

# State estimation using machine learning

by

Avneet Kaur

A thesis  
presented to the University of Waterloo  
in fulfillment of the  
thesis requirement for the degree of  
Doctor of Philosophy  
in  
Applied Mathematics

Waterloo, Ontario, Canada, 2025

© Avneet Kaur 2025

## Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Wei Kang  
Professor, Dept. of Applied Mathematics, Naval Postgraduate School

Supervisor(s): Kirsten Morris  
Professor, Dept. of Applied Mathematics, University of Waterloo

Internal Member: Giang Tran  
Professor, Dept. of Applied Mathematics, University of Waterloo

Internal Member: Jun Liu  
Professor, Dept. of Applied Mathematics, University of Waterloo

Internal-External Member: Stephen L. Smith  
Professor, Dept. of Electrical and Computer Engineering,  
University of Waterloo

## **Author's Declaration**

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

## Statement of contributions

I, Avneet Kaur, am the sole author of Chapters 1, 2, and 6, which were written under the supervision of Dr. Kirsten Morris and were not written for publication. The research presented in Chapters 3, 4 and 5 have been or will be submitted for publication in peer-reviewed journals. The contributions of the authors to these chapters are described below:

**Chapter 3:** The research outcomes presented in this chapter are based on a collaboration between Avneet Kaur, Ruikun Zhou, Prof. Jun Liu, and Prof. Kirsten Morris. The manuscript will be submitted for publication. Avneet Kaur and Prof. Kirsten Morris formulated the problem. Avneet Kaur worked on numerical simulations related to the comparisons of [Jordan recurrent neural network \(JRN\)](#), [extended Kalman filter \(EKF\)](#) and [unscented Kalman filter \(UKF\)](#) for several examples. The idea for input to state stability of error dynamics was jointly formulated by all authors and implemented by Ruikun Zhou and Prof. Jun Liu. The manuscript was jointly written by Avneet Kaur and Ruikun Zhou. It was edited by all authors.

**Chapter 4:** Avneet Kaur is the sole author of the results presented in this chapter under the supervision of Prof. Kirsten Morris. The manuscript corresponding to this research was written by Avneet Kaur and edited by both authors. The manuscript has been submitted to *International Journal of Control* and is conditionally accepted[64].

**Chapter 5:** The results presented in this chapter were obtained by Avneet Kaur under the supervision of Prof. Kirsten Morris. While the simulations verifying the robustness of the Burgers' equation to variations in initial conditions and noise dynamics were executed by Yuhao Chen, the underlying code and implementation were developed by Avneet Kaur. The manuscript corresponding to this research will be submitted to a journal for publication.

## Abstract

State estimation refers to determining the states of a dynamical system that evolves under disturbances, based on noisy measurements, partially known or unknown initial condition, and a known system model. [JRN](#)s have a structure that mimics that of a dynamical system and are thus attractive for estimator design. We show that a [JRN](#) performs better than an [EKF](#) and [UKF](#) for several examples. We also provide a input-to-state stability analysis of the error dynamics of [JRN](#)s. The stability of the error dynamics of several examples is shown.

We then extend the Jordan structure to long-short-term memory networks to obtain a [Jordan long short-term memory network \(JLSTM\)](#) which, as we show in several examples, is comparatively more robust to changes in initial conditions and noise and performs better than a [EKF](#) and [particle filter \(PF\)](#). It also trains faster than an [Elman long short-term memory network \(ELSTM\)](#) for state estimation when trained to achieve a similar normalized [mean square error \(MSE\)](#).

We also compare a shallow and deep [JLSTM](#) and observe that they perform almost similarly in terms of average error across time-steps and [MSE](#) but the deep [JLSTM](#) takes longer to train due to more layers.

We also train a [JLSTM](#) with a modified maximum likelihood equivalent loss function([Jordan long short-term memory network with modified maximum likelihood loss function \(JLSTM-ML\)](#)). We observe that for Gaussian initial conditions and disturbances, the average error at each time step is best for estimates of [JLSTM-ML](#). It is also the most robust to changes in initial conditions and disturbances in the systems considered. The measures, time taken to train, time taken to test, mean squared error, and average error at each time-step were used for comparison for various networks.

We discretized the following systems to use as examples in data generation, training, and testing: mass-spring system, down pendulum, reversed Van der Pol oscillator, Galerkin approximation of Burger's partial differential equation and Kuramoto-Sivashinsky partial differential equation.

## Acknowledgements

I express my deepest gratitude to my supervisor, Prof. Kirsten Morris, for her unwavering support throughout my degree. Her guidance and encouragement made this journey possible. As an immigrant, adjusting to life in Canada was challenging, but her mentorship significantly eased the transition. Her genuine care for both my academic progress and personal well-being is a rare and invaluable quality in a supervisor.

I am thankful to all the members of my thesis committee: Prof. Kirsten Morris, Prof. Jun Liu, Prof. Giang Tran, Prof. Stephen Smith and Prof. Wei Kang, for dedicating their time to review my work and to evaluate my defense.

I also thank Prof. Giang Tran and Prof. Jun Liu for their kindness in stepping in as supervisors during times when Prof. Kirsten Morris was unavailable due to unforeseen circumstances.

I am sincerely thankful to my husband, Anshul Chopra, for his constant support and care during this journey. His presence and love has been a pillar of strength through every challenge.

I am deeply grateful to my family for their unwavering love and support throughout my life. My grandparents, parents, and siblings have always been my foundation. I extend heartfelt thanks to my parents-in-law, Raman and Neeru Chopra, for their loving care of my son during the times I needed to concentrate on my studies. I am especially grateful to my parents, Arvinder Singh and Sandeep Kaur, whose unwavering support and guidance have been a constant source of strength and inspiration throughout my life. To my sisters Sumeet Kaur and Jasmine Kaur, thank you for your constant belief in me and for being a source of emotional strength. I am forever grateful to my grandmother, Charanjit Kaur, whose unwavering belief that teaching was my true calling inspired me deeply. My grandfather, Gurbax Singh, instilled in me a love for mathematics from an early age, setting the foundation for my academic journey. I also extend my thanks to my brother-in-law, Shubham Chopra and sister-in-law, Sakshi Sharma for their continuous support and encouragement. A special thanks to my extended family for their love and support throughout my degree.

I also extend my appreciation to the Department of Mathematics administrative team for their prompt and helpful support throughout my studies.

I am also grateful to have encouraging members in Prof. Kirsten Morris' research group. I would like to thank them all for their constructive feedback on several presentations during my degree.

I am truly grateful to have had Rhythm Kaul and Muskan Kakkar by my side - friends who encouraged me, uplifted my spirits, and helped me stay strong throughout this journey. I would also like to thank all my other friends who provided me with support and encouragement whenever possible.

Lastly, I am thankful to God for the gift of life, a loving family, and the joy of becoming a mother. My son, Anav, has brought immense happiness and strength to my life, reminding me each day why this journey is worth it.

## **Dedication**

This is dedicated to those I love.

# Table of Contents

Examining Committee	ii
Author's Declaration	iii
Statement of Contributions	iv
Abstract	v
Acknowledgements	vi
Dedication	viii
List of Figures	xi
List of Tables	xiii
List of Abbreviations	xiv
<b>1 Introduction</b>	<b>1</b>
<b>2 Background material</b>	<b>4</b>
2.1 Traditional state estimation . . . . .	6
2.2 Data-driven state estimation . . . . .	12
2.3 Stability analysis for the error dynamics . . . . .	17

<b>3</b>	<b>Stability of Jordan recurrent network</b>	<b>22</b>
3.1	Implementation and numerical results . . . . .	28
3.2	Conclusions and limitations . . . . .	33
<b>4</b>	<b>State estimation using Jordan long short-term memory network</b>	<b>35</b>
4.1	Implementation . . . . .	43
4.2	Results, conclusions and limitations . . . . .	49
<b>5</b>	<b>Extensions of Jordan long short-term memory network based estimators</b>	<b>53</b>
5.1	Implementation and numerical examples . . . . .	57
5.2	Results and conclusions . . . . .	61
<b>6</b>	<b>Conclusions and future research</b>	<b>67</b>
6.1	Future directions . . . . .	68
	<b>References</b>	<b>70</b>

# List of Figures

3.1	Jordan recurrent network (JRN) for state estimation showing output-to-hidden recurrent connections. . . . .	23
3.2	Figures 3.2a, 3.2b and 3.2c show the training and validation values for JRN based state estimation. Figures 3.2d 3.2e and 3.2f show average errors at each time-step for JRN, EKF and UKF state estimates. . . . .	30
3.3	Learned ISS Lyapunov function for a down pendulum. . . . .	31
4.1	The structure of Elman recurrent neural network (ERN) for state estimation.	36
4.2	The structure of ELSTM for state estimation. . . . .	41
4.3	The structure of JLSTM for state estimation. . . . .	42
4.4	Average errors over 10 test sequences for 50 seconds of 10 connected springs with a noisy Gaussian initial condition for Kalman filter (KF), JLSTM and ELSTM based state estimates. . . . .	46
4.5	Average errors over 10 test sequences for 50 seconds of 10 connected springs with a noisy Gaussian initial condition outside the training region for KF, JLSTM and ELSTM based state estimates. . . . .	47
4.6	Average errors over 10 test sequences for 40 seconds of down pendulum with a noisy Gaussian initial condition for EKF, JLSTM and ELSTM based state estimates. . . . .	48
4.7	Average errors over 10 test sequences for 40 seconds of down pendulum with a noisy Gaussian initial condition outside the training region for EKF, JLSTM and ELSTM based state estimates. . . . .	48
4.8	This figure compares the performance of EKF, JLSTM and ELSTM using the average errors over 10 test sequences for 30 seconds of reversed Van der Pol oscillator with a noisy Gaussian initial condition. . . . .	49

4.9	This figure compares the performance of EKF, JLSTM and ELSTM using the average errors over 10 test sequences for the first 30 seconds of reversed Van der Pol oscillator with a noisy Gaussian initial condition outside the training region. . . . .	50
5.1	Average estimation error values per time-step for test sequences in experiment 1. . . . .	62
5.2	Training and validation loss graphs for all models in experiment 1. . . . .	63
5.3	$c_R$ (log scale) vs. Mean square error values(in dB) using JLSTM, JLSTM-ML and EKF for different values of $c_Q$ . . . . .	65

# List of Tables

3.1	Training hyperparameters . . . . .	29
3.2	Parameters . . . . .	31
3.3	Root mean square error (root MSE) for EKF, UKF and JRN . . . . .	32
3.4	Best epoch and training time for JRN . . . . .	33
4.1	Normalised MSE for EKF, JLSTM and ELSTM for testing dataset 1 . . . .	50
4.2	Training time(in seconds) for JLSTM and ELSTM . . . . .	50
4.3	Testing time(in seconds) for EKF, JLSTM and ELSTM combined for both testing datasets . . . . .	51
4.4	Normalised MSE for EKF, JLSTM and ELSTM for testing dataset 2 . . . .	51
5.1	Simulation parameters for each numerical example(Experiment 1) . . . . .	61
5.2	MSE for EKF, PF, JLSTM, deep JLSTM and JLSTM-ML(Dataset 1) . . . .	63
5.3	MSE for EKF, PF, JLSTM, deep JLSTM and JLSTM-ML(Dataset 2) . . . .	64
5.4	Training time(in seconds) for JLSTM, deep JLSTM and JLSTM-ML . . . .	64
5.5	Time taken to run test data(in seconds) for EKF, PF, JLSTM, deep JLSTM and JLSTM-ML (Dataset 1 and 2) . . . . .	64

# List of Abbreviations

- EKF** extended Kalman filter [iv](#), [v](#), [xi–xiii](#), [2](#), [3](#), [9](#), [12](#), [13](#), [15](#), [18](#), [22](#), [29](#), [30](#), [32](#), [33](#), [40](#), [44](#), [47–51](#), [57](#), [58](#), [61](#), [63–67](#)
- ELSTM** Elman long short-term memory network [v](#), [xi–xiii](#), [2](#), [3](#), [5](#), [13](#), [14](#), [36](#), [40–51](#), [53](#), [67](#)
- ERN** Elman recurrent neural network [xi](#), [3](#), [13](#), [22](#), [23](#), [36](#), [37](#)
- JLSTM** Jordan long short-term memory network [v](#), [xi–xiii](#), [2](#), [3](#), [36](#), [41–51](#), [53–55](#), [57](#), [58](#), [61](#), [63–69](#)
- JLSTM-ML** Jordan long short-term memory network with modified maximum likelihood loss function [v](#), [xii](#), [xiii](#), [3](#), [53](#), [57](#), [58](#), [63–66](#), [68](#)
- JRN** Jordan recurrent neural network [iv](#), [v](#), [xi](#), [xiii](#), [2](#), [3](#), [13](#), [22](#), [23](#), [28–30](#), [32–34](#), [36–40](#), [42](#), [67](#), [68](#)
- KF** Kalman filter [xi](#), [2](#), [11](#), [14](#), [15](#), [18](#), [20](#), [29](#), [32](#), [33](#), [40](#), [44–47](#), [49](#), [51](#)
- MSE** mean square error [v](#), [xiii](#), [3](#), [9](#), [20](#), [29](#), [32](#), [33](#), [44](#), [45](#), [49–51](#), [58](#), [63–66](#), [68](#)
- PF** particle filter [v](#), [xiii](#), [2](#), [10](#), [18](#), [19](#), [57](#), [58](#), [61](#), [63–67](#)
- UKF** unscented Kalman filter [iv](#), [v](#), [xi](#), [xiii](#), [2](#), [3](#), [9](#), [15](#), [18](#), [22](#), [29](#), [30](#), [32](#), [33](#), [67](#)

# Chapter 1

## Introduction

Several applications, ranging from robots delivering food to heart beat monitors in a hospital, require understanding of internal states without directly observing or measuring all of them. This is where state estimation plays an important role. When driving a car, one cannot see the exact fuel flow or engine temperature at every moment, but one can read the speed, fuel gauge, and maybe some dashboard lights. Based on these measurements and a model for the dynamics, we guess (or estimate) the performance of the car's engine. This is the essence of state estimation: inferring the full internal condition of the system using limited and/or indirect information.

The process of estimating the state of a dynamical system in the presence of disturbances using incomplete and noisy measurements is highly relevant in the fields of modern engineering and scientific inquiry. This process is called state estimation and underpins the functionality of autonomous vehicles navigating complex environments, ensures the stability of vast power networks, optimizes industrial processes, and informs critical decisions in diverse fields. The internal state of a system comprises the minimal set of variables required to fully describe its behavior at any given time, allowing for the prediction of future states given current inputs. However, measurement of some or all state variables is often noisy, impractical, impossible, or prohibitively expensive, necessitating the development of sophisticated algorithms to infer unobservable quantities from available data.

State estimation is crucial in innumerable applications. Autonomous vehicles are becoming immensely popular, but their failure rates are also high as they rely on sensors (such as cameras or LIDAR) to estimate their position, velocity, and surroundings. These readings must be used accurately to estimate the true states. Another powerful application of state estimation is in power grids. Power grids estimate the voltage and load across large

networks in real time. Another growing application is in robots. We see more and more tasks being automated using robots with each passing day. Robots estimate their position and environment to move safely and efficiently. One of the most essential applications is in the health industry, where devices estimate internal body conditions (such as blood glucose) from external signals. Without accurate state estimation, these systems would either operate blindly or become overly cautious and inefficient. Worse, in safety-critical applications - such as health, aircraft control, or nuclear power plants - inaccurate estimates could lead to catastrophic failure.

With powerful computing resources and large datasets becoming more accessible, state estimation research has more recently propelled machine learning paradigms, offering alternatives that can learn intricate system dynamics and noise characteristics directly from measurements. The evolution of state estimation techniques has aligned well with advances in computational power, sensor technology, and theoretical understanding of stochastic processes. Recurrent neural networks in machine learning have become an important tool for state estimation. Their similarity to a dynamical system is one of the main reasons for their popularity. Concurrently, the choice of an appropriate loss function is of utmost importance, as it dictates how estimation errors are penalized and, consequently, affects the statistical properties of the resulting state estimate. Establishing the stability and convergence of these methods is paramount.

The main objective of this research is to develop machine learning-based state estimation methods that integrate the model with the data. We approached this goal by first proving input to state stability of the error dynamics formed using a simpler architecture called [JRN](#)[55]. As our second objective, we then extended its architecture to form a [JLSTM](#)- a combination of [JRN](#) and [ELSTM](#) and compare their training time for several examples. Our third objective was to compare shallow and deep [JLSTMs](#) and compare their efficiency for high-dimensional complex systems. Finally, we used a modified maximum likelihood cost function as a loss function to train a [JLSTM](#) and compare it with other architectures. We then compared whether the newly introduced estimators are more robust to random initial conditions and varying noise levels than traditional [EKF](#). For several experiments, we provide comparisons with [KF](#), [EKF](#), [UKF](#) and [PF](#) where necessary. We use discretized versions of the following systems for comparison: connected springs, down pendulum, reversed Van der Pol oscillator, Galerkin approximation of Burgers and Kuramoto Sivashinsky equation. The last 2 systems were considered to be of order 11 and 41 which are higher than previously done in literature.

The outline of the thesis is given below.

**Chapter 2** We provide a review of the literature on state estimation and recall some

important definitions and concepts that are key to understanding the contents of the thesis.

**Chapter 3** We present an input to state stability verification for the error dynamics when using [JRN](#) for state estimation by training it on data simulated using known information of the model. We also compare the performance of [EKF](#), [UKF](#), and [JRN](#) for several examples. The contents of this chapter have been taken, with minor modifications from the article:

**Kaur, A.**, Zhou, R., Liu, J., & Morris, K. (2025). Stability of Jordan Recurrent Neural Network Estimator. arXiv preprint arXiv:2502.04551. [*To be submitted to Asian Journal of Control*]

**Chapter 4** We extend the universal approximation theorem for [ERN](#) to [JRN](#) for state estimation and present an extension of the Jordan structure to long short-term memory networks. We compare the training time for [JLSTM](#) and [ELSTM](#). We compare the normalized [MSE](#) and average error over time-steps for [EKF](#), [ELSTM](#) and [JLSTM](#) for several examples. The contents of this chapter have been taken, with minor modifications from the article:

**Kaur, A.**, & Morris, K. (2025). State estimator design using Jordan-based long short-term memory networks. arXiv preprint arXiv:2502.04518. [*Conditionally accepted to International Journal of Control*]

**Chapter 5** We train a [JLSTM](#) using a modified maximum likelihood cost function ([JLSTM-ML](#)) as a loss function and compare its performance with shallow and deep [JLSTM](#) and [EKF](#). We compare the robustness to changes in noise and random non-Gaussian initial conditions for [JLSTM](#), [JLSTM-ML](#) and [EKF](#). The most robust in terms of [MSE](#) is concluded to be [JLSTM](#). The contents of this chapter have been taken, with modifications from the article:

**Kaur, A.**, & Morris, K. (2025). Extensions of Jordan-based long short-term memory networks for state estimation. arXiv preprint arXiv:2502.04518. [*To be submitted to Automatica*]

**Chapter 6** We conclude the thesis with closing remarks and a discussion on possible future work and open problems corresponding to the research presented.

# Chapter 2

## Background material

This work considers noisy discrete-time state-space system

$$\begin{aligned}x^{(t+1)} &= f(x^{(t)}, u^{(t+1)}) + \omega^{(t+1)}, & \omega^{(t+1)} &\sim \mathcal{N}(0, Q) \\y^{(t+1)} &= h(x^{(t+1)}) + \nu^{(t+1)}, & \nu^{(t+1)} &\sim \mathcal{N}(0, R) \\x^{(0)} &\sim \mathcal{N}(\hat{x}_0, P_0)\end{aligned}\tag{2.1}$$

where  $x^{(t+1)} \in \mathcal{X} \subseteq \mathbb{R}^n$ ,  $n \in \mathbb{N}$  is the state vector of the system at time-step  $t + 1$ ,  $y^{(t+1)} \in \mathcal{Y} \subseteq \mathbb{R}^m$ ,  $m \in \mathbb{N}$  is the measurement vector at time-step  $t + 1$ ,  $u^{(t)} \in \mathcal{U} \subseteq \mathbb{R}^k$  is the control input vector at time-step  $t$ ,  $\omega^{(t+1)}$  is the process noise vector at time-step  $t + 1$  with zero mean and covariance  $Q$ ,  $\nu^{(t+1)}$  is the Gaussian measurement noise vector at time-step  $t + 1$  with zero mean and covariance  $R$ ,  $x^{(0)}$  is the true Gaussian initial condition with mean  $\hat{x}^0$  and covariance  $P_0$ . The functions  $f$  and  $h$  are known and continuously differentiable. Thus we want to estimate the state  $x^{(t+1)}$  based on new measurement  $y^{(t+1)}$ , control input  $u^{(t)}$ , previous state estimate  $\hat{x}^{(t)}$ , and some or no information about the initial condition  $x^{(0)}$ .

The aim of state estimation is to develop an estimator that is in itself a dynamical system [69, 91], for example,

$$\hat{x}^{(t+1)} = f(\hat{x}^{(t)}) + F(y^{(t)} - h(\hat{x}^{(t)})).\tag{2.2}$$

Here  $F$  is the filter gain and  $y^{(t)} - h(\hat{x}^{(t)})$  is called innovation term that considers the difference between the true and predicted measurement at time-step  $t$ . Solving the estimator system (2.2) may not seem difficult, but in practical applications this is a challenging task due to various reasons. The complexity of high-order systems, the availability of a finite number of measurements, inaccurate measurements, incomplete information about

the initial state, and errors in the model make it difficult to estimate the state of the system.

The early methods of state estimation relied heavily on linear system theory and statistical approximations, giving rise to foundational algorithms like the Kalman filter in 1960s[59]. The Kalman filter is, in fact, an optimal filter for state estimation of linear systems with Gaussian noises[59, 121]. For complex and nonlinear systems, extended and unscented variants of the Kalman filter emerged [51, 57, 56]. However, with the emergence of deep learning, neural networks—particularly recurrent structures—have shown great promise.

While feedforward neural networks[8] process inputs independently and are limited in handling temporal dependencies, recurrent neural networks introduce a form of memory by connecting hidden states across time steps. This structure naturally aligns with dynamical systems, where the future state depends on the current state and input. However, vanilla recurrent neural networks struggle with vanishing or exploding gradients, especially over long sequences, making training difficult and slow [84].

To make training faster and avoid vanishing or exploding gradients, gates were introduced in a recurrent neural network leading to the development of long short-term memory networks by Elman (ELSTM) [111] and gated recurrent units. The gated mechanism of these architectures helps retain information over longer horizons, making them suitable for sequence modeling tasks like state estimation in nonlinear dynamical systems. Recent work has demonstrated the effectiveness of ELSTMs in learning corrections to traditional filters. For instance, in [32], an ELSTM is trained to learn the nonlinear Kalman gain, improving estimation accuracy for systems with moderate nonlinearity. A deep learning architecture KalmanNet [104], integrates the recursive structure of the Kalman filter into a neural network. KalmanNet retains the sequential inference mechanism of Kalman filters while replacing key components—such as the gain matrix—with trainable networks. This architecture achieves better performance in partially known or unknown model scenarios and has been successfully applied to linear and nonlinear systems alike.

Beyond recurrent neural networks and ELSTMs, attention-based models such as transformers have started to gain traction in time-series modeling and state estimation [78]. These models do not rely on recurrence but instead learn dependencies through self-attention mechanisms, enabling efficient modeling of long-range interactions. While still relatively new to the field of filtering and estimation, early work suggests that transformer-based estimators can outperform recurrent neural networks in certain settings due to their global receptive fields and parallelizable training but they have a very complex architecture in comparison leading to a difficult stability analysis.

Another promising direction is the learning of estimators via optimization-based frameworks. In [75], Kunisch and co-authors develop a reduced-order estimator for infinite-dimensional systems by combining model reduction with optimization, offering stability and convergence guarantees. Such approaches form a bridge between traditional estimation theory and modern learning-based methods.

In summary, recurrent neural networks are proving to be a powerful tool for state estimation. Their ability to model nonlinear dependencies and temporal dynamics makes them valuable alternatives or complements to classical methods.

This chapter aims to provide a structured survey of these developments, highlighting the enduring relevance of traditional approaches and the potential of machine learning, alongside a detailed examination of the various cost functions and stability methods that shape the state estimation problem.

## 2.1 Traditional state estimation

A seminal contribution to the estimation of linear systems is the Kalman filter [59]. It is an optimal recursive algorithm for linear systems with Gaussian noises that provides an estimate of the system's state using a series of noisy measurements and a mathematical model of its dynamics. It consists of 2 steps, prediction and correction. The prediction step gives the estimated value of the state  $x^{(t+1)}$  given the measurements  $y^{(1)}, y^{(2)}, \dots, y^{(t)}$ , i.e., the *a priori* estimate,

$$\hat{x}^{(t|t+1)} = E[x^{(t)} | y^{(1)}, y^{(2)}, \dots, y^{(t-1)}],$$

the average value of  $x^{(t)}$  given measurements  $y^{(1)}, y^{(2)}, \dots, y^{(t-1)}$ . This is followed by the filtering step which finds the *a posteriori* estimate, i.e.,

$$\hat{x}^{(t|t)} = E[x^{(t)} | y^{(1)}, y^{(2)}, \dots, y^{(t)}].$$

Thus, the Kalman filter for a linear state space system given by

$$\begin{aligned} x^{(t)} &= Ax^{(t-1)} + Bu^{(t)} + w^{(t)}, w^{(t)} \sim \mathcal{N}(0, Q) \\ y^{(t)} &= Hx^{(t)} + \nu^{(t)}, \nu^{(t)} \sim \mathcal{N}(0, R) \\ x^{(0)} &\sim \mathcal{N}(\hat{x}^{(0)}, P^{(0)}) \end{aligned} \tag{2.3}$$

consists of the following equations:

**Prediction:**

$$\hat{x}^{(t|t-1)} = A\hat{x}^{(t-1|t-1)} + Bu^{(t)}, \quad (2.4)$$

$$P^{(t|t-1)} = AP^{(t-1|t-1)}A^\top + Q. \quad (2.5)$$

**Correction:**

$$K^{(t)} = P^{(t|t-1)}H^\top(HP^{(t|t-1)}H^\top + R)^{-1}, \quad (2.6)$$

$$\hat{x}^{(t|t)} = \hat{x}^{(t|t-1)} + K^{(t)}(y^{(t)} - H\hat{x}^{(t|t-1)}), \quad (2.7)$$

$$P^{(t|t)} = (I - K^{(t)}H)P^{(t|t-1)}. \quad (2.8)$$

It can be derived using the maximum likelihood cost function as follows:

Consider a discrete-time linear dynamical system with process and measurement models as in 2.3. Our goal is to estimate  $x^{(t)}$  using observations  $y^{(0)}, y^{(1)}, \dots, y^{(t)}$ , by maximizing the likelihood of the observations under the Gaussian model.

Given the prior  $p(x^{(t)} | y^{(0:t-1)}) = \mathcal{N}(\hat{x}^{(t|t-1)}, P^{(t|t-1)})$ , and the measurement model, the likelihood of  $y^{(t)}$  given  $x^{(t)}$  is:

$$p(y^{(t)} | x^{(t)}) \propto \exp\left(-\frac{1}{2}(y^{(t)} - Hx^{(t)})^\top R^{-1}(y^{(t)} - Hx^{(t)})\right).$$

Using Bayes' theorem:

$$p(x^{(t)} | y^{(0:t)}) = \frac{p(y^{(t)} | x^{(t)})p(x^{(t)} | y^{(0:t-1)})}{p(y^{(t)} | y^{(0:t-1)})}.$$

Since the denominator is independent of  $x^{(t)}$ , we can write:

$$p(x^{(t)} | y^{(0:t)}) \propto p(y^{(t)} | x^{(t)})p(x^{(t)} | y^{(0:t-1)}).$$

Thus,

$$\begin{aligned} p(x^{(t)} | y^{(0:t)}) &\propto \exp\left(-\frac{1}{2}(x^{(t)} - \hat{x}^{(t|t-1)})^\top (P^{(t|t-1)})^{-1}(x^{(t)} - \hat{x}^{(t|t-1)})\right) \\ &\quad \times \exp\left(-\frac{1}{2}(y^{(t)} - Hx^{(t)})^\top R^{-1}(y^{(t)} - Hx^{(t)})\right). \end{aligned} \quad (2.9)$$

Taking the negative log of the posterior (up to an additive constant), we obtain the following cost function:

$$\begin{aligned} J(x^{(t)}) &= \frac{1}{2}(x^{(t)} - \hat{x}^{(t|t-1)})^\top (P^{(t|t-1)})^{-1}(x^{(t)} - \hat{x}^{(t|t-1)}) \\ &\quad + \frac{1}{2}(y^{(t)} - Hx^{(t)})^\top R^{-1}(y^{(t)} - Hx^{(t)}). \end{aligned} \quad (2.10)$$

Taking the gradient and setting it to zero:

$$\nabla_{x^{(t)}} J = (P^{(t|t-1)})^{-1}(x^{(t)} - \hat{x}^{(t|t-1)}) - H^\top R^{-1}(y^{(t)} - Hx^{(t)}) = 0. \quad (2.11)$$

Solving for  $x^{(t)}$  which is the predicted estimate  $\hat{x}^{(t|t)}$ :

$$\begin{aligned} (H^\top R^{-1}H + (P^{(t|t-1)})^{-1})\hat{x}^{(t|t)} &= H^\top R^{-1}y^{(t)} + (P^{(t|t-1)})^{-1}\hat{x}^{(t|t-1)}, \\ (P^{(t|t-1)})^{-1}(\hat{x}^{(t|t)} - \hat{x}^{(t|t-1)}) &= H^\top R^{-1}(y^{(t)} - H\hat{x}^{(t|t)}) \\ &= H^\top R^{-1}(y^{(t)} - H\hat{x}^{(t|t-1)} - H(\hat{x}^{(t|t)} - \hat{x}^{(t|t-1)})) \\ &= H^\top R^{-1}(y^{(t)} - H\hat{x}^{(t|t-1)}) - H^\top R^{-1}H(\hat{x}^{(t|t)} - \hat{x}^{(t|t-1)}) \end{aligned}$$

$$((P^{(t|t-1)})^{-1} + H^\top R^{-1}H)(\hat{x}^{(t|t)} - \hat{x}^{(t|t-1)}) = H^\top R^{-1}(y^{(t)} - H\hat{x}^{(t|t-1)})$$

Solving for the update:

$$\hat{x}^{(t|t)} - \hat{x}^{(t|t-1)} = ((P^{(t|t-1)})^{-1} + H^\top R^{-1}H)^{-1}H^\top R^{-1}(y^{(t)} - H\hat{x}^{(t|t-1)}) \quad (2.12)$$

We now show that this gain term is equivalent to the standard formula using the Woodbury Matrix Identity:  $(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$ .

Let  $K^{(t)} = ((P^{(t|t-1)})^{-1} + H^\top R^{-1}H)^{-1}H^\top R^{-1}$ . Applying the identity to the first part gives the following:

$$((P^{(t|t-1)})^{-1} + H^\top R^{-1}H)^{-1} = P^{(t|t-1)} - P^{(t|t-1)}H^\top(R + HP^{(t|t-1)}H^\top)^{-1}HP^{(t|t-1)}$$

Substituting this into the expression for  $K^{(t)}$ :

$$\begin{aligned} K^{(t)} &= [P^{(t|t-1)} - P^{(t|t-1)}H^\top(R + HP^{(t|t-1)}H^\top)^{-1}HP^{(t|t-1)}]H^\top R^{-1} \\ &= P^{(t|t-1)}H^\top R^{-1} - P^{(t|t-1)}H^\top(R + HP^{(t|t-1)}H^\top)^{-1}HP^{(t|t-1)}H^\top R^{-1} \end{aligned}$$

Let  $S^{(t)} = R + HP^{(t|t-1)}H^\top$ . Then  $HP^{(t|t-1)}H^\top = S^{(t)} - R$ .

$$\begin{aligned} K^{(t)} &= P^{(t|t-1)}H^\top R^{-1} - P^{(t|t-1)}H^\top(S^{(t)})^{-1}(S^{(t)} - R)R^{-1} \\ &= P^{(t|t-1)}H^\top R^{-1} - P^{(t|t-1)}H^\top(S^{(t)})^{-1}(S^{(t)}R^{-1} - I) \\ &= P^{(t|t-1)}H^\top R^{-1} - P^{(t|t-1)}H^\top(R^{-1} - (S^{(t)})^{-1}) \\ &= P^{(t|t-1)}H^\top R^{-1} - P^{(t|t-1)}H^\top R^{-1} + P^{(t|t-1)}H^\top(S^{(t)})^{-1} \\ K^{(t)} &= P^{(t|t-1)}H^\top(S^{(t)})^{-1}. \end{aligned}$$

Thus, we recover the Kalman gain form:

$$K^{(t)} = P^{(t|t-1)} H^\top (H P^{(t|t-1)} H^\top + R)^{-1}. \quad (2.13)$$

Then, the optimal state estimate is:

$$\hat{x}^{(t)} = \hat{x}^{(t|t-1)} + K^{(t)}(y^{(t)} - H\hat{x}^{(t|t-1)}). \quad (2.14)$$

And the updated error covariance:

$$P^{(t)} = (I - K^{(t)}H)P^{(t|t-1)}. \quad (2.15)$$

We find the maximum likelihood estimate by minimizing the negative log-posterior (2.10). The first term penalizes the deviation of  $x^{(t)}$  from the prior mean  $\hat{x}^{(t|t-1)}$ , weighted by the prior covariance. The second term penalizes the inconsistency between  $x^{(t)}$  and the observed measurement  $y^{(t)}$ , weighted by the measurement noise covariance.

There are several generalizations of this approach that are used for nonlinear estimation. The most obvious generalization of this filter, [EKF](#), is based on linearization to propagate the mean and covariance of the state. It achieves this by linearizing the nonlinear system and measurement models around the current state estimate:

$$A^{(t)} = \left. \frac{\partial f}{\partial x} \right|_{\hat{x}^{(t-1|t-1)}}, \quad H^{(t)} = \left. \frac{\partial h}{\partial x} \right|_{\hat{x}^{(t|t-1)}} \quad (2.16)$$

and then applying Kalman update using  $A^{(t)}$ ,  $H^{(t)}$ . Such a generalization does not work well for many non-linear systems despite local convergence guarantees [69]. This is because there is no global convergence guarantee for systems with severe nonlinearities which cannot be appropriately approximated by a linear function [111]. Jacobian computations can be complex, linearization errors can be high, and poor initialization can lead to divergence. Thus, the method is not suitable for many non-linear systems. Another common approach is to use [UKF](#), which was developed to counteract the linearization limitations of [EKF](#). It uses the idea that instead of using the entire density function to propagate means and covariances, one can pick some specific sample points called "sigma points" that symmetrically capture the mean and covariance of the state distribution. These points are then propagated through the nonlinear system and the measurement function, and their transformed mean and covariance are subsequently used to estimate the state. It uses a minimum [MSE](#) objective as [EKF](#) to estimate  $\hat{x}^{(t)}$  as a weighted average of estimated state at each sigma point. Thus, rather than using a nonlinear transformation on an entire density function, it uses it on a few points, in turn decreasing the linearization error. This

makes it less prone to divergence. But it still assumes Gaussian noises and is not an optimal filter. To handle highly nonlinear systems and non-Gaussian noises, particle filters (PFs), also known as sequential Monte Carlo methods, were introduced. They represent the posterior probability distribution of the state using a set of particles with associated weights and resamples to prevent degeneracy [26]:

$$\hat{x}^{(t)} = \sum_{i=1}^N w_i^{(t)} x^{(t,i)} \quad (2.17)$$

where  $w_i^{(t)}$  is the weight associated with particle  $i$  at time-step  $t$  calculated using importance sampling for our work. Since the number of particles required for favorable estimation of high-dimensional systems is exponential, it can be computationally expensive. Thus, the curse of dimensionality makes it difficult to use this filter. Another drawback is that, without proper resampling of particles, they can converge to a single point, losing diversity. Beyond these widely used filters, there are several other traditional methods that exist, each suited to specific problem characteristics. Simple, fixed-gain alpha-beta filters[111] are often used for tracking, particularly when computational resources are limited. They are essentially simplified Kalman filters for constant velocity or acceleration models. Least squares estimation[42] is a foundational technique for estimating parameters or states by minimizing the sum of squared residuals. Although not a dynamic filter, it forms the basis for many static state estimation problems, such as power system state estimation. Moving horizon estimation[61] is an optimization-based approach that estimates the state by solving an optimal control problem over a finite, moving time window. It can naturally handle constraints and nonlinearities, but is computationally more demanding than recursive filters.

Several cost functions are used for optimal state estimation. The minimum variance estimator, also called the least square estimator [111] which uses variance as a cost function is one of the most commonly used optimal cost function for state estimation. Another optimal cost function is the maximum likelihood cost function. The goal of maximum likelihood estimation is to make inferences about the population that is most likely to have generated the sample. The maximum log-likelihood cost function for systems with white

Gaussian noises is given as follows:

$$\begin{aligned}
& \max \log L(\cdot) \\
&= \max_{\hat{x}^{(0)}, w^{(i)}, v^{(i)}} \log \left( \frac{1}{\sqrt{2\pi|P_0|}} \right) - \left( \frac{1}{2} (x^{(0)} - \hat{x}^{(0)})' P_0^{-1} (x^{(0)} - \hat{x}^{(0)}) \right) \\
&+ \sum_{i=0}^t \left\{ \log \left( \frac{1}{(2\pi)^{\frac{n}{2}} \sqrt{|Q|}} \right) - \left( \frac{1}{2} (x^{(i+1)} - f(x^{(i)}, u^{(i+1)}))' Q^{-1} (x^{(i+1)} - f(x^{(i)}, u^{(i+1)})) \right) \right. \\
&+ \left. \log \left( \frac{1}{(2\pi)^{\frac{n}{2}} \sqrt{|R|}} \right) - \left( \frac{1}{2} (y^{(i)} - h(x^{(i)}))' R^{-1} (y^{(i)} - h(x^{(i)})) \right) \right\}.
\end{aligned} \tag{2.18}$$

Note that maximizing  $\log L(\cdot)$  is equivalent to minimizing its negative log posterior:

$$\begin{aligned}
J(\cdot) &= \frac{1}{2} (x^{(0)} - \hat{x}^{(0)})' P_0^{-1} (x^{(0)} - \hat{x}^{(0)}) \\
&+ \sum_{i=0}^t \frac{1}{2} \left\{ (x^{(i+1)} - f(x^{(i)}, u^{(i+1)}))' Q^{-1} (x^{(i+1)} - f(x^{(i)}, u^{(i+1)})) \right. \\
&\quad \left. + (y^{(i)} - h(x^{(i)}))' R^{-1} (y^{(i)} - h(x^{(i)})) \right\}.
\end{aligned} \tag{2.19}$$

For a linear system with Gaussian noises, **KF** is in fact also called the minimum mean squared error estimator as it minimizes the expected value of estimation error and the minimum variance estimator along with being the maximum likelihood estimator as shown above. Thus, the maximum likelihood cost function and the minimum variance cost function are all equivalent for linear systems with Gaussian disturbances[111]. In order to perform state estimation of nonlinear systems in an optimal manner, Mortensen [92] extended the Kalman filter approach to continuous-time nonlinear state-space systems. He chose to estimate the initial condition noise and process noise which in turn provided the optimal state estimator. He made certain assumptions in order to make sure that an optimum for the maximum likelihood cost function for continuous-time systems exists. Then Moireau [88] utilized the discrete-time formulation to extend Mortensen's results to discrete-time state space systems for nonlinear operators. He showed that the discrete-time Mortensen filter reduces to the Kalman filter for linear operators and affine dynamics under certain conditions [[88], Proposition 2.8]. Hijab [43] further refined Mortensen's estimator [92] and called his approach Minimum Energy Estimation for continuous-time systems.

The minimum energy approach aims to find the noise triple  $\hat{x}_0, w(\cdot), v(\cdot)$  that minimizes the following cost function[71]:

$$\min_{\hat{x}_0, w, v} \frac{1}{2} \left\{ \alpha^t \|\hat{x}^{(0)}\|_{P_0}^2 + \sum_{s=0}^{t-1} \alpha^{t-s} \|w^{(s)}\|_Q^2 + \sum_{s=1}^{\tau} \|v^{(s)}\|_R^2 \right\} \quad (2.20)$$

where  $0 < \alpha \leq 1$  is called the forgetting factor, and  $\tau = t$  or  $\tau = t - 1$ . Krener [70] in 2003 further proved the convergence of the minimum energy estimate to the true state under certain assumptions. However, it is numerically challenging to find the solution in the viscosity sense of the Hamilton-Jacobi partial differential equation (driven by measurements) needed to find the estimate in high dimensions, and the accuracy of the estimate is also limited, thus making this approach not viable for high-dimensional systems[71]. Pequito [97] presented a framework for developing optimal estimators using mean-field games for a general class of nonlinear systems. This method also utilizes the Hamilton-Jacobi-Bellman framework. He also presented an example where the extended Kalman filter diverges, but the proposed estimator converges. But he only proved convergence in the linear case and the method was tested on a nonlinear equation of order 1 only. Krener[71] used Taylor polynomials in combination with a minimum energy approach and showed that this method reduced to EKF when the degree of polynomial was restricted to 1. He emphasized that the degree of the Taylor polynomial and the accuracy of the estimation algorithm are directly proportional. A drawback is that, as the degree of the polynomial increases, the need for higher-order derivatives of high-dimensional functions makes this method difficult to implement. Another common approach is to use an H-infinity filter that minimizes the worst-case estimation error [111]:

$$\min_{\hat{x}^{(t)}} \max_{w^{(t)}, v^{(t)}} \|x^{(t)} - \hat{x}^{(t)}\|^2. \quad (2.21)$$

It focuses on robustness over optimality.

The issue with traditional methods is with generalization to high-order nonlinear systems and their robustness to noise. Thus, researchers are exploring machine learning based methods.

## 2.2 Data-driven state estimation

The advancements in the field of data science due to better computational power and algorithmic sophistication have propelled machine learning to the forefront of many scientific and engineering domains, including state estimation. Machine learning methods offer

a data-driven paradigm, capable of learning complex, nonlinear relationships, and noise characteristics directly from data, often without requiring explicit analytical models of the system [41]. But when a model is available, it may enhance the algorithm or even help counter the black-box nature of neural networks by providing some interpretation. Thus, machine learning algorithms for state estimation have become increasingly common.

Using only input-output data to identify states is called system identification or trajectory tracking. A major difference between these methods and state estimation is that they do not use the model, even when it is available. Recurrent neural networks have been shown to work well for system identification (see for example, [96]). JRN<sub>s</sub> have been used for system identification by several researchers, including [63, 123], and have been shown to work as well as ERN<sub>s</sub>. Convergence analysis, with some assumptions, of JRN<sub>s</sub> as well as ERN<sub>s</sub> for system identification has been discussed in [72].

To predict future values based on observed trends in data is called time-series analysis or forecasting. Recurrent neural networks have been used for forecasting from several years. ERN<sub>s</sub> provided the best estimation properties when compared with the method of least squares and feedforward neural networks [8] for noisy time series data with fully observable states in [37]. In [16], an Elman-based long short-term memory (ELSTM) network architecture is explored for different systems and compared with an EKF for filtering. The authors conclude that the proposed ELSTM structure performs better than the EKF for nonlinear systems. In [60], authors show that the ERN performs at least as well as a feedforward neural network for Lyapunov exponents forecasting, discussing cases where it performs better in detail as well. Deep neural networks have shown remarkable capabilities in learning complex non-linear mappings. For state estimation, it can be trained to directly predict the states of a system given a sequence of past and current measurements. But they require large amounts of measurements and corresponding true states and lack interpretability, are sensitive to training data distribution, and may struggle with extrapolation.

While machine learning usage in neural ordinary/partial differential equations and physics-informed neural networks is fairly common, they focus on learning the vector field whereas in state estimation one learns the estimator. An estimator unlike a basic implementation of a model, uses measurements in the state update.

A machine learning framework that overcame the requirement to find solutions to the Hamilton-Jacobi Bellman partial differential equation in the continuous-time case is discussed in [15]. A neural network to approximate the gradient of the solution of Hamilton-Jacobi Bellman equation is used. The equation is exploited to construct the filter which is referred to as the observer gain. The results show that it coincides with the Kalman-Bucy

filter for a linear problem. Another direction is exemplified by Adhyaru [2], who proposed a deep learning framework for state estimation based on the Hamilton–Jacobi–Bellman equation. Instead of directly estimating the state, they learned an optimal cost-to-go function from which the filter gain can be derived. A neural network is trained to approximate the solution of the HJB equation, and the gradient of this network provides the optimal feedback law used for estimation. This method bridges reinforcement learning with classical control theory and enables optimal state estimation in settings where explicit solutions to HJB equations are unavailable.

A data-driven approach, which utilized the KF framework of correction and prediction for discrete-time systems, is discussed in [30]. Even though the focus of this work was trajectory tracking, the modified assumptions and use of model, led to the problem being that of state estimation. Recurrent neural networks are used, in particular, long short-term memory networks, to approximate the state from observations, initial data and noise estimates. A two step process of prediction and correction is used and two frameworks are presented, one Bayesian and the other non-Bayesian. For the Bayesian approach, the neural network is designed so that it predicts and filters iteratively within the same network. This framework is supported by using Bayes rule. For the non-Bayesian approach two deep ELSTM networks are used, one network for prediction and the other for filtering, based on the fact that neural networks are individually capable of performing both the steps independently. Thus, one network performed prediction without measurements and the other network corrected the predictions by using new measurements.

A nonlinear regression-based estimator for discrete-time systems was constructed in [101]. In their framework, data is generated by simulating both the true system and the dynamics of a classical estimator (such as a Luenberger estimator). This data is then used to train a regression model—typically a neural network—to approximate the mapping from measurements and past estimates to the current state. This strategy removes the need for explicitly computing filter gains, and enables end-to-end learning of the estimator using supervised data from simulation or experiments. Their method is well-suited to settings where a simulator is available but system equations are too complex for classical design. In a similar vein, Peralez and Nadri [98] formulated a discrete-time Luenberger estimator for nonlinear systems using an unsupervised learning approach. Instead of using labeled state data, they trained a neural network to approximate the estimator mapping by enforcing internal consistency with the system dynamics. Specifically, the network receives current and past outputs and estimates the latent state by minimizing a loss function based on the system evolution equations and observation model. This avoids the need for direct access to the state trajectory during training. Their work highlights the potential of self-supervised learning to construct estimators when labeled state data is scarce or unavailable.

Benosman and Borggaard [10] proposed a robust, low-order estimator design for infinite-dimensional nonlinear systems using a combination of proper orthogonal decomposition and extremum seeking. Their reduced-order model captures the dominant modes of systems such as the 1D viscous Burgers equation and the 2D Boussinesq equations. They designed a feedback estimator using the reduced model, and employed a real-time data-driven tuning algorithm to optimize filter gains. This method does not require full-state measurements and is robust to modeling errors, making it particularly suitable for high-dimensional PDE systems. In later extensions [9], the approach was further applied to more complex fluid dynamics settings.

Jun Fu et al. [29] presented a neural-network-based adaptive estimator designed for systems with dynamics evolving on multiple time scales. Their estimator architecture contains separate subnetworks for fast and slow dynamics, enabling better modeling of real-world systems such as robotic manipulators or biochemical systems. The estimation framework is adaptive in the sense that the neural network weights evolve over time using update laws derived from Lyapunov theory. They provided theoretical guarantees for convergence of both the estimation error and the neural network parameters under suitable assumptions.

Reinforcement learning-based approaches are also becoming popular due to recent advancements in the field. In [33], the authors proposed a reinforcement learning framework combined with a KF for tracking ground vehicles using an integrated navigation system, and called it an adaptive KF navigation algorithm. The state of the system was estimated using a KF and then used to calculate rewards for a reinforcement learning framework which updated the process noise covariance. In [112] the problem is viewed as a Bayes-adaptive Markov decision process and solved online using Monte Carlo tree search with a UKF to account for process noise and parameter uncertainty. For details about Monte Carlo tree search see [14].

Use of a reduced-order system is a common approach since it helps overcome the curse of dimensionality. A deep state estimator which used a neural network methodology to develop a nonlinear relationship between the measurements and a reduced order state was developed in [93]. This relationship was commonly approximated as linear and then they used the advancements in the field of deep learning to make better approximations. The approximated reduced-order state was used as an initial condition to estimate the full state. His proposed methodology outperformed common linear estimation algorithms when tested on 1D Burgers equation and 2D Boussinesq equations. A recurrent neural network based on the principles of a decoupled EKF for state estimation was introduced in [125] and used on reduced order models.

Reservoir computing, particularly the echo state network variant, has emerged as a highly efficient paradigm for state estimation and forecasting of nonlinear dynamical systems. Its hallmark architecture comprises a high-dimensional fixed and randomly initialized reservoir alongside a trainable output (readout) layer—only the latter is optimized, typically via linear regression. This design drastically reduces training time and computational cost and sidesteps issues like vanishing or exploding gradients common in fully trained recurrent networks [83, 50]. Such computational simplicity makes it ideal for real-time and resource-constrained applications. Empirical studies demonstrate its strength in modeling chaotic dynamical systems [100], [119]. It has also been applied to state observation and synchronization of chaotic systems: Nazerian et al. [95] used a reservoir-based estimator to reconstruct unmeasured states from limited observed outputs, enabling synchronization of a response system with a drive system under limited measurement condition. To enhance physical consistency, a physics-constrained reservoir computer variant was proposed for turbulent flow prediction: it combined reservoir computing with physical conservation laws to improve the accuracy in predicting extreme events and long-term velocity statistics in chaotic flows under noise [25]. Despite its speed and demonstrated success, it presents notable drawbacks. The fixed random reservoir results in stochastic variability across model instantiations, leading to reproducibility and robustness issues. Crucially, reservoir computing lacks built-in mechanisms to guarantee stability, convergence, or optimality in estimation error, unlike Kalman- or estimator-based methods grounded in control theory. Although physics-constrained reservoir computing adds some structural consistency, reservoir computing models remain fundamentally data-driven and black-box in nature, without direct interpretability or theoretical guarantees.

In [3], they use an ELSTM for state estimation with weighted least squares as the loss function. Although the mean squared error (MSE) and implementation time are significantly less than some other methods, the training time is very large. [23] combine an ELSTM with an EKF to perform state estimation and use reduced-order models to reducing training time. While using a reduced-order model may work well for certain scenarios, high order complex systems may not give optimal results with reduced order models. While these methods give better estimation accuracy than traditional methods, the architecture corresponding to them is complex and takes a long time to train. Thus, there is a need to explore methods that can help reduce training time and preserve accuracy for complex systems.

## 2.3 Stability analysis for the error dynamics

The error dynamics defined as:

$$e^{(t)} := x^{(t)} - \hat{x}^{(t)} = f(x^{(t-1)}, u^{(t-1)}) + \omega^{(t)} - \hat{x}^{(t)} \quad (2.22)$$

describe the evolution of the estimation error over time. Here,  $x^{(t)}$  and  $\hat{x}^{(t)}$  are the true and estimated state at time-step  $t$  respectively.

The reliability and performance of state estimation algorithms are based on the stability of their error dynamics. Reliable state estimation is essential in high-stakes applications—such as autonomous vehicles and power grid operations—where instability or inaccuracies can result in severe or even catastrophic outcomes. In this section, a review of the stability of error dynamics in state estimation, differentiating between traditional model-based filters and emerging machine learning paradigms is provided. We mention the theoretical foundations of stability, including deterministic and stochastic stability, and the conditions under which various filters achieve convergence and boundedness of their estimation errors. We also discuss the challenges posed by nonlinearities of the system, uncertainties in the model, and the black-box nature of data-driven methods. Furthermore, we explore some techniques and research efforts aimed at ensuring robust and verifiable stability for state estimators.

The boundedness of estimation error, its convergence to zero, or to a bounded region around zero, despite the presence of noise and disturbances characterize stability in the context of state estimation. Analyzing and ensuring this stability is a difficult task, varying significantly depending on the underlying system dynamics being linear or nonlinear, noise characteristics being Gaussian or non-Gaussian, and the estimation methodology employed.

**Definition 2.3.1.** *Lyapunov stability:* If for any given  $\epsilon > 0$ , there exists a  $\delta > 0$  such that if  $\|e^{(0)}\| < \delta$ , then  $\|e^{(t)}\| < \epsilon$  for all  $t \geq 0$ .

**Definition 2.3.2.** *Asymptotic stability:* If it is Lyapunov stable and  $\lim_{t \rightarrow \infty} \|e^{(t)}\| = 0$ .

**Definition 2.3.3.** *Exponential stability:* If it is asymptotically stable and the convergence rate is exponential, i.e.,  $\|e^{(t)}\| \leq c\lambda^t \|e^{(0)}\|$  for some constants  $c > 0$  and  $0 < \lambda < 1$ .

These notions are typically analyzed using Lyapunov functions, where a positive definite function  $V(e_t)$  is sought such that  $\Delta V(e^{(t)}) = V(e^{(t+1)}) - V(e^{(t)})$  is negative definite.

When process and/or measurement noise are explicitly modeled as random variables, stability refers to probabilistic notions of error boundedness or convergence, known as stochastic stability. Our work will focus only on deterministic notions.

Traditional state estimation methods rely on explicit mathematical models of system dynamics and noise. Their stability properties are often well-understood due to their strong theoretical foundations. The stability of KF[59] is directly linked to the boundedness and convergence of the estimation error covariance matrix  $P^{(t)}$ . Mean square stability is achieved if  $P^{(t)}$  remains bounded as  $t \rightarrow \infty$ . This implies that the expected value of the squared error is bounded. The KF is asymptotically stable i.e.,  $P^{(t)} \rightarrow \bar{P}$  for some bounded  $\bar{P}$ , if the system is detectable from the measurements, stabilizable by the process noise (i.e., any unstable modes are affected by the process noise or are inherently stable) and,  $Q$  and  $R$  are positive definite. When these conditions are met, the Riccati equation converges to a steady-state solution, leading to a stable and bounded  $P^{(t)}$  [86, 36]. This is a strong theoretical guarantee for the KF but is limited to linear Gaussian noises and initial conditions.

The EKF[51] is an extension of the KF to nonlinear systems which relies on linearization the system dynamics and measurement models around the current state estimate using Taylor series expansions. Its stability analysis is complex than the linear KF. Its stability is typically analyzed as local asymptotic stability, that is, if the initial state estimate is sufficiently close to the true state, and the nonlinearities are not too complex, the estimation error will converge or remain bounded. To achieve this, it must satisfy several conditions: the Jacobian matrices  $A_k$  and  $H_k$  must be observable along the true trajectory, the nonlinear functions  $f(\cdot)$  and  $h(\cdot)$  are often assumed to be Lipschitz continuous [103], the Jacobian matrices should not grow unbounded, process and measurement noise should be sufficiently exciting. Thus the EKF can diverge if the initial estimate is far from the true state, the nonlinearities are complex, the linearization errors are large or the system becomes unobservable along certain trajectories. The practical applications of EKF are thus reliant on careful tuning and validation [86].

The UKF[57, 56] often exhibits better performance and is less prone to divergence than the EKF for highly nonlinear systems because it generally provides a more accurate representation of the propagated mean and covariance. Despite its practical advantages, formal theoretical stability guarantees for the UKF are still challenging and often rely on similar assumptions as the EKF. While the UKF approximates moments more accurately, it still relies on a Gaussian approximation of the state distribution, which may not hold for complex nonlinear systems, impacting global stability guarantees [105].

PFs[39, 26] are capable of handling arbitrary nonlinearities and non-Gaussian noise distributions. They represent the posterior probability distribution of the state using a set of weighted random particles. Their stability is typically analyzed in terms of the convergence of the empirical distribution of particles to the true posterior probability distribution. Under certain regularity conditions like sufficient number of particles, well-behaved likelihoods

and bounded variance of importance weights, PFs are known to converge almost surely to the true posterior distribution as the number of particles  $N \rightarrow \infty$  [21]. This implies that the estimation error can be made arbitrarily small with sufficient particles. There are several challenges though. The number of particles required for accurate representation grows exponentially with the state dimension, severely limiting their applicability to high-dimensional problems. Insufficient particles can lead to particle degeneracy where most particles have negligible weight, hindering convergence and effectively leading to an unstable representation of the posterior. Particle depletion can occur if the posterior distribution is very narrow or has low overlap with the proposal distribution, leading to poor resampling and instability. Effective resampling strategies are crucial for maintaining stability. The high number of particles makes PFs computationally intensive.

Although distinct from stochastic filters that explicitly account for noise, deterministic estimators play a vital role in state estimation across numerous settings. Their effectiveness hinges on the stability of the estimation error dynamics—typically ensuring convergence to zero either in noise-free conditions or under bounded noise[114]. In linear systems, the Luenberger estimator achieves asymptotic error stability provided the system is detectable, with the filter gain matrix designed to place the error dynamics eigenvalues within the unit circle (discrete-time) or the left-half complex plane (continuous-time). For nonlinear systems, a variety of estimator structures exist—including high-gain, sliding mode, and extended Luenberger estimators. Their stability is commonly established via Lyapunov methods, often under assumptions such as Lipschitz continuity, output map invertibility, or canonical observability forms. Depending on the design and assumptions, convergence can be either asymptotic or exponential.

The development of powerful machine learning techniques has opened new avenues for state estimation, particularly in scenarios where dynamic models are highly nonlinear. However, establishing formal stability guarantees for machine learning based estimators is a significant and ongoing challenge. Deep neural networks operate as complex black-box functions making it challenging to derive explicit error dynamics equations or construct Lyapunov functions for them. The stability and performance of these models are mainly dependent on the quality, quantity, and representativeness of the training data. Generalization to unseen data or operating conditions outside the training distribution is generally not guaranteed and can lead to unstable or unexpected estimates. They typically lack strong theoretical guarantees on convergence or boundedness of estimation errors as compared to the traditional filters whose optimality and stability conditions are often derived from first principles. They may be susceptible to small, unmodeled perturbations in input data, leading to sudden and significant errors. Training these models involves non-convex optimization, which can lead to local minima and suboptimal solutions that do not guar-

antee stable error dynamics[54].

When supervised learning models [40])are trained to directly map sequences of measurements to states, their stability is often assessed empirically through validation on unseen data. Performance metrics like root MSE on test sets or average error across time-steps are common. However, these do not provide rigorous guarantees about long-term stability or robustness to sequences unseen during training. Issues like vanishing or exploding gradients during training of the network can hinder the learning process and lead to poor, unstable predictions. Architectural choices mitigate these, but don't guarantee output stability. Some researchers apply input to state stability concepts to neural network-based estimators, treating the network as an input-output system and analyzing how bounded inputs lead to bounded state estimates [115]. However, this is still an active research area and often requires strong assumptions on network architecture and activation functions.

To address the challenge of ensuring stability in machine learning-based state estimators, researchers have proposed frameworks that combine the physical insights and stability guarantees of model-based approaches with the adaptability of data-driven methods [54]. These methods aim to leverage the learning capabilities of machine learning while retaining the theoretical guarantees offered by classical control and estimation theory. Machine learning can be used to learn unknown or time-varying noise covariances or adapt parameters within a well-established filter structure instead of replacing the entire filter. Thus if the underlying filter remains stable under the adapted parameters, overall stability can be maintained. Machine learning models can be trained to estimate and compensate for noisy or unmodeled part of the dynamics, which are then integrated into a model-based filter. If the learned mismatch is bounded and the core filter is robustly stable, the combined system might achieve stable estimation. Architectures that combine the KF structure into a neural network [104, 109] attempt to learn the Kalman gain or parts of the system model. The goal is to retain the stability properties of the Kalman filter while gaining better adaptability. Proving that the learned gain sequence leads to stable error covariance is a key research challenge. Early results show promise, but full theoretical guarantees under general conditions are still elusive. There is also growing interest in equipping machine learning estimators with quantifiable uncertainty measures—using techniques like Bayesian neural networks[30] or deep ensembles—to better assess reliability. Robustness to adversarial or noisy inputs is another active area, ensuring estimators maintain stable behavior under perturbations. Efforts in explainable artificial intelligence seek to clarify the internal decision processes of these models, which is essential for diagnosing potential instability and building trust. Moreover, certifiable stability remains a grand challenge, with researchers developing formal verification tools to provide mathematical guarantees for neural-network-driven dynamics. Finally, online adaptive strategies are being explored,

enabling estimators to adjust their internal structure in real time to maintain stability as system dynamics or environmental conditions evolve.

Ultimately, progress in state estimation hinges on closing the gap between strong empirical performance and rigorous theoretical guarantees, especially in the context of data-driven approaches. The key is to ensure the stability of estimation error dynamics in a manner that is both robust and verifiable. This is essential for enabling autonomous and intelligent systems to function safely and reliably in complex, uncertain real-world settings.

## Chapter 3

# Stability of Jordan recurrent network

Like stability is critical for a control system, in the absence of disturbances, the estimated state must converge to the true state. Small disturbances should increase the error by a small amount. In other words, the error dynamics must be input to state stable[115]. It is known that for a linear system, if the measurements satisfy the weak assumption of detectability, the error dynamics are stable. It was recently shown [4] that if a nonlinear system is locally detectable, then the error dynamics of an **EKF** are locally input to state stable. If the system is uniformly observable and can be put globally into the normal form, along with some other technical assumptions on the system dynamics then the **EKF** error dynamics are stable [34, sec. 2.4]. For a discrete-time **UKF** satisfying certain assumptions, the estimation error remains bounded [124].

The stability of errors of RNNs as state estimators has not been well explored. However, there are several approaches to establishing the stability of RNNs by finding a Lyapunov function [79, 68, 128]. This suggested extending this work to show stability of error dynamics. The issue with **ERNs** in this context is that proving the stability of the estimator is difficult due to its dependence on the hidden neuron state at the previous time-step. This is not the case for a **JRN**. Also, the structure of **JRN** mimics the structure of a dynamical system. We therefore design an estimator using **JRN**s and utilize an input to state stability approach to analyze the stability of the error dynamics. Finding Lyapunov functions is generally challenging. Several papers [17, 128] have shown that Lyapunov functions can be attained and represented by expressions of the compositional structure of the neural network, the correctness being guaranteed by satisfiability modulo theories solvers. We use this approach to establish a condition under which the error dynamics are stable if the original system is stable. Estimation of unstable systems is generally conducted in combination with a stabilizing feedback; our approach assumes that any necessary stabilization

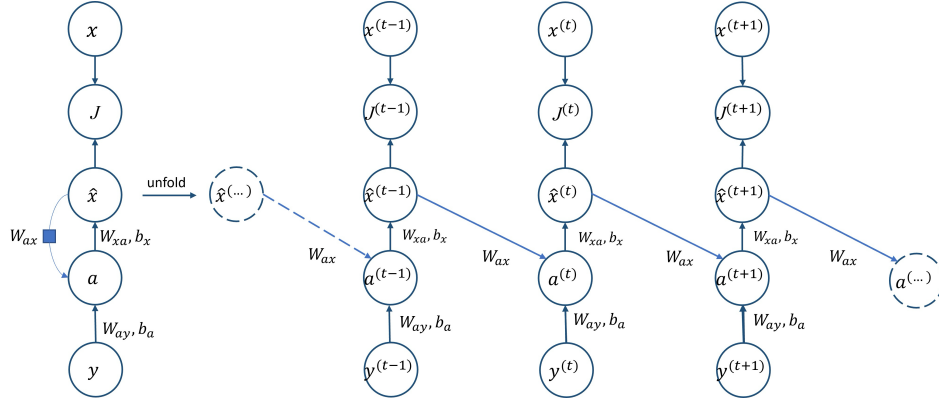


Figure 3.1: Jordan recurrent network (JRN) for state estimation showing output-to-hidden recurrent connections.

has been performed.

Consider the discrete-time nonlinear system in 2.1 with zero input at each time-step. Our aim is to estimate the state vector  $x^{(t+1)}$  by  $\hat{x}^{(t+1)}$  based on the measurement vector  $y^{(t+1)}$  and the previous state vector estimate  $\hat{x}^{(t)}$ .

A JRN has recurrent connections from the output of the previous layer to the hidden state of the next layer. Thus, we use  $y^{(t+1)}$  and  $\hat{x}^{(t+1)}$  as inputs and outputs, respectively, of the JRN. The forward propagation of the network is

$$\begin{aligned} a^{(t+1)} &= \sigma(W_{ay}y^{(t+1)} + W_{ax}\hat{x}^{(t)}), \\ \hat{x}^{(t+1)} &= W_{xa}a^{(t+1)}, \end{aligned} \tag{3.1}$$

where  $\sigma$  is the activation function and  $\hat{x}^{(t+1)}$  is the estimated state vector at time-step  $t+1$ . The hidden layer vector at time-step  $t+1$  is given by  $a^{(t+1)}$ . The weights and biases of the network are represented by  $W_{ay}$ ,  $W_{ax}$  and  $W_{xa}$ . The network is depicted in Fig. 3.1.

This is different from an ERN because the recurrent connections for an ERN are from the previous hidden layer  $a^{(t)}$  to the next hidden layer  $a^{(t+1)}$ , whereas for a JRN, they are from the previous output  $\hat{x}^{(t)}$  to next hidden layer  $a^{(t+1)}$ . Thus, for an ERN, the term  $W_{ax}\hat{x}^{(t)}$  would be replaced by  $W_{aa}a^{(t)}$  where  $W_{aa}$  is a weight matrix. Note that we are considering a bias-free network.

To analyze the stability of the neural estimator, we consider the case with no process

and measurement noise in (2.1), i.e.,  $w^{(t+1)} = v^{(t+1)} = 0$  for all  $t$ :

$$\begin{aligned} x^{(t+1)} &= f(x^{(t)}), \\ y^{(t+1)} &= h(x^{(t+1)}) \\ x^{(0)} &= \hat{x}^{(0)} \end{aligned} \tag{3.2}$$

The solution of system (3.2) is denoted by  $x^{(t)}(\xi)$  at time step  $t$  with initial condition  $x^{(0)} = \xi \in \mathcal{X}$ . We define the error term as  $e^{(t+1)} = x^{(t+1)} - \hat{x}^{(t+1)}$ . Then,

$$\begin{aligned} e^{(t+1)} &= x^{(t+1)} - W_{xa}\sigma(W_{ay}y^{(t+1)} + W_{ax}\hat{x}^{(t)}) \\ &= f(x^{(t)}) - W_{xa}\sigma(W_{ay}h(f(x^{(t)}))) \\ &\quad + W_{ax}x^{(t)} - W_{ax}e^{(t)}. \end{aligned} \tag{3.3}$$

Obviously,  $e^{(t+1)}$  is a function of  $e^{(t)}$  and  $x^{(t)}$ , and we call it the error system, denoted as follows,

$$e^{(t+1)} := g(e^{(t)}, x^{(t)}), \tag{3.4}$$

where  $e^{(t)} \in \mathcal{E} \subseteq \mathbb{R}^n$ , and  $\mathcal{E}$  is the space for the error term. In a similar manner, we denote the solution to system (3.4) as  $e^{(t)}(\eta, x)$  with the initial condition  $e^{(0)} = \eta \in \mathcal{E}$ .

We introduce Lyapunov functions for the error dynamics, regarding  $x^{(t)}$  as the input in (3.4).

**Definition 3.0.1.** A continuous function  $\alpha : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  is a  $\mathcal{K}$ -function if it is strictly increasing and  $\alpha(0) = 0$ . It is a  $\mathcal{K}_{\infty}$ -function if it is a  $\mathcal{K}$ -function and  $\alpha(r) \rightarrow \infty$  as  $r \rightarrow \infty$ .

**Definition 3.0.2.** A continuous function  $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$  is a  $\mathcal{KL}$ -function if, for each  $t$ ,  $\beta(\cdot, t)$  is a  $\mathcal{K}$ -function with respect to  $r$ , and for each  $r$ ,  $\beta(r, \cdot)$  is decreasing with respect to  $t$ , and  $\beta(r, t) \rightarrow 0$  as  $t \rightarrow \infty$ .

**Definition 3.0.3.** The origin is globally asymptotically stable for (3.2) if, for each  $\varepsilon > 0$ , there is a  $\delta = \delta(\varepsilon) > 0$  such that

$$\|\xi\| < \delta \implies \|x^{(t)}\| < \varepsilon, \forall t \geq 0, \text{ and } \lim_{t \rightarrow \infty} x^{(t)} = 0. \tag{3.5}$$

In addition, there exists a  $\mathcal{KL}$ -function  $\beta$  such that

$$\|x^{(t)}(\xi)\| \leq \beta(\|\xi\|, t) \quad \forall \xi \in \mathcal{X}, \forall t \in \mathbf{Z}_+. \tag{3.6}$$

**Definition 3.0.4.** [52] *The discrete-time system (3.4) is input-to-state stable if there exists a  $\mathcal{KL}$ -function  $\beta$  and a  $\mathcal{K}$ -function  $\gamma$ , such that for each  $t \in \mathbf{Z}_+$ ,*

$$|e^{(t)}(\eta, x)| \leq \beta(\|\eta\|, t) + \gamma(\|x\|), \quad (3.7)$$

for all  $\eta \in \mathcal{E}$  and for all  $x \in \mathcal{X}$ .

**Definition 3.0.5.** [52] *A continuous function  $V : \mathbb{R}^n \rightarrow \mathbb{R}_{\geq 0}$  is called an ISS-Lyapunov function for the discrete-time system (3.4) if there exist  $\mathcal{K}_\infty$ -functions  $\alpha_1, \alpha_2, \alpha_3$ , and a  $\mathcal{K}$ -function  $\gamma$  such that*

$$\alpha_1(\|e^{(t)}\|) \leq V(e^{(t)}) \leq \alpha_2(\|e^{(t)}\|), \quad \forall e^{(t)} \in \mathcal{E}, \quad (3.8)$$

and

$$\begin{aligned} V(g(e^{(t)}, x^{(t)})) - V(e^{(t)}) &\leq -\alpha_3(\|e^{(t)}\|) + \gamma(\|x^{(t)}\|) \\ \forall e^{(t)} \in \mathcal{E}, \forall x^{(t)} \in \mathcal{X}. \end{aligned} \quad (3.9)$$

**Theorem 3.0.6.** *If the error system (3.4) with  $x^{(t)}$  as input is input to state stable and the origin of the discrete-time system (3.2) is globally asymptotically stable, then the origin of the cascade system (3.2) and (3.4) is globally asymptotically stable.*

*Proof.* The solutions of (3.2) and (3.4) satisfy:

$$\begin{aligned} \|x^{(t)}(\xi)\| &\leq \beta_1(\|\xi\|, t), \\ \|e^{(t)}(\eta, x)\| &\leq \beta_2(\|\eta\|, t) + \gamma(\|x^{(t)}\|), \end{aligned}$$

where  $\beta_1, \beta_2$  are  $\mathcal{KL}$ -functions. Thus,

$$\|e^{(t)}(\eta, x)\| \leq \beta_2(\|\eta\|, t) + \gamma(\beta_1(\|\xi\|, t)). \quad (3.10)$$

Let  $s^{(t)}$  denote the concatenation of the state  $x^{(t)}$  and the error state  $e^{(t)}$ , and  $\zeta$  denote the origin of this cascade system. We have  $\|x^{(t)}\| \leq \|s^{(t)}\|$  and  $\|e^{(t)}\| \leq \|s^{(t)}\|$ , and  $\|s^{(t)}\| \leq \|x^{(t)}\| + \|e^{(t)}\|$  (because for any 2 non-negative real numbers  $a$  and  $b$ ,  $\sqrt{a^2 + b^2} \leq a + b$ ). Defining  $\beta(\cdot, \cdot) = \beta_1(\cdot, \cdot) + \beta_2(\cdot, \cdot) + \gamma(\beta_1(\cdot, \cdot))$ , this yields

$$\|s^{(t)}(\zeta)\| \leq \beta(\|\zeta\|, t) \quad \forall t \in \mathbf{Z}_+. \quad (3.11)$$

It can be easily verified that  $\beta$  is a  $\mathcal{KL}$ -function.  $\square$

Two different approaches were used for finding the Lyapunov function, depending on whether the system is linear or nonlinear.

**Linear Systems.** Consider a linear discrete-time system defined by, for matrices  $A$  and  $H$ ,

$$\begin{aligned}x^{(t+1)} &= Ax^{(t)}, \\y^{(t+1)} &= Hx^{(t+1)}.\end{aligned}\tag{3.12}$$

Using the identity function as the activation function  $\sigma$  in (3.1), the error dynamics of the linear state estimator can be written as

$$\begin{aligned}e^{(t+1)} &= Ax^{(t)} - W_{xa}(W_{ay}y^{(t+1)} + W_{ax}\hat{x}^{(t)}) \\&= Ax^{(t)} - W_{xa}(W_{ay}Hx^{(t+1)} + W_{ax}(x^{(t)} - e^{(t)})) \\&= (A - W_{xa}W_{ay}HA - W_{xa}W_{ax})x^{(t)} + W_{xa}W_{ax}e^{(t)}.\end{aligned}\tag{3.13}$$

Defining  $\mathcal{A} = W_{xa}W_{ax}$  and  $\mathcal{B} = A - W_{xa}W_{ay}HA - W_{xa}W_{ax}$ , the error system is a linear system with  $x^{(t)}$  as the input:

$$e^{(t+1)} = \mathcal{A}e^{(t)} + \mathcal{B}x^{(t)}.\tag{3.14}$$

If this discrete-time system is input to state stable then there is a quadratic Lyapunov function

$$V(e) = e^T P e,\tag{3.15}$$

where  $P$  is a positive definite matrix obtained by solving

$$\mathcal{A}^T P \mathcal{A} - P + Q = 0.\tag{3.16}$$

Here,  $Q$  is a symmetric positive definite matrix. In this case, it is easy to show that both properties in Def. 3.0.5 are satisfied with  $\alpha_1(x) = \lambda_{\min}(P)x^2$ ,  $\alpha_2(x) = \lambda_{\max}(P)x^2$ ,  $\alpha_3(x) = \frac{1}{2}\lambda_{\min}(Q)x^2$ , and  $\gamma(x) = (\frac{2|\mathcal{A}^T P \mathcal{B}|^2}{\lambda_{\min}(Q)} + |\mathcal{B}^T P \mathcal{B}|^2)x^2$  [52].

**Nonlinear systems.** For nonlinear systems, a counterexample-guided method with verification provided by SMT solvers was used to synthesize the ISS-Lyapunov functions for the error system (3.4).

Inspired by [1], we use a one-hidden layer feed-forward neural network with zero bias terms for all layers to learn a Lyapunov function of the following form:

$$V(e; \theta) = \sigma(W_2 \sigma(W_1 e)),\tag{3.17}$$

where  $W_1$  and  $W_2$  are the weights for the hidden layer and the output layer respectively,  $\theta$  denotes all hyperparameters,  $\sigma$  is the activation function,  $e$  is the input vector, which is the shorthand notation for the error state at time  $t$ . Similarly,  $x$  denotes the state at time  $t$ .

We use the square function as the activation function  $\sigma$  in this network, which results in sum-of-squares-like quadratic Lyapunov functions. Then  $V(0) = 0$ . As a result, a valid input to state stability Lyapunov function is attained when properties (3.8) and (3.9) are satisfied. Its falsification constraints can then be written as a first-order logic formula over the real numbers. This yields

$$\begin{aligned} & \left( \sum_{i=1}^n e_i^2 \leq r_e \vee \sum_{i=1}^n x_i^2 \leq r_x \right) \wedge \\ & \left( (V(x) - \alpha_1(|e|) \leq 0) \vee (V(x) - \alpha_2(|e|) \geq 0) \vee \right. \\ & \left. (V(g(e, x)) - V(e) + \alpha_3(|e|) - \gamma(|x|) \geq 0) \right), \end{aligned} \quad (3.18)$$

where,  $r_e$  and  $r_x$  are the radii of the so-called valid region for the error state and the state respectively, on which we verify the condition using satisfiability-modulo-theory solvers. When the SMT solver returns UNSAT, a valid ISS Lyapunov function is obtained. Otherwise, it produces counterexamples that can be added to the training dataset of the neural network for finding Lyapunov function candidates. Due to the nature of the satisfiability-modulo-theory solvers, we typically need to exclude a small region around the origin when verifying the falsification constraints. Around the origin, the linearization of the nonlinear system dominates, and we implement the method for the linearized model, described above, to provide stability guarantees. We refer the readers to [79] for a detailed proof and algorithm.

The loss function is consistent with the falsification constraints:

$$\begin{aligned} L(\theta) = & \frac{1}{MN} \sum_{j=1}^M \sum_{i=1}^N \max \left( 0, V_\theta(g(e_i, x_j)) - V_\theta(e_i) \right. \\ & \left. + \alpha_3(|e_i|) - \gamma(|x_j|) \right) + \max \left( 0, V_\theta(e_i) - \alpha_2(|e_i|) \right) \\ & + \max \left( 0, \alpha_1(|e_i|) - V_\theta(e_i) \right). \end{aligned} \quad (3.19)$$

This is known as the positive penalty for violating the conditions in Definition 3.0.5.

### 3.1 Implementation and numerical results

The datasets were obtained by discretizing common ordinary differential equations. We use a zero-order hold discretization for linear systems. For nonlinear continuous-time state-space systems, we used the RK-45 discretization using Python’s `scipy.integrate.solve_ivp` function. The initial condition was considered Gaussian with the mean  $\hat{x}^{(0)}$  sampled uniformly from the interval  $[-1, 1] \times [-1, 1] \subset \mathbb{R} \times \mathbb{R}$  for each sequence and the known covariance. Process and measurement noises are assumed to be Gaussian with zero mean and known covariance. For the systems under consideration, the above 3 covariance matrices are assumed to be  $0.01 \times I$  where  $I$  is an identity matrix of appropriate dimensions. In addition, knowledge of the functions  $f$  and  $h$  in (2.1) is used to generate the data.

For the linear system, we consider a total of 100 sequences and 200 for nonlinear systems. The data set is divided into three parts: training, validation and testing in the ratio 80 : 10 : 10 sequences. Each of the generated sequences is independent of the other sequences in the dataset.

A custom network with forward propagation as in (3.1) is implemented. The weight matrix  $W_{ay}$  is initialized using Xavier uniform distribution while weight matrices  $W_{ya}$  and  $W_{xa}$  are initialized to be (semi) orthogonal matrices using *Pytorch*’s `torch.nn.init` module. The function  $\sigma(z)$  is chosen to be equal to  $z$  for the linear system and  $\tanh(z)$  for the nonlinear systems. The backward propagation is implemented using *PyTorch*’s `backward()` function. We consider the mean squared error loss function at time-step  $t + 1$  for each sequence

$$J_i^{(t+1)}(\phi) = \frac{1}{n} \sum_{i=1}^n (x_i^{(t+1)} - \hat{x}_{i,\phi}^{(t+1)})^2, \tag{3.20}$$

where  $1 \leq i \leq n$  and  $n$  is the number of states,  $x_i^{(t+1)}$  represents the true value of state  $x_i$  at time-step  $t + 1$  and  $\hat{x}_{i,\phi}^{(t+1)}$  represents the estimated value of state  $x_i$  at time-step  $t + 1$  by [JRN](#) where  $\phi$  denotes the weights of the network.

Hyperparameter tuning was performed for all examples individually. *Adam* optimization is used for training the network. The optimal learning rate is chosen based on the network performance in the validation dataset from the range of  $10^{-1}$  to  $10^{-4}$ . For each example, a batch size of 40 and a hidden unit size of 50 are considered. Early stopping with a fixed patience value is used to decide the number of epochs needed. In each epoch, the network performance on the validation data is used as a measure to decide whether to proceed or keep training for another epoch. The maximum number of epochs is set to 600 in case early stopping patience is not reached. For each lower validation loss value than

Table 3.1: Training hyperparameters

State space system	Time-steps	Learning rate	Patience
Mass spring	300	0.01	10
Down pendulum	300	0.01	25
r. Van der Pol	200	0.001	25

the previous, the model is saved and after training reloaded to the one corresponding to the least validation loss.

We compare the results with **KF** for the linear system as it is an optimal filter for systems with additive white Gaussian noise. For nonlinear systems with additive white Gaussian noise, the best filters are considered to be **EKF** and **UKF**. Hence, we compare our nonlinear systems results with them. Standard implementations of these 3 filters are used, see for example [111]. For **EKF**, the equilibrium point around which the systems are linearized is considered to be the origin.

We compare **JRN**, **EKF** and **UKF** graphically using average error at time-step  $t$  over all features and all test sequences

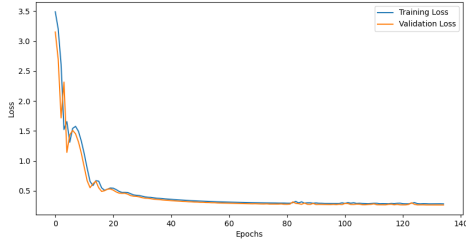
$$\text{Error}(t) = \frac{1}{m_{test}n} \sum_{k=1}^{m_{test}} \sum_{i=1}^n (x_i^{(t)[k]} - \hat{x}_i^{(t)[k]})^2, \quad (3.21)$$

where  $m_{test}$  is the number of test sequences,  $n$  is the number of states,  $x_i^{(t)[k]}$  represents the true value of state  $x_i$  at time-step  $t$  for the  $k^{th}$  sequence and  $\hat{x}_i^{(t)[k]}$  represents the estimated value of state  $x_i$  at time-step  $t$  for the  $k^{th}$  sequence. We also use root mean square error to compare different methods. It is calculated as

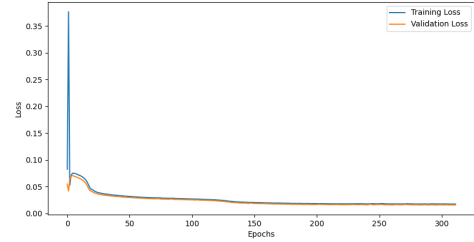
$$\text{root MSE} = \sqrt{\frac{1}{m_{test}nT} \sum_{k=1}^{m_{test}} \sum_{j=1}^T \sum_{i=1}^n (x_i^{(j)[k]} - \hat{x}_i^{(j)[k]})^2}, \quad (3.22)$$

where  $m_{test}$  is the number of test sequences,  $T$  is the number of time-steps in each sequence,  $n$  is the number of states,  $x_i^{(j)[k]}$  represents the true value of state  $x_i$  at time-step  $j$  for the  $k^{th}$  sequence and  $\hat{x}_i^{(j)[k]}$  represents the estimated value of state  $x_i$  at time-step  $j$  for the  $k^{th}$  sequence.

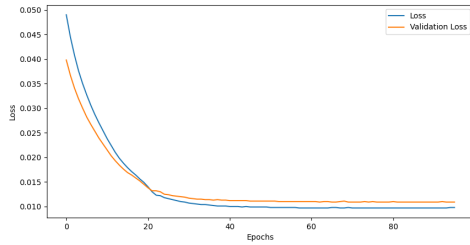
We illustrate the proposed method with 3 discrete-time systems. The ODEs considered are mass-spring damper system, down pendulum and reversed Van der Pol oscillator. The



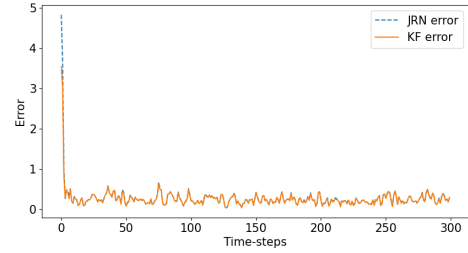
(a) Training and validation loss for mass-spring system



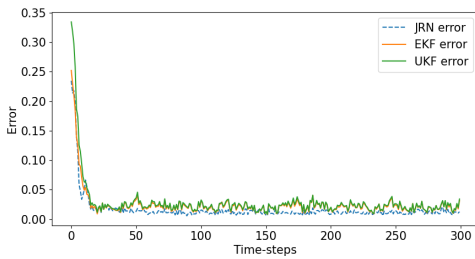
(b) Training and validation loss for down pendulum



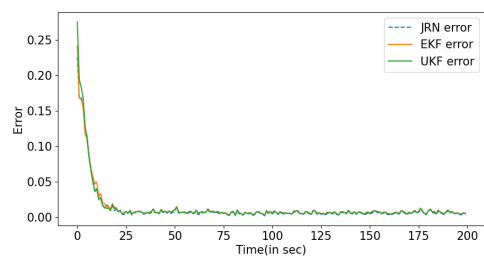
(c) Training and validation loss for reversed Van der Pol oscillator



(d) Average error at each timestep for mass-spring system



(e) Average error at each timestep for down pendulum



(f) Average error at each timestep for reversed Van der Pol oscillator

Figure 3.2: Figures 3.2a, 3.2b and 3.2c show the training and validation values for JRN based state estimation. Figures 3.2d 3.2e and 3.2f show average errors at each time-step for JRN, EKF and UKF state estimates.

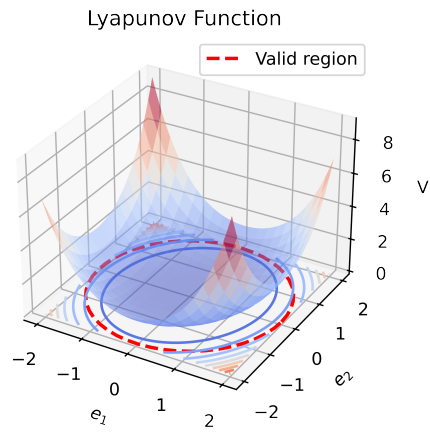


Figure 3.3: Learned ISS Lyapunov function for a down pendulum.

Table 3.2: Parameters

State space system	Parameters
Mass spring	$m = 10 \text{ kg}, b = 6 \text{ kg/sec}, k = 800 \text{ kg/sec}^2$
Down pendulum	$m = 2 \text{ kg}, b = 0.9 \text{ kg/sec}, l = 1m.$

measurement vector at time-step  $t$  for each of the systems is  $y^{(t)} = x_1^{(t)} + \nu^{(t)}$  where  $x_1(t)$  refers to the position and  $\nu^{(t)}$  is the measurement noise at time-step  $t$ . Important parameters for each of these systems are summarised in Tables 3.1 and 3.2. A remote Ubuntu server was used for all calculations. The codes used are available on Github at <https://github.com/avneetkaur96/JRNs-for-state-estimation>.

The performance of the **JRN** and the **KF** for mass-spring system state estimation is summarized in Figures 3.2d and 3.2a. As shown in Figure 3.2d, both the **JRN** and **KF** achieve rapid error reduction within the first few time steps, with the **KF** maintaining slightly lower error overall as expected due to its optimal nature. The **JRN** closely tracks the **KF**'s performance after the initial steps, indicating effective learning of the system dynamics. The difference of estimation in the first few timesteps can be attributed to the fact that the **KF** is provided with an estimate of the initial condition while the **JRN** has no information of the initial condition. The training and validation loss curves for the **JRN** (Figure 3.2a) demonstrate fast convergence, with loss stabilizing after approximately 40 epochs. The close alignment of training and validation loss throughout training indicates good generalization and minimal overfitting. The root **MSE** on the test set is 0.5132 for the **KF** and 0.5268 for the **JRN** (Table 3.3, confirming that the **JRN** provides comparable estimation accuracy. The model was restored to its state at the best epoch for testing. For this system, the origin is asymptotically stable. We then implement the method for stability of linear systems. Setting  $Q = I$ , it is easy to establish that  $P = \begin{bmatrix} 45.504 & -3.5737 \\ -3.5737 & 1.4461 \end{bmatrix}$ . With this ISS-Lyapunov function, by Theorem 3.0.6, both the original system and the error system are asymptotically stable. This proves the stability of the NN estimator error.

As shown in Figures 3.2e and 3.2f, all three approaches exhibit a rapid reduction in error within the first 25 time-steps, stabilizing thereafter. **JRN** achieves the lowest estimation error over time, outperforming both **EKF** and **UKF** for the down pendulum. The errors for both systems and for all methods stabilize after approximately 50 time-steps, with

Table 3.3: Root mean square error (root **MSE**) for **EKF**, **UKF** and **JRN**

State space system	Estimator		
	<i><b>EKF</b></i>	<i><b>UKF</b></i>	<i><b>JRN</b></i>
Mass spring	<b>0.5132</b>	-	0.5268
Down pendulum	0.1594	0.1693	<b>0.1298</b>
r. Van der Pol	0.1153	0.1158	<b>0.1102</b>

Table 3.4: Best epoch and training time for **JRN**

State space system	Best Epoch	Training time(in sec)
Mass spring	135	20
Down pendulum	313	101
r. Van der Pol	96	32

**JRN** maintaining a consistently lower error throughout for the down pendulum. Figures 3.2b and 3.2c demonstrate the convergence of the **JRN** training process. Both training and validation losses decrease steadily, indicating good generalization and absence of overfitting. The **JRN** achieves the lowest root **MSE** compared to the **EKF** and **UKF** for both the nonlinear systems, indicating improved accuracy (See Table 3.3). These results highlight the effectiveness and efficiency of the **JRN** in nonlinear state estimation tasks. For both nonlinear systems, an ISS Lyapunov function was learned using a neural network and verified by an SMT solver. For the down pendulum, this function is shown in Fig. 3.3, verified by an SMT solver, dReal [31], on a  $[-2, 2] \times [-2, 2]$  for the error states. In this case, we used  $\alpha_1(\cdot) = \alpha_3(\cdot) = 0.01|\cdot|$  and  $\alpha_2(\cdot) = \gamma(\cdot) = 100|\cdot|$ . This provides the asymptotic stability for the error dynamics by Theorem 3.0.6.

## 3.2 Conclusions and limitations

The **JRN**-based estimator achieved results as good as **KF** for the linear system and better than **EKF** and **UKF** for the chosen nonlinear systems. Although training time is a drawback in the case of neural networks, the testing time of **JRN**-based estimators is much lower than the classical approaches of **UKF** and **EKF**.

The main result is the verification of the stability of the error dynamics of the estimator based on **JRN** using an input-to-state stability approach, taking advantage of the structural properties of an unbiased **JRN** and recent results in learning Lyapunov functions. For linear systems, quadratic Lyapunov equations can be computed analytically for the error dynamics to provide stability. For nonlinear systems, a counter-example guided method was used to learn an ISS Lyapunov function. The linear approach is used near equilibrium. The effectiveness of the method was illustrated with three examples. Notably, **JRN**-based estimation led to a notable decrease in run time which would be beneficial for real-time applications.

The main limitation in using [JRNs](#) is the issue of vanishing or exploding gradients during training for longer sequences.

# Chapter 4

## State estimation using Jordan long short-term memory network

In this work, we use a neural network structure that mimics the structure of a dynamical system to construct an estimator. While feedforward neural networks[8] do not link outputs or the hidden states at the previous time-step to the one at the next time-step, recurrent neural networks have varying structures linking outputs and/or hidden states at different time-steps. This property is of great importance, since in dynamical systems, the state at the previous time-step, plus any inputs, determines the state at the next time-step. Another advantage of recurrent neural networks is that they can have a nonlinear activation function which is helpful for estimating the states of nonlinear systems. Since Jordan recurrent networks have the previous output connected as an input to the next time-step, this suggests that they would be a good choice for state estimation of dynamical systems. However, when it comes to longer sequences, training them can be extremely time-consuming(see for example, [84]). This motivates the use of long short-term memory networks [44]. Furthermore, choosing the right type of connections in a network is important. This can help the network to converge faster leading to reduced training time. Thus, in this paper, we present an extension of the Jordan architecture to long short-term memory networks and investigate their utility for state estimation. We present a universal approximation theorem for Jordan-based long short-term memory networks(JLSTMs) and compare the training time of JLSTM with ELSTM for several discrete-time state space systems. We observe that JLSTMs take significantly lower time to train as compared to ELSTMs. We also compare the long short-term memory architectures with an EKF for nonlinear systems and KF for a linear system. Both neural network-based estimators compare well to traditional Kalman filters.

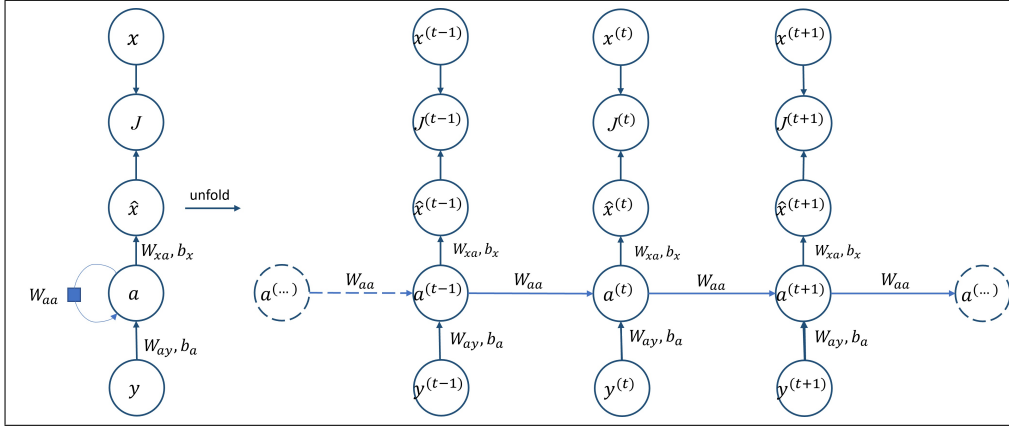


Figure 4.1: The structure of ERN for state estimation.

We combine the idea of a state estimation algorithm with the tools of machine learning. We discuss the structure of an ERN and a JRN followed by a universal approximation theorem for state estimation using JRN. The JRN is more closely related to the structure of a dynamical system than ERN which suggests that it might be more appropriate for state estimation. The ELSTM framework is thus modified to a JLSTM network and shown to have reduced training time. Both neural network-based estimators compare well to the traditional Kalman filters. However, the JLSTM network takes a shorter training time compared to an ELSTM to achieve similar performance.

This work considers noisy discrete-time state space system 2.1. The aim is to estimate the state  $x^{(t+1)}$  based on new measurement  $y^{(t+1)}$ , previous state estimate  $\hat{x}^{(t)}$ , and some information on the distribution of the initial condition  $x^{(0)}$ .

Below is the definition of activation function used for this work.

**Definition 4.0.1** (Activation function). *A function  $\sigma(\cdot) : \mathbb{R} \rightarrow [a, b]$  is called an activation function, if  $\sigma$  is monotonically increasing,  $\lim_{z \rightarrow -\infty} \sigma(z) = a$  and  $\lim_{z \rightarrow \infty} \sigma(z) = b$ .*

A ERN has been used for state estimation. In ERN, the hidden layer at the previous time-step connects to the hidden layer at the next time-step(Fig. 4.1). ERNs are proven to be universal approximators for dynamical systems in [106]. For state estimation, one can choose the inputs of the ERN at time-step  $t$  to be the measurements at time-step  $t$  and the outputs of the ERN to be the estimated state at time-step  $t$ . Thus, an ERN acts

like a filter. The forward propagation of an **ERN** for state estimation is

$$\begin{aligned} a^{(t)} &= \sigma(W_{ay}y^{(t)} + W_{aa}a^{(t-1)} + b_a) \\ \hat{x}^{(t)} &= W_{xa}a^{(t)} + b_x \end{aligned} \tag{4.1}$$

where  $\sigma(\cdot)$  is the activation function and  $\hat{x}^{(t)}$  is the estimated state vector at time-step  $t$ . The hidden layer vector at time-step  $t$  is  $a^{(t)}$ . The measurement vector at time-step  $t$  is denoted by  $y^{(t)}$ . The weights and biases of the network are represented by  $W_{ay}$ ,  $W_{aa}$ ,  $W_{xa}$ ,  $b_a$  and  $b_x$  respectively. The approximation theorem for **ERNs** of a dynamical system presented in [106] can be extended to estimation with slight modifications. Since the proof is very similar to that in [106], it is omitted.

In a **JRN**, instead of using the previous hidden vector in calculation of the updated estimate, the estimate at the previous time-step  $t$  is to update the state (Fig. 3.1). Thus, a Jordan-based recurrent neural network structure mimics the structure of a dynamical system (2.1). This suggests that training and accuracy of an estimator may be improved over other structures by using a **JRN**. The forward propagation of a **JRN** for state estimation (Fig.3.1) is

$$\begin{aligned} a^{(t)} &= \sigma(W_{ay}y^{(t)} + W_{ax}\hat{x}^{(t-1)} + b_a) \\ \hat{x}^{(t)} &= W_{xa}a^{(t)} + b_x \end{aligned} \tag{4.2}$$

where  $\sigma(\cdot)$  is the activation function and  $\hat{x}^{(t)}$  is the estimated state vector at time-step  $t$ . The hidden layer vector at time-step  $t$  is given by  $a^{(t)}$ . The weights and biases of the network are represented by  $W_{ay}$ ,  $W_{ax}$ ,  $W_{xa}$ ,  $b_a$  and  $b_x$  respectively.

It will now be shown that **JRN**s are universal approximators for an estimator. We first recall the definitions and main results first presented in [46] and later mentioned in [106].

**Definition 4.0.2.** Let  $A^n, n \in \mathbb{N}$  be the set of all affine functions  $A(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}$  defined as  $A(z) = w \cdot z - b$  where  $w, z \in \mathbb{R}^n, b \in \mathbb{R}$  and ‘ $\cdot$ ’ denotes dot product.

**Definition 4.0.3.** For any Borel-measurable function  $\sigma(\cdot) : \mathbb{R} \rightarrow \mathbb{R}$  and  $n \in \mathbb{N}$  define

$$\begin{aligned} \Sigma^n(\sigma) &= \{(NN : \mathbb{R}^N \rightarrow \mathbb{R}) : NN(z) = \sum_{j=1}^J \theta_j \sigma(A_j(z)), \\ & z \in \mathbb{R}^n, \theta_j \in \mathbb{R}, A_j \in A^n, J \in \mathbb{N}\}. \end{aligned} \tag{4.3}$$

The function class  $\Sigma^n(\sigma)$  can be written in matrix form as

$$NN(z) = \theta \sigma(Wz - b),$$

where  $x \in \mathbb{R}^n, \theta, b \in \mathbb{R}^{\tilde{h}}$  and  $W \in \mathbb{R}^{\tilde{h} \times n}$ . Here function  $\sigma$  is applied component-wise. Similarly, the for function class  $JRN^{m,n}(\sigma)$ ,  $\sigma$  is defined component-wise.

Let  $\mathcal{M}^n$  and  $\mathcal{C}^n$  be the set of all Borel-measurable and continuous functions (both from  $\mathbb{R}^n \rightarrow \mathbb{R}$ ) respectively. Let the Borel  $\sigma$  algebra of  $\mathbb{R}^n$  be denoted by  $\mathbb{B}^n$ . Thus,  $(\mathbb{R}^n, \mathbb{B}^n)$  refers to the  $n$ -dimensional Borel measurable space.

Clearly, for every continuous Borel-measurable function  $\sigma$ ,

$$\Sigma^n(\sigma) \subset \mathcal{C}^n \subset \mathcal{M}^n.$$

**Definition 4.0.4.** Let  $(X, \rho)$  be a metric space and  $U, T \subseteq X$ . Then  $U$  is  $\rho$ -dense in  $T$ , if  $\forall \epsilon > 0$  and  $\forall t \in T, \exists u \in U$ , such that  $\rho(u, t) < \epsilon$ .

**Definition 4.0.5.** A subset  $U$  of  $\mathcal{C}^n$  is said to be uniformly dense on a compact domain in  $\mathcal{C}^n$  if for every compact subset  $K \subseteq \mathbb{R}^n, U$  is  $\rho_K$ -dense in  $\mathcal{C}^n$ , where for  $g_1, g_2 \in \mathcal{C}^n$ ,

$$\rho_K(g_1, g_2) \equiv \sup_{x \in K} |g_1(x) - g_2(x)|.$$

**Definition 4.0.6.** Let  $\mu$  be a probability measure on  $(\mathbb{R}^n, \mathbb{B}^n)$ . Functions  $g_1, g_2 \in \mathcal{M}^n$  are said to be  $\mu$ -equivalent if  $\mu\{x \in \mathbb{R}^n : g_1(x) = g_2(x)\} = 1$ .

**Definition 4.0.7.** Define the metric  $\rho_\mu : \mathcal{M}^n \times \mathcal{M}^n \rightarrow \mathbb{R}^+$  on  $(\mathbb{R}^n, \mathbb{B}^n)$  as

$$\rho_\mu(g_1, g_2) = \inf\{\epsilon > 0 : \mu\{x \in \mathbb{R}^n : |g_1(x) - g_2(x)| > \epsilon\} < \epsilon\} \quad (4.4)$$

where  $\mu$  is a probability measure on  $(\mathbb{R}^n, \mathbb{B}^n)$ .

Note that if functions  $g_1$  and  $g_2$  are  $\mu$ -equivalent, then  $\rho_\mu(g_1, g_2) = 0$ .

**Theorem 4.0.8.** [46] (Universal approximation theorem for feed-forward neural networks) For any activation function  $\sigma : \mathbb{R} \rightarrow [0, 1]$ , any dimension  $n$  and any probability measure  $\mu$  on  $(\mathbb{R}^n, \mathbb{B}^n), \Sigma^n(\sigma)$  is uniformly dense on a compact domain in  $\mathcal{C}^n$  and  $\rho_\mu$ -dense in  $\mathcal{M}^n$ .

Let the set of all continuous functions from  $\mathbb{R}^m$  to  $\mathbb{R}^n, m, n \in \mathbb{N}$ , be denoted by  $\mathcal{C}^{m,n}$  and the one of (Borel-)measurable functions from  $\mathbb{R}^m$  to  $\mathbb{R}^n$  by  $\mathcal{M}^{m,n}$  respectively.

**Corollary 4.0.9.** [46] Theorem 4.0.8 holds for the approximation of functions  $\mathcal{C}^{m,n}$  and  $\mathcal{M}^{m,n}$  by the extended function class  $\Sigma^{m,n}$ . Thereby the metric  $\rho_\mu$  is replaced by

$$\rho_\mu^n := \sum_{l=1}^n \rho_\mu(f_l, g_l).$$

We now introduce the following definition to extend the universal approximation theorem for feedforward neural networks to universal approximation of internal states of a state-space system by Jordan recurrent networks.

**Definition 4.0.10.** For any Borel-measurable function  $\sigma$  and  $m, n \in \mathbb{N}$  define  $JRN^{m,n}(\sigma)$  as the class of functions

$$x^{(t+1)} = W_{xa}\sigma(W_{ay}y^{(t)} + W_{ax}x^{(t)} - b)$$

where  $x^{(t)} \in \mathbb{R}^n, y^{(t)} \in \mathbb{R}^m$ , for  $t = 1, 2, \dots, T$ . Also,  $W_{ax} \in \mathbb{R}^{\tilde{h} \times n}, W_{ay} \in \mathbb{R}^{\tilde{h} \times m}, W_{xa} \in \mathbb{R}^{n \times \tilde{h}}$  are weight matrices and  $\theta \in \mathbb{R}^{\tilde{h}}$  is the corresponding bias where  $\tilde{h}$  is the dimension of the hidden state.

**Theorem 4.0.11.** (Universal approximation theorem of state estimators using Jordan recurrent neural networks(JRNs)) Let  $F(\cdot) : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$  be continuous and  $h(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^m$  be continuous, the states  $x^{(t)} \in \mathbb{R}^n$ , and the measurements  $y^{(t)} \in \mathbb{R}^m$  where  $t = 1, 2, \dots, T$  for finite  $T$ . Then, any state estimator of the form

$$\bar{x}^{(t+1)} = F(\bar{x}^{(t)}, y^{(t)})$$

for the discrete-time dynamical system

$$\begin{aligned} x^{(t+1)} &= f(x^{(t)}) \\ y^{(t)} &= h(x^{(t)}) \end{aligned} \tag{4.5}$$

can be approximated by an element of the function class  $JRN^{m,n}(\sigma)$  with an arbitrary accuracy, where  $\sigma$  is a continuous activation function.

*Proof.* We want to conclude that the estimator

$$\bar{x}^{(t+1)} = F(\bar{x}^{(t)}, y^{(t)})$$

can be approximated by a JRN of the form

$$\hat{x}^{(t+1)} = W_{xa}\sigma(W_{ay}y^{(t)} + W_{ax}\hat{x}^{(t)} - b) \quad \forall t = 1, 2, \dots, T$$

with an arbitrary accuracy, where  $W_{xa}, W_{ax}$  and  $W_{ay}$  are appropriate weight matrices,  $b$  refers to bias and  $\sigma$  is the required activation function. We will begin by using the universal approximation theorem for feed-forward neural networks and then use the continuity of  $F$  to reach the conclusion.

Choose any  $\epsilon > 0$  and let  $K \subseteq \mathbb{R}^n \times \mathbb{R}^m$  be a compact set which contains  $(\bar{x}^{(t)}, y^{(t)})$  and  $(\hat{x}^{(t)}, y^{(t)})$ ,  $\forall t = 1, 2, \dots, T$ . From Theorem 4.0.8 and Corollary 4.0.9, we know that for any measurable function  $F(\bar{x}^{(t)}, y^{(t)}) : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$  and arbitrary  $\epsilon > 0$ , a function

$$NN(\bar{x}^{(t)}, y^{(t)}) = W_{xa}\sigma(W_{ay}y^{(t)} + W_{ax}\bar{x}^{(t)} - b)$$

with weight matrices  $W_{xa} \in \mathbb{R}^{n \times \tilde{h}}$ ,  $W_{ax} \in \mathbb{R}^{\tilde{h} \times n}$ ,  $W_{ay} \in \mathbb{R}^{\tilde{h} \times m}$ , bias  $b \in \mathbb{R}^{\tilde{h}}$ , and a component-wise applied continuous activation function  $\sigma(\cdot) : \mathbb{R}^{\tilde{h}} \rightarrow \mathbb{R}^{\tilde{h}}$  exists, such that  $\forall t = 1, 2, \dots, T$ ,

$$\sup_{\bar{x}^{(t)}, y^{(t)}} \|F(\bar{x}^{(t)}, y^{(t)}) - NN(\bar{x}^{(t)}, y^{(t)})\|_{\infty} < \epsilon. \quad (4.6)$$

Here,  $\tilde{h}$  refers to the dimension of the hidden state.

Since  $F$  is continuous and  $T$  is finite,

$$\|\bar{x}^{(t+1)} - \hat{x}^{(t+1)}\|_{\infty} < \epsilon$$

with the recurrent neural network architecture

$$\hat{x}^{(t+1)} = W_{xa}\sigma(W_{ay}y^{(t)} + W_{ax}\hat{x}^{(t)} - b) \quad \forall t = 1, 2, \dots, T.$$

This completes the proof. □

Thus, [JRN](#)s are a suitable choice for state estimation.

While simple recurrent neural networks can be trained to perform as well as a [KF](#) for linear systems and better than the [EKF](#) for nonlinear systems (see [65]), it has been observed that for complex problems and long-term predictions, long short-term memory networks are generally a better choice as they help to resolve the vanishing gradient problem (see, for example, [84]) and tend to converge faster.

Each long short-term memory network consists of a cell. In order to handle long-term dependencies, several gates are introduced. The forget gate decides the information that should not be carried forward to the next time-step. The input gate determines the new information that needs to be remembered in the cell state. The output gate decides what information will be output from the cell state. A more detailed explanation of these gates for [ELSTM](#) can be found in, for example, [84].

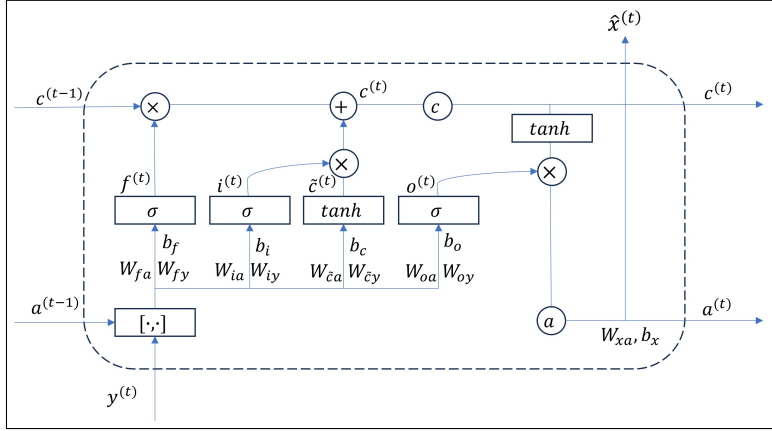


Figure 4.2: The structure of **ELSTM** for state estimation.

The forward propagation equations of **ELSTM** (see Fig. 4.2) for state estimation are

$$\begin{aligned}
 f^{(t)} &= \sigma(W_{fy}y^{(t)} + W_{fa}a^{(t-1)} + b_f) \\
 i^{(t)} &= \sigma(W_{iy}y^{(t)} + W_{ia}a^{(t-1)} + b_i) \\
 o^{(t)} &= \sigma(W_{oy}y^{(t)} + W_{oa}a^{(t-1)} + b_o) \\
 \tilde{c}^{(t)} &= \tanh(W_{cy}y^{(t)} + W_{ca}a^{(t-1)} + b_{\tilde{c}}) \\
 c^{(t)} &= f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)} \\
 a^{(t)} &= o^{(t)} \odot \tanh(c^{(t)}) \\
 \hat{x}^{(t)} &= W_{xa}a^{(t)} + b_x
 \end{aligned} \tag{4.7}$$

where  $f^{(t)}$ ,  $i^{(t)}$  and  $o^{(t)}$  are the forget, input and output gate's activation vectors respectively. Also,  $\sigma(\cdot)$  is the recurrent activation function. The cell state vector and the cell input activation vector are represented by  $c^{(t)}$  and  $\tilde{c}^{(t)}$  respectively. The hidden layer vector is represented by  $a^{(t)}$ . The weight matrices and biases are represented by  $W_{fy}$ ,  $W_{fa}$ ,  $W_{iy}$ ,  $W_{ia}$ ,  $W_{oy}$ ,  $W_{oa}$ ,  $W_{cy}$ ,  $W_{ca}$ ,  $W_{xa}$ ,  $b_f$ ,  $b_i$ ,  $b_o$  and  $b_{\tilde{c}}$ . The symbol  $\odot$  represents element-wise product.

We now introduce here a new structure called **JLSTM** for state estimation. The forward

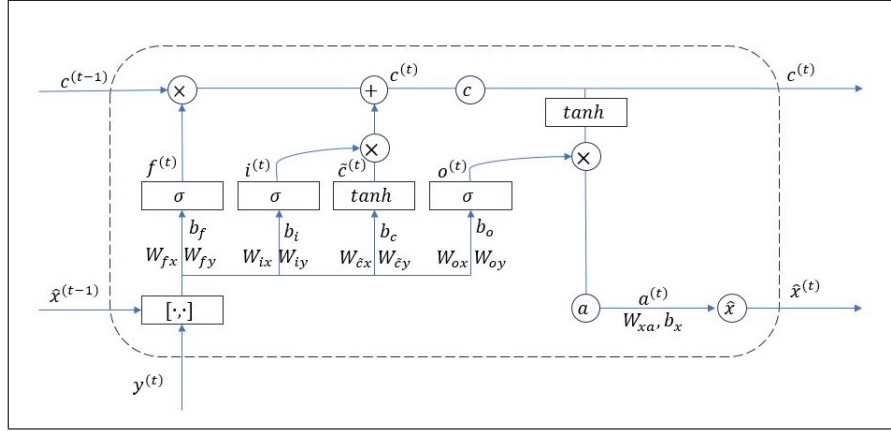


Figure 4.3: The structure of **JLSTM** for state estimation.

propagation equations of **JLSTM** (see Fig. 4.3) for state estimation will be:

$$\begin{aligned}
 f^{(t)} &= \sigma(W_{fy}y^{(t)} + W_{fx}\hat{x}^{(t-1)} + b_f) \\
 i^{(t)} &= \sigma(W_{iy}y^{(t)} + W_{ix}\hat{x}^{(t-1)} + b_i) \\
 o^{(t)} &= \sigma(W_{oy}y^{(t)} + W_{ox}\hat{x}^{(t-1)} + b_o) \\
 \tilde{c}^{(t)} &= \tanh(W_{\tilde{c}y}y^{(t)} + W_{\tilde{c}x}\hat{x}^{(t-1)} + b_{\tilde{c}}) \\
 c^{(t)} &= f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)} \\
 a^{(t)} &= o^{(t)} \odot \tanh(c^{(t)}) \\
 \hat{x}^{(t)} &= W_{xa}a^{(t)} + b_x
 \end{aligned} \tag{4.8}$$

where  $f^{(t)}$ ,  $i^{(t)}$  and  $o^{(t)}$  are the forget, input and output gate's activation vectors, respectively, and function similar to the **ELSTM** except that they are now connected to the previous output layer instead of the previous hidden layer. The function  $\sigma(\cdot)$  is the recurrent activation function. The cell state vector and the cell input activation vector are represented by  $c^{(t)}$  and  $\tilde{c}^{(t)}$  respectively. The hidden layer vector is represented by  $a^{(t)}$ . The weight matrices and biases are represented by  $W_{fy}$ ,  $W_{fx}$ ,  $W_{iy}$ ,  $W_{ix}$ ,  $W_{oy}$ ,  $W_{ox}$ ,  $W_{\tilde{c}y}$ ,  $W_{\tilde{c}x}$ ,  $W_{xa}$ ,  $b_f$ ,  $b_i$ ,  $b_o$  and  $b_{\tilde{c}}$ . The symbol  $\odot$  represents element-wise product. The connections in a **JLSTM** resemble the connections in a **JRN**. In particular, the previous output  $\hat{x}^{(t-1)}$  is fed back into the network at the next time-step as can be seen in (4.8) and Fig. 4.3.

Note that simple recurrent neural networks are a special case of long short term memory networks where the gates are not involved. The introduction of  $f^{(t)}$ ,  $i^{(t)}$ ,  $o^{(t)}$ ,  $\tilde{c}^{(t)}$  and  $c^{(t)}$  in

(4.7), (4.8), Fig. 4.2 and Fig. 4.3 is to handle long term dependencies better. Thus, the universal approximation theorems discussed in the previous section are valid.

## 4.1 Implementation

The fact that the functions  $f$  and  $h$  are known in (2.1) was used in data generation. We considered sequences starting from time 0. Gaussian initial conditions were sampled over an interval with covariance of  $(0.01) \times I$  where  $I$  is an identity matrix of appropriate dimensions. Gaussian process and measurement noises with zero mean and known covariance were sampled for each time-step. This was done for each sequence in the data set. Measurement and process noise covariance were considered to be  $(0.01) \times I$  where the matrix  $I$  is of appropriate dimensions. The number of hidden units considered were 50 for both networks. A total of 100 sequences were generated. These were divided into three parts: training, validation and testing sequences in the ratio 80 : 10 : 10. The testing dataset from this has been labeled as testing dataset 1. The mean of initial conditions were sampled from a fixed interval specific to different examples for this dataset. Another testing dataset was generated with mean of initial conditions sampled from an interval outside the interval used to generate testing dataset 1. This dataset has been labeled testing dataset 2. This dataset is useful for verifying if the network was able to learn the model well.

ELSTM and JLSTM with forward propagation as in (4.7) and (4.8) respectively are implemented. The weight matrices  $W_{fy}, W_{iy}, W_{oy}, W_{\tilde{c}y}, W_{fa}, W_{ia}, W_{oa}, W_{\tilde{c}a}, W_{fx}, W_{ix}, W_{ox}$  and  $W_{\tilde{c}x}$  are initialised using Glorot uniform distribution while the weight matrix  $W_{xa}$  is initialised to be an orthogonal matrix using *Tensorflow's tensorflow.keras.initializers* module. The recurrent activation function  $\sigma(z)$  is chosen to be to be *sigmoid*( $z$ ). The backward propagation is implemented using *Tensorflow's apply\_gradients* module.

The loss function at time-step  $t$  for each sequence is

$$J^{(t)}(\phi) = \frac{1}{n} \sum_{i=1}^n (x_i^{(t)} - \hat{x}_{i,\phi}^{(t)})^2, \quad (4.9)$$

where  $1 \leq i \leq n$  and  $n$  is the number of states,  $x_i^{(t)}$  represents the true value of state  $x_i$  at time-step  $t$  and  $\hat{x}_{i,\phi}^{(t)}$  represents the estimated value of state  $x_i$  at time-step  $t$  by the respective network where  $\phi$  denotes hyperparameters of the network. This is called the minimum-variance cost function and was considered because we want the estimated state to be as close to the true state as possible. Adam optimization was used to train the

networks. The number of hidden units and the maximum number of epochs were kept same for a particular example for both networks to make sure the comparison is accurate. Early stopping with a fixed patience value (the same for both networks) to decide the number of epochs needed is used. In each epoch, the network performance on validation data was used as the measure to decide whether to proceed or keep training for another epoch. The learning rate is chosen from the set  $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$  so that both networks achieve a normalised mean square error on test data with a maximum difference of 0.01 on test dataset 1.

We tested **JLSTM** for 3 different examples and compared the results with **ELSTM** and **KF** for the linear system and **EKF** for the nonlinear systems. Standard implementations of **KF** and **EKF** are used; see, for example, [111]. We compare **JLSTM**, **ELSTM** and **KF/EKF** using average error at time  $t$  over all features and all test sequences:

$$\text{Error}(t) = \frac{1}{m_{test}n} \sum_{k=1}^{m_{test}} \sum_{i=1}^n (x_i^{(t)[k]} - \hat{x}_i^{(t)[k]})^2, \quad (4.10)$$

where  $m_{test}$  is the number of test sequences,  $n$  is the number of states,  $x_i^{(t)[k]}$  represents the true value of state  $x_i$  at time-step  $t$  for the  $k^{th}$  sequence and  $\hat{x}_i^{(t)[k]}$  represents the estimated value of state  $x_i$  at time-step  $t$  for the  $k^{th}$  sequence. We also used normalised **MSE** to compare different methods. It is calculated as

$$\text{normalised MSE} = \frac{1}{m_{test}nT} \sum_{k=1}^{m_{test}} \sum_{j=1}^T \sum_{i=1}^n (x_i^{(j)[k]} - \hat{x}_i^{(j)[k]})^2, \quad (4.11)$$

where  $m_{test}$  is the number of test sequences,  $T$  is the number of time-steps in each sequence,  $n$  is the number of states,  $x_i^{(j)[k]}$  represents the true value of state  $x_i$  at time-step  $j$  for the  $k^{th}$  sequence and  $\hat{x}_i^{(j)[k]}$  represents the estimated value of state  $x_i$  at time-step  $j$  for the  $k^{th}$  sequence. To keep the comparison fair, the **KF** and **EKF** were not changed for testing dataset 2.

*Tensorflow* was used for the implementation of both networks. Codes were run on a T4 GPU.

We first considered a zero-order hold time discretization with  $\Delta t = 0.1s$  of a system of

10 connected springs:

$$\begin{aligned}
m_1 \ddot{x}_1(t) &= -k_1 x_1(t) + k_2(x_2(t) - x_1(t)) \\
&\quad - d_1 \dot{x}_1(t) + d_2(\dot{x}_2(t) - \dot{x}_1(t)) \\
m_2 \ddot{x}_1(t) &= -k_2(x_2(t) - x_1(t)) + k_3(x_3(t) - x_2(t)) \\
&\quad - d_2(\dot{x}_2(t) - \dot{x}_1(t)) + d_3(\dot{x}_3(t) - \dot{x}_2(t)) \\
&\quad \vdots \\
m_9 \ddot{x}_9(t) &= -k_9(x_9(t) - x_8(t)) + k_{10}(x_{10}(t) - x_9(t)) \\
&\quad - d_9(\dot{x}_9(t) - \dot{x}_8(t)) + d_{10}(\dot{x}_{10}(t) - \dot{x}_9(t)) \\
m_{10} \ddot{x}_{10}(t) &= -k_{10}(x_{10}(t) - x_9(t)) - d_{10}(\dot{x}_{10}(t) - \dot{x}_9(t))
\end{aligned} \tag{4.12}$$

where  $m_i = 10$ ,  $d_i = 6$  and  $k_i = 800$  for all  $i$ . The state vector consists of 20 discretized states, the position  $x_i(t)$  and the velocity  $\dot{x}_i(t)$  for each of the 10 springs. Process noise is added to each state after discretization. The discretised measurement vector at time-step  $t$  is given by

$$y^{(t)} = \begin{bmatrix} x_1^{(t)} \\ x_2^{(t)} \\ \vdots \\ x_{10}^{(t)} \end{bmatrix} + \nu^{(t)}$$

where  $x_i^{(t)}$  is the discretised state representing position of spring  $i$  and  $\nu^{(t)}$  is the measurement noise at time-step  $t$ . We considered sequences of length 500 starting at time 0 seconds. Gaussian initial conditions for each sequence were chosen randomly from  $[-1, 1]^{20}$  with a covariance of  $(0.01) \times I_{20}$  where  $I_k$  represents an identity matrix of order  $k$  for the generation of training, validation and testing dataset 1. All noises were assumed to be Gaussian with zero mean and known covariance. Measurement and process noise covariance were considered to be  $(0.01) \times I$  where the matrix  $I$  is of appropriate dimensions. A total of 100 sequences were considered with a batch size of 10. The number of hidden units considered were 50. Early stopping was implemented with a patience of 50. The maximum number of epochs considered was 8000. Training using Adam optimization with a learning rate of  $10^{-3}$  for **ELSTM** as well as **JLSTM** achieved a close normalised **MSE** value for both networks. While the **ELSTM** and **JLSTM** have almost equal errors at all time-steps (see Fig. 4.4), the **KF** has the largest error at the initial time-step but converges quickly. For testing outside the initial interval of mean of initial condition (testing dataset 2), in this example, we used the interval  $[1, 1.5]^{20} \sim \mathbb{R}^{20}$  to sample the mean of the initial conditions. As can be seen from Fig. 4.5, the **ELSTM** and **JLSTM** converge slower than the **KF**. This

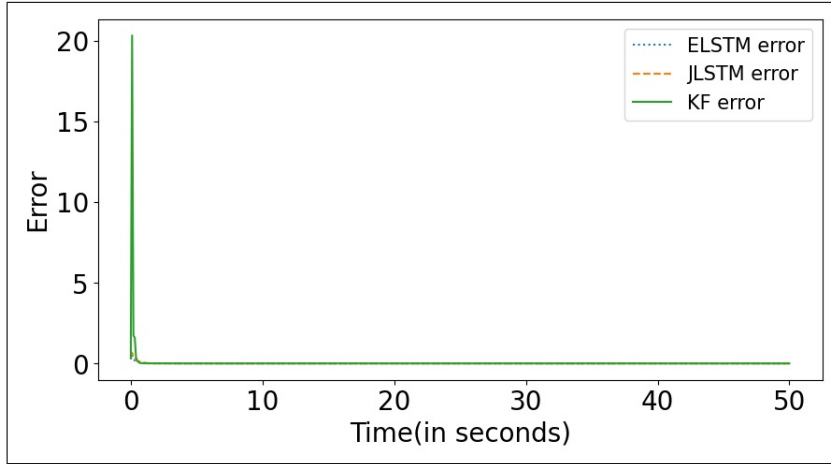


Figure 4.4: Average errors over 10 test sequences for 50 seconds of 10 connected springs with a noisy Gaussian initial condition for **KF**, **JLSTM** and **ELSTM** based state estimates.

is not surprising as theory shows the **KF** to be the best estimator for a linear system. However, the closeness of the errors of the **ELSTM** and **JLSTM** validates the training of the networks.

We then considered an RK-45 discretization with  $\Delta t = 0.01s$  of the down pendulum:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -\frac{g}{l} \sin(x_1(t)) - \frac{b}{m} x_2(t) \end{bmatrix}, \quad (4.13)$$

where  $x_1(t)$  and  $x_2(t)$  are the states representing position and velocity of the pendulum respectively. The parameters are  $g = 9.8m/sec^2$ ,  $m = 2kg$ ,  $b = 0.9kg/sec$  and  $l = 1m$ . The process noise is added to each state after discretization. The discretised measurement vector at time-step  $t$  is  $y^{(t)} = x_1^{(t)} + \nu^{(t)}$  where  $x_1^{(t)}$  is the discretised state representing position of the pendulum and  $\nu^{(t)}$  is the measurement noise at time-step  $t$ . We considered sequences of length 4000 starting at time 0 seconds. Gaussian initial conditions for each sequence were chosen randomly from  $[-2, 2] \times [-2, 2]$  with an initial condition covariance of  $(0.01) \times I_2$  where  $I_k$  represents an identity matrix of order  $k$  for the generation of training, validation and testing dataset 1. All noises were assumed to be Gaussian with zero mean and known covariance. Measurement and process noise covariance were considered to be  $(0.01) \times I$  where matrix  $I$  is of appropriate dimensions. A total of 100 sequences were considered with a batch size of 20. The number of hidden units considered were 50. Early stopping was implemented with a patience of 50. The maximum number of epochs considered was 3000. Training using Adam optimization with a learning rate of  $10^{-4}$  for

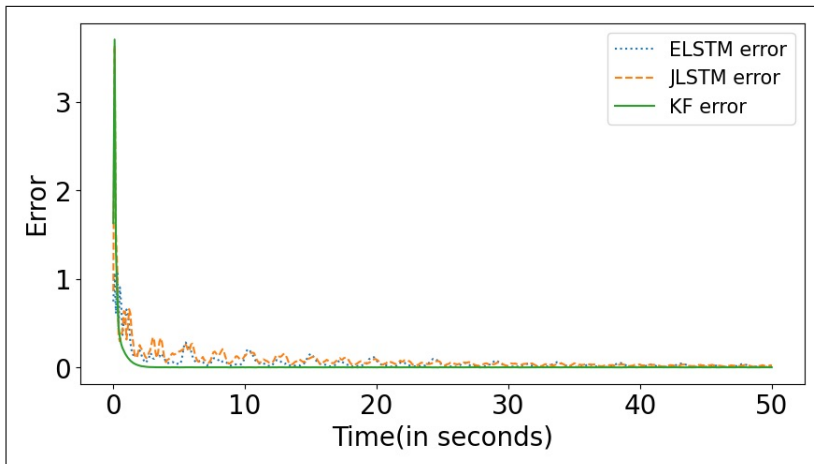


Figure 4.5: Average errors over 10 test sequences for 50 seconds of 10 connected springs with a noisy Gaussian initial condition outside the training region for **KF**, **JLSTM** and **ELSTM** based state estimates.

**ELSTM** and  $10^{-3}$  for **JLSTM** achieved a close NMSE value for both networks. While the **ELSTM** and **JLSTM** have almost equal errors at all time-steps(see Fig. 4.6), the **EKF** has the largest error at the initial time-step and converges later than the other two. For testing dataset 2, in this example, we used the interval  $[2, 2.5]^2 \sim \mathbb{R}^2$ . As can be seen from Fig. 4.7, the **ELSTM** and **JLSTM** perform better than the **EKF** especially at the initial time-steps.

We also considered an RK-45 discretization with  $\Delta t = 0.1s$  of the reversed Van der Pol oscillator:

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -x_2(t) \\ x_1(t) + ((x_1(t))^2 - 1)x_2(t) \end{bmatrix} \quad (4.14)$$

where  $x_1(t)$  and  $x_2(t)$  is the position and velocity of the oscillator. The discretized measurement vector at time-step  $t$  is  $y^{(t)} = x_1^{(t)} + \nu^{(t)}$  where  $\nu^{(t)}$  is the measurement noise at time-step  $t$ . For this example, we have considered sequences of length 300 starting from time 0. The Gaussian initial condition was sampled from  $[-1, 1] \times [-1, 1]$  with covariance  $(0.01) \times I_2$  where  $I_k$  represents an identity matrix of order  $k$ , for the generation of training, validation and testing dataset 1. All noises were assumed to be Gaussian with zero mean and known covariance. Measurement and process noise covariance were again  $(0.01) \times I$  where matrix  $I$  is of appropriate dimensions. A total of 100 sequences were considered with a batch size of 20. The number of hidden units considered were 50. Early stopping was implemented with a patience of 15. The maximum number of epochs was set to be

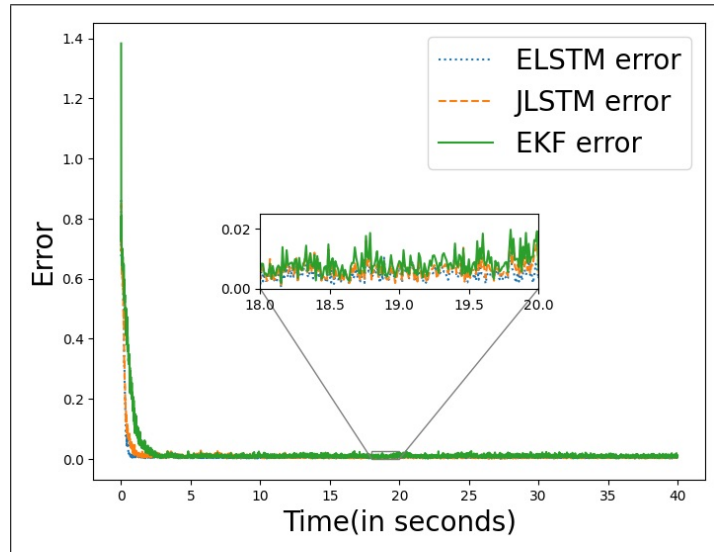


Figure 4.6: Average errors over 10 test sequences for 40 seconds of down pendulum with a noisy Gaussian initial condition for [EKF](#), [JLSTM](#) and [ELSTM](#) based state estimates.

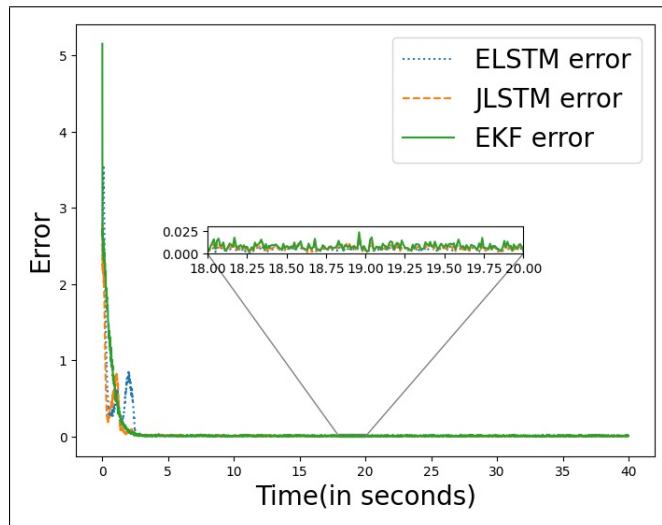


Figure 4.7: Average errors over 10 test sequences for 40 seconds of down pendulum with a noisy Gaussian initial condition outside the training region for [EKF](#), [JLSTM](#) and [ELSTM](#) based state estimates.

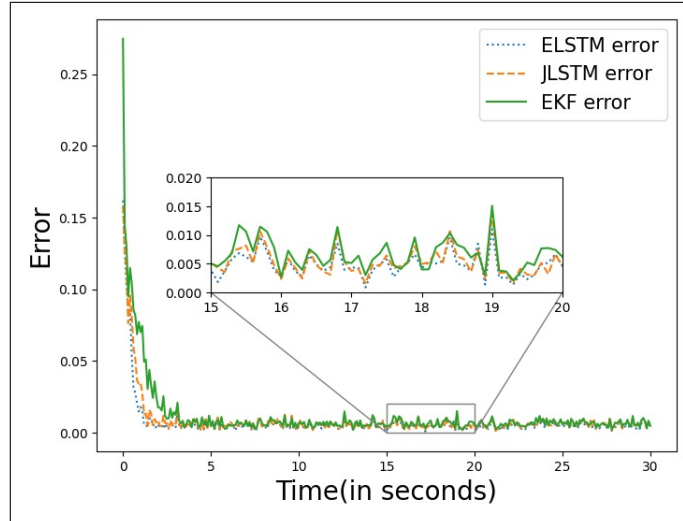


Figure 4.8: This figure compares the performance of [EKF](#), [JLSTM](#) and [ELSTM](#) using the average errors over 10 test sequences for 30 seconds of reversed Van der Pol oscillator with a noisy Gaussian initial condition.

3000. Training using Adam optimization with a learning rate of  $10^{-3}$  for [ELSTM](#) and  $10^{-2}$  for [JLSTM](#) achieves a close normalised [MSE](#) for both networks. While the [ELSTM](#) and [JLSTM](#) have almost equal errors at all time-steps(see Fig. 4.8), the [EKF](#) has the largest error at the initial time-step and converges later than the other two estimators. This is similar to the case of the down pendulum. For testing dataset 2, in this example, we used the interval  $[1, 1.5]^2 \sim \mathbb{R}^2$ . As can be seen from Fig. 4.9, the [ELSTM](#) and [JLSTM](#) perform better than the [EKF](#) especially at the initial time-steps.

## 4.2 Results, conclusions and limitations

We have compared [JLSTM](#), [ELSTM](#) and KF for a 100 dimensional linear system and the [JLSTM](#), [ELSTM](#) and [EKF](#) for two nonlinear systems of order 2, the down pendulum and the reversed Van der Pol oscillator. The results in Table 4.1 indicate that the normalised [MSE](#) value corresponding to [JLSTM](#) and [ELSTM](#) is close to the [KF](#)'s normalised [MSE](#) value for the linear system. For both nonlinear systems, both [JLSTM](#) and [ELSTM](#) have a smaller normalised [MSE](#) than the [EKF](#). This is likely because there is less linearization error. Thus, the presence of a nonlinear activation function appears advantageous. Because the learning rate was chosen so that the error values for [ELSTM](#) and [JLSTM](#) became comparable for

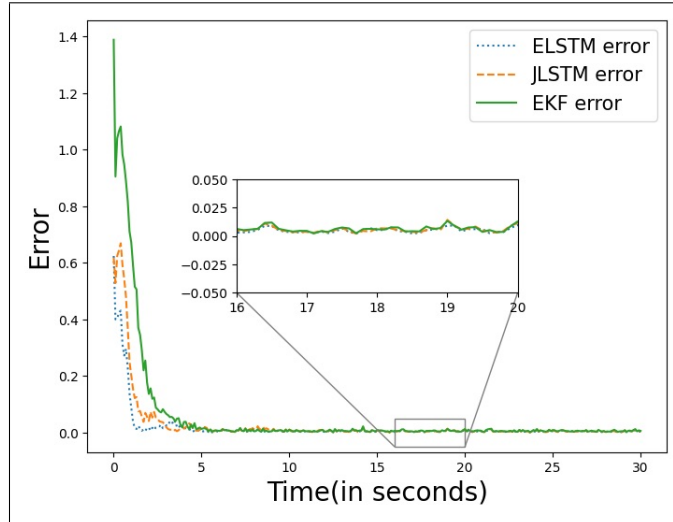


Figure 4.9: This figure compares the performance of [EKF](#), [JLSTM](#) and [ELSTM](#) using the average errors over 10 test sequences for the first 30 seconds of reversed Van der Pol oscillator with a noisy Gaussian initial condition outside the training region.

Table 4.1: Normalised [MSE](#) for [EKF](#), [JLSTM](#) and [ELSTM](#) for testing dataset 1

State space system	Estimator		
	<i>EKF</i>	<i>JLSTM</i>	<i>ELSTM</i>
Connected springs	0.0174	0.0162	0.0153
Down pendulum	0.0695	0.0542	0.0461
r. Van der Pol	0.0635	0.0485	0.0441

Table 4.2: Training time(in seconds) for [JLSTM](#) and [ELSTM](#)

State space system	Estimator	
	<i>JLSTM</i>	<i>ELSTM</i>
Connected springs	15983	16298
Down pendulum	95644	99473
r. Van der Pol	619	3389

Table 4.3: Testing time(in seconds) for **EKF**, **JLSTM** and **ELSTM** combined for both testing datasets

State space system	Estimator		
	<i>EKF</i>	<i>JLSTM</i>	<i>ELSTM</i>
Connected springs	0.8	0.3	0.3
Down pendulum	38.2	2.7	2.7
r. Van der Pol	3.2	0.2	0.2

Table 4.4: Normalised **MSE** for **EKF**, **JLSTM** and **ELSTM** for testing dataset 2

State space system	Estimator		
	<i>EKF</i>	<i>JLSTM</i>	<i>ELSTM</i>
Connected springs	0.0179	0.0567	0.0459
Down pendulum	0.0850	0.0687	0.0700
r. Van der Pol	0.0927	0.0586	0.0497

all 3 examples, a comparison of their training times is relevant. The time taken to train an **ELSTM** was considerably larger than the time taken to train a **JLSTM** (Table 4.2). This suggests that having direct connections with state estimates at the previous time-step helps to minimize the loss function faster. While the time taken to train both the **JLSTM** and **ELSTM** is large as can be seen in Table 4.2, the testing time only involves simple operations and is quite fast. We show their comparison for time taken to test 10 sequences for each example in Table 4.3. Both the long short-term memory based architectures are faster than the **KF** and the **EKF** in the respective cases. The execution time for both the **ELSTM** and **JLSTM** is almost the same. When tested on data generated using initial conditions outside the training range, we observe from Table 4.4 that the normalised **MSE** for **JLSTM** and **ELSTM** is much better than that for **EKF** for non-linear examples but more than the normalised **MSE** for **KF** for the linear example. It does appear from this that the networks are learning the model better for the nonlinear examples.

We thus conclude that both **JLSTM** and **ELSTM** provide errors smaller than **EKF** for nonlinear systems. Using a **JLSTM** instead of an **ELSTM** appears preferable because the **JLSTM** network has a considerably shorter training time to achieve the same error.

Although this method achieves better performance, a stability and convergence analysis

is pending.

# Chapter 5

## Extensions of Jordan long short-term memory network based estimators

In chapter 4, we introduced the structure of a **JLSTM** as in Figure 4.3 for state estimation, and its forward propagation equations were mentioned in (4.8). We compared it with an **ELSTM** and established that it trains faster than an **ELSTM** for the systems considered. We used the following loss function for training:

$$J^{(t)}(\phi) = \frac{1}{n} \sum_{i=1}^n \|x_i^{(t)} - \hat{x}_{i,\phi}^{(t)}\|^2, \quad (5.1)$$

where  $1 \leq i \leq n$  and  $n$  is the number of states,  $x_i^{(t)}$  represents the true value of state  $x_i$  at time-step  $t$  and  $\hat{x}_{i,\phi}^{(t)}$  represents the estimated value of state  $x_i$  at time-step  $t$  and  $\phi$  denotes hyperparameters of the network. **JLSTM** implemented here is the same as above but the number of hidden neurons was changed to accommodate the growing dimensions of nonlinear examples. The deep **JLSTM** considered consists of 2 **JLSTM** layers.

**JLSTM-ML** varies in the loss function used for training. For optimal state estimation, a widely used cost function is the maximum likelihood cost function, which aims to maximize the joint probability of states and measurements. This cost function is equivalent to

---

**Algorithm 1** Deep JLSTM Forward Pass

---

**Require:** Measurement sequence  $\{y^{(t)}\}_{t=1}^T$ , input sequence  $\{u^{(t)}\}_{t=1}^T$  (if any), number of layers  $L$ , model parameters  $\{W^{[\ell]}, b^{[\ell]}\}$  for each layer  $\ell = 1, \dots, L$

**Ensure:** Output sequence  $\{\hat{x}^{(t)}\}_{t=1}^T$  from the final layer

- 1: Initialize  $\hat{x}^{(0)[\ell]} = \mathbf{0}$ ,  $c^{(0)[\ell]} = \mathbf{0}$  for all layers  $\ell = 1, \dots, L$
- 2: **for**  $t = 1$  to  $T$  **do**
- 3:      $z^{(t)[0]} \leftarrow [y^{(t)}, u^{(t)}]$
- 4:     **for**  $\ell = 1$  to  $L$  **do**
- 5:         Compute gates at layer  $\ell$ :

$$\begin{aligned} f^{(t)[\ell]} &\leftarrow \sigma(W_{fy}^{[\ell]} z^{(t)[\ell-1]} + W_{fx}^{[\ell]} \hat{x}^{(t-1)[\ell]} + b_f^{[\ell]}) \\ i^{(t)[\ell]} &\leftarrow \sigma(W_{iy}^{[\ell]} z^{(t)[\ell-1]} + W_{ix}^{[\ell]} \hat{x}^{(t-1)[\ell]} + b_i^{[\ell]}) \\ o^{(t)[\ell]} &\leftarrow \sigma(W_{oy}^{[\ell]} z^{(t)[\ell-1]} + W_{ox}^{[\ell]} \hat{x}^{(t-1)[\ell]} + b_o^{[\ell]}) \\ \tilde{c}^{(t)[\ell]} &\leftarrow \tanh(W_{cy}^{[\ell]} z^{(t)[\ell-1]} + W_{cx}^{[\ell]} \hat{x}^{(t-1)[\ell]} + b_{\tilde{c}}^{[\ell]}) \end{aligned}$$

- 6:         Update cell and hidden activation:

$$\begin{aligned} c^{(t)[\ell]} &\leftarrow f^{(t)[\ell]} \odot c^{(t-1)[\ell]} + i^{(t)[\ell]} \odot \tilde{c}^{(t)[\ell]} \\ a^{(t)[\ell]} &\leftarrow o^{(t)[\ell]} \odot \tanh(c^{(t)[\ell]}) \end{aligned}$$

- 7:         Compute output:

$$\hat{x}^{(t)[\ell]} \leftarrow W_{xa}^{[\ell]} a^{(t)[\ell]} + b_x^{[\ell]}$$

- 8:     **end for**

- 9: **end for**

- 10: **return**  $\{\hat{x}^{(t)[L]}\}_{t=1}^T$  (final layer outputs)
-

---

**Algorithm 2** Training and Evaluation of JLSTMs

---

**Require:** Training dataset  $\mathcal{D}_{\text{train}} = \{(y_i, u_i, x_i)\}$ , validation inputs  $y_{\text{val}}, u_{\text{val}}$ , validation output  $x_{\text{val}}$ , test inputs  $y_{\text{test}}, u_{\text{test}}, y_{\text{test}}^{\text{diff}}, u_{\text{test}}^{\text{diff}}$ , and outputs  $x_{\text{test}}, x_{\text{test}}^{\text{diff}}$ , learning rate `patience_lr`, early stopping patience `patience` and minimum learning rate

$\eta_{\text{min}}$

**Ensure:** Predicted outputs on test sets:  $\hat{x}_{\text{test}}, \hat{x}_{\text{test}}^{\text{diff}}$

- 1: Initialize model with input size, hidden size and output size
- 2: Set learning rate  $\eta \leftarrow \eta_0$ , patience counters `counter`  $\leftarrow 0$ , `counter_lr`  $\leftarrow 0$
- 3: Initialize optimizer and checkpoint manager
- 4: **for** epoch = 1 to  $N_{\text{epochs}}$  **do**
- 5:     **for all** mini-batches  $(y, u, x)$  in  $\mathcal{D}_{\text{train}}$  **do**
- 6:         Compute prediction  $\hat{x} \leftarrow \text{model}(y, u)$
- 7:         Compute loss  $\mathcal{L}_{\text{train}} \leftarrow J(x, \hat{x})$
- 8:         Backpropagate and update model parameters using Adam
- 9:     **end for**
- 10:     Compute validation loss:  $\mathcal{L}_{\text{val}} \leftarrow J(\text{model}(y_{\text{val}}, u_{\text{val}}), x_{\text{val}})$
- 11:     **if**  $\mathcal{L}_{\text{val}}$  improves **then**
- 12:         Save model checkpoint
- 13:         Reset `counter`  $\leftarrow 0$
- 14:     **else**
- 15:         Increment `counter` and `counter_lr`
- 16:     **end if**
- 17:     **if** `counter`  $\geq$  `patience` **then**
- 18:         **break** (early stopping)
- 19:     **end if**
- 20:     **if** `counter_lr` == `patience_lr` **then**
- 21:         Reduce learning rate:  $\eta \leftarrow \max(\eta \cdot 0.5, \eta_{\text{min}})$
- 22:         Reset `counter_lr`  $\leftarrow 0$
- 23:     **end if**
- 24: **end for**
- 25: Load best model from checkpoint
- 26: Predict on test set:  $\hat{x}_{\text{test}} \leftarrow \text{model}(y_{\text{test}}, u_{\text{test}})$
- 27: Compute test loss  $\mathcal{L}_{\text{test}} \leftarrow \text{MSE}(\hat{x}_{\text{test}}, x_{\text{test}})$
- 28: Predict on larger IC test set:  $\hat{x}_{\text{test}}^{\text{diff}} \leftarrow \text{model}(y_{\text{test}}^{\text{diff}}, u_{\text{test}}^{\text{diff}})$
- 29: Compute shifted test loss  $\mathcal{L}_{\text{test}}^{\text{diff}} \leftarrow \text{MSE}(\hat{x}_{\text{test}}^{\text{diff}}, x_{\text{test}}^{\text{diff}})$
- 30: **return**  $\hat{x}_{\text{test}}, \hat{x}_{\text{test}}^{\text{diff}}$

---

minimizing

$$\begin{aligned}
\tilde{J}_{\text{MLE}}(x^{(0:T)}) &= (x^{(0)} - \hat{x}^{(0)})^\top P_0^{-1} (x^{(0)} - \hat{x}^{(0)}) \\
&\quad + \sum_{k=1}^T (x^{(k)} - f(x^{(k-1)}, u^{(k)}))^\top Q^{-1} (x^{(k)} - f(x^{(k-1)}, u^{(k)})) \\
&\quad + \sum_{k=1}^T (y^{(k)} - h(x^{(k)}))^\top R^{-1} (y^{(k)} - h(x^{(k)})).
\end{aligned} \tag{5.2}$$

For a linear system, this was proved in Chapter 2 to be equivalent to the maximum likelihood cost function. To make it suitable for use in an estimator, (5.2) takes the form:

$$\begin{aligned}
J_{\text{MLE}}(x^{(0:T)}) &= (x^{(0)} - \hat{x}^{(0)})^\top P_0^{-1} (x^{(0)} - \hat{x}^{(0)}) \\
&\quad + \sum_{k=1}^T (x^{(k)} - f(\hat{x}^{(k-1)}, u^{(k)}))^\top Q^{-1} (x^{(k)} - f(\hat{x}^{(k-1)}, u^{(k)})) \\
&\quad + \sum_{k=1}^T (y^{(k)} - h(\hat{x}^{(k)}))^\top R^{-1} (y^{(k)} - h(\hat{x}^{(k)})).
\end{aligned} \tag{5.3}$$

An obvious way to use it in a neural network could be to replace  $\hat{x}$  with  $\hat{x}_{NN}$ . But when the above equation was used as a loss function with a recurrent neural network for training, the network took a very long time to train due to the dependence on gradients of the function  $f$  during backpropagation. Thus, we replaced the function  $f(\hat{x}^{(k-1)}, u^{(k)})$  with  $\hat{x}^{(k)}$ , the output of the neural network at timestep  $k$  as it depends on both  $\hat{x}^{(k-1)}$  and  $u^{(k)}$ . The cost function then becomes:

$$\begin{aligned}
J_{\text{MLE}}(x^{(0:T)}) &= (x^{(0)} - \hat{x}^{(0)})^\top P_0^{-1} (x^{(0)} - \hat{x}^{(0)}) + \sum_{k=1}^T (x^{(k)} - \hat{x}^{(k)})^\top Q^{-1} (x^{(k)} - \hat{x}^{(k)}) \\
&\quad + \sum_{k=1}^T (y^{(k)} - h(\hat{x}^{(k)}))^\top R^{-1} (y^{(k)} - h(\hat{x}^{(k)})).
\end{aligned} \tag{5.4}$$

The covariance matrices that we will consider will have the form  $c * I$  where  $c$  is a scalar and  $I$  is an identity matrix of appropriate order. Thus, the inverse of the covariance matrices will have the form  $\frac{1}{c}I$ . Since  $Iz = z$  for any vector  $z$  of appropriate dimensions,

thus these matrices when multiplied with a vector lead to the multiplication of the vector by the scalar  $\frac{1}{c}$ . Also,  $z.z^\top = \|z\|^2$  for any vector  $z$ . This leads to the following simplified loss function:

$$J_{MLE}^{(t)}(\phi) = \frac{1}{c_{P(0)}} \sum_{i=1}^n \|x_i^{(0)} - \hat{x}_{i,\phi}^{(0)}\|^2 + \sum_{k=1}^T \left( \frac{1}{c_Q} \sum_{i=1}^n \|x_i^{(k)} - \hat{x}_{i,\phi}^{(k)}\|^2 + \frac{1}{c_R} \sum_{i=1}^n \|y_i^{(k)} - h(\hat{x}_{i,\phi}^{(k)})\|^2 \right) \quad (5.5)$$

where  $n$  is the batch size,  $c_{P(0)}$ ,  $c_Q$  and  $c_R$  are the constants that form initial condition covariance, process noise covariance and measurement noise covariance, respectively. But this led to exploding gradients for very large values of  $\frac{1}{c}$ . Thus, we divided these values by a number  $m$  calculated as the maximum of  $\frac{1}{c_{P_0}}$ ,  $\frac{1}{c_Q}$  and  $\frac{1}{c_R}$ . This helped overcome the issue of exploding gradients.

In this chapter, we compare these approaches using 2 experiments. First, we compared shallow and deep **JLSTMs** for various systems. We then trained a **JLSTM** using (5.5) as training and validation loss (**JLSTM-ML**). We have tested it on several systems and compared the results with **JLSTM**, deep **JLSTM**, **EKF** and **PF**. We conclude that **JLSTM**-based approaches, though take longer to train, provide a much better mean square error and are faster even including training time for long sequences of high-order systems considered. We also note that **JLSTM-ML** generalizes best for the considered non-chaotic systems.

As part of the final comparison, we analyze the robustness to random initial conditions and changes in noise levels of **JLSTM**, **JLSTM-ML**, and **EKF**. We observe that **JLSTM** and **JLSTM-ML** are more robust than **EKF**. We used 2 high-order examples (order 11 and 41) for our comparisons along with a low order example (order 2). The order of some examples is considered to be much higher than those in literature which are usually up to order 8.

## 5.1 Implementation and numerical examples

**For experiment 1:** We generate 2 types of datasets for consisting of 50 sequences and  $T$  time-steps for each example:

- The first dataset is generated by sampling the mean of initial condition from the hypercube  $\hat{x}^{(0)} \sim \mathcal{U}(x_0 - k_1, x_0 + k_1) \subset \mathbb{R}^n$  where  $x_0$  is the Galerkin approximated initial condition for the partial differential equations and is zero vector for the down pendulum. The initial condition  $x^{(0)}$  is then sampled from  $\mathcal{N}(\hat{x}_0, P_0)$ . The dataset is divided in the ratio of 80 : 10 : 10 for training, validation, and testing, respectively.

- The second dataset is used purely for testing to see if the network is able to generalize to unseen examples. In this case, the mean of the initial condition is sampled as  $\hat{x}^{(0)} \sim \mathcal{U}(x_0 - k_2, x_0 + k_2) \subset \mathbb{R}^n$  where  $k_2 > k_1$ .

We compare the performance of 5 estimators: [EKF](#), [PF](#), [JLSTM](#), deep [JLSTM](#) and [JLSTM-ML](#) on down pendulum (order 2),  $N$ -node Galerkin approximation of Burgers' partial differential equation (order 11,  $N = 5$ ) and modal approximation of Kuramoto Sivashinsky partial differential equation (order 41,  $N = 20$ ). All examples were discretized in time using forward Euler to obtain discrete-time systems. The simulation parameter values for this experiment are summarized for each example in [Table 5.1](#). The symbol  $n$  represents order of the state variable,  $m$  represents number of available measurements,  $T$  is the number of time-steps,  $c_{P^{(0)}}$ ,  $c_Q$  and  $c_R$  are such that

$$P^{(0)} = c_{P^{(0)}}I_n, \quad Q = c_Q I_n \quad \text{and} \quad R = c_R I_m \quad (5.6)$$

where  $I_r$  is an identity matrix of dimensions  $r \times r$ . To compare estimator performance, we use 2 standard measures, [MSE](#) and estimation error at each time-step averaged over all test sequences and states.

**For experiment 2:** We generate 2000 sequences of 100 time-steps each for  $N$ -node Galerkin approximation of Burgers' partial differential equation (order 11) and Kuramoto Sivashinsky partial differential equation (order 41). Each sequence starts with a random initial condition. The process and measurement noise are Gaussian and sampled from  $\mathcal{N}(\mathbf{0}, Q)$  and  $\mathcal{N}(\mathbf{0}, R)$ , respectively where  $Q$  and  $R$  are as in [\(5.6\)](#). The value of  $c_Q$  is fixed to be 0.01 and  $c_R \in [10^{-2}, 10^{-1}, 1, 10, 100]$  for comparison. We thus compare the robustness of 3 estimators: [EKF](#), [JLSTM](#) and [JLSTM-ML](#) to random initialization and changing noise levels. To quantify estimator performance in logarithmic scale, we compute the mean squared error ([MSE](#)) in decibels (dB) as follows:

$$\text{MSE}_{\text{dB}} = 10 \log_{10} \left( \frac{1}{N} \sum_{i=1}^N \|\hat{x}^{(i)} - x^{(i)}\|^2 \right), \quad (5.7)$$

where  $\hat{x}^{(i)}$  denotes the estimated state at sample  $i$ ,  $x^{(i)}$  is the corresponding ground truth state, and  $N$  is the total number of samples over which the average is computed. This transformation emphasizes relative improvements and compresses the scale of the MSE, making it easier to visualize small differences in estimator accuracy, especially in low-error regimes.

Forward propagation for [JLSTM](#) is used as in [Equation 4.8](#). For deep [JLSTM](#), 2 layers with hidden size 16 are stacked. Early stopping with fixed patience depending on each

network was implemented. Adam optimization is used with an adaptive learning rate to achieve the best performance. All network weights were initialized with *GlorotUniform*, except  $W_{xa}$ , which used an *Orthogonal* initializer to ensure stable feedback from the output to the hidden state. All biases were initialized as zero vectors. Z-score normalization for the data was used to avoid gradients from exploding. Also, the model was saved each time the validation loss improved from the previous epoch and restored to the best model for testing.

We simulate a planar, damped pendulum with torque input under gravity. The nonlinear dynamics are governed by:

$$\begin{aligned} \dot{x}_1 &= x_2, \\ \dot{x}_2 &= -\frac{g}{l} \sin(x_1) - \frac{b}{m} x_2 + \frac{1}{m} u(t), \end{aligned} \tag{5.8}$$

where  $x_1 \in \mathbb{R}$  is the pendulum angle (radians),  $x_2 \in \mathbb{R}$  is the angular velocity (rad/s) and  $u(t) \in \mathbb{R}$  is the control torque. The constants  $g = 9.8 \text{ m/s}^2$  is for acceleration due to gravity,  $l = 1 \text{ m}$  is the length of the pendulum,  $b = 0.9$  is the damping coefficient,  $m = 2.0 \text{ kg}$  is the mass of the pendulum. The system state is  $x = [x_1, x_2]^\top \in \mathbb{R}^2$ , with scalar control input  $u \in \mathbb{R}$ . The control torque  $u(t)$  is a step input defined as:

$$u(t) = \begin{cases} 0 & \text{if } t < 1.0 \text{ s,} \\ 1 & \text{if } t \geq 1.0 \text{ s.} \end{cases}$$

This input is shared across all trajectories. We discretize the dynamics using a forward Euler integrator:

$$x^{(t+1)} = x^{(t)} + \Delta t \cdot f(x^{(t)}, u^{(t)}) + w^{(t)},$$

with  $\Delta t = 10^{-3} \text{ s}$ . Only the angle  $x_1$  is observed. The measurement matrix is  $H = [1 \ 0]$ . This data-set provides a benchmark nonlinear control-affine system with partial noisy measurements and time-varying input.

We simulate the 1D viscous Burgers' equation with periodic boundary conditions:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in [-\pi, \pi], \quad t \geq 0, \tag{5.9}$$

using a spectral Galerkin method. The viscosity coefficient is set to  $\nu = 0.5$ .

The solution is approximated in a truncated Fourier basis:

$$u(x, t) \approx a_0(t) + \sum_{k=1}^N \left[ a_k(t) \cos\left(\frac{k\pi x}{L}\right) + b_k(t) \sin\left(\frac{k\pi x}{L}\right) \right], \quad L = \pi.$$

Using  $N = 5$  modes, the reduced-order state vector becomes:

$$x(t) = [a_0, a_1, \dots, a_5, b_1, \dots, b_5]^\top \in \mathbb{R}^{11}.$$

The dynamics include a linear diffusion term  $-\nu k^2$  for each mode and nonlinear convolution terms from the advection  $u\partial_x u$ . The time evolution is modeled as:

$$\frac{dx}{dt} = f(x) + w^{(t)}, \quad w^{(t)} \sim \mathcal{N}(0, Q), \quad Q = c_Q I,$$

and integrated using a forward Euler scheme with time step  $\Delta t = 10^{-6}$ . Only the cosine-mode coefficients  $a_k$  (excluding  $a_0$ ) are observed. The measurement matrix  $H \in \mathbb{R}^{5 \times 11}$  is defined as:

$$H[i, j] = \begin{cases} 1 & \text{if } j = 2i + 1, \\ 0 & \text{otherwise,} \end{cases} \quad \text{for } i = 0, 1, \dots, 4.$$

The initial condition mean  $\hat{x}^{(0)}$  for experiment 1 is chosen to excite multiple frequencies:

$$x_0[1 : n] = 0.5 \cdot \sin\left(\frac{\pi}{n}[0, 1, \dots, n - 1]\right), \quad x_0[0] = 0.5.$$

This dataset captures the noisy evolution of a moderately high-dimensional nonlinear system with partial observations, used to evaluate the performance of state estimators.

We construct synthetic trajectories from the one-dimensional Kuramoto–Sivashinsky equation:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \nu \frac{\partial^4 u}{\partial x^4} = 0, \quad x \in [-2\pi, 2\pi], \quad t \geq 0, \quad (5.10)$$

with periodic boundary conditions and hyper-viscosity coefficient  $\nu = 0.01$ . We project the solution onto a truncated Fourier basis:

$$u(x, t) \approx a_0(t) + \sum_{k=1}^N \left[ a_k(t) \cos\left(\frac{k\pi x}{L}\right) + b_k(t) \sin\left(\frac{k\pi x}{L}\right) \right], \quad L = 2\pi.$$

Using  $N = 20$  modes yields a state vector of dimension  $(2N + 1) = 41$ , i.e.,

$$x(t) = [a_0(t), a_1(t), \dots, a_{20}(t), b_1(t), \dots, b_{20}(t)]^\top.$$

The Galerkin system satisfies:

$$\frac{dx}{dt} = f(x) + w^{(t)}, \quad w^{(t)} \sim \mathcal{N}(0, Q).$$

Table 5.1: Simulation parameters for each numerical example(Experiment 1)

State space system	Parameters							
	$k_1$	$k_2$	$n$	$m$	$T$	$c_{P(0)}$	$c_Q$	$c_R$
Down pendulum	2	10	2	1	5000	0.01	0.01	0.01
Burgers' equation	0.2	0.5	11	5	100	10	0.1	0.1
Kuramoto-Sivashinsky	0.5	0.8	41	20	3000	0.0001	0.0001	0.0001

The right-hand side  $f(x)$  consists of a linear part with eigenvalues  $\lambda_k = -(\nu k^4 + k^2)$ , and a nonlinear part computed from the convolution sums over Fourier coefficients, reflecting the quadratic nonlinearity  $u\partial_x u$ . This system is integrated using the forward Euler method with step size  $\Delta t = 10^{-6}$ . We use a nonlinear-sensing measurement model based on random spatial sampling. The measurement matrix  $H \in \mathbb{R}^{20 \times 41}$  is constructed by evaluating the Fourier basis at randomly chosen sensor locations  $\{z_i\}_{i=1}^{20} \subset [-2\pi, 2\pi]$ :

$$H[i, 0] = 1, \quad H[i, j] = \cos\left(\frac{(j) \cdot 2\pi z_i}{L}\right), \quad H[i, j + N] = \sin\left(\frac{(j) \cdot 2\pi z_i}{L}\right), \quad j = 1, \dots, N.$$

We define a symbolic initial condition  $u_0(z) = 5z - 0.5z^2 - 4$  and project it onto the Fourier basis to generate:

$$x_0 = \left[ \int u_0(z)\phi_0(z)dz, \int u_0(z)\phi_1(z)dz, \dots, \int u_0(z)\phi_N(z)dz, \int u_0(z)\psi_1(z)dz, \dots, \int u_0(z)\psi_N(z)dz \right]$$

for experiment 1 where:

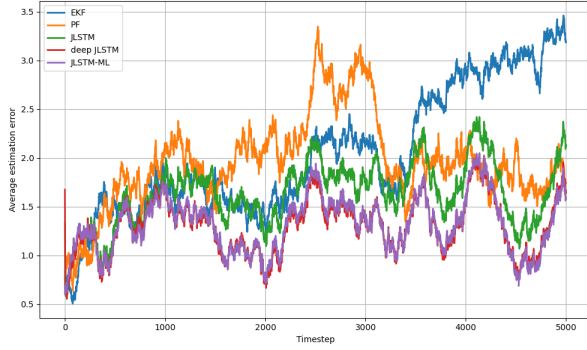
$$\phi_0(z) = \frac{1}{\sqrt{2\pi}}, \quad \phi_k(z) = \frac{1}{\sqrt{\pi}} \cos(kz), \quad \psi_k(z) = \frac{1}{\sqrt{\pi}} \sin(kz).$$

This dataset represents a high-dimensional chaotic system with structured nonlinear dynamics, rich spatial modes, and noisy partial measurements — suitable for evaluating nonlinear state estimators.

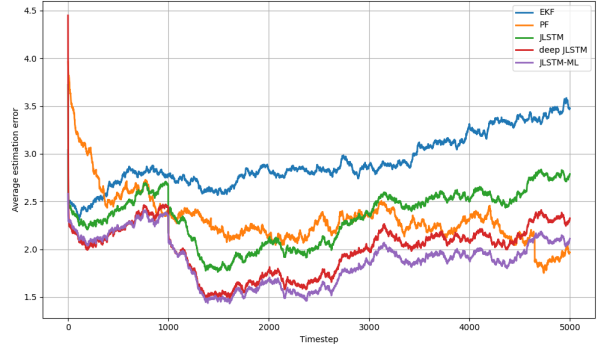
Discrete-time [EKF](#) and [PF](#) are implemented as discussed in [\[111\]](#).

## 5.2 Results and conclusions

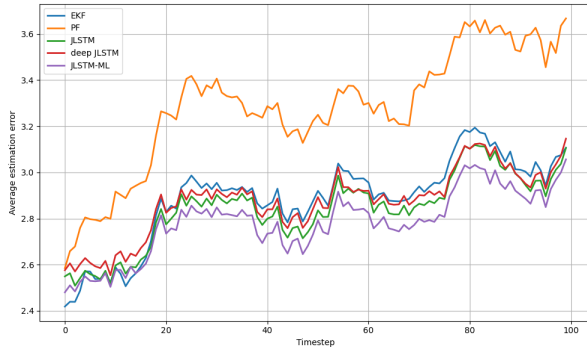
**For Experiment 1:** Table [5.2](#) and Table [5.3](#) report the mean squared error (MSE) for the three systems across both datasets. Across all settings, the best performance is consistently achieved by [JLSTM](#)-based architectures.



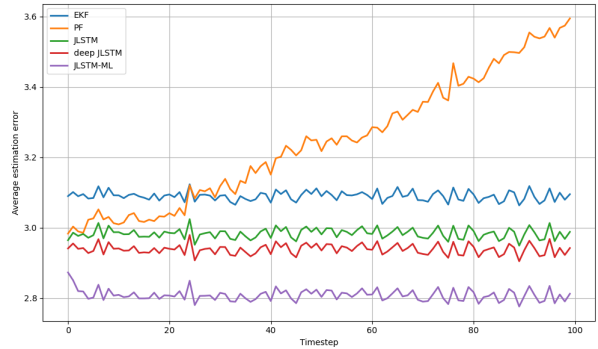
(a) For down pendulum (Dataset 1)



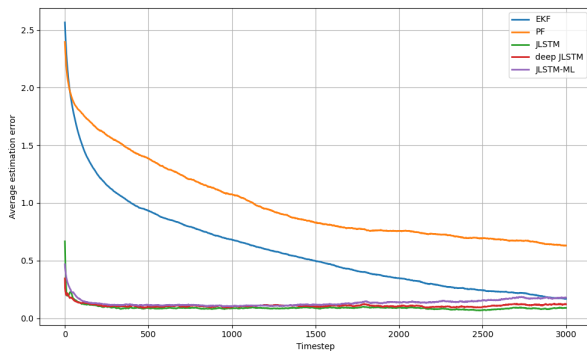
(b) For down pendulum (Dataset 2)



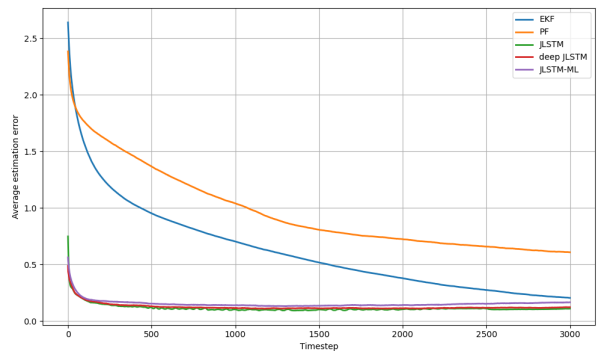
(c) For Burgers' equation (Dataset 1)



(d) For Burgers' equation (Dataset 2)

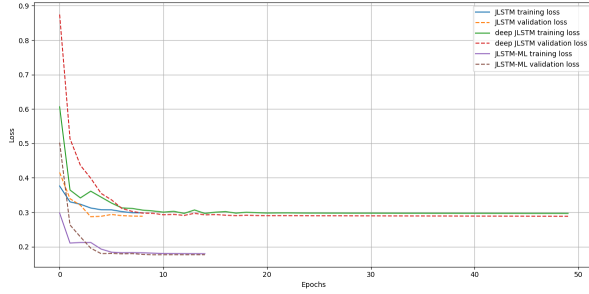


(e) For Kuramoto-Sivashinsky equation (Dataset 1)

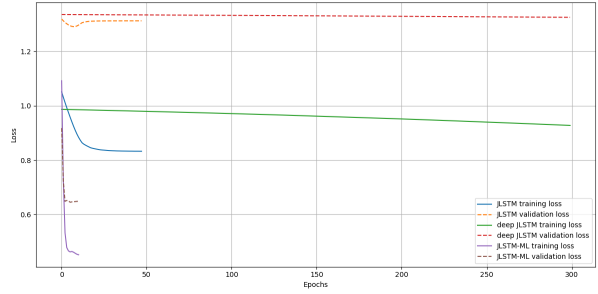


(f) For Kuramoto-Sivashinsky equation (Dataset 2)

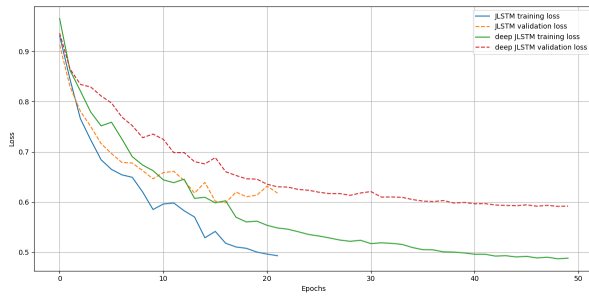
Figure 5.1: Average estimation error values per time-step for test sequences in experiment 1.



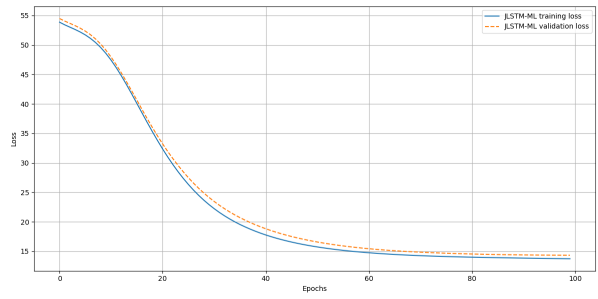
(a) For down pendulum



(b) For Burgers' equation



(c) For Kuramoto-Sivashinsky equation (JLSTM and deep JLSTM)



(d) For Kuramoto-Sivashinsky equation (for JLSTM-ML)

Figure 5.2: Training and validation loss graphs for all models in experiment 1.

Table 5.2: MSE for EKF, PF, JLSTM, deep JLSTM and JLSTM-ML (Dataset 1)

State space system	Estimator				
	<i>EKF</i>	<i>PF</i>	<i>JLSTM</i>	<i>deep JLSTM</i>	<i>JLSTM-ML</i>
Down pendulum	10.7958	11.4522	5.1431	<u>4.7202</u>	4.7948
Burgers' equation	13.0858	15.9311	12.6022	12.7035	<u>11.9985</u>
Kuramoto Sivashinsky	1.3871	3.8146	<u>0.0212</u>	0.0327	0.0541

Table 5.3: MSE for EKF, PF, JLSTM, deep JLSTM and JLSTM-ML(Dataset 2)

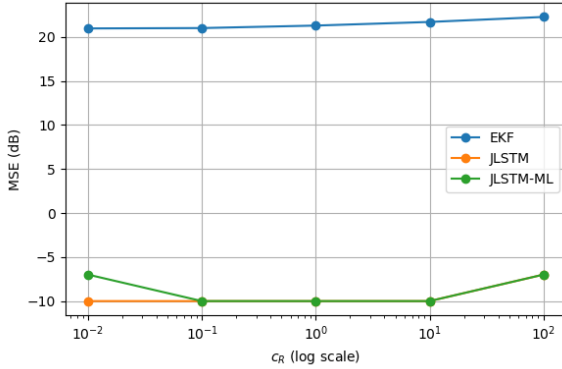
State space system	Estimator				
	<i>EKF</i>	<i>PF</i>	<i>JLSTM</i>	<i>deep JLSTM</i>	<i>JLSTM-ML</i>
Down pendulum	26.7037	22.3576	12.1941	11.3733	<u>9.6780</u>
Burgers'	13.4751	15.4967	12.4293	12.0469	<u>11.3317</u>
Kuramoto-Sivashinsky	1.4971	3.7971	<u>0.0429</u>	0.0463	0.0652

Table 5.4: Training time(in seconds) for JLSTM, deep JLSTM and JLSTM-ML

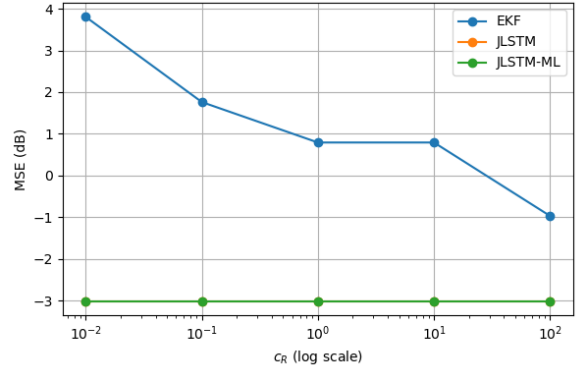
State space system	Estimator		
	<i>JLSTM</i>	<i>deep JLSTM</i>	<i>JLSTM-ML</i>
Down pendulum	<u>1251</u>	13756	1921
Burgers' equation	127	1787	<u>31</u>
Kuramoto-Sivashinsky	<u>2508</u>	8136	8564

Table 5.5: Time taken to run test data(in seconds) for EKF, PF, JLSTM, deep JLSTM and JLSTM-ML (Dataset 1 and 2)

State space system	Estimator				
	<i>EKF</i>	<i>PF</i>	<i>JLSTM</i>	<i>deep JLSTM</i>	<i>JLSTM-ML</i>
Down pendulum	13.86	1352	14.72	28.63	<u>12.87</u>
Burgers'	2.99	718	<u>0.25</u>	0.50	0.26
Kuramoto-Sivashinsky	9649	89840	10.82	15.35	<u>8.04</u>



(a) For Kuramoto-Sivashinsky equation



(b) For Burger's equation

Figure 5.3:  $c_R$ (log scale) vs. Mean square error values(in dB) using **JLSTM**, **JLSTM-ML** and **EKF** for different values of  $c_Q$

For Dataset 1, where initial conditions are sampled from the same region as training data, we observe that all **JLSTM**-based models outperform the classical **EKF** and **PF** across all systems. The difference in MSE between shallow **JLSTM** and deep **JLSTM** is small for the Burgers and Kuramoto–Sivashinsky systems, but deep **JLSTM** slightly outperforms the shallow version for the down pendulum. **JLSTM-ML** performs comparably to, or better than, all other models—achieving the lowest average estimation error in every case. **PF** struggles most significantly in high-dimensional systems due to its scaling inefficiencies with particle count.

For Dataset 2, which introduces initial conditions sampled from a larger range than training data, **EKF** and **PF** exhibit large performance drops on the down pendulum, often diverging. In contrast, **JLSTM**-based models generalize well, with **JLSTM-ML** again achieving the best results. For Burgers and Kuramoto-Sivashinsky equation, the **MSE** remains stable across datasets, indicating that the increased initial condition range (parameterized by  $k_2$ ) does not significantly affect performance in these systems. Figure 5.1 illustrates that **JLSTM**-based models achieve lower per-timestep error compared to classical filters. **PF** performs the worst, especially under Gaussian noise and over long time horizons.

Training time (Table 5.4) varies across models and systems. Deep **JLSTM** models are the slowest to train due to added complexity. **JLSTM-ML** takes longer to train than shallow **JLSTM** for 2 systems, but remains more efficient than deep models in most cases. Notably, **JLSTM-ML** trains fastest for the Burgers system.

At testing time (Table 5.5), all **JLSTM**-based models are significantly faster than **EKF** and **PF**. For the Kuramoto-Sivashinsky system, the classical filters are prohibitively slow—especially the particle filter, which takes nearly 90,000 seconds. **JLSTM-ML** is the fastest at test time for all systems except Burgers, where it is nearly identical to shallow **JLSTM**. **PF** is the slowest model across all tasks, requiring more time to test than it takes to train most networks.

Training and validation loss curves (Figures 5.2b, 5.2c, 5.2d, and 5.2a) show that **JLSTM-ML** maintains closer alignment between training and validation loss, suggesting improved generalization and reduced overfitting. Deep **JLSTM** can achieve competitive accuracy but often converges more slowly and may exhibit mild overfitting. While **JLSTM-ML** may converge more slowly for complex systems like KS, it ultimately achieves the best validation loss.

**For experiment 2:** Figure 5.3 shows the **MSE** (in dB) across various values of  $c_R$  on a log scale, with a fixed process noise level  $c_Q = 0.01$ . Results for both Burgers and Kuramoto-Sivashinsky equations indicate that **JLSTM**-based models are significantly more robust than **EKF** across all noise levels. The improvement is particularly evident in high-noise regimes, where classical filters degrade sharply. **JLSTM-ML** remains effective even under challenging noise settings, reinforcing its utility for real-world scenarios with unknown or varying noise statistics.

**Conclusion:** Across all systems and experimental conditions, **JLSTM**-based models offer superior performance in terms of estimation accuracy, robustness, and inference speed. Among all tested models, **JLSTM-ML** consistently delivers the best trade-off between stability, generalization, and efficiency. While training time is long for neural network based methods, very low implementation time helps balance the cost.

We observed that going 2 layers deep did not improve the results significantly thus pointing towards the need to explore deeper layers ( $> 2$ ). One possible reason is that adding another layer of **JLSTM** aims to enhance the output, but it significantly increases training time while offering only marginal performance gains. It is also possible that the complexity of the nonlinear systems considered requires representation with deeper layers for more improvement.

# Chapter 6

## Conclusions and future research

This thesis focused on designing neural network-based state estimators for nonlinear dynamical systems, guided by theoretical principles to ensure robust and reliable performance. We began by using [JRN](#) for state estimation and observed that, across a variety of nonlinear dynamical systems, the error dynamics associated with [JRN](#)-based estimators exhibited ISS behavior. These results, consistent across different systems, served as the motivation for extending the architectural principles of [JRN](#)s to more powerful sequential models. This led to the development of the [JLSTM](#) architecture — a recurrent neural network that incorporates output feedback in a Jordan-style configuration while leveraging the gating mechanisms of [ELSTM](#). The key design objective was to retain the dynamical system-like structure of [JRN](#)s while improving memory, reducing exploding and vanishing gradients and enhancing nonlinear representation capabilities via long short-term memory cells.

Several important outcomes emerged from the design and evaluation of Jordan structure-based estimators. By feeding outputs back into the hidden-state dynamics, [JRN](#)s align more closely with classical estimator structures, making them more interpretable and allowing for a more tractable analysis of their stability properties. Their extension, [JLSTM](#) also mitigates this issue by reducing the direct dependence of the error dynamics on the internal hidden-state evolution. As a result, it is less sensitive to internal perturbations and random initializations during training. Empirical results demonstrated that [JLSTM](#)-based estimators consistently outperformed the [ELSTM](#) model, [EKF](#), [UKF](#) and [PF](#) in scenarios with varying process noise, measurement noise and initialization uncertainties. This robustness is crucial in real-world systems where exact noise models are rarely available and initial conditions are either unknown or only partially known.

Another key contribution of this work was the integration of a maximum likelihood-

based cost function in training of a [JLSTM](#), aligning the learning objective with optimal estimation principles. Compared to traditional [MSE](#) losses, the maximum likelihood formulation provided a more stable training and validation dynamics. It also contributed to faster convergence in some cases and reduced overfitting, especially in high-order (order 11 and 41) or chaotic systems. The [JLSTM](#) model showed promising results in examples: connected springs, the down pendulum, the reversed van der Pol oscillator, Burgers’ equation, and the Kuramoto-Sivashinsky system. These represent a wide range of dynamic behavior — from low-dimensional systems with external inputs to high-dimensional nonlinear partial differential equation discretizations — further validating the flexibility and adaptability of the proposed estimator design.

The results in this work suggest that incorporating domain-related architectural constraints (e.g. Jordan-style output feedback or model-based cost functions) can significantly improve the reliability of learning-based estimators. The structure [JLSTM-ML](#) exemplifies how blending classical estimation theory with machine learning can lead to architectures that not only perform well but also exhibit properties such as bounded error dynamics and robustness.

We also observed that the run time for [JLSTM](#) based estimators is extremely low in comparison to other methods, especially for high-order systems. Thus, making them a preferable choice for real-time state estimation.

In particular, this work contributes to a growing body of research that aims to bridge the gap between black-box neural networks and theory-driven estimator design, where interpretability, stability, and generalization are essential.

## 6.1 Future directions

While the [JLSTM](#) model demonstrates many desirable properties, several open questions remain. While input to state stability behavior was shown for [JRN](#) error dynamics, a theoretical proof of stability for [JLSTM](#) error dynamics remains an open challenge. The issue one faces is the dependence of errors on the cell state, which prevents the results in Chapter 3 from being extended without further assumptions. Future work may involve Lyapunov-based analyses or establishing sufficient conditions for boundedness using tools from nonlinear systems theory. The testing of architectures on higher-dimensional (> 41) and real-world datasets is a crucial next step.

The inclusion of process noise in data generation does account to some extent for model errors. However, an important scenario to test would be when there is model mismatch

between the estimator and the true dynamical system.

Another future direction is to estimate the filter gain (see equation 2.2) with JLSTM. Some preliminary work has been done and research is ongoing.

# References

- [1] Alessandro Abate, Daniele Ahmed, Alec Edwards, Mirco Giacobbe, and Andrea Peruffo. FOSSIL: a software tool for the formal synthesis of Lyapunov functions and barrier certificates using neural networks. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2021.
- [2] Dipak M Adhyaru. State observer design for nonlinear systems using neural network. *Applied Soft Computing*, 12(8):2530–2537, 2012.
- [3] Faya Safirra Adi, Yee Jin Lee, and Hwachang Song. State estimation for dc microgrids using modified long short-term memory networks. *Applied Sciences*, 10(9):3028, 2020.
- [4] Sepideh Afshar, Fabian Germ, and Kirsten Morris. Extended Kalman filter based observer design for semilinear infinite-dimensional systems. *IEEE Transactions on Automatic Control*, 69(6):3631–3646, 2023.
- [5] Rasha Al Jamal and Kirsten Morris. Linearized stability of partial differential equations with application to stabilization of the kuramoto–sivashinsky equation. *SIAM Journal on Control and Optimization*, 56(1):120–147, 2018.
- [6] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. *Advances in neural information processing systems*, 32, 2019.
- [7] N.E. Barabanov and D.V. Prokhorov. Stability analysis of discrete-time recurrent neural networks. *IEEE Transactions on Neural Networks*, 13(2):292–303, 2002.
- [8] George Bebis and Michael Georgiopoulos. Feed-forward neural networks. *Ieee Potentials*, 13(4):27–31, 2002.

- [9] Mouhacine Benosman and Jeff Borggaard. Data-driven robust state estimation for reduced-order models of 2d boussinesq equations with parametric uncertainties. *Computers & Fluids*, 214:104773, 2021.
- [10] Mouhacine Benosman and Jeff Borggaard. Robust nonlinear state estimation for a class of infinite-dimensional systems using reduced-order models. *International Journal of Control*, 94(5):1309–1320, 2021.
- [11] Alain Bensoussan. *Estimation and control of dynamical systems*, volume 48. Springer, 2018.
- [12] Pauline Bernard, Vincent Andrieu, and Daniele Astolfi. Observer design for continuous-time dynamical systems. *Annual Reviews in Control*, 2022.
- [13] José M. Bernardo and Adrian F. M. Smith. *Bayesian theory*. John Wiley & Sons, 2000.
- [14] Dimitri Bertsekas. *Reinforcement learning and optimal control*, volume 1. Athena Scientific, 2019.
- [15] Tobias Breiten and Karl Kunisch. Neural network based nonlinear observers. *Systems & Control Letters*, 148:104829, 2021.
- [16] Juan Pedro Llerena Cana, Jesus Garcia Herrero, and Jose Manuel Molina Lopez. Forecasting nonlinear systems with LSTM: analysis and comparison with EKF. *Sensors*, 21(5):1805, 2021.
- [17] Ya-Chien Chang, Nima Roohi, and Sicun Gao. Neural Lyapunov control. *Advances in Neural Information Processing Systems*, 32, 2019.
- [18] Chi-Tsong Chen. *Linear system theory and design*. Saunders college publishing, 1984.
- [19] S Kumar Chenna, Yogesh Kr Jain, Himanshu Kapoor, Raju S Bapi, Narri Yadaiah, Atul Negi, V Seshagiri Rao, and Bulusu Lakshmana Deekshatulu. State estimation and tracking problems: a comparison between Kalman filter and recurrent neural networks. In *Neural Information Processing: 11th International Conference, ICONIP 2004, Calcutta, India, November 22-25, 2004. Proceedings 11*, pages 275–281. Springer, 2004.
- [20] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. John Wiley & Sons, 2 edition, 2006.

- [21] Dan Crisan and Arnaud Doucet. A survey of convergence results on particle filtering methods for practitioners. *IEEE Transactions on signal processing*, 50(3):736–746, 2002.
- [22] Orlando De Jesus and Martin T Hagan. Backpropagation algorithms for a broad class of dynamic networks. *IEEE Transactions on Neural Networks*, 18(1):14–27, 2007.
- [23] Sarupa Debnath, Soumya Ranjan Sahoo, Bernard Twum Agyeman, and Jinfeng Liu. Input-output selection for lstm-based reduced-order state estimator design. *Mathematics*, 11(2):400, 2023.
- [24] Ronald Deep. *Probability and statistics: with integrated software routines*. Elsevier, 2005.
- [25] Nguyen Anh Khoa Doan, Wolfgang Polifke, and Luca Magri. Short-and long-term predictions of chaotic flows and extreme events: a physics-constrained reservoir computing approach. *Proceedings of the Royal Society A*, 477(2253):20210135, 2021.
- [26] Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors. *Sequential Monte Carlo methods in practice*. Springer, 2001.
- [27] Pierre Dubois, Thomas Gomez, Laurent Planckaert, and Laurent Perret. Machine learning for fluid flow reconstruction from limited measurements. *Journal of Computational Physics*, 448:110733, 2022.
- [28] Ronald A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 222(594-604):309–368, 1922.
- [29] Zhi-Jun Fu, Wen-Fang Xie, and Jing Na. Robust adaptive nonlinear observer design via multi-time scales neural network. *Neurocomputing*, 190:217–225, 2016.
- [30] Chang Gao, Junkun Yan, Shenghua Zhou, Pramod K Varshney, and Hongwei Liu. Long short-term memory-based deep recurrent neural networks for target tracking. *Information Sciences*, 502:279–296, 2019.
- [31] Sicun Gao, Soonho Kong, and Edmund M Clarke. dReal: an SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

- [32] Tao Gao, Gang Wang, and Xuefeng Chen. DLKFN: a deep learning-based Kalman filter network for nonlinear state estimation. *Neurocomputing*, 401:96–105, 2020.
- [33] Xile Gao, Haiyong Luo, Bokun Ning, Fang Zhao, Linfeng Bao, Yilin Gong, Yimin Xiao, and Jinguang Jiang. RL-akf: An adaptive kalman filter navigation algorithm based on reinforcement learning for ground vehicles. *Remote Sensing*, 12(11):1704, 2020.
- [34] Jean-Paul Gauthier and Ivan Kupka. *Deterministic observation theory and applications*. Cambridge university press, 2001.
- [35] Roman Geiselhart, Rob H Gielen, Mircea Lazar, and Fabian R Wirth. An alternative converse Lyapunov theorem for discrete-time systems. *Systems & Control Letters*, 70:49–59, 2014.
- [36] Arthur Gelb, editor. *Applied optimal estimation*. MIT Press, 1974.
- [37] Ramazan Gencay and Tung Liu. Nonlinear modelling and prediction with feedforward and recurrent networks. *Physica D: Nonlinear Phenomena*, 108(1-2):119–134, 1997.
- [38] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.
- [39] Neil J. Gordon, David J. Salmond, and Adrian F. M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F - Radar and Signal Processing*, 140(2):107–113, 1993.
- [40] Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.
- [41] Kevin Gurney. *An introduction to neural networks*. CRC press, 2018.
- [42] Herman O Hartley and Aaron Booker. Nonlinear least squares estimation. *The Annals of mathematical statistics*, 36(2):638–650, 1965.
- [43] Omar Hijab. Asymptotic nonlinear filtering and large deviations. In *Advances in Filtering and Optimal Stochastic Control: Proceedings of the IFIP-WG 7/1 Working Conference Cocoyoc, Mexico, February 1–6, 1982*, pages 170–176. Springer, 2005.
- [44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

- [45] Paul W. Holland and Roy E. Welsch. Robust regression using iteratively reweighted least-squares. *Communications in Statistics-Theory and Methods*, 6(9):813–827, 1977.
- [46] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [47] Peter J Huber. *Robust estimation of a location parameter*. Springer, 1992.
- [48] Janne MJ Huttunen, Jari P Kaipio, and Heikki Haario. Approximation error approach in spatiotemporally chaotic models with application to kuramoto–sivashinsky equation. *Computational Statistics & Data Analysis*, 123:13–31, 2018.
- [49] Alberto Isidori. *Nonlinear control systems: an introduction*. Springer, 1985.
- [50] Herbert 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*, 148(34):13, 2001.
- [51] Andrew H. Jazwinski. *Stochastic processes and filtering theory*. Academic Press, 1970.
- [52] Zhong-Ping Jiang and Yuan Wang. Input-to-state stability for discrete-time nonlinear systems. *Automatica*, 37(6):857–869, 2001.
- [53] Kam-Chuen Jim, C Lee Giles, and Bill G Horne. An analysis of noise in recurrent neural networks: convergence and generalization. *IEEE Transactions on neural networks*, 7(6):1424–1438, 1996.
- [54] Xue-Bo Jin, Ruben Jonhson Robert Jeremiah, Ting-Li Su, Yu-Ting Bai, and Jian-Lei Kong. The new trend of state estimation: From model-driven to hybrid-driven methods. *Sensors*, 21(6):2085, 2021.
- [55] Michael I Jordan. Serial order: a parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.
- [56] Simon J. Julier and Jeffrey K. Uhlmann. A new extension of the Kalman filter to nonlinear systems. *Aerospace/Defense Sensing, Simulation, and Controls*, 3068:182–193, 1997.

- [57] Simon J. Julier, Jeffrey K. Uhlmann, and Hugh F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of the 1995 American Control Conference*, volume 3, pages 1628–1632. IEEE, 1995.
- [58] John L Junkins. *An introduction to optimal estimation of dynamical systems*. Springer, 1978.
- [59] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.
- [60] Engin Kandiran and Avadis Hacinliyan. Comparison of feedforward and recurrent neural network in forecasting chaotic dynamical system. *AJIT-e: Academic Journal of Information Technology*, 10(37):31–44, 2019.
- [61] Wei Kang. Moving horizon numerical observers of nonlinear control systems. *IEEE Transactions on Automatic Control*, 51(2):344–350, 2006.
- [62] Wei Kang and Lucas C Wilcox. Mitigating the curse of dimensionality: sparse grid characteristics method for optimal feedback control and hjb equations. *Computational Optimization and Applications*, 68(2):289–315, 2017.
- [63] Zolidah Kasiran, Zaidah Ibrahim, and Muhammad Syahir Mohd Ribuan. Mobile phone customers churn prediction using elman and jordan recurrent neural network. In *2012 7th international conference on computing and convergence technology (IC-CCT)*, pages 673–678. IEEE, 2012.
- [64] Avneet Kaur and Kirsten A. Morris. State estimator design using Jordan-based long short-term memory networks. *arXiv preprint arxiv:2502.04518*, 2025.
- [65] Avneet Kaur, Ruikun Zhou, Jun Liu, and Kirsten Morris. Stability of Jordan recurrent neural network estimator. *preprint*, 2024.
- [66] Hassan K Khalil and Jessy W Grizzle. *Nonlinear systems*, volume 3. Prentice hall Upper Saddle River, NJ, 2002.
- [67] Tawsif Khan, Kirsten Morris, and Marek Stastna. Computation of the optimal sensor location for the estimation of an 1-d linear dispersive wave equation. In *2015 American Control Conference (ACC)*, pages 5270–5275. IEEE, 2015.
- [68] James N Knight. *Stability analysis of recurrent neural networks with applications*. Colorado State University, 2008.

- [69] Arthur J Krener. The convergence of the extended kalman filter. In *Directions in mathematical systems theory and optimization*, pages 173–182. Springer, 2002.
- [70] Arthur J Krener. The convergence of the minimum energy estimator. In *New Trends in Nonlinear Dynamics and Control and their Applications*, pages 187–208. Springer, 2004.
- [71] Arthur J Krener. Minimum energy estimation applied to the lorenz attractor. In *Numerical Methods for Optimal Control Problems*, pages 165–182. Springer, 2019.
- [72] Chung-Ming Kuan, Kurt Hornik, and Halbert White. A convergence result for learning in recurrent neural networks. *Neural Computation*, 6(3):420–440, 1994.
- [73] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.
- [74] Sudeep Kundu and Karl Kunisch. Policy iteration for Hamilton-Jacobi-Bellman equations with control constraints. *Computational Optimization and Applications*, 87(3):785–809, 2024.
- [75] Karl Kunisch, Stefan Volkwein, and Claudia Wulff. A reduced-order observer for nonlinear infinite-dimensional systems with applications to PDEs. *Mathematical Control and Related Fields*, 11(1):1–25, 2021.
- [76] Harold Joseph Kushner. Stochastic stability. In *Stability of Stochastic Dynamical Systems: Proceedings of the International Symposium Organized by “The Control Theory Centre”, University of Warwick, July 10–14, 1972 Sponsored by the “International Union of Theoretical and Applied Mechanics”*, pages 97–124. Springer, 2006.
- [77] Anna Kutschireiter, Simone Carlo Surace, and Jean-Pascal Pfister. The hitchhiker’s guide to nonlinear filtering. *Journal of Mathematical Psychology*, 94:102307, 2020.
- [78] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International journal of forecasting*, 37(4):1748–1764, 2021.
- [79] Jun Liu, Yiming Meng, Maxwell Fitzsimmons, and Ruikun Zhou. Physics-informed neural network Lyapunov functions: PDE characterization, learning, and verification. *Automatica*, 175:112193, 2025.

- [80] Lennart Ljung. *System identification: theory for the user*. PTR PReNTice Hall Information ad System Sciences Series, 1999.
- [81] David G. Luenberger. *Optimization by vector space methods*. John Wiley & Sons, 1969.
- [82] David G Luenberger. Observing the state of a linear system. *IEEE transactions on military electronics*, 8(2):74–80, 2007.
- [83] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer science review*, 3(3):127–149, 2009.
- [84] Navin Kumar Manaswi. RNN and LSTM). *Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras*, pages 115–126, 2018.
- [85] Danilo P Mandic and Jonathon Chambers. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. John Wiley & Sons, Inc., 2001.
- [86] Peter S. Maybeck. *Stochastic models, estimation, and control*. Academic Press, 1979.
- [87] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [88] Philippe Moireau. A discrete-time optimal filtering approach for non-linear systems as a stable discretization of the mortensen observer. *ESAIM: Control, Optimisation and Calculus of Variations*, 24(4):1815–1847, 2018.
- [89] Kirsten Morris and Carmeliza Navasca. Approximation of low rank solutions for linear quadratic control of partial differential equations. *Computational Optimization and Applications*, 46(1):93–111, 2010.
- [90] Kirsten A Morris. *Introduction to feedback control*. Academic Press, Inc., 2000.
- [91] Kirsten A Morris. *Controller design for distributed parameter systems*. Springer, 2020.
- [92] Richard E Mortensen. Maximum-likelihood recursive nonlinear filtering. *Journal of Optimization Theory and Applications*, 2(6):386–394, 1968.

- [93] Nirmal J Nair and Andres Goza. Leveraging reduced-order models for state estimation using deep learning. *Journal of Fluid Mechanics*, 897:R1, 2020.
- [94] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive deep learning for high-dimensional Hamilton-Jacobi-Bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021.
- [95] Amirhossein Nazerian, Chad Nathe, Joseph D Hart, and Francesco Sorrentino. Synchronizing chaos using reservoir computing. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(10), 2023.
- [96] Jieun Park, Dokkyun Yi, and Sangmin Ji. Analysis of recurrent neural network and predictions. *Symmetry*, 12(4):615, 2020.
- [97] Sergio Pequito, A Pedro Aguiar, Bruno Sinopoli, and Diogo A Gomes. Nonlinear estimation using mean field games. In *International Conference on NETWORK Games, Control and Optimization (NetGCooP 2011)*, pages 1–5. IEEE, 2011.
- [98] Johan Peralez and Madiha Nadri. Deep learning-based luenberger observer design for discrete-time nonlinear systems. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 4370–4375. IEEE, 2021.
- [99] Alexander S Poznyak, Edgar N Sanchez, and Wen Yu. *Differential neural networks for robust nonlinear control: identification, state estimation and trajectory tracking*. World Scientific, 2001.
- [100] Bhukya Ramadevi and Kishore Bingi. Chaotic time series forecasting approaches using machine learning techniques: A review. *Symmetry*, 14(5):955, 2022.
- [101] Louise da C Ramos, Florent Di Meglio, Valery Morgenthaler, Luís F Figueira da Silva, and Pauline Bernard. Numerical design of luenberger observers for nonlinear systems. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 5435–5442. IEEE, 2020.
- [102] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of interior-point methods to model predictive control. *Journal of Optimization Theory and Applications*, 116(2):241–255, 2003.
- [103] Konrad Reif, Stefan Gunther, Engin Yaz, and Rolf Unbehauen. Stochastic stability of the discrete-time extended Kalman filter. *IEEE Transactions on Automatic control*, 44(4):714–728, 1999.

- [104] Jonathan Revach, Gal Dahan, Guy Zarchi, and Yonina C. Eldar. KalmanNet: neural network Kalman filtering using learned dynamics. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4):1470–1484, 2023.
- [105] Simo Särkkä and Lennart Svensson. *Bayesian filtering and smoothing*, volume 17. Cambridge university press, 2023.
- [106] Anton Maximilian Schäfer and Hans-Georg Zimmermann. Recurrent neural networks are universal approximators. *International journal of neural systems*, 17(04):253–263, 2007.
- [107] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.
- [108] Fred C. Schweppe, J. Wildes, and D. Rom. Static state estimation in electric power systems: Part i. theoretical formulation. *IEEE Transactions on Power Apparatus and Systems*, PAS-93(3):120–125, 1974.
- [109] Siyuan Shen, Jichen Chen, Guanfeng Yu, Zhengjun Zhai, and Pujie Han. Kalmanformer: using transformer to model the kalman gain in kalman filters. *Frontiers in Neurorobotics*, 18:1460255, 2025.
- [110] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung. Fixed-point performance analysis of recurrent neural networks. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 976–980. IEEE, 2016.
- [111] Dan Simon. *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.
- [112] Patrick Slade, Zachary N Sunberg, and Mykel J Kochenderfer. Estimation and control using sampling-based bayesian reinforcement learning. *IET Cyber-Physical Systems: Theory & Applications*, 5(1):127–135, 2020.
- [113] Valentin Sonntag, Jean-Marc Le Caillec, Alain Peres, and Stéphane Devaud. Transformer-based state estimation for tracking: maneuvering target and multi-target capabilities. In *2024 IEEE Radar Conference (RadarConf24)*, pages 1–6. IEEE, 2024.
- [114] Eduardo D Sontag. *Mathematical control theory: deterministic finite dimensional systems*, volume 6. Springer Science & Business Media, 2013.

- [115] Eduardo D Sontag et al. On the input-to-state stability property. *Eur. J. Control*, 1(1):24–36, 1995.
- [116] Kai Sun, Junjian Qi, and Wei Kang. Power system observability and dynamic state estimation for stability monitoring using synchrophasor measurements. *Control Engineering Practice*, 53:160–172, 2016.
- [117] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. MIT Press, 2 edition, 2018.
- [118] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.
- [119] Pantelis-Rafail Vlachas, Jaideep Pathak, Brian R Hunt, Themistoklis P Sapsis, Michelle Girvan, Edward Ott, and Petros Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, 2020.
- [120] Zhanshan Wang, Jidong Wang, and Yanming Wu. State estimation for recurrent neural networks with unknown delays: a robust analysis approach. *Neurocomputing*, 227:29–36, 2017.
- [121] Greg Welch and Gary Bishop. *An introduction to the Kalman filter*. UNC-Chapel Hill, 1995. TR 95-041.
- [122] Fan Wu, Haiyong Luo, Hongwei Jia, Fang Zhao, Yimin Xiao, and Xile Gao. Predicting the noise covariance with a multitask learning model for kalman filter-based gnss/ins integrated navigation. *IEEE Transactions on Instrumentation and Measurement*, 70:1–13, 2020.
- [123] Wei Wu, Shu-Yi An, Peng Guan, De-Sheng Huang, and Bao-Sen Zhou. Time series analysis of human brucellosis in mainland China by using Elman and Jordan recurrent neural networks. *BMC infectious diseases*, 19:1–11, 2019.
- [124] Kai Xiong, HY Zhang, and CW Chan. Performance evaluation of UKF-based nonlinear filtering. *Automatica*, 42(2):261–270, 2006.
- [125] N Yadaiah, Raju S Bapi, Lakshman Singh, and BL Deekshatulu. DEKF based recurrent neural network for state estimation of nonlinear dynamical systems. In *2011 IEEE Recent Advances in Intelligent Computational Systems*, pages 311–316. IEEE, 2011.

- [126] Zhang Yi. *Convergence analysis of recurrent neural networks*, volume 13. Springer Science & Business Media, 2013.
- [127] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, 31(7):1235–1270, 2019.
- [128] Ruikun Zhou, Thanin Quartz, Hans De Sterck, and Jun Liu. Neural Lyapunov control of unknown nonlinear systems with stability guarantees. *Advances in Neural Information Processing Systems*, 35:29113–29125, 2022.