parameter	Bounds	Prior
input scaling, ω_{in}	$[0.1, 2.5]$	uniform
reservoir spectral radius, ρ_{res}	$[0.2, 1.2]$	uniform
leaking rate, α	$[0.5, 1.0]$	uniform
noise, f_{noise}	$[10^{-4}, 10^{-1}]$	log-uniform
regularization, λ_{reg}	$[10^{-9}, 10^{-3}]$	log-uniform

Table 8.5: ESN hyper-parameters, their imposed bounds and assumed priors.

BPTT trained RNNs

The hyper-parameters here too are selected in the same manner as in subsection 8.2.5. The different hyper-parameters and their selection strategies are given in Table 8.6. The bounds on the hyper-parameters λ_{reg} and f_{noise} are given in Table 8.7. Similar to the ESN case, the hyper-parameters are found for the smallest n^r value networks and then reused for the higher n^r cases (if applicable).

[4] denoted by a in subsection 4.3.4
[5] denoted by β in subsection 4.3.4

Hyper-parameter	Selection method	Value
learning rate(s)	constant, pre-set	$\{10^{-3}, 10^{-4}, 10^{-5}\}$
λ_{reg}	Bayesian optimization	*problem dependent*
f_{noise}	Bayesian optimization	*problem dependent*
early stopping patience	constant, pre-set	10
maximum epochs (per round)	constant, pre-set	200
number of rounds	constant, pre-set	3
batch size	constant, pre-set	32

Table 8.6: Hyper-parameters and their selection methods.

Hyper-parameter	Bounds	Prior
λ_{reg}	$[10^{-9}, 10^{-3}]$	log-uniform
f_{noise}	$[10^{-4}, 10^{-1}]$	log-uniform

Table 8.7: Hyper-parameters, their imposed bounds and assumed priors.

8.4. Combined AE-RNN

So far, the RNNs have been trained in a single-step prediction mode, which differs from the objective on which the combined AE-RNN models are evaluated - multi-step prediction. To this end, the combined AE-RNN model is trained in an auto-regressive setting, as described in the following section.

8.4.1. Training

Auto-regressive training

Auto-regressive training in the context of RNNs involves a unique approach, where instead of operating on the ground-truth data, the network is trained to predict trajectories by treating its own outputs as inputs for the next time-step. This self-predictive mechanism imbues the RNN with the ability to better estimate future trajectories recursively. It not only helps the RNNs learn to operate on their imperfect outputs rather than the true data values, but also allows the auto-encoder and RNN to *enmesh* and account for each others' errors. A flow-chart representing this is provided in Figure 8.16.

A sequential training protocol is adopted, as done in [75], wherein the network is trained to predict a progressively increasing number of output steps. The network is 'warmed up' with one Lyapunov time's worth of time-steps to properly intitialize the RNNs' internal states. Since the system is chaotic and the trajectories are bound to diverge, in [75] the authors attach a progressively decreasing weight to each time-step so as to not have the training focus on latter time-steps, and this practice too is adopted.

An additional penalty inspired from [77] is added to the loss function, in terms of the Frobenius norm of the difference between the covariance matrices of the outputs. This is a penalty on a statistical property of the output, and thus still model-independent. It was observed to aid in AR training in the initial experiments and was suitably used throughout the project then.

$$J(\boldsymbol{\theta}) = \text{MSE}\left(\mathbf{x}^{\text{true}}, \mathbf{x}^{\text{predict}}\right) + \lambda_{\text{reg}} L^2_{penalty}(\boldsymbol{\theta}) + \lambda_{\text{covmat}} L^{\text{Fr}}_{penalty}\left(\underline{C}^{\text{true}} - \underline{C}^{\text{predict}}\right), \tag{8.13}$$

where $\underline{C}$ is the covariance matrix of the 'snap-shots', as defined in Equation 5.5 in subsection 5.1.1, and $L^{\text{Fr}}_{penalty}(\cdot)$ the Frobenius norm. The hyper-parameter λ_{covmat} is not explicitly optimized, and chosen such that the added penalty is of the same order as the L^2 regularization penalty.

Lastly, in initial experiments it was observed that for higher output time-steps the training would not proceed as expected and the network would end up learning nothing, with prediction performance degrading. Upon closer inspection, this was attributed to unstable back-propagated gradients characterized by high values of the gradient's norm. The vanishing/exploding gradient problem is well known in the context of training RNNs [32]. This is dealt with by scaling the gradients for successive output time-step training rounds, with the maximum value of the gradient's norm from the previous time-step round.

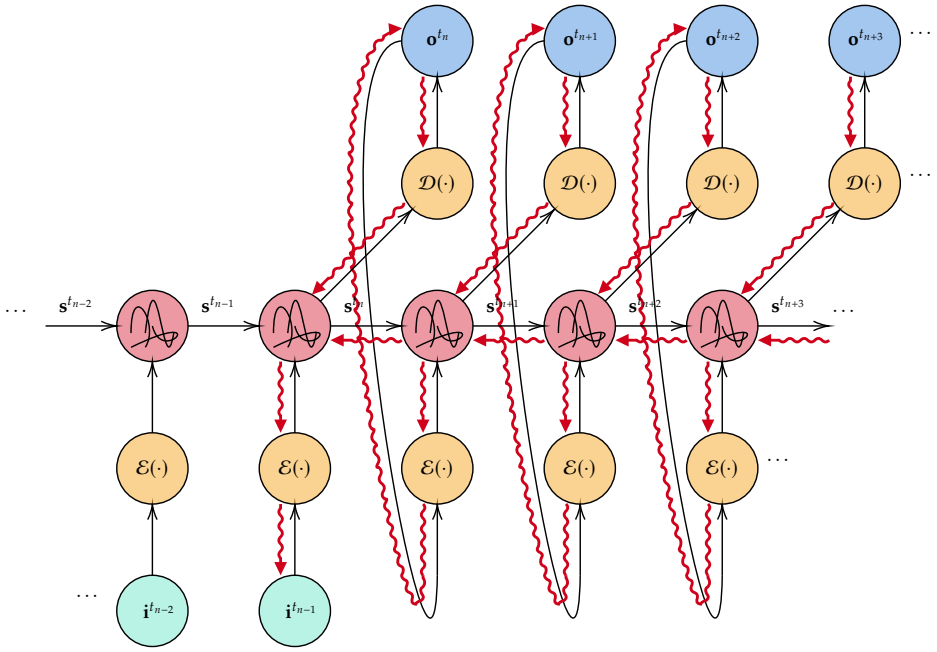


Figure 8.16: Auto-regressive training flow-chart of the combined AE-RNN. $\mathcal{E}(\cdot)$ and $\mathcal{D}(\cdot)$ represent the application of the encoder and decoder networks (respectively), and the remaining symbols are defined as in Figure 8.13

8.4.2. Long-term Prediction Statistics

In addition to testing the short-term predictive capabilities (by means of the prediction horizons), the long-term statistical behaviour of the AE-RNN models w.r.t. the true data is also analyzed. This is done via the Wasserstein distance (also known as the Kantorovich-Wasserstein metric) [34, 4, 42, 7]. This metric measures the similarity between two probability distributions, and the empirical distributions of the *attractors* obtained from the long-term predictions are compared against those obtained from the true data using it. It is defined in terms of a linear programming problem, wherein one needs to compute the *optimal amount of flow* ($\pi_{i,j}$) required to transform one distribution into the other, and is defined as:

$$W_p(\mu,\nu) := \left(\min_{\pi \in U} \sum_{i=1}^{n} \sum_{j=1}^{m} d(\mathbf{x}_i, \mathbf{y}_j)^p \pi_{ij} \right)^{\frac{1}{p}}, \tag{8.14}$$

where W_p is the p^{th} order Wasserstein distance, μ, ν are the two probability distributions being compared ($\sum_{i=1}^{n} \mu_i = 1$ and $\sum_{i=1}^{m} \nu_i = 1$), $d(\mathbf{x}_i, \mathbf{y}_j)$ is suitably defined distance metric between the two elements $\mathbf{x}_i$ and $\mathbf{y}_j$ and the set U is defined as:

$$U := \left\{ \begin{array}{ll} \sum_{j=1,\ldots,m} \pi_{ij} \leq \mu_i & i = 1,\ldots,n \\ \sum_{i=1,\ldots,n} \pi_{ij} \geq \nu_j & j = 1,\ldots,m \\ \pi_{ij} \geq 0 & i = 1,\ldots,n, j = 1,\ldots,m. \end{array} \right\} \tag{8.15}$$

As can be observed, computing this metric for multi-dimensional distributions is computationally challenging, and its efficient computation is still an active area of research [7, 4]. However, for 1D PDFs, the W_1 metric has an easy-to-compute analytic solution [72, Theorem 2.18 on p.74]:

$$W_1(\mu,\nu) = \int_{\mathbb{R}} |F_\mu(x) - F_\nu(x)| \mathrm{d}x, \tag{8.16}$$

where F_μ and F_ν are the cumulative distribution functions corresponding to the probability distributions μ and ν. The `scipy` library provides a standard implementation of this in the function `scipy.stats.wasserstein_distance`, which is used in the present project.

The quantities used to track the attractor trajectories are defined in Table 8.8. These are the quantities whose individual PDFs are compared using the W_1 metric, and presented in Chapter 10.

System	**Quantities used to represent the attractor**
Lorenz '63	x_1, x_2, x_3
Charney-DeVore	$x_1, x_2, x_3, x_4, x_5, x_6$
Kuramoto-Sivashinsky	turbulent kinetic energy (TKE) and mean dissipation rate (D)
Kolmogorov Flow	turbulent kinetic energy (TKE) and mean dissipation rate (D)

Table 8.8: Quantities used to track attractor trajectories for different systems.

8.5. Software and Libraries

This project is written in `Python` 3.8, with `NumPy`, `SciPy` and `Matplotlib` being the major libraries used for numerical analysis and plotting. The well-known `Tensorflow` [54] library with the Keras backend is employed for the machine learning aspects. The `scikit-optimize` library is used for the Bayesian optimization of the hyper-parameters.

PART IV
Results and Discussion

9

Dimensionality Reduction Using Auto-encoders

This chapter details the outcomes of the data compression achieved by means of an autoencoder applied to the different chaotic systems. The research questions tackled here those pertaining to the auto-encoders (item 1 from Section 2.2 of Chapter 2), specifically:

1. *Does the incorporation of a contractive loss improve accuracy?*
2. *Can the model parameters be incorporated into the AE architecture in a meaningful way such that it produces a multi-regime AE?*
3. *For the CNN-based AE, how much does the incorporation of self-attention affect accuracy?*

Also compared against these results is the traditional POD model reduction technique, both in terms of the computed reduced modes and the total variance captured. All the metrics presented here are computed in the testing data-set of their respective cases. Unless specified otherwise, the analysis is conducted for the single-parameter-set case.

9.1. The Lorenz '63 System

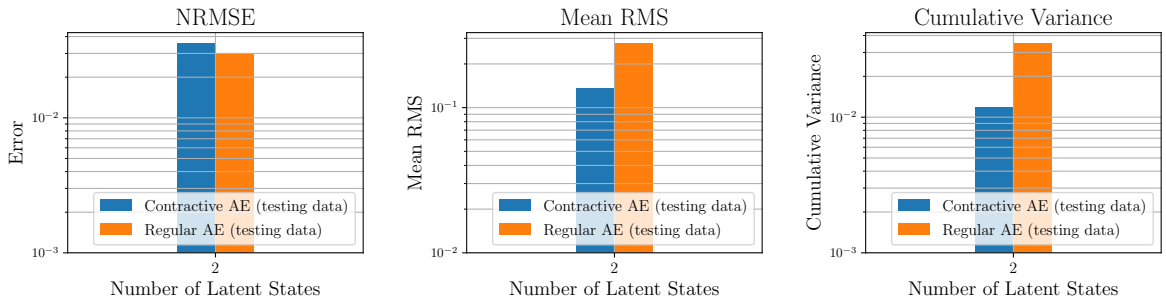
The Lorenz '63 system is fairly simple in complexity. The KY dimension and MLE for the different parameter-cases are given in subsection A.1.1. Note that the maximum KY dimension for any of the tested cases is 2.06, thus making the value $\lceil 2D_{KY} \rceil = 5$ greater than the dimensionality of the original data-set itself. Nonetheless, the auto-encoders are constructed with the latent space's dimensionality chosen to be 2. The compression exercise undertaken here is more to familiarize oneself with the auto-encoders and POD, and test the idea of a multi-regime auto-encoder.

By applying Bayesian optimization to the hyperparameters f_{noise} and λ_{reg}, we obtain the values 2.68×10^{-4} and 10^{-7} (respectively). The evolution of Mean Squared Error (MSE) across iterations is visualized in Figure B.1a. Interestingly, the optimal value is discovered within the initial point-set, and despite exploring the search domain, the algorithm fails to find any further improvements.

9.1.1. Contractive Loss

With the noise and regularisation hyper-parameters fixed, the contractive loss constant $\lambda_{\text{contractive}}$ is searched for. The iteration-wise evolution of the MSE is shown in Figure B.1b, and the optimal value is computed to be 4.34×10^{-4}. The NRMSE (reconstruction error), the mean RMS values and the cumulative variance of the latent states, for both the cases with and without a contractive loss are shown in Figure 9.1.

Clearly, the reconstruction error is marginally worse than if no contractive loss is used, and the mean RMS and cumulative variance are less than half of their non-contractive-loss counterpart. This suggests that the contractive penalty appears to force the network into learning latent space representations



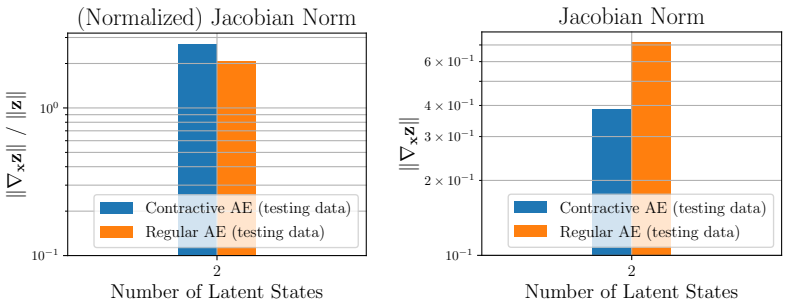
(a) Reconstruction error, NRMSE. **(b)** Mean RMS values of the latent states. **(c)** Cumulative variance of the latent states.

Figure 9.1: Various computed metrics for the auto-encoders with and without contractive loss.

with smaller norm values, as opposed to any meaningful additional contractions. This effect is further clarified when the penalty term is itself analyzed a bit more:

$$\underbrace{\|\nabla_{\mathbf{x}}\mathbf{z}\|}_{\text{Jacobian norm}} = \underbrace{(\|\nabla_{\mathbf{x}}\mathbf{z}\| \;/\; \|\mathbf{z}\|)}_{\substack{\text{normalized} \\ \text{Jacobian norm}}} \times \|\mathbf{z}\| \tag{9.1}$$

The Jacobian's norm and the *normalized* norm[1] as defined above are both shown in Figure 9.2. As can be seen here, the contractive penalty succeeds in penalizing the norm of the Jacobian (Figure 9.2b), and it is indeed lower for the contractive auto-encoder. However since the normalized norm (Figure 9.2a) is in fact marginally higher, it can be concluded that the reduction in the penalty term is brought about by the reduction in the norm of the latent state variable $\mathbf{z}$ (Figure 9.1b) rather than any meaningful changes to the latent manifold. The reason this is important is that the latent states are standardized once again before being fed into the RNNs, and hence any reductions brought about by changes in $\|\mathbf{z}\|$ will get cancelled out by said re-scaling.



(a) Normalized norm of the Jacobian. **(b)** Norm of the Jacobian.

Figure 9.2: Normalized and original values of the mean Jacobian norm, for the auto-encoders with and without contractive loss.

Owing to the above observations, and the fact that the reconstruction error is not bettered by the addition of the contractive penalty, the regular non-contractive auto-encoder is used for all further studies for this system.

9.1.2. AE and POD Comparison

In this section, the dimensionality reduction achieved by means of the auto-encoder is compared against that achieved by the POD method. The reconstruction errors are plotted and compared in Figure 9.3 and Table 9.1. Figure 9.4 and Table 9.2 show the difference in the captured statistical information by the

[1]Note the term norm here refers to the Frobenius norm.

auto-encoder and the POD model, via the cumulative variances:

$$\eta = \frac{\sum_i \left(\sigma_i^{\text{reconstructed}}\right)^2}{\sum_i \left(\sigma_i^{\text{true}}\right)^2} \tag{9.2}$$

The auto-encoder, by virtue of being a non-linear method, can capture slightly more of the cumulative variance as compared to the linear POD method. The reconstruction error, on the other hand, is almost an order of magnitude lower. This can also be observed in Figure 9.6c, where the AE can be seen to be accurately reconstructing the eigenvalues of the covariance matrix, but the POD method by definition only models the first two. This re-inforces earlier comments on how even statistically insignificant modes are important when modelling chaotic systems.

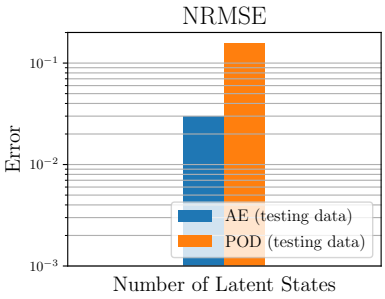


Figure 9.3: Reconstruction error (NRMSE).

Latent Space Dimension	AE	POD
2	2.86×10^{-2}	1.55×10^{-1}

Table 9.1: Reconstruction error (NRMSE).

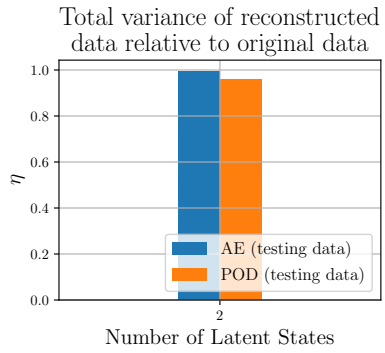


Figure 9.4: Total variance captured by the reconstructed data, w.r.t. the original data.

Latent Space Dimension	AE	POD
2	99.66%	96.11%

Table 9.2: Total variance captured by the reconstructed data, w.r.t. the original data.

Mode comparison

The analysis detailed in subsection 8.2.6 is carried out here, and the decoded principal directions of the latent space compared with the principal directions of the original data. Figure 9.5 shows the respective modes, and Figure 9.6a shows a heat map of the correlation matrix representing the relationships between the two cases.

As can be observed, the first decoded vector is simply the flipped version of the first POD mode, implying that it is simply the same vector but in the opposite direction. This can also be observed in the deeply negative correlation coefficient between the two, and the almost zero correlation coefficients it possesses with the other principal directions. The second decoded mode is harder to analyse, and would appear to be a mixture of the remaining POD modes, and as much can be gleaned from its correlation coefficients.

Lastly, note that the first principal direction in the latent space is the dominant one, and accounts for about 76% of the total variance of the latent states (Figure 9.6b).

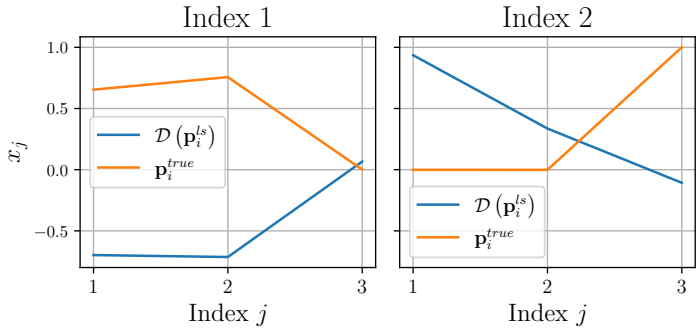
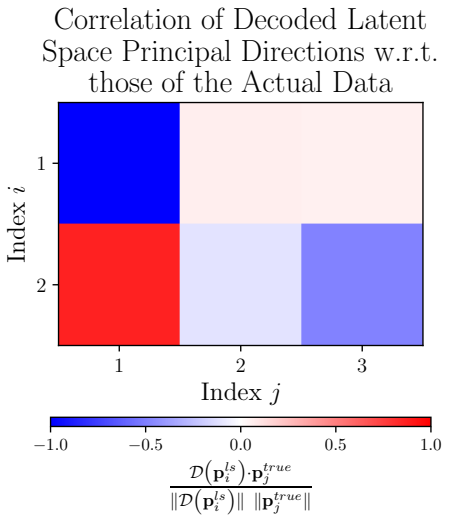
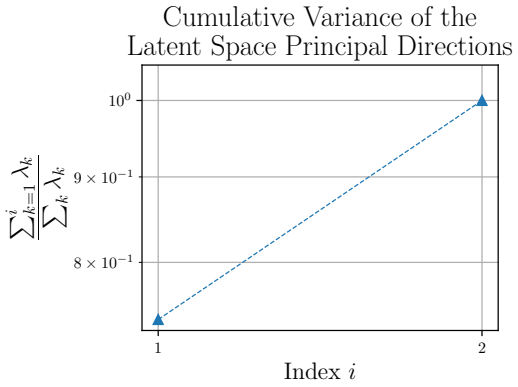


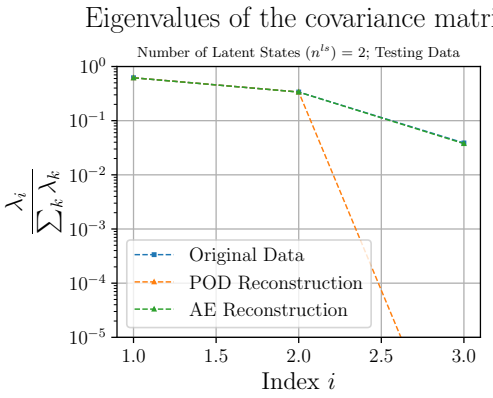
Figure 9.5: Decoded latent space principal directions, and POD principal directions



(a) Correlation between decoded latent space principal directions, and POD principal directions



(b) cumulative variance of the latent space principal directions



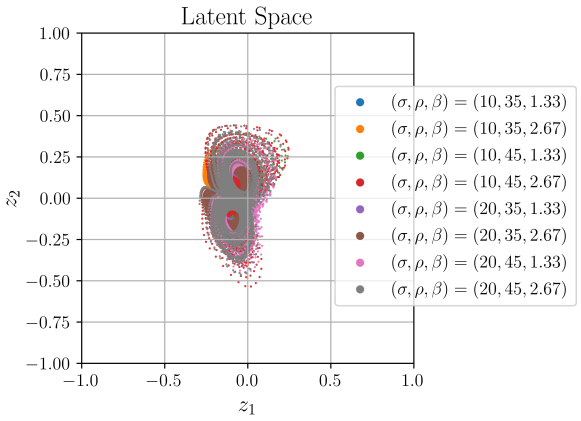
(c) Eigenvalues of the covariance matrix of the reconstructed data.

Figure 9.6: Different metrics representing the comparisons between the auto-encoder and the POD method.

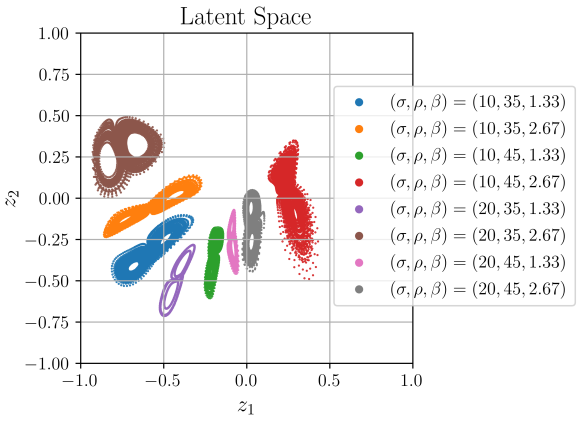
9.1.3. Multi-regime AE

As detailed in subsection 8.2.1, a multi-regime auto-encoder is also constructed, by supplying the regime parameters σ, β, ρ to the auto-encoder. The latent space of the auto-encoder with the parameters supplied and that without can be seen in Figure 9.7. Clearly, the different regimes are well separated in the case of the supplied parameters, and this physical separation in the latent space also lends itself

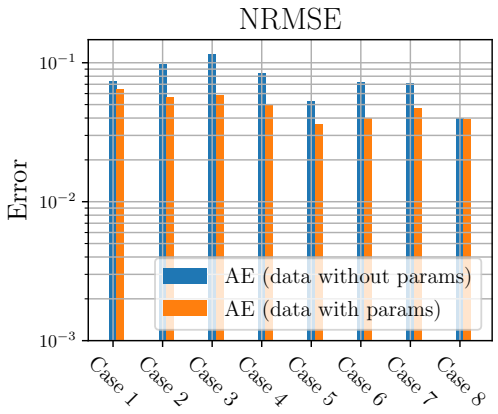
to generally lower errors (Figure 9.7c). Further, this physical separation in the latent space can be an implicit signal to the time-series modelling scheme, should one choose to pursue it. Note also, that the individual reconstruction errors here are slightly higher than that of the single parameter-set case (Table 9.1), while being of the same order and lower than the POD reconstruction error. This indicates that this multi-regime sharing of the latent space is not a free-lunch, and a marginal performance penalty has to be paid.



(a) Latent space of the auto-encoder that operates on only the system state.



(b) Latent space of the auto-encoder that operates on data with the parameters appended to the system state.



(c) Reconstruction errors for the two tested multi-regime schemes.

Figure 9.7: Multi-regime auto-encoders.

9.2. The Charney-DeVore System

The CDV system is a step up from the Lorenz '63 system in terms of complexity, but is still an ODE system. Its KY dimension and MLE are given in subsection A.1.2. The maximum KY dimension (out of the two parameter-sets), which happens to be that of the single-parameter case, is $D_{KY} = 2.32$. Thus, based on the earlier presented reasoning, the latent space's dimensionality is picked to be $n^{ls} = \lceil 2D_{KY} \rceil = 5$.

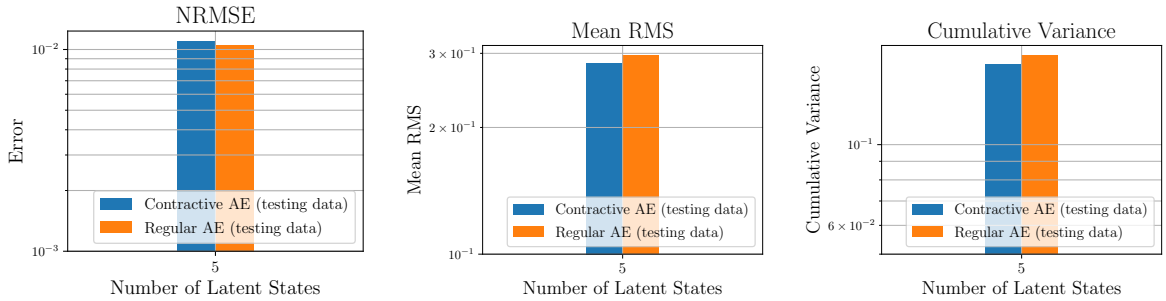
The Bayesian optimization of the hyper-parameters f_{noise} and λ_{reg} yields the values 1.93×10^{-3} and 3.73×10^{-7} (respectively). The iteration-wise evolution of the MSE is plotted in Figure B.2a. Much like the Lorenz '63 case, the optimal value is found in a point from the initial point-set, and while the algorithm explores the search domain, it is unable to find any better values.

9.2.1. Contractive Loss

The search for the optimal value of $\lambda_{\text{contractive}}$ yields 1.68×10^{-6}, and the iteration-wise evolution of the MSE is plotted in Figure B.2b. The reconstruction error (NRMSE), mean RMS values and cumulative variance of the latent states are plotted in Figure 9.8. The normalized norm and regular norm of the Jacobian $\nabla_{\mathbf{x}}\mathbf{z}$ too are observable in Figure 9.9a. The trends are the same as in the Lorenz '63 system's

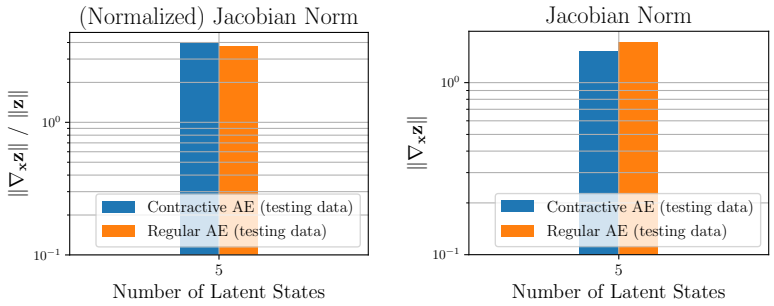
case, and the same analysis and observations from subsection 9.1.1 apply here too.

Note that since the optimal value of $\lambda_{\text{contractive}}$ is quite small in magnitude, smaller even than the Lorenz '63 case, the effect this penalty has on the network is also more negligible. Where, in the previous case, there was a difference of just over a factor of two between the mean RMS values and the cumulative variances, here that difference is only marginal. The same can be said of all the plotted metrics. For all these reasons, here too the proceeding analyses are conducted with a regular non-contractive auto-encoder.



(a) Reconstruction error, NRMSE. **(b)** Mean RMS values of the latent states. **(c)** Cumulative variance of the latent states.

Figure 9.8: Various computed metrics for the auto-encoders with and without contractive loss.



(a) Normalized norm of the Jacobian. **(b)** Norm of the Jacobian.

Figure 9.9: Normalized and original values of the mean Jacobian norm, for the auto-encoders with and without contractive loss.

9.2.2. AE and POD

As done previously, the data compression achieved by the auto-encoder is compared with that achieved through POD. Since the CDV system is of higher dimensionality than the Lorenz '63 system, latent spaces with dimensionality from $\lfloor D_{KY} \rfloor$ to $\lceil 2D_{KY} \rceil$ are tested. The reconstruction errors can be observed in Figure 9.10 and Table 9.3, while the captured total variance in Figure 9.11 and Figure 9.11.

The reconstruction errors clearly decrease with increasing latent space dimensionality, as expected, and are roughly an order of magnitude smaller than their POD counterparts. All the latent spaces are able to capture over 99% of the total variance of the original data-set, while for the POD cases this number fast decreases with decreasing principal modes. Worth noting here is the fact that even though the captured total variance might not vary much, the relative change in reconstruction error is not marginal.

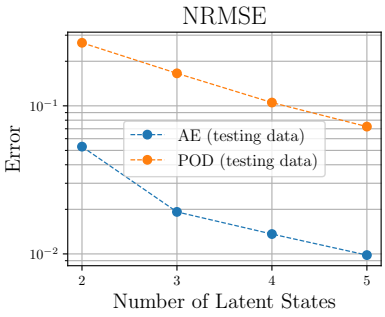


Figure 9.10: Reconstruction error (NRMSE).

Latent Space Dimension	AE	POD
2	5.30×10^{-2}	2.67×10^{-1}
3	1.92×10^{-2}	1.66×10^{-1}
4	1.36×10^{-2}	1.05×10^{-1}
5	9.82×10^{-3}	7.24×10^{-2}

Table 9.3: Reconstruction error (NRMSE).

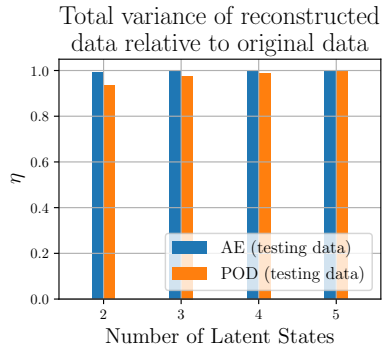


Figure 9.11: Total variance captured by the reconstructed data, w.r.t. the original data.

Latent Space Dimension	AE	POD
2	99.25%	93.35%
3	99.95%	97.60%
4	99.93%	98.85%
5	99.85%	99.56%

Table 9.4: Total variance captured by the reconstructed data, w.r.t. the original data.

Mode comparison

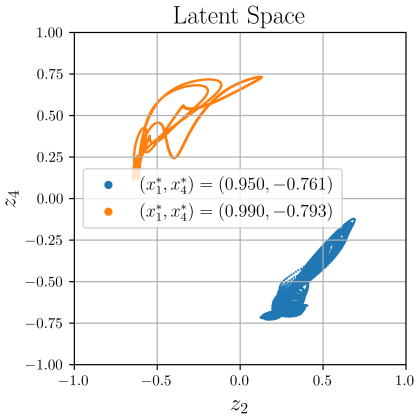
The decoded latent space principal directions and the POD principal directions are plotted in Figure 9.13. Note the prominent matching of the first modes. This can be explained by noting the primacy of the first POD principal mode (Figure 9.14c), which contains around 68% of the total variance of the original data. Hence, it stands to reason that this mode would be well represented in the latent space too; as much is observed.

The second POD principal mode accounts for roughly 25% of the total variance, and it too has reasonable overlap with the second decoded latent space principal direction. They resemble roughly flipped versions of each other, which can be seen in the significantly negative correlation coefficient that they share (Figure 9.14a). Lastly, the remaining decoded modes appear to be non-linear combinations of the POD principal modes, which is also reflected in their correlation coefficients.

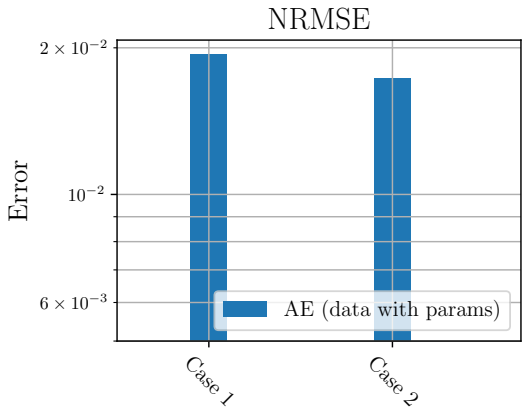
9.2.3. Multi-regime AE

The multi-regime auto-encoder is constructed, by supplying the regime parameters $(x_1^*, x_4^*, C, \beta, \gamma, b)$ to the auto-encoder. Since the parameter-supplied auto-encoder from subsection 9.1.3 worked better than the one without, this approach has been utilised for the present and all proceeding chaotic systems.

The projection of the latent states of the auto-encoder on to the $z_2 - z_4$ plane is shown in Figure 9.12, where the separation of the two regimes is clearly visible. The same observations as subsection 9.1.3 apply here too. Note the minimal increase in the reconstruction errors compared to the single-regime case, indicating the cost of sharing the latent space. Also on display here are the periodic and chaotic nature of the two regimes, where latent states corresponding to the periodic case appear to form a closed limit cycle while the ones corresponding to the chaotic case display more erratic structures. Despite the compression, the structures displayed here resemble those observed in the $x_1 - x_4$ plane of the original data (Figure 8.2), suggesting that the attractors get modelled accurately.



(a) Latent space of the auto-encoder that operates on data with the parameters appended to the system state.



(b) Reconstruction errors for the two tested multi-regime schemes.

Figure 9.12: Multi-regime auto-encoder.

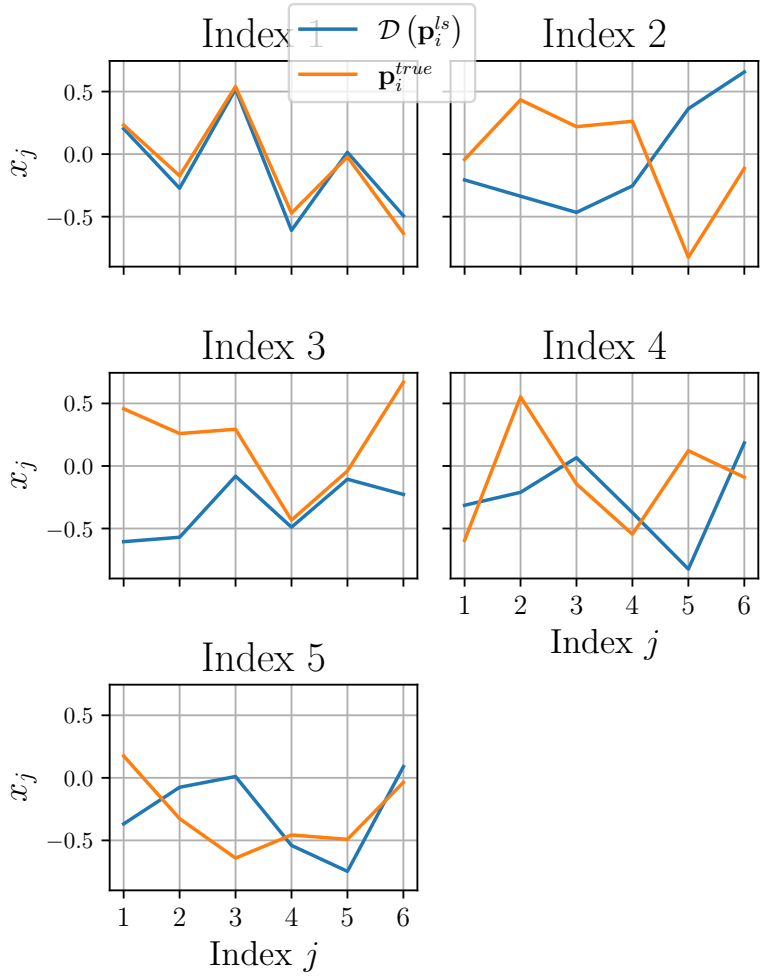
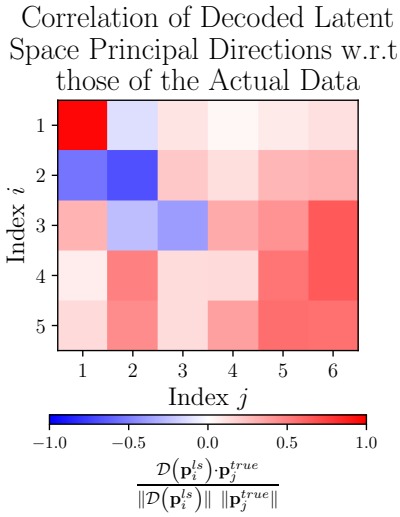
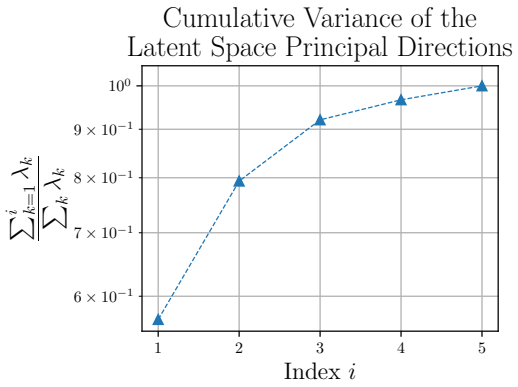


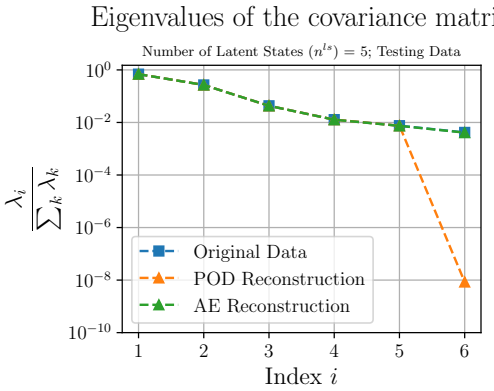
Figure 9.13: Decoded latent space principal directions, and POD principal directions



(a) Correlation between decoded latent space principal directions, and POD principal directions



(b) cumulative variance of the latent space principal directions



(c) Eigenvalues of the covariance matrix of the reconstructed data.

Figure 9.14: Different metrics representing the comparisons between the auto-encoder and the POD method.

9.3. The Kuramoto-Sivashinsky System

The KS system is the first PDE system to be analyzed, and as such has more intricate relationships to model. Its KY dimension and MLE are given in subsection A.1.3. The maximum KY dimension (out of the four parameter-sets), is $D_{KY} = 7.86$. Thus, the latent space's dimensionality is picked to be $n^{ls} = \lceil 2D_{KY} \rceil = 16$.

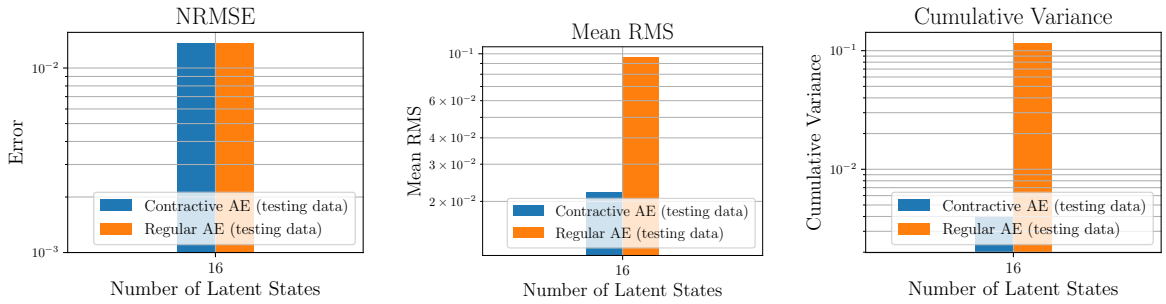
The Bayesian optimization of hyperparameters f_{noise} and λ_{reg} results in the values 2.68×10^{-4} and 1.00×10^{-7} (respectively). The plot in Figure B.3a illustrates the iterative evolution of Mean Squared Error (MSE). Similar to the Lorenz '63 and CDV cases, the optimal value is located within the initial point-set., and despite exploring the search domain, the algorithm fails to find any superior values.

9.3.1. Contractive Loss

The optimal value of $\lambda_{\text{contractive}}$ is found to be 1.69×10^{-4}, and the iteration-wise evolution of the MSE is plotted in Figure B.3b. The reconstruction error (NRMSE), mean RMS values and cumulative variance of the latent states are plotted in Figure 9.15. The normalized norm and regular norm of the Jacobian $\nabla_{\mathbf{x}}\mathbf{z}$ too are observable in Figure 9.16a. The trends are the same as in the Lorenz '63 and CDV systems' cases.

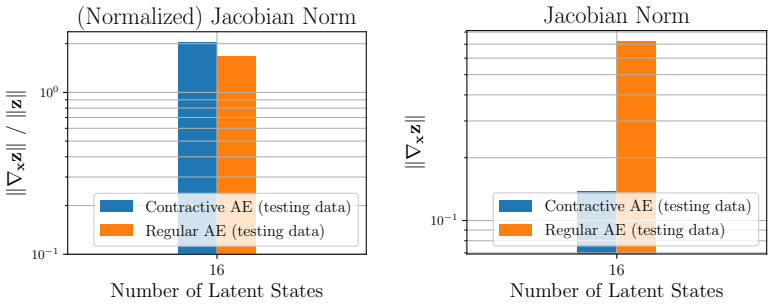
Worth noting here is the fact that even though the value of $\lambda_{\text{contractive}}$ is of the same order as that of the Lorenz '63 system, the much higher dimensionality of the KS system leads to it having a magnified effect on the Jacobian. The value of the Jacobian's norm is roughly 6 times smaller in the contractive auto-encoder's case, while the mean RMS of $\mathbf{z}$ is around 5 times smaller. The normalized norms of the Jacobian are quite similar in magnitude, implying that here too the reduction is owed to the reduction

in $\|\mathbf{z}\|$ and the same analyses from from subsection 9.1.1 are applicable. The reconstruction errors on the other hand match up quite well, with the contractive auto-encoder's being ever so slightly larger. For these reasons, once again the proceeding analyses are done using the regular non-contractive auto-encoder.



(a) Reconstruction error, NRMSE. **(b)** Mean RMS values of the latent states. **(c)** Cumulative variance of the latent states.

Figure 9.15: Various computed metrics for the auto-encoders with and without contractive loss.



(a) Normalized norm of the Jacobian. **(b)** Norm of the Jacobian.

Figure 9.16: Normalized and original values of the mean Jacobian norm, for the auto-encoders with and without contractive loss.

9.3.2. AE and POD

Latent spaces with dimensionality ranging from 7 (= $\lfloor D_{KY} \rfloor$) to 18 are tested, and compared against their POD method counterparts. The reconstruction errors are shown in Figure 9.17 and Table 9.5, while the captured total variance in Figure 9.18 and Table 9.6.

Observably, the reconstruction errors decrease for higher latent space dimensionality but then plateau out after $n^{ls} = 14$. The error is anywhere from 3 to 10 times lower (depending on n^{ls}), and is almost eight times smaller for the chosen $n^{ls} = 16$, for the auto-encoders compared to those of the POD reconstructions. The auto-encoders capture at least 98% of the total variance at all value of n^{ls}, whereas this sharply decreases for the POD cases with decreasing principal directions to account for. This is also observable in Figure 9.21c, wherein the auto-encoder's reconstruction is able to accurately reconstruct many more eigenvalues of the covariance matrix as opposed to the POD method. These observations showcase the auto-encoder's superior non-linear data-compression abilities in contrast to the linear POD method.

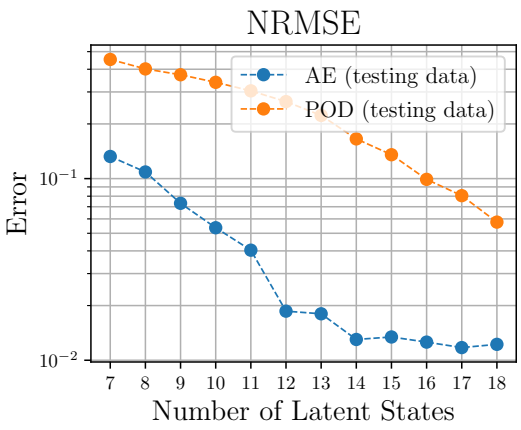


Figure 9.17: Reconstruction error (NRMSE).

Latent Space Dimension	AE	POD
7	1.32×10^{-1}	4.53×10^{-1}
8	1.09×10^{-1}	4.02×10^{-1}
9	7.31×10^{-2}	3.73×10^{-1}
10	5.37×10^{-2}	3.39×10^{-1}
11	4.04×10^{-2}	3.04×10^{-1}
12	1.86×10^{-2}	2.65×10^{-1}
13	1.90×10^{-2}	2.23×10^{-1}
14	1.30×10^{-2}	1.65×10^{-1}
15	1.34×10^{-2}	1.35×10^{-1}
16	1.26×10^{-2}	9.90×10^{-2}
17	1.17×10^{-2}	8.06×10^{-2}
18	1.22×10^{-2}	5.76×10^{-2}

Table 9.5: Reconstruction error (NRMSE).

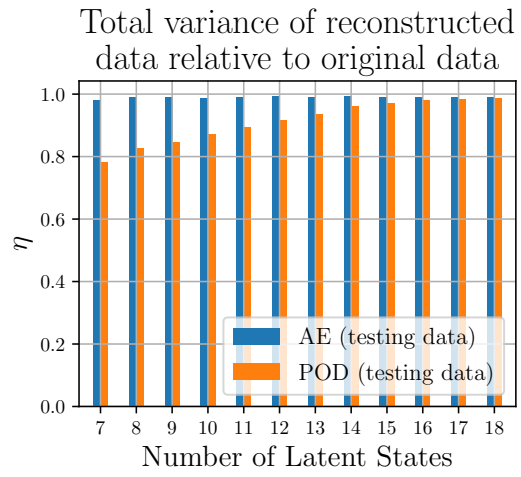


Figure 9.18: Amount of total variance captured by the reconstructed data, w.r.t. the original data.

Latent Space Dimension	AE	POD
7	98.04%	78.18%
8	98.97%	82.63%
9	99.10%	84.82%
10	98.82%	87.24%
11	99.09%	89.45%
12	99.24%	91.64%
13	99.18%	93.76%
14	99.25%	96.23%
15	99.21%	97.18%
16	99.17%	98.16%
17	99.17%	98.51%
18	99.21%	98.86%

Table 9.6: Amount of total variance captured by the reconstructed data, w.r.t. the original data.

Mode comparison

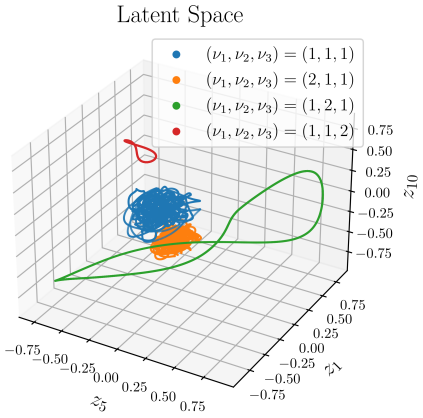
The first eight principal modes of the original data, and the decoded principal modes of the latent space are plotted in Figure 9.20. Observe that the POD modes of the original data appear to be regular sinusoids, each row representing a particular frequency and each column entry being a phase-shifted version of its neighbour. Counting the periods by simple visual inspection, the first row appears to have 4 periods, the second 3 periods, the third 5 periods and the fourth 6 periods. This pairing is also observable in Figure 9.21c, where the eigenvalues occur in pairs, indicating the equal importance that is attached to these mode pairs.

A heat map for the correlation matrix of all the decoded principal directions with the first twenty principal directions of the original data is shown in Figure 9.21a. Based on similar visual inspection, the decoded principal modes appear to possess *roughly* the same number of periods as their respective counterparts. This can be observed by the strong diagonal in the correlation matrix. Further, note that they are also clearly a combination of a set of different modes, with minor contributions from the modes that came before them, as depicted by the more intensely coloured triangle under the diagonal in Figure 9.21a.

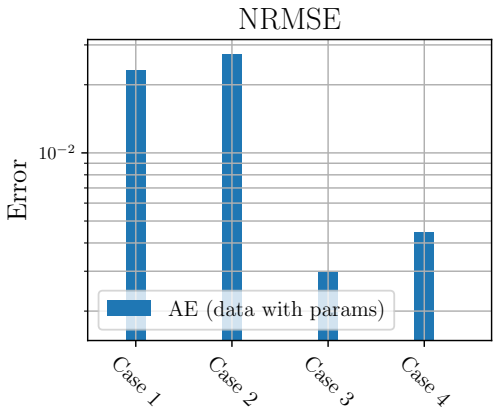
9.3.3. Multi-regime AE

The multi-regime auto-encoder is constructed in a manner consistent with previous approaches, by incorporating the regime parameters (ν_1, ν_2, ν_3) into the auto-encoder design. The projection of the latent states onto the $z_1 - z_5 - z_{10}$ space is visualized in Figure 9.19, clearly demonstrating the distinct separation of the four regimes. Essential to note here is that as observed previously, sharing the latent space comes at a cost, with the reconstruction errors increasing compared to the single-regime case (for the chaotic regimes). The reconstruction error for the $(\nu_1, \nu_2, \nu_3) = (1, 1, 1)$ case is almost twice that of the single regime auto-encoder.

As discussed in subsection 9.2.3, the latent states vividly exhibit the periodic and chaotic nature of the different regimes. For the periodic case, the latent states appear to form a closed limit cycle, while the chaotic case displays more irregular formations. Despite this, the chaotic regimes appear to be confined to their respective regions and orbiting a central trajectory.



(a) Latent space of the auto-encoder that operates on data with the parameters appended to the system state.



(b) Reconstruction errors for the two tested multi-regime schemes.

Figure 9.19: Multi-regime auto-encoder.

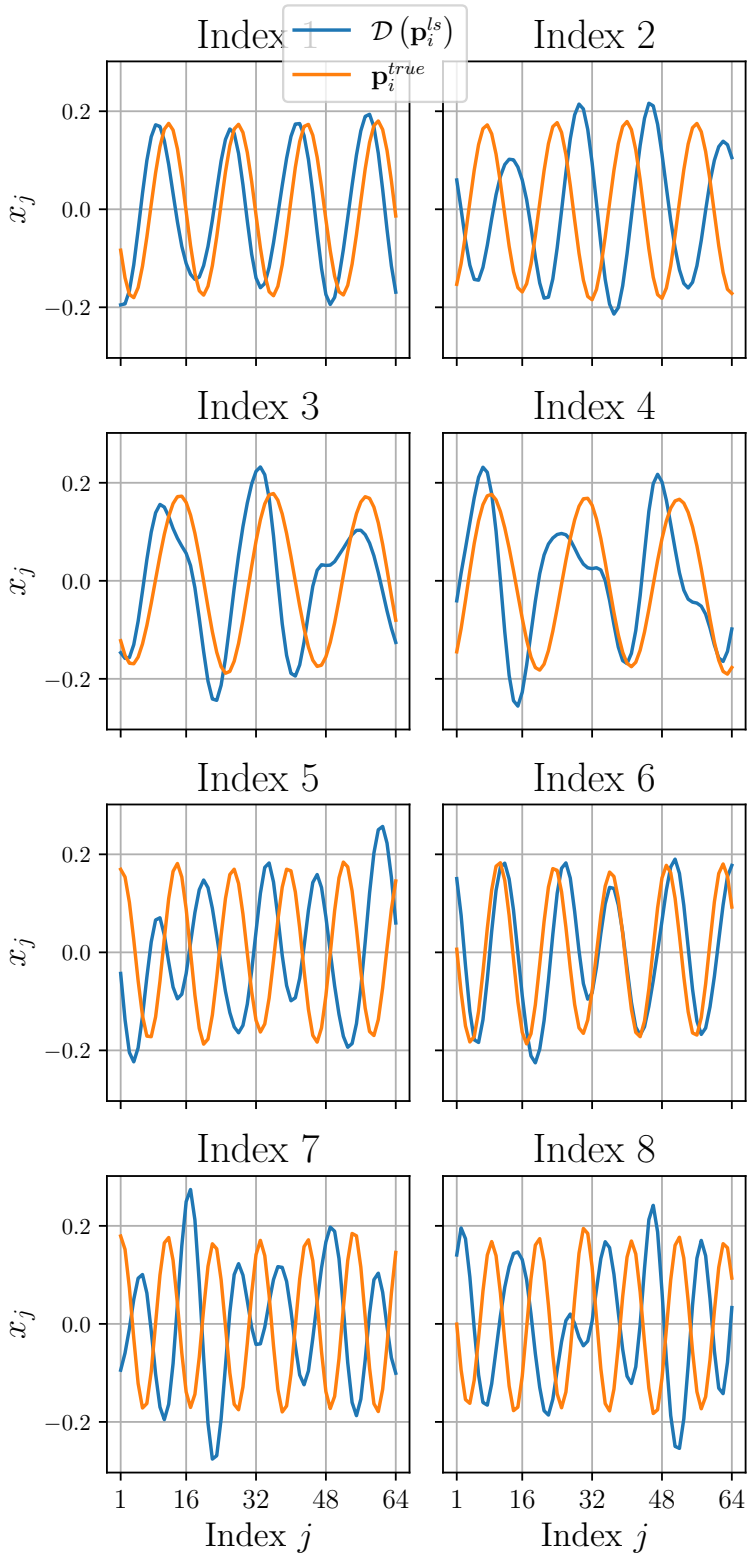
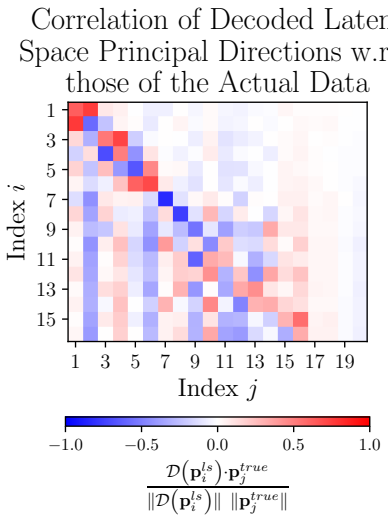
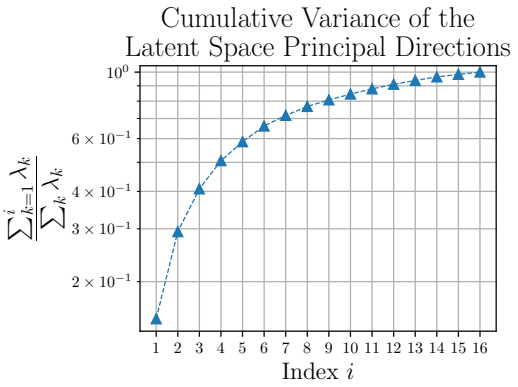


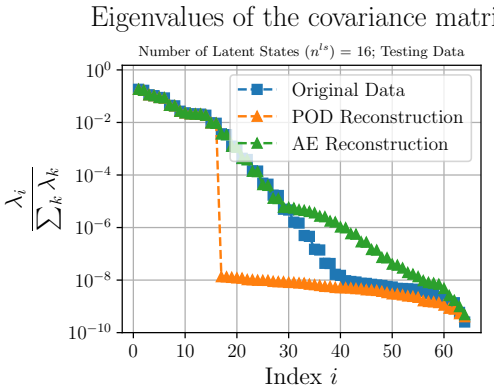
Figure 9.20: Decoded latent space principal directions, and POD principal directions



(a) Correlation between decoded latent space principal directions, and POD principal directions



(b) cumulative variance of the latent space principal directions



(c) Eigenvalues of the covariance matrix of the reconstructed data.

Figure 9.21: Different metrics representing the comparisons between the auto-encoder and the POD method.

9.4. Kolmogorov Flow

The 2D incompressible Navier-Stokes equations, or the Kolmogorov flow is the final system to be tested. Its KY dimension and MLE are given in subsection A.1.4, with the largest KY dimension being 12.19 (that of the $Re = 40$ case). Thus, $n^{ls} = \lceil 2D_{KY} \rceil = 25$ is the dimensionality of the ideal auto-encoder. However, owing to the convolutional auto-encoder construction (subsection A.2.1), the latent space is of dimensions $\{c^{ls}, 3, 3\}$, c^{ls} being the number of channels of the final encoder output. Thus, $c^{ls} = 3$ is chosen for a total of 27 latent states, in order to have the next closest n^{ls} value to 25.

The Bayesian optimization of the noise and regularization hyper-parameters yields the values $f_{\text{noise}} = 1.72 \times 10^{-2}$ and $\lambda_{\text{reg}} = 2.57 \times 10^{-7}$. The iteration-wise MSE evolution can be found in Figure B.4.

9.4.1. Contractive Loss

The contractive auto-encoders have failed to produce any meaningful improvements of the latent space in any of the past three chaotic systems, ODE and PDE. Based on this observation, it was chosen not to test contractive auto-encoders on this system since they were deemed unlikely to produce any significant findings.

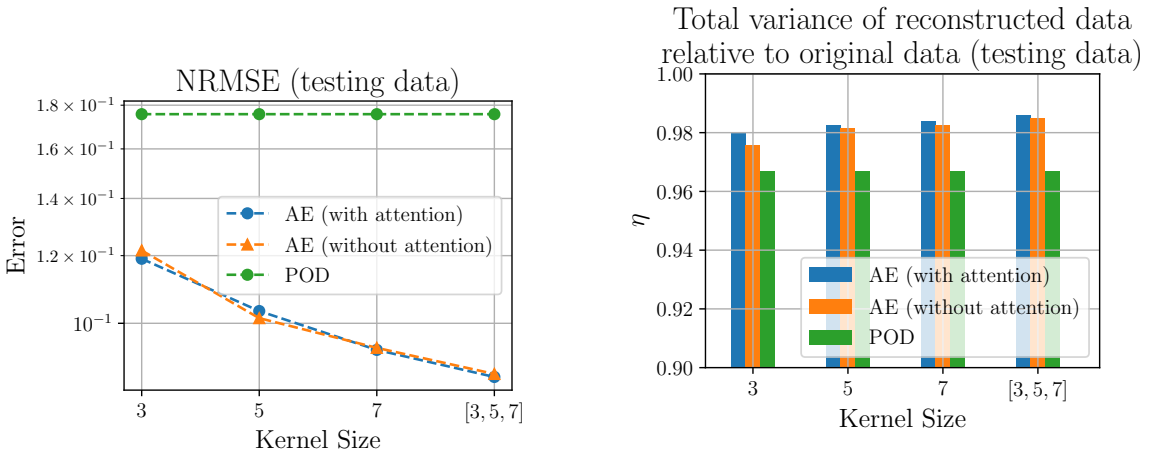
9.4.2. AE and POD

The auto-encoders are constructed using individual kernels of different filter sizes - 3×3, 5×5 and 7×7 - and finally a multi-scale one using all three kernel filter sizes. Additionally, variants of each with and without attention are constructed, and all of their performances compared against each other and

against the POD method. The reconstruction errors are shown in Figure 9.22a and Table B.1, while the captured total variance in Figure 9.22b and Table B.2. Note the captured variances are capped at the bottom at $\eta = 0.9$ to better see the differences.

Clearly, auto-encoders have lower reconstruction errors and higher captured total variance than the POD method, albeit not by a huge margin. The attention-incorporated auto-encoder has largely the same reconstruction errors as the regular one, but captures modestly greater amounts of total variance. In fact, the kernel size has a greater impact on these performance metrics than the use of self-attention. This can be attributed to two facts. Firstly, the problem being analysed is relatively simple in its nature. Attention-incorporation excels at modelling large scale spatial relationships that are otherwise neglected, such as the flow past cylinder and/or bluff bodies example used in [78]. In the present scenario there are no such devices/mechanisms in the flow that could induce such large-scale spatial contrasts and hence attention-incorporation is only marginally beneficial. Second, the *receptive field* of a neuron in the encoder network's output is by itself quite large when compared to the input spatial dimensions. The receptive field of a neuron is the size of the field of neurons in the input layer that influence its value. The receptive fields for kernel filter sizes 3×3, 5×5 and 7×7 are 31×31, 61×61 and 91×91[2] (respectively). As can be seen, the output neurons of the encoder network are already accounting for global information in this case by means of their receptive fields, and hence attention-incorporation does not offer in significant improvements.

Further note that the reconstruction errors in the present case are around the value 10^{-1}, whereas in all previously tested cases, the errors have been found to be closer to 10^{-2}. This could be pointing to the fact that the chosen layer configuration might not be optimal, and there might be a better layer and/or channel configuration than the one tested here. This relatively poorer reconstruction error will have an impact on the time-series prediction capabilities of the AE-RNNs as well.



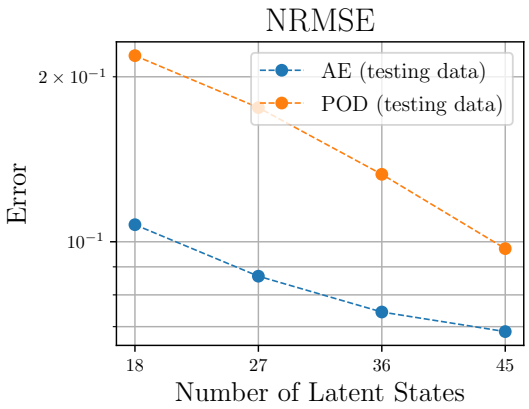
(a) Reconstruction error (NRMSE).

(b) Amount of total variance captured by the reconstructed data, w.r.t. the original data.

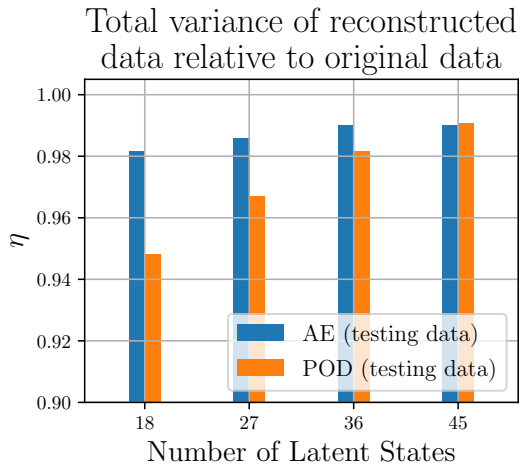
Figure 9.22: Performance metrics for the reconstructed data for a fixed latent space size and varying kernel filter size (Kolmogorov flow system, $Re = 40$).

Further tested (using the multi-scale attention-incorporated auto-encoder) are the effects of increasing latent space dimensions. The reconstruction errors are shown in Figure 9.23a and Table B.3 while the captured total variance in Figure 9.23b and Table B.4. Once again, the auto-encoders out-perform the POD method and have lower reconstruction errors (by a factor of 1/2), though those of the POD too seem to be fast decreasing with greater numbers of principal directions. In fact for the case with 45 latent states the POD method manages to capture slightly more of the total variance of the system (while still having a higher reconstruction error). This behaviour is similar to what has been observed earlier in subsection 9.3.2 wherein the auto-encoder's performance metrics plateau out but those of the POD method keep improving with added principal directions.

[2]These receptive fields are larger than the input's spatial dimensions, but that is only because periodic padding has been utilized.



(a) Reconstruction error (NRMSE).



(b) Amount of total variance captured by the reconstructed data, w.r.t. the original data.

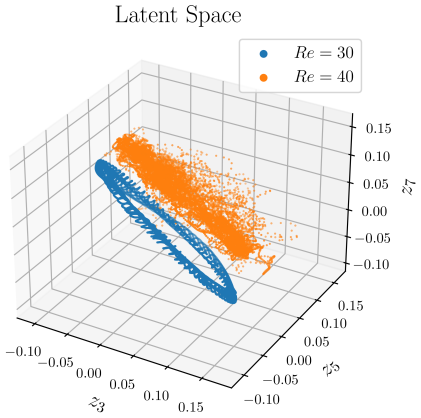
Figure 9.23: Performance metrics for the reconstructed data for a fixed kernel size (multi-scale) and varying latent space size (Kolmogorov flow system, $Re = 40$).

Mode comparison

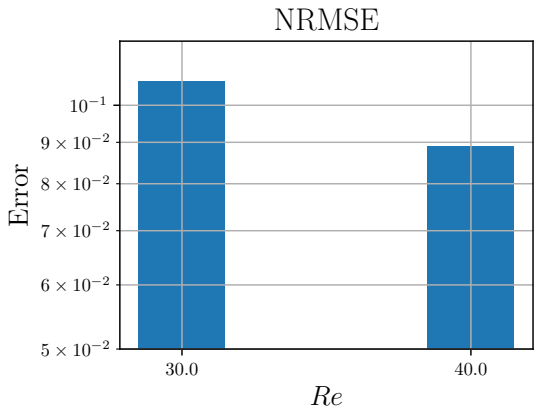
Here too the decoded principal modes of the latent space are compared with the principal modes of the original data. Note from Figure 9.25c the large vales of the first four modes; collectively they account for roughly 89% of the total variance of the original data. Hence, these four modes are well represented in the latent space as well. Note the first four prominent modes of the latent states in Figure 9.25b, which account for roughly 91% of the latent states' total variance. Naturally, there is decent agreement between their decoded counterparts and the first four POD modes of the original data. This can be observed in Figure 9.26 where the first 6 modes are shown. The u and v components appear to be the exact opposites for the first three modes, whereas the fourth decoded mode seems to align with the principal direction. Their collective correlations can be observed in the heat map of the correlation matrix and note the strong first four components on the diagonal. However, of interest amongst all these decoded modes is the fact that they possess additional vortical structures that are missing from the principal directions. This indicates some amount mixing and is indicative of the auto-encoder's non-linear compression.

9.4.3. Multi-regime AE

The convolutional multi-regime auto-encoder is constructed as described in subsection 8.2.1, by passing in an additional channel made up solely of the Reynolds number. The projection of the latent states onto the $z_3 - z_5 - z_7$ space is visualized in Figure 9.24a, displaying the distinct separation of the two flow regimes. As observed previously, sharing the latent space comes at a cost, with the reconstruction errors increasing compared to the single-regime case (for the $Re = 40$ regime). The reconstruction error for the $Re = 40$ case is slightly higher than that of the single regime auto-encoder. As discussed in in previous sections, the chaotic and quasi-periodic nature of the regimes is visible in the latent space as well.

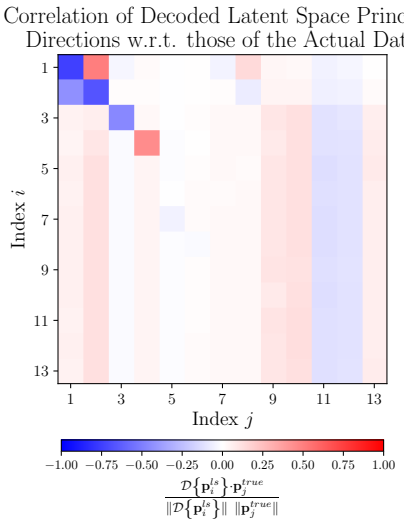


(a) Latent space of the auto-encoder that operates on data with the parameters appended to the system state.

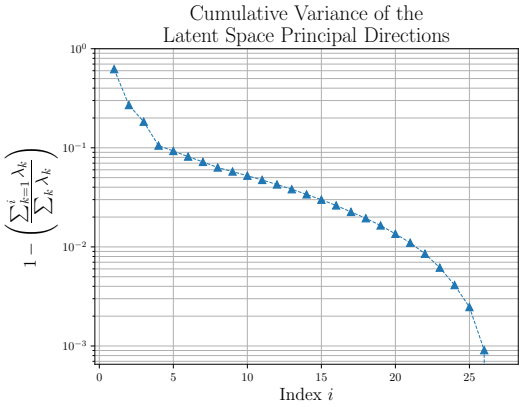


(b) Reconstruction errors for the two tested multi-regime schemes.

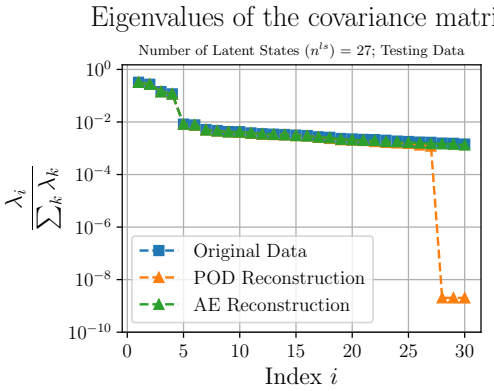
Figure 9.24: Multi-regime auto-encoder.



(a) Correlation between decoded latent space principal directions, and POD principal directions

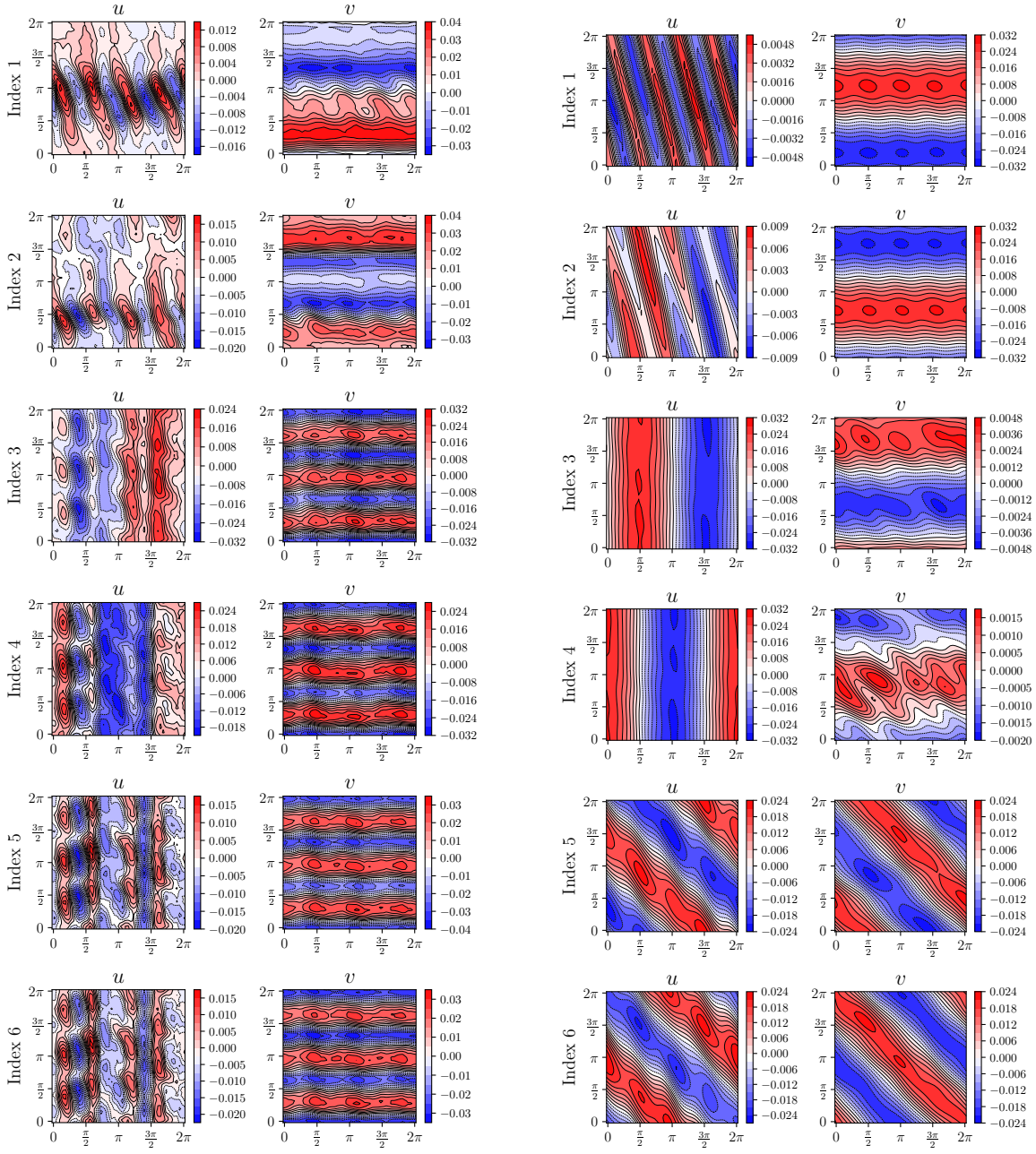


(b) cumulative variance of the latent space principal directions



(c) Eigenvalues of the covariance matrix of the reconstructed data.

Figure 9.25: Different metrics representing the comparisons between the auto-encoder and the POD method.



(a) Decoded latent space principal directions

(b) POD principal directions

Figure 9.26: Principal directions

9.5. Summary

This chapter compares dimensionality reduction using auto-encoders with the traditional POD method. AEs prove to be more than capable of dimensionality reduction and accurately representing chaotic system behaviour, with consistent trends across the tested systems. The POD method is regularly outperformed in terms of the reconstruction error, and the captured total variance (an analogue for energy). Further, multi-regime AEs having been tested, prove to be effective in modelling multiple regimes with well-separated latent spaces, albeit at a small performance penalty. This answers all the posited research questions for this section of the project (as outlined in the introduction):

- The *contractive loss* by itself proves to be insufficient in modelling 'smoother' latent spaces, and achieves its reduction by reducing the norm of the latent states instead (as shown in subsection 9.1.1,

subsection 9.2.1 and subsection 9.3.1). This shows that straightforward adaptation of ideas from the ML world to the realm of ROM does not always work, and tweaking the penalty term to instead account for the *normalized* norm of the Jacobian might be a valid direction for future research.

- Multi-regime auto-encoders work quite well and are able to represent multiple dynamical regimes in a single latent space, albeit at a small performance penalty (subsection 9.1.3, subsection 9.2.3, subsection 9.3.3, subsection 9.4.3). This is not trivial, and has implications beyond the scope of the present study. It remains to be explored how well this works as a generative space, meaning how well do the decoders 'reconstruct' regimes whose parameters they have not been trained on. Further meriting exploration is the question whether single RNNs can be trained to operate on these multi-regime latent spaces, with the separation acting as an implicit signal about the nature of the modelled regime, thus leading to truly multi-regime model-free methods for predicting chaotic systems.
- Lastly, attention-incorporation is observed to be marginally beneficially to CNN based AEs in the tested case (subsection 9.4.2), with kernel sizes playing more impactful roles. This once again shows how straightforward application of ML concepts is not always helpful, since a deeper analysis reveals the receptive fields of the encoder's outputs to be growing with increasing kernel filter sizes, eventually becoming large enough to cover the entire input space. Thus, the kind of global information the self-attention skip layer was designed to provide is already being accounted for in the AE.

10

Time Series Predictions using RNNs

Once the auto-encoders are trained, the latent states are computed and their time-series modelled. This chapter details the outcomes of said time-series modelling, and the performance of the combined AE-RNN systems. The research questions dealt with here are those pertaining to the 'model-free prediction' aspect of this thesis project, namely:

1. **Which series-to-series ML models are better suited to the combined AE-RNN ROM paradigm, according to a suitably defined prediction horizon (as the accuracy metric)?**
   (a) *How is the BPTT-trained RNN's performance accuracy affected when they are trained auto-regressively vs. when they are trained using teacher-forcing?*
   (b) *How does an additional BPTT-based auto-regressive training campaign affect an already (non-BPTT) trained ESN's accuracy?*
   (c) *How do these ML-based ROMs compare against a traditional POD-Galerkin approach?* Are there improvements in
      i. prediction horizons?
      ii. long-term statistical behaviour of predicted data?
2. **How does RK-inspired layering affect accuracy in BPTT-trained RNNs, specifically GRUs?**

Note that all the POD-Galerkin models have been solved using the time-step used in the original data generation (for the respective systems), and the RK4 time-integration method.

10.1. The Lorenz '63 System

The Lorenz '63 system for the single regime case ($\sigma = 10, \rho = 28, \beta = 2.67$) has a Lyapunov time of $t_L = 1.10$. The original data was generated using a time-step $\Delta t = 0.01$, which ends up assigning a 110 time-steps to each Lyapunov time interval. Considering the system's relative simplicity, and to have a more manageable time-step count per t_L, the RNNs were trained on snapshots separated by $\Delta t_{RNN} = 0.1$ instead.

The values yielded by the Bayesian optimization of the hyper-parameters for the ESN (with $n^r = 200$) are shown in Table 10.1. The iteration-wise evolution of the prediction horizon is shown in Figure C.1a. Similarly for the BPTT-trained RNNs, the optimization yields $f_{\text{noise}} = 5.18 \times 10^{-3}$ and $\lambda_{\text{reg}} = 3.73 \times 10^{-7}$, and its iteration-wise evolution is given in Figure C.1b.

Hyper-parameter	Optimized value
input scaling, ω_{in}	1.91×10^{0}
reservoir spectral radius, ρ_{res}	4.57×10^{-1}
leaking rate, α	7.54×10^{-1}
noise, f_{noise}	2.73×10^{-2}
regularization, λ_{reg}	1.00×10^{-9}

Table 10.1: Optimized ESN hyper-parameters.

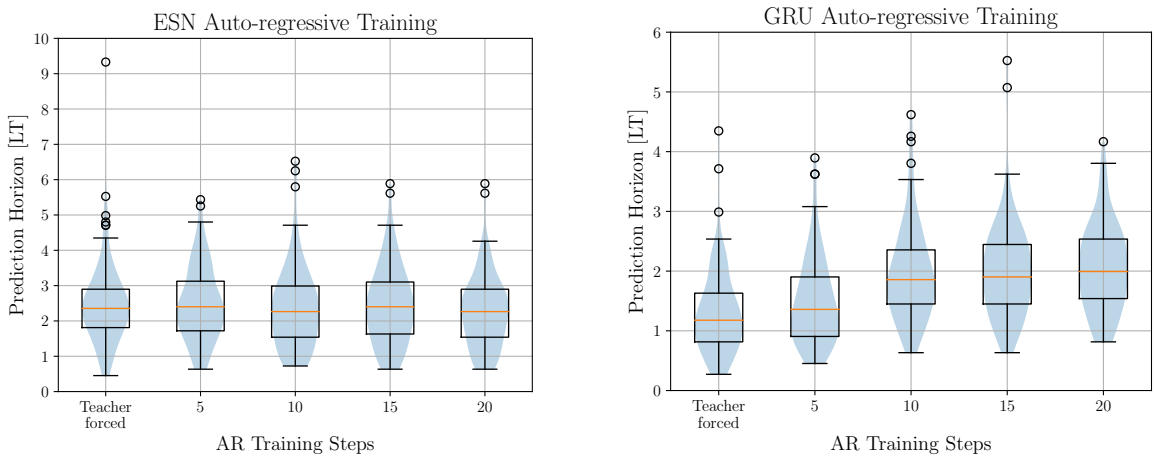
It can be observed that even though Bayesian optimization was unable to find better optimal points than those in its initial point-set when applied to the auto-encoders' hyper-parameter search, here it is quite beneficial and succeeds in finding better and better values (especially for the ESNs).

10.1.1. Comparison of teacher-forced and AR-trained networks

This section examines the impact of auto-regressive training on the combined AE-RNN model, as detailed in Section 8.4. The results of this investigation are presented here in terms of the distribution of the prediction horizons, computed over 100 different initializations from the testing data-set. The prediction horizons pertaining to the ESN and GRU based AE-RNNs are shown in Figure 10.1 and Table 10.2. Only the GRU network is presented here since the effect on other BPTT-trained networks is similar, and can be seen in subsection C.1.2. The *valid prediction time* (VPT) is defined as the 50^{th} percentile of this prediction horizon data-set.

Interestingly, the application of auto-regressive training yields disparate outcomes for the ESN and GRU networks. When considering the ESNs, the results demonstrate that auto-regressive training does not significantly enhance prediction performance. The prediction horizon distribution remains consistent across different training rounds, as showcased in Figure 10.1a. This suggests that the combined AE-ESN model is already functioning optimally, and introducing additional auto-regressive training does not lead to appreciable improvements. This phenomenon might arise from the fact that most of the parameters associated with the ESN, such as the reservoirs and input matrices, are pre-determined and fixed. Consequently, there is limited room for training to effectively enhance the system's performance in this context.

Conversely, the impact of auto-regressive training on the GRU network is strikingly evident (Figure 10.1b). The VPT experiences a substantial increase, rising from 1.18 LT to 2.00 LT — a boost of nearly 70%. Additionally, even the minimum and maximum values of the prediction horizons appear to improve. This enhancement can be attributed to the fundamental difference between the GRU and ESN architectures, where unlike the ESN, all of the GRU's parameters are trainable, allowing them to be fine-tuned during the auto-regressive training process. This malleability leads to a significant improvement in prediction performance, underlining the pivotal role that auto-regressive training plays in optimizing the predictive capabilities of the GRU-based AE-RNN model[1].



(a) Single layer ESN networks ($n^r = 200$).

(b) Single layer GRU networks ($n^r = 20$).

Figure 10.1: Prediction horizons for the teacher-forced and AR-trained networks (combined AE-RNN, AR Training Steps' refer to the number of output time-steps the network is auto-regressively trained on).

[1]The same arguments apply to the LSTM and Simple RNN case as well

AR Training Steps	ESN	GRU
teacher-forced	2.35	1.18
5	2.40	1.35
10	2.27	1.85
15	2.42	1.90
20	2.28	2.00

Table 10.2: VPT [LT] at different stages, for the ESN ($n^r = 200$) and GRU ($n^r = 20$) based AE-RNN.

n^r	ESN	GRU
200/20	2.42	2.00
500/50	2.48	2.01
800/80	2.32	2.72

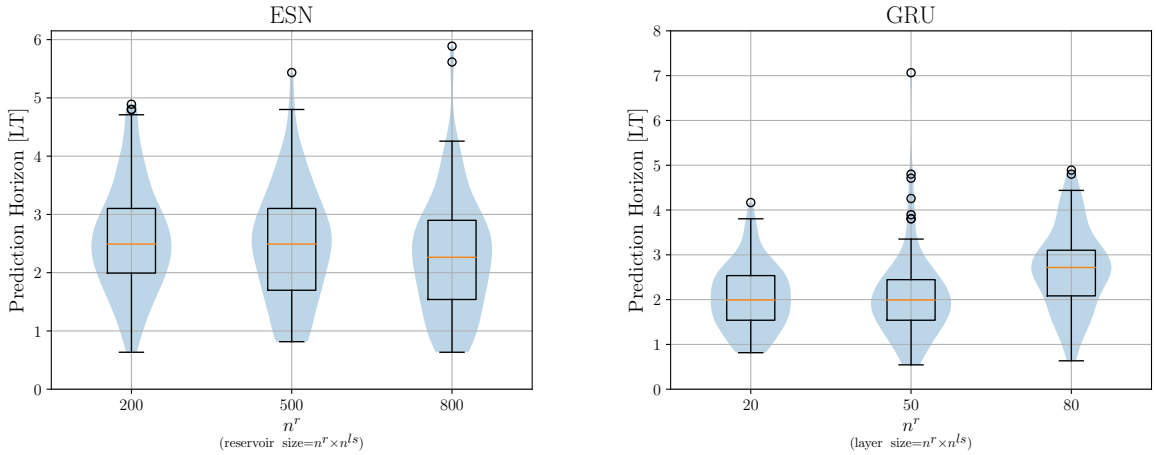
Table 10.3: VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN.

10.1.2. Increasing layer sizes

The impact of increasing layer sizes is studied w.r.t. both the ESN and GRU based AE-RNN models. The distributions of prediction horizons are shown in Figure 10.2 and Table 10.3.

In the case of the ESN model, augmenting the layer sizes results in marginal to harmful effects on its prediction performance. This is evident from the observed decrease in VPT as the layer sizes increase, contrary to initial expectations. The ESN's predictive capacity appears to plateau and then decline as the layer-size grows (Figure 10.2a). This phenomenon could be attributed to the inherent architecture of ESNs, where excessively large layers might introduce noise or irrelevant information into the learning process, in a system that is quite simple to begin with (remember, $n^{ls} = 2$), leading to suboptimal performance. Note that the larger layer sizes have even larger fractions of their total parameters that are decided apriori and held fixed (Section C.5), thus possibly necessitating more ensemble members to account for the random variances. Lastly, recall that the ESN hyper-parameters are optimized for $n^r = 200$ and then re-used for the larger reservoir sizes. The results here appear to point to the fact that this heuristic might not be entirely valid over a large range of reservoir sizes, and individually optimized hyper-parameters might possibly lead to better performances.

On the other hand, the GRU-based AE-RNN model showcases an opposite pattern in response to increasing layer sizes. Here, augmenting the layers exhibits a somewhat positive effect on prediction performance. The model's predictive capabilities plateau and then improves as the layers expand, as evident in Figure 10.2b.



(a) Single layer ESN networks.

(b) Single layer GRU networks.

Figure 10.2: Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN).

10.1.3. Comparison of single-layer networks

This section makes a comparative analysis between different AE-RNN models, each based on distinct recurrent architectures, as well as the traditional POD-Galerkin method (as described in Section 5.1). Considered RNNs are all single-layered, and include the GRU, LSTM, ESN and Simple RNN variants, all operating at their respective highest chosen layer sizes. The POD-Galerkin method employs two

POD components, to match $n^{ls} = 2$. The prediction horizon distributions are provided in Figure 10.3 and Table 10.4.

Upon inspection, the ESN, GRU, and LSTM based AE-RNN models exhibit relatively comparable performance levels, with the GRU taking the lead. In contrast, the Simple RNN based model lags behind in terms of prediction accuracy, implying its relatively limited capacity to grasp complex temporal dependencies. This is to be expected, since both the GRU and LSTM architectures account for additional 'information channels' between time-steps (via the forget/reset gates), and the ESN too possesses an analogue for this in the form of the leaking rate α.

Furthermore, the performance of the POD-Galerkin method paints a rather unfavourable picture. Despite employing two POD components, the method falls short, failing to surpass even a single Lyapunov time unit in terms of its VPT. This lacklustre performance underscores the challenges inherent in traditional techniques like POD-Galerkin when dealing with chaotic systems. The comparison brings to light the comparative advantages of the AE-RNN models over the traditional POD-Galerkin approach, particularly in their ability to capture complex behaviours and provide reasonable predictions even for chaotic dynamics.

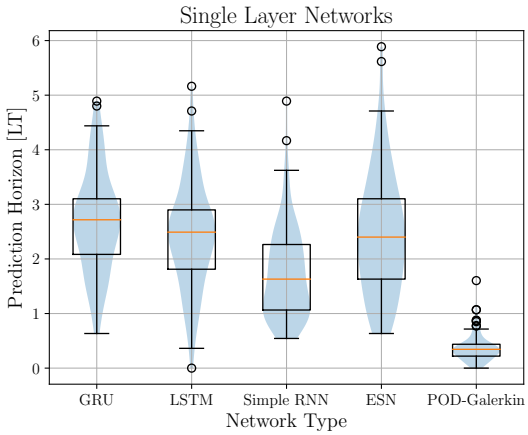


Figure 10.3: Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN).

RNN Type	VPT [LT]
GRU	2.72
LSTM	2.49
Simple RNN	1.63
ESN	2.32
POD-Galerkin	0.35

Table 10.4: VPT for the best performing RNN networks (highest layer size, combined AE-RNN).

Long-term Predictions and Statistical Behaviour

The long-term statistical behaviour of these methods is evaluated using the W_1 Wasserstein distance on their respective PDFs, which is a measure of 'closeness' of two distributions. These are evaluated on a time interval spanning 50 Lyapunov time units and across 50 different initializations from the testing data-set. Clearly, the GRU based AE-RNN model performs the best, both in terms of short-term prediction (symbolized by the VPT) and matching long-term statistical properties (symbolized by the W_1 distances). The LSTM based network, while having good short-term performance appears to have poor long-term statistical behaviour. Surprisingly, the simple RNN and the ESN, despite having differeing short-term performances appear to possess similar statistical behavior. The POD-Galerkin, unsurprisingly, is the worst performer in all scenarios.

The evolution of a sample trajectory is shown in Figure 10.5, for the AE-GRU model and the POD-Galerkin method. As observable, the AE-GRU model diverges from the true data after a point, owing to the chaotic nature of the system, yet maintains behaviour consistent with the evolving system. The POD-Galerkin method too diverges from the true data, but appears to peter out to a constant value. This explains the extremely high W_1 metrics that are observed for this method. Plots of sample trajectories for other AE-RNN models are given in Figure C.3. It is clear from the AE-LSTM evolution that despite good short-term predictions, the model eventually diverges from the behaviour of the true data (around the 25 LT mark), which explains its high W_1 metrics too.

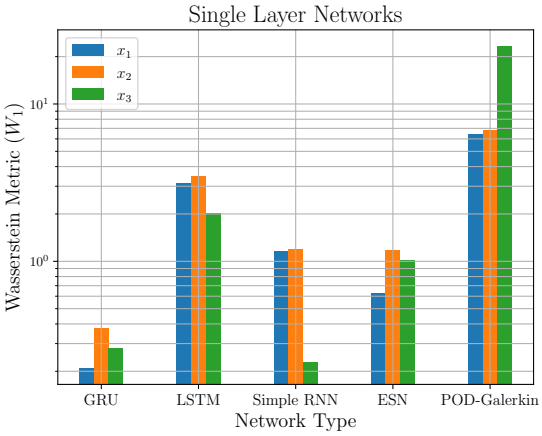
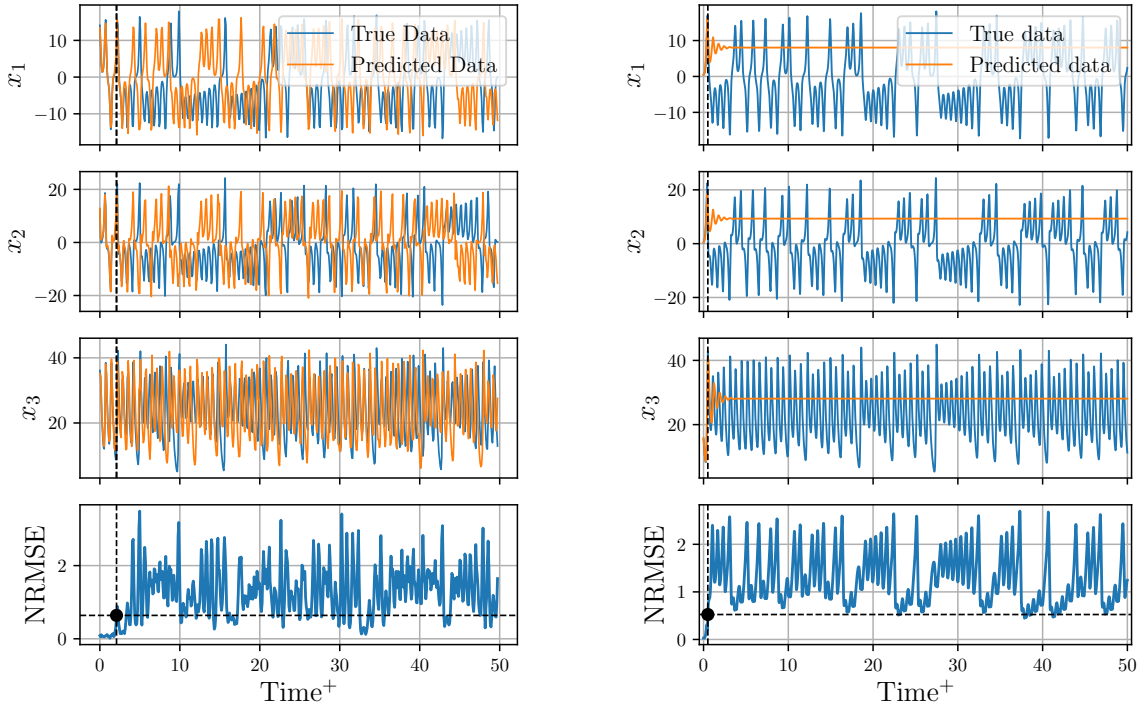


Figure 10.4: W_1 Wasserstein distances computed for the different x_j PDFs, with predictions taken over a time-interval of 50 t_L.



(a) Sample trajectory evolution for the AE-GRU model.

(b) Sample trajectory evolution for the POD-Galerkin method.

Figure 10.5: Long-term evolution of sample trajectories for the Lorenz '63 system.

10.1.4. Effect of RK-inspired layering in the GRUs

The distributions of prediction horizons for the various tested RK-inspired layering configurations (subsection 8.3.3) are shown in Figure 10.6. These configurations are tested on GRU based AE-RNN models. From Figure 10.6 it is apparent that the introduction of additional layers yields only marginal benefits to the prediction performance. The VPT actually decreases for the RK1 and RK2 layering methods, before eventually increasing. The variations in prediction horizons across different layering strategies are generally subtle (roughly ±7% compared to the single-layered case).

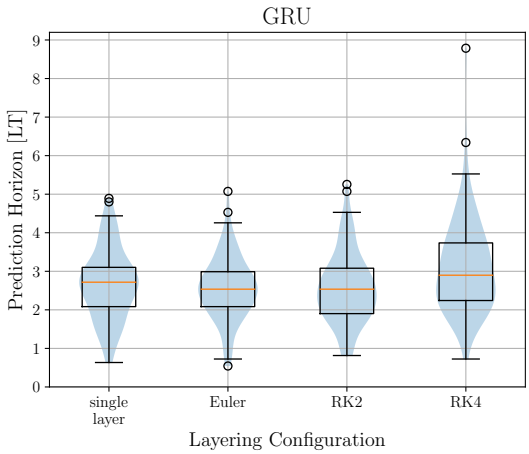


Figure 10.6: Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

Layering Type	VPT [LT]
Single layer	2.72
Euler layer (RK1 / skip layer)	2.54
RK2	2.54
RK4	2.91

Table 10.5: VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

10.2. The Charney-DeVore System

The Lyapunov time for the single regime case ($x_1^* = 0.95$, $x_4^* = 0.76095$) of the Charney-DeVore system is $t_L = 37.09$. The original data was generated using a time step of $\Delta t = 0.1$, leading to 371 time-steps within each Lyapunov time interval. As done previously, to ensure a more manageable number of time steps per t_L and noting the system's ODE nature, the RNNs were trained using snapshots separated by $\Delta t_{RNN} = 0.5$.

The outcomes of the Bayesian optimization process for hyper-parameters of the Echo State Network (ESN) with $n^r = 200$ are presented in Table 10.6. The progression of the prediction horizon throughout the iterations is illustrated in Figure C.4a. Likewise, for the BPTT-trained RNNs, the optimization results in $f_{\text{noise}} = 1.93 \times 10^{-3}$ and $\lambda_{\text{reg}} = 1.39 \times 10^{-6}$. The iterative evolution of the prediction horizon in this case is provided in Figure C.4b. Note that the optimized ρ_{res} value is greater than one, and yet the echo-state property is not violated.

Hyper-parameter	Optimized value
input scaling, ω_{in}	5.00×10^{-1}
reservoir spectral radius, ρ_{res}	1.03×10^{0}
leaking rate, α	5.00×10^{-1}
noise, f_{noise}	4.98×10^{-2}
regularization, λ_{reg}	1.87×10^{-5}

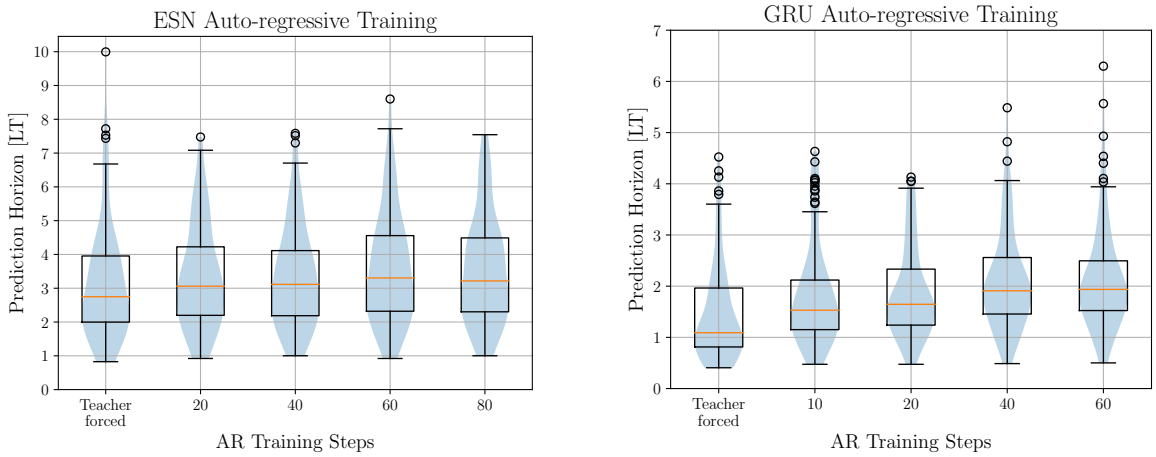
Table 10.6: Optimized ESN hyper-parameters.

10.2.1. Comparison of teacher-forced and AR-trained networks

The effect of auto-regressive training on GRU and ESN based AE-RNN models is examined, focusing on the distribution of prediction horizons as provided in Figure 10.7 and Table 10.7. Again, for brevity, only the GRU based network's outcomes are showcased in this section, as the impact on other BPTT-trained RNNs follows a similar trend and can be observed in subsection C.2.2.

Much like subsection 10.1.1, auto-regressive training yields distinct outcomes for the ESN and GRU networks, and the same arguments and observations apply here too. In the context of ESNs, auto-regressive training's contribution to prediction performance enhancement is found to be limited. The prediction horizons display a consistency across training rounds (Figure 10.7a), suggesting that the combined AE-ESN model is already performing optimally. Again, this can be attributed to the majority of the parameters linked to the ESN — its reservoirs and input matrices — being predetermined and fixed, consequently limiting the scope for training to significantly enhance system performance.

In contrast, the impact of auto-regressive training on the GRU network is more pronounced (Figure 10.7b). The VPT undergoes a substantial augmentation, escalating from 1.09 LT to 1.92 LT — an enhancement of roughly 75%. Further note, the maximum values of the VPT appear to improve while the minimum values show marginal improvement. As noted in subsection 10.1.1, this improvement can be attributed to the fundamental dissimilarity between the GRU and ESN architectures, where unlike the ESN, the GRU's parameters are freely trainable and get fine-tuned during the auto-regressive training process.



(a) Single layer ESN networks.

(b) Single layer GRU networks.

Figure 10.7: Prediction horizons for the teacher-forced and AR-trained networks (combined AE-RNN).

AR Training Steps	ESN	GRU
teacher-forced	2.74	1.09
10	–	1.53
20	3.07	1.65
40	3.12	1.92
60	3.31	1.95
80	3.22	–

Table 10.7: VPT [LT] at different stages, for the ESN ($n^r = 200$) and GRU ($n^r = 20$) based AE-RNN.

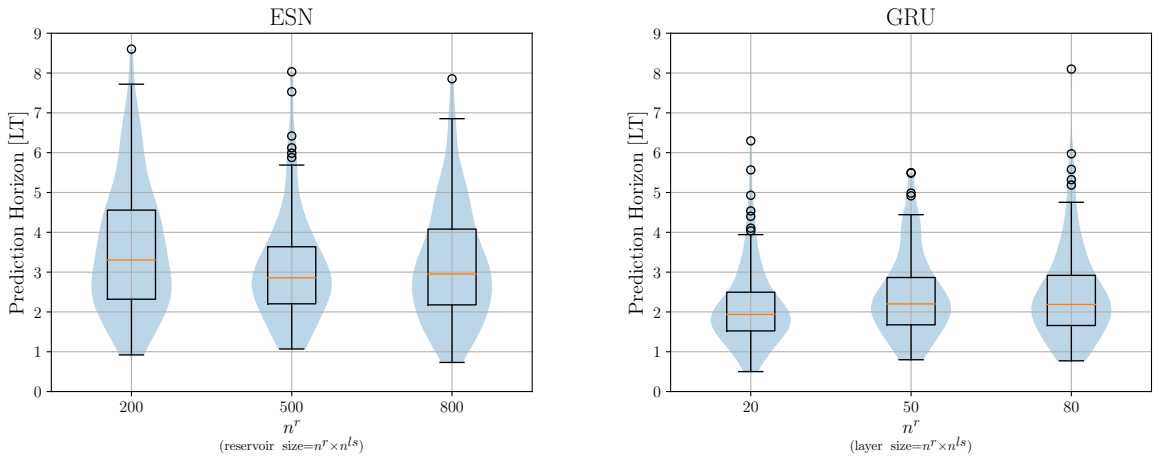
n^r	ESN	GRU
200/20	3.30	1.93
500/50	2.86	2.20
800/80	2.95	2.19

Table 10.8: VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN.

10.2.2. Increasing layer sizes

The impact of increasing layer sizes is investigated on both the ESN and GRU based AE-RNN models, and the distributions of prediction horizons are shown in Figure 10.8 and Table 10.8.

The trends followed here are the same as those in subsection 10.1.2, and the same observations and analysis apply. For the ESN based model, increasing layer sizes have marginal to detrimental effects on performance and the AE-ESN's VPT actually dips a little (Figure 10.8a). Once again, this phenomenon could be attributed to the inherent structure of AE-ESNs where excessively large layers introduce irrelevant information into the learning process of a system already characterized by simplicity ($n^{ls} = 5$), ultimately leading to sub-optimal performance. Also, larger layer sizes contain an even more substantial proportion of fixed parameters (Section C.5), potentially necessitating a greater number of ensemble members to account for inherent random variances. Also possibly at fault is the heuristic used to decide the hyper-parameters for the different layer sizes, where optimized hyper-parameters $n^r = 200$ are simply reused. Hyper-parameter optimization for individual layer sizes could potentially yield superior outcomes. Juxtaposed on this, the GRU-based AE-RNN model showcases a marginally positive impact (+14%) on prediction performance with increasing layer sizes (Figure 10.8b).



(a) Single layer ESN networks.

(b) Single layer GRU networks.

Figure 10.8: Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN).

10.2.3. Comparison of single-layer networks

This section presents a comparative analysis involving various AE-RNN models, alongside the conventional POD-Galerkin method. All the RNN models here are single-layered networks at their respective highest layer sizes. The POD-Galerkin approach employs five POD components, aligning with $n^{ls} = 5$. The distribution of prediction horizons is shown in Figure 10.9 and Table 10.9.

The AE-RNN models based on ESN, GRU, and LSTM architectures once again demonstrate relatively comparable levels of performance. However here the ESNs slightly outperforming the other two. In contrast, the Simple RNN model falls short in terms of prediction accuracy, showcasing its relatively constrained internal dynamics and limited ability to comprehend intricate temporal dependencies. As mentioned in subsection 10.1.3, this is not surprising considering the additional cross-time information channels in both the GRU and LSTM architectures (via the forget and reset gates). Similarly, the ESN integrates an anlogous mechanism in the form of the leaking rate α.

Furthermore, the performance of the POD-Galerkin method is once again lacking. Despite employing five POD components (just one short of the true dimensionality of the system), this method does not surpass one Lyapunov time unit in terms of its VPT. This underwhelming performance again highlights the challenges posed by conventional techniques like POD-Galerkin when tackling chaotic systems.

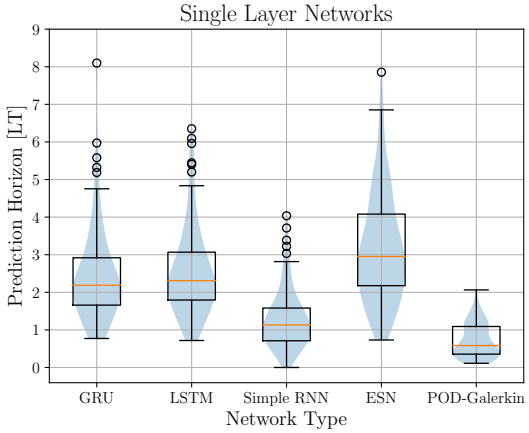


Figure 10.9: Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN).

RNN Type	VPT [LT]
GRU	2.18
LSTM	2.31
Simple RNN	1.14
ESN	2.96
POD-Galerkin	0.60

Table 10.9: VPT for the best performing RNN networks (highest layer size, combined AE-RNN).

Long-term Predictions and Statistical Behaviour

As done previously, the long-term statistical behaviour of these AE-RNN models is tested using the W_1 distance. Clearly, the GRUs perform superbly here too, with comparable values for the ESN and LSTM based networks. The simple RNN based network suffers from poorer matching, whilst the POD-Galerkin method is by far the worst performing. The evolution of a sample trajectory is shown in Figure 10.11, for the AE-GRU model and the POD-Galerkin method. As observable, both methods eventually diverge from the true trajectory, owing to the chaotic nature of the system. Yet only the AE-GRU model maintains behaviour consistent with the evolving system. The POD-Galerkin method, again, appears to peter out to constant values, thus explaining its high W_1 metrics. Plots of sample trajectories for other AE-RNN models are given in subsection C.2.3.

From these AE-RNN trajectories, it can be observed how all the models (excepting the Simple RNN based one) show both flow regimes - blocked flow and zonal flow - thus leading to similar W_1 metrics. The Simple RNN based model however only learns to predict the zonal flow regime, and not once does it switch to the blocked flow mode. This explains its relatively higher W_1 metrics.

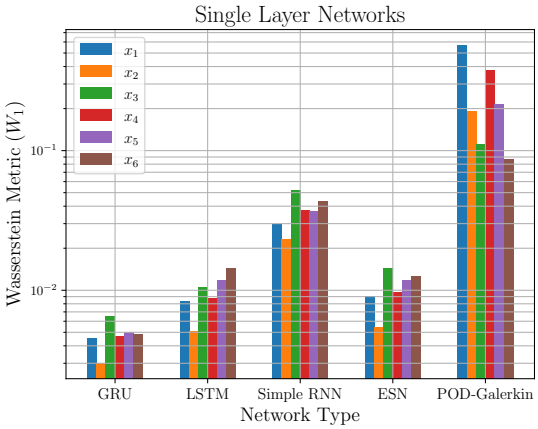
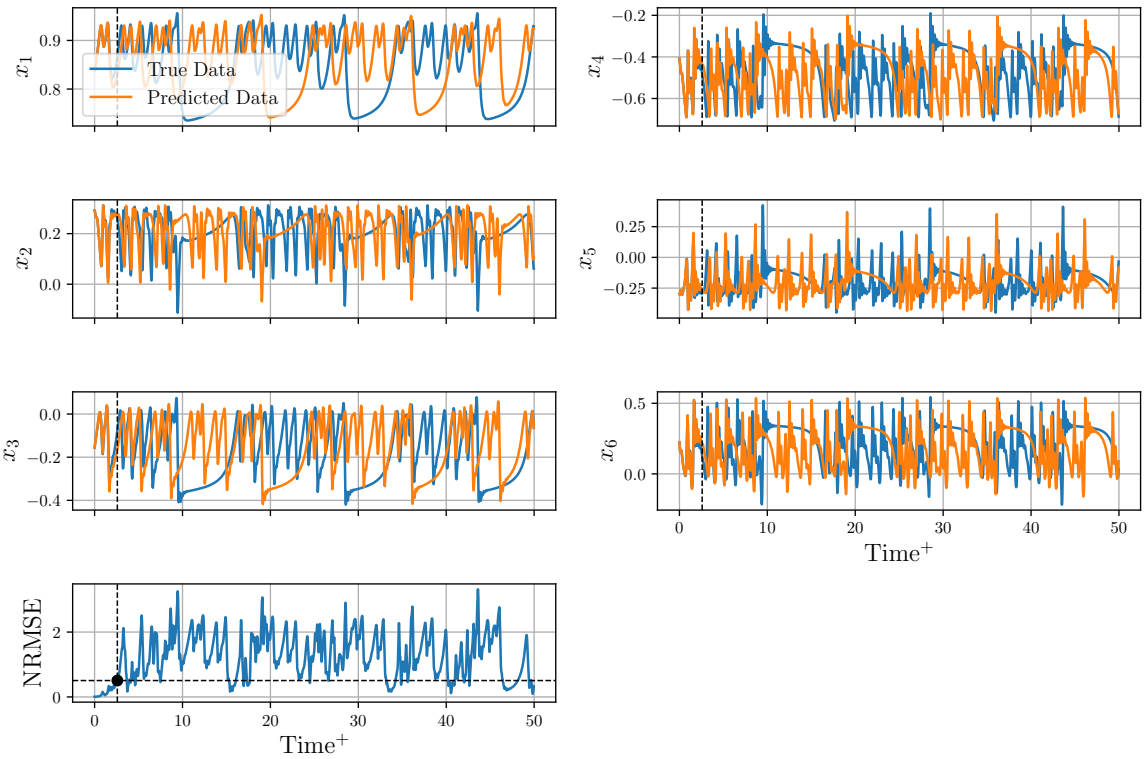


Figure 10.10: W_1 Wasserstein distances computed for the different x_j PDFs, with predictions taken over a time-interval of 50 t_L.

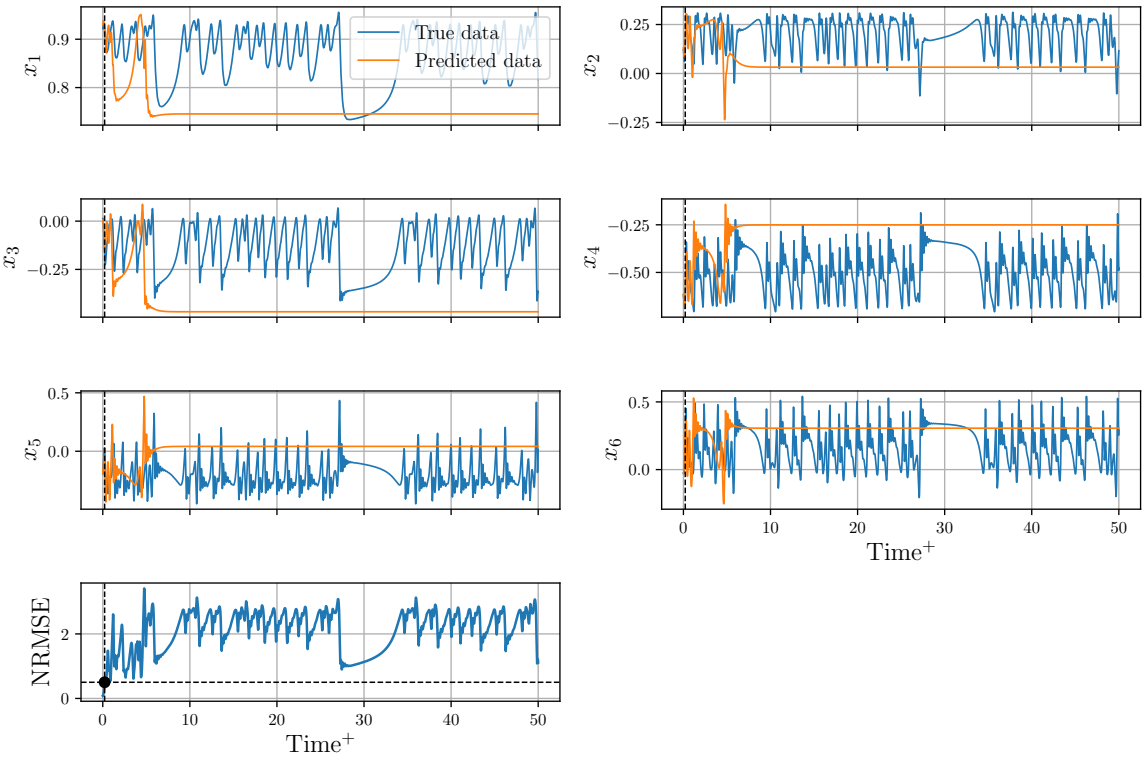
10.2.4. Effect of RK-inspired layering in the GRUs

The prediction horizon distributions corresponding to the various RK-inspired layering configurations tested on GRU-based AE-RNN models are shown in Figure 10.12 and Table 10.10. This exploration was detailed in subsection 8.3.3.

From Figure 10.12, it becomes evident that as observed previously, the incorporation of additional layers yields modest enhancements in prediction performance. A distinct enhancement is noticeable in the Euler layering configuration, resulting in an VPT increase from 2.19 LT to 2.78 LT (around 25%). Beyond this point, however, the addition of further layers fails to provide substantial performance improvements. This could be owing to more difficult training. With more loops in the RK connections, the gradients too would be getting multiplied multiple times leading to a scenario similar to the vanishing/exploding gradient problem [32]. Thus, RK-inspired layers augment performance only to a certain extent, and their impact eventually levels off.



(a) Sample trajectory evolution for the AE-GRU model.



(b) Sample trajectory evolution for the POD-Galerkin method.

Figure 10.11: Long-term evolution of sample trajectories for the Charney-DeVore system.

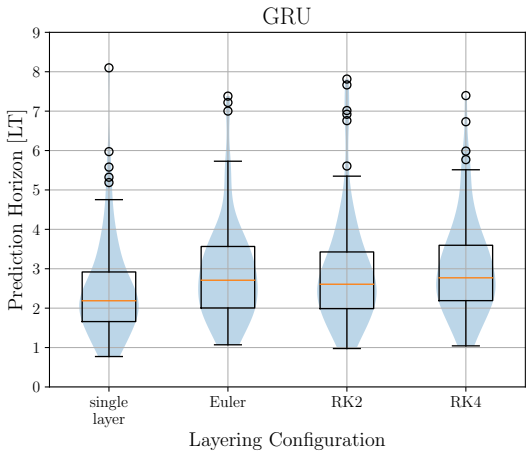


Figure 10.12: Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

Layering Type	VPT [LT]
Single layer	2.19
Euler layer (RK1 / skip layer)	2.71
RK2	2.61
RK4	2.78

Table 10.10: VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

10.3. The Kuramoto-Sivashinsky System

The Lyapunov time linked to the single regime case ($\nu_1 = \nu_2 = \nu_3 = 1$) of the Kuramoto-Sivashinsky system is $t_L = 13.95$. The generation of the original data employed a time step of $\Delta t = 0.1$, corresponding to 140 discrete time steps within each Lyapunov time interval. As done previously, to achieve a more suitable count of time steps per t_L, the RNNs are trained on snapshots spaced apart by $\Delta t_{RNN} = 0.2$. Note that this system represents a PDE, as opposed to the low-dimensional ODEs of the previous systems and hence its dynamics are inherently more complex and tougher to model.

The outcomes resulting from the Bayesian optimization procedure applied to the hyper-parameters of the ESN with a reservoir size of $n^r = 200$ are shown in Table 10.11. The evolution of the prediction horizon across various iterations is depicted in Figure C.9a. Similarly, in the context of BPTT-trained RNNs, the optimization yields values of $f_{\text{noise}} = 9.84 \times 10^{-2}$ and $\lambda_{\text{reg}} = 5.75 \times 10^{-7}$. The stepwise advancement of the prediction horizon for this case is plotted in Figure C.9b.

Hyper-parameter	Optimized value
input scaling, ω_{in}	1.43×10^{-1}
reservoir spectral radius, ρ_{res}	8.53×10^{-1}
leaking rate, α	5.00×10^{-1}
noise, f_{noise}	6.05×10^{-2}
regularization, λ_{reg}	1.00×10^{-7}

Table 10.11: Optimized ESN hyper-parameters.

10.3.1. Comparison of teacher-forced and AR-trained networks

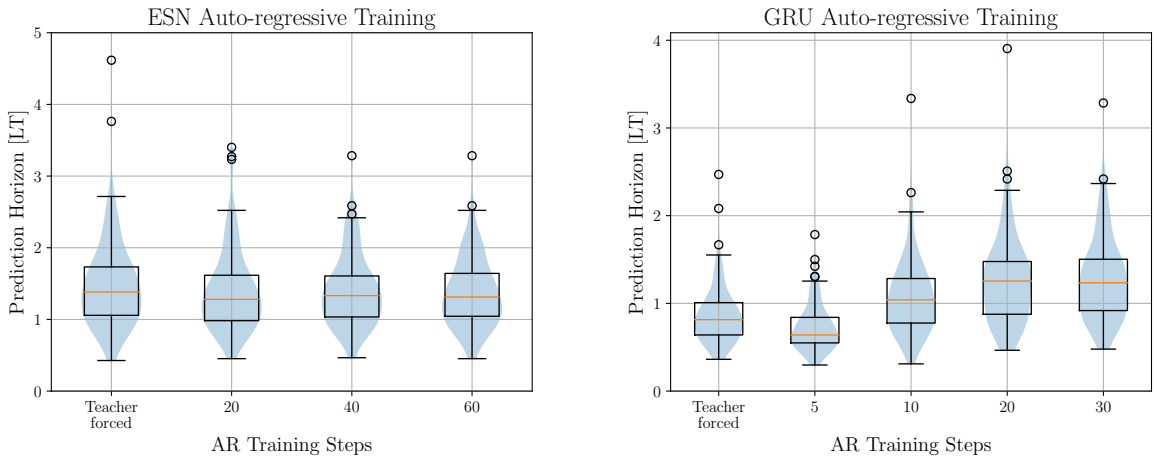
The effect of auto-regressive training on GRU and ESN based AE-RNN models are examined in this section, focusing on the distribution of prediction horizons. Figure 10.13 and Table 10.12 display the results. For brevity, only the GRU based network's outcomes are showcased in this section, the impact on other BPTT-trained RNNs can be observed in subsection C.3.2.

Similar to the chaotic systems discussed previously, the application of auto-regressive training leads to distinct outcomes for the ESN and GRU based models. The same arguments and observations that were presented previously hold true in this context as well. When considering ESNs, the impact of auto-regressive training on enhancing prediction performance is found to be restrained. The prediction horizons across various training rounds look almost exactly alike (Figure 10.13a), indicating the already optimal functioning of the combined AE-ESN model. This outcome can again be attributed to the substantial proportion of the ESN's parameters, encompassing its reservoirs and input matrices, that are

predetermined and fixed, effectively curtailing the extent to which training can substantially enhance the system's overall performance.

In contrast, the impact of auto-regressive training on the GRU based model is more pronounced (Figure 10.13b). The VPT undergoes a substantial augmentation, escalating from 0.81 LT to 1.23 LT — an enhancement of roughly 50%. Here too the minimum and maximum values of the prediction horizons improve. As noted in previous sections, this improvement can be attributed to the fundamental dissimilarity between the GRU and ESN architectures, where unlike the ESN, the GRU's parameters are freely trainable and get fine-tuned during the auto-regressive training process. Also note the sudden dip and subsequent rise of the $VPTs$. This occurs because the 5-output-step training round essentially causes the model to *overfit* to the extreme short-term, leading to degradation of predictive performance.

Note that while the LSTM-based model trains equivalently, the SimpleRNN-based model sees its performance degrade massively, despite all the measures taken to ensure adequate training. The additional information channels in GRUs and LSTMs are the only source of difference between these networks. Their importance in both improving prediction accuracy and stabilizing training is on clear display here, in a system with more complex dynamics.



(a) Single layer ESN networks.

(b) Single layer GRU networks.

Figure 10.13: Prediction horizons for the teacher-forced and AR-trained networks (combined AE-RNN).

AR Training Steps	ESN	GRU
teacher-forced	1.38	0.81
5	–	0.64
10	–	1.04
20	1.28	1.25
30	–	1.23
40	1.33	–
60	1.31	–

Table 10.12: VPT [LT] at different stages, for the ESN ($n^r = 200$) and GRU ($n^r = 20$) based AE-RNN.

n^r	ESN	GRU
200/20	1.38	1.25
500/50	1.49	1.29
800/80	1.53	1.33

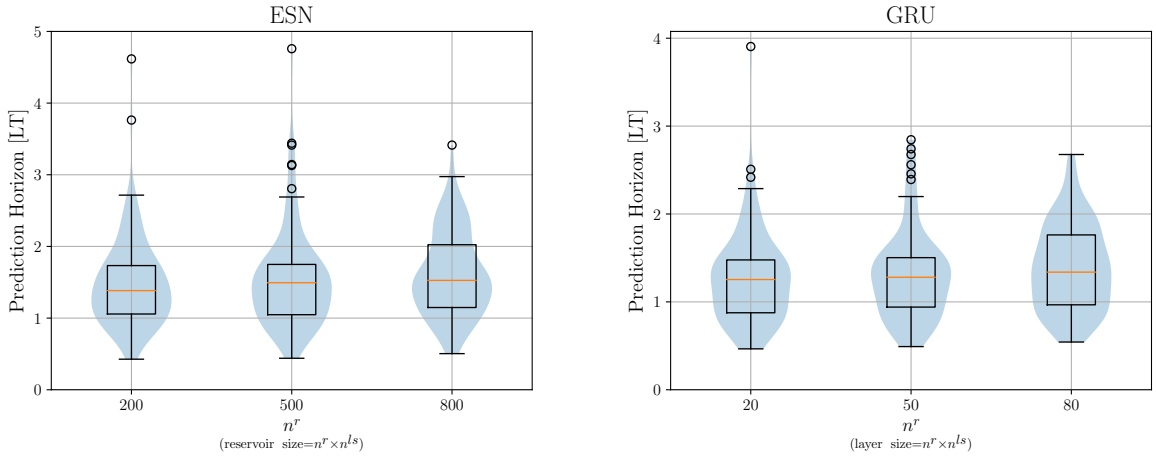
Table 10.13: VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN.

10.3.2. Increasing layer sizes

The influence of varying layer sizes on both the ESN and GRU based AE-RNN models on their corresponding prediction horizon distributions is presented in Figure 10.14 and Table 10.13.

The trends observed here closely mirror those observed in the preceding cases, and the same set of insights and analyses are applicable. For the ESN-based model, the increase in layer sizes yields marginal effects on performance. The predictive capabilities of the AE-ESN plateau as layer sizes expand (Figure 10.14a). This recurrence of the phenomenon can be ascribed to the inherent architecture of AE-ESNs, where larger layers consist of an even higher proportion of fixed parameters (Section C.5), which

might require a larger ensemble to address the resulting random variances. Additionally, the method of employing the same optimized hyper-parameters (those of the $n^r = 200$ network) across various layer sizes could potentially lead to sub-optimal performance. Tailored hyper-parameter optimization for each layer size might lead to better outcomes. In contrast, the GRU-based AE-RNN model presents a slightly positive impact on prediction performance with increasing layer sizes (Figure 10.14b).



(a) Single layer ESN networks.

(b) Single layer GRU networks.

Figure 10.14: Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN).

10.3.3. Comparison of single-layer networks

A comparative analysis as done previously is presented, involving the same range of AE-RNN models and the conventional POD-Galerkin method outlined in Section 5.1. The POD-Galerkin technique employs sixteen POD components, corresponding with $n^{ls} = 16$. The distribution of prediction horizons is provided in Figure 10.15.

The AE-RNN models based on ESN, GRU, and LSTM RNNs once again exhibit relatively similar performance levels, with ESNs slightly outperforming the other two. In contrast, the Simple RNN model shows much poorer prediction performance, pointing to its inability to accurately model time-dependencies as the complexity of the system increases. The discussion on additional information channels is applicable here too.

Furthermore, the performance of the POD-Galerkin method once again reveals its limitations. This method again fails to surpass even one Lyapunov time unit in terms of its valid prediction time (VPT). Not just this, with a VPT of just 0.13, this is the worst outcome of the POD-Galerkin application to all systems up until now. This highlights the challenges that conventional techniques like POD-Galerkin encounter when dealing with chaotic systems, especially when complexity increases, like in the present case (a PDE). The POD model reduction already has a reconstruction error that is an order of magnitude higher than the auto-encoder's (subsection 9.3.2). Further added are the errors introduced in time-propagation by the flat Galerkin approximation (subsection 5.1.3), thereby resulting in the poor performance observed here. The comparison further emphasizes AE-RNN models' relative advantages over traditional approaches, shedding light on their capability to accommodate the complexities of chaotic systems more effectively.

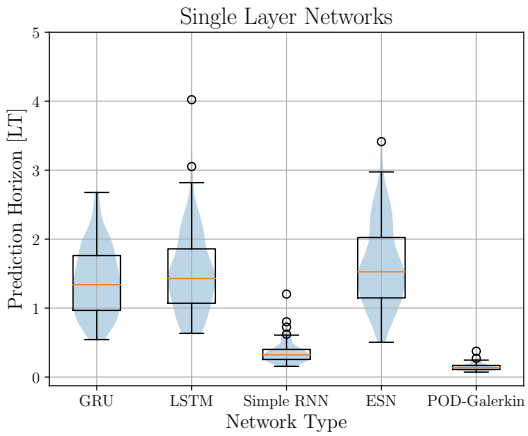


Figure 10.15: Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN).

RNN Type	*VPT* [LT]
GRU	1.33
LSTM	1.43
Simple RNN	0.32
ESN	1.52
POD-Galerkin	0.13

Table 10.14: *VPT* for the best performing RNN networks (highest layer size, combined AE-RNN).

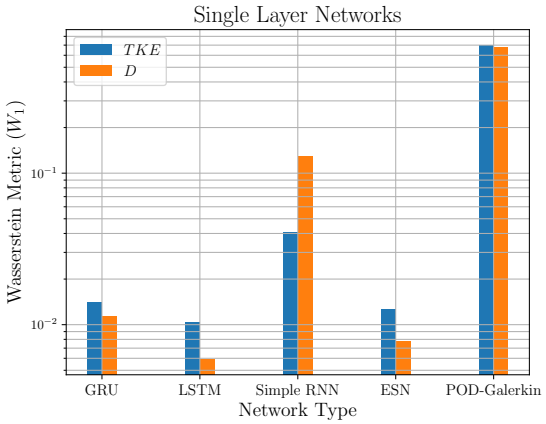
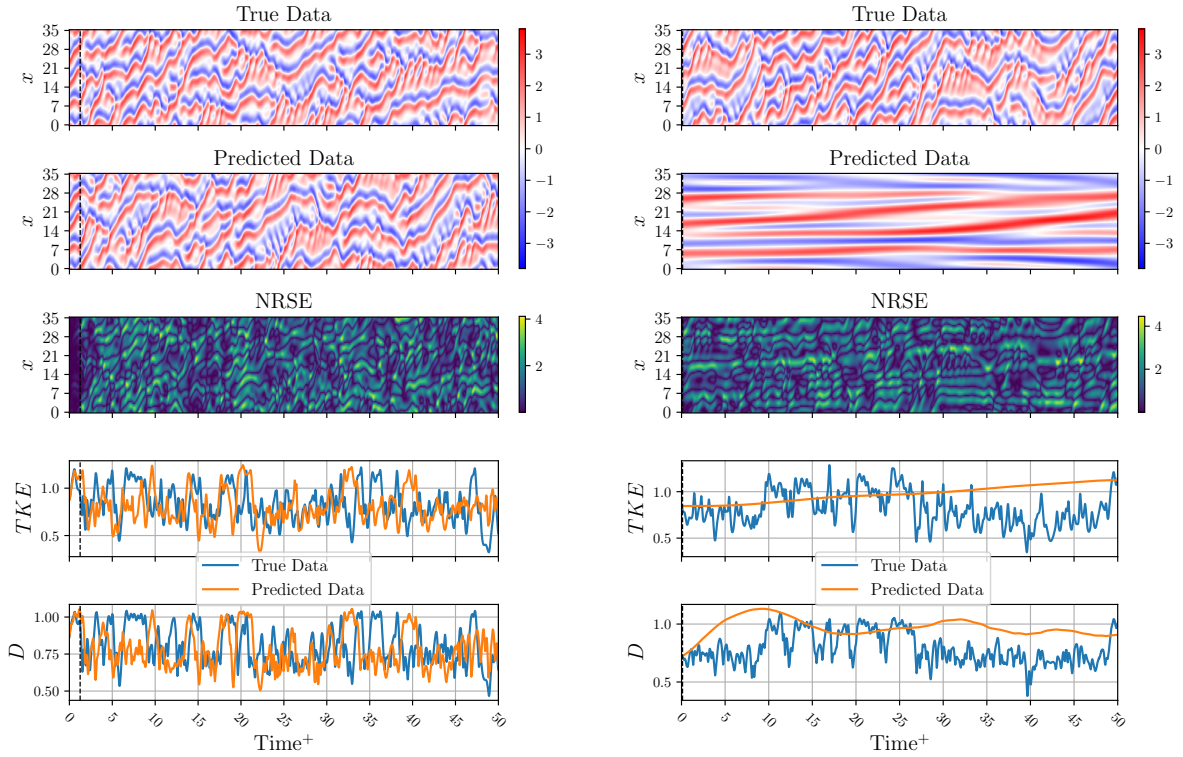


Figure 10.16: W_1 Wasserstein distances computed for the TKE and D PDFs, with predictions taken over a time-interval of 50 t_L.

Long-term Predictions and Statistical Behaviour

Once again, the long-term statistical performance is investigated, by means of the W_1 Wasserstein distance. Here however, the attractors are represented by the TKE (the turbulent kinetic energy) and D (the mean dissipation rate). Clearly, the ESN, GRU and LSTM based AE-RNNs perform similarly, whereas the simple RNN based model and the POD-Galerkin method both possess terrible long-term behaviour in addition to poor short-term performance.

The progression of a sample trajectory is depicted in Figure 10.17, illustrating the trajectories of both the AE-GRU model and the POD-Galerkin method. Both approaches eventually deviate from the true trajectory due to the inherently chaotic nature of the system. However, only the AE-GRU model maintains behaviour consistent with the evolving dynamics of the system. Conversely, the POD-Galerkin method tends to converge to near-constant values, accounting for its elevated W_1 metrics. Additional trajectory plots for different AE-RNN models are provided in subsection C.3.3. Across these AE-RNN trajectories, a typical pattern emerges among the AE-RNN models (excluding the Simple RNN-based model), leading to similar W_1 metrics. Notably, the Simple RNN-based model experiences intervals of nearly constant behavior before transitioning back to expected patterns. This unexpected switching is also reflected in its comparatively higher W_1 metrics, highlighting its distinct long-term behaviour.



(a) Sample trajectory evolution for the AE-GRU model.

(b) Sample trajectory evolution for the POD-Galerkin method.

Figure 10.17: Long-term evolution of sample trajectories for the Kuramoto-Sivashinsky system.

10.3.4. Effect of RK-inspired layering in the GRUs

The prediction horizon distributions, stemming from the experimentation with various RK-inspired layering configurations applied to GRU-based AE-RNN models, is given in Figure 10.18 and Table 10.15.

Analyzing Figure 10.12, it becomes evident that here too the results follow a similar trend as seen in previous systems. A noticeable advancement is particularly visible in the Euler layering configuration, leading to an increase in valid prediction time (VPT) from 1.28 LT to 1.59 LT, marking an approximate enhancement of 24%. However, subsequent layers cease to make substantial enhancements in performance. As observed previously, additional RK-layering configurations provide detrimental to diminishing returns. This could possibly be owing to difficult training on account of the many looped connections introduced by the higher-order RK methods.

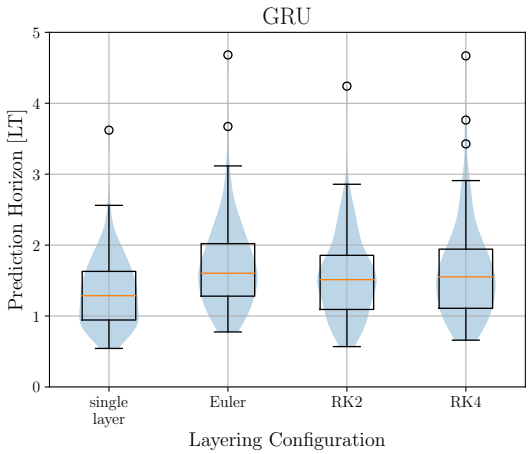


Figure 10.18: Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

Layering Type	VPT [LT]
Single layer	1.33
Euler layer (RK1 / skip layer)	1.60
RK2	1.50
RK4	1.55

Table 10.15: VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

10.4. Kolmogorov Flow

The Lyapunov time associated with the single regime scenario ($Re = 40$) of the Kolmogorov flow system is $t_L = 13.06$. The generation of the original data-set involved using a time step of $\Delta t = 0.01$, with snapshots stored every $\Delta t_{\text{store}} = 0.25$. This translates to 53 discrete time steps within each Lyapunov time interval. This time-step value is also adopted for training the RNNs.

The outcomes of the Bayesian optimization of the hyper-parameters of the ESN, with a reservoir size corresponding to $n^r = 200$, are presented in Table 10.16. The evolution of the prediction horizon across different iterations is illustrated in Figure C.12a. Similarly, in the context of BPTT-trained RNNs, the optimization results in values of $f_{\text{noise}} = 9.84 \times 10^{-2}$ and $\lambda_{\text{reg}} = 5.75 \times 10^{-7}$. The incremental advancement of the prediction horizon for this scenario is plotted in Figure C.12b.

Hyper-parameter	Optimized value
input scaling, ω_{in}	1.37×10^{0}
reservoir spectral radius, ρ_{res}	7.92×10^{-1}
leaking rate, α	8.26×10^{-1}
noise, f_{noise}	1.00×10^{-1}
regularization, λ_{reg}	1.00×10^{-3}

Table 10.16: Optimized ESN hyper-parameters.

10.4.1. Comparison of teacher-forced and AR-trained networks

The effect of auto-regressive training on GRU based AE-RNN models is examined in this section, focusing on the distribution of prediction horizons. Since auto-regressive training has been found to be in-effectual for the ESNs in all the previously tested systems (ODE and PDE), it is skipped for this system. Figure 10.19 and Table 10.17 display the prediction horizon distributions. For brevity, only the GRU based network's outcomes are showcased in this section, as the impact on other BPTT-trained RNNs follows a similar trend and can be observed in subsection C.4.2.

Similarly to the chaotic systems discussed earlier, auto-regressive training leads the VPT to undergo a general improvement, escalating from 0.72 LT to 1.07 LT — an enhancement of roughly 49%. The maximum values of the prediction horizons are also seen to be improving with increasing output steps. Lastly, note that the minimum values of the prediction horizons are quite low, almost zero, indicating the presence of such initializations where the errors diverge at the first few predicted time-steps themselves.

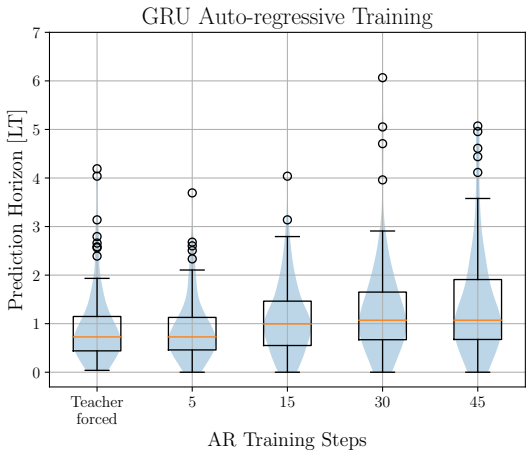


Figure 10.19: Prediction horizons for the teacher-forced and AR-trained single layer GRU based models (combined AE-RNN, $n^r = 20$).

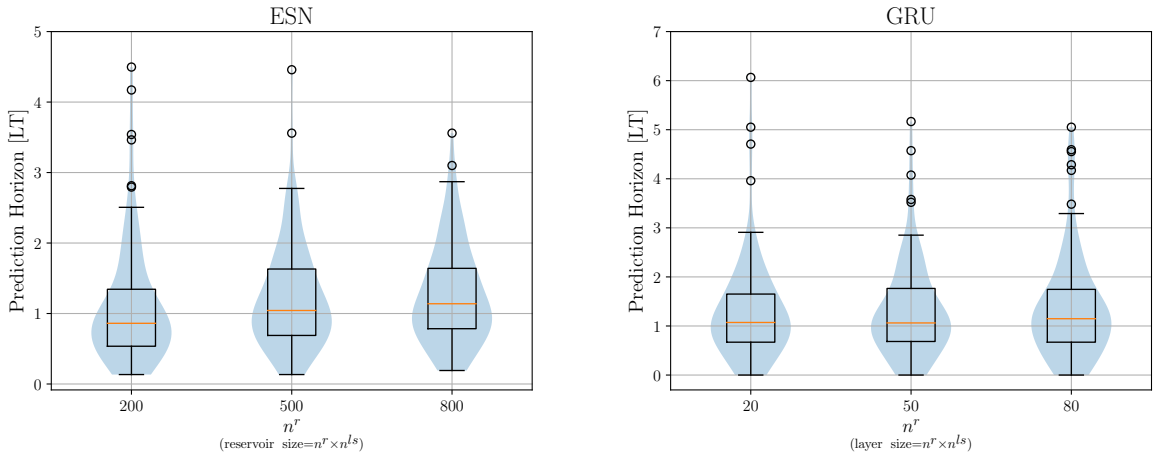
AR Training Steps	GRU
teacher-forced	0.72
5	0.73
15	0.99
30	1.07
45	1.07

Table 10.17: VPT [LT] at different stages, for the GRU ($n^r = 20$) based AE-RNN.

10.4.2. Increasing layer sizes

The investigation into the influence of varying layer sizes on both the ESN and GRU based AE-RNN models, with the corresponding distribution of prediction horizons is presented in Figure 10.20 and Table 10.18.

The trends observed here closely mirror those observed in the preceding cases, and the same set of insights and analyses are applicable. For the ESN-based model, the layer size increase yields marginal performance gains (Figure 10.20a). This recurrence of the phenomenon can be ascribed to the inherent architecture of AE-ESNs, where larger layers consist of an even higher proportion of fixed parameters (Section C.5), which might require a larger ensemble to address the resulting random variances. Additionally, the method of employing the same optimized hyper-parameters (those of the $n^r = 200$ network) across various layer sizes could potentially lead to sub-optimal performance. Tailored hyper-parameter optimization for each layer size might lead to better outcomes. The GRU-based AE-RNN model presents a similar marginal effect on prediction performance with increasing layer sizes (Figure 10.20b).



(a) Single layer ESN networks.

(b) Single layer GRU networks.

Figure 10.20: Prediction horizons for the best performing RNN networks with changing layer sizes (combined AE-RNN).

n^r	ESN	GRU
200/20	0.86	1.07
500/50	1.04	1.06
800/80	1.13	1.14

Table 10.18: VPT [LT] at different layer sizes, for the ESN and GRU based AE-RNN.

10.4.3. Comparison of single-layer networks

This section presents a comparative analysis, covering the same array of AE-RNN models and the traditional POD-Galerkin method. The POD-Galerkin technique employs twenty-seven POD components, to match $n^{ls} = 27$. The distribution of prediction horizons is presented in Figure 10.21 and Table 10.19.

The AE-RNN models based on ESN, GRU, and LSTM RNNs showcase similar prediction performances. As observed in the previous PDE case (the KS equation), the Simple RNN model fails miserably in prediction accuracy, highlighting its inadequate internal dynamics and limited ability to capture complex temporal dependencies. Further, the performance of the POD-Galerkin method once again exposes its limitations. Despite employing twenty-seven POD components, it showcases a VPT of merely 0.07. This represents the poorest outcome of the POD-Galerkin approach across all the investigated systems. The abysmal performance of this method in the PDE cases spotlights this method's pitfalls when grappling with chaotic systems, particularly as complexity escalates. This comparison too, points out the relative advantages that AE-RNN models offer over traditional approaches, elucidating their capacity to effectively navigate the intricacies of chaotic systems.

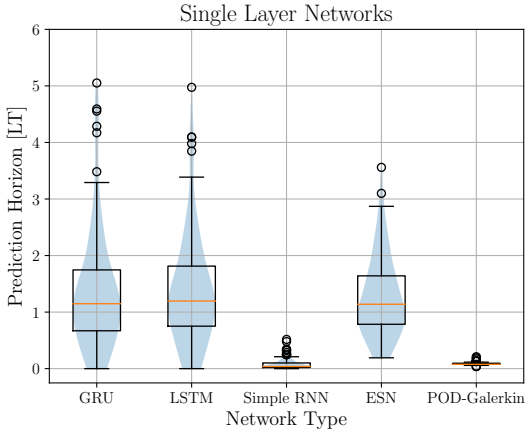


Figure 10.21: Prediction horizons for the best performing RNN networks (highest layer size, combined AE-RNN).

RNN Type	VPT [LT]
GRU	1.14
LSTM	1.19
Simple RNN	0.04
ESN	1.14
POD-Galerkin	0.07

Table 10.19: VPT for the best performing RNN networks (highest layer size, combined AE-RNN).

Long-term Predictions and Statistical Behaviour

Also compared are the long-term statistical performances of these models, in terms of the W_1 Wasserstein distances between the predicted and actual PDFs of the TKE and the mean dissipation D (Figure 10.22). Clearly, the AE-ESN possesses the lowest W_1 metrics, indicating the closest matching in long-term statistics, with the AE-LSTM and AE-GRU close behind. The simple RNN based model and the POD-Galerkin method are much worse off, as is to be expected.

The trajectory evaluations are displayed in Figure 10.23, showcasing those generated by both the AE-GRU model and the POD-Galerkin method. Over time, both methodologies exhibit deviations from the actual trajectory due to the inherently chaotic nature of the system. However, as observed previously, only the AE-GRU model maintains behaviour consistent with the evolving dynamics of the system. In contrast, the POD-Galerkin method not only tends to over-predict values by nearly a factor of two, but its variations are also considerably stronger than those observed in the true trajectories. These factors contribute to the noticeably elevated W_1 metrics associated with the method.

Additional trajectory plots for distinct AE-RNN models can be found in subsection C.4.3. Analyzing these AE-RNN trajectories reveals similarly evolving patterns among the AE-RNN models (excluding the Simple RNN-based model), leading to similar W_1 metrics. Interestingly, the Simple RNN-based

model exhibits values orbiting around the mean of the depicted quantities, displaying minor variations. This tendency of a regression-twowards-the-mean is also evident in its relatively higher W_1 metrics, highlighting its poor long-term behaviour. Further plots of snapshots of the x-velocity u, the y-velocity v and the vorticity ω for the predictions from different models are provided in subsection C.4.4, subsection C.4.5, subsection C.4.6, subsection C.4.7 and subsection C.4.8. Note the deeply coloured (i.e. strongly high valued) nature of the predictions of the POD-Galerkin method in subsection C.4.8, alluding to its high-valued TKE and D. Also of note is the small errors in ω for the AE-ESN's predictions. Since the mean dissipation rate D is related to the magnitude of ω (Equation 6.15), this agrees with the low W_1 metric of D observed for the AE-ESN model.

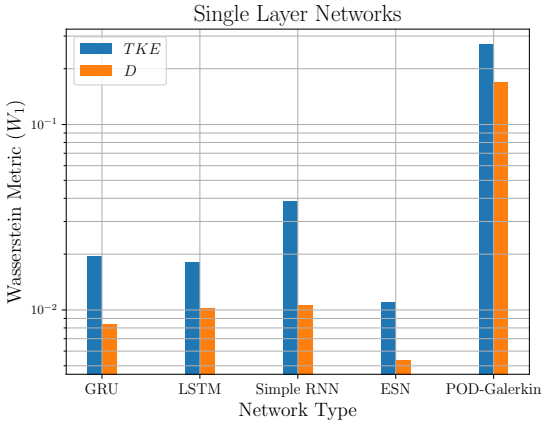
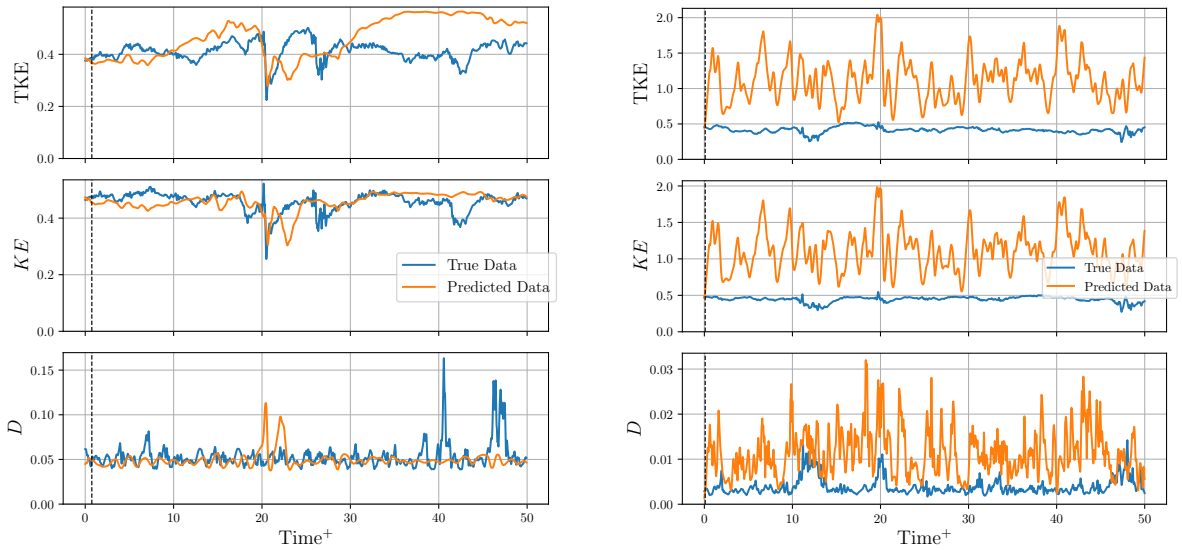


Figure 10.22: W_1 Wasserstein distances computed for the different x_j PDFs, with predictions taken over a time-interval of 50 t_L.



(a) Sample trajectory evolution for the AE-GRU model.

(b) Sample trajectory evolution for the POD-Galerkin method.

Figure 10.23: Long-term evolution of sample trajectories for the Kolmogorov flow system.

10.4.4. Effect of RK-inspired layering in the GRUs

The distributions of prediction horizons for the various tested RK-inspired layering configurations are shown in Figure 10.24. From Figure 10.24 it is apparent that the introduction of additional layers yields only marginal benefits to the prediction performance. The VPT stays relatively constant, and actually decreases mildly for the RK4 layering methods. The variations in prediction horizons across different layering strategies are generally subtle (roughly ±8% compared to the single-layered case). This could be owing to the added complexity of training a complete multi-scale convolutional auto-encoder in tandem, which overshadows any effect the layering could have had.

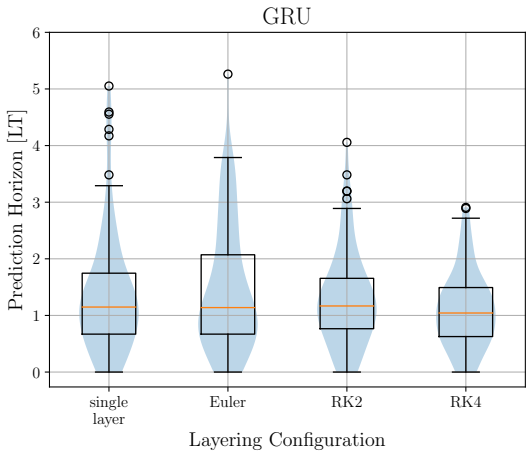


Figure 10.24: Prediction horizons for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

Layering Type	VPT [LT]
Single layer	1.14
Euler layer (RK1 / skip layer)	1.14
RK2	1.16
RK4	1.04

Table 10.20: VPT for the best performing GRU networks with different layering architectures (highest layer size, combined AE-RNN).

10.5. Summary

This chapter compares the combined AE-RNN models with each other and with the traditional POD-Galerkin approach. The AE-RNNs prove to be superior in all aspects, be it short-term prediction accuracy or long-term statistical behaviour. These gaps become even more pronounced as the system's complexity increases. In fact, with the switch from ODEs to PDEs, not only does the POD-Galerkin method's performance deteriorate, so does that of the simple RNN based AE-RNN model's. This is attributed to the fact that it carries no additional cross-time-step information channels, as done by the GRUs and LSTMs (and even the ESNs by means of α), which limits its time-comprehension abilities. Further tested is the effect of RK-inspired layering, and increasing layer sizes. These answer all the highlighted research questions from the introduction:

- The ESNs, GRUs and LSTMs based models all have quite similar performances across the systems, especially as complexity increases, and:
  - an auto-regressive training campaign is crucial for improving the prediction performances of a BPTT trained model such as the LSTM or GRU, with $VPTs$ increasing by factors of 25% to 75% to the base teacher-forced model.
  - auto-regressive training has little to no effect on the performance of the AE-ESNs, possibly because they are already performing optimally. Additionally, important to note is that accurate hyper-parameter selection would appear to be a greater benefactor since the performance across increasing layer sizes does not improve (as would be expected).
- The effect of RK-inspired layering displays inconsistent behaviour across the systems. For the Lorenz '63 and the Kolmogorov flow systems, additional layering has little to no impact on the predictive performances. While for the Charney-DeVore and Kuramoto-Sivashinsky systems it offers limited benefits, with prediction performances generally increasing ($20 - 24\%$) with an initial Euler-method like connection (also known as a skip-layer), before eventually plateauing out. Thus it is harder to draw any cross-system conclusions about these layering methods.
- Compared to the POD-Galerkin method, these AE-RNN models (with the exception of the simple RNN based models) possess superior predictive capabilities, especially in the more complex PDE systems. Not only do they outperform the POD-Galerkin method in short-term predictions, but they also possess better long-term statistical behaviours. All the discussed pitfalls of the POD-Galerkin approach - its inability to model statistically smaller but dynamically important modes, poor generalisation properties - are all seen first-hand.

PART V
Conclusions and Recommendations

11

Conclusion

In this concluding chapter, reflections upon the key findings and outcomes from the extensive exploration of different aspects of ML-based ROMs are presented. This thesis aimed to explore certain novel ideas, such as the contractive and multi-regime auto-encoders, as well as to perform a standardized comparative study on the networks involved in time-series modelling, across four chaotic systems of increasing complexity. This investigation has shed light on various aspects of this ROM process, revealing both successes and limitations. These insights contribute to a deeper understanding of deep learning methodologies' capabilities and reveal potential new directions for future exploration/investigation.

Chapter 9 presented the results of the various investigations into dimensionality reduction using auto-encoders. It was observed that *technically* the contractive losses 'worked as expected' and were effective in minimizing the norm of the Jacobian matrix of the latent states w.r.t. the inputs. However, a deeper analysis revealed that this was achieved not through any meaningful transformations of the latent space, but rather through an unintended minimization of the mean norm of the latent states. This minimization was reasoned to be ineffective, since the latent states act as inputs to the RNNs and are normalized once more before being fed into said networks. Thus, any changes to the norm would get negated, underscoring the need for careful consideration of model architectures and objectives.

Further discussed in Chapter 9 were the multi-regime auto-encoders, constructed by passing the regime parameters in with the system state as the input to the auto-encoders. This strategy was found to work quite well across the different tested systems and their disparate dynamical regimes (from periodic to fully chaotic). The latent spaces were observed to be well separated, with a minor performance penalty in the form of mildly higher reconstruction errors. This opens up the door to potential multi-regime model-free ROMs, and other possibilities discussed in Chapter 12. While for the first three systems straighforward fully-connected networks were used for the encoder-decoder architecture, in the Kolmogorov flow's case, owing to its spatial structure a multi-scale convolutional auto-encoder was employed. The effect of the kernel size and attention-incorporation on its reconstruction error and captured total variance was studied. It was found the kernel size had a greater impact in the relatively spatial domain considered here, since the receptive fields of the different kernel sizes eventually grew to encompass the entire input thereby accounting for global information. Thus, this marginalised the benefits of the self-attention layer, whose express purpose is to introduce global correlations. All the auto-encoders were also compared against a traditional POD approach, and found to outperform it by comfortable margins. This was attributed to its non-linear compression, the effects of which were picked up in the decoded latent space principal directions as well.

Chapter 10 presented the prediction performance results of the combined AE-RNN models, and undertook a comprehensive comparison between these combined AE-RNN models and the traditional POD-Galerkin approach, highlighting their respective strengths and weaknesses. The AE-RNNs consistently emerged as superior performers across various performance metrics, excelling in both short-term prediction accuracy and long-term statistical behaviour. These disparities became even more pronounced as the system's complexity increased. Notably, the transition from ODEs to PDEs had

a substantial impact: both the POD-Galerkin method and the simple RNN-based AE-RNN model's performance deteriorated rapidly. This decline was attributed to the latter's lack of presence of additional information channels like those in GRUs and LSTMs (forget/reset gates), and ESNs (through α), thus limiting its ability to comprehend temporal nuances effectively. Further analysis delved into the effects of RK-inspired layering and increasing layer sizes. These investigations successfully addressed the research questions posed for this section:

- The models based on ESNs, GRUs, and LSTMs exhibit remarkably consistent performance across different systems, especially as system complexity increases. Key observations include:
  - Auto-regressive training significantly enhances the prediction capabilities of BPTT-trained models such as GRUs and LSTMs. This technique leads to remarkable improvements, with $VPTs$ increasing by factors ranging from 25% to 75% compared to the base teacher-forced models.
  - Auto-regressive training has minimal impact on AE-ESNs' performance, potentially due to their optimal operational state. Additionally, precise hyper-parameter selection might instead be the critical factor, as performance improvements across varying layer sizes remain limited.
  - Naturally, this leads to the observation that with their performances being so similar, instead of going through the rigmarole of auto-regressively training a BPTT RNN, it might be easier to construct an ensemble of ESNs with properly chosen hyper-parameters.
- The application of RK-inspired RNN layering yields inconsistent results across the tested systems. In the CDV and KS systems, it yields modest benefits, generally resulting in prediction performance gains ranging from 20% to 24% with an initial Euler-method-like connection (skip-layer) before reaching a plateau. However, this pattern doesn't hold for the Lorenz '63 and the final Kolmogorov flow system, possibly due to the simultaneous training of a large multi-scale convolutional auto-encoder. Hence, it is harder to draw generalized conclusions.
- Compared to the POD-Galerkin method, all AE-RNN models (except for the simple RNN-based models) demonstrate remarkable superiority, especially in complex PDE systems. These AE-RNNs excel in short-term predictions and exhibit superior long-term statistical performance (measured via the W_1 Wasserstein distance metric between the true and predicted attractor PDFs). The limitations of the POD-Galerkin approach - its inability to capture statistically smaller yet dynamically significant modes and its poor generalization properties - are vividly demonstrated by contrasting them with these AE-RNN models.

In summation, this project comprehensively addresses the research questions posited in Chapter 2 and validates the efficacy of the proposed AE-RNN models, emphasizing their ability to outperform traditional methods, especially in complex chaotic systems with evolving dynamics. Further, the systematic framework followed here, with the selection of latent space dimensions and RNN layer sizes should serve as standardising launch-board for future investigations. Lastly, the potential of the multi-regime auto-encoders remains to be explored, and whether or not a truly multi-regime model-free prediction method can be effected using it merits investigation.

12

Recommendations

While this project covered a lot of ground, there is always room for improvement. The justification provided for choosing the dimensionality of the latent space of the auto-encoder in subsection 8.2.1, while 'sounding correct' is clearly not mathematically rigorous, and an investigation in this direction might help provide a solid theoretical footing for contructing ideal auto-encoders.

The contractive auto-encoder did not work as expected in this project case because it would go about minimizing its objective by minimizing the norm of the latent space variables. So, an alternative strategy worth exploring could be to provide the normalized Jacobian norm as the penalty term, and see how the network trains. Another possibility would be to keep the same contractive penalty as before, but introduce noise in the inputs to the decoder as well. This would effectively place a limit on how small in magnitude the latent space variables could be squeezed, since they would have to have large enough magnitudes for the decoder to meaningfully differentiate them from the added noise.

For the multi-regime auto-encoders, the performance penalty (in terms of the increased reconstruction error) might be owed to the fact that the input parameters stay constant even as the system states vary. This effectively has the network 'focusing too much' on the accurate reconstruction of the parameters versus that of the system states. This could be investigated by splitting the MSE - as done in the convolutional auto-encoder's case - into a system-state-MSE and parameter-vector-MSE - and then using their weighted sum as the loss function. The weight assigned to the parameter-vector-MSE could be optimized using something like Bayesian optimization to arrive at the lowest system-state-MSE while still maintaining latent space separation. Additionally, the structure of a coherently shared latent space also merits investigation. There are so many questions to be investigated - how does the variation of the parameter-vector play out in the latent space? Can the decoder act as a generative network? How well does the auto-encoder generalize to parameter sets and regimes it has not been trained on, and how are its interpolation and extrapolation capabilities vis-à-vis these regimes? Can a multi-regime RNN be effectively trained on such a latent space, with the position in the latent space acting as an implicit signal about the dynamics of the system? If possible, how well does this AE-RNN model generalize to parameter sets and regimes it hasn't been trained on?

Moving on to the AE-RNN modelling aspects. The general AE-RNN model used in this project (Figure 8.16) involves passing the output of the RNN through the decoder and then back through the encoder and into the RNN again. This unnecessary passing through the decoder-encoder networks introduces additional errors that accumulate over time. It would be worth exploring how these models perform if instead of this arrangement the output of the RNN is directly fed into the next time-instance, without passing it through the decoder-encoder chain.

The RK-inspired layering did not produce any meaningful variations in the results of this project, and the reason was hinted at earlier (subsection 10.3.4). In this project, the entire AE-RNN network is constructed, with the RK-inspired connections already set, and then trained. This likely leads to the gradient looping around through the RK-layers a couple of times, leading to a problem similar to the

vanishing/exploding gradient when auto-regressively training RNNs. This could be mitigated, by training only a network with a single skip layer, and once trained, putting the RNN cell of this skip layer into the RK-inspired connections after the fact. This could possibly alleviate the training problem and lead to better results.

References

[1] Nikolas O. Aksamit, Themistoklis P. Sapsis, and George Haller. *Machine-Learning Ocean Dynamics from Lagrangian Drifter Trajectories*. 2019. arXiv: 1909.12895 [math.DS].

[2] John David Anderson and John Wendt. *Computational fluid dynamics*. Vol. 206. Springer, 1995.

[3] KE ArunKumar et al. "Comparative analysis of Gated Recurrent Units (GRU), long Short-Term memory (LSTM) cells, autoregressive Integrated moving average (ARIMA), seasonal autoregressive Integrated moving average (SARIMA) for forecasting COVID-19 trends". In: *Alexandria Engineering Journal* 61.10 (2022), pp. 7585–7603.

[4] Gennaro Auricchio et al. "Computing Kantorovich-Wasserstein Distances on d-dimensional histograms using $(d + 1)$-partite graphs". In: *Advances in Neural Information Processing Systems* 31 (2018).

[5] Gregory L Baker and Jerry P Gollub. *Chaotic dynamics: an introduction*. Cambridge university press, 1996.

[6] Dor Bank, Noam Koenigstein, and Raja Giryes. "Autoencoders". In: *arXiv preprint arXiv:2003.05991* (2020).

[7] Federico Bassetti, Stefano Gualandi, and Marco Veneroni. "On the Computation of Kantorovich–Wasserstein Distances Between Two-Dimensional Histograms by Uncapacitated Minimum Cost Flows". In: *SIAM Journal on Optimization* 30.3 (2020), pp. 2441–2469.

[8] Peter Benner, Serkan Gugercin, and Karen Willcox. "A survey of projection-based model reduction methods for parametric dynamical systems". In: *SIAM review* 57.4 (2015), pp. 483–531.

[9] Peter Benner et al. *Model reduction of parametrized systems*. Springer, 2017.

[10] John Charles Butcher. "A history of Runge-Kutta methods". In: *Applied numerical mathematics* 20.3 (1996), pp. 247–260.

[11] Shengze Cai et al. "Physics-informed neural networks (PINNs) for fluid mechanics: A review". In: *Acta Mechanica Sinica* (2022), pp. 1–12.

[12] Jule G Charney and John G DeVore. "Multiple flow equilibria in the atmosphere and blocking". In: *Journal of Atmospheric Sciences* 36.7 (1979), pp. 1205–1216.

[13] Ashesh Chattopadhyay, Pedram Hassanzadeh, and Devika Subramanian. "Data-driven predictions of a multiscale Lorenz 96 chaotic system using machine-learning methods: reservoir computing, artificial neural network, and long short-term memory network". In: *Nonlinear Processes in Geophysics* 27.3 (2020), pp. 373–389.

[14] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[15] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[16] Peter Constantin and Ciprian Foias. *Navier-stokes equations*. University of Chicago Press, 2020.

[17] Steven M Cox and Paul C Matthews. "Exponential time differencing for stiff systems". In: *Journal of Computational Physics* 176.2 (2002), pp. 430–455.

[18] Daan T Crommelin, JD Opsteegh, and F Verhulst. "A mechanism for atmospheric regime behavior". In: *Journal of the atmospheric sciences* 61.12 (2004), pp. 1406–1419.

[19] Nguyen Anh Khoa Doan, Wolfgang Polifke, and Luca Magri. "Auto-encoded reservoir computing for turbulence learning". In: *International Conference on Computational Science*. Springer. 2021, pp. 344–351.

[20] Nguyen Anh Khoa Doan, Wolfgang Polifke, and Luca Magri. "Physics-informed echo state networks". In: *Journal of Computational Science* 47 (2020), p. 101237.

[21] Xiaofeng Du et al. “Single image super-resolution based on multi-scale competitive convolutional neural network”. In: *Sensors* 18.3 (2018), p. 789.

[22] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[23] Gerald A Edgar and Gerald A Edgar. *Measure, topology, and fractal geometry*. Vol. 2. Springer, 2008.

[24] Russell A Edson et al. “Lyapunov exponents of the Kuramoto–Sivashinsky PDE”. In: *The ANZIAM Journal* 61.3 (2019), pp. 270–285.

[25] Mohammad Farazmand. “An adjoint-based approach for finding invariant solutions of Navier–Stokes equations”. In: *Journal of Fluid Mechanics* 795 (2016), pp. 278–312.

[26] Mohammad Farazmand and Themistoklis P Sapsis. “A variational approach to probing extreme events in turbulent dynamical systems”. In: *Science advances* 3.9 (2017), e1701533.

[27] Bengt Fornberg. *A practical guide to pseudospectral methods*. 1. Cambridge university press, 1998.

[28] Douglas G Fox and Steven A Orszag. “Pseudospectral approximation to two-dimensional turbulence”. In: *Journal of Computational Physics* 11.4 (1973), pp. 612–619.

[29] Peter I. Frazier. *A Tutorial on Bayesian Optimization*. 2018. arXiv: 1807.02811 [stat.ML].

[30] Paul Frederickson et al. “The Liapunov dimension of strange attractors”. In: *Journal of differential equations* 49.2 (1983), pp. 185–207.

[31] Tilmann Gneiting and Adrian E. Raftery. “Weather Forecasting with Ensemble Methods”. In: *Science* 310.5746 (2005), pp. 248–249. DOI: 10.1126/science.1115255. eprint: https://www.science.org/doi/pdf/10.1126/science.1115255. URL: https://www.science.org/doi/abs/10.1126/science.1115255.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[33] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[34] Stefano Gualandi. *An Informal and Biased Tutorial on Kantorovich-Wasserstein Distances*. http://stegua.github.io/blog/2018/12/31/wasserstein-distances-an-operations-research-perspective/. [Online; accessed 28-Dec-2022]. 2018.

[35] Michael D Hartl. “Lyapunov exponents in constrained and unconstrained ordinary differential equations”. In: *arXiv preprint physics/0303077* (2003).

[36] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[37] Jan S Hesthaven, Gianluigi Rozza, Benjamin Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*. Vol. 590. Springer, 2016.

[38] Saddam Hijazi et al. “Data-driven POD-Galerkin reduced order model for turbulent flows”. In: *Journal of Computational Physics* 416 (2020), p. 109513.

[39] Sepp Hochreiter and Jürgen Schmidhuber. “LSTM can solve hard long time lag problems”. In: *Advances in neural information processing systems* 9 (1996).

[40] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[41] Herbert Jaeger. “The “echo state” approach to analysing and training recurrent neural networks-with an erratum note”. In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13.

[42] Leonid V Kantorovich. “On the translocation of masses”. In: *Journal of mathematical sciences* 133.4 (2006), pp. 1381–1382.

[43] Aly-Khan Kassam and Lloyd N Trefethen. “Fourth-order time-stepping for stiff PDEs”. In: *SIAM Journal on Scientific Computing* 26.4 (2005), pp. 1214–1233.

[44] Diederik P Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv e-prints* (2014), arXiv–1412.

[45] Yoshiki Kuramoto. “Diffusion-induced chaos in reaction systems”. In: *Progress of Theoretical Physics Supplement* 64 (1978), pp. 346–367.

[46] Sangseung Lee and Donghyun You. “Data-driven prediction of unsteady flow over a circular cylinder using deep learning”. In: *Journal of Fluid Mechanics* 879 (2019), pp. 217–254.

[47] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.

[48] Jichao Li, Xiaosong Du, and Joaquim RRA Martins. “Machine learning in aerodynamic shape optimization”. In: *Progress in Aerospace Sciences* 134 (2022), p. 100849.

[49] Edward N Lorenz. “Deterministic nonperiodic flow”. In: *Journal of atmospheric sciences* 20.2 (1963), pp. 130–141.

[50] Mantas Lukoševičius. “A practical guide to applying echo state networks”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 659–686.

[51] John L Lumley. “Coherent structures in turbulence”. In: *Transition and turbulence*. Elsevier, 1981, pp. 215–242.

[52] John Leask Lumley. “The structure of inhomogeneous turbulent flows”. In: *Atmospheric turbulence and radio wave propagation* (1967), pp. 166–178.

[53] Ingo Lütkebohle. *Bayesian Optimization*. https://www.youtube.com/watch?v=C5nqEHpdyoE. [Online; Conference on Uncertainty in AI 2018 Tutorial]. 2018.

[54] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[55] Arvind Mohan et al. “Compressed convolutional LSTM: An efficient deep learning framework to model high fidelity 3D turbulence”. In: *arXiv preprint arXiv:1903.00033* (2019).

[56] Ali H Nayfeh and Balakumar Balachandran. *Applied nonlinear dynamics: analytical, computational, and experimental methods*. John Wiley & Sons, 2008.

[57] Jaideep Pathak et al. “Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach”. In: *Physical review letters* 120.2 (2018), p. 024102.

[58] Jaideep Pathak et al. “Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 27.12 (2017), p. 121102.

[59] Suraj Pawar et al. “A deep learning enabler for nonintrusive reduced order modeling of fluid flows”. In: *Physics of Fluids* 31.8 (2019), p. 085101.

[60] Nathan Platt, L Sirovich, and N Fitzmaurice. “An investigation of chaotic Kolmogorov flows”. In: *Physics of Fluids A: Fluid Dynamics* 3.4 (1991), pp. 681–696.

[61] Stefan Posch et al. “Development of a tool for the preliminary design of large engine prechambers using machine learning approaches”. In: *Applied Thermal Engineering* 191 (2021), p. 116774.

[62] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced basis methods for partial differential equations: an introduction*. Vol. 92. Springer, 2015.

[63] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Multistep neural networks for data-driven discovery of nonlinear dynamical systems”. In: *arXiv preprint arXiv:1801.01236* (2018).

[64] Michael T Rosenstein, James J Collins, and Carlo J De Luca. “A practical method for calculating largest Lyapunov exponents from small data sets”. In: *Physica D: Nonlinear Phenomena* 65.1-2 (1993), pp. 117–134.

[65] Themistoklis P Sapsis and Andrew J Majda. “Blended reduced subspace algorithms for uncertainty quantification of quadratic systems with a stable mean state”. In: *Physica D: Nonlinear Phenomena* 258 (2013), pp. 61–76.

[66] Gregory I Sivashinsky. “Nonlinear analysis of hydrodynamic instability in laminar flames—I. Derivation of basic equations”. In: *Acta astronautica* 4.11 (1977), pp. 1177–1206.

[67] A. Skopenkov. *Embedding and knotting of manifolds in Euclidean spaces*. 2006. arXiv: math/0604045 [math.GT].

[68] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf.

[69] *Tangent Plane of a Sphere*. https://math.stackexchange.com/questions/1686437/what-is-the-union-of-all-the-tangent-plane-at-every-point-of-a-sphere. [Online; accessed 16-May-2022]. 2016.

[70] H. Tennekes and J.L. Lumley. *A First Course in Turbulence*. MIT Press, 2018. ISBN: 9780262536301. URL: https://books.google.nl/books?id=4L34DwAAQBAJ.

[71] Marco Tezzele, Francesco Ballarin, and Gianluigi Rozza. "Combined parameter and model reduction of cardiovascular problems by means of active subspaces and POD-Galerkin methods". In: *Mathematical and numerical modeling of the cardiovascular system and applications* (2018), pp. 185–207.

[72] Cédric Villani. *Topics in optimal transportation*. Vol. 56. American Mathematical Soc., 2003.

[73] Pantelis R Vlachas et al. "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2213 (2018), p. 20170844.

[74] Pantelis R Vlachas et al. "Forecasting of spatio-temporal chaotic dynamics with recurrent neural networks: A comparative study of reservoir computing and backpropagation algorithms". In: *arXiv preprint arXiv:1910.05266* (2019).

[75] Zhong Yi Wan et al. "Data-assisted reduced-order modeling of extreme events in complex dynamical systems". In: *PloS one* 13.5 (2018), e0197704.

[76] Jonathan A. Weyn, Dale R. Durran, and Rich Caruana. "Can Machines Learn to Predict Weather? Using Deep Learning to Predict Gridded 500-hPa Geopotential Height From Historical Weather Data". In: *Journal of Advances in Modeling Earth Systems* 11.8 (2019), pp. 2680–2693. DOI: https://doi.org/10.1029/2019MS001705. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019MS001705. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019MS001705.

[77] Jin-Long Wu et al. "Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems". In: *Journal of Computational Physics* 406 (2020), p. 109209.

[78] Pin Wu et al. "Reduced order model using convolutional auto-encoder with self-attention". In: *Physics of Fluids* 33.7 (2021), p. 077107.

[79] Zhe Wu et al. "Machine learning modeling and predictive control of nonlinear processes using noisy data". In: *AIChE Journal* 67.4 (2021), e17164.

[80] Jin Xu et al. "Reluplex made more practical: Leaky ReLU". In: *2020 IEEE Symposium on Computers and Communications (ISCC)*. 2020, pp. 1–7. DOI: 10.1109/ISCC50000.2020.9219587.

[81] Xinghui Yan et al. "Aerodynamic shape optimization using a novel optimizer based on machine learning techniques". In: *Aerospace Science and Technology* 86 (2019), pp. 826–835.

[82] Han Zhang et al. "Self-attention generative adversarial networks". In: *International conference on machine learning*. PMLR. 2019, pp. 7354–7363.

[83] Qinyu Zhuang et al. "Model order reduction based on Runge–Kutta neural networks". In: *Data-Centric Engineering* 2 (2021).

PART VI
Appendices

A

General Workflow

A.1. Simulation and Data Generation

A.1.1. The Lorenz '63 System

The MLE, the KY dimension and the Lyapunov time of the different cases are given in Table A.1. Note the $(\sigma, \rho, \beta) = (10, 35, 1.33)$ case, while definitely chaotic (since it does not settle down at any attractor position), has a high Lyapunov time implying that it has relatively stable orbital trajectories around the attractor positions. This can also be seen in Figure 8.1e.

(σ, ρ, β)	λ_{MLE}	KY dim	t_L
$(10, 28, 2.67)$	9.06×10^{-1}	2.06	1.10
$(10, 35, 1.33)$	7.36×10^{-1}	2.06	1.36
$(10, 35, 2.67)$	1.04×10^{0}	2.07	0.96
$(10, 45, 1.33)$	9.30×10^{-1}	2.07	1.08
$(10, 45, 2.67)$	1.21×10^{0}	2.08	0.82
$(10, 35, 1.33)$	1.51×10^{-3}	1.14	659.20
$(10, 35, 2.67)$	1.04×10^{0}	2.04	0.96
$(10, 45, 1.33)$	6.06×10^{-1}	2.02	1.65
$(10, 45, 2.67)$	1.17×10^{0}	2.05	0.85

Table A.1: MLE, KY dimension and Lyapunov times for different parameter sets of the Lorenz system

A.1.2. The Charney-DeVore System

The MLE, the KY dimension and the Lyapunov time of the different cases are given in Table A.2.

(x_1^*, x_4^*)	λ_{MLE}	KY dim	t_L
$(0.95, -0.76095)$	2.70×10^{-2}	2.32	37.09
$(0.99, -0.79299)$	3.05×10^{-4}	1.20	3275.35

Table A.2: MLE, KY dimension and Lyapunov times for different parameter sets of the CDV system

The high Lyapunov time, and periodic-looking behaviour in Figure 8.3 prompted a deeper look into the case $(x_1^*, x_4^*) = (0.99, -0.79299)$. The normalized auto-covariance function, as defined Equation A.1, is plotted in Figure A.1. Clearly, the regular spikes in intervals of 140 time-units confirms the strong periodicity of this case. Note that not only are there spikes present in the ACF, but they also peak at exactly 1.0, indicating perfect matching with the original un-shifted data at these intervals.

$$ACF(\tau) = \mathbb{E}_t \left[\frac{1}{|\Omega|} \int_\Omega \left\langle \left(\frac{\boldsymbol{u}(t) - \overline{\boldsymbol{u}}}{\mathrm{std}(\boldsymbol{u})} \right), \left(\frac{\boldsymbol{u}(t+\tau) - \overline{\boldsymbol{u}}}{\mathrm{std}(\boldsymbol{u})} \right) \right\rangle \mathrm{d}\Omega \right] \tag{A.1}$$

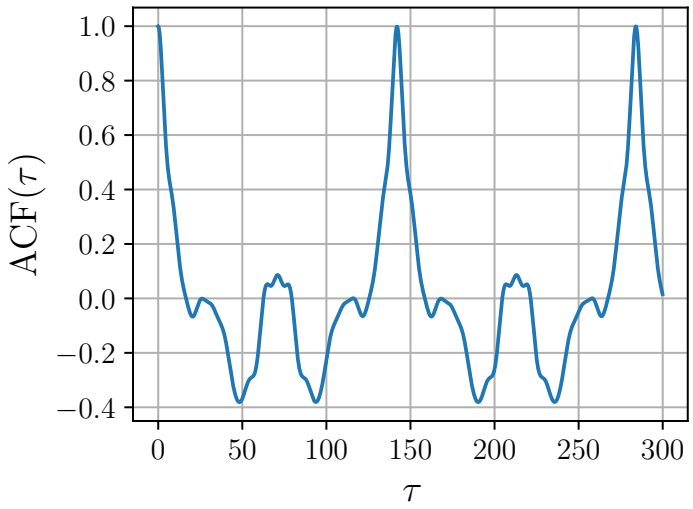


Figure A.1: $ACF(\tau)$ for the CDV system with $(x_1^*, x_4^*) = (0.99, -0.79299)$

A.1.3. The Kuramoto-Sivashinsky System

The mean kinetic energy, mean turbulent kinetic energy and mean dissipation rate are defined below and plotted in Figure A.2, Figure A.3 and Figure A.4 (respectively).

$$KE(\boldsymbol{u}) = \frac{1}{|\Omega|}\int_\Omega \frac{1}{2}|\boldsymbol{u}|^2 \mathrm{d}\Omega \tag{A.2}$$

$$TKE(\boldsymbol{u}) = \frac{1}{|\Omega|}\int_\Omega \frac{1}{2}|\boldsymbol{u} - \overline{\boldsymbol{u}}|^2 \mathrm{d}\Omega \tag{A.3}$$

$$D(\boldsymbol{u}) = \frac{1}{|\Omega|}\int_\Omega |\nabla \boldsymbol{u}|^2 \mathrm{d}\Omega \tag{A.4}$$

From these, it can clearly be observed that only the first two cases are truly chaotic, with messy KE, TKE and D time-evolution. The last two cases appear to have a constant KE, with periodic variations in the TKE and D. Also, note that the KE and TKE are much larger in magnitude for the case $(\nu_1, \nu_2, \nu_3) = (1, 2, 1)$. This can be attributed to the fact that ν_2 is directly responsible for adding energy into the system (Section 6.3), and a higher value of ν_2 leads to a more energetic system.

The MLE, the KY dimension and the Lyapunov time of the different cases are given in Table A.3.

(ν_1, ν_2, ν_3)	λ_{MLE}	KY dim	t_L
$(1, 1, 1)$	7.17×10^{-2}	7.83	13.95
$(2, 1, 1)$	7.55×10^{-2}	7.86	13.25
$(1, 2, 1)$	3.12×10^{-3}	1.67	320.76
$(1, 1, 2)$	2.28×10^{-3}	1.63	438.57

Table A.3: MLE, KY dimension and Lyapunov times for different parameter sets of the KS system

The similarly high Lyapunov times and periodic-looking behaviour prompted further analysis using the ACF as defined above. The resulting plots for the last two parameter sets are shown in Figure A.5. Once again, the clear spikes peaking at exactly 1.0 confirm periodicity in these cases (at 44.8 time-units and 91.6 time-units, respectively).

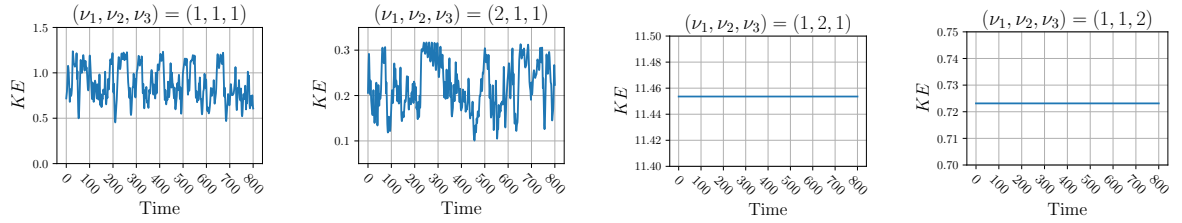


Figure A.2: Time evolution of the mean kinetic energy (KE) of the KS system with different parameter sets.

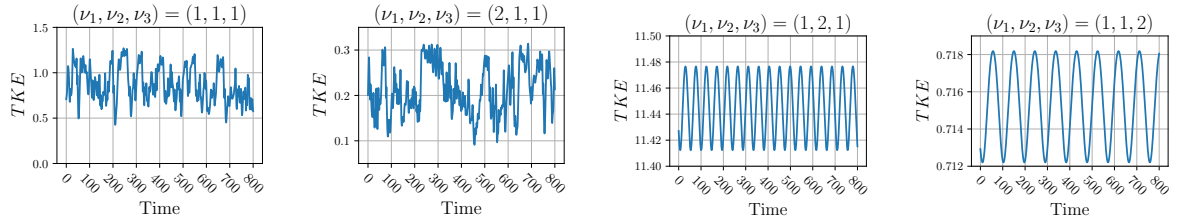


Figure A.3: Time evolution of the mean turbulent kinetic energy (TKE) of the KS system with different parameter sets.

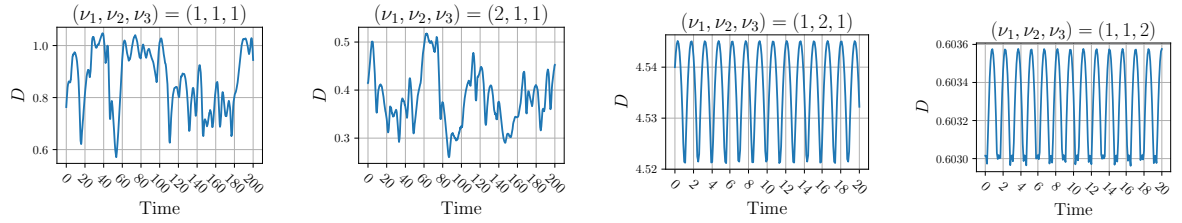


Figure A.4: Time evolution of the mean dissipation rate (D) of the KS system with different parameter sets.

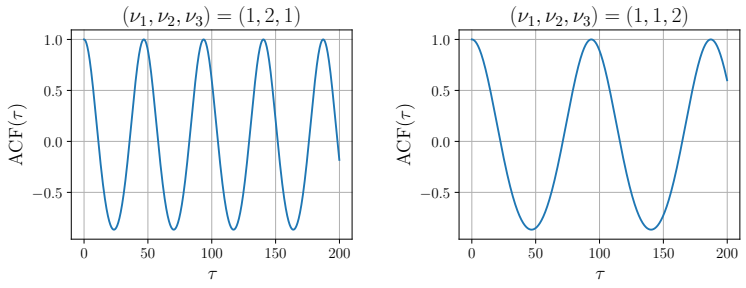


Figure A.5: $ACF(\tau)$ for the KS system with $(\nu_1, \nu_2, \nu_3) = (1, 2, 1)$a, $(1, 1, 2)$b

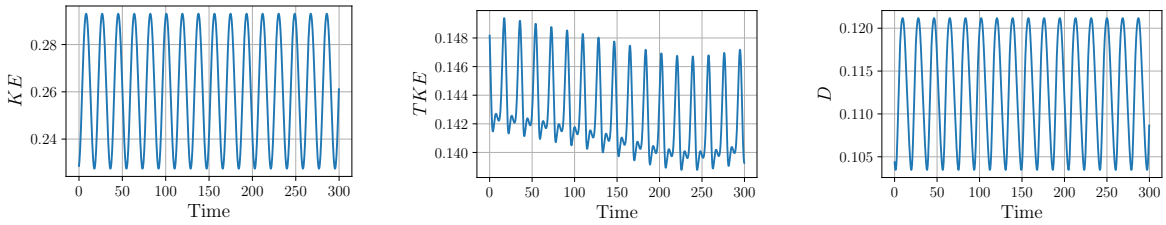
A.1.4. The Kolmogorov Flow System

The time-evolution of the mean kinetic energy, mean turbulent kinetic energy and mean dissipation rate (as defined in Equation A.2, Equation A.3 and Equation A.4) is plotted in Figure A.6 and Figure A.7. Note the characteristic spikes in KE and D in Figure A.7 for the $Re = 40$ case, and the quasi-periodic variation in the $Re = 30$ case.

The MLE, the KY dimension and the Lyapunov time of the different cases are given in Table A.4. The high Lyapunov time of the $Re = 30$ case is unsurprising since this system is known to be quasi-periodic [60][1].

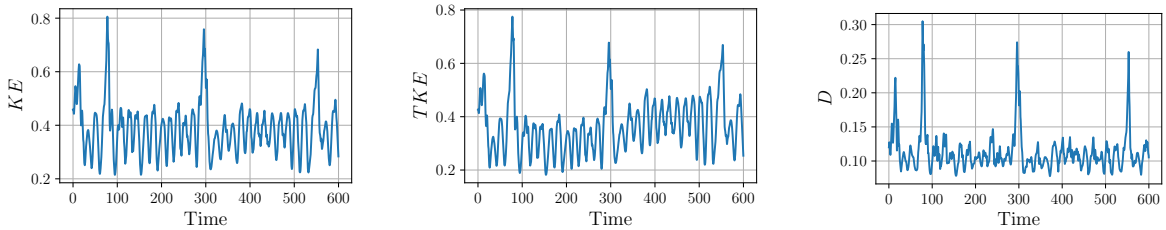
Re	λ_{MLE}	KY dim	t_L
30	3.64×10^{-3}	3.90	274.78
40	7.65×10^{-2}	12.19	13.06

Table A.4: MLE, KY dimension and Lyapunov times for different Reynolds numbers (Kolmogorov flow)



(a) **(b)** **(c)**

Figure A.6: Snippets of the time-evolution of the KE, TKE and D, for the Kolmogorov flow system ($Re = 30$).



(a) **(b)** **(c)**

Figure A.7: Snippets of the time-evolution of the KE, TKE and D, for the Kolmogorov flow system ($Re = 40$).

A.2. Auto-encoders

A.2.1. Layering and Network Architecture

The auto-encoder layer sizes for the different chaotic systems are given below. Note that for the 2D incompressible Navier Stokes system, the layering scheme for only the kernel filter size 7×7 is presented. The layering schemes for the other filter sizes are the same, with only the padding and cropping layers being affected.

[1]Check the discussion on the $4.72 \leq \Omega/\Omega_c < 5.9$ regime. The case of $Re = 30$ translates to $\Omega/\Omega_c = 5.3$. Platt, Sirovich, and Fitzmaurice note the dominance of one single frequency in this regime and call it 'meta-stable'.

	layer type	layer size
	input	3 / 6
Encoder	intermediate	16
	intermediate	8
	intermediate	4
	final	2
Decoder	intermediate	4
	intermediate	8
	intermediate	16
	final	3 / 6

Table A.5: Auto-encoder layer sizes for the Lorenz '63 system.

	layer type	layer size
	input	6 / 12
Encoder	intermediate	16
	intermediate	8
	intermediate	8
	final	5
Decoder	intermediate	8
	intermediate	8
	intermediate	16
	final	6 / 12

Table A.6: Auto-encoder layer sizes for the CDV system.

	layer type	layer size
	input	64 / 67
Encoder	intermediate	128
	intermediate	96
	intermediate	64
	intermediate	48
	intermediate	32
	intermediate	24
	final	16
Decoder	intermediate	24
	intermediate	32
	intermediate	48
	intermediate	64
	intermediate	96
	intermediate	128
	final	64 / 67

Table A.7: Auto-encoder layer sizes for the KS system.

	layer type	output size
	input	$\{2, 50, 50\}$ / $\{3, 50, 50\}$
Encoder	periodic padding	$\{2, 54, 54\}$ / $\{3, 54, 54\}$
	convolution (stride 2)	$\{8, 24, 24\}$
	batch normalization	$\{8, 24, 24\}$
	activation (ELU)	$\{8, 24, 24\}$
	periodic padding	$\{8, 30, 30\}$
	convolution (stride 2)	$\{16, 12, 12\}$
	batch normalization	$\{16, 12, 12\}$
	activation (ELU)	$\{16, 12, 12\}$
	attention module	$\{16, 12, 12\}$
	periodic padding	$\{16, 18, 18\}$
	convolution (stride 2)	$\{32, 6, 6\}$
	batch normalization	$\{32, 6, 6\}$
	activation (ELU)	$\{32, 6, 6\}$
	attention module	$\{32, 6, 6\}$
	periodic padding	$\{32, 10, 10\}$
	convolution (stride 2)	$\{3, 3, 3\}$
	batch normalization	$\{3, 3, 3\}$
	activation (tanh)	$\{3, 3, 3\}$
Decoder	periodic padding	$\{3, 9, 9\}$
	transposed convolution (stride 2)	$\{32, 23, 23\}$
	cropping	$\{32, 6, 6\}$
	batch normalization	$\{32, 6, 6\}$
	activation (elu)	$\{32, 6, 6\}$
	periodic padding	$\{32, 12, 12\}$
	transposed convolution (stride 2)	$\{16, 29, 29\}$
	cropping	$\{16, 12, 12\}$
	batch normalization	$\{16, 12, 12\}$
	activation (elu)	$\{16, 12, 12\}$
	attention module	$\{16, 12, 12\}$
	periodic padding	$\{16, 18, 18\}$
	transposed convolution (stride 2)	$\{8, 41, 41\}$
	cropping	$\{8, 24, 24\}$
	batch normalization	$\{8, 24, 24\}$
	activation (elu)	$\{8, 24, 24\}$
	attention module	$\{8, 24, 24\}$
	periodic padding	$\{8, 30, 30\}$
	transposed convolution (stride 2)	$\{2, 65, 65\}$ / $\{3, 65, 65\}$
	cropping	$\{2, 50, 50\}$ / $\{3, 50, 50\}$
	batch normalization	$\{2, 50, 50\}$ / $\{3, 50, 50\}$
	activation (elu)	$\{2, 50, 50\}$ / $\{3, 50, 50\}$

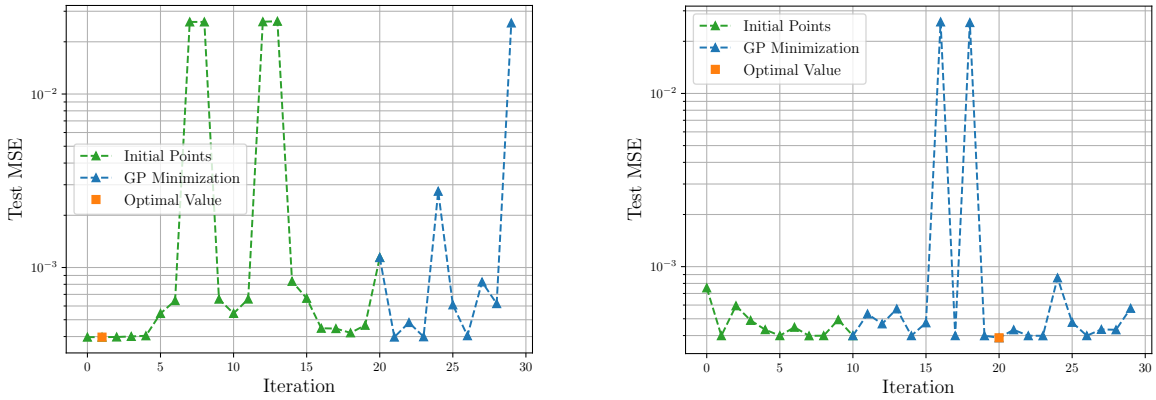
Table A.8: Auto-encoder layer sizes for the Kolmogorov flow system, with the kernel filter size set to 7×7. Other filter sizes follow the same layer structures, with different values for the periodic padding and cropping layers.

B

Dimensionality Reduction Using Auto-encoders

B.1. The Lorenz '63 System

B.1.1. Bayesian Optimization



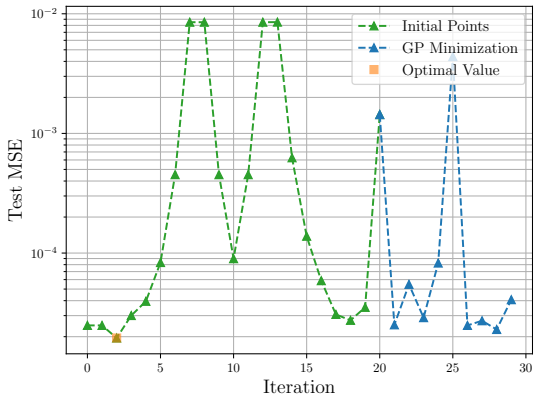
(a) Search for f_{noise} and λ_{reg}.

(b) Search for $\lambda_{\text{contractive}}$.

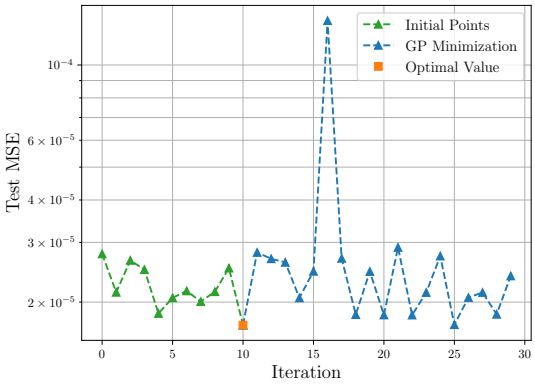
Figure B.1: Iteration-wise evolution of the MSE for the Bayesian optimization.

B.2. The Charney-DeVore System

B.2.1. Bayesian Optimization



(a) Search for f_{noise} and λ_{reg}.

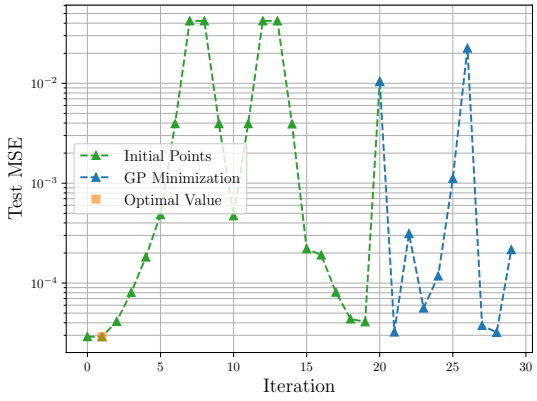


(b) Search for $\lambda_{\text{contractive}}$.

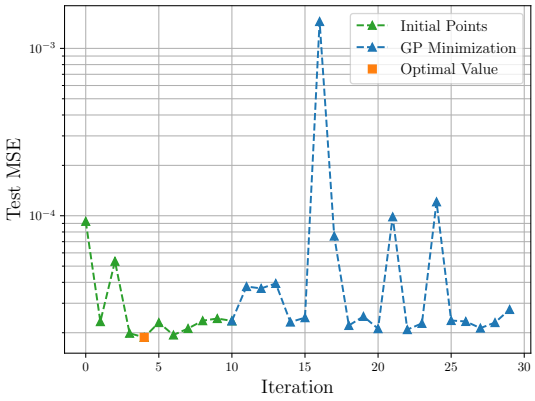
Figure B.2: Iteration-wise evolution of the MSE for the Bayesian optimization.

B.3. The Kuramoto-Sivashinsky System

B.3.1. Bayesian Optimization



(a) Search for f_{noise} and λ_{reg}.



(b) Search for $\lambda_{\text{contractive}}$.

Figure B.3: Iteration-wise evolution of the MSE for the Bayesian optimization.

B.4. The Kolmogorov Flow System

B.4.1. Bayesian Optimization

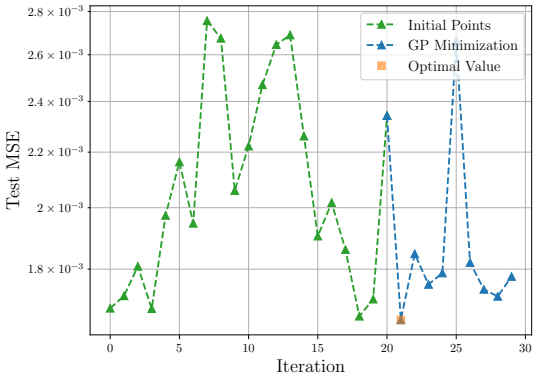


Figure B.4: Search for f_{noise} and λ_{reg}, and the iteration-wise evolution of the MSE.

B.4.2. AE and POD

Kernel Filter Size	Attention Layer	AE	POD
3×3	No	1.22×10^{-1}	1.76×10^{-1}
5×5		1.01×10^{-1}	
7×7		9.36×10^{-2}	
3×3, 5×5, 7×7		8.72×10^{-2}	
3×3	Yes	1.19×10^{-1}	
5×5		1.03×10^{-1}	
7×7		9.31×10^{-2}	
3×3, 5×5, 7×7		8.65×10^{-2}	

Table B.1: Reconstruction error (NRMSE) for the auto-encoders with changing kernel filter sizes.

Kernel Filter Size	Attention Layer	AE	POD
3×3	No	97.56%	96.67%
5×5		98.15%	
7×7		98.26%	
3×3, 5×5, 7×7		98.51%	
3×3	Yes	97.97%	
5×5		98.26%	
7×7		98.39%	
3×3, 5×5, 7×7		98.58%	

Table B.2: Captured total variance for the auto-encoders with changing kernel filter sizes.

Latent Space Dimension	AE	POD
$\{2, 3, 3\}$	1.07×10^{-1}	2.19×10^{-1}
$\{3, 3, 3\}$	8.65×10^{-2}	1.76×10^{-1}
$\{4, 3, 3\}$	7.44×10^{-2}	1.33×10^{-1}
$\{5, 3, 3\}$	6.86×10^{-2}	9.72×10^{-2}

Table B.3: Reconstruction error (NRMSE) for the multi-scale auto-encoder with changing latent space dimensionality.

Latent Space Dimension	AE	POD
$\{2, 3, 3\}$	94.47%	91.26%
$\{3, 3, 3\}$	94.89%	93.07%
$\{4, 3, 3\}$	95.29%	94.48%
$\{5, 3, 3\}$	95.30%	95.35%

Table B.4: Captured total variance for the multi-scale auto-encoder with changing latent space dimensionality.

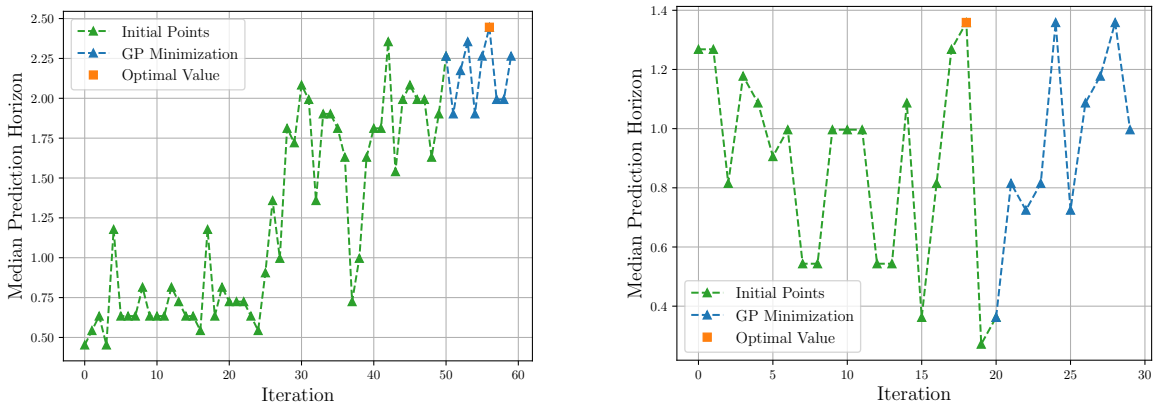
C

Time Series Predictions using RNNs

C.1. The Lorenz '63 System

Additional plots for Section 10.1.

C.1.1. Bayesian Optimization

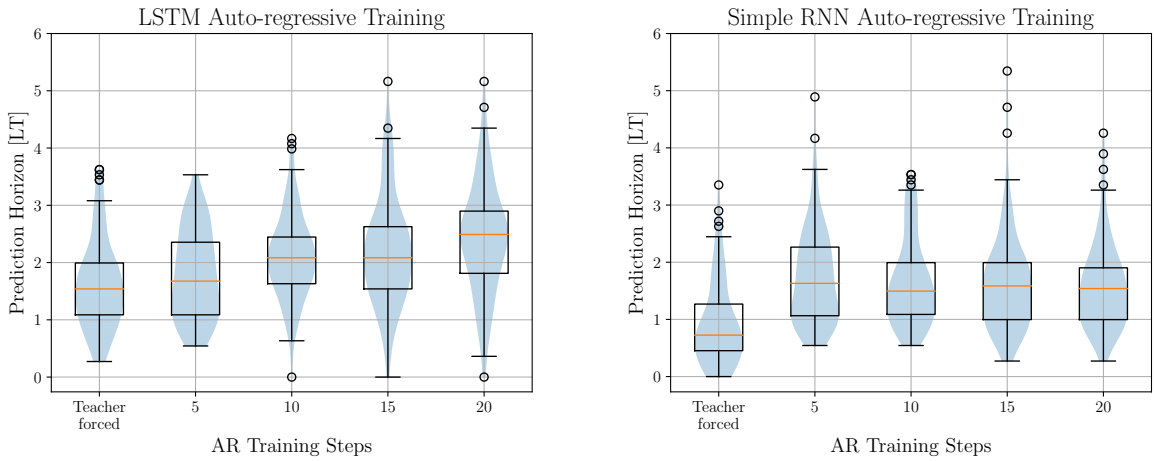


(a) Search for ω_{in}, ρ_{res}, α, f_{noise} and λ_{reg} for the ESN.

(b) Search for f_{noise} and λ_{reg} for the GRU.

Figure C.1: Iteration-wise evolution of the MSE for the Bayesian optimization.

C.1.2. Auto-regressively Trained BPTT RNNs

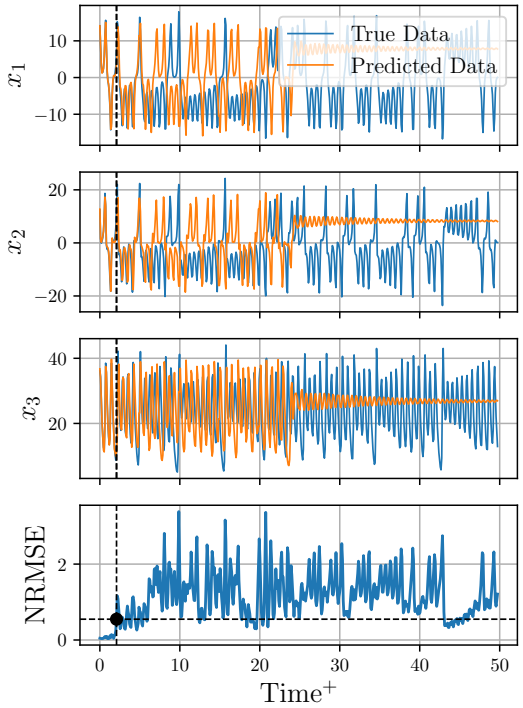


(a) Auto-regressive training of the LSTM ($n^r = 80$).

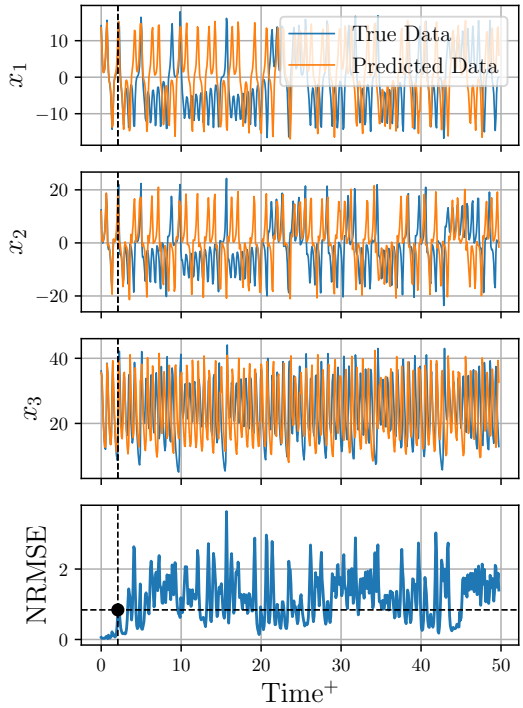
(b) Auto-regressive training of the Simple RNN ($n^r = 80$).

Figure C.2: Evolution of the prediction horizon distribution as auto-regressive training progresses.

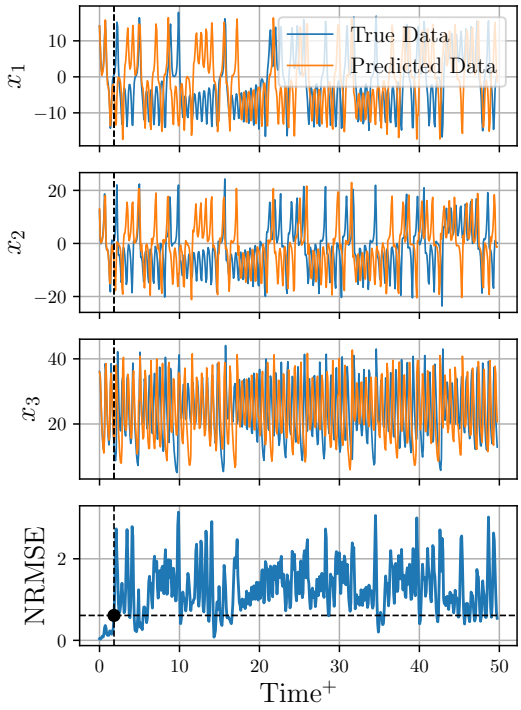
C.1.3. Long-term Evolution of Sample Trajectories



(a) Sample trajectory evolution for the AE-LSTM model.



(b) Sample trajectory evolution for the AE-ESN model.



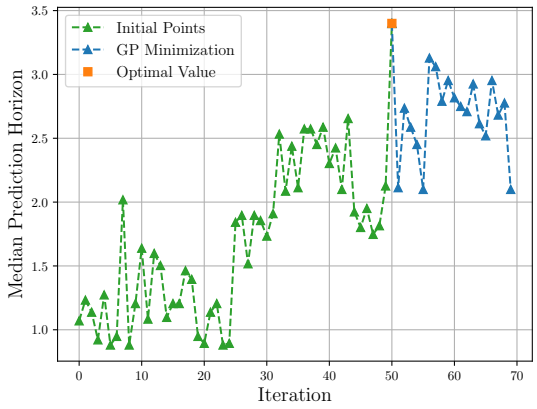
(c) Sample trajectory evolution for the AE-SimpleRNN model.

Figure C.3: Long-term evolution of sample trajectories for the Lorenz '63 system.

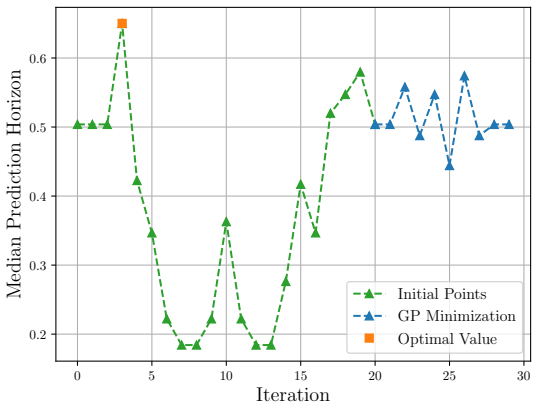
C.2. The Charney-DeVore System

Additional plots for Section 10.2.

C.2.1. Bayesian Optimization



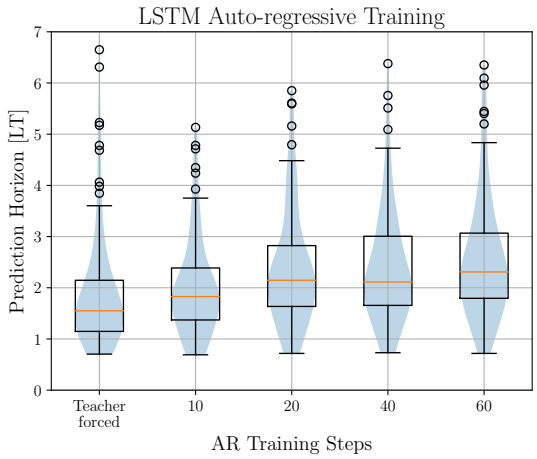
(a) Search for ω_{in}, ρ_{res}, α, f_{noise} and λ_{reg} for the ESN.



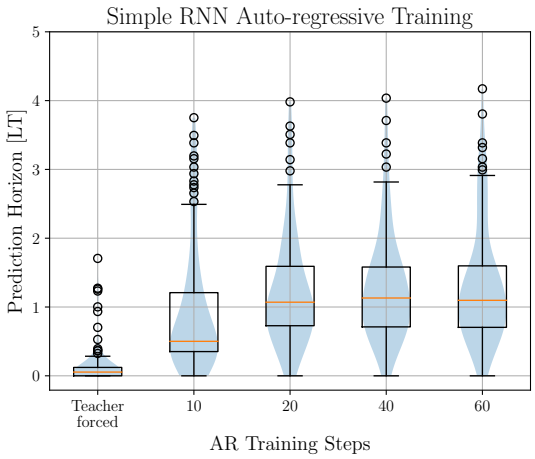
(b) Search for f_{noise} and λ_{reg} for the GRU.

Figure C.4: Iteration-wise evolution of the MSE for the Bayesian optimization.

C.2.2. Auto-regressively Trained BPTT RNNs



(a) Auto-regressive training of the LSTM ($n^r = 80$).



(b) Auto-regressive training of the Simple RNN ($n^r = 80$).

Figure C.5: Evolution of the prediction horizon distribution as auto-regressive training progresses.

C.2.3. Long-term Evolution of Sample Trajectories

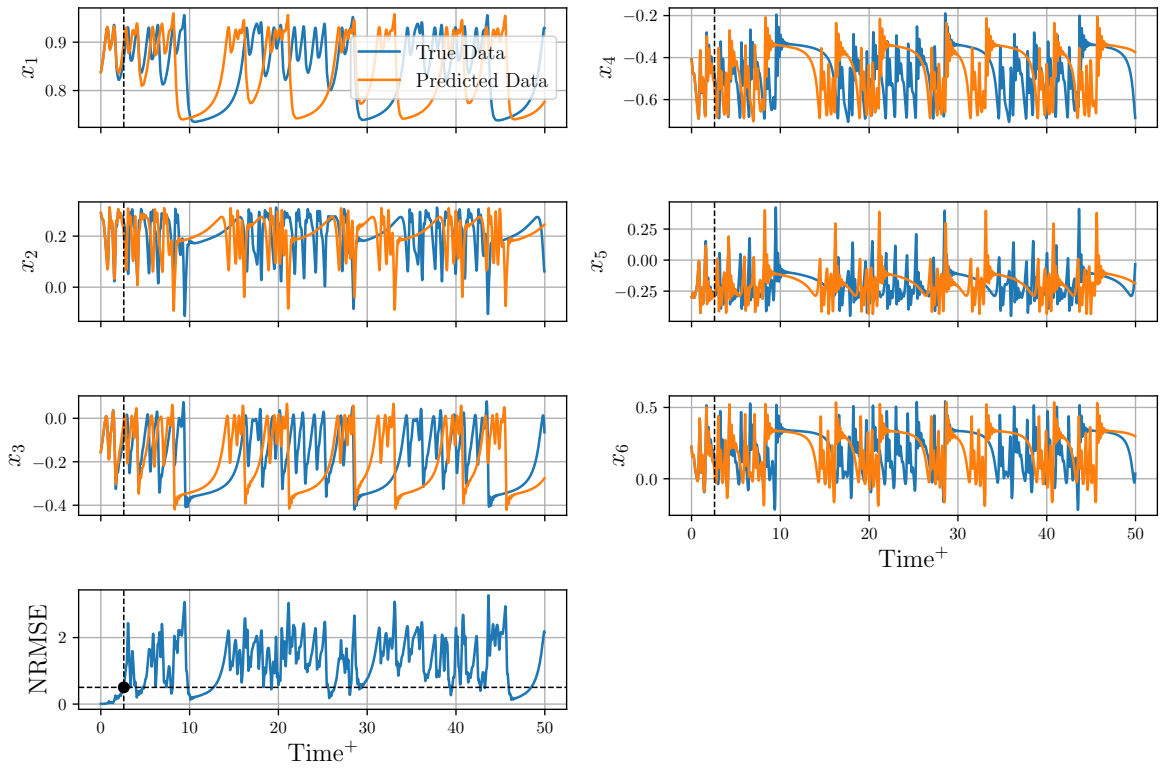


Figure C.6: Sample trajectory evolution for the AE-LSTM model.

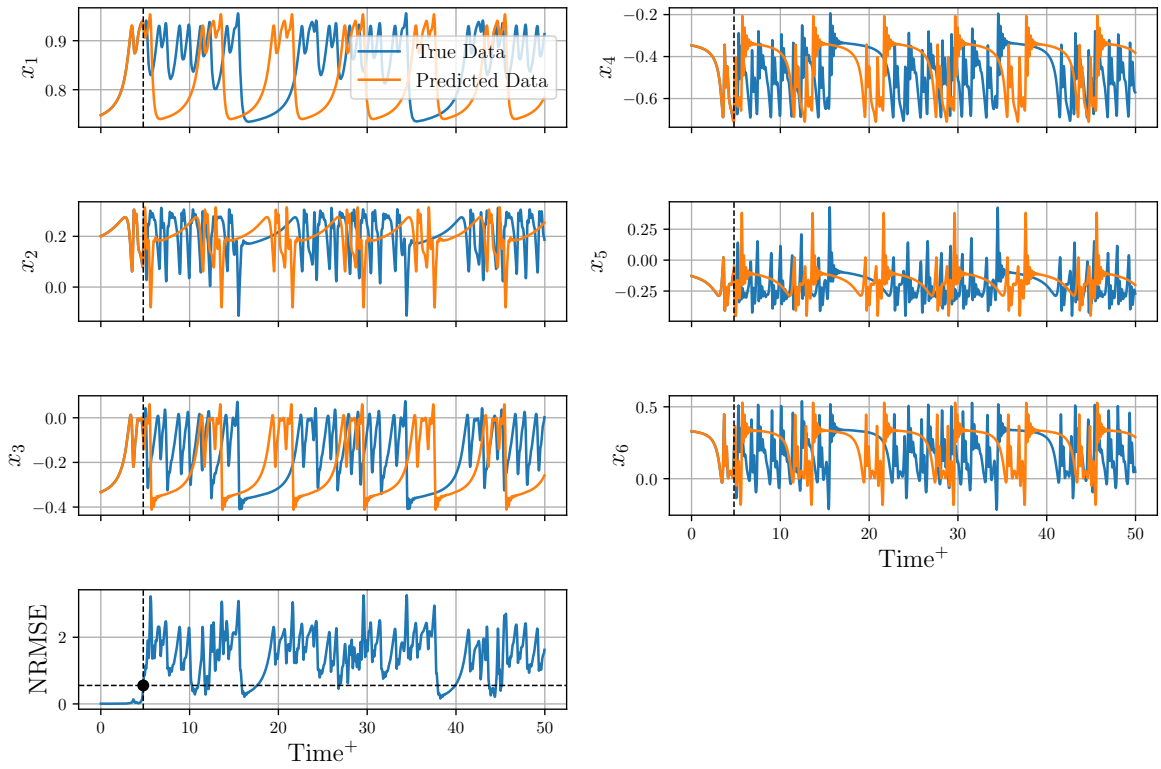


Figure C.7: Sample trajectory evolution for the AE-ESN model.

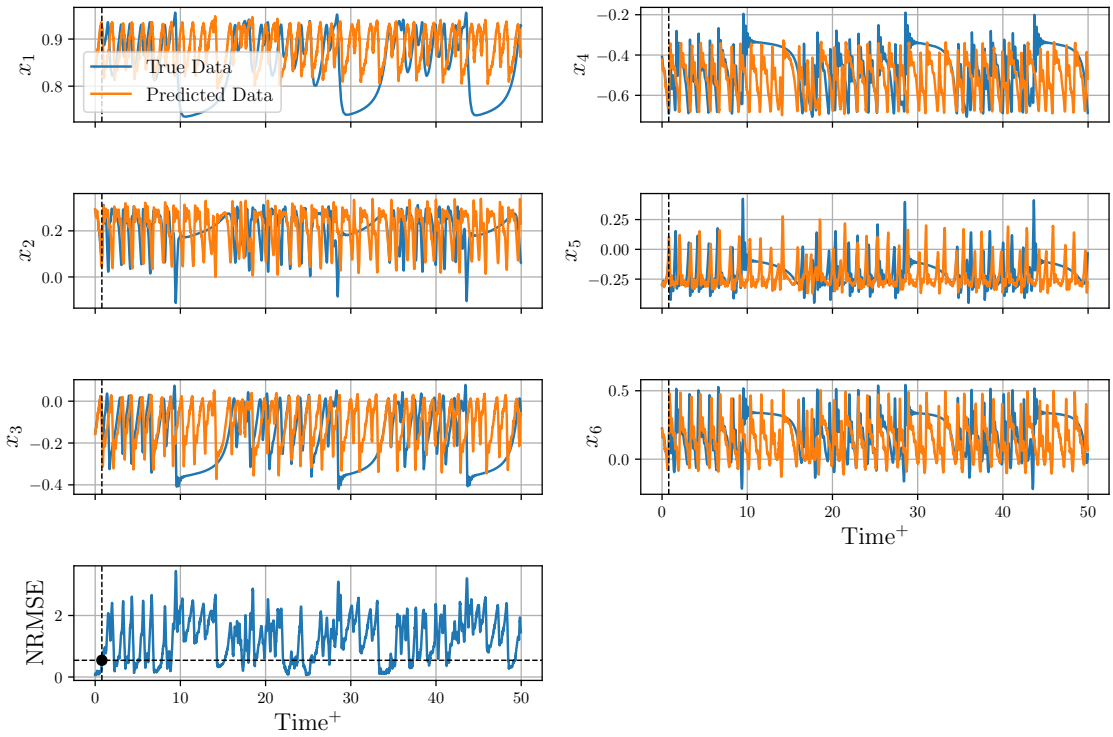
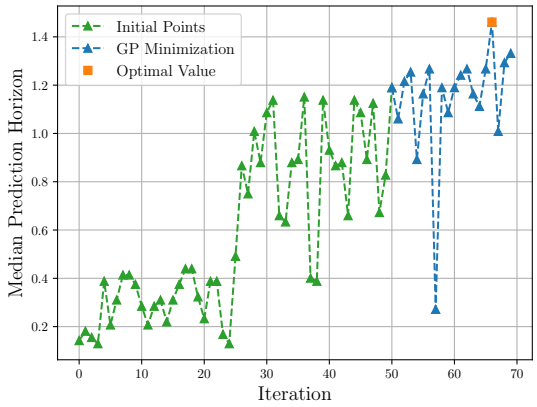


Figure C.8: Sample trajectory evolution for the AE-SimpleRNN model.

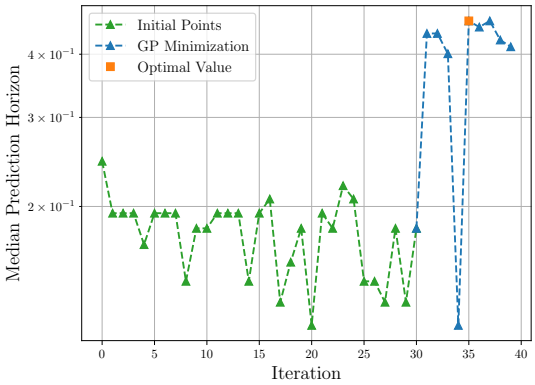
C.3. The Kuramoto-Sivashinsky System

Additional plots for Section 10.3.

C.3.1. Bayesian Optimization



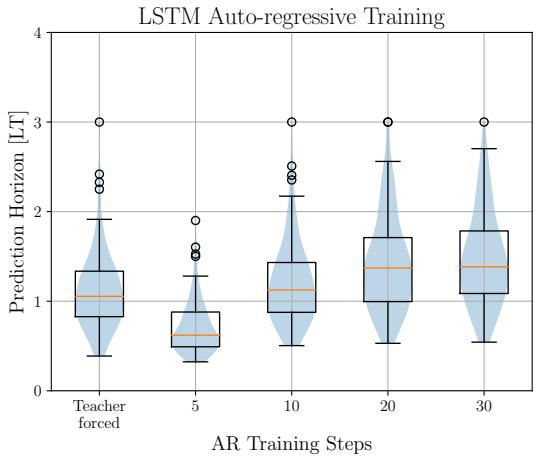
(a) Search for ω_{in}, ρ_{res}, α, f_{noise} and λ_{reg} for the ESN.



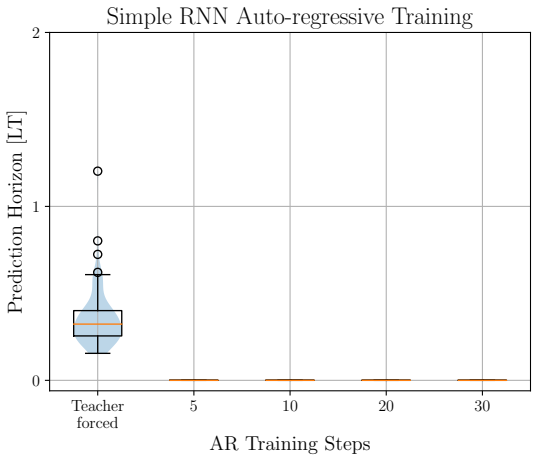
(b) Search for f_{noise} and λ_{reg} for the GRU.

Figure C.9: Iteration-wise evolution of the MSE for the Bayesian optimization.

C.3.2. Auto-regressively Trained BPTT RNNs



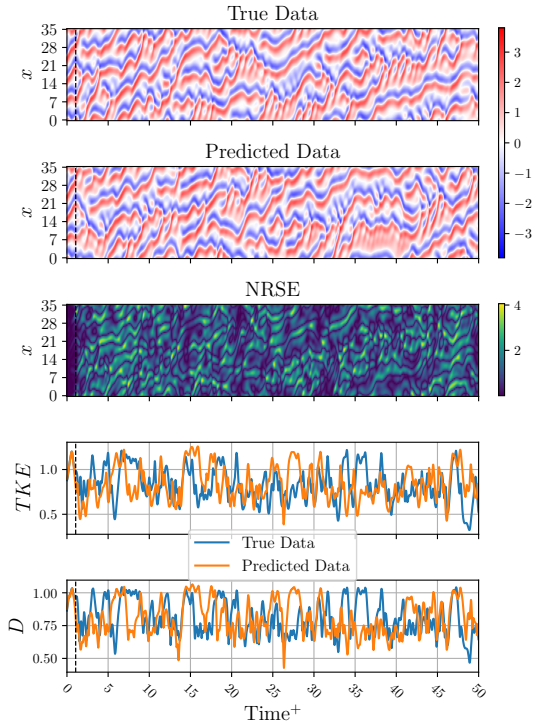
(a) Auto-regressive training of the LSTM ($n^r = 80$).



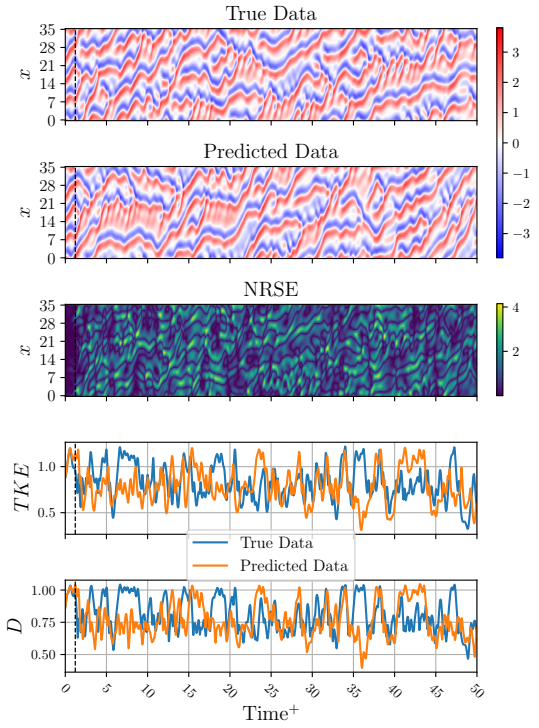
(b) Auto-regressive training of the Simple RNN ($n^r = 80$).

Figure C.10: Evolution of the prediction horizon distribution as auto-regressive training progresses.

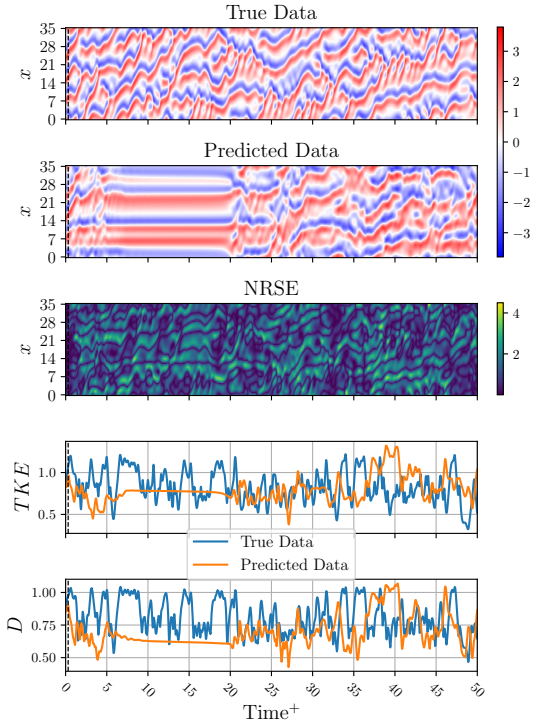
C.3.3. Long-term Evolution of Sample Trajectories



(a) Sample trajectory evolution for the AE-LSTM model.



(b) Sample trajectory evolution for the AE-ESN model.



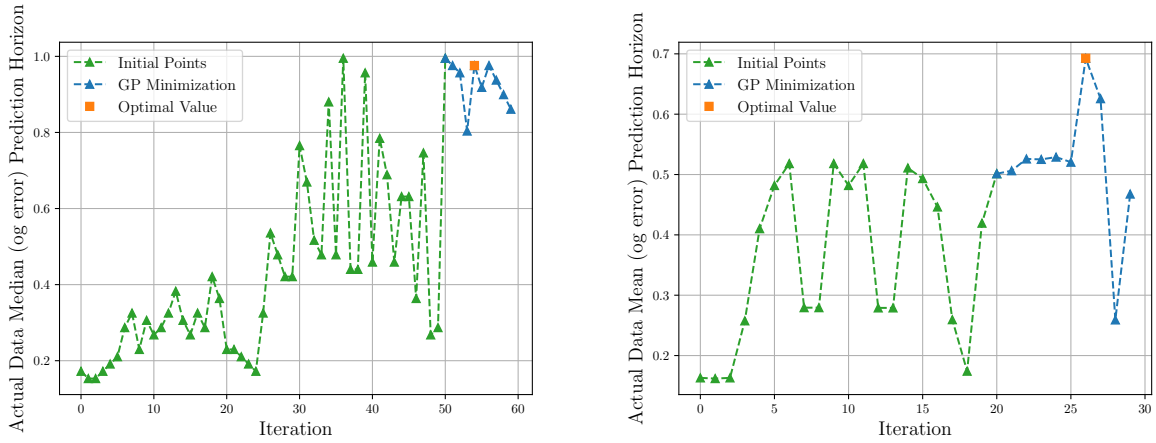
(c) Sample trajectory evolution for the AE-SimpleRNN model.

Figure C.11: Long-term evolution of sample trajectories for the Kuramoto-Sivashinsky system.

C.4. The Kolmogorov Flow System

Additional plots for Section 10.4.

C.4.1. Bayesian Optimization

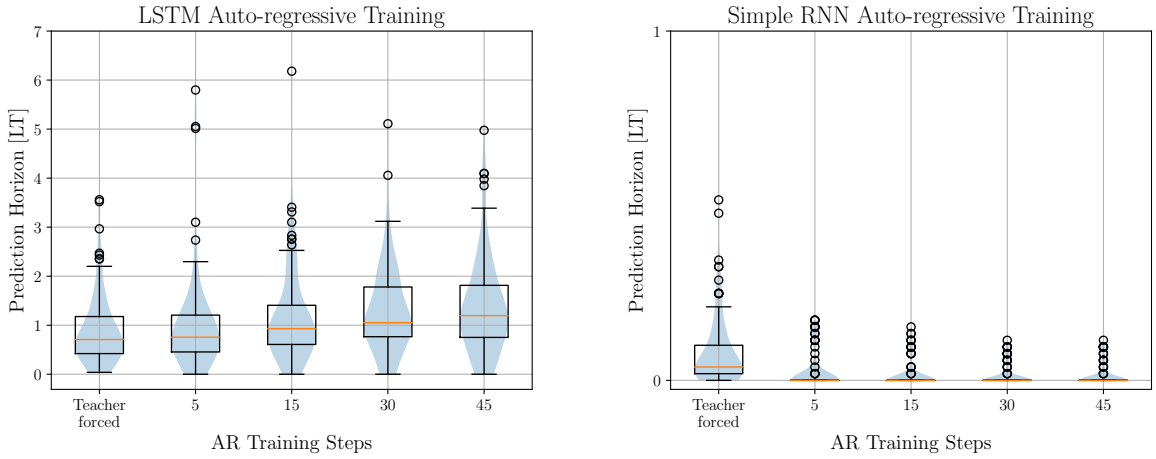


(a) Search for ω_{in}, ρ_{res}, α, f_{noise} and λ_{reg} for the ESN.

(b) Search for f_{noise} and λ_{reg} for the GRU.

Figure C.12: Iteration-wise evolution of the MSE for the Bayesian optimization.

C.4.2. Auto-regressively Trained BPTT RNNs

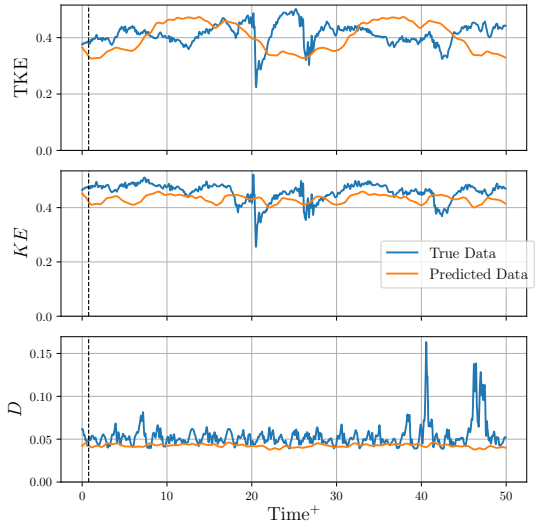


(a) Auto-regressive training of the LSTM ($n^r = 80$).

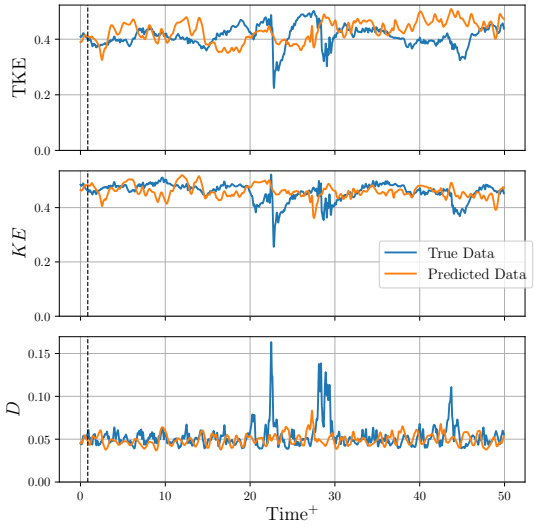
(b) Auto-regressive training of the Simple RNN ($n^r = 80$).

Figure C.13: Evolution of the prediction horizon distribution as auto-regressive training progresses.

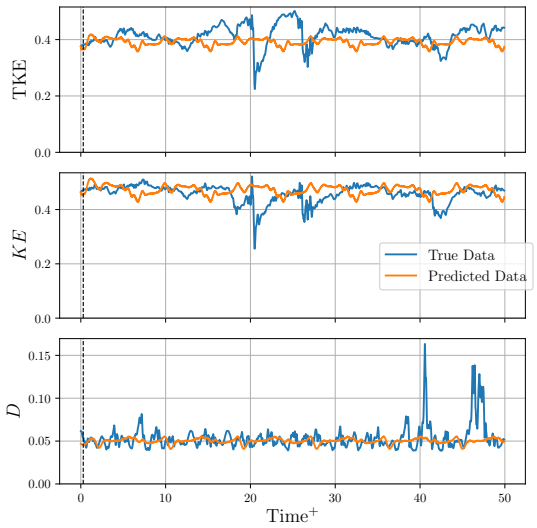
C.4.3. Long-term Evolution of Sample Trajectories



(a) Sample trajectory evolution for the AE-LSTM model.



(b) Sample trajectory evolution for the AE-ESN model.



(c) Sample trajectory evolution for the AE-SimpleRNN model.

Figure C.14: Long-term evolution of sample trajectories for the Kolmogorov flow system.

C.4.4. Typical Evolution of u, v and ω for a Sample Trajectory - AE-GRU Model

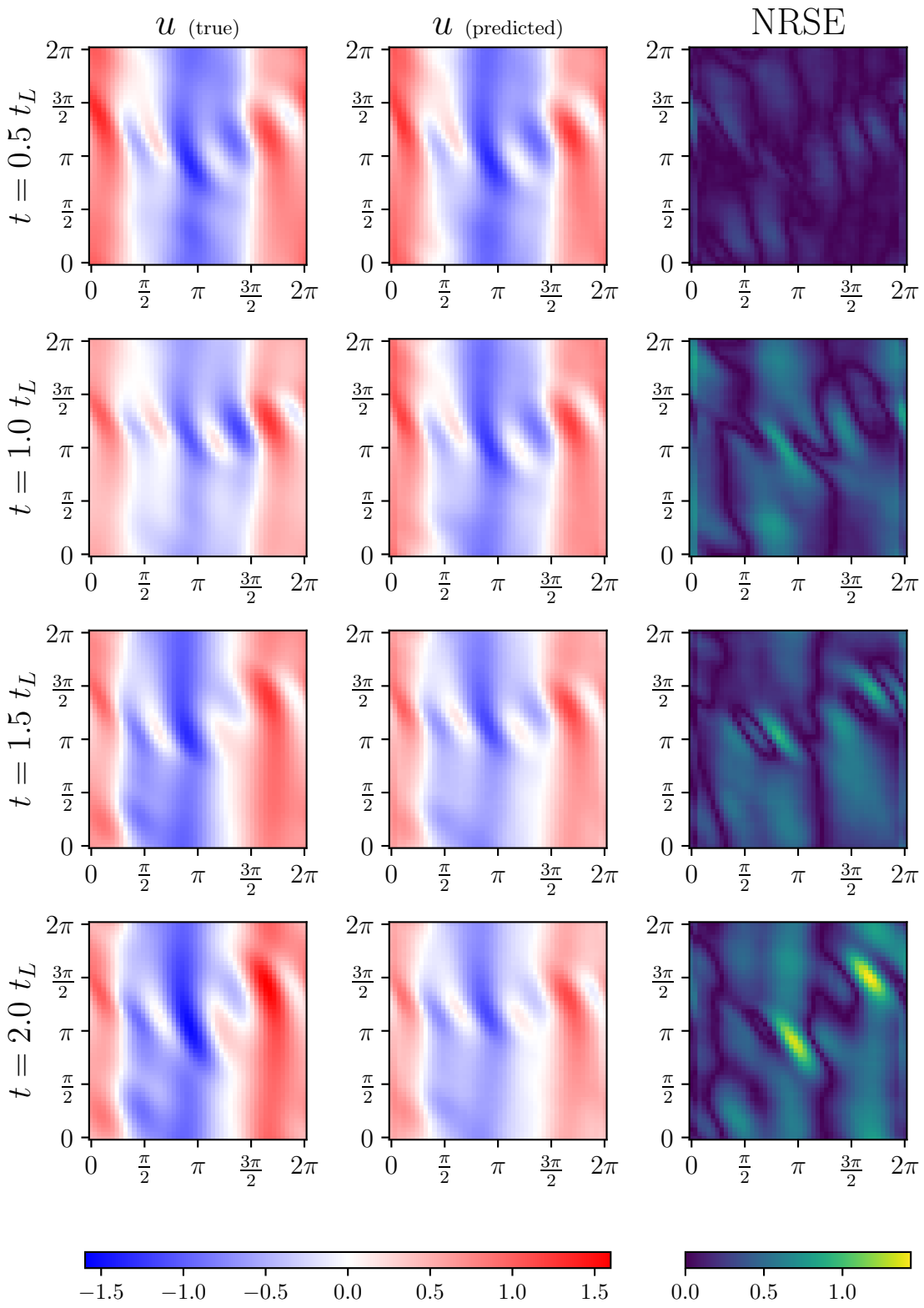


Figure C.15: Evolution of u at different time-steps of a sample trajectory, for the AE-GRU model.

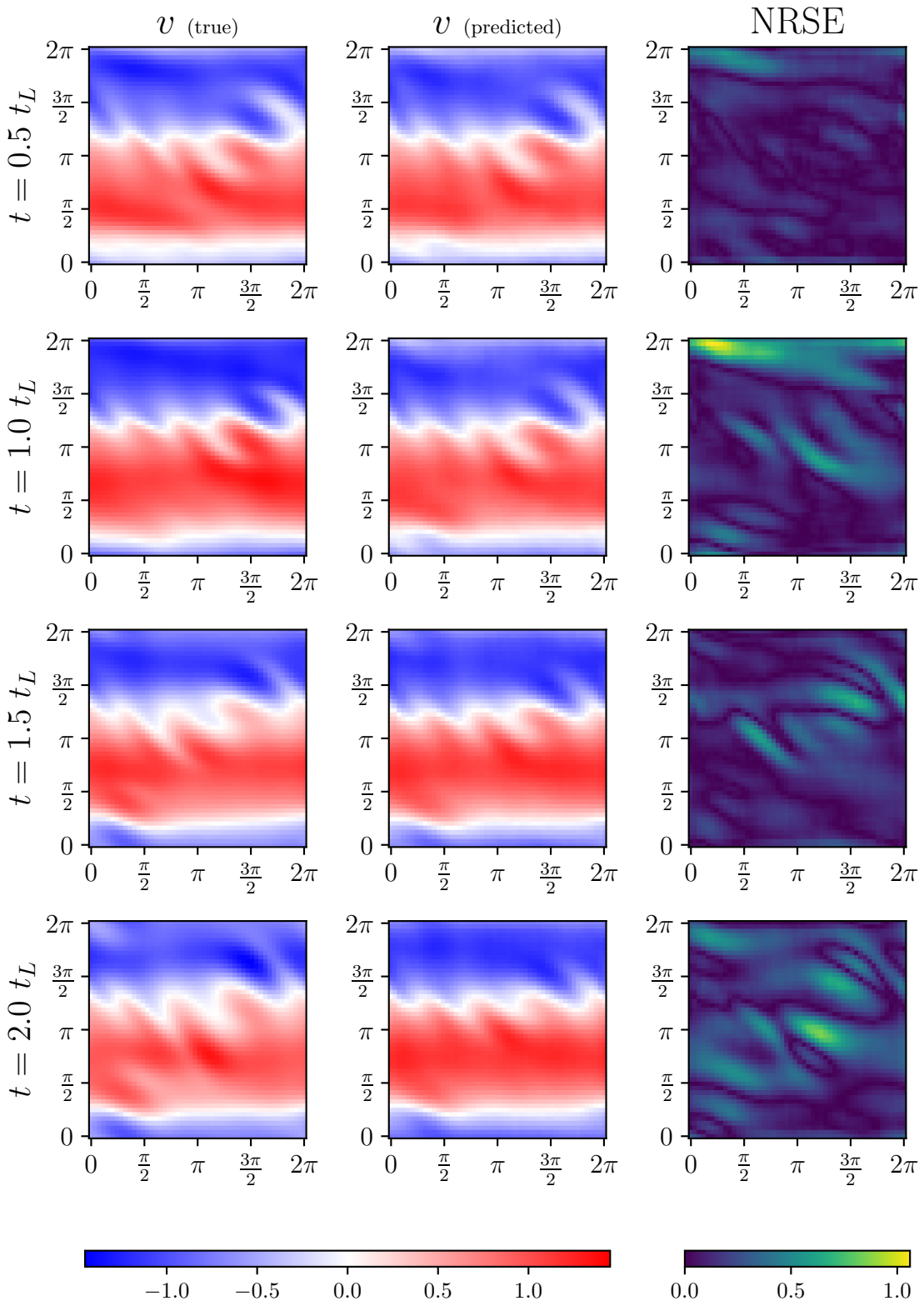


Figure C.16: Evolution of v at different time-steps of a sample trajectory, for the AE-GRU model.

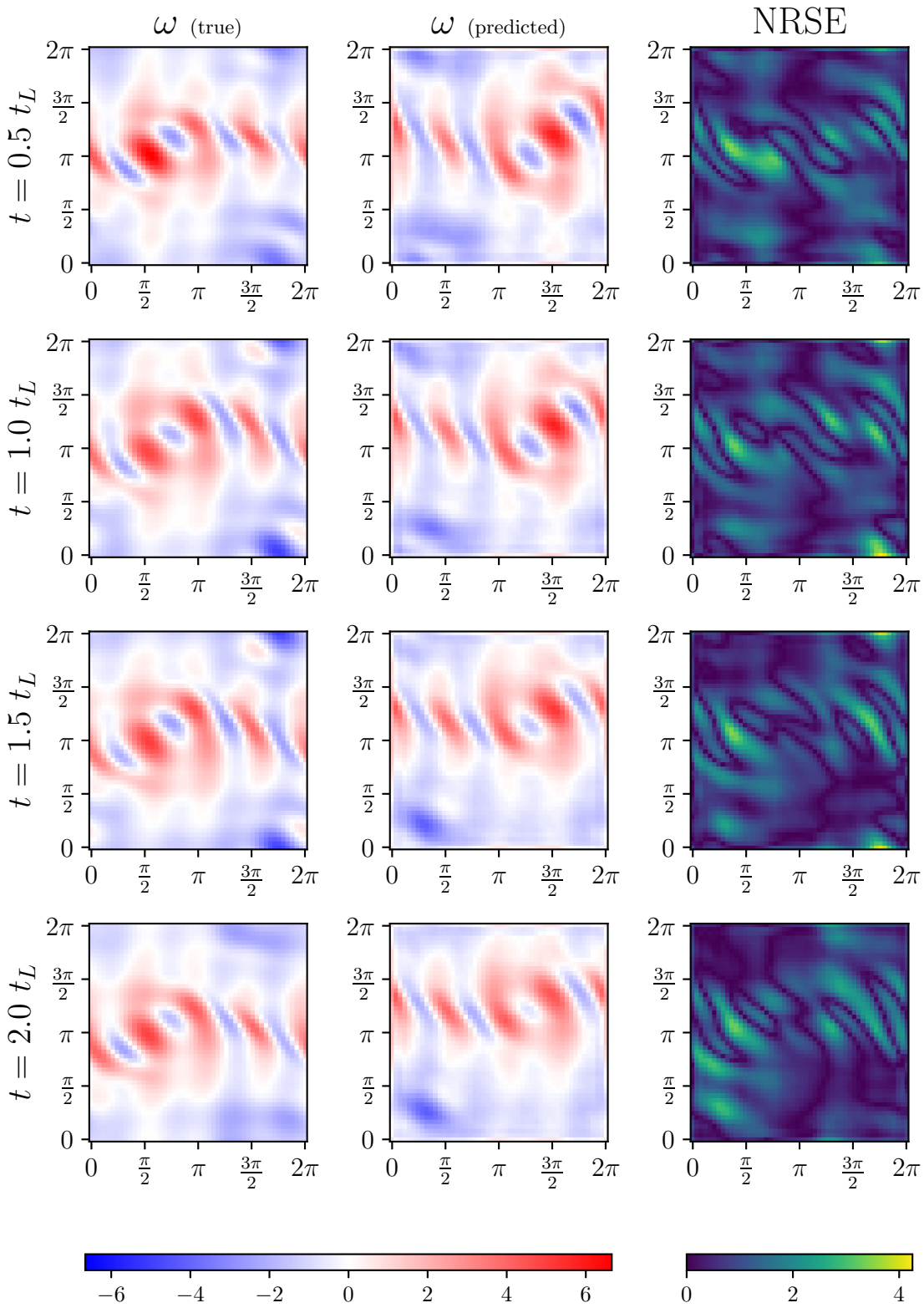


Figure C.17: Evolution of ω at different time-steps of a sample trajectory, for the AE-GRU model.

C.4.5. Typical Evolution of u, v and ω for a Sample Trajectory - AE-LSTM Model

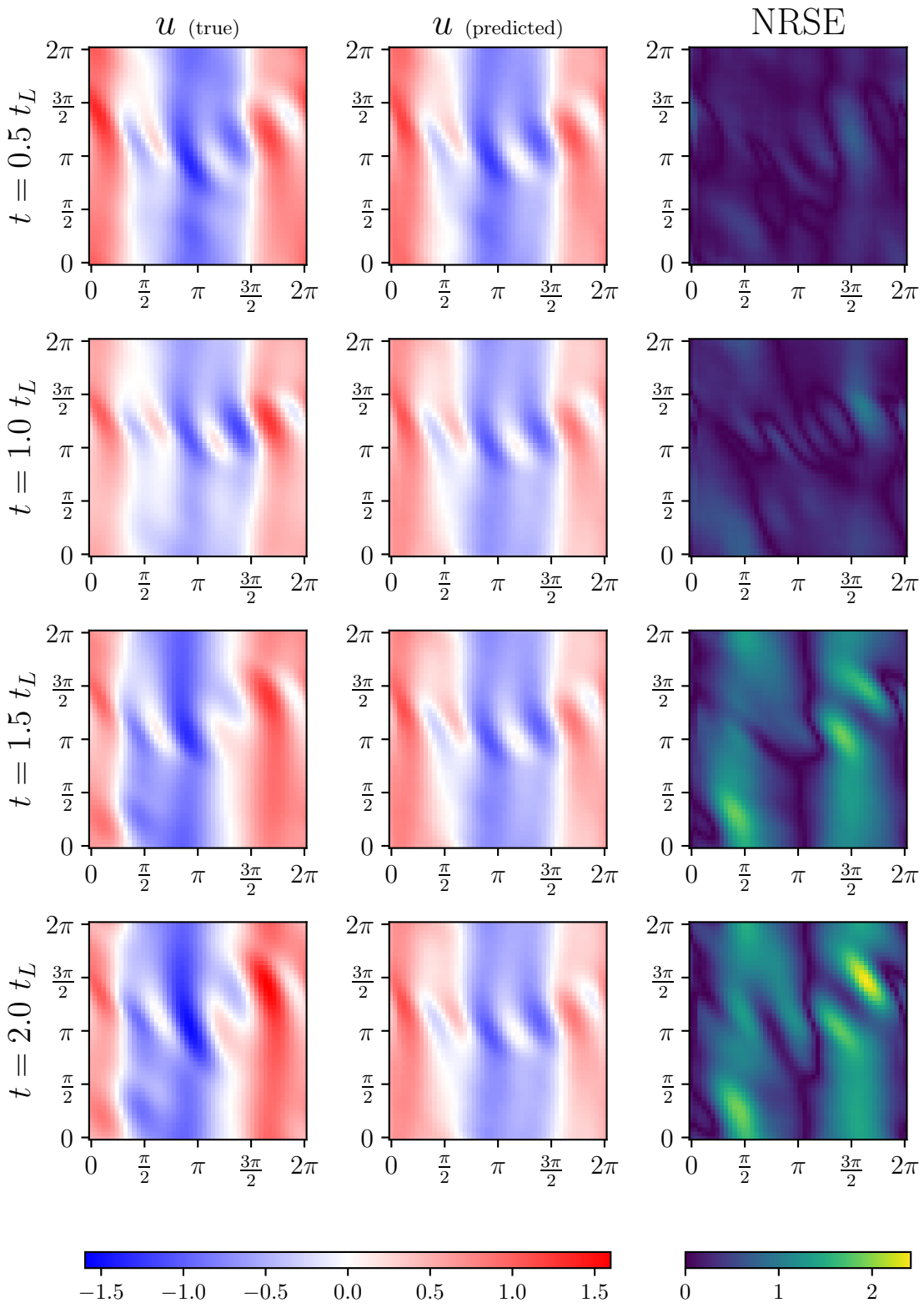


Figure C.18: Evolution of u at different time-steps of a sample trajectory, for the AE-LSTM model.

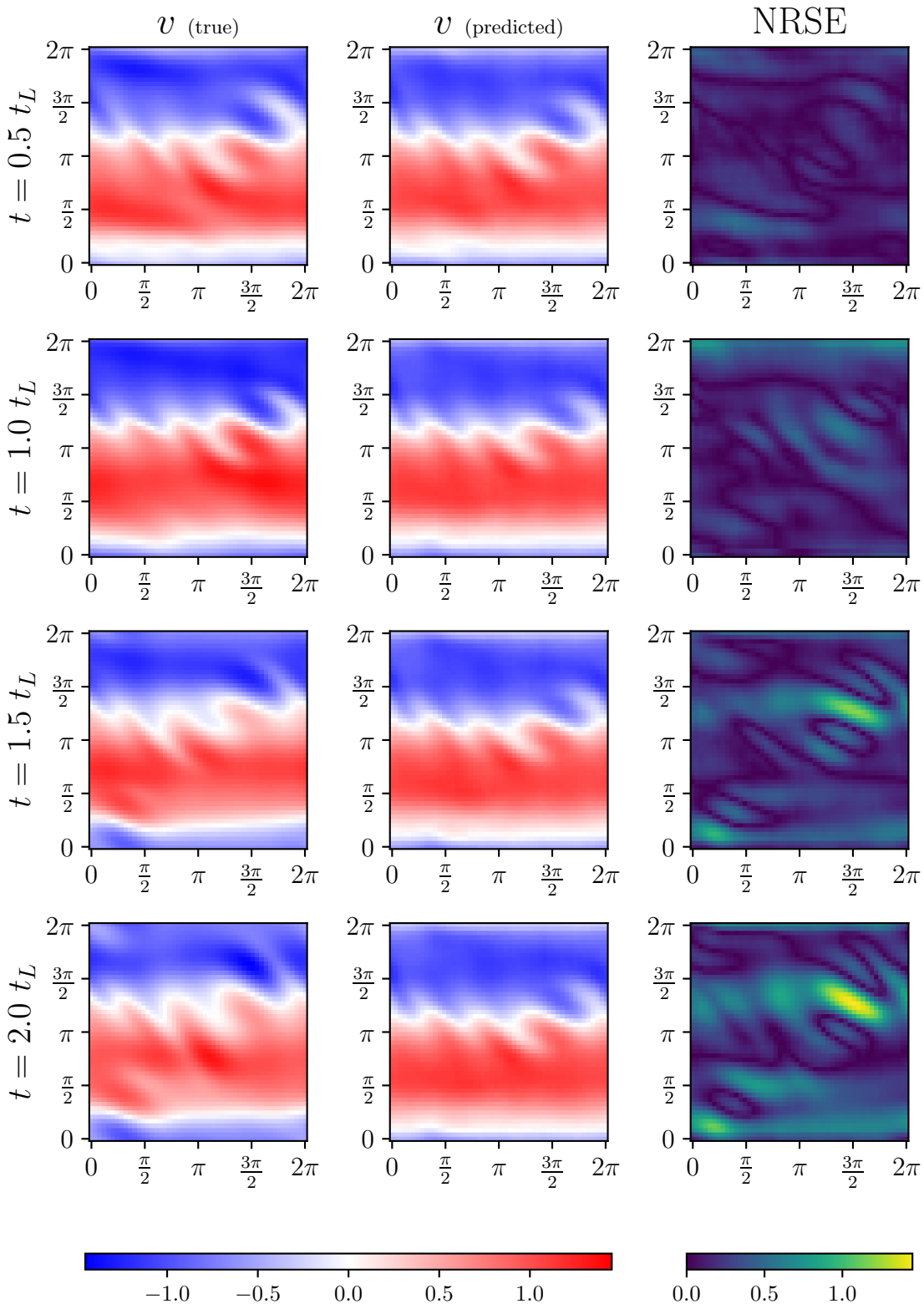


Figure C.19: Evolution of v at different time-steps of a sample trajectory, for the AE-LSTM model.

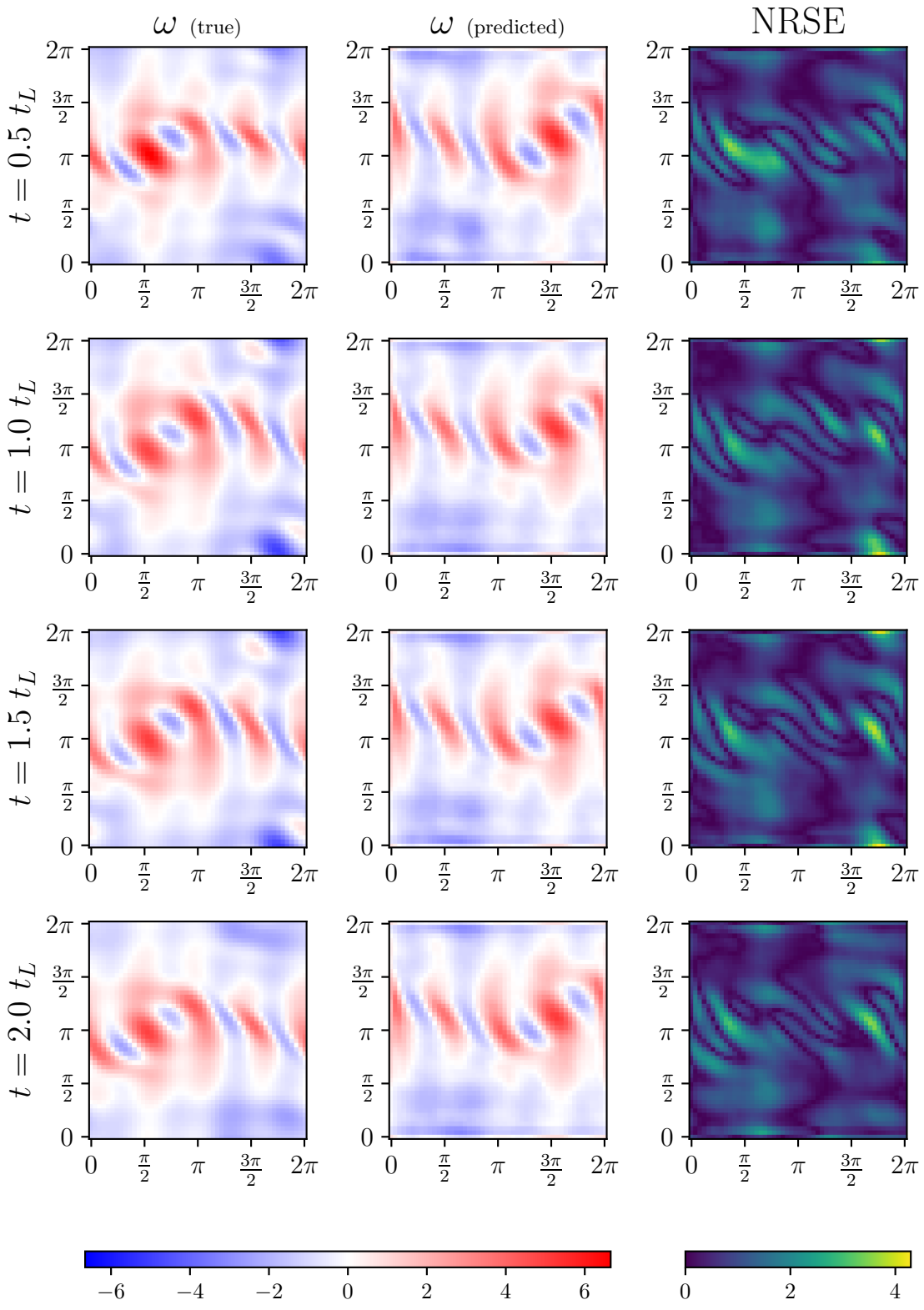


Figure C.20: Evolution of ω at different time-steps of a sample trajectory, for the AE-LSTM model.

C.4.6. Typical Evolution of u, v and ω for a Sample Trajectory - AE-ESN Model

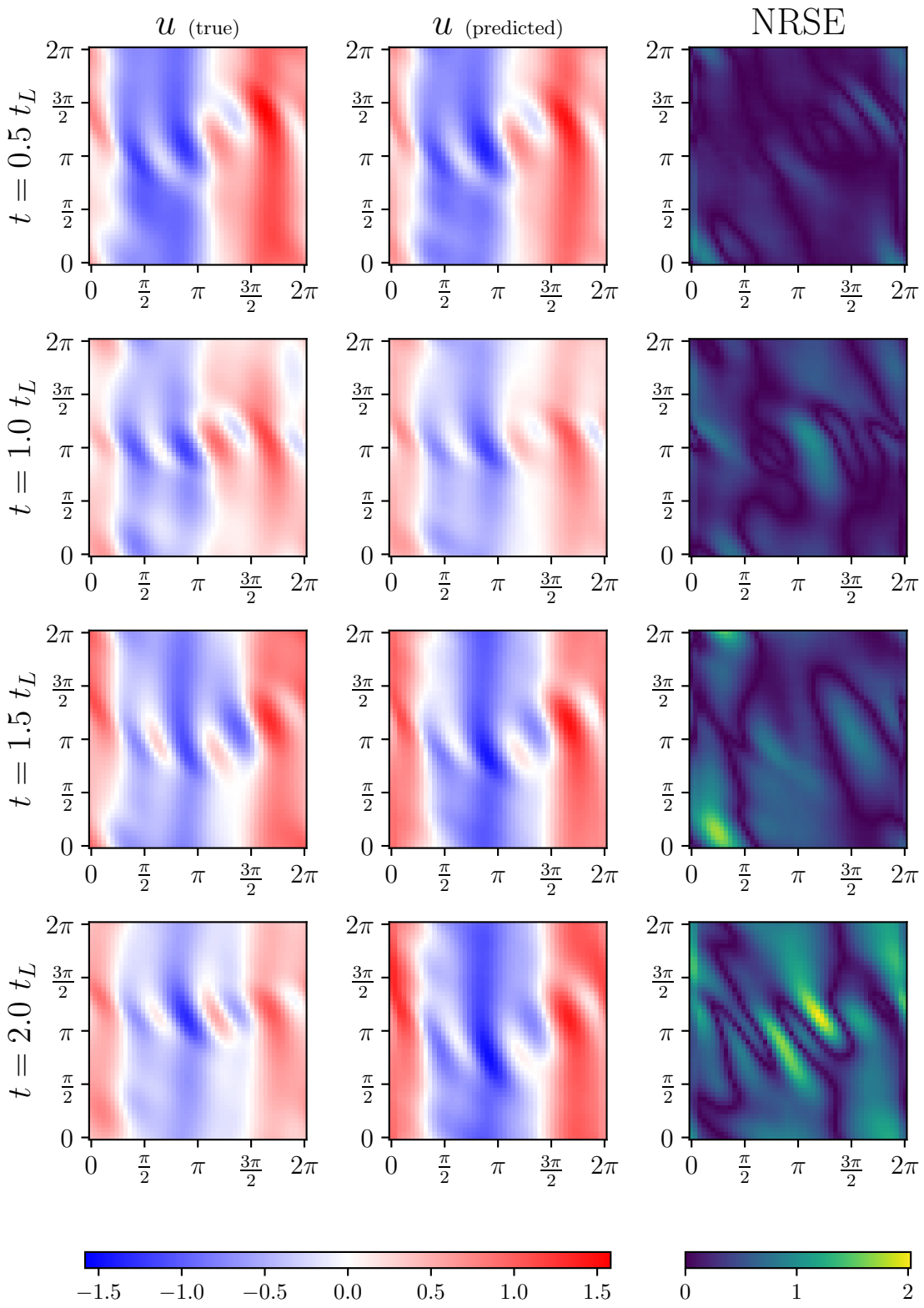


Figure C.21: Evolution of u at different time-steps of a sample trajectory, for the AE-ESN model.

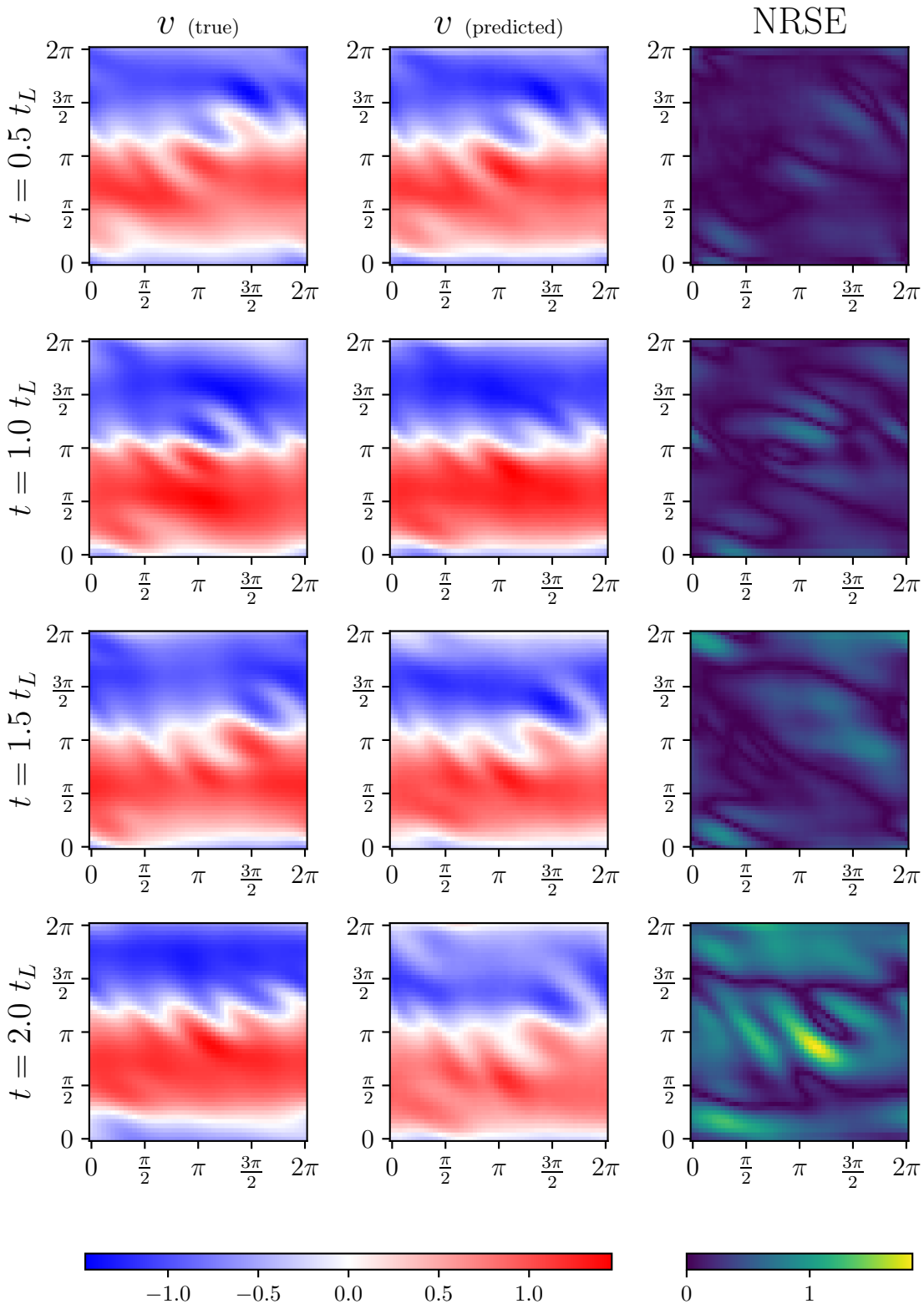


Figure C.22: Evolution of v at different time-steps of a sample trajectory, for the AE-ESN model.

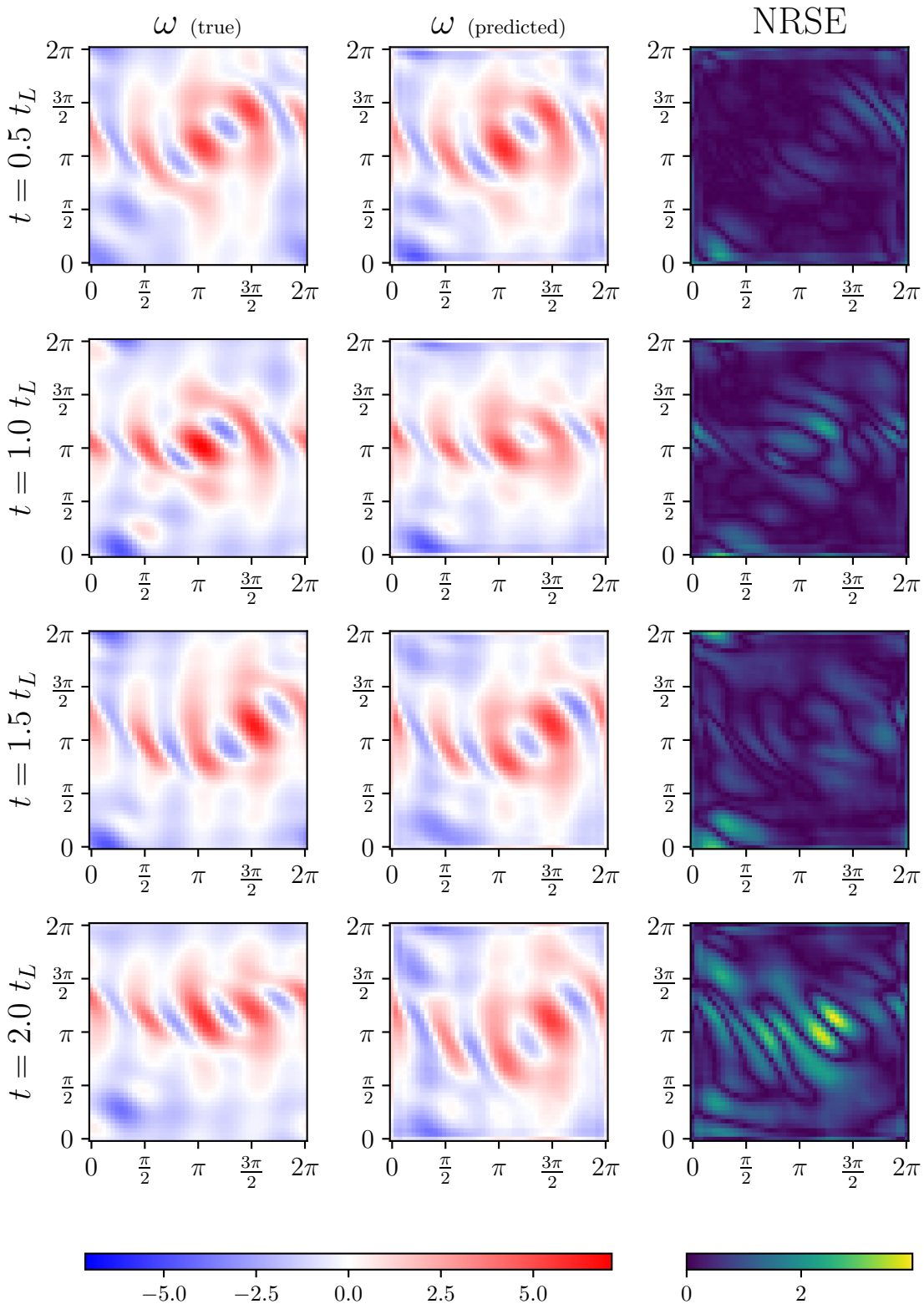


Figure C.23: Evolution of ω at different time-steps of a sample trajectory, for the AE-ESN model.

C.4.7. Typical Evolution of u, v and ω for a Sample Trajectory - AE-SimpleRNN Model

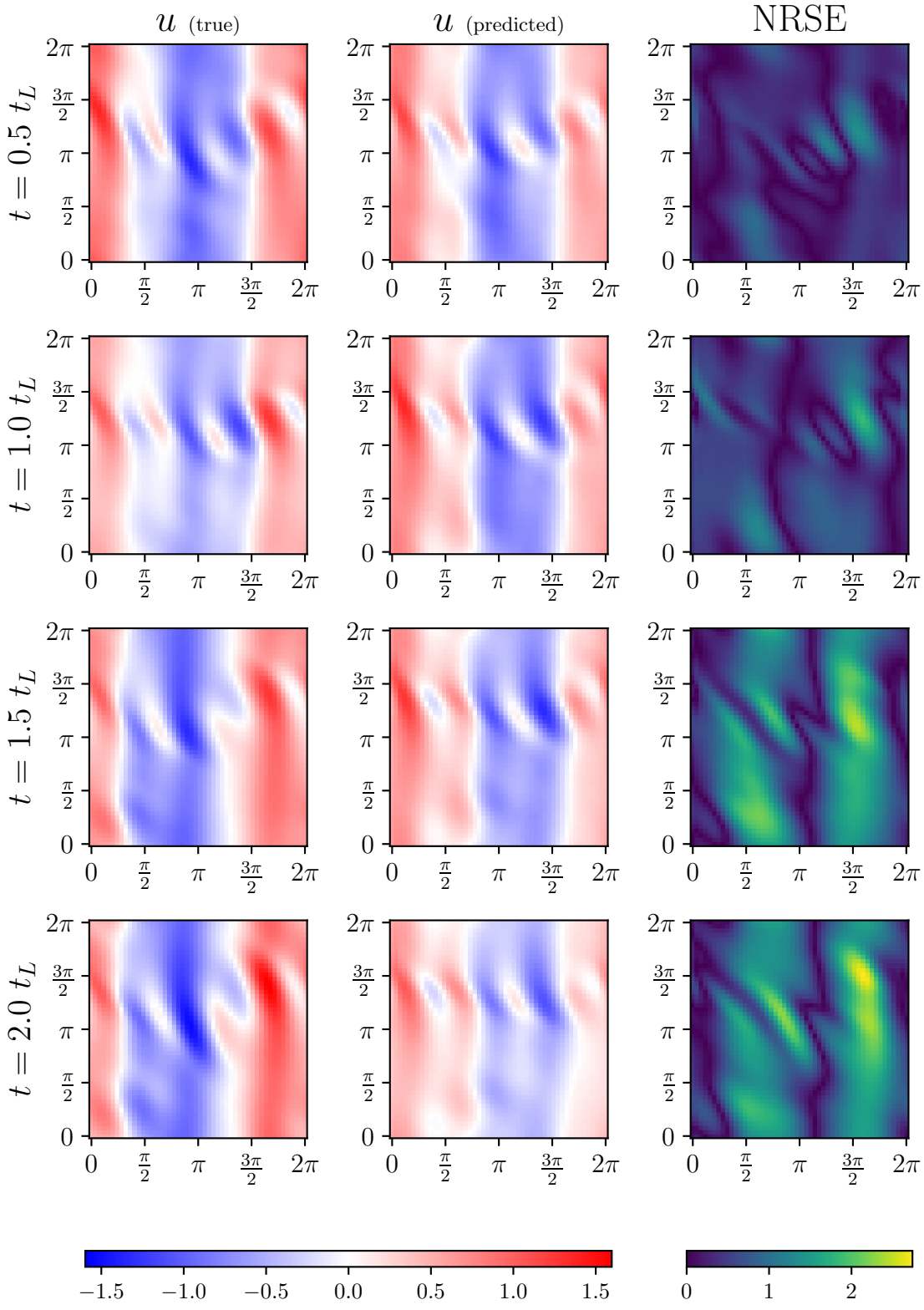


Figure C.24: Evolution of u at different time-steps of a sample trajectory, for the AE-SimpleRNN model.

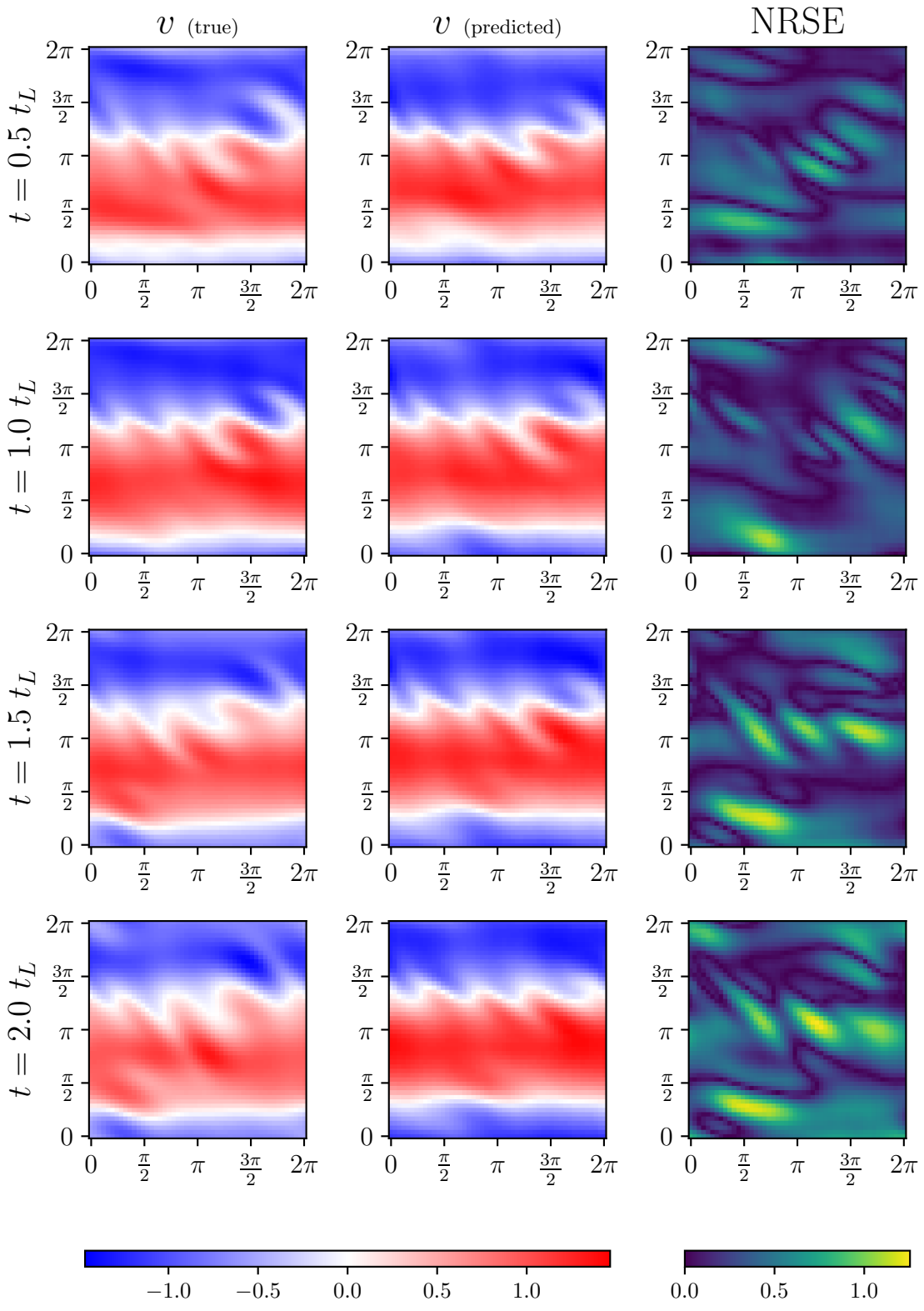


Figure C.25: Evolution of v at different time-steps of a sample trajectory, for the AE-SimpleRNN model.

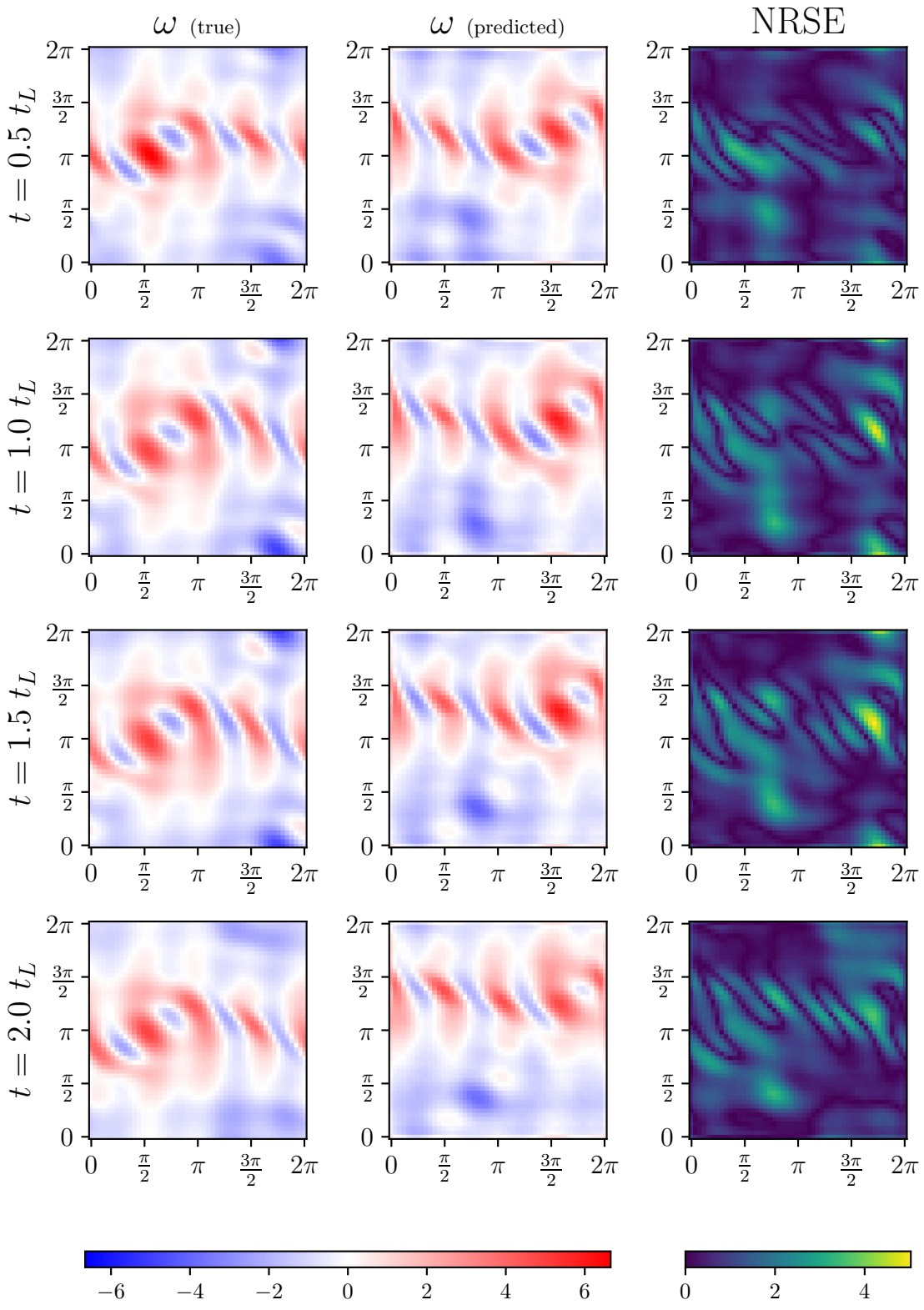


Figure C.26: Evolution of ω at different time-steps of a sample trajectory, for the AE-SimpleRNN model.

C.4.8. Typical Evolution of u, v and ω for a Sample Trajectory - POD-Galerkin Method

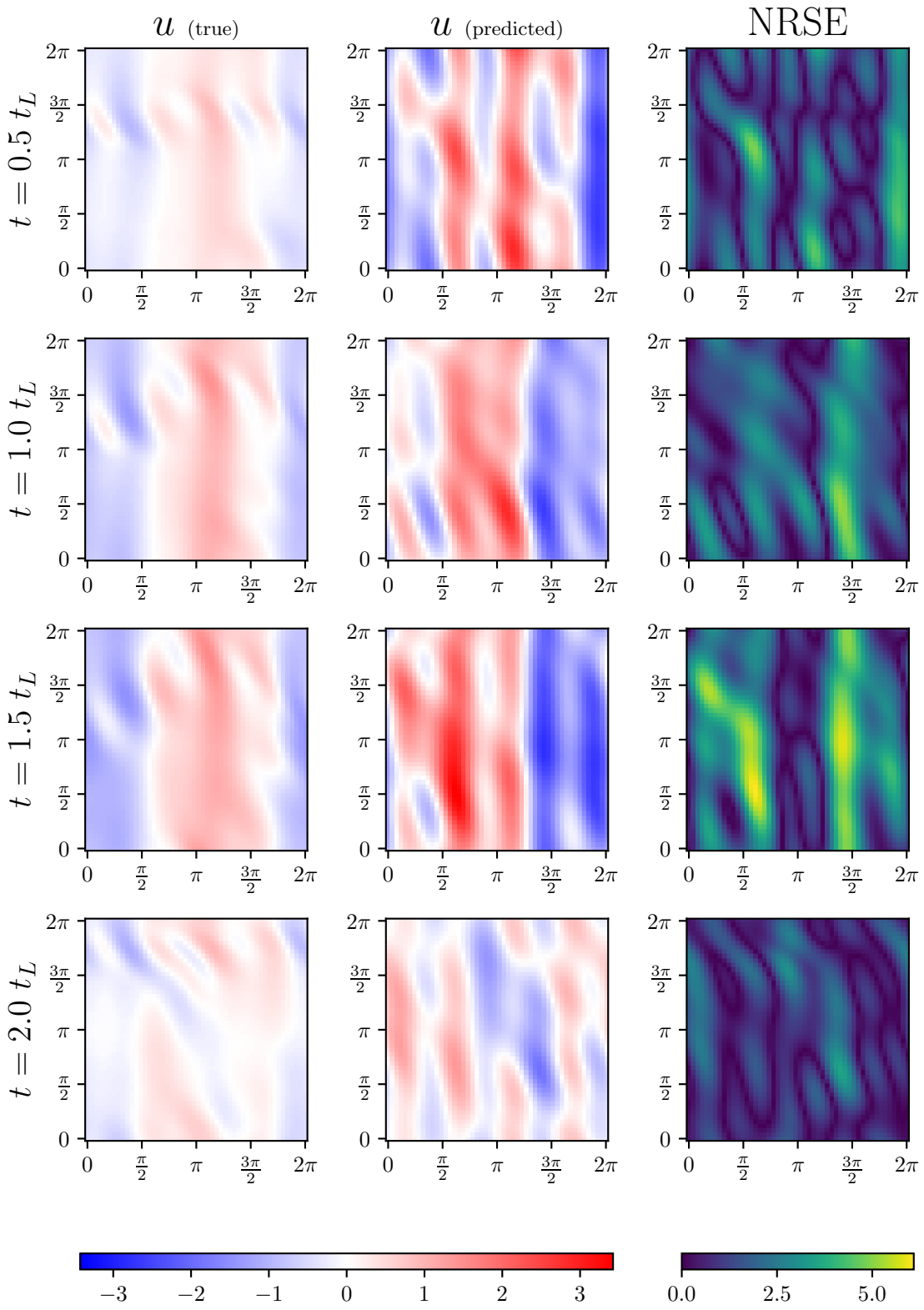


Figure C.27: Evolution of u at different time-steps of a sample trajectory, for the POD-Galerkin method.

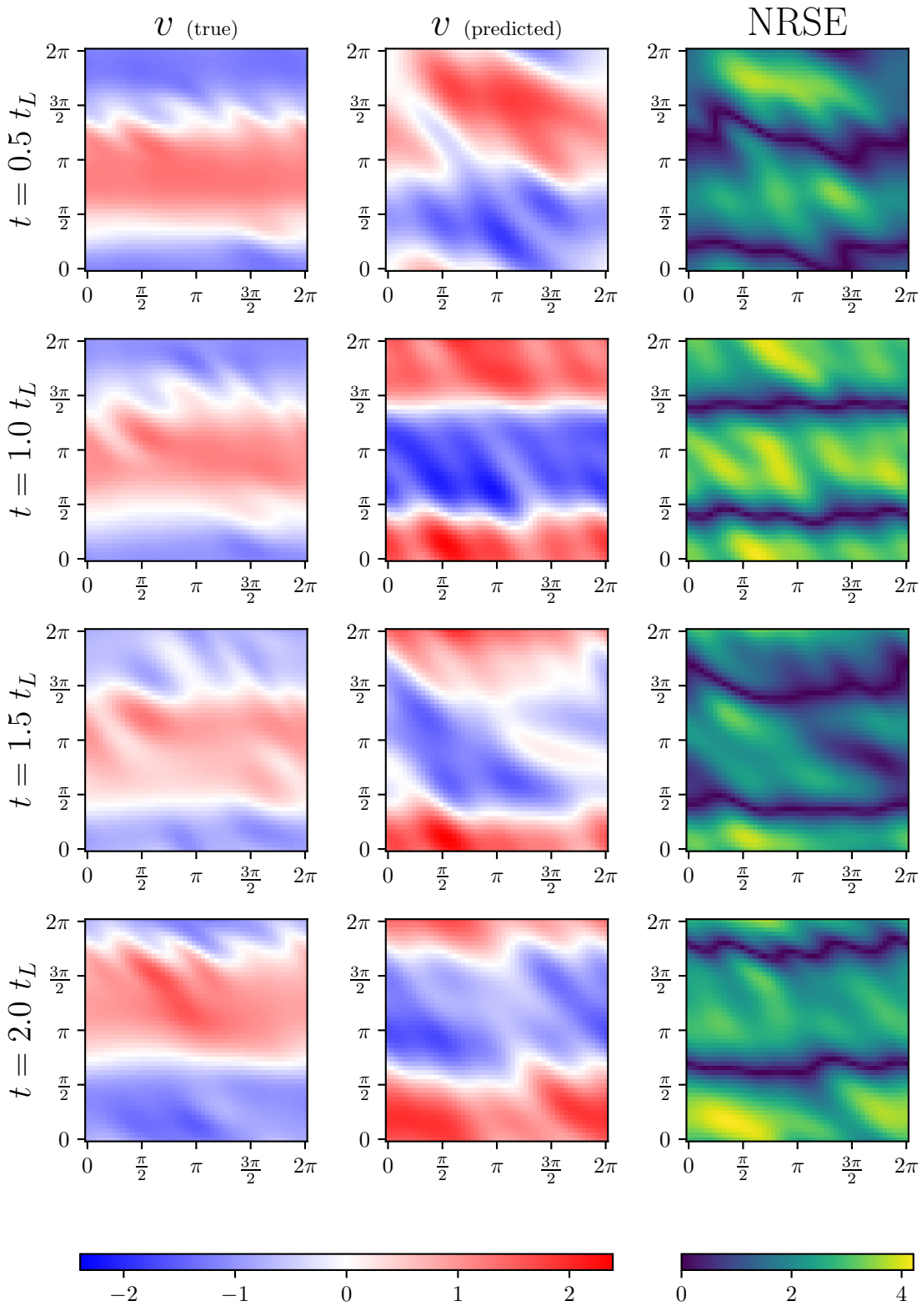


Figure C.28: Evolution of v at different time-steps of a sample trajectory, for the POD-Galerkin method.

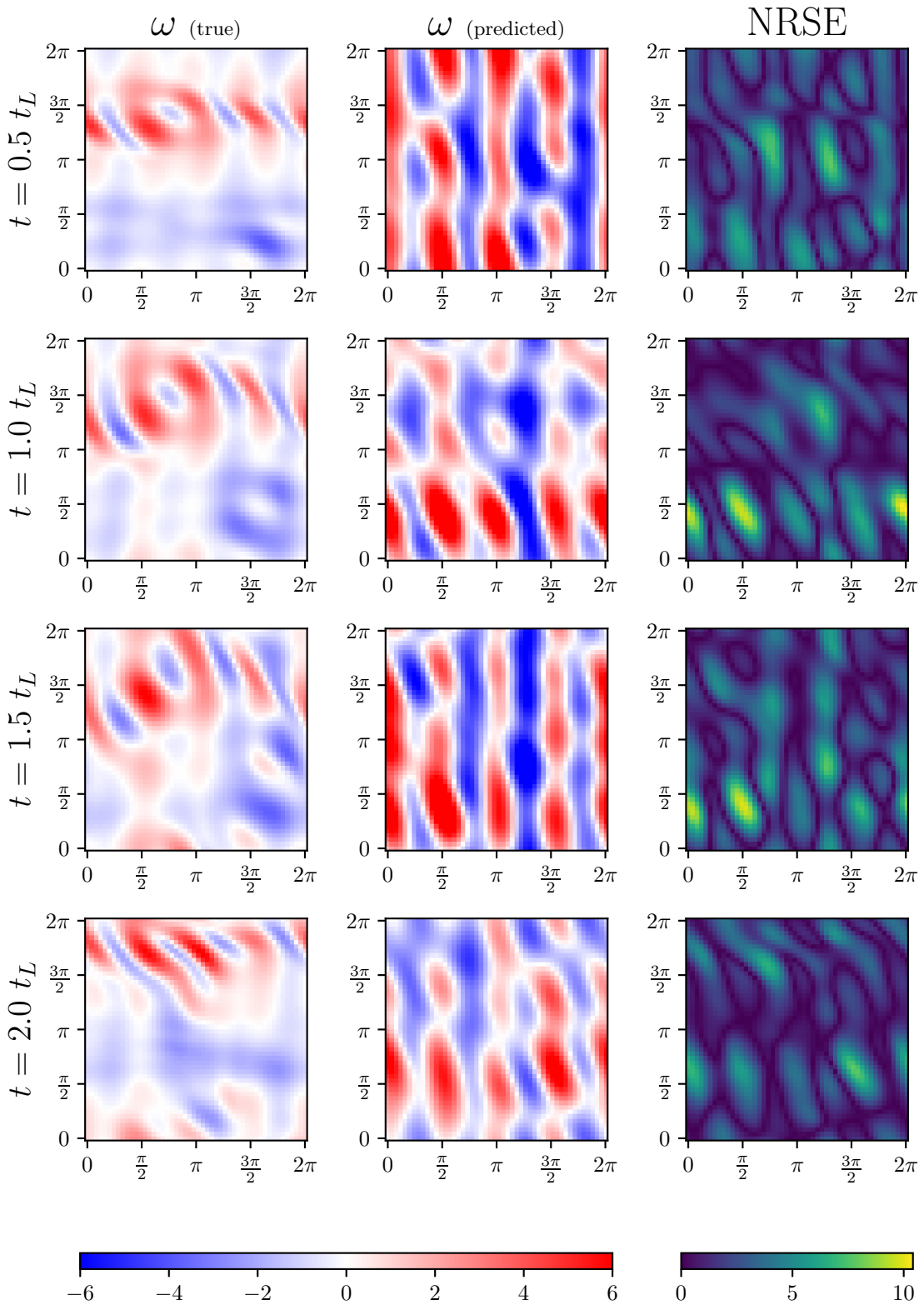


Figure C.29: Evolution of ω at different time-steps of a sample trajectory, for the POD-Galerkin method.

C.5. Fraction of Non-trainable parameters in an ESN

The input matrix lies in the space $\mathbb{R}^{n^r n^{ls} \times n^{ls}}$ and has a bias vector belonging to the space $\mathbb{R}^{n^r n^{ls}}$. The reservoir matrix lies in the space $\mathbb{R}^{n^r n^{ls} \times n^r n^{ls}}$. The output matrix lies in the space $\mathbb{R}^{n^{ls} \times n^r n^{ls}}$ with a bias vector belonging to $\mathbb{R}^{n^{ls}}$. Since the input and reservoir matrices are fixed, the fraction of parameters that are non-trainable is:

$$
\begin{aligned}
f(n^r, n^{ls}) &= \frac{n^r (n^{ls})^2 + n^r n^{ls} + (n^r n^{ls})^2}{n^r (n^{ls})^2 + n^r n^{ls} + (n^r n^{ls})^2 + n^r (n^{ls})^2 + n^{ls}} \\
&= 1 - \underbrace{\left(\frac{n^{ls}}{2(n^{ls})^2 + \left(n^r n^{ls} + 1/n^r \right) + 1} \right)}_{A} - \underbrace{\left(\frac{1}{n^r \left(2n^{ls} + n^r n^{ls} + 1 \right) + 1} \right)}_{B}
\end{aligned}
$$

Clearly, both A and B decrease in value as n^r increases, thus implying $-(A + B)$ becomes less negative as n^r increases. This implies that $f(n^r, n^{ls})$, the fraction of non-trainable parameters w.r.t. the total number of ESN parameters, grows as n^r grows.