Deep Learning Prediction and Uncertainty Quantification of High-Dimensional Time-Series Data

By

Justin LO Tian Wen

Supervisor:
Associate Professor Alexandre Hoang THIERY

DSA4199 Honours Project in Data Science & Analytics

National University of Singapore

2021/2022

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Justin Lo

1 April 2022

Acknowledgments

I would like to thank my supervisor, Associate Professor Alexandre Thiery for his unwavering support over the past year. A/ Prof Alex is always available for discussions and I have learnt a lot through my interactions with him. This thesis would not have been possible without his guidance and encouragement.

A big thank you to my family and friends who listened to me talk about my FYP and helped me bounce ideas when I was not too sure how to proceed.

Lastly, thank you to Lin Qian for creating this publicly available thesis template which helped me to easily organise and format my thesis.

Contents

Abstract

Deep Learning Prediction and Uncertainty Quantification of High-Dimensional Time-Series Data

by

Justin Lo

Bachelor of Science in Data Science & Analytics

National University of Singapore

This thesis explores the use of modern deep learning techniques to perform long-term predictions in time-series data. We use the Lorenz-96 model and the system generated by the Kuramoto–Sivashinsky equation to form baselines for analysis as these dynamical systems display chaotic behaviour and have real-world applications in time-series analysis. The thesis compares the predictive performance of Recurrent Neural Networks (RNN), Long Short-Term Memory Networks (LSTM), Reservoir Computing (RC) and Koopman Autoencoders, and discusses the impact of differing size and dimensionality of the datasets. Thereafter, the thesis explores the use of easily implementable uncertainty quantification methods which help to provide meaningful bounds for the predictions. We evaluate the methods using a common metric and discuss ways to further improve the techniques explored.

List of Figures

List of Tables

Chapter 1

Introduction

Recent developments in data collection and computational power have allowed people to develop a wider variety of models to perform predictions. There has been an increase in interest in performing time-series predictions as they have many applications such as in financial predictions [8, 41], transport [48] and within Internet of Things (IoT) infrastructures [4].

With the increased availability of data, the dimensionality of datasets used for training has also greatly increased. Traditional time-series forecasting methods such as autoregressive (AR), autoregressive integrated moving average (ARIMA) or exponential smoothing models are not able to handle the large volume and dimension of data [37]. Hence, this thesis seeks to explore modern developments in time series predictions and compare the effectiveness of some methods. These methods focus on deep learning as it has been shown that among the paradigms that are used for big data, deep learning performs very well as the scale of the data increases [42]. The thesis analyses recurrent neural networks (RNN), long short-term memory networks (LSTM), reservoir computing (RC) and Koopman Autoencoders and creates a benchmark to compare these methods side-by-side.

However, neural networks are poor at quantifying uncertainty within predictions and can produce overconfident estimates [27]. This could result in uninformed decisions being made which could be harmful for the intended users. Leslie [29] discusses the need for outcome transparency where results should be interpretable and properly inform the judgements of those who use the model. Confidence intervals are becoming crucial requirements within many sectors such as in physics, the financial sector and the energy sector, and point estimates are insufficient to

provide comprehensive information for the end users [14, 32].

There are two main classifications of uncertainty that need to be tackled. Aleatoric uncertainty arises from the uncertainty due to randomness of the data distribution. For example, aleatoric uncertainty could arise from flipping a coin, where even the best model will not be able to provide an assured answer and hence result in variability in predictions. This form of uncertainty is considered an irreducible portion of uncertainty. The other form of uncertainty is epistemic uncertainty which refers to the uncertainty that arises due to the inadequate knowledge or data. For example, the presence of noisy data or the lack of training data could result in the model not being able to learn an accurate representation, even when one could exist. Therefore, epistemic uncertainty is also known as systematic uncertainty and is the reducible part of total uncertainty [1, 19]. This thesis seeks to code and evaluate uncertainty estimation methods which quantify the total uncertainty that arises from both aleatoric and epistemic uncertainty, thereby providing accurate prediction intervals so that users will be able to generate interpretable predictions.

1.1 Dynamical Systems

As a benchmark for analysis, this thesis investigates the prediction and uncertainty estimation of the Lorenz-96 and Kuramoto-Sivashinsky (KS) equations. These two equations model high-dimensional dynamical systems and are known for exhibiting chaotic behaviour.

A dynamical system describes the evolution of a finite-dimensional state, x, across time t [20]. In the Lorenz-96 and KS equations, the system is a continuous-time process, with the time intervals t being non-discrete. In such continuous-time dynamical systems, the dynamics are described using a differential equation in the form of $\frac{dx}{dt} = f(x, t)$, where $f(x, t)$ is typically a non-linear function [28].

The Lorenz-96 system is a dynamical system created by Edward Lorenz in 1996 [30]. The Lorenz-96 model was originally created as an atmospheric model and is commonly used to study the predictability in forecasting weather. The model has K variables, $X_1, X_2, \ldots, X_K$, and is defined by a system of K coupled ordinary

differential equation. Each of the K equations are defined as follows

$$\frac{dX_k}{dt} = -X_{k-2}X_{k-1} + X_{k-1}X_{k+1} - X_k + F \tag{1.1}$$

$$x_{k-K} = x_{k+K} = x_k \tag{1.2}$$

In Equation 1.1, F is independent of K and represents the forcing constant, the linear term represents a damping term, and the quadratic terms simulate advection and conserve the total energy of the system. Equation 1.2 allows the system to be represented as a circular array, and can be thought of as values of atmospheric quantities in K sectors in a latitude circle [23, 24, 30].

The KS equation is a nonlinear fourth-order partial differential equation developed independently by Kuramoto [25], where it was used in reaction-diffusion systems within the field of nonlinear chemical kinetics, and by Sivashisky [39], where it was used to describe the instability in flame fronts. In the one-dimensional version, the partial differential equation is defined as

$$u_t + u_{xx} + u_{xxxx} + uu_x = 0 \tag{1.3}$$

$$u(x + L, t) = u(x, t) \tag{1.4}$$

The equation is defined in the domain $[0, L)$, which defines the spatial domain. As the value of L increases, the system displays increasing levels of spatio-temporal chaotic turbulence [12].

Both dynamical systems can be used with applications within time-series forecasting. Over the past few years, there has been an increase in research done for data-driven methods within the field of dynamical systems due to the fast and accurate predictions it provides. In addition, such methods are useful when only observational data is available [11]. This is relevant to time-series as we have discussed earlier that many dynamical systems can be used to model real-life time-series applications, while the data-driven approach is applicable to how we currently collect and analyse time-series data.

1.2 Chaos and Lyapunov Exponents

Another reason to analyse the Lorenz-96 and KS equation dynamical systems is their ability to exhibit chaotic behaviour [23, 35, 44]. Chaotic behaviour, also

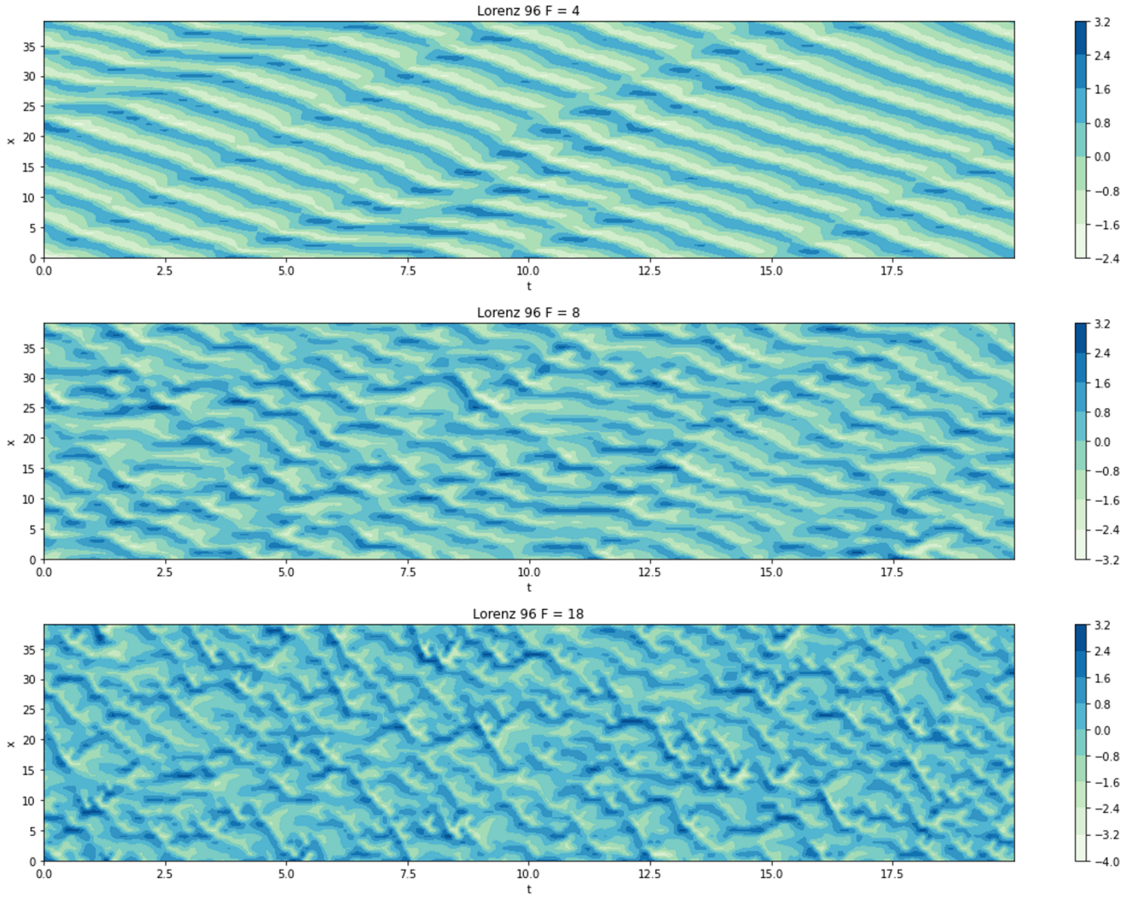


Figure 1.1: Comparison of trajectories generated by different forcing constants in the Lorenz-96 system. (Top) $F = 4$ (Middle) $F = 8$ (Bottom) $F = 18$

commonly described as the butterfly effect, is defined by the dependence of a system to its initial conditions. In a chaotic system, a small change in the initial conditions can result in exponential differences in neighbouring trajectories [23]. Chaos is a realistic occurrence that is witnessed in our daily lives. Besides helping us understand various weather behaviours, chaos theory is also widely studied in and applied to applications such as within computer security [28].

In the Lorenz-96 system, the forcing constant, F, determines the level of chaos. $F = 8$ and above have been shown to display chaotic behaviour within the system [24]. Figure 1.1 shows a comparison of the trajectories generated from different values of the forcing constant. As seen, the larger the value of F, the trajectory becomes increasingly irregular and unpredictable.

For the system generated by the KS equation, the value of L in the KS equation

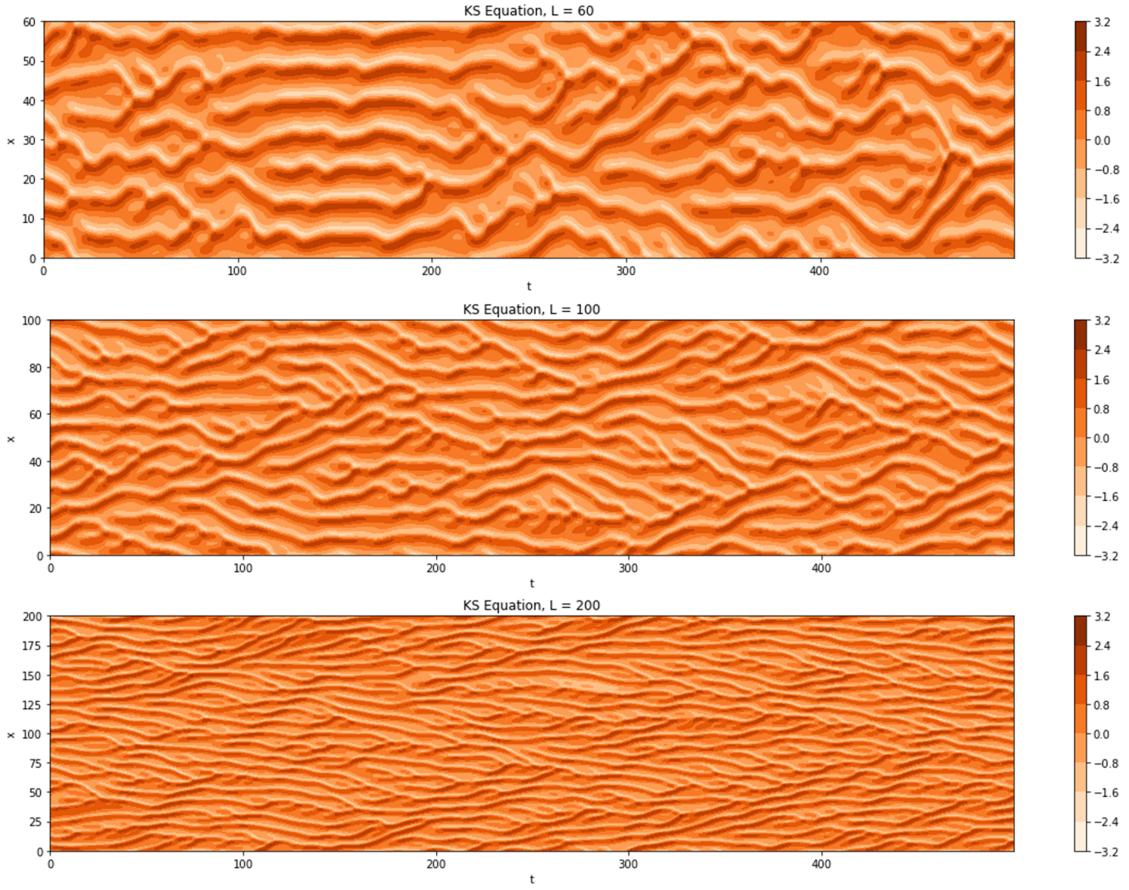


Figure 1.2: Comparison of trajectories generated by the KS Equation with different spatial dimensions. (Top) $L = 60$ (Middle) $L = 100$ (Bottom) $L = 200$

controls the chaotic behaviour within the system [44]. As the value of L increases, the dimensionality of the chaotic attractor also increases, increasing the difficulty of predicting the system. This can be seen in Figure 1.2, where trajectories resulting from different values of L are shown.

To quantify the level of chaos in a system, a metric called the Lyapunov exponent is used. The Lyapunov exponent measures the rate which neighbouring trajectories diverge under a chaotic system. A spectrum of Lyapunov exponents is calculated, $\Lambda_1, \ldots, \Lambda_N$, where N is the dimensionality of the state space. The largest of these values, known as the Maximal Lyapunov Exponent (MLE) $\Lambda_{\max}$, is of importance and is used to calculate the duration of a Lyapunov time. A Lyapunov time is calculated as the inverse of the MLE, i.e. $\Lambda_{\max}^{-1}$ and represents the amount of time for a trajectory to diverge by a factor of e.

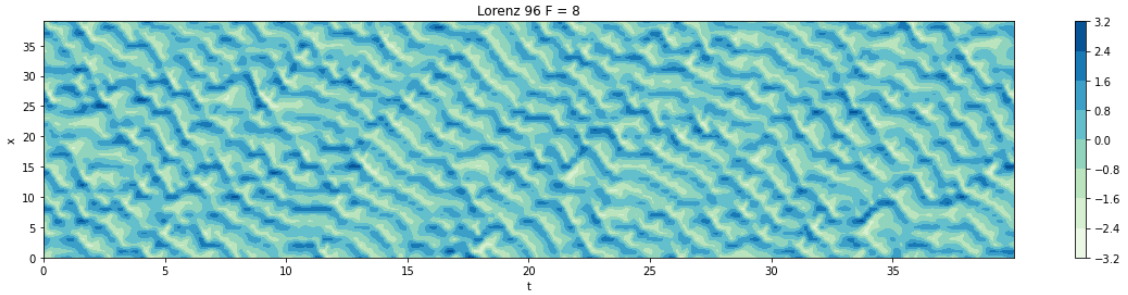


Figure 1.3: Portion of trajectory generated from the Lorenz-96 model with $F = 8$

1.3 Generating Data

The Lorenz-96 system is generated with $F = 8$ using the Runge-Kutta method of order 5 using Python's scipy, with a random starting point and an output dimension $d = 40$. (Figure 1.3). The data is generated till $t = 2\,000$ in intervals of $\delta t = 0.01$, resulting in a total of 2×10^5 points. The first 20 000 are discarded to ensure that transient behaviour has decayed. The data is normalized and then split equally into a train and test set. Among the points within the test set, 100 starting points are randomly selected and a horizon of 400 timesteps (4 seconds) are created for each of the starting points; these 100 trajectories are set as the test trajectories for the Lorenz-96 model. The Lyapunov exponent was calculated using a code adapted from Vlachas et al. [44]. 10 points from the dataset were randomly selected and a noise of variance 1e-8 was added to these points to create slight perturbations. The two points were then stepped forward and the log absolute norm was calculated at each time-step. This created 10 different slopes of the error where the mean of the slopes estimated the Maximal Lyapunov Exponent. For the Lorenz-96 system with a forcing constant of 8, the Maximal Lyapunov Exponent was calculated to be 1.64, and hence the 1 Lyapunov time was calculated as $1/1.64 = 0.61$ seconds.

The KS system is created numerically using spectral methods from code provided in [16]. The spatial domain size was set at $L = 60$ and was created on a grid of 240 equally spaced spatial points. This meant that the output of the prediction models would resemble a 240-dimension dataset (Figure 1.4). The data is generated till $t = 50\,000$ in intervals of $\delta t = 0.25$, resulting in a total of 2×10^5 points. Similar to the Lorenz-96 system, 10% of the data was removed to allow the transient effects to

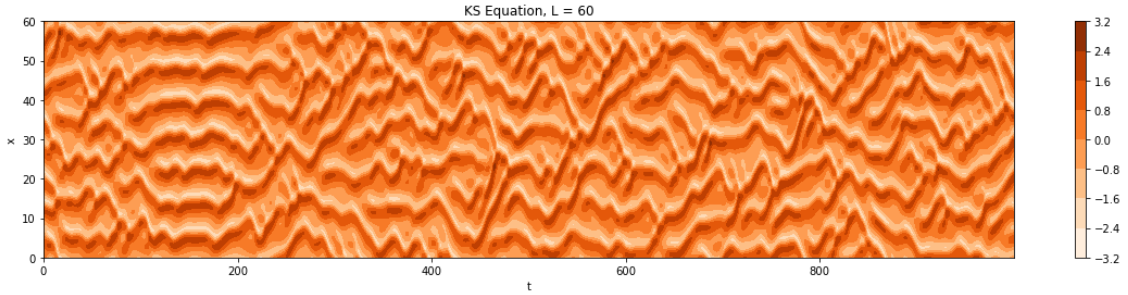


Figure 1.4: Portion of trajectory generated using the KS Equation with $L = 60$

be removed. The data was then normalized and split into the train and test sets in an identical manner to the Lorenz-96 system, resulting in 100 test trajectories spanning a horizon of 400 timesteps (100 seconds) each. The Maximal Lyapunov Exponent was taken from Edson et al.'s work on calculating Lyapunov Exponents of the KS Partial Differential Equation [12] and is determined to be 0.089, signifying that a Lyapunov time of $1/0.089 = 11.2$ seconds.

1.4 Thesis Outline

This thesis provides a baseline for comparing recent advancements in deep learning and their performance in chaotic dynamical systems. In Chapter 2, the concepts behind the various deep learning methods are explained. For each of the methods, the important hyperparameters which affect training are identified and the training methods used for the experiments are described. In Chapter 3, the models are tuned and tested against the Lorenz-96 dynamical system and the high-dimension KS partial differential equation. The results are tabulated to compare the efficacy and efficiency of the methods. The section also explores a limited data setting to understand each method's reliance on data to perform accurate predictions. To provide useful uncertainty estimates for the predictions, Chapter 4 introduces some simple uncertainty quantification methods and their possible adaptations. Chapter 5 shares the experiment results of these uncertainty quantification methods and further studies the impact of data noise in quantifying uncertainty. The thesis concludes in Chapter 6 where the main results are summarised, and future research directions are discussed based on the observations within the thesis.

Chapter 2

Prediction Methods

In this thesis, 4 different methods are discussed and evaluated on the datasets. These methods are a (i) Vanilla Recurrent Neural Network (RNN), (ii) Long Short-Term Memory Network (LSTM), (iii) Reservoir Computing (RC), and (iv) Koopman Autoencoders. This section discusses the concepts behind these methods, as well as describes the training process utilised during the experiments.

2.1 Recurrent Neural Networks (RNN)

In a regular feedforward network, the network only takes in one input, which is then processed and subsequently used to calculate an output as the prediction. However, in many applications, the input data is not independent of each other, and we want to capture the interdependence of the data. In the context of time-series, the sequential nature of the data makes feedforward networks insufficient to perform good predictions. RNNs are designed to learn dependencies by utilising the hidden state variables, which functions as the memory of the network.

2.1.1 Equations and Parameters of the RNN

The concept of RNNs were first introduced in the early 1980s as the Hopfield Network [18] and further explored using low-dimensional RNNs in 1990 by Elman [13]. The idea of the recurrent network is to have a hidden state that is brought forward through time. Hence, a hidden unit at time t, h_t, is calculated as the function of the previous hidden state h_{t-1} and the input at time t, x_t [31]. This can be represented

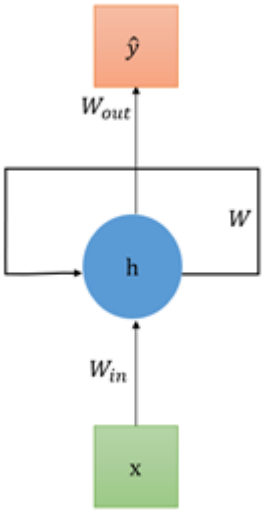


Figure 2.1: Structure of RNN Cell

as an equation:

$$h_t = f(h_{t-1}, x_t) \tag{2.1}$$

In the case of a vanilla RNN, the definition in Equation 2.1 is implemented directly, where

$$h_t = \sigma(W h_{t-1} + W_{in} x(t) + b_h), \quad t = 1, \ldots, T \tag{2.2}$$

In Equation 2.2, W is the weight matrix between hidden states, W_{in} is the weight matrix connecting the input to the hidden state, b is the bias of the hidden state, and σ represents an activation function which is usually selected to be the tanh function. In this way, the RNN creates a feedback loop where information from both the past and present are encoded into the hidden state.

The prediction at time t can be calculated as a function of the hidden state. This is represented as

$$\hat{y}_t = W_{out} h_t + b_{out} \tag{2.3}$$

where y_t represents the predicted output at time t, W_{out} is the weight matrix which connects the hidden state to the predicted output, and b_{out} is the bias of the output state. The overall RNN network can be represented visually as seen in Figure 2.1.

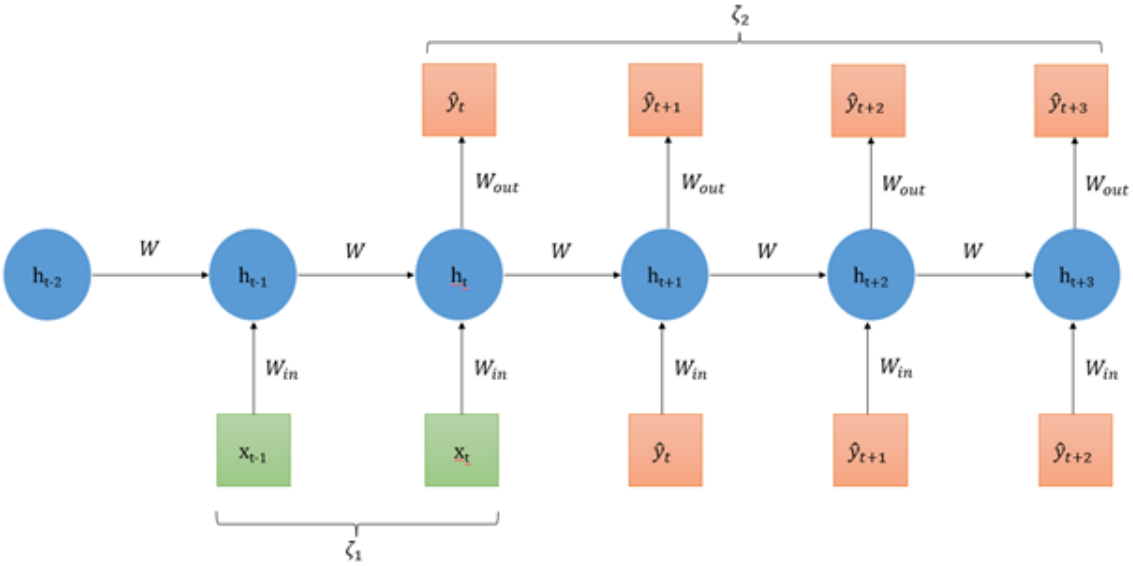


Figure 2.2: RNN Training Process

2.1.2 Training of the RNN

As with vanilla neural networks, training is conducted by updating the parameters via gradient descent. These gradients are calculated via backpropagation of the error backwards to determine $\frac{\partial E}{\partial W}$. In the context of RNNs which have a sequence of inputs and outputs, an extension called Backpropagation through Time (BPTT) [45] has been developed in 1990. BPTT 'unrolls' the RNN and allows the error to be backpropagated to the previous time steps that were used to determine the error calculated for the prediction of future time-steps.

In the backpropagation process, there are 2 important hyperparameters that affect the process. First, we need to determine the sequence length ζ_1, which denotes the number of time-steps used for backpropagation of the gradient. This parameter helps to encapsulate the temporal patterns of the data and a larger value denotes that a longer sequence length is fed into the network. Although a larger sequence length may seem good, it slows down training and may lead to vanishing gradients [44]. The second parameter ζ_2 denotes the number of time-steps forward in which the error is evaluated and accumulated. In this thesis, the mean squared error (MSE) is used to evaluate the error of the network.

In summary (refer to Figure 2.2 for a visual representation), a hidden state,

$h_{-\zeta_1}$ is initialised at 0. This hidden state is then combined with the ζ_1 inputs, and subsequent hidden states $h_{t-\zeta_1+1} \dots h_{t-1}$, to determine the value of h_t. The predicted value of y_t, $\hat{y}_t$, is then calculated based on h_t. For the next $k = 1 \dots \zeta_2 - 1$ time-steps, a predicted y_k value is used as the input x_{k+1}, thereby generating the next hidden state h_{k+1} and the next predicted value y_{k+1}. The ζ_2 predicted values, $\hat{y}_t \dots \hat{y}_{t+\zeta_2-1}$, are then compared against the true outputs, $y_t \dots y_{t+\zeta_2-1}$, and the error is computed and backpropagated through the $\zeta_1 + \zeta_2 - 1$ steps.

For the experiments conducted in this thesis, training for the RNN was done Adam optimizer with a geometrically decaying learning rate schedule. 10% of the training data was carved out as a validation set and used to perform early stopping if the validation loss did not improve after 30 consecutive epochs.

During testing, to help the hidden state to be more stable and better than its random initialization state [47], the test input is fed into the network for 2000 warm-up steps to update the hidden state without producing an output.

In the RNN, 3 main hyperparameters were tuned to determine the optimal model. They are (i) the size of the hidden state, (ii) the length of the time series used as input during training ζ_1, and (iii) the number of time-steps forward considered when calculating the loss during training, ζ_2.

As tuning for all 3 hyperparameters simultaneously will be computationally expensive, the hidden state size was tuned first using $\zeta_1 = 4$ and $\zeta_2 = 8$, followed by performing a 2D grid search on ζ_1 and ζ_2 while using the optimal hidden state size. The range of hyperparameters tested and the optimal parameters selected are reported in Appendix A.

2.2 Long Short-Term Memory (LSTM) Network

A problem that RNNs suffer from is vanishing gradients which result in the inability to capture the patterns of long sequential data [43]. Vanishing gradients occur when the gradients become too small, causing the model to stop learning. In the context of the RNN implemented in this thesis, this could be due to the hyperbolic tangent activation function which has a gradient between 0 and 1, which is then multiplied by the weight matrix W. As chain rule is applied during backpropagation, the gradients calculated become either exponentially smaller or larger the further

back the gradients are propagated, causing respectively minimal or excessive updates of the parameters. Hence, a solution that has been developed is the use of LSTM networks.

2.2.1 Equations and Parameters of the LSTM

The LSTM has the same basic structures of a RNN, but in addition to having a hidden state, the LSTM network also contains a cell state. LSTM solves the problem of vanishing gradients by introducing gates that control the flow of information from one time-step to another [11]. A LSTM is defined as follows:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f) \tag{2.4}$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i) \tag{2.5}$$

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o) \tag{2.6}$$

$$\tilde{C}_t = \tanh(W_g h_{t-1} + U_g x_t + b_g) \tag{2.7}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{2.8}$$

$$h_t = \tanh(C_t) \odot o_t \tag{2.9}$$

The forgot, input and output gates are represented by f_t, i_t, o_t respectively. $\tilde{C}_t$ is the cell input activation vector and is used to calculate the cell state, C_t. σ represents a sigmoid activation function and $\odot$ represents element-wise multiplication. All the W, U matrices are weight matrices, while the b vectors are the corresponding biases. In the LSTM, both the cell state and hidden state are propagated through time and used to calculate future states. Unlike in RNNs, the problem of vanishing gradient is avoided. In the backpropagation of the cell state, since it only consists of element-wise multiplication, the gradient is simply the value of the forget gate. The forget gate has a range of 0 to 1 due to the sigmoid activation function, and is not multiplied by the weight matrix (unlike the case in RNNs), hence heavily mitigating both the problems of a vanishing and exploding gradients. The backpropagation of the hidden state only contains one tanh function, hence the gradient is $o_t(1 - \tanh^2(C_t))$, which is a single value that will not result in a vanishing gradient. For more information on the parameters and derivation of the RNN and LSTM, the reader may read [38].

The prediction of the network is calculated identical to that of the RNN, where

$$\hat{y}_t = W_{out} h_t + b_{out}$$

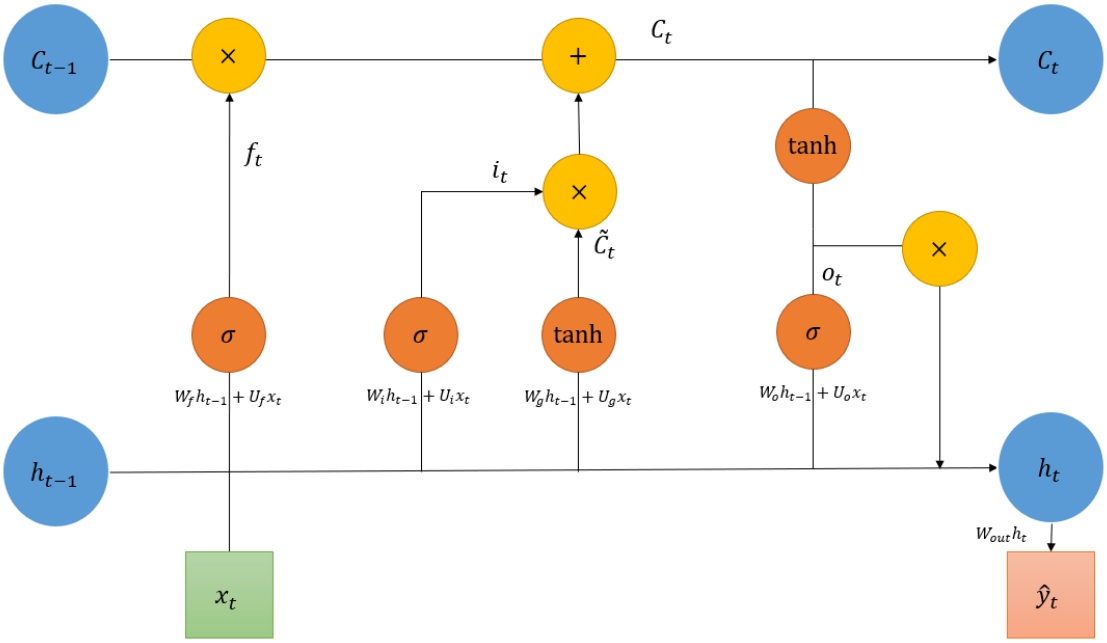


Figure 2.3: Structure of LSTM Cell

A visualisation of the LSTM network is shown in Figure 2.3.

2.2.2 Training of the LSTM

To train the LSTM, backpropagation through time is again utilized with the hyperparameters of ζ_1 and ζ_2. For a detailed explanation, the reader may refer to § 2.1.2.

The optimization of the LSTM was done with the ADAM optimizer using a geometrically decaying learning rate schedule. 10% of the training data was carved out as a validation set and used to perform early stopping if the validation loss did not improve after 30 consecutive epochs.

Similar to the RNN, a warm-up phase was implemented during testing to help stabilise the hidden state values. 3 main hyperparameters were tuned during the training of the LSTM, namely (i) the size of the hidden state, (ii) the length of the time series used as input during training ζ_1, and (iii) the number of time-steps forward considered when calculating the loss during training, ζ_2.

The hidden state size was tuned first, with $\zeta_1 = 4, \zeta_2 = 8$, followed by performing a 2D grid search on the remaining parameters using the optimal hidden state size.

For the full list of hyperparameters tested and used, please refer to Appendix A.

2.3 Reservoir Computing

Reservoir Computing separates two different portions of the network – a dynamic reservoir and a recurrence-free readout [31]. It works on the premise that the dynamic reservoir is a nonlinear time-based function that can encode the input history and patterns, while the readout can combine the signals from the reservoir into an output. Hence, the reservoir is created in a way to allow a diverse representation, while the readout is a function that is not time-dependent and is easy to learn.

2.3.1 Equations and Parameters of the RC

This thesis models the reservoir computing architecture following Echo State Networks (ESN) as introduced by Jaeger [21]. In ESNs, the reservoir is modelled as a fixed random RNN. This means that the equations governing the network are identical to in the RNN (Equation 2.2 and Equation 2.3). In some cases, a leaking rate α [46, 2] is introduced to the equation (Equation 2.10). α measures the speed that the reservoir is updated with accordance to new input.

$$h_t = (1 - \alpha) \cdot h_{t-1} + \alpha \cdot \sigma(W h_{t-1} + W_{in} x(t)) \tag{2.10}$$

When α is set to 1, the update rule is equivalent to the regular Reservoir Computing equation. As the value of α decreases to 0, the system keeps more information from previous states and reacts to new data less gradually.

With the reservoir being a fixed RNN, it signifies that it is not trained whereby W_{in}, W and b_h are not adjusted during training. Instead, a larger emphasis is placed on creating a good reservoir that can ensure better performance of the ESN. The aim of the reservoir is to create an extensive group of dynamics that can represent the time series. Hence, we want the reservoir to be big, sparsely, and randomly connected to ensure that the dynamics represented are respectively plentiful, loosely intercorrelated, and different from one another [31]. This is achieved by ensuring that the size of the hidden layer is large, the hidden layer weight matrix W is sparse, and the weights of W are generated randomly from a uniform distribution of [-ω, ω], where ω is a hyperparameter that can be tuned.

Another important element for the ESN is that the reservoir needs to adhere to the echo state property. The purpose of this property is to ensure that the effect of a previous hidden state and previous output should be removed gradually within subsequent time-steps. Although not a sufficient condition, it is usually enough in practice to ensure that the spectral radius of the system is below 1. It is difficult to construct a system where the echo state property is not satisfied while the spectral radius is less than 1 [31]. The spectral radius can be measured by the largest absolute eigenvalue of the hidden state weight matrix, W. Hence, W can be scaled so that the spectral radius is of a desired value. According to Jaeger [22], a larger spectral radius implies that the memory of the network will be longer and will take longer to forget the starting state. On the contrary, a lower spectral radius would be useful for problems where having a long memory may be detrimental.

With the reservoir being fixed after creation, only W_{out} and b_{out} are trained, which can save significant amounts of time. To include more nonlinearity, it is also possible to adapt the linear vector to include the squared term of the nodes

$$\hat{y}_t = W_{out}[h_t|h_t^2] + b_{out} \tag{2.11}$$

where | represents a vertical concatenation of the vectors. This allows the network to perform as a stronger universal approximator [15] and has been shown to perform better for dynamical systems [35].

2.3.2 Training of the RC

Due to the structure of the network, only the readout layer is needed to be trained. A set of hidden states are generated using the principles discussed above. Afterward, a subset of the initial hidden states is removed to account for the time required for the network to forget the starting state. In this way, the network will be trained to be independent of the random initial state.

A linear regression is trained using the targets against the remaining hidden states. To prevent overfitting, W_{out} is trained with regularized least-squares with Tikhonov regularization (also known as ridge regularization). However, due to the large hidden state size of the reservoir, the dimension of W_{out} is very large, rendering computational issues. Specifically, for a hidden state dimension $h \in \mathbb{R}^{d_h}$ and output dimension $o \in \mathbb{R}^{d_o}$, solving the linear regression $Y = W_{out}H$, where H represents

the concatenation of the linear and squared hidden state value, will require solving for $W_{out} \in \mathbb{R}^{d_o \cdot d_h * 2}$. Hence, a stochastic gradient descent method was employed. In this thesis, the stochastic gradient descent was optimized using the Adam optimizer with a learning rate of 1e-4.

Initial experimentations showed that the network performed best when $\alpha = 1$ in Equation 2.10, which meant that new hidden states were calculated purely based on the dot product of the previous hidden state and the input. In addition, ω (the range of the distribution W is sampled from) was discovered to not have a major impact on the predictions. Therefore, both α and ω were not further tuned in the network.

4 of the other main hyperparameters were tuned during the experiments conducted, namely (i) the size of the hidden state, (ii) the value of the Tikhonov regularization, (iii) the spectral radius, and (iv) the connectivity c of the network. The sparse connections were implemented by ensuring that for each row of the W matrix, only c of them were non-zero. For a full list of specifications of the hyperparameters used during the experiments and in the final model, the reader may refer to Appendix A.

2.4 Koopman Autoencoders

Koopman Autoencoders have two main aspects that are solved. First, we need to encode the data into a small latent space via autoencoders, followed by secondly, using this representation to learn an operator that governs how the data transforms overtime.

To understand Koopman Autoencoders, the thesis will first explore how Autoencoders work and how they help the application of the Koopman operator.

2.4.1 Introduction to Autoencoders

Autoencoders (AE) are a type of neural network that is widely used as an unsupervised learning method to help extract features and perform dimensionality reduction on data [10]. Autoencoders are implemented with 3 main portions, namely the encoder, the bottleneck and the decoder. The input is fed through the network with the aim of going through various projection operators to obtain an output

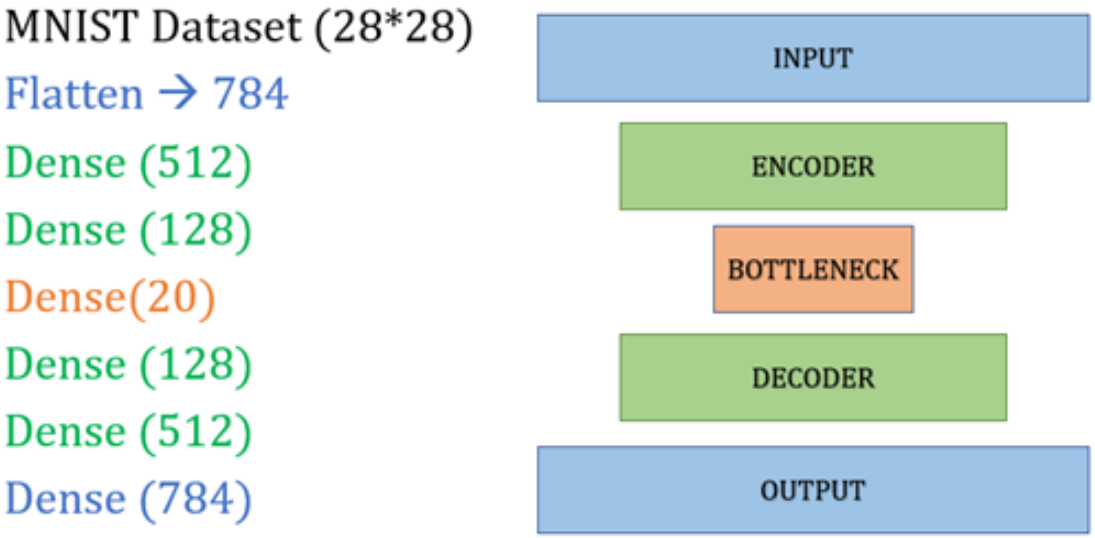


Figure 2.4: Autoencoder Architecture on MNIST Example

that is like the input. In the process of this projection, the input goes through a bottleneck, which is typically in a lower dimension. The representation at the bottleneck hence represents the important dimensions of the data, eliminating the redundant dimensions. The input, encoder layers, bottleneck layer, decoder layers and output are connected by fully connected hidden states. As an example (Figure 2.4), if an autoencoder was trained on a flattened MNIST dataset, the original dimension of the data would be 784, and it could be reduced in the encoder and bottleneck to a smaller dimension (e.g., 20). The decoder will then use the 20-dimension hidden layer to decode the output and produce a 784-dimension output which should resemble the input.

To train the network to learn the low-dimensional representation of the input, backpropagation and optimization of the network weights are done with the loss defined as the reconstruction loss. The reconstruction loss is the mean-squared error between the output and the input, denoting the effectiveness of the network at recreating the output. As a primer to Koopman Autoencoders and to understand the autoencoder network, an autoencoder was implemented and trained on the MNIST data. As seen from the outputs (Figure 2.5), the reconstructed images are not identical to the original images but contain the important features that make them recognizable as the original digits.

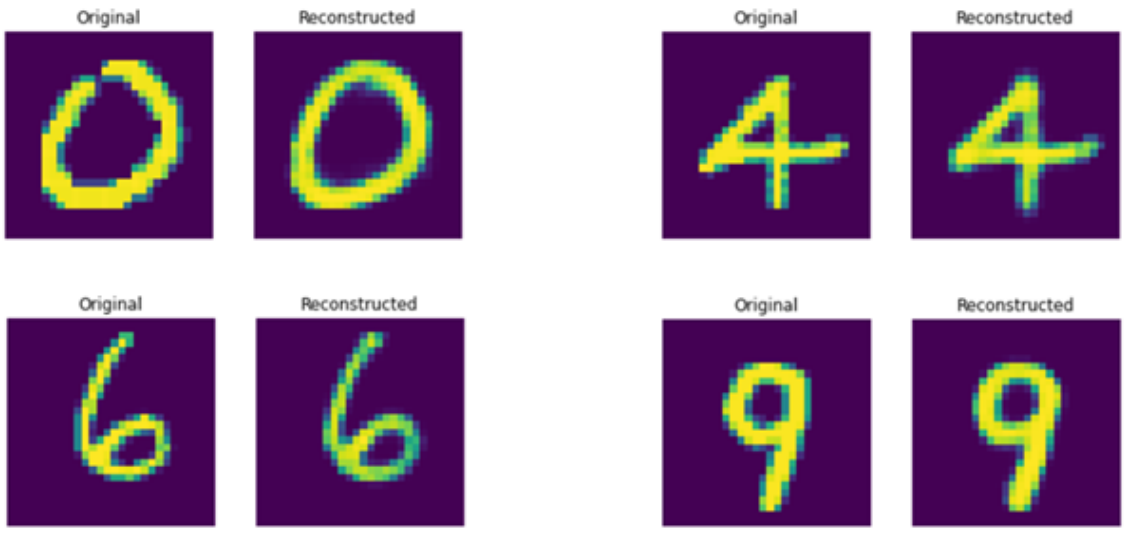


Figure 2.5: Example outputs produced by Autoencoder on MNIST data. In each pair, the left image is the original input, and the right image is the output after encoding and decoding

2.4.2 Equation and Parameters of Koopman Autoencoders

Koopman autoencoders build on the idea of autoencoders by trying to map the data into a linear space z which has the minimum essential information to describe the system. Koopman theory is based on the premise that a non-linear dynamical system can be expressed as a linear operator which is transmitted through time. In a dynamical system of the form $\frac{d}{dt}x(t) = f(x(t))$, it can be represented as $x_{t+1} = F(x_t)$, such that F defines how the dynamical system progresses from a step to the next. In the viewpoint of Koopman operator theory, there exist a function g which is a set of measurement functions derived from the state x. The Koopman operator, K is then a linear operator that advances the measurements functions forward.

$$K_\varphi g = g \circ F \tag{2.12}$$

For more details on Koopman theory for dynamical systems, the reader may refer to [7].

The goal of the Koopman Autoencoder (Figure 2.6) is that by using the encoder, the network seeks to find a set of measurement functions that can effectively encode the information of the input. After which, the algorithm learns the Koopman operator to learn how the measurement functions develop with time. There are several ways

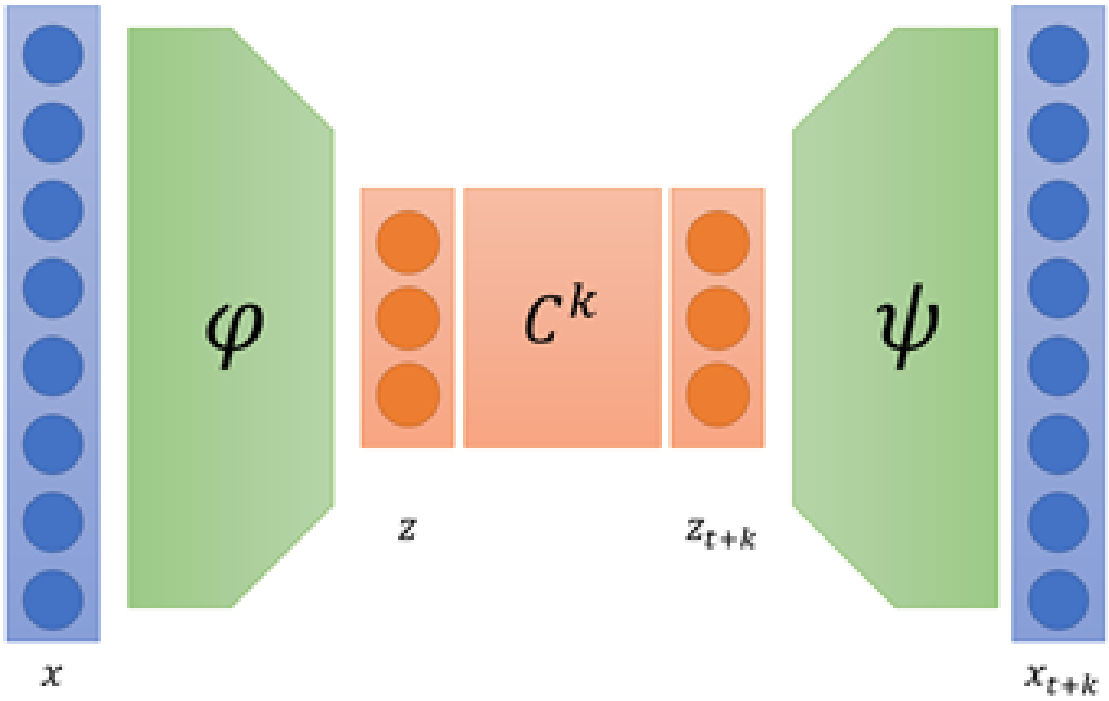


Figure 2.6: Structure of Koopman Autoencoder

that the Koopman operator can be represented. In the SINDy Autoencoders developed by Champion [9], the operator is represented as a non-linear sparse matrix. In Azencot and Erichson's research [3] that this thesis will be modelling after, the operator is learned as linear dynamic. This allows for the results to be more interpretable and simplifies the testing of new points. However, there is a larger requirement on the encoder-decoder network and may make it more difficult to learn the dynamic required [6].

After applying the linear operator, the data is then decoded using the learned autoencoder to produce the desired output at a future time-step Equation 2.13.

$$\hat{x}_{t+1} = (\varphi \circ C \circ \psi)(x_t) \tag{2.13}$$

where φ represents the encoder network, C is the forward Koopman operator and ψ is the decoder network. A major advantage of this network is that to predict k steps ahead, we can multiply by C^k instead.

In Azencot and Erichson's work [3], they have also included the usage of a backward dynamic; given a time-step t, the network is also trained to predict $t-1$.

This is useful for dynamics that may be invariant to the direction of time in the system. Hence, the prediction can be defined as follows:

$$\hat{x}_{t-1} = (\varphi \circ D \circ \psi)(x_t) \tag{2.14}$$

where D represents the backward operator.

2.4.3 Training of Koopman Autoencoders

The Koopman Autoencoder network is trained using backpropagation and optimization of the loss function. Due to the various parts of the network, the loss function comprises of different parts to encourage the accurate learning of the dynamics.

Similar to the vanilla autoencoder, there is a reconstruction loss that measures the quality of the encoder-decoder network. For N points from $x_1, \ldots, x_N$,

$$\epsilon_{recon} = \frac{1}{2N} \sum_{t=1}^{N} (\hat{x}_t - x_t)^2 \tag{2.15}$$

During training, we can choose how many forecast steps to include into our loss. This hyperparameter l can be tuned to improve the generalization of the network. To account for this, the forward and backward prediction loss is the average of the mean-squared errors of the predictions.

$$\epsilon_{fwd} = \frac{1}{2Nl} \sum_{k=1}^{l} \sum_{t=1}^{N} (\hat{x}_{t+k} - x_{t+k})^2 \tag{2.16}$$

$$\epsilon_{bwd} = \frac{1}{2Nl} \sum_{k=1}^{l} \sum_{t=1}^{N} (\hat{x}_{t-k} - x_{t-k})^2 \tag{2.17}$$

Lastly, to link the C and D matrices such that $CD = I_\kappa$, an error to encourage consistency is included.

$$\epsilon_{con} = \sum_{k=1}^{\kappa} \frac{1}{2k} ||D_{k*}C_{*k} - I_k||^2 + \frac{1}{2k} ||C_{k*}D_{*k} - I_k||^2 \tag{2.18}$$

where κ is the size of the matrices, D_{k*}, C_{*k} are the upper k rows of D and leftmost k columns of C respectively. The losses are then summed with different weights placed on the losses via the loss coefficients to make up the total loss, ϵ.

$$\epsilon = \lambda_{recon}\epsilon_{recon} + \lambda_{fwd}\epsilon_{fwd} + \lambda_{bwd}\epsilon_{bwd} + \lambda_{con}\epsilon_{con} \tag{2.19}$$

In this thesis, the optimization of the weights through backpropagation was done using an Adam optimizer with a learning rate schedule that diminishes by a factor of 10. To model the more complicated dynamics of the chaotic systems, 3 hidden layers were used in the encoder-decoder network. The number of nodes in each hidden layer was tuned, including trying an overcomplete autoencoder. In the overcomplete autoencoder, the data is mapped into a higher dimension, where it is theorized that non-linear dynamics become more linear in high dimension representations [7], allowing the Koopman operator to be learned. However, this reduces the interpretability of the latent space and is not suitable for problems that require interpretable results.

In addition to the network architecture, the number of steps used during the calculation of loss, l was also tuned. Lastly, the relative weights of the reconstruction, forward, backward and consistency losses were also tuned to encourage emphasis in different parts of the Koopman Autoencoder. A full list of the hyperparameters tuned and parameters used can be found in Appendix A.

Chapter 3

Results of Forecast Prediction

3.1 Evaluation Metrics for Predictions

Two main metrics are evaluated to provide comparison between results. The first metric is the Normalized Root Mean Squared Error (NRMSE) of the prediction, represented as

$$NRMSE = \frac{1}{N}\sqrt{\frac{(\hat{\mathbf{y}} - \mathbf{y})^2}{\sigma_{\mathbf{y}}^2}} \tag{3.1}$$

where N is the length of the prediction horizon. Since the data was normalized § 1.3, $\sigma_{\mathbf{y}}^2 = 1$, and the NRMSE is reduced to the root of the squared difference between the predicted output and the target, averaged over the prediction horizon. As a baseline, the NRMSE of a random guess is approximately 1, regardless of the prediction horizon. Hence, the NRMSE of the models are expected to be significantly less than 1, especially in the short-term.

The second metric measures the time taken for the NRMSE to cross a specified threshold. For a threshold k, the k-valid prediction horizon (PH_k) is defined as

$$PH_k = \arg\max_t (NRMSE(t) < k) \tag{3.2}$$

To ensure a consistent baseline across different models and datasets, t is calculated in terms of Lyapunov time (Refer to § 1.2 for more details). The determination of optimal hyperparameters was done by comparing $PH_{0.5}$ and selecting the configurations that resulted in the longest prediction horizon.

In this section, for each of the optimal models, the median test trajectory is taken from the 100 test trajectories and used to report the metrics. For a more

complete set of results, refer to Appendix B. The thesis discusses the $PH_{0.5}$ and the NRMSE at $[0.5, 1, 2, 5]$ Lyapunov times (represented as $NRMSE_t$), representing the error in the short and long horizon. As an extra consideration, the time taken to train and test each of the models is also reported.

3.2 Lorenz-96 System

The 4 methods were coded out and tuned for the hyperparameters using the Lorenz-96 dynamical system. The results are summarized in Table 3.1 and Figure 3.1.

Method	$PH_{0.5}$	$NRMSE_{0.5}$	$NRMSE_1$	$NRMSE_2$	$NRMSE_5$
RNN	1.89	0.077	0.183	0.542	1.233
LSTM	2.79	0.037	0.089	0.255	1.154
RC	1.79	0.091	0.225	0.618	1.250
Koopman AE	0.00	0.834	0.894	0.944	0.975

Table 3.1: Summary of metrics for prediction of the Lorenz-96 system. Green represents the best, while red represents the worst

The final RNN and LSTM models were both trained using a 500-sized hidden state while the RC had a reservoir size of 12 000. In this example, the RNN and RC methods behaved similarly, exceeding a NRMSE of 0.5 after around 1.8 Lyapunov time. The LSTM performed the best and was able to predict under a NRMSE of 0.5 for an additional Lyapunov time. However, the Koopman Autoencoder approach was unable to capture the dynamics of the system. Even at 1 time-step ahead, the NRMSE had already exceeded 0.5 and the model was predicting an approximately constant value after 1.5 Lyapunov time (Figure 3.2e). This resulted in the system having a NRMSE of 1.0 for majority of the prediction horizon, which is equivalent to the error exhibited by a random guess. In the other methods the NRMSE had exceeded 1.0 after 3 to 4 Lyapunov time, which indicates that the predictions are unreliable in the long-term.

Since the Koopman Autoencoder was unable to find useful dynamics during training, it hit the early stopping criteria very quickly, resulting in the low train times (Table 3.2). Although the Reservoir Computing (RC) approach was supposed to be faster due to not needing to train the internal reservoir, the Reservoir Computing approach required a very large reservoir to compute the dynamics accurately. This

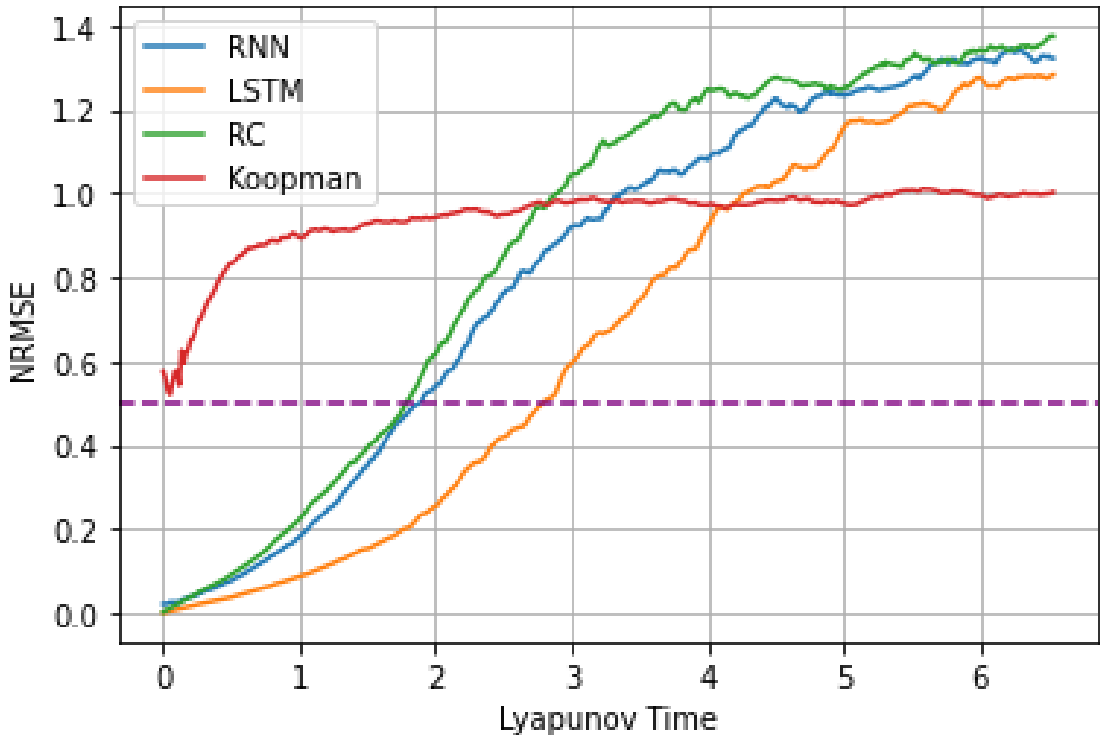
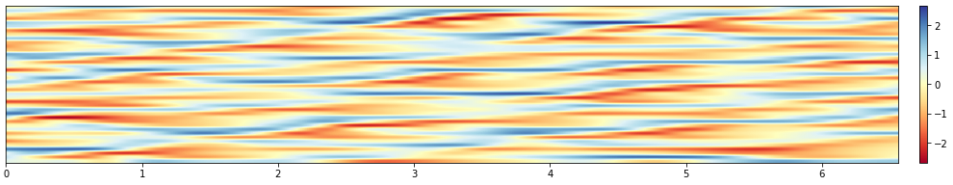


Figure 3.1: Prediction Horizon of methods in the Lorenz-96 system

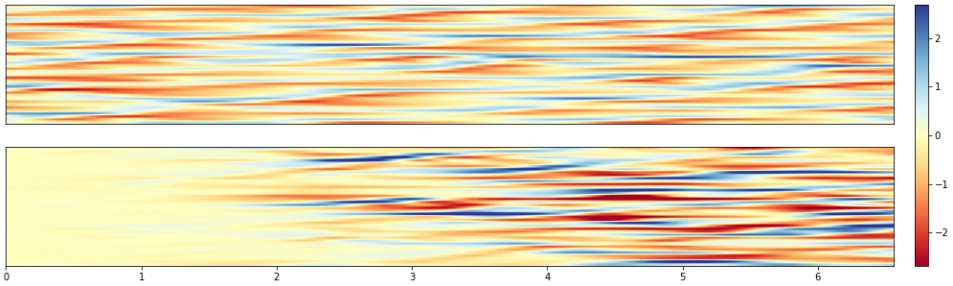
Method	Train Time (s)	Test Time (s)
RNN	1346.14	0.38
LSTM	1274.72	1.04
RC	4551.81	223.66
Koopman AE	388.51	1.58

Table 3.2: Time taken for training and testing of the Lorenz-96 system Green represents the best, while red represents the worst

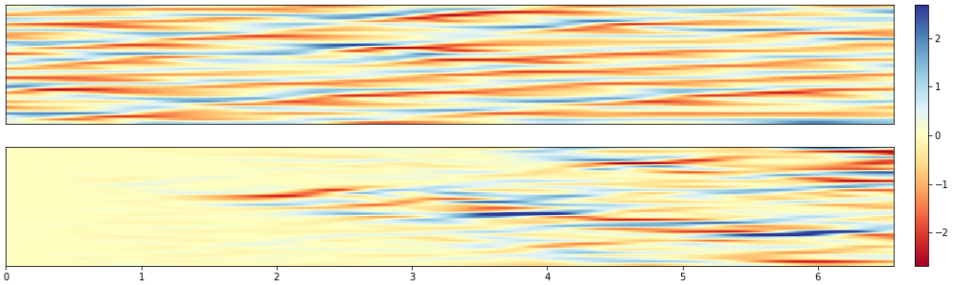
caused major strain when needing to perform large matrix multiplications both during training and testing, resulting in times that were many times longer than that of the RNN and LSTM. Of note, a reservoir size of 3 000 took 881.83 seconds to train and 72.71 seconds to test, which is less time than it takes to train the 500-sized LSTM network. However, the smaller reservoir is only able to accurately predict for 1 Lyapunov time, and hence was not selected as the optimal model. To achieve an accuracy equal to the 12 000-sized RC, a LSTM with a hidden state size of 100 was sufficient and took only 714.15 seconds to train.



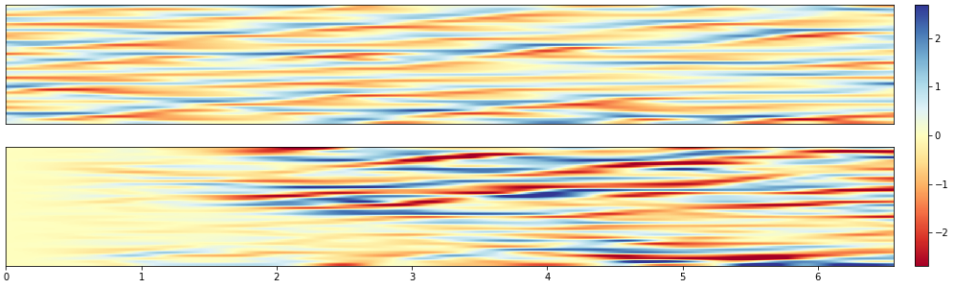
(a) Actual Trajectory



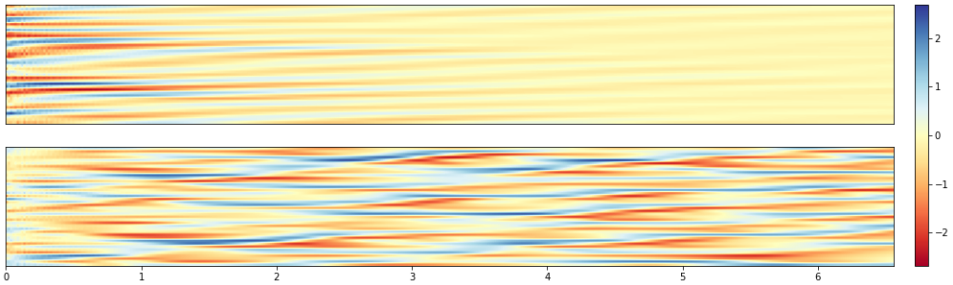
(b) RNN Trajectory (Top) Prediction (Bottom) Error



(c) LSTM Trajectory (Top) Prediction (Bottom) Error



(d) RC Trajectory (Top) Prediction (Bottom) Error



(e) Koopman AE Trajectory (Top) Prediction (Bottom) Error

Figure 3.2: Trajectories and Errors of a sample Lorenz-96 system (Lyapunov time)

3.3 KS System

As an additional experiment, the 4 methods were also tested against the KS Equation which was spatially discretized into 240 dimensions. This system has a larger input dimension than that of the Lorenz-96 system tested, which was tested with a dimension of 40. The models were tuned for the optimal hyperparameters, and the relevant metrics were calculated. The results are summarized in Table 3.3 and Figure 3.3.

Method	$PH_{0.5}$	$NRMSE_{0.5}$	$NRMSE_1$	$NRMSE_2$	$NRMSE_5$
RNN	1.38	0.120	0.311	0.869	1.395
LSTM	2.23	0.059	0.138	0.400	1.353
RC	2.19	0.055	0.142	0.427	1.345
Koopman AE	0.25	0.791	1.002	1.099	1.071

Table 3.3: Summary of metrics for prediction of the KS system. Green represents the best, while red represents the worst

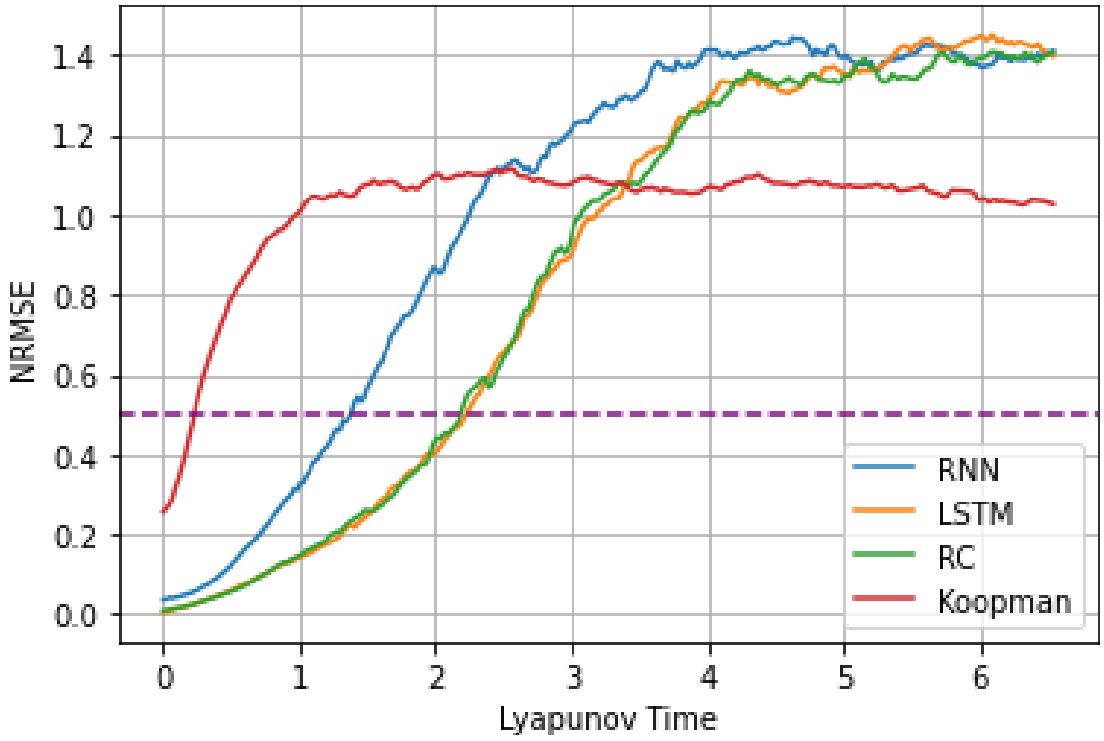
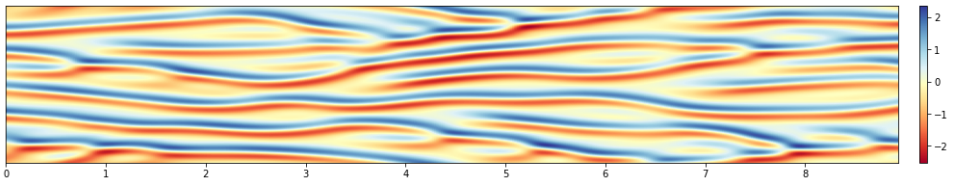
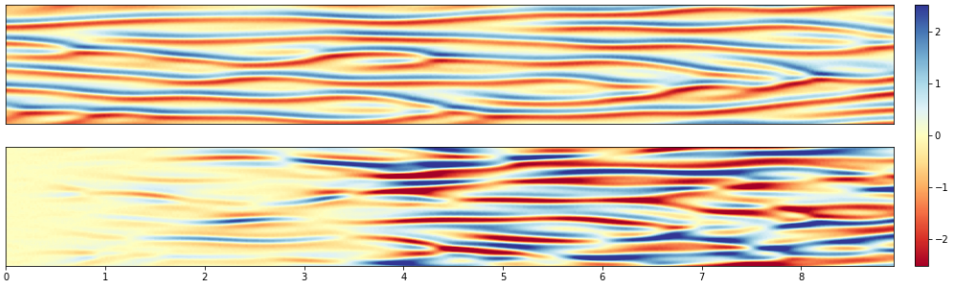


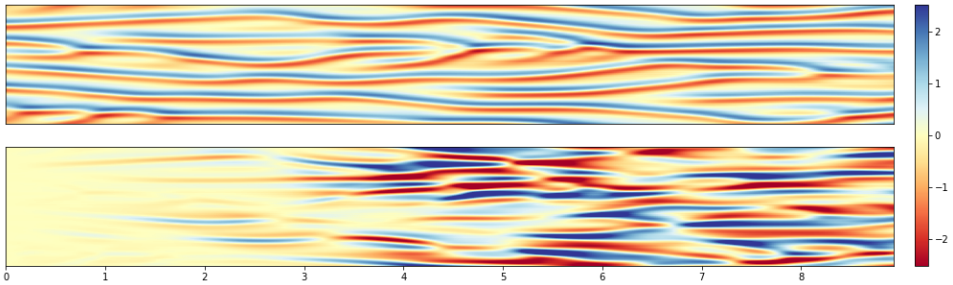
Figure 3.3: Prediction Horizon of methods in the KS system



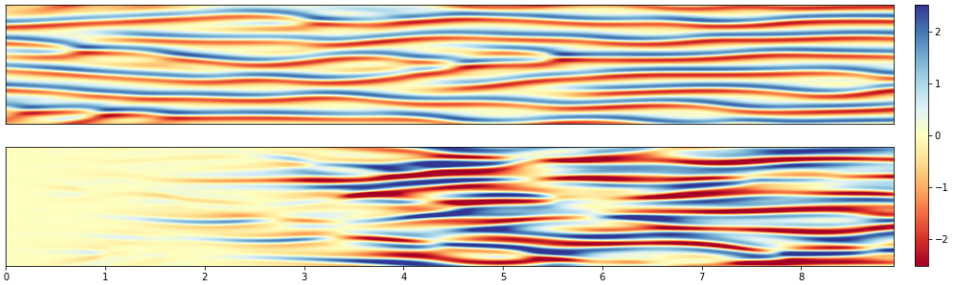
(a) Actual Trajectory



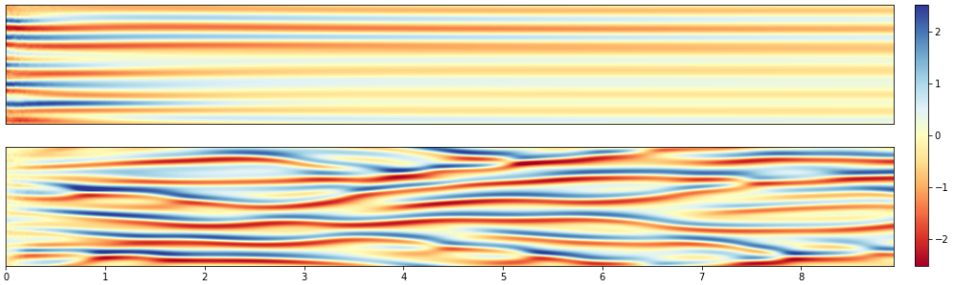
(b) RNN Trajectory (Top) Prediction (Bottom) Error



(c) LSTM Trajectory (Top) Prediction (Bottom) Error



(d) RC Trajectory (Top) Prediction (Bottom) Error



(e) Koopman AE Trajectory (Top) Prediction (Bottom) Error

Figure 3.4: Trajectories and Errors of a sample KS system (Lyapunov time)

For the RNN and LSTM, a hidden state size of 500 still performed the best, with larger sizes showing signs of overfitting. However, the RC approach performed better with larger hidden state sizes, with a 15000-sized reservoir. It was observed that although LSTM still outperformed the other methods, the gap between the RC predictions and the LSTM predictions were a lot smaller. The prediction horizon of LSTM was only 0.04 Lyapunov time better than the RC approach. This hints at the possibility of RC performing better under high-dimensional settings. The Koopman Autoencoder approach was still unable to capture the dynamics of the system, where the optimal network architecture did not reduce the size of the input dimension, resulting in a bottleneck of size 240. The predictions after a longer horizon still became constant (Figure 3.4e), resulting in a NRMSE of 1.0 after 1 Lyapunov time.

Method	Train Time (s)	Test Time (s)
RNN	2107.61	0.39
LSTM	2762.25	0.83
RC	9498.11	672.32
Koopman AE	2025.50	0.23

Table 3.4: Time taken for training and testing of the Lorenz-96 system Green represents the best, while red represents the worst

With the larger dimension of the dataset, the training times of the methods were generally longer (Table 3.4). The RC approach utilised larger hidden state sizes, thereby causing the training times to be inflated compared to the Lorenz-96 system. Since the Koopman Autoencoder performed best with a complete autoencoder, the final neural network was complex and had many parameters to tune, resulting in a longer training time.

3.4 Investigating the Impact of Size of Dataset

Traditional Neural Network methods are known to require a lot of data to perform well. To test the impact of the size of the data on the 4 methods discussed, the models were trained with only 20% of the Lorenz-96 training dataset and evaluated on the same test set as before. The prediction horizon and NRMSE were then calculated and shown in Table 3.5 and Figure 3.5.

Method	$PH_{0.5}$	$NRMSE_{0.5}$	$NRMSE_1$	$NRMSE_2$	$NRMSE_5$
RNN	0.51	0.499	0.884	1.137	1.248
LSTM	0.20	0.858	0.973	1.028	1.025
RC	1.02	0.223	0.493	0.992	1.352
Koopman AE	0.18	0.865	0.962	0.983	0.995

Table 3.5: Summary of metrics for prediction of the small Lorenz-96 system. Green represents the best, while red represents the worst

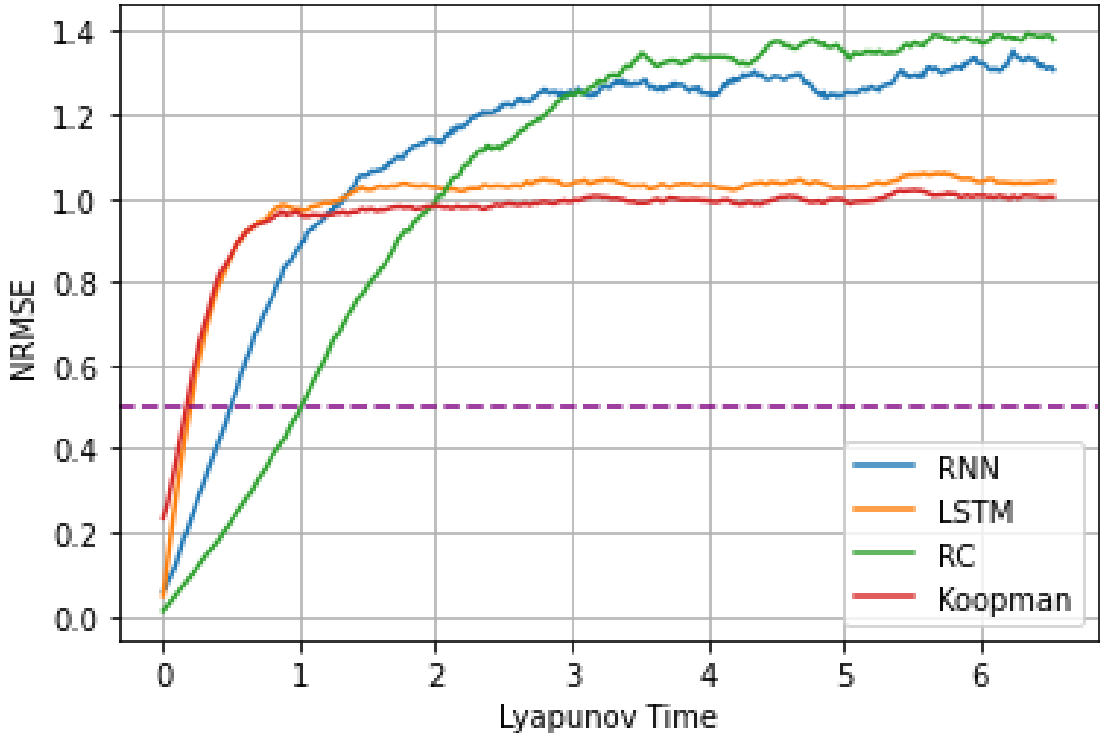
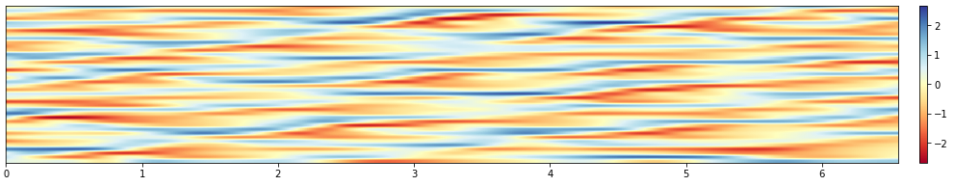
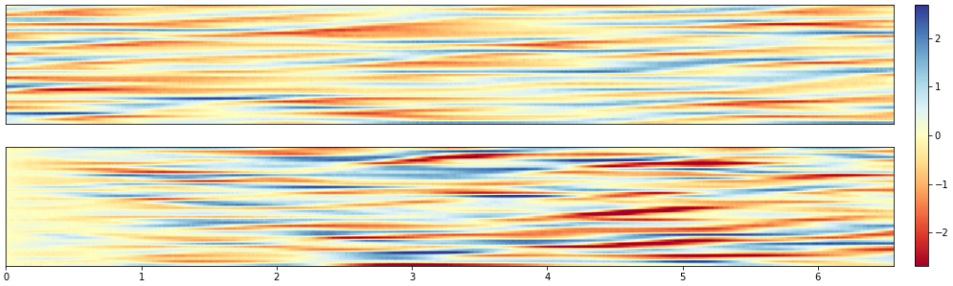


Figure 3.5: Prediction Horizon of methods in the small Lorenz-96 system

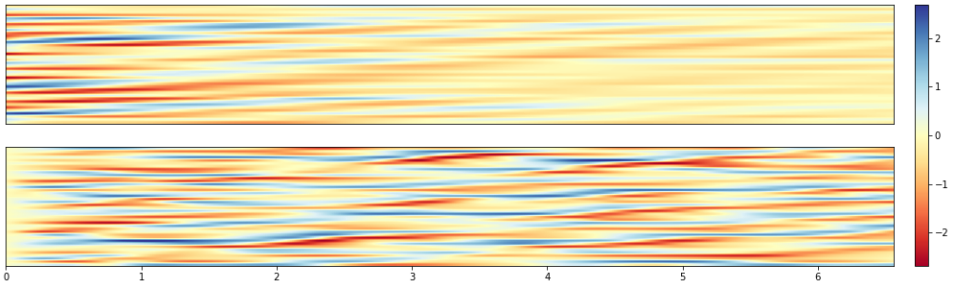
Expectedly, the training was a lot more unstable and the prediction horizon for all the methods were worse than before. However, the LSTM network was affected very heavily and was not able to perform much meaningful predictions (Figure 3.6c). Interestingly, the Koopman Autoencoder approach performed best with an overcomplete autoencoder network, though it largely still predicted a constant value after a short time (Figure 3.6e). The RC network was still able to perform predictions for 1 Lyapunov time, indicating that it may be more resilient to the lack of data. Although a reservoir of hidden state size 15 000 was used, a hidden state size of 3 000 performed had a prediction horizon of only 0.13 Lyapunov time less.



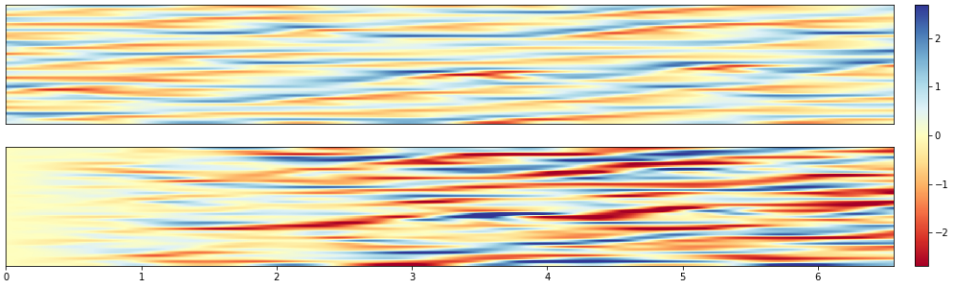
(a) Actual Trajectory



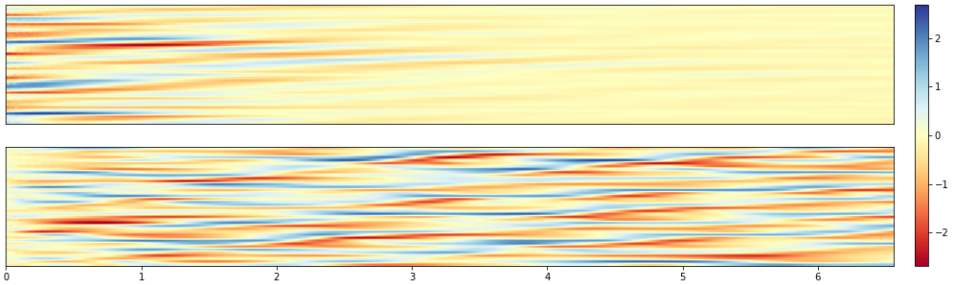
(b) RNN Trajectory (Top) Prediction (Bottom) Error



(c) LSTM Trajectory (Top) Prediction (Bottom) Error



(d) RC Trajectory (Top) Prediction (Bottom) Error



(e) Koopman AE Trajectory (Top) Prediction (Bottom) Error

Figure 3.6: Trajectories and Errors of a small Lorenz-96 system (Lyapunov time)

Method	Train Time (s)	Test Time (s)
RNN	607.94	0.39
LSTM	741.39	1.02
RC	874.03	240.77
Koopman AE	342.43	0.38

Table 3.6: Time taken for training and testing of the Lorenz-96 system Green represents the best, while red represents the worst

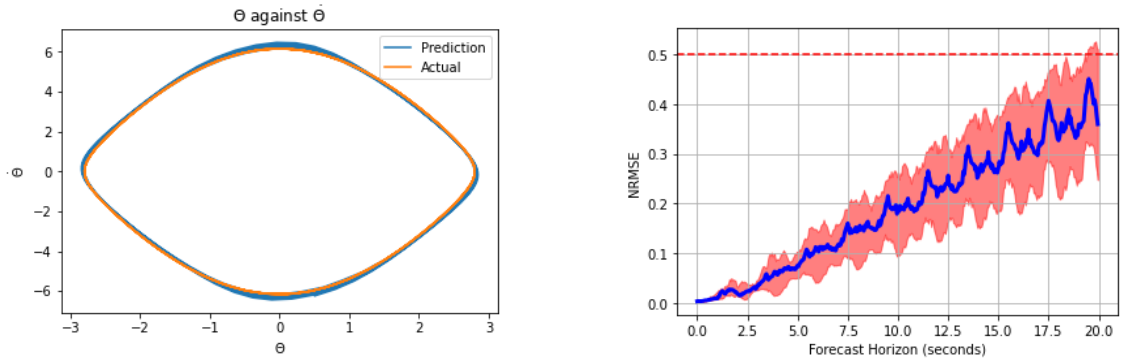
The RC network was still slowest the method in training and testing (Table 3.6). However, due to most of the computation time being on the matrix multiplication during the loss calculation, the reduction in dataset size drastically reduces the time required to train the model. Using a hidden state size of 3000 took only 225.76 seconds to train and 77.38 seconds to test, outperforming the other methods while still having a longer prediction horizon than them.

3.5 Discussion

Through the 3 experiments conducted, there are a few observations that are crucial when determining the usefulness of each of the methods.

Firstly, the chaotic behaviour of the dynamical systems affects the quality of the predictions. In all the experiments, it was observed that the Koopman Autoencoder method was unable to produce meaningful results. The Lorenz-96 and the KS system have complicated non-linear relationships which may not be easily represented by a network. The Koopman Autoencoder was tested against a pendulum dynamical system represented by $\frac{d^2\theta}{dt^2} - \frac{g}{L}sin\theta = 0$, where $g = 9.8, L = 1$. The system was projected into a 64-dimensional space using QR decomposition and trained to test if the model could learn the original dynamics. As seen in Figure 4.7, when the predicted output was mapped to the 2-dimensional space, the Koopman Autoencoder was able to predict the trajectory very well even after 20 seconds. Therefore, the Koopman Autoencoder method may be useful for simpler dynamical systems but needs to be adapted and improved for chaotic systems.

Secondly, the size of the dataset has a large impact on the quality of the algorithms, both in terms of time and performance. In situations of limited data, the RC approach performs better than the other data-hungry neural networks. In



(a) Comparison of actual and predicted values for $\dot{\theta}$ against θ

(b) NRMSE of predictions across 20 seconds

Figure 3.7: Trajectory and Error of Koopman Autoencoder predictions on a pendulum system

addition, it is noticed that the prediction horizon is asymptotic with the increase of the RC network size. Hence, under some situations, it is possible to train a network with a small reservoir size and achieve decent results. It may be thus worth considering using RC networks as a quick training model to estimate results in a small to moderate sized datasets. However, in other situations where data is abundant, RC networks perform very slowly due to the many high-dimension matrix multiplications that need to occur while a LSTM network is more optimized to train and perform the predictions required.

Lastly, the dimension of the system also plays a role in the performance of a prediction method. As discussed in § 3.4, the RC method seems to still be able to perform even when the dimension of the system is high. More rigorous work can be conducted in the future to verify the relationship between the dimensionality of the data and the relative performance of the various algorithms.

Chapter 4

Uncertainty Quantification Methods

Uncertainty quantification is important for the users to know how much trust to put into a forecast provided by the model. This is especially important in situations where a wrong prediction can result in major detrimental effects [26]. However, current literature on uncertainty quantification focuses on the estimation of either low-dimension problems or single-step regression problems [27, 33, 34, 36, 40]. Alternatively, complex Bayesian Neural Networks are discussed, which although can provide reliable uncertainty estimates, are difficult to implement and time-consuming to train [26, 33]. Hence, this thesis aims to explore uncertainty quantifications methods that are effective yet simple to implement for high-dimensional chaotic systems. In addition, the thesis also discusses if the form of network used has an impact on the uncertainty estimates that can be provided by the network.

Uncertainty can be split into epistemic and aleatoric uncertainty. Epistemic uncertainty refers to the uncertainty of the model parameters and measures the accuracy of the estimate of the true regression. This can be solved by having more data or a more accurate network. By estimating the epistemic uncertainty of a model, we can obtain confidence intervals for the predictions. On the other hand, aleatoric uncertainty refers to the noise from the data itself due to the randomness of the data generation process or intrinsic noise that may exist from measurement machines. Combining the uncertainties can provide us with a prediction interval, which considers the accuracy with which we predict the target themselves [17, 33].

In this section, the thesis explores the theory behind two uncertainty quantification methods, namely the use of deep ensembles, and the mean variances estimation

method. The deep ensembles are aimed at targeting the epistemic uncertainty and providing a confidence interval, while the use of mean variance estimation targets the aleatoric uncertainty and helps to provide a prediction interval for the predictions.

4.1 Deep Ensembles

Initially introduced by Heskes [17] as a bootstrap method, the method proposes to run n_{run} networks and calculate the mean and variance of the bootstrap, whereby

$$\hat{\mu} = \frac{1}{n_{run}} \sum_{i=1}^{n_{run}} \hat{y}_i \tag{4.1}$$

$$\hat{\sigma}^2 = \frac{1}{n_{run} - 1} \sum_{i=1}^{n_{run}} (\hat{y}_i - \hat{\mu})^2 \tag{4.2}$$

In the proposed bootstrap method, each ensemble is trained on different subsets of the training data. This is useful for algorithms such as convex optimization where there is no intrinsic randomization in the training process. However, it has been observed that for neural networks, this is not needed and is instead detrimental to the learning process. Neural networks have many local optima and rely on large number of data points to learn the concept, hence performing multiple runs of the neural network with different initializations is sufficient to create an ensemble of results [27]. After training the ensemble, a confidence interval can be created by

$$\hat{y} \pm \hat{\sigma} \cdot C(\alpha) \tag{4.3}$$

where $C(\alpha)$ represents the confidence level $1 - \alpha$.

4.2 Mean Variance Estimation with Sampling

In the traditional neural network methods described, the goal of a regression problem is to output a single point prediction, $\hat{y}$ and optimize the loss for the mean-squared error between the predicted output and the actual target, $MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2$. However, this method only accounts for the accuracy of the system but does not provide information on the confidence of the predictions. Inspired by Nix and Weigend [34], we want the network to also be able to compute a standard deviation, σ. This can be achieved by ensuring that the network instead has two

outputs, one representing the mean, μ, and a second to estimate the standard deviation, σ. To ensure that the standard deviation is strictly positive, the output of the network is instead set to optimize for $\log \sigma$, which is then transformed using the exponent function as below:

$$\sigma_t = \exp(W_s h_t + b_s) \tag{4.4}$$

where W_s is the weight matrix between the hidden state and the output $\log(\sigma)$ node, h_t is the hidden state at time t and b_s represents the bias of the hidden state to output node regression. In this case, to ensure a simple and quick estimation, we treat σ as a constant which hence assumes the noise in the data is perturbed by a similar distribution.

To account for the standard deviation, the neural network loss will instead be calculated using the negative log-likelihood loss,

$$NLL = \frac{1}{2}(d\log(2\pi) + d\log(\sigma) + \frac{|x-\mu|^2}{\sigma^2}) \tag{4.5}$$

where d is the output dimension of the data. Note that the first term is a constant and in practice, can be left out in the optimization process.

However, since the standard deviation is not fed back into the system, this means that the standard deviation of the predictions will be relatively constant, regardless of the prediction horizon. This is counter-intuitive to the idea of a long-term prediction of sequential data as we expect predictions to become increasingly uncertain due to initial variance propagating forward and resulting diverging predictions. To include the time aspect in the uncertainty, we propose that using the mean, μ_{t_0} and variance, $\sigma^2_{t_0}$, generated for time t_0, we take a sample from the distribution $N \sim (\mu_{t_0}, \sigma^2_{t_0})$ and use it as the first output. This is then fed into the next hidden state and used to calculate the mean and variance of the remainder of the horizon. This is repeated for n_{traj} different trajectories such that for each time step t, there will be n_{traj} μ_t and n_{traj} σ^2_t. This creates a mixture of Gaussians whose mean μ_* and variance σ^2_* are

$$\mu_* = \frac{1}{n_{traj}} \sum_{i=1}^{n_{traj}} \mu_i \tag{4.6}$$

$$\sigma^2_* = [\frac{1}{n_{traj}} \sum_{i=1}^{n_{traj}} (\sigma_i^2 + \mu_i^2)] - {\mu_*}^2 \tag{4.7}$$

4.3 Mean Variance Estimation Deep Ensemble

Alternatively, another way to incorporate the model uncertainty is to combine the idea of the Deep Ensemble (§ 4.1) and the Mean Variance Estimation (§ 4.2). We trained an ensemble of models that had 2 output nodes and were trained to optimize the negative log-likelihood. The output will be n_{run} sets of μ and σ^2 with the full prediction horizon. Similar to Mean Variance Estimation with Sampling (§ 4.2), this forms a mixture of Gaussian models, which we can then use to derive $\mu_* = \frac{1}{n_{run}} \sum_{i=1}^{n_{run}} \mu_i$ and $\sigma_*^2 = [\frac{1}{n_{run}} \sum_{i=1}^{n_{run}} (\sigma_i^2 + \mu_i^2)] - {\mu_*}^2$.

Chapter 5

Results of Uncertainty Quantification

In this section, the thesis will only explore the two methods that were more successful in performing predictions, namely the LSTM and Reservoir Computing (RC) networks. The uncertainty quantification methods discussed in Chapter 4 will be experimented on the Lorenz-96 dataset using the two methods and compared to provide a judgement on the networks and the uncertainty quantification methods.

5.1 Evaluation Metrics for Uncertainty Quantification

To determine the quality of an uncertainty quantification method, we use the negative log-likelihood to capture the predictive uncertainty. The likelihood of a system is the probability of observing an output given the mean and variance (Equation 5.1). The negative logarithm of the likelihood is then taken to obtain the negative log-likelihood (Equation 5.2).

$$\begin{aligned} LH &= P(Y|\mu,\sigma) \\ &= \frac{1}{(2\pi)^{d/2}} \cdot \frac{1}{\sigma^{d/2}} \cdot \exp \frac{|x-\mu|^2}{2\sigma^2} \end{aligned} \tag{5.1}$$

$$\begin{aligned} -\log LH &= -(-\frac{d}{2}\log(2\pi) - \frac{d}{2}\log\sigma - \frac{|x-\mu|^2}{2\sigma^2}) \\ &= \frac{1}{2}(d\log(2\pi) + d\log(\sigma) + \frac{|x-\mu|^2}{\sigma^2}) \end{aligned} \tag{5.2}$$

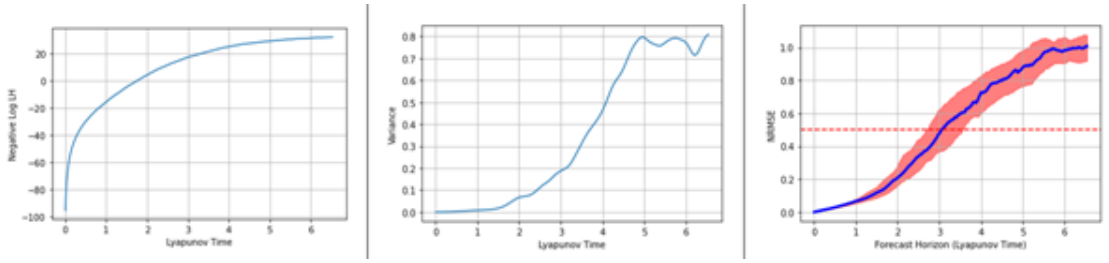
The lower the value of the negative log-likelihood, the better the quality of the prediction uncertainty. Applying a baseline prediction of mean 0 with variance 1, the negative log-likelihood is approximately 37.26 throughout the prediction horizon.

Although not the main metric of evaluating the prediction uncertainty, the thesis also discusses the MSE of the predictions to gauge the impact of the uncertainty quantification methods applied.

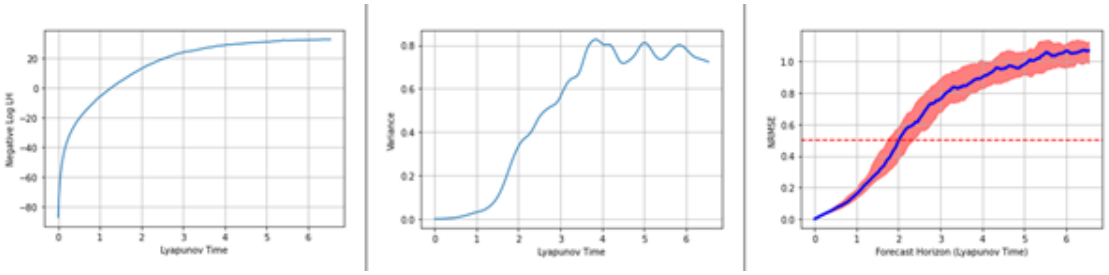
5.2 Deep Ensembles

The Deep Ensemble was trained by applying the optimal parameters (as determined in Chapter 3) and training $n_{run} = 5$ models using different seeds. The mean and variance of the predictions were then calculated to form a prediction bound for the test set. As seen in Figure 5.1, the negative log-likelihood at short intervals of Lyapunov time are small, and gradually increase as the system is less certain of its prediction of the actual output. The variance of the mixture also represents this uncertainty by reaching a variance of 0.8 in the longer-term.

One of the benefits of using Deep Ensembles is the by-effect of having an



(a) LSTM Network

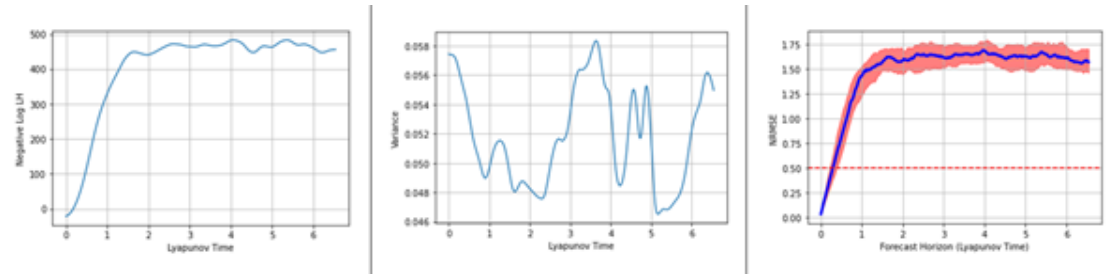


(b) RC Network

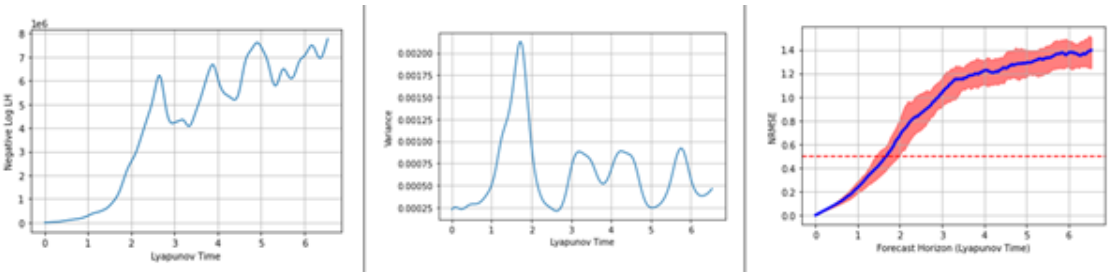
Figure 5.1: Deep Ensembles results for LSTM and RC Networks. (Left) Negative Loglikelihood (Middle) Variance (Right) NRMSE

improvement point prediction. In the LSTM network, the $PH_{0.5}$ increased from 2.79 Lyapunov time to 3.05 Lyapunov time while in the RC network, the $PH_{0.5}$ increased from 1.79 Lyapunov time to 2.02 Lyapunov time. However, ensemble models are also computationally expensive, multiplying the time required to train the model by the number of ensembles created.

5.3 Mean Variance Estimation with Sampling



(a) LSTM Network

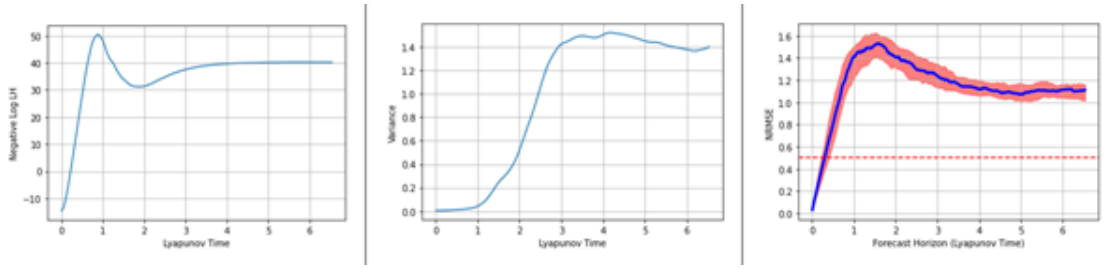


(b) RC Network

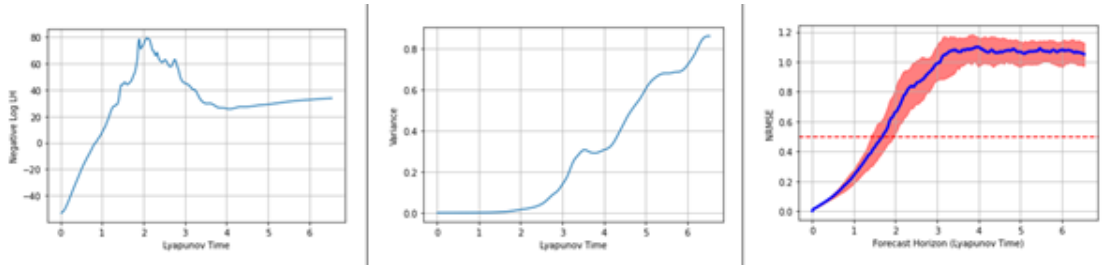
Figure 5.2: Original MVE results for LSTM and RC Networks. (Left) Negative Loglikelihood (Middle) Variance (Right) NRMSE

The LSTM and RC networks were first trained with the original Mean Variance Estimation (Figure 5.2). As expected, the variance of the network remained relatively stable throughout. This caused the negative log-likelihood to increase exponentially as the network was unreasonably confident in the longer-term predictions. In the LSTM network (Figure 5.2a), it was also observed that the network was overfitting during training. The training log-likelihood was achieving values below -80 but the validation log-likelihood surpassed 10 000. Hence, the network was constantly terminated due to early-stopping and could not learn the dynamics effectively.

The sampling process (as described in § 4.2) was implemented to improve the quality of the uncertainty quantification. In the case of the RC network, only



(a) LSTM Network



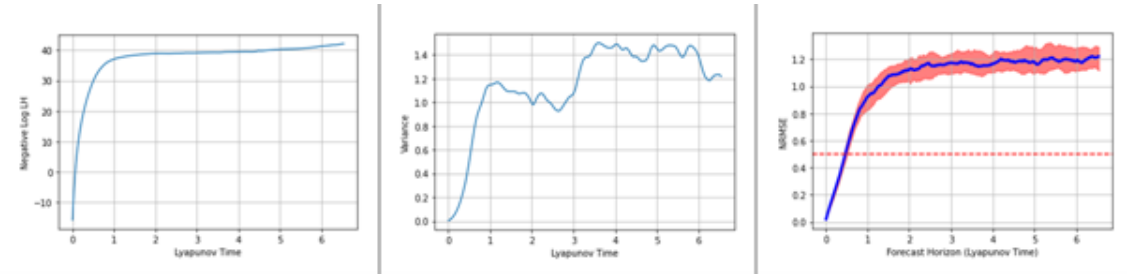
(b) RC Network

Figure 5.3: MVE with Sampling results for LSTM and RC Networks. (Left) Negative Loglikelihood (Middle) Variance (Right) NRMSE

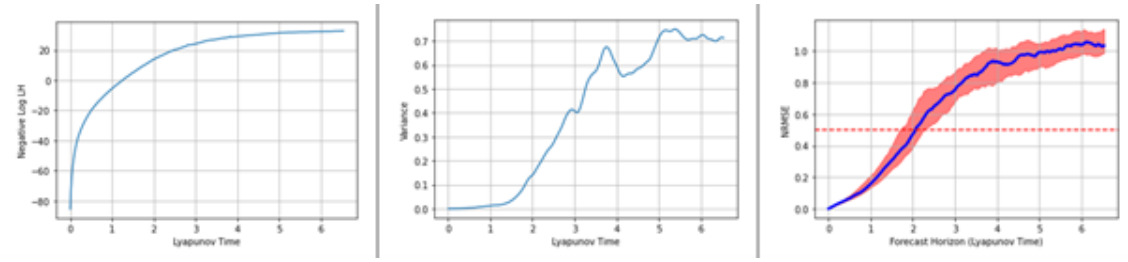
$n_{traj} = 20$ (instead of $n_{traj} = 100$ as in the LSTM) trajectories were sampled from the initial starting point due to the excessive computational cost of generating trajectories. Although there was an improvement in the negative log-likelihood for both networks, the LSTM network exceeded the baseline negative log-likelihood after 0.61 Lyapunov time while the RC network exceeded the baseline after 1.41 Lyapunov time.

5.4 Mean Variance Estimation Deep Ensemble

The Mean Variance Estimation Deep Ensemble was trained by training $n_{run} = 5$ different neural networks on an architecture which output both a mean and a standard deviation. Similar to in the Mean Variance Estimation, the LSTM network overfitted and was not able to attain desirable results. On the other hand, the RC approach performed similarly to in the Deep Ensemble method, having a mean negative log likelihood value of 15.87 over the test horizon.



(a) LSTM Network



(b) RC Network

Figure 5.4: Deep Ensemble MVE results for LSTM and RC Networks. (Left) Negative Loglikelihood (Middle) Variance (Right) NRMSE

5.5 Summary of Methods

The mean negative log likelihood was calculated for each of the methods over 400 time-steps (≈ 6.56 Lyapunov time) and can be seen in Table 5.1.

Method	Deep Ensemble	MVE w/o Sampling	MVE w/ Sampling	MVE Deep Ensemble
LSTM	10.14	406.40	35.91	37.32
RC	15.87	4178810.66	29.71	16.18

Table 5.1: Summary of metrics for uncertainty quantification methods. Green represents the best, while red represents the worst

For the LSTM network, due to the overfitting when performing the Mean Variance Estimation method, the negative log-likelihood was observed to be significantly worse than when using Deep Ensembles. The Reservoir Computing approach performed slightly better, and the Mean Variance Estimation Deep Ensembles performed similarly to the Deep Ensembles. However, this signifies that the Mean Variance Estimation provided little information to the uncertainty quantification. A possible explanation to overfitting is the lack of data noise in the dataset. As the training and test data were generated from the same process, there was no noise present in

the data. Hence, the Mean Variance Estimation, which is targeted towards aleatoric uncertainty, was extremely certain of its predictions and underestimated its variance, resulting in extremely high negative log-likelihood values when the predictions were not exact.

One benefit of the Mean Variance Estimation method is to relatively quick estimation time required. In the case of the LSTM, the estimation and sampling only took 6.30 seconds, which is considerably faster than needing to train a full ensemble of models. Hence, it is useful to explore ways to improve the flaws of the Mean Variance Estimation to provide a comparable negative log-likelihood value.

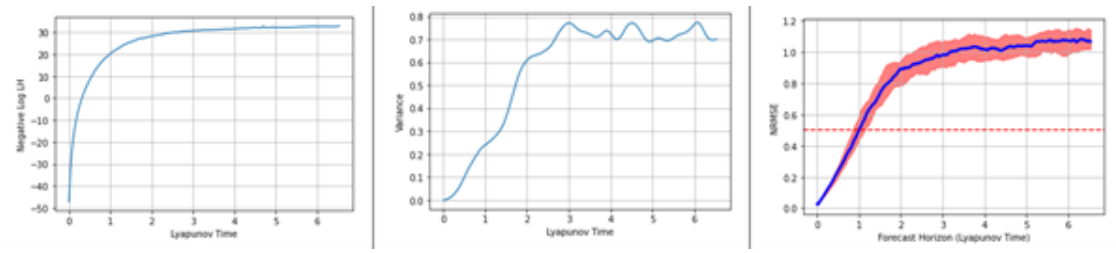
5.6 Adding Noise to Training Dataset

As explained earlier, the issue with the Mean Variance Estimation was possibly due to underestimation of the variance from a lack of noise in the training data. Hence, a further preliminary test was conducted to investigate the impact of noise on the training data. The training data was perturbed by adding values drawn from a normal distribution $X \sim N(0, 0.01)$ while the test data was left unadjusted. The RC network was then trained using the same uncertainty quantification methods and measured with the same metrics.

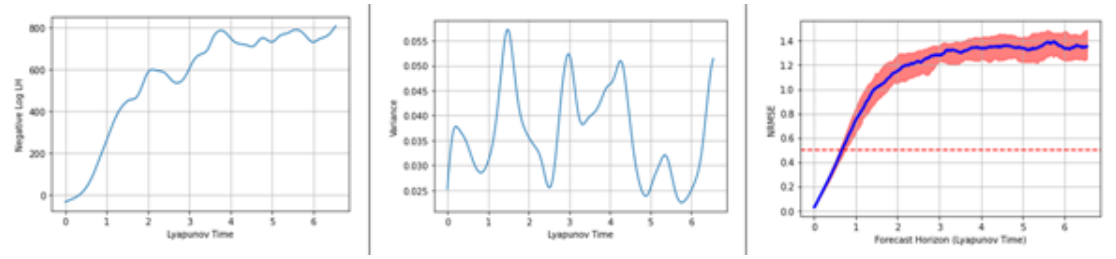
Method	Deep Ensemble	MVE w/o Sampling	MVE w/ Sampling	MVE Deep Ensemble
RC	26.21	563.27	31.12	26.44

Table 5.2: Mean Negative Loglikelihood of Uncertainty Quantification Methods in Noisy Setting. Green represents the best, while red represents the worst

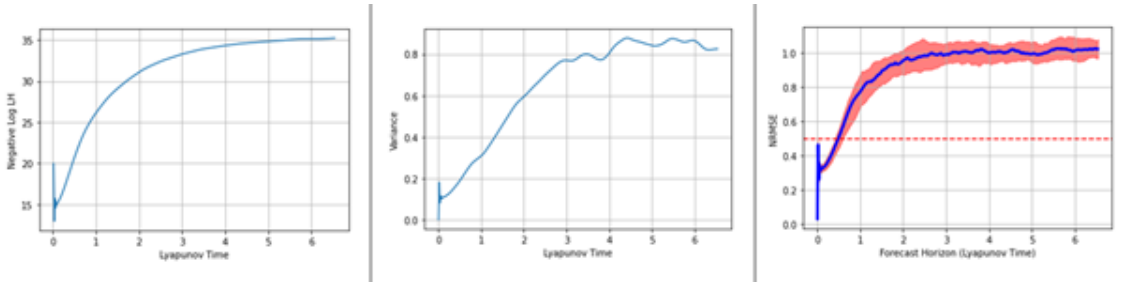
The results can be seen in Figure 5.5 and Table 5.2. Although the original Mean Variance Estimation method still performed badly, the variance plot (Figure 5.5b) shows that the algorithm was able to learn to slightly increase the variance as compared to the original training set (Figure 5.2a), resulting in a better uncertainty quantification than before. The Deep Ensemble and Mean Variance Estimation Deep Ensemble performed very similarly, achieving a negative likelihood difference of 0.9% from one another. However, the Mean Variance Estimation Deep Ensemble predicted a smaller variance, resulting in a smaller prediction interval, which may be investigated on in future work.



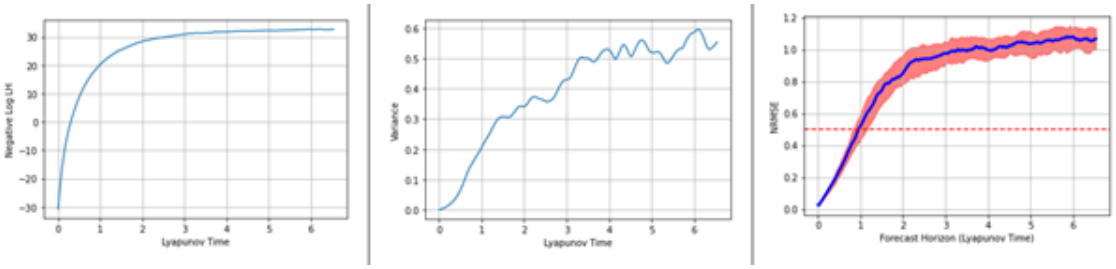
(a) Deep Ensemble



(b) MVE without Sampling



(c) MVE with Sampling



(d) Deep Ensemble MVE

Figure 5.5: Comparison of Uncertainty Quantification Methods in Noisy Training Data using RC. (Left) Negative Loglikelihood (Middle) Variance (Right) NRMSE

5.7 Discussion

Overall, the Deep Ensembles seem to be the most effective in estimating uncertainty due to a large portion of the uncertainty being from the model uncertainty. For the Mean Variance Estimation methods, the LSTM network is a lot more susceptible to the exponential increase in negative log likelihood due to its ability to overfit. To alleviate this problem, there are two areas that can be explored in future research for the Mean Variance Estimation to provide a better estimate of the data uncertainty present.

Firstly, methods can be taken to directly reduce the overfitting present in learning. One way as suggested by Lakshminarayanan et al. [27] is to impose a prior and perform maximum a posteriori (MAP) estimation instead.

Secondly, as seen from the example with data noise, irregularities in the data can help algorithms learn a more suitable variance estimate. Inspired by this, it may be worth considering to add new training trajectories that have a similar starting value at time t_0, but slightly perturbed values of $t_1, t_2, \ldots$ generated by sampling from a symmetric distribution. This will include innate uncertainty while the model is being trained so that the model will not be able to predict the training data with absolute certainty.

Chapter 6

Conclusion and Future Work

This thesis aimed to compare various deep learning methods used in time-series predictions. Using two dynamical system as benchmarks, experiments were conducted which compared the effectiveness of the methods at predicting chaotic processes for an extended period. LSTMs were found to provide the longest valid prediction horizon, while Koopman Autoencoders were not suitable for the prediction of chaotic systems. However, LSTMs cannot provide good predictions under low data settings, instead being outperformed by the Reservoir Computing approach. In situations with limited low data, Reservoir Computing was not only able to provide the best prediction horizon, but also could approximate good results using small networks. Hence, LSTM networks are useful under situations with plentiful data while Reservoir Computing networks can be used for small datasets where it can be used to perform quick and accurate short-term predictions.

Uncertainty quantifications methods were also explored to provide useful bounds so that users can have more actionable insights with their predictions. The goal of the uncertainty quantification methods was to find simple yet effective methods for estimating uncertainty in predictions. The thesis compared Deep Ensembles with Mean Variance Estimation methods and implemented some adaptations to improve the methods. Using Deep Ensembles provided the best performance in negative log-likelihood, while also having the added benefit of improving the prediction error.

Looking ahead, there are several trajectories that can be explored. Firstly, work can be applied onto real-world time-series data. Although dynamical systems are a good proxy for understanding and modelling systems in the world, real-world time-series data may have properties that better utilise some of the time-dependencies in the models discussed such as the hidden memory α or the impact of the spectral

radius in Reservoir Computing networks. Secondly, more rigorous experiments can be done to verify the relationship between the dimension or size of the data, and the performance of the various methods. This can help us determine the exact situations that some models may be more ideal for time-series analysis compared to others. Lastly, research can be done in innovating the Mean Variance Estimation method. As a quick and easily implementable method, we believe there are benefits to exploring how we can inject noise or regularisation into the training process to help the network learn a more representative variance.

Code and Data Availability

The code used for this thesis is available at `https://github.com/justlotw/Deep-Learning-Prediction-and-Uncertainty-Quantification-of-High-Dimensional-Time-Series-Data`. The creation of data and further instructions can be found on the README on the page. The code was primarily written using Python 3.8.12 and utilised the JAX framework [5].

Bibliography

[1] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, V. Makarenkov, and S. Nahavandi, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges", *Information Fusion*, vol. 76, pp. 243–297, Dec. 2021, ISSN: 1566-2535. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2021.05.008.

[2] E. A. Antonelo, C. A. Flesch, and F. Schmitz, "Reservoir computing for detection of steady state in performance tests of compressors", 2017. arXiv: 1706.00782 [cs.OH].

[3] O. Azencot, N. B. Erichson, V. Lin, and M. W. Mahoney, "Forecasting sequential data using consistent koopman autoencoders", 2020. arXiv: 2003.02236 [physics.comp-ph].

[4] H. V. Bitencourt and F. G. Guimarães, "High-dimensional multivariate time series forecasting in iot applications using embedding non-stationary fuzzy time series", 2021. arXiv: 2107.09785 [cs.LG].

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: Composable transformations of Python+NumPy programs", version 0.2.5, 2018. [Online]. Available: http://github.com/google/jax.

[6] S. Brunton. "Deep learning to discover coordinates for dynamics: Autoencoders & physics informed machine learning", Youtube. (2021), [Online]. Available: https://www.youtube.com/watch?v=KmQkDgu-Qp0.

[7] S. L. Brunton, M. Budišić, E. Kaiser, and J. N. Kutz, "Modern koopman theory for dynamical systems", 2021. arXiv: 2102.12086 [math.DS].

[8] R. Budhiraja, M. Kumar, M. K. Das, A. S. Bafila, and S. Singh, "A reservoir computing approach for forecasting and regenerating both dynamical and time-delay controlled financial system behavior", *PLOS ONE*, vol. 16, no. 2, pp. 1–24, Feb. 2021. [Online]. Available: https://doi.org/10.1371/journal.pone.0246737.

[9] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations", 2019. arXiv: 1904.02107 [stat.OT].

[10] D. Charte, F. Charte, S. García, M. J. del Jesus, and F. Herrera, "A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines", *Information Fusion*, vol. 44, pp. 78–96, Nov. 2018, ISSN: 1566-2535. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2017.12.007.

[11] A. Chattopadhyay, P. Hassanzadeh, and D. 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", *Nonlinear Processes in Geophysics*, vol. 27, no. 3, pp. 373–389, Jul. 2020, ISSN: 1607-7946. [Online]. Available: http://dx.doi.org/10.5194/npg-27-373-2020.

[12] R. A. Edson, J. E. Bunder, T. W. Mattner, and A. J. Roberts, "Lyapunov exponents of the kuramoto-sivashinsky pde", 2019. arXiv: 1902.09651 [math.DS].

[13] J. L. Elman, "Finding structure in time", *COGNITIVE SCIENCE*, vol. 14, no. 2, pp. 179–211, 1990.

[14] L. Frau, G. A. Susto, T. Barbariol, and E. Feltresi, "Uncertainty estimation for machine learning models in multiphase flow applications", *Informatics*, vol. 8, no. 3, 2021, ISSN: 2227-9709. [Online]. Available: https://www.mdpi.com/2227-9709/8/3/58.

[15] D. J. Gauthier, E. Bollt, A. Griffith, and W. A. S. Barbosa, "Next generation reservoir computing", *Nature Communications*, vol. 12, no. 1, Sep. 2021, ISSN: 2041-1723. [Online]. Available: http://dx.doi.org/10.1038/s41467-021-25801-2.

[16] M. the Great, "Solving numerically the 1d kuramoto-sivashinsky equation using spectral methods", Computational Science Stack Exchange, [Online:] https://scicomp.stackexchange.com/questions/37336/solving-numerically-the-1d-kuramoto-sivashinsky-equation-using-spectral-methods, 2021. [Online]. Available: https://scicomp.stackexchange.com/questions/37336/solving-numerically-the-1d-kuramoto-sivashinsky-equation-using-spectral-methods.

[17] T. Heskes, "Practical confidence and prediction intervals", in *Advances in Neural Information Processing Systems*, M. C. Mozer, M. Jordan, and T. Petsche, Eds., vol. 9, MIT Press, 1996. [Online]. Available: https://proceedings.neurips.cc/paper/1996/file/7940ab47468396569a906f75ff3f20ef-Paper.pdf.

[18] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.79.8.2554. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.79.8.2554.

[19] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods", *Machine Learning*, vol. 110, no. 3, pp. 457–506, Mar. 2021, ISSN: 1573-0565. [Online]. Available: http://dx.doi.org/10.1007/s10994-021-05946-3.

[20] J. K. Hunter, "Introduction to dynamical systems", 2011. [Online]. Available: https://www.math.ucdavis.edu/~hunter/m207/m207.pdf.

[21] H. Jaeger, "Echo state network", *Scholarpedia*, vol. 2, no. 9, p. 2330, 2007, revision #196567.

[22] H. Jaeger, "The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'", *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, Jan. 2001.

[23] A. Karimi and M. R. Paul, "Extensive chaos in the lorenz-96 model", *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 20, no. 4, p. 043 105, Dec. 2010, ISSN: 1089-7682. [Online]. Available: http://dx.doi.org/10.1063/1.3496397.

[24] J. Kerin and H. Engler, "On the lorenz '96 model and some generalizations", 2020. arXiv: 2005.07767 [math.DS].

[25] Y. Kuramoto and T. Tsuzuki, "On the Formation of Dissipative Structures in Reaction-Diffusion Systems: Reductive Perturbation Approach", *Progress of Theoretical Physics*, vol. 54, no. 3, pp. 687–699, Sep. 1975, ISSN: 0033-068X. eprint: https://academic.oup.com/ptp/article-pdf/54/3/687/5402406/54-3-687.pdf. [Online]. Available: https://doi.org/10.1143/PTP.54.687.

[26] Y. Lai, Y. Shi, Y. Han, Y. Shao, M. Qi, and B. Li, "Exploring uncertainty in deep learning for construction of prediction intervals", 2021. arXiv: 2104.12953 [cs.LG].

[27] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles", 2017. arXiv: 1612.01474 [stat.ML].

[28] G. C. Layek, "Continuous dynamical systems", in *An Introduction to Dynamical Systems and Chaos*. New Delhi: Springer India, 2015, pp. 1–35, ISBN: 978-81-322-2556-0. [Online]. Available: https://doi.org/10.1007/978-81-322-2556-0_1.

[29] D. Leslie, "Understanding artificial intelligence ethics and safety", *CoRR*, vol. abs/1906.05684, 2019. arXiv: 1906.05684. [Online]. Available: http://arxiv.org/abs/1906.05684.

[30] E. Lorenz, "Predictability: A problem partly solved", in *Seminar on Predictability, 4-8 September 1995*, ECMWF, vol. 1, Shinfield Park, Reading: ECMWF, 1995, pp. 1–18. [Online]. Available: https://www.ecmwf.int/node/10829.

[31] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training", *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009, ISSN: 1574-0137. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013709000173.

[32] A. A. Mishra, A. Edelen, A. Hanuka, and C. Mayes, "Uncertainty quantification for deep learning in particle accelerator applications", *Phys. Rev. Accel. Beams*, vol. 24, p. 114 601, 11 Nov. 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevAccelBeams.24.114601.

[33] A. Murad, F. A. Kraemer, K. Bach, and G. Taylor, "Probabilistic deep learning to quantify uncertainty in air quality forecasting", *Sensors*, vol. 21, no. 23, p. 8009, Nov. 2021, ISSN: 1424-8220. [Online]. Available: http://dx.doi.org/10.3390/s21238009.

[34] D. Nix and A. Weigend, "Estimating the mean and variance of the target probability distribution", in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, vol. 1, 1994, 55–60 vol.1.

[35] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, "Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach", *Phys. Rev. Lett.*, vol. 120, p. 024 102, 2 Jan. 2018. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.120.024102.

[36] T. Pearce, M. Zaki, A. Brintrup, and A. Neely, "High-quality prediction intervals for deep learning: A distribution-free, ensembled approach", 2018. arXiv: 1802.07167 [stat.ML].

[37] R. Sen, H.-F. Yu, and I. Dhillon, "Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting", 2019. arXiv: 1905.03806 [stat.ML].

[38] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network", *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, Mar. 2020, ISSN: 0167-2789. [Online]. Available: http://dx.doi.org/10.1016/j.physd.2019.132306.

[39] G. Sivashinsky, "Nonlinear analysis of hydrodynamic instability in laminar flames—i. derivation of basic equations", *Acta Astronautica*, vol. 4, no. 11, pp. 1177–1206, 1977, ISSN: 0094-5765. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0094576577900960.

[40] D. Su, Y. Y. Ting, and J. Ansel, "Tight prediction intervals using expanded interval minimization", 2018. arXiv: 1806.11222 [cs.LG].

[41] C. Tang and Y. Shi, "Forecasting high-dimensional financial functional time series: An application to constituent stocks in dow jones index", *Journal of Risk and Financial Management*, vol. 14, no. 8, 2021, ISSN: 1911-8074. [Online]. Available: https://www.mdpi.com/1911-8074/14/8/343.

[42] J. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, and A. Troncoso, "Deep learning for time series forecasting: A survey", *Big Data*, vol. 9, Dec. 2020.

[43] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos, "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks", *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 474, no. 2213, p. 20 170 844, May 2018, ISSN: 1471-2946. [Online]. Available: http://dx.doi.org/10.1098/rspa.2017.0844.

[44] P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics", 2020. arXiv: 1910.05266 [eess.SP].

[45] P. Werbos, "Backpropagation through time: What it does and how to do it", *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[46] Q. Wu, E. Fokoue, and D. Kudithipudi, "On the statistical challenges of echo state networks and some potential remedies", 2018. arXiv: 1802.07369 [stat.ML].

[47] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning", 2021. arXiv: 2106.11342 [cs.LG].

[48] L. Zhu and N. Laptev, "Deep and confident prediction for time series at uber", *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, Nov. 2017. [Online]. Available: http://dx.doi.org/10.1109/ICDMW.2017.19.

Appendix A

List of Experimented Model Hyperparameters

The hyperparameters experimented on each model are listed in the following tables: RNN & LSTM [Table A.1], RC [Table A.2], Koopman Autoencoder [Table A.3, Table A.4].

Hyperparameter	Values
Hidden state size	[100, 500, 1000, 1500]
Sequence length used in training, ζ_1	[1, 4, 8]
Time-steps forward used in training loss, ζ_2	[4, 8, 16]
Testing warm-up (steps)	2000
Batch size	128

Table A.1: Model Hyperparameters for RNN and LSTM

Hyperparameter	Values
Reservoir size	[3000, 6000, 9000, 12000, 15000]
Tikhonov regularization	[1e-2, 1e-4, 1e-6]
Spectral radius	[0.1, 0.3, 0.5, 0.7]
Connectivity	[4, 8]
Scale of values for matrix W, ω	1
Hidden state memory, α	1
Training transient period (steps)	200
Testing warm-up (steps)	2000
Batch size	200

Table A.2: Model Hyperparameters for RC

Hyperparameter	Values
Neural network architecture	(32, 16, 16, 8) (16, 8, 8, 4) (64, 128, 128, 256) [(32, 24, 24, 16)] (36, 32, 32, 24) (40, 40, 40, 40)
Time-steps forward used in training loss, l	[4, 8, 16, 24, 32]
Loss coefficients ($\lambda_{recon}, \lambda_{fwd}, \lambda_{bwd}, \lambda_{con}$)	(1, 1, 0.1, 0.01) (1, 2, 0.1, 0.01) [(1, 1, 0, 0)] (1, 0.5, 0.5, 0.01) (1, 1, 1, 0.01)
Batch size	128

Table A.3: Model Hyperparameters for Koopman Autoencoder (Lorenz)

Hyperparameter	Values
Neural network architecture	(128, 64, 32, 16) (100, 50, 24, 12) (160, 128, 64, 32) [(200, 160, 128, 96)] (200, 176, 160, 144) (240, 240, 240, 240)
Time-steps forward used in training loss, l	[4, 8, 16, 24, 32]
Loss coefficients ($\lambda_{recon}, \lambda_{fwd}, \lambda_{bwd}, \lambda_{con}$)	(1, 1, 0.1, 0.01) (1, 2, 0.1, 0.01) [(1, 1, 0, 0)] (1, 0.5, 0.5, 0.01) (1, 1, 1, 0.01)
Batch size	128

Table A.4: Model Hyperparameters for Koopman Autoencoder (KS Equation)

As described in the training process in Chapter 2, each model was tuned for the various hyperparameters and the optimal model was used for comparison. The list of optimal parameters for each of the datasets are listed in Table A.5 (RNN), Table A.6 (LSTM), Table A.7 (RC) and Table A.8 (Koopman Autoencoder).

Hyperparameter	Lorenz-96	KS Equation	Lorenz-96(S)
Hidden state size	500	500	1000
Sequence length used in training, ζ_1	8	1	1
Time-steps forward used in training loss, ζ_2	16	16	8

Table A.5: Optimal Hyperparameters used for RNN

Hyperparameter	Lorenz-96	KS Equation	Lorenz-96(S)
Hidden state size	500	500	500
Sequence length used in training, ζ_1	4	4	4
Time-steps forward used in training loss, ζ_2	4	8	8

Table A.6: Optimal Hyperparameters used for LSTM

Hyperparameter	Lorenz-96	KS Equation	Lorenz-96(S)
Reservoir size	12000	15000	15000
Tikhonov regularization	1e-6	1e-2	1e-4
Spectral radius	0.1	0.3	0.5
Connectivity	4	4	4

Table A.7: Optimal Hyperparameters used for RC

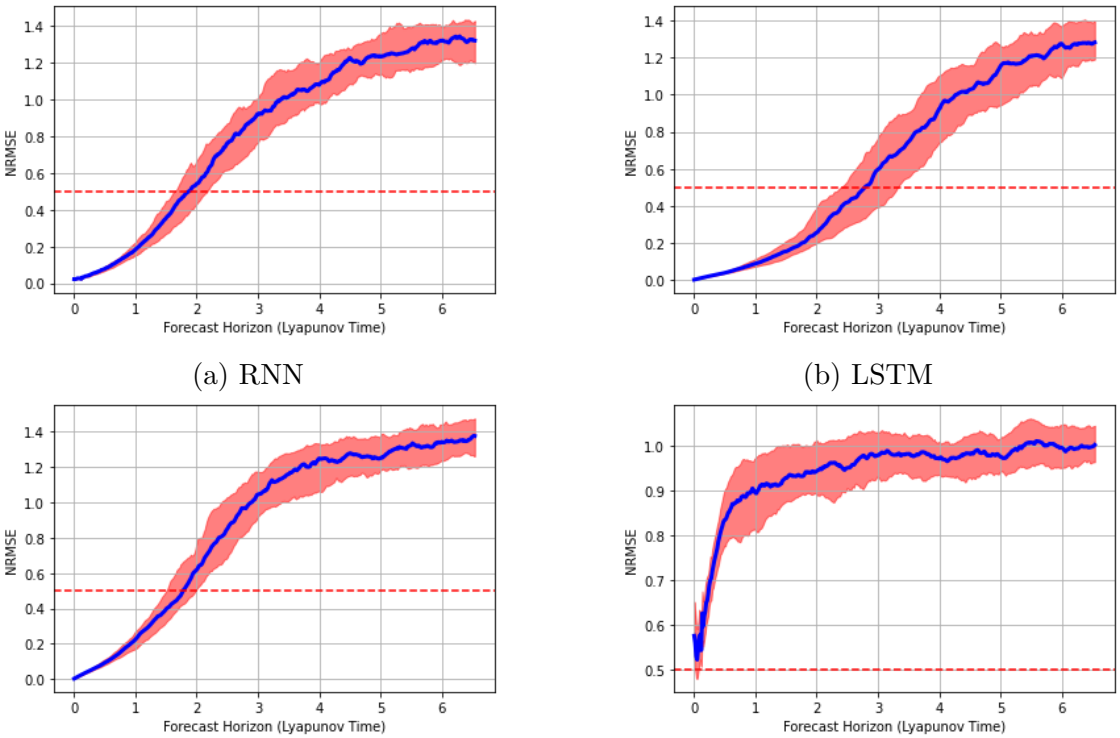
Hyperparameter	Lorenz-96	KS Equation	Lorenz-96(S)
Neural network architecture	[40, 40, 40, 40]	[240, 240, 240, 240]	[64, 128, 128, 256]
Time-steps forward used in training loss	8	16	16
Loss coefficients ($\lambda_{recon}, \lambda_{fwd}, \lambda_{bwd}, \lambda_{con}$)	[1, 1, .1, .01]	[1, 2, .1, .01]	[1, 1, .1, .01]

Table A.8: Optimal Hyperparameters used for Koopman Autoencoder

Appendix B

Prediction Horizon for Models

For each experiment, the results were evaluated against a test set of 100 different trajectories and the NRMSE of each trajectory was calculated. To provide an estimate of the range of the predictions, the 25th, 50th and 75th percentile errors were extracted and plotted. In each of the graphs (Figure B.1, Figure B.2, Figure B.3) below, the blue line represents the trajectory with the median NRMSE, while the red area represents the bounds of the 25th to 75th percentile trajectories NRMSE.



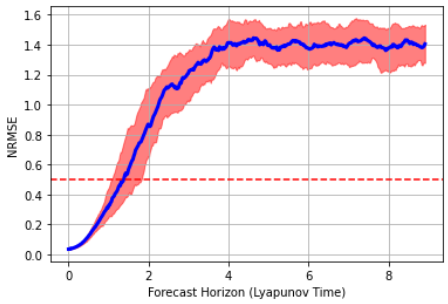
(a) RNN

(b) LSTM

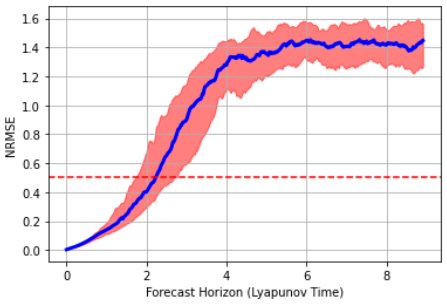
(c) RC

(d) Koopman Autoencoder

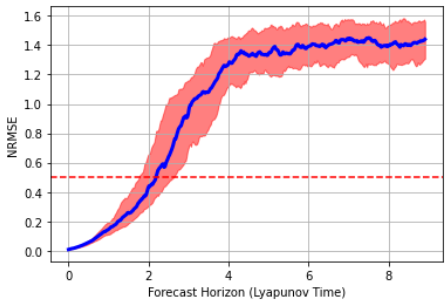
Figure B.1: Prediction Horizon of Lorenz-96 System



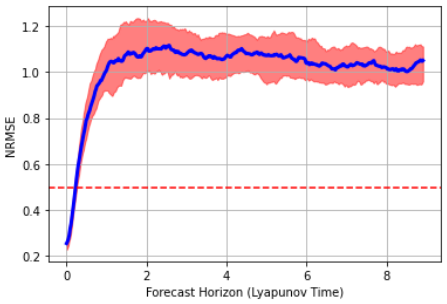
(a) RNN



(b) LSTM

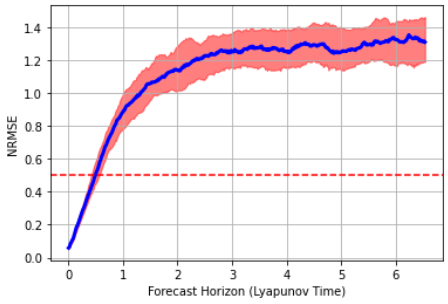


(c) RC

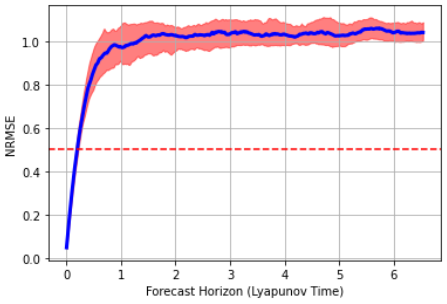


(d) Koopman Autoencoder

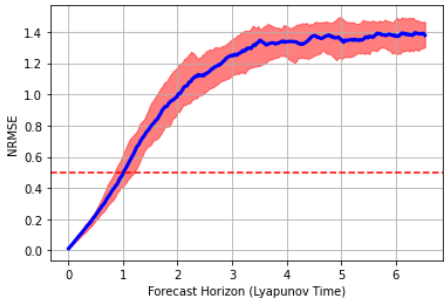
Figure B.2: Prediction Horizon of KS System



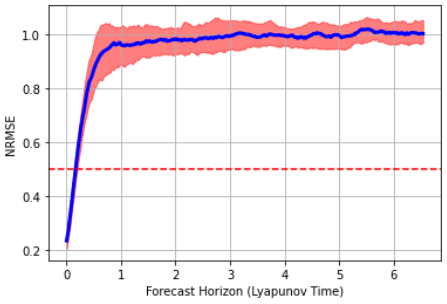
(a) RNN



(b) LSTM



(c) RC



(d) Koopman Autoencoder

Figure B.3: Prediction Horizon of Small Lorenz-96 System